

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Інформаційно – аналітична система моніторингу
споживання електроенергії»**

Виконав студент групи ЗПЗм-23-1
ПАВИЦЬКА Юлія Володимирівна

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
ВЛАСЕНКО Олег Васильович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«01» жовтня 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ПАВИЦЬКА Юлія Володимирівна**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Інформаційно-аналітична система моніторингу споживання електроенергії»**, затверджена Наказом закладу вищої освіти від **«01» жовтня 2024 р. № 521/с**

Вихідні дані для роботи: **Моніторинг введених і заданих даних лічильників для прогнозування нейромережами майбутнього споживання і ефективної економії.**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю.М.	01.10.2024	01.10.2024
2	Єфремов Ю.М.	01.10.2024.	01.10.2024
3	Єфремов Ю.М.	01.10.2024	01.10.2024

РЕФЕРАТ

Кваліфікаційна робота освітнього ступеня «магістр» присвячена розробці екологічно чистої системи моніторингу споживання енергії. Пояснювальна записка складається зі вступу, трьох розділів, висновків та списку використаної літератури. Пояснювальна записка до кваліфікаційної магістерської роботи містить 80 сторінок, 17 ілюстрацій, 1 додаток та 34 посилань на зовнішні джерела.

Метою роботи є створення веб-застосунку для моніторингу та оптимізації споживання електроенергії вдома без підключення до домашніх приладів, що сприяє зменшенню екологічного сліду користувачів. В роботі описано процес розробки веб-застосунку, який дозволяє користувачам вводити дані про своє щомісячне споживання електроенергії, аналізувати їх за допомогою інформативних графіків та звітів, а також отримувати персоналізовані рекомендації щодо зниження споживання. Досліджено методи збору та обробки даних про споживання електроенергії, а також принципи побудови зручного та інтуїтивно зрозумілого інтерфейсу користувача. Проведено тестування веб-застосунку згідно основних функціональних можливостей.

КЛЮЧОВІ СЛОВА: МОНІТОРИНГ ЕНЕРГОСПОЖИВАННЯ, ЕНЕРГОЕФЕКТИВНІСТЬ, ВЕБ-ЗАСТОСУНОК, АНАЛІТИКА ДАНИХ, ОПТИМІЗАЦІЯ, ЕКОЛОГІЧНА СВІДОМІСТЬ.

ABSTRACT

The qualification work for the Master's degree is devoted to the development of an environmentally friendly energy consumption monitoring system. The explanatory note consists of an introduction, three chapters, conclusions, and a list of references. The explanatory note to the Master's qualification work contains 80 pages, 17 illustrations, 1 appendix, and 34 references to external sources.

The aim of the work is to create a web application for monitoring and optimizing home energy consumption without connecting to home appliances, which helps to reduce the environmental footprint of users. The work describes the process of developing a web application that allows users to enter data on their monthly energy consumption, analyze it using informative graphs and reports, and also receive personalized recommendations for reducing consumption. Methods for collecting and processing energy consumption data, as well as the principles of building a convenient and intuitive user interface, were investigated. The web application was tested according to the main functional capabilities.

KEYWORDS: ENERGY CONSUMPTION MONITORING, ENERGY EFFICIENCY, WEB APPLICATION, DATA ANALYTICS, OPTIMIZATION, ENVIRONMENTAL AWARENESS.

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	9
1.1. Проблеми та виклики у сфері споживання електроенергії та огляд існуючих методів моніторингу й оптимізації	9
1.2. Постановка задачі дослідження та вибір засобів реалізації веб-застосунку	15
1.3. Вплив інформаційних технологій на підвищення енергоефективності.....	19
1.4. Проблеми впровадження існуючих рішень.....	22
Висновки до розділу 1	24
РОЗДІЛ 2. РОЗРОБКА ТА ПРОЕКТУВАННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ МОНИТОРИНГУ ТА ОПТИМІЗАЦІЇ СПОЖИВАННЯ ЕЛЕКТРОЕНЕРГІЇ.....	26
2.1. Розробка веб-застосунку та інтерфейсу користувача для моніторингу споживання енергії.....	26
2.2. Аналіз даних, автентифікація та підготовка даних	30
2.3. Система повідомлень, автоматичних звітів та їх налаштування	37
2.4. Архітектура веб-застосунку: база даних, Frontend та API.....	43
Висновки до розділу 2	56
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГНОЗУВАННЯ СПОЖИВАННЯ ЕЛЕКТРОЕНЕРГІЇ ТА ВІДОБРАЖЕННЯ РЕЗУЛЬТАТІВ.....	57
3.1. Інтеграція моделі Chronos для прогнозування, реалізація логіки, взаємодія з користувачем та візуалізація результатів	58
3.2. Генерація та налаштування звітів	63
3.3. Реалізація механізму оновлення прогнозів	64
3.4. Огляд функціональних можливостей веб-застосунку	66

Висновки до розділу 3	72
ВИСНОВКИ	75
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	78
ДОДАТКИ.....	81

ВСТУП

Актуальність теми. В епоху цифрових технологій та підвищеної уваги до питань енергоефективності, розробка інструментів для моніторингу та аналізу споживання електроенергії в побуті набуває особливої актуальності. Сучасний світ стикається з безпрецедентними викликами, пов'язаними зі стрімким зростанням споживання ресурсів, і електроенергія, як один з ключових ресурсів, не є винятком.

Дана робота присвячена розробці веб-застосунку, спрямованого на вирішення цієї актуальної задачі, надаючи користувачам можливість не тільки моніторити, але й детально аналізувати власне енергоспоживання, а також отримувати персоналізовані рекомендації щодо його оптимізації.

Стан розробки проблеми. На сьогоднішній день існує ряд рішень для моніторингу енергоспоживання, проте більшість з них вимагають встановлення спеціального обладнання та прямого підключення до побутових приладів, що може бути складним та дорогавартісним для пересічного користувача. Розроблюваний веб-застосунок пропонує альтернативний підхід, що ґрунтується на аналізі даних, які вводяться користувачем вручну, а також на загальній статистичній інформації про споживання електроенергії в різних типах домогосподарств. Такий підхід робить систему доступною для широкого кола користувачів, незалежно від наявності у них спеціального обладнання, та дозволяє охопити широку аудиторію.

Мета роботи. Розробка зручного, інтуїтивно зрозумілого та ефективного веб-застосунку для моніторингу та оптимізації споживання електроенергії в побуті без необхідності підключення до домашніх приладів.

Для досягнення поставленої мети необхідно вирішити деякі основні завдання. Дослідити існуючі методи моніторингу та аналізу споживання електроенергії. Розробити архітектуру та дизайн веб-застосунку. Створити інтерфейс користувача для введення даних про споживання електроенергії. Розробити систему аналізу споживання на основі статистичних даних та

даних, введених користувачем. Спроекувати та реалізувати механізм генерації персоналізованих рекомендацій щодо зниження споживання електроенергії. Протестувати та впровадити розроблений веб-застосунок. Розробити систему реєстрації та аутентифікації користувачів. Розробити систему збору та збереження даних про споживання електроенергії. Розробити систему аналітики та звітів. Розробити систему нагадувань та сповіщень.

Об'єкт дослідження. Процес споживання електроенергії в побуті.

Предмет дослідження. Методи та алгоритми моніторингу, аналізу та оптимізації споживання електроенергії в побуті на основі даних, введених користувачем, та загальної статистичної інформації.

Наукова новизна роботи. Полягає у розробці нового підходу до моніторингу та оптимізації споживання електроенергії в побуті, що не вимагає встановлення спеціального обладнання та прямого підключення до домашніх приладів. Застосунок базується на аналізі даних, введених користувачем, а також на загальній статистичній інформації про споживання електроенергії в різних типах домогосподарств, що дозволяє охопити широку аудиторію. Також, новизна полягає у розробці алгоритмів для генерації персоналізованих рекомендацій щодо зниження споживання електроенергії на основі зібраних даних.

Практичне значення. Розроблений веб-застосунок дозволить користувачам ефективно моніторити та аналізувати власне енергоспоживання, отримувати персоналізовані рекомендації щодо його оптимізації, що сприятиме зменшенню витрат на електроенергію та зниженню негативного впливу на довкілля. Застосунок може бути використаний як індивідуальними користувачами, так і управителями багатоквартирних будинків, ОСББ, а також енергоаудиторами.

Апробація. За темою кваліфікаційної магістерської роботи було опубліковано одну тезу конференції [1].

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Проблеми та виклики у сфері споживання електроенергії та огляд існуючих методів моніторингу й оптимізації

Сучасне суспільство, яке характеризується високим рівнем технологічного розвитку, комфорту та доступності різноманітних побутових приладів, стикається з низкою серйозних проблем та викликів, пов'язаних зі стрімким зростанням споживання електроенергії в побуті. Це зростання зумовлює необхідність пошуку ефективних шляхів оптимізації та підвищення енергоефективності, як на індивідуальному рівні, так і в масштабах всієї енергосистеми. Одним із ключових факторів, що впливають на збільшення енергоспоживання, є значне зростання кількості побутових електроприладів у сучасних домогосподарствах. Практично кожна сфера побуту сьогодні автоматизована та включає використання різноманітних пристроїв, від холодильників та пральних машин до комп'ютерів, телевізорів та систем освітлення. Окрім зростання кількості, спостерігається тенденція до збільшення потужності окремих приладів та часу їх використання протягом доби, що особливо відчутно в години пікових навантажень на енергосистему.

На рис. 1.1 зображено розподіл споживання електроенергії за категоріями побутових приладів у типовому домогосподарстві, наочно демонструючи, які саме прилади є найбільшими споживачами. Збільшення навантаження на енергосистему призводить не тільки до зростання витрат на комунальні послуги для окремих домогосподарств, але й має серйозні екологічні наслідки. Виробництво електроенергії, особливо з використанням традиційних джерел, таких як вугілля та газ, супроводжується значними викидами парникових газів, що посилює негативний вплив на навколишнє середовище та прискорює зміни клімату. Таким чином, питання енергоефективності перестає бути суто економічним і набуває глобального екологічного значення.



Рис. 1.1. Розподіл споживання електроенергії за категоріями побутових приладів у типовому домогосподарстві.

Неефективне використання електроенергії також має вагомі економічні наслідки для домогосподарств. Зростання споживання веде до збільшення витрат на комунальні послуги, що стає особливо відчутним в умовах постійного зростання тарифів на електроенергію. При цьому, проблема не обмежується лише кількістю та потужністю приладів. Значну роль відіграють і неефективні звички споживання електроенергії, такі як залишення приладів у режимі очікування, використання застарілого та енергоємного обладнання, недостатня теплоізоляція житла, нераціональне використання освітлення та інші фактори. Всі ці аспекти в сукупності призводять до марнотратного використання електроенергії та збільшення фінансового навантаження на сімейні бюджети.

Однією з ключових перешкод на шляху до ефективного управління енергоспоживанням є недостатня обізнаність самих споживачів про структуру та обсяги власного енергоспоживання. Багато людей не мають чіткого уявлення про те, які саме прилади та аспекти їхньої поведінки призводять до найбільших витрат електроенергії у їхньому домогосподарстві. Така необізнаність унеможливорює цілеспрямовану оптимізацію, оскільки

користувачі не знають, на що саме потрібно звернути увагу в першу чергу. Відсутність детальної інформації про споживання значно ускладнює процес прийняття рішень щодо зміни поведінки, впровадження енергозберігаючих заходів та вибору енергоефективного обладнання. Крім того, традиційні методи обліку електроенергії, такі як щомісячне зняття показників лічильника та ведення записів у паперовому вигляді або за допомогою табличних редакторів, є незручними, трудомісткими і не надають достатньої оперативності, деталізації та наочності даних. Вони не дозволяють проводити глибокий аналіз структури споживання, виявляти приховані резерви економії та відстежувати ефективність вжитих заходів у динаміці.

Вирішення вищезгаданих проблем потребує комплексного підходу, що поєднує в собі як впровадження новітніх технологій для моніторингу, аналізу та управління споживанням, так і заходи з підвищення рівня обізнаності та освіченості споживачів щодо раціонального використання електроенергії. На сьогоднішній день існує широкий спектр методів та інструментів для моніторингу та оптимізації споживання електроенергії, які відрізняються за рівнем складності, вартістю впровадження, ефективністю та ступенем автоматизації.

Одним з найпоширеніших та найбільш технологічно просунутих методів є використання інтелектуальних лічильників, так званих "smart meters". Ці пристрої дозволяють в режимі реального часу збирати детальні дані про споживання електроенергії та автоматично передавати їх до енергопостачальної компанії через мережу зв'язку. Інтелектуальні лічильники надають можливість не тільки точно вимірювати обсяги споживання з високою дискретністю, але й відстежувати його динаміку протягом доби, виявляти періоди пікового навантаження, аналізувати структуру споживання та диференціювати тарифи в залежності від часу доби. Це, в свою чергу, стимулює користувачів до зміщення споживання на нічний час, коли тарифи є нижчими, що дозволяє не тільки економити кошти, але й вирівнювати графік навантаження на енергосистему.

Іншим перспективним методом, що набуває все більшої популярності, є використання спеціалізованих пристроїв, які підключаються до окремих електроприладів або груп приладів та надають детальну інформацію про їхнє енергоспоживання. Такі пристрої, які часто називають "розумними розетками" або "енергомоніторами", дозволяють користувачам точно визначити, які саме прилади є найбільш енергоємними, та оптимізувати їх використання. Наприклад, користувач може вимикати прилади з мережі, коли вони не використовуються, замість того, щоб залишати їх у режимі очікування, або ж замінювати застаріле та енергоємне обладнання на більш сучасні та енергоефективні моделі. В рис. 1.2 продемонстровано приклад інтерфейсу мобільного додатку, який дозволяє користувачам відстежувати енергоспоживання окремих приладів у режимі реального часу.



Рис. 1.2. Приклад інтерфейсу мобільного додатку для моніторингу споживання електроенергії.

Однак, незважаючи на свою високу ефективність та точність, методи, що базуються на використанні інтелектуальних лічильників та спеціалізованих пристроїв, часто вимагають значних інвестицій у додаткове обладнання, його встановлення, налаштування та інтеграцію з існуючою інфраструктурою. Це може бути суттєвим бар'єром для багатьох домогосподарств, особливо з обмеженим бюджетом, які не мають можливості інвестувати значні кошти в

подібні системи. Крім того, встановлення та налаштування інтелектуальних лічильників та "розумних розеток" може потребувати залучення кваліфікованих фахівців, що тягне за собою додаткові витрати та незручності для користувачів. Альтернативним підходом, що набуває все більшого поширення в останні роки, є використання програмних рішень, таких як мобільні додатки та веб-застосунки, які не потребують встановлення додаткового обладнання.

Ці рішення дозволяють користувачам вручну вводити дані про своє споживання, наприклад, показники звичайних електромеханічних або електронних лічильників, та отримувати доступ до аналітики та персоналізованих рекомендацій щодо оптимізації. Такий підхід є більш доступним та гнучким, оскільки не вимагає придбання та встановлення додаткового обладнання, а також може бути адаптований до різних типів лічильників та систем обліку, що існують у різних країнах та регіонах.

Окрім технологічних рішень, існує також цілий ряд поведінкових методів оптимізації споживання електроенергії, які спрямовані на зміну звичок та моделей поведінки користувачів. До таких методів належить, наприклад, вимикання світла у приміщеннях, які не використовуються, відключення приладів від мережі замість використання режиму очікування, використання енергоефективних лампочок та побутової техніки, а також раціональне використання електроприладів, що потребують нагріву або охолодження, таких як пральні та посудомийні машини, бойлери, кондиціонери тощо. Ці методи не потребують значних фінансових інвестицій і можуть бути легко впроваджені будь-яким користувачем, проте їх ефективність значною мірою залежить від рівня обізнаності, мотивації та дисциплінованості самих користувачів.

Незважаючи на значну кількість існуючих методів моніторингу та оптимізації споживання електроенергії, їх широке впровадження та використання стикається з низкою проблем. Як вже згадувалося раніше, інтелектуальні лічильники та спеціалізовані пристрої, такі як "розумні

розетки", вимагають значних фінансових витрат на обладнання, його встановлення та налаштування, а також інтеграцію з існуючими системами. Це створює суттєві бар'єри для домогосподарств із обмеженим бюджетом, які не завжди можуть дозволити собі подібні інвестиції. Більше того, налаштування таких систем часто потребує залучення кваліфікованих фахівців, що додає додаткових витрат і складнощів. Навіть програмні рішення, які є більш доступними завдяки відсутності необхідності встановлення обладнання, все ще залишаються незручними для користувачів, оскільки потребують ручного введення даних, що займає час і може бути помилковим через людський фактор. Багато користувачів не готові витрачати свій час на регулярне введення даних, особливо якщо вони не бачать миттєвої вигоди від цього. Крім того, деякі користувачі можуть мати обмежені навички роботи з комп'ютером або мобільними пристроями, що ускладнює для них використання веб-застосунків та мобільних додатків.

Таким чином, незважаючи на значний прогрес у розробці технологій та методів для моніторингу та оптимізації енергоспоживання, залишається потреба у створенні більш доступних, зручних та ефективних рішень, які б враховували потреби та можливості широкого кола користувачів. Перевага запропонованого у даній роботі рішення полягає у створенні більш доступного, гнучкого та економічного способу оптимізації енергоспоживання, який не потребує значних інвестицій чи спеціальних знань. Використання інтегрованих технологій дозволяє об'єднати переваги автоматизованих систем моніторингу та програмних інструментів, забезпечуючи простоту використання, зручність і доступність для широкого кола користувачів. Це рішення спрямоване на подолання існуючих недоліків, зокрема високих витрат і складності впровадження, стимулюючи користувачів до раціонального споживання електроенергії без значних фінансових витрат чи зусиль.

1.2. Постановка задачі дослідження та вибір засобів реалізації веб-застосунку

Основною метою даного дослідження є розробка ефективного, зручного та доступного веб-застосунку, який дозволить користувачам здійснювати моніторинг та оптимізацію власного споживання електроенергії в побуті. Важливою вимогою є те, що застосунок має функціонувати без необхідності прямого підключення до домашніх приладів чи встановлення складного та дорогого обладнання, що робить його доступним для широкого кола користувачів, незалежно від їх технічних знань, фінансових можливостей та наявності спеціального обладнання.

Для досягнення цієї мети необхідно вирішити ряд взаємопов'язаних завдань, які охоплюють різні аспекти розробки та функціонування веб-застосунку. По-перше, слід розробити інтуїтивно зрозумілий, зручний та привабливий інтерфейс користувача, який дозволить легко вводити дані про споживання електроенергії, тарифи та інші важливі параметри, що впливають на розрахунки та аналітику. Інтерфейс має бути адаптованим для різних пристроїв, включаючи смартфони та планшети, та забезпечувати максимально простий, швидкий та інтуїтивно зрозумілий процес введення даних, мінімізуючи можливість помилок та дублювання інформації, а також надавати підказки та інструкції для користувачів.

По-друге, необхідно створити систему аналізу введених даних, яка буде здатна виявляти тенденції, аномалії та закономірності у споживанні електроенергії. Ця система має враховувати не лише абсолютні значення споживання, але й динаміку змін, а також, за можливості, порівнювати дані користувача з середніми показниками по регіону чи серед схожих домогосподарств, що дозволить користувачеві краще зрозуміти своє місце у загальній картині споживання та отримати орієнтири для оптимізації. На основі проведеного аналізу система повинна надавати користувачеві детальну інформацію про структуру його споживання, виділяючи основні фактори, що

впливають на витрати електроенергії, та ідентифікуючи потенційні можливості для економії.

По-третє, важливою складовою є розробка механізму генерації персоналізованих рекомендацій щодо оптимізації споживання, які б враховували індивідуальні особливості домогосподарства, звички користувачів та їхні можливості щодо впровадження енергозберігаючих заходів. Рекомендації мають бути чіткими, зрозумілими, практично застосовними та адаптованими до конкретної ситуації користувача, а також супроводжуватися оцінкою потенційного економічного ефекту від їх впровадження, щоб мотивувати користувача до їх виконання.

Додатково, веб-застосунок повинен забезпечувати можливість зберігання історії споживання електроенергії за тривалий період часу, що дозволить користувачам відстежувати свій прогрес у досягненні цілей з енергоефективності, аналізувати ефективність впроваджених заходів та коригувати свою поведінку у разі потреби. Також, необхідно передбачити можливість інтеграції з іншими системами та сервісами, наприклад, з системами електронних рахунків від енергопостачальних компаній або з платформами "розумного будинку", що дозволить автоматизувати процес імпорту даних, розширити можливості аналізу та підвищити точність прогнозування.

Окрім того, слід забезпечити можливість гнучкого налаштування параметрів моніторингу та отримання сповіщень, що дозволить користувачам адаптувати роботу застосунку під свої потреби та вподобання, обираючи найбільш зручний для себе режим взаємодії з системою та отримуючи тільки ту інформацію, яка є для них важливою та актуальною.

Усі ці завдання повинні бути вирішені з урахуванням вимог до безпеки, надійності та масштабованості системи, а також з дотриманням принципів захисту персональних даних користувачів. Необхідно забезпечити конфіденційність та цілісність даних, що зберігаються та обробляються в системі, а також захист від несанкціонованого доступу та зловмисних дій,

використовуючи сучасні методи шифрування та аутентифікації. Розробка повинна вестися з урахуванням принципів "security by design" та "privacy by design", що передбачає інтеграцію заходів безпеки на всіх етапах життєвого циклу програмного продукту, починаючи з етапу проектування і закінчуючи тестуванням та впровадженням. Результатом дослідження має стати повнофункціональний, зручний, безпечний та надійний веб-застосунок, який допоможе користувачам не тільки знизити споживання електроенергії та зменшити витрати на комунальні послуги, але й підвищити рівень екологічної свідомості та відповідальності, сприяючи формуванню культури ощадливого ставлення до ресурсів та сталого розвитку суспільства.

Для реалізації веб-застосунку, що відповідає поставленим вимогам та меті дослідження, було обрано ряд сучасних технологій та інструментів, які забезпечують високу продуктивність, масштабованість, зручність розробки, підтримки та подальшого розвитку продукту. При виборі засобів реалізації особлива увага приділялася таким критеріям, як надійність, безпека, швидкодія, наявність розвиненої спільноти та документації, а також можливість інтеграції з іншими сервісами та системами, що можуть бути корисними для розширення функціоналу в майбутньому.

В якості основної мови програмування для backend частини було обрано Python, який є потужною, гнучкою та популярною мовою з великою кількістю бібліотек для обробки даних, машинного навчання та створення веб-додатків. Його простий та зрозумілий синтаксис, а також наявність великої спільноти розробників, значно прискорюють процес розробки та полегшують підтримку коду. Для розробки REST API було обрано фреймворк FastAPI, який відрізняється високою швидкістю роботи, автоматичною генерацією документації, підтримкою асинхронності та сучасних стандартів програмування, що є важливим для забезпечення високої продуктивності та масштабованості застосунку.

Для розробки frontend частини було обрано мову TypeScript, яка є надбудовою над JavaScript і додає статичну типізацію, що дозволяє підвищити

надійність коду, спростити його підтримку та зменшити кількість помилок на етапі розробки. В якості фреймворку для створення інтерфейсу користувача було обрано React, який є однією з найпопулярніших бібліотек для розробки сучасних веб-додатків, завдяки своїй гнучкості, продуктивності та компонентній архітектурі. React дозволяє створювати складні, динамічні та інтерактивні інтерфейси, забезпечуючи при цьому високу швидкість роботи та зручність розробки, а також має велику спільноту розробників та детальну документацію.

Для керування станом додатку та оптимізації роботи з асинхронними запитами було використано бібліотеку React Query, яка значно спрощує взаємодію з сервером, кешування даних та оновлення інтерфейсу в режимі реального часу, а також надає зручні інструменти для роботи з запитами та мутаціями. Для стилізації компонентів та забезпечення адаптивного дизайну, що коректно відображається на різних пристроях, було обрано бібліотеку Chakra UI, яка надає широкий набір готових компонентів та інструментів для створення привабливого, зручного та доступного інтерфейсу користувача, а також дозволяє легко налаштовувати теми та стилі відповідно до вимог проекту.

Для зберігання даних було обрано реляційну систему керування базами даних PostgreSQL, яка є надійною, масштабованою та підтримує розширені можливості роботи з даними, включаючи JSONB для зберігання неструктурованих даних та PostGIS для роботи з геопросторовими даними, що може бути корисним для реалізації додаткового функціоналу в майбутньому, наприклад, для візуалізації даних на карті. Взаємодія між backend частиною та базою даних здійснюється за допомогою ORM (Object-Relational Mapping) SQLAlchemy, яка базується на SQLAlchemy та Pydantic. SQLAlchemy дозволяє зручно працювати з даними, використовуючи об'єктно-орієнтований підхід, описувати структуру бази даних за допомогою класів Python та автоматично генерувати SQL-запити, що спрощує розробку та підвищує надійність коду. Крім того, SQLAlchemy забезпечує автоматичну валідацію даних, що надходять

від клієнта, на відповідність заданим типам та обмеженням, що підвищує безпеку та цілісність даних, а також дозволяє виявляти помилки на ранніх стадіях.

Для забезпечення безпеки та автентифікації користувачів застосовано JWT токени (JSON Web Tokens), які є стандартом де-факто для сучасних веб-додатків. JWT дозволяє безпечно передавати дані між клієнтом та сервером у вигляді компактного та самодостатнього JSON-об'єкта, що містить всю необхідну інформацію про користувача та його права доступу, а також цифровий підпис, що гарантує цілісність та автентичність даних. Обраний стек технологій дозволяє створити сучасний, швидкий, надійний та масштабований веб-застосунок, здатний ефективно вирішувати поставлені завдання з моніторингу та оптимізації споживання електроенергії, а також забезпечує гнучкість для подальшого розвитку та розширення функціоналу відповідно до потреб користувачів та змін у предметній області.

1.3. Вплив інформаційних технологій на підвищення енергоефективності

Інформаційні технології відіграють дедалі важливішу роль у підвищенні енергоефективності в побутовому секторі, надаючи користувачам нові можливості для моніторингу, аналізу та оптимізації споживання електроенергії. Сучасні веб-застосунки, мобільні додатки та пристрої Інтернету речей (IoT) забезпечують збір детальних даних про енергоспоживання в режимі реального часу. Це дозволяє виявляти неефективні моделі використання ресурсів та ідентифікувати потенціал для економії. Завдяки інтеграції з інтелектуальними лічильниками та іншими датчиками, користувачі отримують доступ до гранульованої інформації про споживання окремих приладів. Як наслідок, вони можуть зрозуміти, які саме пристрої та аспекти їхньої поведінки призводять до найбільших витрат. Крім того, застосування алгоритмів машинного навчання та штучного інтелекту для аналізу зібраних даних дозволяє виявляти приховані закономірності та

формувати персоналізовані рекомендації щодо оптимізації, враховуючи індивідуальні особливості кожного домогосподарства.

Візуалізація даних у вигляді зручних та інформативних графіків і діаграм є ще одним важливим аспектом, який сприяє кращому розумінню структури енергоспоживання та полегшує прийняття рішень щодо енергозбереження. Можливість переглядати дані в розрізі різних часових періодів, порівнювати поточне споживання з минулими періодами або з середніми показниками по регіону, а також відстежувати прогрес у досягненні поставлених цілей, допомагає користувачам більш усвідомлено ставитися до використання електроенергії.

Застосування гейміфікації, соціальних механік та систем зворотного зв'язку, таких як сповіщення про досягнення цілей, персоналізовані поради та порівняння з іншими користувачами, додатково стимулює залучення користувачів до процесу оптимізації та сприяє формуванню екологічно свідомої поведінки. Розвиток мобільних технологій та поширення смартфонів роблять ці інструменти доступними для широкого кола користувачів, дозволяючи їм контролювати своє енергоспоживання будь-де та будь-коли, що є особливо зручним у сучасному динамічному світі.

Окрім безпосереднього впливу на поведінку користувачів, інформаційні технології також сприяють підвищенню ефективності роботи енергосистем в цілому. Збір та аналіз агрегованих даних про споживання електроенергії в різних домогосподарствах дозволяє енергопостачальним компаніям краще прогнозувати попит, оптимізувати виробництво та розподіл електроенергії, а також впроваджувати програми управління попитом.

Ці дані є цінним ресурсом для планування розвитку інфраструктури та впровадження нових технологій, спрямованих на підвищення енергоефективності. Інтеграція з системами "розумного будинку" та пристроями IoT відкриває нові можливості для автоматизації процесів енергозбереження, таких як автоматичне регулювання освітлення, опалення та

кондиціонування в залежності від присутності людей у приміщенні, погодних умов та інших факторів.

Варто зазначити, що розвиток інформаційних технологій у сфері енергоефективності не обмежується лише збором та аналізом даних. Перспективним напрямком є створення інтелектуальних систем управління енергоспоживанням, які здатні не тільки надавати рекомендації, але й автоматично керувати роботою побутових приладів, оптимізуючи їх використання в залежності від поточних тарифів, наявності електроенергії з відновлюваних джерел та індивідуальних потреб користувача.

Такі системи можуть самостійно вимикати прилади, що не використовуються, регулювати температуру в приміщенні, а також перемикає споживання на нічний час, коли тарифи на електроенергію є нижчими. Це дозволить досягти значної економії ресурсів та знизити витрати на комунальні послуги без необхідності активної участі користувача в процесі управління.

Впровадження інформаційних технологій у сферу енергоефективності стикається з певними викликами, такими як забезпечення конфіденційності та безпеки даних, розробка зручних та інтуїтивно зрозумілих інтерфейсів, а також подолання опору користувачів, пов'язаного з недовірою до нових технологій або небажанням змінювати звичні моделі поведінки.

Для подолання цих викликів необхідні комплексні рішення, що включають не тільки розробку технологічних платформ, але й проведення просвітницької роботи серед населення, демонстрацію переваг та вигод від використання нових інструментів, а також забезпечення підтримки та навчання користувачів. Важливим фактором успіху є також співпраця між розробниками технологій, енергопостачальними компаніями та органами державної влади, що дозволить створити сприятливе середовище для впровадження інновацій та стимулюватиме користувачів до активної участі в процесі оптимізації енергоспоживання.

В перспективі, розвиток технологій розподіленої генерації та зберігання енергії, поєднаний з інтелектуальними системами управління, дозволить створити більш гнучкі та стійкі енергосистеми, що базуються на відновлюваних джерелах енергії та активно залучають споживачів до процесу балансування попиту та пропозиції. Інформаційні технології відіграватимуть ключову роль у цьому процесі, забезпечуючи ефективний обмін даними між усіма учасниками системи, автоматизуючи процеси управління та оптимізації, а також надаючи користувачам зручні інструменти для моніторингу та контролю власного енергоспоживання. Це сприятиме не тільки підвищенню енергоефективності, але й зростанню частки відновлюваних джерел енергії в загальному енергобалансі, що є важливим кроком на шляху до сталого розвитку та боротьби зі змінами клімату.

1.4. Проблеми впровадження існуючих рішень

Існуючі рішення для моніторингу та оптимізації енергоспоживання, попри свої потенційні переваги, стикаються з низкою проблем, що ускладнюють їх широке впровадження та обмежують ефективність. Однією з ключових перешкод є висока вартість обладнання та його встановлення, особливо коли мова йде про інтелектуальні лічильники та спеціалізовані пристрої для моніторингу, такі як "розумні розетки". Ці витрати можуть бути непосильними для багатьох домогосподарств, особливо з низьким рівнем доходу, що створює бар'єри для доступу до сучасних технологій енергозбереження та поглиблює існуючу нерівність. Додатково, встановлення та налаштування такого обладнання часто вимагає залучення кваліфікованих фахівців, що тягне за собою додаткові витрати та незручності для користувачів, які не завжди готові до складних технічних процедур.

Складність інтеграції існуючих рішень з різноманітними системами обліку електроенергії та побутовими приладами є ще одним суттєвим бар'єром. Відсутність єдиних стандартів та протоколів обміну даними між різними

пристроями та платформами ускладнює створення комплексних систем моніторингу та управління, що охоплюють усі аспекти енергоспоживання в домогосподарстві. Користувачі часто стикаються з несумісністю обладнання різних виробників, необхідністю використання окремих додатків для кожного пристрою та відсутністю можливості централізованого збору та аналізу даних. Це призводить до фрагментації інформації, зниження зручності використання та, як наслідок, зменшення мотивації користувачів до впровадження таких систем.

Навіть програмні рішення, які є більш доступними з фінансової точки зору, оскільки не вимагають встановлення додаткового обладнання, часто мають свої недоліки, що обмежують їх ефективність. Однією з таких проблем є необхідність ручного введення даних користувачами, що є трудомістким процесом та може призводити до помилок і неточностей в даних, особливо при тривалому використанні. Людський фактор відіграє значну роль, оскільки користувачі можуть забувати вчасно вводити дані, робити помилки при введенні або просто втрачати інтерес до процесу через його монотонність. Відсутність автоматизації збору даних знижує оперативність та достовірність інформації, а також зменшує загальну зручність використання таких програмних рішень.

Питання безпеки та конфіденційності даних є ще одним важливим аспектом, який викликає занепокоєння у користувачів та стримує впровадження існуючих рішень. Збір та зберігання детальної інформації про споживання електроенергії в домогосподарствах створює потенційні ризики, пов'язані з витоком даних, несанкціонованим доступом та використанням їх у зловмисних цілях. Недостатня прозорість політик конфіденційності та відсутність чітких механізмів контролю за використанням даних з боку користувачів посилюють ці побоювання. Багато людей не готові довіряти свої персональні дані, особливо коли мова йде про інформацію, яка може бути використана для ідентифікації їхніх звичок та розпорядку дня, третім сторонам, не маючи при цьому гарантій їхньої безпеки.

Окрім технічних та економічних аспектів, суттєвою проблемою є недостатня обізнаність та низький рівень мотивації користувачів щодо впровадження енергоефективних заходів. Багато людей не усвідомлюють потенціал економії від оптимізації свого енергоспоживання або не мають достатніх знань та навичок для ефективного використання існуючих інструментів. Відсутність чітких та зрозумілих стимулів, а також недостатня інформаційна підтримка з боку держави та енергопостачальних компаній, призводять до того, що користувачі не бачать сенсу у впровадженні нових рішень, особливо якщо це вимагає від них додаткових зусиль або фінансових витрат. Подолання цього бар'єру потребує комплексного підходу, що включає не тільки розробку технологічних рішень, але й проведення освітніх кампаній, спрямованих на підвищення обізнаності та формування культури енергозбереження.

Висновки до розділу 1

У першому розділі було проведено детальний аналіз предметної області, що дозволило виявити основні проблеми та виклики у сфері споживання електроенергії в побуті, а також сформулювати вимоги до веб-застосунку, який би допомагав користувачам моніторити та оптимізувати своє енергоспоживання. Було встановлено, що недостатня обізнаність користувачів про власне енергоспоживання, а також відсутність зручних інструментів для його моніторингу та аналізу є ключовими факторами, що перешкоджають ефективній оптимізації. В ході аналізу було розглянуто існуючі методи моніторингу та оптимізації, такі як інтелектуальні лічильники, спеціалізовані пристрої та програмні рішення, а також проаналізовано їх переваги та недоліки. Визначено, що розробка веб-застосунку, який не потребує прямого підключення до приладів, а базується на даних, введених користувачем, є перспективним напрямком для вирішення існуючих проблем, оскільки такий підхід забезпечує доступність та гнучкість рішення для широкого кола

користувачів, незалежно від їх технічної підготовки та наявності спеціального обладнання.

На основі проведеного аналізу було сформульовано мету та завдання дослідження, які полягають у розробці зручного, ефективного та доступного веб-застосунку для моніторингу та оптимізації споживання електроенергії. Було детально обґрунтовано вибір засобів реалізації, зокрема мов програмування Python та TypeScript, фреймворків FastAPI та React, системи керування базами даних PostgreSQL, а також інших інструментів та бібліотек. Визначено, що такий стек технологій дозволить створити сучасний, швидкий, надійний, безпечний та масштабований продукт, що відповідає вимогам користувачів та цілям дослідження. Особливу увагу було приділено питанням безпеки, забезпечення цілісності даних та зручності використання, що є критично важливими для успішної реалізації проекту та задоволення потреб користувачів.

В результаті детального аналізу предметної області було визначено ключові аспекти, які необхідно врахувати при розробці веб-застосунку, такі як зручність інтерфейсу, можливості аналізу даних, персоналізація рекомендацій та інтеграція з іншими системами. Було проведено ґрунтовне дослідження існуючих рішень та технологій, що дозволило обрати найбільш оптимальні засоби для реалізації проекту. Обґрунтування вибору технологій та інструментів базується на ретельному аналізі їх можливостей, переваг та відповідності поставленим завданням. Обрані технології, такі як Python, FastAPI, TypeScript, React, PostgreSQL та інші, є сучасними, надійними та ефективними рішеннями, що дозволяють створити продукт, який відповідає найвищим стандартам якості та вимогам користувачів, а також забезпечує гнучкість для подальшого розвитку та масштабування.

РОЗДІЛ 2. РОЗРОБКА ТА ПРОЕКТУВАННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ МОНІТОРИНГУ ТА ОПТИМІЗАЦІЇ СПОЖИВАННЯ ЕЛЕКТРОЕНЕРГІЇ

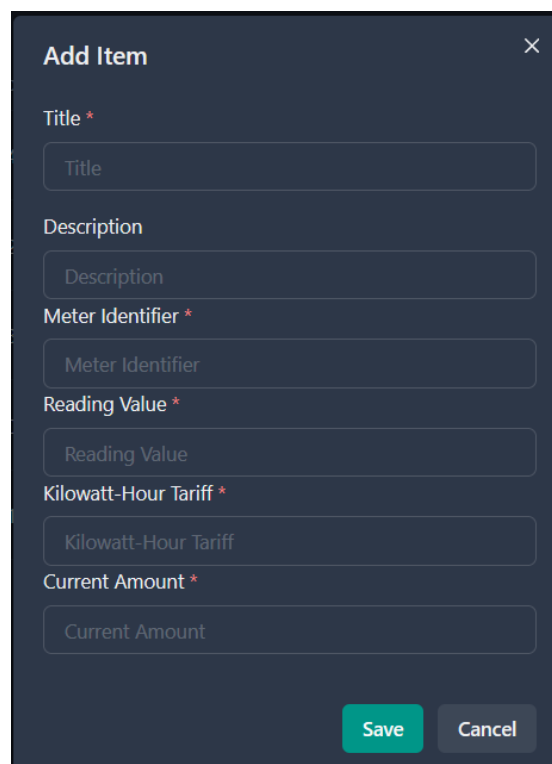
2.1. Розробка веб-застосунку та інтерфейсу користувача для моніторингу споживання енергії

Розробка веб-застосунку для моніторингу та оптимізації споживання енергії - це комплексний, багатоетапний процес, що вимагає ретельного планування та чіткого дотримання обраної методології. Одним із першочергових завдань є створення серверної частини, або backend, яка відповідає за обробку даних, взаємодію з базою даних, реалізацію бізнес-логіки та забезпечення безпеки. Для швидкої та ефективної розробки REST API (Representational State Transfer Application Programming Interface), через який клієнтська частина (frontend) обмінюватиметься даними, було обрано фреймворк FastAPI. На цьому етапі визначаються ключові сутності системи, такі як "користувач", "показник споживання", "тариф", "звіт", "рекомендація", а також розробляються відповідні моделі даних та методи їх обробки, включаючи CRUD операції (Create, Read, Update, Delete).

Другим, не менш важливим етапом, є розробка клієнтської частини, яка забезпечує зручний інтерфейс взаємодії з користувачем та ефективну візуалізацію даних. Для цього було обрано бібліотеку React, одну з найпопулярніших JavaScript бібліотек для побудови інтерфейсів, у поєднанні з TypeScript, що додає статичну типізацію. Такий стек технологій дозволяє створити зручний, динамічний, адаптивний та швидкий інтерфейс, який коректно відображається на різних пристроях, включаючи комп'ютери, планшети та смартфони. На етапі розробки frontend частини створюються компоненти для введення даних, відображення інформації у вигляді таблиць, графіків та діаграм, візуалізації аналітики та генерації звітів. Особлива увага приділяється забезпеченню інтуїтивно зрозумілого та дружнього до

користувача дизайну, що сприятиме легкому освоєнню та використанню застосунку.

Одним з ключових елементів веб-застосунку, що безпосередньо впливає на зручність, простоту використання та ефективність його роботи, є інтерфейс користувача, призначений для введення даних про споживання електроенергії. Цей інтерфейс має бути максимально простим, інтуїтивно зрозумілим та зручним, щоб мінімізувати зусилля, необхідні для введення інформації, та забезпечити точність і повноту даних. Для досягнення цієї мети було розроблено спеціальну форму (рис. 2.1), яка містить поля для введення показників лічильника, дати та часу зняття показників, а також іншої необхідної інформації, такої як діючі тарифи на електроенергію та ідентифікатор адреси користувача. Передбачено можливість вибору дати з інтерактивного календаря.



Add Item ×

Title *

Title

Description

Description

Meter Identifier *

Meter Identifier

Reading Value *

Reading Value

Kilowatt-Hour Tariff *

Kilowatt-Hour Tariff

Current Amount *

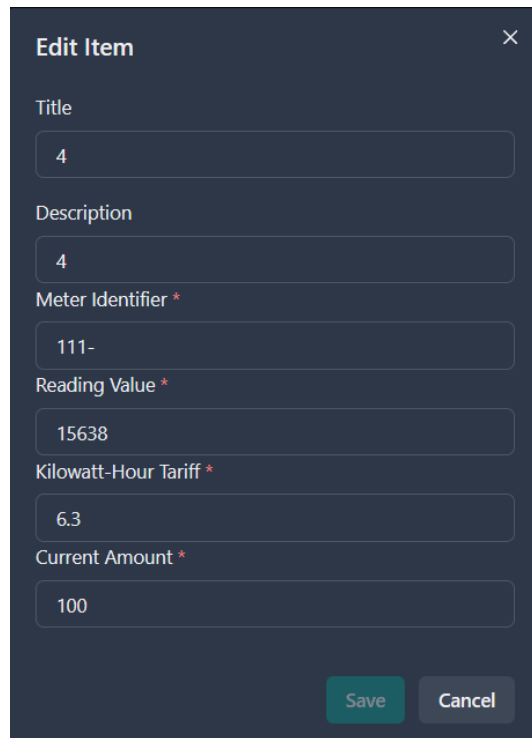
Current Amount

Save Cancel

Рис 2.1 Введення даних веб-застосунку з полями для показників лічильника

Для забезпечення зручності та швидкості введення даних, реалізовано базовий функціонал, що дозволяє користувачам легко вносити показники лічильника. Користувач може вказати ідентифікатор адреси, безпосередньо

сам показник лічильника, тариф, що діє на момент зняття показника, а також суму до сплати. Хоча розширений функціонал збереження шаблонів наразі не передбачений, базова логіка дозволяє користувачам вводити всі необхідні дані про споживання. Крім того, система надає можливість редагувати та видаляти раніше введені дані, що забезпечує гнучкість та можливість виправлення помилок у разі їх виникнення (рис. 2.2).



The image shows a dark-themed 'Edit Item' dialog box. It contains the following fields and values:

- Title: 4
- Description: 4
- Meter Identifier *: 111-
- Reading Value *: 15638
- Kilowatt-Hour Tariff *: 6.3
- Current Amount *: 100

At the bottom right, there are two buttons: 'Save' (highlighted in green) and 'Cancel'.

Рис 2.2 Редагування раніше введених даних

Важливим аспектом розробленого інтерфейсу є реалізація функціоналу валідації введених даних, який здійснює перевірку їх на коректність, відповідність заданим форматам та допустимим значенням. Наприклад, система перевіряє, чи є введені показники лічильника числовими значеннями, чи відповідає дата зняття показників поточному часовому проміжку. У разі виявлення помилки або невідповідності, користувачу миттєво виводиться відповідне повідомлення з підказкою щодо її виправлення, що допомагає уникнути некоректних даних та забезпечує цілісність інформації в системі. Валідація даних відбувається як на стороні клієнта, за допомогою JavaScript, що забезпечує миттєвий зворотній зв'язок з користувачем та покращує

загальний досвід використання, так і на стороні сервера, за допомогою оголошених моделей і типів запитів, що гарантує цілісність та достовірність даних, які зберігаються в базі.

Інтерфейс користувача розроблено з урахуванням принципів адаптивного дизайну, що дозволяє йому коректно відображатися та функціонувати як на настільних комп'ютерах з великими екранами, так і на мобільних пристроях з екранами меншого розміру, таких як планшети та смартфони. Це забезпечує доступність застосунку для широкого кола користувачів, незалежно від того, яким пристроєм вони користуються. Адаптивний дизайн передбачає автоматичну зміну розташування та розміру елементів інтерфейсу залежно від розміру екрану, а також підтримку сенсорного вводу для мобільних пристроїв. Розробка інтерфейсу велася з урахуванням принципів юзабіліті та з використанням сучасних підходів до дизайну, що забезпечує інтуїтивно зрозумілу навігацію, логічне розташування елементів керування та приємний візуальний вигляд, який не перевантажує користувача зайвою інформацією.

В процесі розробки особлива увага приділялася питанням безпеки та захисту даних, що є критично важливим аспектом для будь-якого веб-застосунку, який працює з персональною інформацією користувачів. Для аутентифікації та авторизації використовуються сучасні методи, такі як JWT-токени (JSON Web Tokens), що дозволяє забезпечити безпечний доступ до функціоналу застосунку та захистити дані від несанкціонованого доступу. Всі дані, що передаються між клієнтською та серверною частинами, шифруються з використанням протоколу HTTPS, який гарантує конфіденційність та цілісність інформації.

Для забезпечення цілісності та конфіденційності даних, що зберігаються в базі даних, застосовуються механізми резервного копіювання, розмежування доступу та шифрування. Окрім основних етапів, важливим аспектом розробки є ретельне тестування та налагодження застосунку, що дозволяє виявити та усунути помилки, а також переконатися у відповідності продукту вимогам, що

були сформульовані на етапі аналізу. Для забезпечення якості програмного продукту використовуються різні методи тестування, включаючи модульне, інтеграційне, функціональне та приймальне тестування. Автоматизовані тести дозволяють швидко виявляти та виправляти помилки, а також забезпечувати стабільну роботу застосунку після внесення змін до коду або оновлення компонентів.

Ретельне тестування на різних етапах розробки дозволяє гарантувати надійність, безпеку, зручність використання та відповідність застосунку вимогам, що були визначені на етапі аналізу предметної області, а також забезпечити високий рівень задоволеності користувачів. В цілому, розроблений інтерфейс користувача для введення даних про споживання електроенергії є зручним, інтуїтивно зрозумілим та ефективним інструментом, який дозволяє користувачам легко вносити необхідну інформацію, мінімізуючи при цьому ймовірність помилок. Завдяки продуманому дизайну, адаптивності та функціоналу валідації даних, цей інтерфейс є важливою складовою веб-застосунку, що забезпечує його ефективність та зручність для користувачів.

2.2. Аналіз даних, автентифікація та підготовка даних

Важливою та невід'ємною частиною розробленого веб-застосунку є система аналізу споживання електроенергії, яка базується на зібраних статистичних даних та надає користувачам цінну інформацію для прийняття обґрунтованих рішень щодо оптимізації власного енергоспоживання. Ця система дозволяє виявляти тенденції, аномалії, закономірності та взаємозв'язки у споживанні, що є основою для формування персоналізованих рекомендацій, оцінки ефективності вжитих заходів та досягнення цілей з енергозбереження (рис. 2.3) (рис. 2.4).

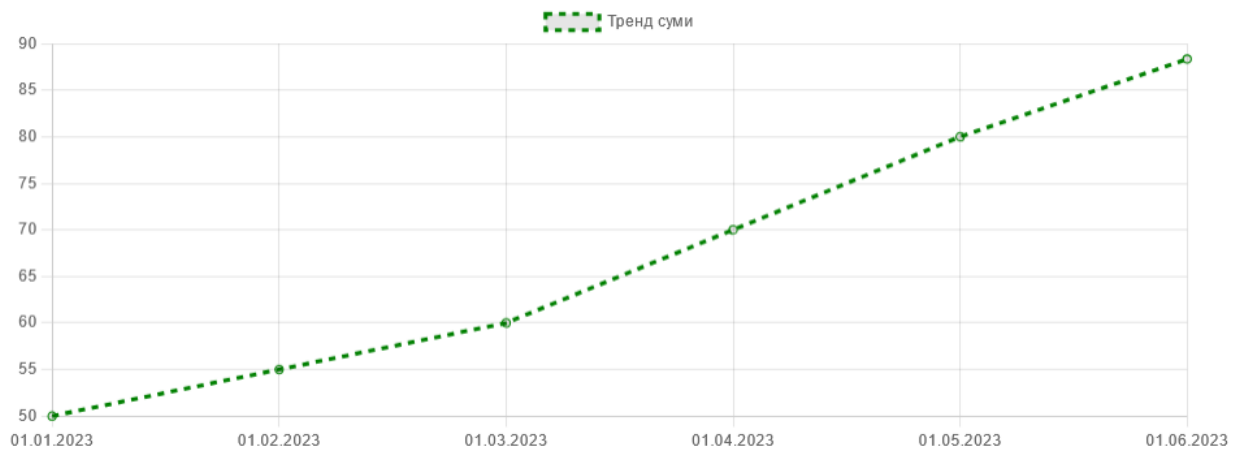


Рис 2.3. Приклад графіку тренду суми

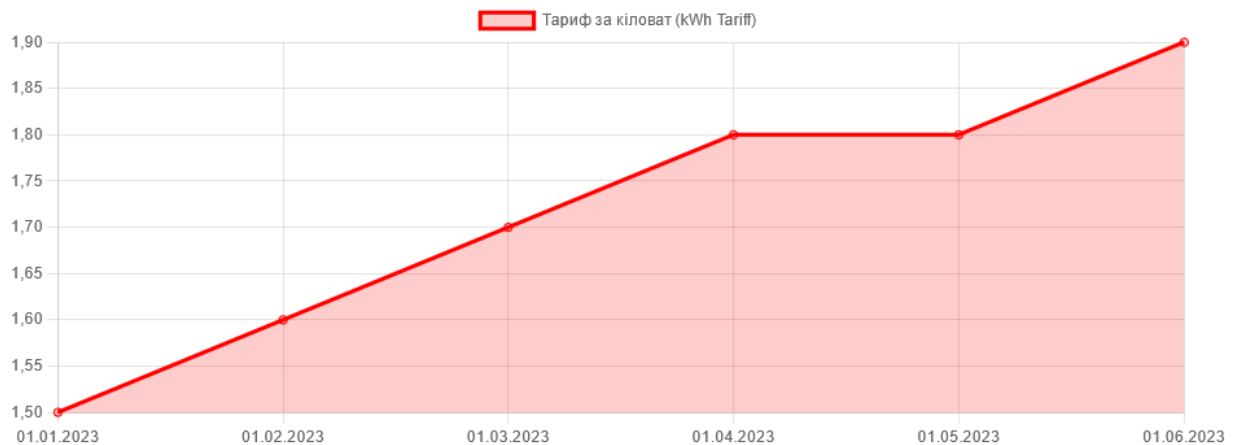


Рис 2.4. Приклад графіку тренду тарифу

Для реалізації цього функціоналу використовуються методи статистичного аналізу, а також базові розрахунки, які дозволяють обробляти великі обсяги інформації та виявляти приховані залежності. Зокрема, на даному етапі розробки, застосовуються алгоритми для розрахунку дельти між введеними показниками, розрахунку вартості спожитої електроенергії. Також реалізовано алгоритми прогнозування, які дозволяють передбачити майбутнє споживання на основі наявної інформації, використовуючи можливості моделі Chronos (Зображення 2.5).

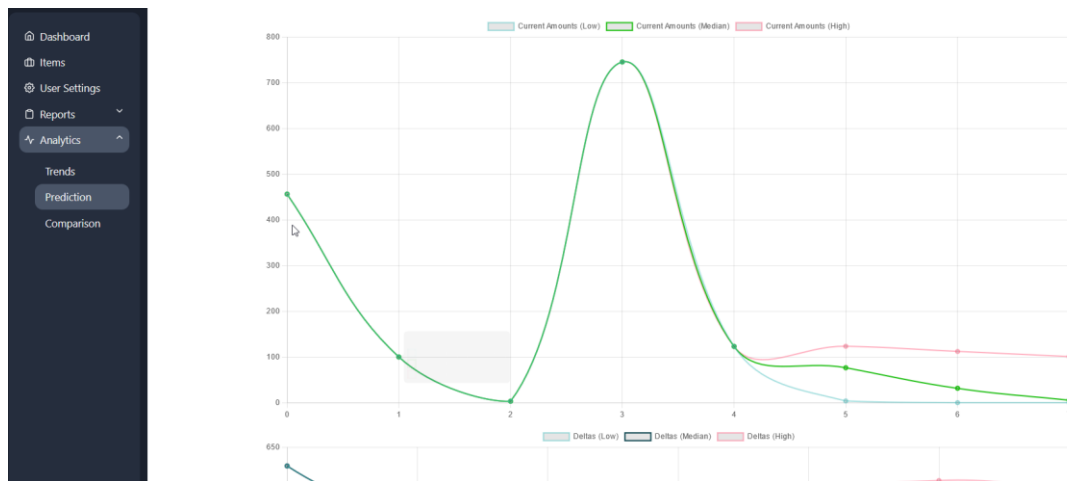


Рис 2.5. Приклад графіку прогнозування на основі моделі Chronos

На даний момент система аналізу враховує введені користувачем дані, такі як показники лічильника та тарифи. На основі цієї інформації система будує графіки, які відображають динаміку споживання, а також розраховує базові показники, такі як загальне споживання за період, середнє споживання. Присутні графіки зміни показників лічильника, тарифів, загальної суми, а також дельти між сусідніми показниками. Також реалізований графік трендів, який відображає зміну загальної суми та тарифу. Система надає користувачам зручний інструмент для аналізу власного споживання та прийняття рішень щодо його оптимізації.

Результати аналізу візуалізуються у вигляді зрозумілих та інформативних графіків, діаграм, які відображаються в особистому кабінеті користувача, що дозволяє швидко та зручно сприймати інформацію. Додаток надає можливість будувати графіки зміни показників лічильника, зміни тарифів, зміни загальної суми, зміни дельти між показниками, а також графік трендів. Користувач може обирати різні часові проміжки для аналізу, від одного дня до року, що дозволяє відстежувати як короткострокові, так і довгострокові тенденції. Також передбачена можливість порівнювати власне споживання з попередніми періодами, що допомагає оцінити ефективність вжитих заходів з енергозбереження. Наприклад, користувач може порівняти графіки споживання за поточний та минулий місяці, щоб побачити, чи

призвели його зусилля до зменшення витрат електроенергії. Система аналізу, реалізована у веб-застосунку, є важливим інструментом, що допомагає користувачам краще розуміти власне споживання електроенергії та приймати обґрунтовані рішення щодо його оптимізації. Завдяки зручній візуалізації даних та можливості детального аналізу, користувачі можуть виявляти основні фактори, що впливають на їх витрати, та розробляти ефективні стратегії економії.

Ефективний аналіз та оптимізація споживання електроенергії неможливі без ретельної та продуманої підготовки даних, що є критично важливим етапом всього процесу, від якого безпосередньо залежить точність та достовірність отриманих результатів аналізу, а також ефективність розроблених рекомендацій щодо оптимізації. Підготовка даних включає в себе декілька послідовних кроків, починаючи зі збору даних з різних джерел. На даному етапі розвитку веб-застосунку основним джерелом даних є інформація, що вводиться користувачем вручну через зручний та інтуїтивно зрозумілий інтерфейс.

Користувачі вносять дані про показники лічильників, дату та час зняття показників, а також інформацію про діючі тарифи, ідентифікатори лічильників та адрес. На етапі розробки, система передбачає базовий функціонал ручного вводу даних, який є основою для подальшого аналізу та візуалізації. Після введення, дані проходять етап первинної обробки та валідації, на якому система автоматично перевіряє їх на коректність, відповідність заданим форматам та допустимим значенням. Наприклад, система контролює, чи є введені показники лічильника числовими значеннями. У разі виявлення помилок або невідповідностей, користувачу миттєво виводяться відповідні повідомлення з підказками щодо їх виправлення, що допомагає уникнути некоректних даних та забезпечити цілісність інформації на етапі введення.

Після первинної обробки та валідації дані зберігаються у базі даних, де вони проходять подальшу підготовку до аналізу. На цьому етапі, наразі, дані не агрегуються, а зберігаються в первісному вигляді, що забезпечує гнучкість

при подальшому аналізі та дозволяє використовувати різні методи обробки в залежності від потреб. Підготовлені дані зберігаються у структурованому вигляді в реляційній базі даних PostgreSQL, що забезпечує швидкий та зручний доступ до них для аналітичних модулів та інших компонентів системи, а також гарантує цілісність та збереженість даних. Ретельна підготовка даних є запорукою отримання достовірних результатів аналізу та формування ефективних рекомендацій, що, в свою чергу, сприятиме досягненню цілей з енергозбереження та оптимізації витрат.

Розробка надійної та безпечної системи автентифікації та авторизації є критично важливим аспектом створення веб-застосунку, особливо коли мова йде про роботу з чутливими даними користувачів, такими як інформація про споживання електроенергії. Ця система відповідає за перевірку особи користувача, надання йому доступу до відповідних ресурсів та функцій застосунку, а також за захист його облікового запису та персональних даних від несанкціонованого доступу.

В рамках даного застосунку було приділено особливу увагу розробці механізмів автентифікації та авторизації, що відповідають сучасним стандартам безпеки та забезпечують зручність використання для користувачів. Першим кроком у розробці системи є реалізація функціоналу реєстрації нових користувачів, яка дозволяє їм створити власний обліковий запис у системі. Процес реєстрації включає в себе введення користувачем унікальної адреси електронної пошти, яка буде використовуватись як логін, та надійного пароля, який відповідає встановленим вимогам безпеки.

Для забезпечення високого рівня безпеки паролів, система застосовує сучасні методи хешування, що перетворюють пароль у незворотній набір символів, який неможливо відновити до початкового вигляду. Це гарантує, що навіть у разі витоку даних зломисники не зможуть отримати доступ до облікових записів користувачів, знаючи лише їхні хешовані паролі. Після успішної реєстрації на вказану користувачем електронну адресу надсилається лист з підтвердженням, що містить унікальне посилання. Перехід за цим

посиланням активує обліковий запис та завершує процес реєстрації. Такий підхід дозволяє переконатися, що користувач має доступ до вказаної електронної пошти, а також захищає від створення фейкових акаунтів. Додатково, система може перевіряти надійність введеного пароля, вимагаючи від користувача використання комбінації літер, цифр та спеціальних символів певної мінімальної довжини, що суттєво ускладнює процес підбору пароля злоумисниками.

Для автентифікації користувачів, тобто перевірки їхньої особи при вході в систему, використовується комбінація електронної пошти та пароля, які користувач вводить у спеціальній формі. Після введення даних система порівнює їх з інформацією, що зберігається в базі даних, і, у разі успішної перевірки, надає користувачеві доступ до його облікового запису та відповідних функцій веб-застосунку.

Для забезпечення зручності та безпеки, а також для уникнення необхідності повторної автентифікації при кожному запиті, застосовується механізм JWT-токенів (JSON Web Tokens). Після успішної автентифікації сервер генерує унікальний токен, який містить інформацію про користувача та його права доступу, а також цифровий підпис, що гарантує цілісність та автентичність токена. Цей токен передається клієнту та зберігається на його боці, у локальному сховищі браузера.

При кожному наступному запиті до сервера, що вимагає авторизації, клієнт автоматично надсилає JWT-токен разом із запитом у заголовку `Authorization`. Сервер перевіряє отриманий токен, верифікує його цифровий підпис та, у разі успішної перевірки, надає доступ до запитуваного ресурсу. Такий підхід дозволяє уникнути необхідності зберігати стан сесії на сервері, що підвищує масштабованість та відмовостійкість системи. Крім того, JWT-токени мають обмежений термін дії, після закінчення якого користувачеві необхідно буде знову пройти процедуру автентифікації, що є додатковим заходом безпеки. Важливо зазначити, що при зберіганні токенів на стороні клієнта необхідно дотримуватись заходів безпеки, таких як використання

HTTPS для захисту передачі даних та зберігання токенів у захищеному сховищі, наприклад, з атрибутом `HttpOnly` для `cookie`, що унеможливорює доступ до токена з JavaScript.

Для підвищення рівня безпеки та захисту від несанкціонованого доступу, у веб-застосунку передбачена можливість увімкнення двофакторної автентифікації (2FA). При використанні 2FA, окрім введення логіна та пароля, користувачеві необхідно додатково підтвердити свою особу за допомогою другого фактора, одноразового коду, згенерованого мобільним додатком. Це значно ускладнює процес злому облікового запису, оскільки зловмиснику необхідно буде отримати доступ не тільки до логіна та пароля користувача, але й до його мобільного пристрою. Користувач може самостійно увімкнути або вимкнути двофакторну автентифікацію у налаштуваннях свого профілю, що надає йому можливість вибору між підвищеним рівнем безпеки та зручністю використання.

Окрім основних функцій реєстрації та автентифікації, система також надає можливість відновлення пароля у разі його втрати користувачем. Для цього користувач може скористатися спеціальною формою, вказавши свою електронну адресу, на яку буде надіслано лист з інструкціями щодо відновлення пароля. Лист містить унікальне посилання з тимчасовим токеном, перехід за яким дозволяє користувачеві задати новий пароль для свого облікового запису. Цей токен має обмежений термін дії, що забезпечує додатковий захист від несанкціонованого відновлення пароля. Після успішної зміни пароля користувач може увійти в систему, використовуючи свої нові облікові дані.

Система авторизації у веб-застосунку передбачає розмежування прав доступу користувачів залежно від їхніх ролей. Наприклад, звичайні користувачі мають доступ лише до своїх даних та обмеженого функціоналу, тоді як адміністратори мають розширені права, що дозволяють їм керувати обліковими записами інших користувачів, переглядати загальну статистику та налаштовувати параметри системи. Такий підхід дозволяє забезпечити

конфіденційність даних та захистити систему від несанкціонованих дій з боку звичайних користувачів. Розподіл прав доступу реалізовано на рівні backend частини, де кожен запит до API перевіряється на наявність необхідних дозволів у користувача, що здійснив цей запит.

Розроблена система автентифікації та авторизації є надійним та безпечним рішенням, що відповідає сучасним вимогам та стандартам. Вона забезпечує захист персональних даних користувачів, запобігає несанкціонованому доступу до функціоналу веб-застосунку та надає зручні механізми для керування обліковими записами. Гнучкість системи дозволяє легко розширювати її функціонал у майбутньому, додаючи нові методи автентифікації, інтегруючись з іншими системами та адаптуючись до змінних вимог безпеки. Використання сучасних технологій, таких як JWT-токени та асиметричне шифрування, гарантує високий рівень захисту та зручність використання, що є важливими факторами для забезпечення довіри користувачів та успішного розвитку веб-застосунку.

2.3. Система повідомлень, автоматичних звітів та їх налаштування

Важливою складовою веб-застосунку є система повідомлень та нагадувань, що забезпечує своєчасне інформування користувачів про важливі події, зміни у їхньому енергоспоживанні, а також нагадує про необхідність виконання певних дій, таких як введення даних лічильника. Розробка цієї системи передбачає створення гнучкого та надійного механізму, який дозволяє користувачам отримувати персоналізовані повідомлення у зручному форматі, а адміністраторам системи - ефективно керувати процесом інформування.

Для реалізації системи повідомлень та нагадувань було обрано декілька каналів комунікації, включаючи електронну пошту та сповіщення безпосередньо у веб-застосунку, що дозволяє охопити різні категорії користувачів та забезпечити своєчасну доставку інформації. Користувачі

можуть самостійно обирати бажаний спосіб отримання повідомлень у налаштуваннях свого профілю, а також налаштовувати частоту та типи повідомлень, які вони хочуть отримувати.

Наприклад, користувач може увімкнути щомісячні нагадування про необхідність введення показників лічильника, а також сповіщення про значні зміни у структурі його енергоспоживання або про досягнення поставлених цілей з енергозбереження. Система також надає можливість відмовитися від отримання певних типів повідомлень, що дозволяє користувачам контролювати інформаційний потік та отримувати лише ту інформацію, яка є для них важливою та корисною.

Розробка системи передбачає створення зручного інтерфейсу для керування повідомленнями та нагадуваннями як для користувачів, так і для адміністраторів системи. Користувачі можуть переглядати історію отриманих повідомлень, змінювати свої налаштування, а також позначати повідомлення як прочитані або видаляти їх. Адміністратори, в свою чергу, мають можливість створювати та редагувати шаблони повідомлень, керувати розсилками, а також відстежувати статистику доставки та прочитання повідомлень.

Це дозволяє адміністраторам ефективно комунікувати з користувачами, інформувати їх про важливі події, а також аналізувати ефективність різних типів повідомлень та оптимізувати процес інформування. Важливим аспектом є забезпечення надійності та своєчасності доставки повідомлень. Для цього система використовує чергу повідомлень та асинхронну обробку, що дозволяє відправляти велику кількість повідомлень без затримок та з мінімальним навантаженням на сервер. У випадку виникнення помилок при доставці, система автоматично повторює спробу відправлення через певний проміжок часу.

Для забезпечення високої доступності та відмовостійкості системи використовуються сучасні технології та архітектурні рішення, такі як кластеризація, балансування навантаження та резервне копіювання даних. Повідомлення буде надіслано із мінімальною затримкою встановленою для

циклу обробки повідомлень. Цей цикл завжди перевіряє чи потрібно надіслати нове повідомлення для користувача. Фронтенд частина реалізує складний механізм доступу компонентів до повідомлень користувача а також дає змогу різним компонентам прив'язуватись до них щоб отримати якесь необхідне повідомлення.

Окремою, важливою функцією є розроблена система автоматичної генерації звітів, яка дозволяє користувачам регулярно отримувати узагальнену та структуровану інформацію про власне споживання електроенергії у зручному для аналізу форматі. Одним із ключових аспектів є забезпечення гнучкості системи, що дозволяє користувачам налаштовувати параметри звітів відповідно до своїх потреб та вподобань.

Зокрема, користувачі мають можливість самостійно обирати типи даних, які будуть включені до звіту, такі як загальне споживання за обраний період, середньодобове споживання, витрати на електроенергію, а також динаміку зміни цих показників. Окрім того, передбачена опція вибору бажаного формату файлу звіту, наприклад, PDF або CSV, що надає користувачам можливість подальшої обробки даних у зручному для них програмному забезпеченні.

Важливим елементом системи є можливість налаштування розкладу автоматичної генерації звітів. Користувачі можуть вказати бажану періодичність отримання звітів, наприклад, щомісяця, щокварталу або щороку, а також обрати конкретну дату та час формування звіту. Це дозволяє автоматизувати процес отримання інформації та позбавляє користувачів необхідності вручну генерувати звіти кожного разу.

Система автоматично створює звіт у вказаний час, враховуючи обрані користувачем параметри, та надсилає його на електронну пошту, вказану в профілі користувача. Такий підхід забезпечує регулярне інформування користувачів про стан їхнього енергоспоживання та сприяє формуванню звички аналізувати ці дані для прийняття обґрунтованих рішень щодо оптимізації.

Для забезпечення надійності та безперебійності роботи системи автоматичної генерації звітів було розроблено спеціальний модуль, який відповідає за фонове виконання завдань зі створення та відправки звітів. Цей модуль функціонує незалежно від основної частини веб-застосунку, що дозволяє уникнути впливу на його продуктивність та забезпечити своєчасне виконання завдань навіть у періоди пікового навантаження. Модуль взаємодіє з базою даних для отримання необхідної інформації про налаштування звітів для кожного користувача, а також з модулем генерації звітів, передаючи йому відповідні параметри. Після успішного створення звіту, модуль надсилає його на вказану електронну пошту користувача, використовуючи налаштований SMTP-сервер.

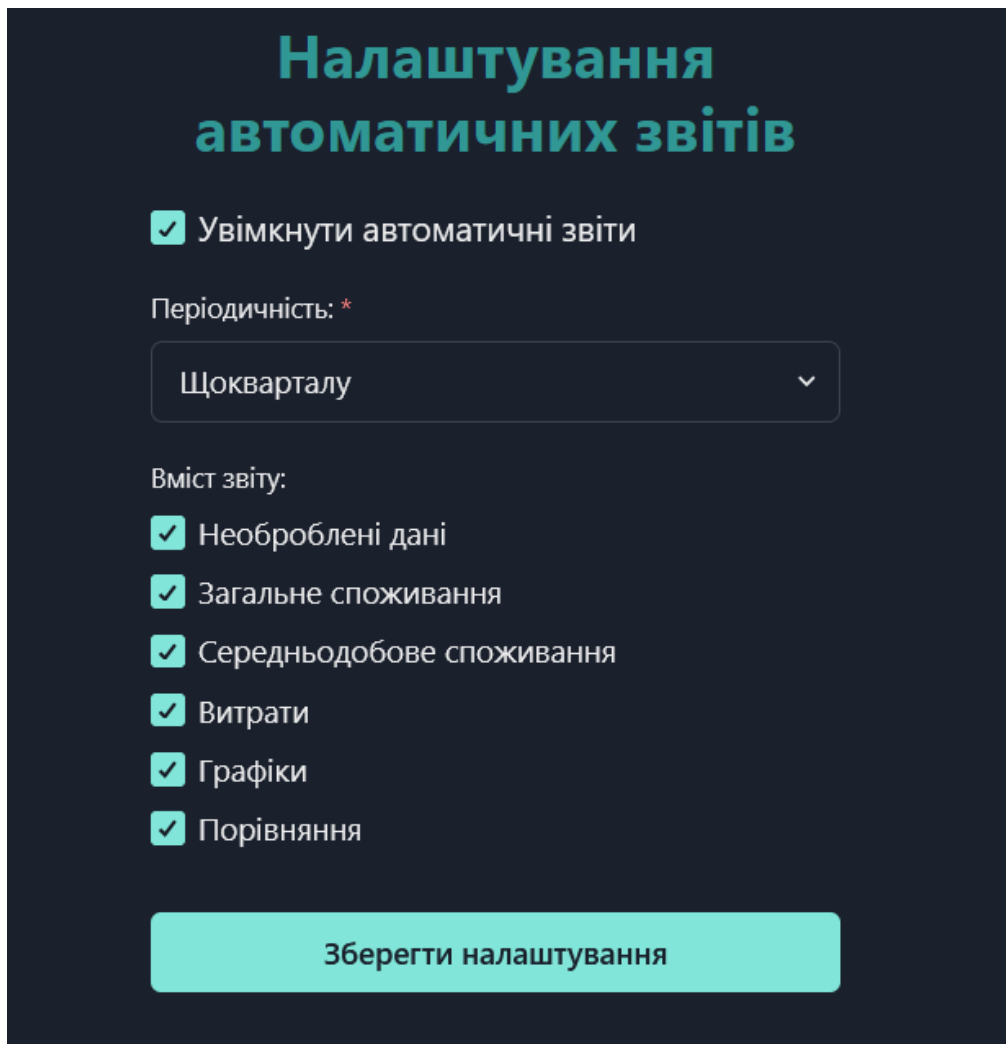
Окрім основного функціоналу, система автоматичної генерації звітів передбачає можливість розширення та адаптації до потреб користувачів у майбутньому. Наприклад, планується додати опцію включення до звіту порівняльного аналізу з середніми показниками по регіону або з аналогічними домогосподарствами, що дозволить користувачам краще оцінити ефективність власного енергоспоживання.

Також розглядається можливість інтеграції з зовнішніми сервісами для отримання додаткової інформації, такої як прогнози погоди, що може бути корисною для аналізу впливу кліматичних умов на споживання електроенергії. В перспективі планується додати можливість генерувати звіти не тільки у форматі PDF, але й у вигляді інтерактивних дашбордів, що дозволить користувачам більш детально аналізувати дані та виявляти цікаві закономірності.

Для зручного налаштування параметрів автоматичної генерації та доставки звітів було розроблено спеціальний інтерфейс користувача. Цей інтерфейс є важливою складовою веб-застосунку, що забезпечує користувачам зручний та інтуїтивно зрозумілий спосіб керування функціоналом автоматичних звітів. При розробці інтерфейсу особливу увагу було приділено простоті використання, наочності та доступності налаштувань,

щоб користувачі з будь-яким рівнем технічної підготовки могли легко скористатися цією функцією. Інтерфейс виконано у єдиному стилі з іншими елементами веб-застосунку, що забезпечує цілісний та гармонійний користувацький досвід.

Інтерфейс налаштування автоматичних звітів (Зображення 2.6) реалізовано у вигляді окремого розділу в особистому кабінеті користувача, що забезпечує легкий доступ до нього та логічно відокремлює його від інших функцій. При переході до цього розділу користувач бачить форму з декількома полями, які дозволяють йому вказати бажані параметри генерації звітів. Насамперед, користувач може обрати періодичність отримання звітів, обравши один із запропонованих варіантів: щомісяця, щокварталу або щороку.



**Налаштування
автоматичних звітів**

☒ Увімкнути автоматичні звіти

Періодичність: *

Щокварталу

Вміст звіту:

- ☒ Необроблені дані
- ☒ Загальне споживання
- ☒ Середньодобове споживання
- ☒ Витрати
- ☒ Графіки
- ☒ Порівняння

Зберегти налаштування

Рис. 2.6. Інтерфейс налаштування автоматичних звітів

Окрім налаштувань періодичності, інтерфейс надає користувачам можливість кастомізації вмісту звіту. Користувач може обрати, які саме дані будуть включені до звіту, наприклад, загальне споживання за період, середньодобове споживання, витрати на електроенергію, графіки зміни показників, порівняння з попередніми періодами. Також передбачена можливість вибору формату файлу звіту, наразі підтримується тільки PDF тому інтерфейс вибору приховано. Для забезпечення високого рівня персоналізації, користувачі можуть вказати додаткові параметри, такі як назва звіту.

Важливим елементом інтерфейсу є система валідації введених даних, яка перевіряє коректність та повноту наданої інформації. Наприклад, система перевіряє, чи правильно вказано адресу електронної пошти, чи обрано коректний період та чи не суперечать одне одному вибрані параметри. У разі виявлення помилок, користувачу виводяться відповідні повідомлення з підказками щодо їх виправлення, що дозволяє уникнути некоректного налаштування системи та забезпечити безперебійну роботу функціоналу автоматичних звітів. Після внесення змін та успішної валідації даних, користувач може зберегти налаштування, які будуть автоматично застосовані системою.

Розроблена система автоматичної генерації звітів є потужним інструментом, що значно підвищує цінність веб-застосунку для користувачів. Вона дозволяє автоматизувати процес отримання та аналізу інформації про споживання електроенергії, заощаджуючи час та зусилля користувачів. Гнучкість налаштувань, можливість вибору періоду, формату та вмісту звіту, а також автоматична доставка на електронну пошту роблять цю систему зручною та ефективною. Завдяки автоматичним звітам користувачі можуть регулярно відстежувати свій прогрес у сфері енергозбереження, вчасно виявляти проблеми та приймати обґрунтовані рішення щодо оптимізації свого енергоспоживання, що сприятиме зменшенню витрат та зниженню негативного впливу на навколишнє середовище.

Розроблений інтерфейс для налаштування автоматичних звітів є зручним та інтуїтивно зрозумілим інструментом, який дозволяє користувачам легко керувати процесом отримання інформації про власне енергоспоживання. Простота використання, гнучкість налаштувань та візуальна привабливість інтерфейсу роблять його доступним для широкого кола користувачів, незалежно від їхнього рівня технічної підготовки. Завдяки цьому рішення користувачі можуть автоматизувати рутинні операції з аналізу даних, регулярно отримувати узагальнену інформацію у зручному форматі та зосередитися на прийнятті рішень щодо оптимізації свого енергоспоживання, що сприятиме як економії коштів, так і зменшенню негативного впливу на навколишнє середовище.

Розроблена система повідомлень та нагадувань є гнучким та ефективним інструментом для взаємодії з користувачами, що дозволяє не тільки інформувати їх про важливі події, але й стимулювати до активної участі у процесі моніторингу та оптимізації енергоспоживання. Завдяки якісним налаштуванням, користувачі можуть отримувати лише ту інформацію, яка є для них дійсно важливою та корисною. Це сприяє підвищенню рівня залученості користувачів, формуванню у них свідомого ставлення до споживання ресурсів та, як наслідок, досягненню цілей з енергозбереження та зниження негативного впливу на навколишнє середовище.

2.4. Архітектура веб-застосунку: база даних, Frontend та API

Розробка ефективного та надійного веб-застосунку для моніторингу та оптимізації споживання електроенергії вимагає ретельного проектування та реалізації його архітектури, що включає як базу даних, так і клієнтську частину (Frontend), а також їх взаємодію через API. Проектування структури бази даних є одним з найважливіших етапів, оскільки від неї залежить ефективність зберігання та обробки даних, а також можливості масштабування та підтримки системи в майбутньому. У даному проекті було

обрано реляційну модель даних, як таку, що найкраще відповідає характеру даних та потребам застосунку. Для керування базою даних було обрано СУБД PostgreSQL, яка відома своєю надійністю, продуктивністю та підтримкою розширених можливостей, включаючи роботу з JSON даними та наявність спеціалізованих розширень. При проектуванні структури бази даних особливу увагу було приділено нормалізації, що дозволяє уникнути дублювання даних, забезпечити їх цілісність та спростити виконання запитів. Також було враховано вимоги до швидкодії та масштабованості, що зумовило вибір відповідних типів даних та створення індексів для оптимізації пошуку. Основними сутностями, що були визначені на етапі аналізу предметної області, є "Користувач", "Показник лічильника" ("Item"), "Тариф", "Звіт", "Прогноз" ("Forecast"), "Налаштування звітів" та інші допоміжні сутності. Кожна з цих сутностей представлена окремою таблицею в базі даних, яка містить відповідні атрибути (поля).

Таблиця "Користувач" є однією з центральних у системі та містить інформацію про зареєстрованих користувачів веб-застосунку. Кожен запис у цій таблиці відповідає окремому користувачеві та має унікальний ідентифікатор, що представлений у вигляді UUID (Universally Unique Identifier). Такий підхід забезпечує унікальність ідентифікаторів у глобальному масштабі та дозволяє уникнути колізій при розподіленій роботі з даними.

Окрім ідентифікатора, таблиця "Користувач" містить такі атрибути, як повне ім'я користувача, що зберігається у вигляді рядка тексту, адресу електронної пошти, яка використовується для автентифікації та зв'язку з користувачем, а також хешований пароль, що забезпечує безпечне зберігання облікових даних. Для зберігання паролів використовується сучасний алгоритм хешування, що гарантує незворотність перетворення та унеможливорює відновлення оригінального пароля з хешу. Також у таблиці присутні поля, що вказують на статус користувача (активний чи неактивний) та його роль у

системі (адміністратор чи звичайний користувач), що дозволяє реалізувати гнучку систему розмежування доступу до функцій веб-застосунку.

Таблиця "Показник лічильника", або "Item", є однією з найважливіших у системі, оскільки саме в ній зберігаються дані про споживання електроенергії, введені користувачами. Кожен запис у цій таблиці відповідає одному показнику лічильника, знятому в певний момент часу, та містить ряд атрибутів, що детально описують цей показник. Поле "id" є унікальним ідентифікатором запису у вигляді UUID. Поле "title" типу VARCHAR(255) представляє собою назву показника, а "description" типу VARCHAR(255) - його опис, які можуть бути заповнені користувачем для зручності ідентифікації.

Зв'язок з таблицею "Користувач" здійснюється через зовнішній ключ "owner_id", що вказує на ідентифікатор користувача, який ввів даний показник. Це дозволяє визначити, якому саме користувачеві належить кожен запис у таблиці "Item". Реалізовано каскадне видалення, тобто при видаленні користувача автоматично видаляються всі його записи з таблиці "Item". Окрім зв'язку з користувачем, таблиця "Item" містить безпосередньо дані про споживання електроенергії та іншу важливу інформацію. Поле "meter_identifier" (VARCHAR) ідентифікує адресу споживача або конкретний лічильник, для якого було знято показник.

Поле "reading_value" (INTEGER) містить безпосередньо чисельне значення показника лічильника на момент його фіксації. Поле "kwh_tariff" (FLOAT) зберігає значення тарифу на електроенергію, що діяв на момент зняття показника, а "current_amount" (FLOAT) відображає загальну суму до сплати за електроенергію, розраховану на основі введених даних. Поле "reading_date" (TIMESTAMP) фіксує точну дату та час зняття показника, що дозволяє відстежувати динаміку споживання та будувати графіки за різні часові проміжки. Всі ці поля є обов'язковими для заповнення, що забезпечує повноту та цілісність даних.

Таблиця "Прогноз", або "Forecast", призначена для зберігання результатів прогнозування споживання електроенергії, згенерованих моделлю Chronos. Кожен запис у цій таблиці містить інформацію про один прогноз, зроблений для конкретного користувача та певного графіка. Поле "id" є унікальним ідентифікатором запису у вигляді UUID. Поле "name_nn_model" (VARCHAR(100)) містить назву моделі машинного навчання, що була використана для генерації прогнозу, в даному випадку "chronos".

Поле "chart_name" (VARCHAR(100)) ідентифікує графік, для якого було зроблено прогноз, наприклад, "reading_values", "kwh_tariffs", "current_amounts", "deltas", "costs". Поле "user_id" є зовнішнім ключем, що посиляється на таблицю "Користувач" та вказує, для якого користувача було зроблено прогноз. Особливу увагу варто звернути на поля, що зберігають безпосередньо дані прогнозу. "prediction_low", "prediction_median" та "prediction_high" (JSON) містять, відповідно, нижню межу, медіану та верхню межу довірчого інтервалу прогнозу.

Ці дані представлені у форматі JSON, що дозволяє гнучко зберігати прогнози різної структури та розмірності. Поля "prediction_time_steps" та "prediction_forecast_steps" (JSON) містять часові мітки для точок, що відповідають історичним даним, та для точок, що відповідають прогнозованим значенням. Це дозволяє правильно відображати прогнози на графіках та співставляти їх з реальними даними. Поле "created_at" (TIMESTAMP) фіксує дату та час створення запису про прогноз, що дозволяє відстежувати актуальність прогнозів та, за необхідності, оновлювати їх.

Взаємозв'язок між таблицями "Item", "Forecast" та "User" дозволяє будувати складні запити для аналізу даних та генерації звітів. Наприклад, можна отримати всі прогнози, зроблені для конкретного користувача, або всі показники лічильника, введені певним користувачем за певний період часу. Також можна поєднувати дані з різних таблиць для отримання більш повної картини, наприклад, порівнювати фактичне споживання з прогнозованим або аналізувати вплив зміни тарифів на загальну суму до сплати. Наявність поля

"created_at" у таблиці "Forecast" дозволяє відстежувати історію прогнозування та оцінювати точність моделі Chronos, порівнюючи прогнози з реальними даними, що були отримані пізніше.

Для забезпечення швидкого доступу до даних та оптимізації продуктивності запитів було створено індекси для деяких полів таблиць. Зокрема, створено індекс для поля "owner_id" у таблиці "Item", що дозволяє швидко знаходити всі показники лічильника, введені конкретним користувачем. Також створено індекси для полів "user_id", "name_nn_model" та "chart_name" у таблиці "Forecast", що прискорює пошук прогнозів за відповідними критеріями. Використання індексів є особливо важливим при роботі з великими обсягами даних, оскільки дозволяє значно скоротити час виконання запитів та зменшити навантаження на базу даних.

Розроблена структура бази даних є гнучкою та масштабованою, що дозволяє розширювати її в майбутньому, додаючи нові сутності та атрибути. Наприклад, у разі потреби, можна додати таблиці для зберігання інформації про різні типи побутових приладів та їх енергоспоживання, що дозволить реалізувати функціонал з надання більш детальних рекомендацій щодо оптимізації. Також можна додати таблицю для зберігання історії зміни тарифів на електроенергію, що дозволить більш точно розраховувати вартість спожитої електроенергії за різні періоди часу.

Завдяки використанню UUID як первинних ключів, забезпечується унікальність ідентифікаторів навіть при розподіленій роботі з даними, що є важливим для забезпечення цілісності та несуперечності інформації. Окрім основних таблиць, що зберігають дані про користувачів, показники лічильників та прогнози, в базі даних також присутня таблиця "UserReportSettings", яка призначена для збереження налаштувань автоматичної генерації звітів для кожного користувача. Ця таблиця містить унікальний ідентифікатор запису "id" (UUID), зовнішній ключ "user_id", що посилається на таблицю "User", а також поля, що визначають параметри звітів. Поле "report_enabled" (BOOLEAN) вказує, чи увімкнено автоматичну

генерацію звітів для даного користувача. Поле "report_period" (VARCHAR) визначає періодичність створення звітів, наприклад, "monthly" або "quarterly". Поле "last_reported_at" (TIMESTAMP) містить дату та час останньої генерації звіту для даного користувача, що дозволяє системі визначати, коли необхідно створювати наступний звіт а також інші поля.

Така структура таблиці "UserReportSettings" дозволяє гнучко налаштовувати параметри автоматичних звітів для кожного користувача окремо. Наприклад, один користувач може увімкнути щомісячні звіти, а інший - щоквартальні. Також, користувачі можуть тимчасово вимкнути автоматичну генерацію звітів, змінивши значення поля "report_enabled" на "false". Завдяки наявності поля "last_reported_at" система може легко визначати, для яких користувачів настав час генерувати наступний звіт, та уникати повторного створення звітів за один і той самий період.

Це забезпечує ефективне використання ресурсів та дозволяє уникнути надмірного навантаження на систему. Варто зазначити, що при розробці структури бази даних було приділено увагу питанням безпеки та захисту персональних даних користувачів. Зокрема, паролі користувачів зберігаються у зашифрованому вигляді з використанням надійних алгоритмів хешування, що унеможливорює їх відновлення у разі несанкціонованого доступу до бази даних. Також, доступ до чутливої інформації, такої як показники лічильників та прогнози споживання, обмежено для звичайних користувачів і надається лише адміністраторам системи та самому користувачу, якому належать ці дані. Для забезпечення цілісності та доступності даних використовуються механізми резервного копіювання та реплікації, що дозволяє відновити роботу системи у разі збоїв чи аварійних ситуацій.

Архітектура frontend частини веб-застосунку, розробленого з використанням бібліотеки React, базується на компонентному підході. Цей підхід передбачає розбиття інтерфейсу користувача на окремі незалежні компоненти, кожен з яких відповідає за певну частину інтерфейсу та має власну логіку та стан. Такий підхід забезпечує модульність, перевикористання

коду, а також спрощує розробку, тестування та підтримку застосунку. Компоненти React можуть бути як функціональними, так і класовими, проте в даному застосунку перевага надається функціональним компонентам у поєднанні з хуками, що дозволяє створювати більш лаконічний та зрозумілий код. Одним із ключових аспектів розробки React-застосунків є управління станом.

Стан (state) - це дані, які можуть змінюватися з часом і впливають на те, що відображається в інтерфейсі користувача. У React кожен компонент може мати власний локальний стан, який керується за допомогою хуків `useState` та `useReducer`. Хук `useState` є найпростішим способом керування станом і підходить для випадків, коли стан складається з одного або декількох простих значень. Наприклад, `useState` можна використовувати для зберігання значення, введеного користувачем у текстове поле, або стану перемикача (ввімкнено/вимкнено).

Для керування більш складним станом, що включає об'єкти з багатьма полями або складну логіку оновлення, використовується хук `useReducer`. Цей хук приймає функцію-редюсер та початковий стан, а повертає поточний стан та функцію `dispatch`, яка використовується для відправки дій (actions), що змінюють стан. Редюсер - це чиста функція, яка приймає поточний стан та дію, і повертає новий стан на основі цієї дії. Такий підхід дозволяє чітко визначити, як саме може змінюватися стан компонента, та спрощує тестування логіки оновлення стану.

Однак, у великих застосунках часто виникає необхідність передачі даних між компонентами, що знаходяться на різних рівнях ієрархії. Передача даних через пропси (props drilling) може призвести до ускладнення коду та зниження його читабельності. Для вирішення цієї проблеми використовується механізм `React Context`. Контекст дозволяє створити спільне сховище даних, до якого можуть звертатися будь-які компоненти, що знаходяться всередині провайдера контексту. Це особливо зручно для даних, які є глобальними для всього застосунку або для великої його частини. Наприклад, у нашому веб-

застосунку контекст використовується для зберігання інформації про автентифікованого користувача, включаючи його ім'я, роль та налаштування.

Завдяки контексту, будь-який компонент, якому потрібні дані користувача, може отримати до них доступ, не вимагаючи явної передачі цих даних через пропси від батьківських компонентів. Це значно спрощує структуру компонентів та робить код більш зрозумілим. Оновлення даних у контексті призводить до автоматичного перерендерингу всіх компонентів, які ці дані використовують, що забезпечує консистентність інтерфейсу користувача. Варто зазначити, що контекст найкраще підходить для даних, які змінюються відносно рідко, оскільки часті оновлення контексту можуть призвести до зайвих перерендерів компонентів.

Для керування більш складним глобальним станом застосунку, який часто оновлюється та потребує більш складних операцій, використовується бібліотека Zustand. Zustand дозволяє створити централізований стор даних, до якого можуть звертатися будь-які компоненти застосунку. На відміну від контексту, Zustand оптимізований для роботи зі станом, що часто змінюється, і забезпечує високу продуктивність завдяки використанню селекторів та мемоізації. У глобальному сторі, керованому Zustand, зберігаються дані, що є критично важливими для роботи всього застосунку. Наприклад, токен автентифікації користувача, який використовується для доступу до захищених ресурсів API, а також кешовані дані, отримані з сервера.

Взаємодія між React-компонентами, контекстом та глобальним стором Zustand є чітко організованою та структурованою. Компоненти можуть підписуватися на зміни окремих частин глобального стану за допомогою селекторів, і реагувати на ці зміни, оновлюючи свій власний стан або виконуючи певні дії. Наприклад, компонент, що відображає ім'я користувача, може підписатися на зміни поля `user.name` в глобальному сторі, і автоматично оновлюватися при зміні імені. Zustand також надає зручні механізми для виконання асинхронних операцій та оновлення стану на основі результатів цих операцій.

Оновлення даних у глобальному сторі Zustand відбувається за допомогою функцій-мутацій, які визначені в самому сторі. Це дозволяє інкапсулювати логіку зміни стану в одному місці та забезпечити консистентність даних. Компоненти не мають прямого доступу до даних у сторі, а лише викликають відповідні мутації, передаючи їм необхідні параметри. Такий підхід спрощує тестування та налагодження коду, оскільки вся логіка зміни стану зосереджена в одному місці. Крім того, Zustand автоматично оптимізує перерендеринг компонентів, гарантуючи, що оновлюються лише ті компоненти, які дійсно залежать від змінених даних.

Розроблена архітектура frontend частини також передбачає чіткий поділ відповідальності між компонентами та сервісами. Компоненти, що відповідають за відображення даних, не мають прямого доступу до API та не займаються бізнес-логікою. Вони отримують дані через пропси від батьківських компонентів-контейнерів, які, в свою чергу, взаємодіють із сервісами, що інкапсулюють логіку взаємодії з backend частиною. Сервіси відповідають за виконання HTTP-запитів до API, обробку відповідей, перетворення даних у формат, зручний для використання в компонентах, а також за обробку помилок.

Такий підхід забезпечує слабе зв'язування між компонентами, що полегшує їхнє тестування, рефакторинг та повторне використання. Для виконання асинхронних запитів до API та керування станом цих запитів використовується бібліотека React Query. Вона надає зручні хуки, такі як `useQuery` та `useMutation`, які дозволяють легко виконувати запити, кешувати отримані дані, автоматично оновлювати дані у фоновому режимі та обробляти помилки. React Query автоматично керує станами запиту (завантаження, успіх, помилка), що дозволяє розробнику зосередитися на логіці роботи з даними, а не на низькорівневих деталях реалізації асинхронних запитів. Крім того, React Query забезпечує оптимальне оновлення інтерфейсу, коли зміни в UI відображаються миттєво, ще до отримання відповіді від сервера, що покращує сприйняття користувача та створює відчуття швидкодії застосунку.

Використання TypeScript у frontend частині веб-застосунку додає ще один рівень контролю та надійності. Статична типізація дозволяє виявляти багато помилок ще на етапі розробки, а не під час виконання коду. Це особливо важливо при роботі зі складними структурами даних, такими як об'єкти, що повертаються з API. TypeScript допомагає уникнути помилок, пов'язаних з неправильним використанням типів даних, а також полегшує розуміння коду та його рефакторинг. Завдяки чіткому визначенню типів даних, розробникам легше орієнтуватися в коді та розуміти, як саме працюють ті чи інші компоненти та функції.

Загалом, розроблена архітектура frontend частини є гнучкою, масштабованою та зручною для розробки та підтримки. Вона базується на сучасних підходах та інструментах, що дозволяє створювати складні та інтерактивні інтерфейси користувача, забезпечуючи при цьому високу продуктивність та надійність роботи застосунку. Чіткий поділ відповідальності між компонентами, використання контексту та глобального стору, а також застосування хуків та сервісів дозволяють створити ефективну та добре структуровану систему, яка забезпечує зручну роботу з даними та швидкий відгук інтерфейсу на дії користувача.

Для забезпечення ефективної взаємодії між frontend та backend частинами веб-застосунку було розроблено RESTful API за допомогою фреймворку FastAPI. FastAPI було обрано завдяки його високій продуктивності, простоті використання, підтримці асинхронних запитів та автоматичній генерації інтерактивної документації. В основі FastAPI лежить бібліотека Starlette, яка забезпечує обробку HTTP-запитів та відповідей, а також бібліотека Pydantic, яка використовується для валідації даних та визначення моделей даних. Розроблений API надає набір кінцевих точок (endpoints), що дозволяють клієнтській частині взаємодіяти з backend частиною, виконувати CRUD операції над даними, а також запускати процеси прогнозування та генерації звітів. Кожна кінцева точка відповідає певному ресурсу в системі, такому як користувач, показник лічильника, прогноз, звіт

тощо, та підтримує відповідні HTTP-методи (GET, POST, PUT, DELETE) для виконання операцій над цими ресурсами.

Для забезпечення безпеки API було реалізовано механізм автентифікації та авторизації на основі JWT-токенів (JSON Web Tokens). При успішній автентифікації користувача сервер генерує JWT-токен, який містить інформацію про користувача та його права доступу, а також цифровий підпис, що гарантує цілісність та автентичність токена. Цей токен передається клієнту та зберігається на його боці. При кожному наступному запиті до захищених ресурсів API клієнт надсилає токен у заголовок Authorization, а сервер перевіряє його валідність та, у разі успішної перевірки, надає доступ до запитуваного ресурсу. Такий підхід дозволяє забезпечити безпечний доступ до API та захистити дані користувачів від несанкціонованого доступу. Крім того, використання JWT-токенів дозволяє уникнути необхідності зберігання стану сесії на сервері, що спрощує масштабування застосунку.

Однією з ключових переваг FastAPI є автоматична генерація документації API у форматі OpenAPI (Swagger). На основі анотацій типів, рядків документації та визначених моделей даних Pydantic, FastAPI автоматично створює інтерактивну документацію, доступну за URL-адресами /docs та /redoc. Ця документація містить детальний опис всіх кінцевих точок API, включаючи опис параметрів запитів, форматів відповідей, кодів помилок, а також приклади запитів та відповідей. Це значно полегшує розробку та тестування як backend, так і frontend частини, а також спрощує інтеграцію з іншими системами. Розробники frontend частини можуть використовувати цю документацію для розуміння логіки роботи API та правильного формування запитів до сервера.

Для забезпечення зручної та ефективної взаємодії frontend частини з REST API було використано інструмент open-ts (що є скороченням від OpenAPI TypeScript), який дозволяє автоматично генерувати TypeScript клієнт на основі OpenAPI специфікації. Цей інструмент приймає на вхід JSON-файл зі специфікацією OpenAPI, згенерованою FastAPI, та створює набір TypeScript

інтерфейсів та класів, що відповідають моделям даних та кінцевим точкам API. Згенерований клієнт містить функції для виконання HTTP-запитів до кожної кінцевої точки API, автоматично оброблюючи серіалізацію та десеріалізацію даних, а також перевірку типів. Використання open-ts дозволяє уникнути ручного написання великої кількості коду для взаємодії з API, що значно прискорює розробку та зменшує ймовірність помилок.

Інтеграція згенерованого TypeScript клієнта у frontend частину, розроблену на React, здійснюється шляхом імпорту відповідних модулів та використання наданих функцій у компонентах React. Наприклад, для отримання списку показників лічильника, компонент може викликати функцію `getItems` з відповідного сервісу, згенерованого open-ts, передавши їй необхідні параметри, такі як ідентифікатор користувача та діапазон дат. React Query в свою чергу забезпечить виконання запиту, кешування результатів та оновлення стану компонента при отриманні відповіді від сервера. Завдяки статичній типізації, забезпеченій TypeScript, розробники отримують підказки та перевірку типів під час написання коду, що допомагає уникнути помилок та покращує якість коду.

Використання open-ts для генерації TypeScript клієнта має ряд суттєвих переваг. По-перше, це забезпечує типобезпеку при взаємодії з API, оскільки типи даних, що передаються між клієнтом та сервером, чітко визначені та перевіряються на етапі компіляції. Це дозволяє виявляти багато помилок ще до запуску застосунку, що підвищує його надійність та стабільність роботи. По-друге, автоматична генерація клієнта економить час розробників та зменшує кількість рутинного коду, який необхідно писати вручну. Зміни в API автоматично відображаються у згенерованому клієнті після повторної генерації, що спрощує процес підтримки та оновлення коду. По-третє, згенерований клієнт містить детальну документацію, що відповідає документації OpenAPI, що полегшує його використання та інтеграцію в проект.

Розроблений REST API, разом із згенерованим TypeScript клієнтом, утворюють єдину екосистему, що забезпечує ефективну та безшовну взаємодію між frontend та backend частинами веб-застосунку. Такий підхід дозволяє чітко розділити відповідальність між розробниками frontend та backend, а також спрощує тестування та налагодження окремих компонентів системи. Frontend розробники можуть зосередитися на створенні зручного та привабливого інтерфейсу користувача, використовуючи готові функції для взаємодії з API, а backend розробники, в свою чергу, можуть зосередитися на реалізації бізнес-логіки, обробці даних та забезпеченні безпеки та продуктивності сервера.

Варто зазначити, що використання OpenAPI специфікації та автоматичної генерації коду не обмежується лише створенням TypeScript клієнта. На основі цієї специфікації можна також генерувати клієнтські бібліотеки для інших мов програмування, таких як Java, Kotlin, Python, що дозволяє легко інтегрувати розроблений API з іншими системами та сервісами. Це робить веб-застосунок більш гнучким та розширюваним, а також сприяє його інтеграції в існуючу IT-інфраструктуру. Крім того, наявність формалізованої специфікації API дозволяє автоматизувати процес тестування серверної частини, створюючи автоматичні тести, що перевіряють відповідність реалізованого API заявленій специфікації.

Таким чином, розроблений REST API на базі FastAPI та згенерований TypeScript клієнт утворюють єдиний, добре інтегрований стек технологій, що забезпечує ефективну розробку, тестування та підтримку веб-застосунку. Використання сучасних інструментів та підходів, таких як OpenAPI, open-ts, TypeScript та React, дозволяє створити надійне, масштабоване та зручне у використанні рішення, яке відповідає високим стандартам якості та безпеки. Цей підхід також закладає міцний фундамент для подальшого розвитку та розширення функціоналу веб-застосунку відповідно до потреб користувачів та змін у предметній області.

Висновки до розділу 2

У другому розділі було детально розглянуто процес розробки та проектування веб-застосунку, призначеного для моніторингу та оптимізації споживання електроенергії в побуті. Значну увагу було приділено створенню як серверної, так і клієнтської частин застосунку, з використанням сучасних технологій та інструментів, таких як FastAPI, React, TypeScript та інших. Ретельний вибір стеку технологій дозволив забезпечити високу продуктивність, масштабованість та зручність розробки, а також заклав надійний фундамент для подальшого розвитку та розширення функціоналу. Важливим аспектом розробки стала реалізація інтуїтивно зрозумілого та зручного інтерфейсу користувача, який дозволяє легко вводити дані про споживання, переглядати історію показників, отримувати прогнози та налаштовувати параметри звітів. Застосування компонентного підходу в React у поєднанні з бібліотекою Chakra UI забезпечило створення адаптивного та привабливого дизайну, що коректно відображається на різних типах пристроїв.

Одним із ключових компонентів розробленого веб-застосунку є система аналізу даних, яка надає користувачам цінні інсайти щодо їхнього енергоспоживання. Ця система здійснює первинну обробку та валідацію введених даних, зберігає їх у структурованому вигляді в базі даних PostgreSQL та надає потужні інструменти для візуалізації у вигляді різноманітних графіків. Користувачі можуть будувати графіки зміни показників лічильника, тарифів, загальної суми до сплати, а також аналізувати дельти між сусідніми показниками та відстежувати тренди. Наочне представлення даних у вигляді лінійних графіків та гістограм значно полегшує сприйняття інформації та дозволяє користувачам швидко виявляти тенденції, аномалії та закономірності у своєму споживанні. Важливою деталлю є те що графіки є інтерактивними, користувачі можуть з ними взаємодіяти, змінювати масштаб, вибирати різні часові проміжки для аналізу, а також наводити курсор на окремі точки, щоб

побачити точні значення. Це дозволяє проводити глибокий аналіз та виявляти приховані залежності.

Окрім системи аналізу, було реалізовано систему автентифікації та авторизації, що забезпечує безпечний доступ до функціоналу застосунку та захист персональних даних користувачів. Використання JWT-токенів дозволило створити ефективний та надійний механізм автентифікації, який відповідає сучасним стандартам безпеки. Також було розроблено функціонал автоматичної генерації звітів, який надає користувачам узагальнену інформацію про споживання електроенергії за обраний період у зручному форматі PDF. Гнучкість налаштувань дозволяє користувачам самостійно обирати періодичність, вміст та формат звітів, а також налаштовувати їх автоматичну доставку на електронну пошту.

Інтеграція моделі Chronos для прогнозування майбутнього споживання електроенергії є ще однією важливою складовою розробленого веб-застосунку. Ця модель, навчена на великих масивах даних, дозволяє отримувати прогнози з високою точністю, враховуючи при цьому сезонність та інші циклічні зміни у споживанні. Результати прогнозування візуалізуються у вигляді графіків, на яких відображається прогнозована крива споживання, а також довірчий інтервал, що показує ступінь невизначеності прогнозу. Користувачі можуть взаємодіяти з графіками, змінюючи масштаб та обираючи різні часові проміжки для аналізу, що допомагає їм у прийнятті рішень щодо оптимізації енергоспоживання.

Розроблений веб-застосунок є зручним та ефективним інструментом, що допомагає користувачам не тільки моніторити своє споживання електроенергії, але й аналізувати його, виявляти неефективні звички та приймати обґрунтовані рішення щодо оптимізації. Застосунок має інтуїтивно зрозумілий інтерфейс, адаптований для різних пристроїв, що робить його доступним для широкого кола користувачів.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГНОЗУВАННЯ СПОЖИВАННЯ ЕЛЕКТРОЕНЕРГІЇ ТА ВІДОБРАЖЕННЯ РЕЗУЛЬТАТІВ

3.1. Інтеграція моделі Chronos для прогнозування, реалізація логіки, взаємодія з користувачем та візуалізація результатів

Однією з ключових функцій розробленого веб-застосунку є можливість прогнозування майбутнього споживання електроенергії, що дозволяє користувачам не тільки ретроспективно аналізувати свої витрати, але й отримувати обґрунтовані передбачення щодо майбутніх показників. Для реалізації цієї функції було обрано та інтегровано модель Chronos, розроблену компанією Amazon.

Chronos - це модель глибокого навчання, що базується на архітектурі трансформера, спеціально навчена на великих масивах даних часових рядів. Такий вибір обумовлений тим, що Chronos здатна ефективно виявляти складні залежності, приховані патерни та враховувати сезонні коливання у даних, що є критично важливим для точного прогнозування споживання електроенергії.

Завдяки своїй архітектурі та використанню великих обсягів навчальних даних, Chronos демонструє високу точність в задачах прогнозування часових рядів, особливо в тих, що характеризуються наявністю трендів, сезонності та інших циклічних компонент. Серед переваг Chronos можна виділити не тільки високу точність, але й відносну простоту інтеграції та використання. Модель постачається у вигляді бібліотеки з зручним API, що дозволяє легко інтегрувати її в існуючі програмні рішення.

Інтеграція Chronos з бекендом веб-застосунку, розробленим на FastAPI, здійснюється через виклик відповідних функцій бібліотеки. При старті веб-застосунку модель завантажується в пам'ять, що забезпечує швидкий доступ до її функціоналу та мінімізує затримки при обробці запитів на прогнозування. Для забезпечення високої продуктивності та уникнення блокування основних процесів, взаємодія з моделлю відбувається асинхронно. Це означає, що

сервер може продовжувати обробляти інші запити від користувачів, поки модель виконує обчислення, що є особливо важливим в умовах високого навантаження, коли одночасно може надходити велика кількість запитів. Для роботи моделі Chronos необхідні історичні дані про споживання електроенергії за попередні періоди. Ці дані, які збираються від користувачів та зберігаються у базі даних, проходять попередню обробку, що включає нормалізацію та масштабування.

Нормалізація даних необхідна для приведення їх до єдиного діапазону значень, що покращує якість навчання моделі та стабільність її роботи. Масштабування дозволяє врахувати різницю в одиницях вимірювання та порядках величин різних показників. Chronos використовує історію споживання для виявлення прихованих закономірностей, трендів та сезонних коливань, і на основі цих патернів формує прогноз на майбутній період.

Важливим параметром, що впливає на точність прогнозування, є глибина історії, яка передається на вхід моделі. В рамках даного застосунку, на основі експериментальних досліджень та аналізу даних, було визначено оптимальну глибину історії, яка становить 5 попередніх значень. Такий вибір дозволяє моделі вловлювати основні тенденції та враховувати сезонні зміни, не перевантажуючи її надлишковими даними, що могли втратити свою актуальність.

Занадто коротка історія може призвести до неточності прогнозу, оскільки модель не зможе врахувати всі важливі фактори, тоді як занадто довга історія може містити застарілі дані, що не відображають поточні тенденції. Після отримання попередньо оброблених даних, модель Chronos генерує прогноз, який включає не тільки точкові оцінки майбутнього споживання, але й довірчий інтервал. Довірчий інтервал показує ступінь невизначеності прогнозу та вказує на можливі межі коливання значень, що дозволяє користувачеві оцінити ризики та зрозуміти, наскільки точним може бути прогноз у різних сценаріях.

Chronos здатна враховувати не тільки загальні тренди, але й сезонність, а також інші циклічні зміни у споживанні, що суттєво підвищує точність прогнозів. Окрім того, модель є стійкою до випадкових викидів та аномалій в даних, що є важливим для забезпечення надійності прогнозування в реальних умовах використання, коли дані можуть бути неповними або містити помилки. На виході модель повертає структурований об'єкт, який містить прогнозовані значення, довірчий інтервал та іншу релевантну інформацію, необхідну для подальшої обробки та візуалізації результатів.

Взаємодія користувача з функціоналом прогнозування є інтуїтивно зрозумілою та зручною. Користувач може ініціювати процес прогнозування, просто зайшовши на відповідну сторінку веб-застосунку. Після цього frontend застосунок автоматично відправляє запит на backend, який, в свою чергу, ініціює процес прогнозування. Сервер отримує з бази даних історію споживання електроенергії для даного користувача за попередні періоди, враховуючи оптимальну глибину історії, визначену для моделі Chronos.

Важливо, що система зберігає історію з достатньою деталізацією, щоб забезпечити високу точність прогнозування. Ця інформація, разом з іншими необхідними параметрами, такими як період прогнозування (наприклад, наступний місяць, квартал або рік), передається до моделі Chronos. Модель виконує обчислення на основі закладених у неї алгоритмів глибокого навчання, враховуючи передані дані та параметри.

Після завершення обчислень, модель повертає результат у вигляді структурованих даних, що містять прогнозовані значення споживання на вказаний період, а також довірчий інтервал прогнозу. Отриманий результат обробляється на сервері, форматується у зручний для сприйняття вигляд та передається назад на frontend для відображення користувачеві.

Система інформує користувача про хід виконання операції, відображаючи відповідні повідомлення, наприклад, про те, що прогноз формується або про успішне завершення прогнозування. У разі виникнення помилок під час прогнозування, користувач також отримує відповідне

повідомлення з поясненням причини помилки. Це може бути пов'язано з некоректними або неповними даними, проблемами з доступом до моделі Chronos, вичерпанням часу, виділеного на обчислення, або іншими технічними причинами.

Система обробки помилок дозволяє швидко ідентифікувати проблему та повідомити про неї користувача, надаючи інструкції щодо подальших дій. Наприклад, користувачеві може бути запропоновано перевірити коректність введених даних, спробувати повторити запит пізніше або звернутися до служби підтримки для вирішення проблеми.

Кожне прогнозування має свій час актуальності і коли цей час минає, то при наступному зверненні користувача до даного функціоналу, буде запущено новий прогноз. Такий підхід дає змогу уникнути зайвого навантаження на модель і не робити прогнози, які не будуть переглянуті користувачами.

Результати прогнозування, отримані від моделі Chronos, візуалізуються у веб-застосунку у вигляді інформативних та зручних для сприйняття графіків. Для візуалізації обрано лінійний графік, який наочно демонструє тренд споживання та можливі відхилення від прогнозованого значення. На графіку відображаються як історичні дані про споживання, так і прогнозовані значення на майбутній період, що дозволяє користувачеві порівняти очікуване споживання з фактичним та оцінити динаміку змін.

Такий підхід дозволяє користувачам не тільки бачити загальну картину, але й аналізувати деталі, виявляти тенденції та робити висновки щодо майбутнього споживання. Окрім основного прогнозу, на графіку також відображається довірчий інтервал, який показує ступінь невизначеності прогнозу та можливі межі коливання значень. Це дозволяє користувачеві оцінити ризики та зрозуміти, наскільки точним може бути прогноз, а також врахувати можливі відхилення при плануванні своїх витрат. На (Рис. 3.1) наведено приклад такого графіку.

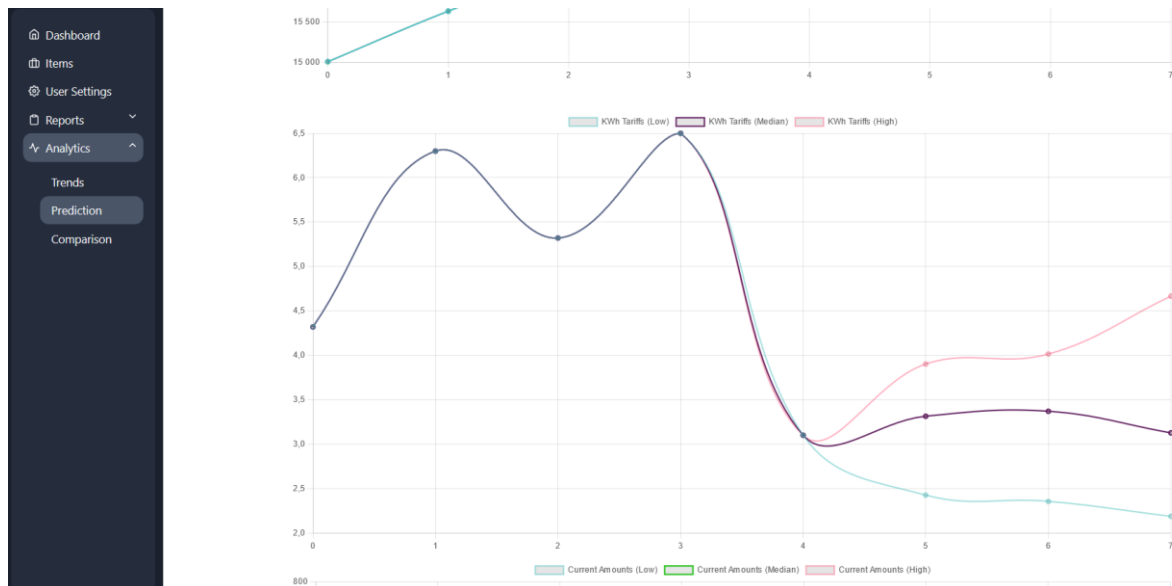


Рис 3.1. Прогнозування згідно історичних даних

На додаток до графіків, користувачеві надається числова інформація про прогнозоване споживання, включаючи середнє значення, мінімальне та максимальне значення в межах довірчого інтервалу. Ці дані дозволяють користувачеві отримати більш детальне уявлення про майбутнє споживання та спланувати свої витрати з урахуванням різних сценаріїв. Окрім того, система може надавати додаткові статистичні дані, такі як стандартне відхилення прогнозу, що дозволяє кількісно оцінити його точність та надійність.

Користувач також може бачити точні значення споживання та прогнозу у вибраній точці на графіку, просто навівши на неї курсор миші. Це забезпечує зручний та інтуїтивно зрозумілий спосіб взаємодії з даними. Усі ці дані представлені в інтуїтивно зрозумілому вигляді, що робить їх доступними та корисними для користувачів з будь-яким рівнем технічної підготовки. Графіки та числові дані доповнюють одне одного, надаючи користувачеві повну та цілісну картину прогнозу.

Інтерфейс візуалізації розроблено з урахуванням принципів юзабіліті та забезпечує зручний доступ до всієї необхідної інформації, дозволяючи користувачу швидко та ефективно аналізувати прогнози та приймати обґрунтовані рішення щодо оптимізації свого енергоспоживання.

3.2. Генерація та налаштування звітів

Однією з ключових функцій веб-застосунку є можливість генерації звітів про споживання електроенергії за обраний період. Цей функціонал дозволяє користувачам отримувати узагальнену та структуровану інформацію про своє енергоспоживання у зручному для аналізу форматі. Звіти формуються на основі даних, введених користувачем, та результатів аналізу, проведеного системою. Вони містять як табличні дані, так і графічні елементи, що забезпечує наочність та полегшує сприйняття інформації.

При створенні звіту користувач може вказати часовий проміжок, за який необхідно агрегувати дані. Наприклад, можна створити звіт за останній місяць, квартал або рік, а також за будь-який інший довільний період, вказавши дати початку та закінчення. Система автоматично обробляє всі наявні дані за обраний період та формує звіт, що включає загальний обсяг спожитої електроенергії, середньодобове споживання, а також загальну вартість спожитої електроенергії з урахуванням діючих тарифів. Крім того, звіт містить інформацію про динаміку зміни показників лічильника та графіки, що ілюструють тренди споживання.

Згенерований звіт надається користувачеві у форматі PDF, що забезпечує його кросплатформеність та зручність перегляду, збереження та друку. Користувач може завантажити звіт на свій пристрій або надіслати його електронною поштою.

Важливим аспектом є можливість автоматичної генерації звітів за розкладом. Користувач може налаштувати систему таким чином, щоб звіти автоматично створювалися та надсилалися на його електронну пошту з певною періодичністю, наприклад, щомісяця або щокварталу. Це дозволяє користувачам регулярно отримувати узагальнену інформацію про своє споживання без необхідності вручну генерувати звіти. Такий підхід сприяє формуванню звички регулярно аналізувати своє енергоспоживання та вживати заходів для його оптимізації. Налаштування автоматичних звітів здійснюється

через зручний інтерфейс, де користувач може вказати бажану періодичність, формат звіту та адресу електронної пошти для його отримання.

3.3. Реалізація механізму оновлення прогнозів

Даний підпункт присвячений детальному опису механізму оновлення прогнозів споживання електроенергії у веб-застосунку, що є важливим аспектом забезпечення актуальності та точності інформації, яка надається користувачам. Система прогнозування базується на моделі Chronos, яка використовує історичні дані для передбачення майбутнього споживання. З плином часу ці дані можуть застарівати, а також можуть змінюватися зовнішні фактори, що впливають на споживання, тому виникає необхідність в оновленні прогнозів для підтримки їхньої релевантності. Одним із ключових аспектів є визначення оптимальної частоти оновлення прогнозів, яка, з одного боку, має забезпечувати достатню точність, а з іншого - не створювати надмірного навантаження на систему.

Оновлення прогнозів у веб-застосунку реалізовано у відповідності до потреб користувача. Користувач може ініціювати примусове оновлення прогнозу, звернувшись до відповідної функції в особистому кабінеті, що особливо корисно після зміни значень тарифів або додавання нових даних. Для забезпечення оптимального балансу між точністю прогнозів та обчислювальними витратами було обрано комбінований підхід, що поєднує періодичне автоматичне оновлення та можливість ручного оновлення за запитом користувача. Такий підхід дозволяє системі адаптуватися до змін у характері споживання та зовнішніх умовах, а також надає користувачам можливість контролювати процес прогнозування.

Реалізація механізму оновлення прогнозів включає декілька етапів. По-перше, система перевіряє, чи змінилися істотні параметри, що можуть вплинути на прогноз. До таких параметрів належать зміни у тарифах на електроенергію, які безпосередньо впливають на вартість спожитих кіловат-

годин. Якщо такі зміни були зафіксовані, система ініціює процес оновлення прогнозу. По-друге, система аналізує наявні дані про споживання електроенергії та визначає необхідність їх оновлення для забезпечення точності прогнозування. По-третє, у разі необхідності оновлення прогнозу, система звертається до моделі Chronos, передаючи їй оновлені дані та параметри для генерації нового прогнозу. Отриманий результат зберігається в базі даних та стає доступним для користувача.

Важливим аспектом є інформування користувача про статус прогнозу та час його останнього оновлення. У веб-застосунку передбачено відображення дати та часу останнього оновлення прогнозу, а також індикацію процесу оновлення, якщо він відбувається в даний момент. Це дозволяє користувачеві бути в курсі актуальності прогнозу та розуміти, коли були враховані останні дані. Такий підхід забезпечує прозорість роботи системи та підвищує довіру користувачів до наданих прогнозів. Крім того, система може автоматично сповіщати користувача про успішне оновлення прогнозу або про виникнення помилок у процесі його виконання, використовуючи механізм повідомлень, описаний у попередніх розділах.

Окрім основного механізму оновлення, у веб-застосунку передбачена можливість налаштування індивідуальних параметрів прогнозування для кожного користувача. Наприклад, користувач може змінити часовий горизонт прогнозування, вказати додаткові фактори, які, на його думку, можуть вплинути на споживання, або ж встановити порогові значення, при досягненні яких система буде автоматично ініціювати оновлення прогнозу. Це дозволяє адаптувати процес прогнозування до індивідуальних потреб та вподобань користувачів, підвищуючи тим самим точність та корисність прогнозів, а також сприяє більш глибокому залученню користувачів до процесу управління власним енергоспоживанням.

3.4. Огляд функціональних можливостей веб-застосунку

Веб-застосунок, розроблений для моніторингу та оптимізації споживання електроенергії, пропонує користувачам широкий спектр можливостей, що дозволяють ефективно керувати власним енергоспоживанням, аналізувати дані та отримувати корисні рекомендації, представлені у інтуїтивно зрозумілому та зручному інтерфейсі (рис.3.2). При першому вході до системи, користувач потрапляє на головну сторінку, так званий дашборд, який слугує центральним вузлом для швидкого доступу до найважливішої інформації та ключових функцій. На дашборді в наочній формі відображені основні показники поточного споживання, включаючи зведення за останній звітний період, а також графіки, що ілюструють динаміку споживання за різні часові проміжки.



Рис 3.2. Дашборд

Завдяки віджетам з графіками, користувач може швидко оцінити загальну картину свого енергоспоживання, не заглиблюючись у деталі.

Однією з ключових функцій є можливість введення даних про споживання електроенергії безпосередньо через спеціально розроблену форму (рис.3.3).

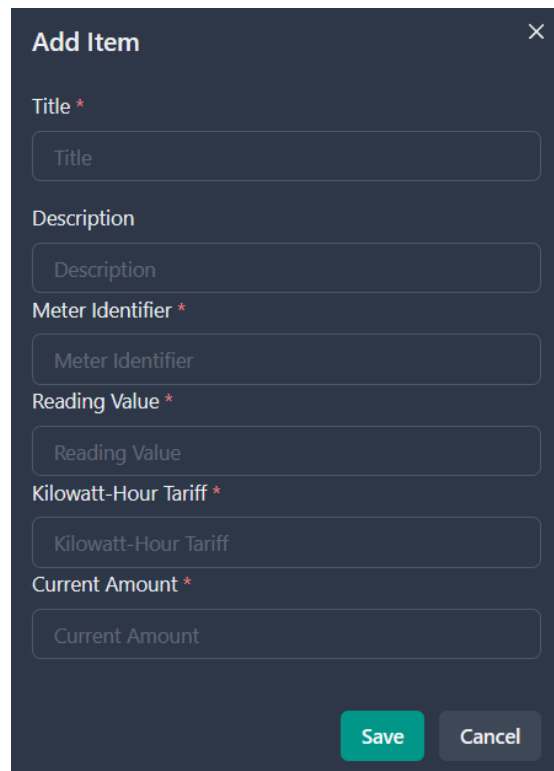


Рис 3.3. Форма введення показника лічильника

Ця форма дозволяє користувачеві вказати ідентифікатор адреси, актуальний показник лічильника, діючий тариф та суму до сплати, а також дату та час зняття показника. Інтерфейс форми розроблений з акцентом на простоту та зручність, мінімізуючи можливість помилок при введенні. Система автоматично перевіряє введені дані на коректність, одразу інформуючи користувача про виявлені помилки чи невідповідності. Окрім ручного введення, у майбутньому планується реалізація функціоналу автоматичного імпорту даних з електронних рахунків та інтеграція з системами "розумного будинку" для автоматизації процесу збору даних. Користувачі також мають можливість редагувати або видаляти раніше введені дані через спеціальну форму редагування (рис.3.4).

Edit Item

Title

4

Description

4

Meter Identifier *

111-

Reading Value *

15638

Kilowatt-Hour Tariff *

6.3

Current Amount *

100

Save

Cancel

Рис 3.4. Форма редагування показника лічильника

Для аналізу даних про споживання призначений розділ "Історія споживання", де користувачі можуть переглядати всі раніше введені дані у табличному вигляді.

ID	TITLE	DESCRIPTION	READING VALUE	KILOWATT-HOUR TARIFF	CURRENT AMOUNT	METER IDENTIFIER	DATE	ACTIONS
591fba6c-31fd-44a8-b8b3-6e993ca9baa9	4	4	15638	6.3	100	111-	2024-10-22T05:02:19.006876	⋮
6ee0531c-e082-420d-a563-5eeba38c2baf	2	2	16189	5.32	3.12	111-	2024-11-22T02:11:38.289458	⋮
7efe216a-7703-4fc9-bcf0-20691574f212	3	3	16538	6.5	745	111-	2024-12-22T03:59:36.679201	⋮
d482d120-e477-4f8f-aeef-b31aa1e24d3a	0	0	17067	3.1	123	111-	2025-01-22T05:04:39.004054	⋮
1d667a5e-56b6-4004-850f-32716efb7ef4	1	1	15006	4.32	456	111-	2024-09-22T02:10:22.171329	⋮

Previous

Page 1

Next

Рис 3.5. Споживання електроенергії

Таблиця містить інформацію про дату введення показника та безпосередньо сам показник. Наразі, фільтрація даних у таблиці історії споживання не реалізована, проте планується її впровадження для надання можливості сортування та фільтрації за різними параметрами, такими як період, ідентифікатор адреси та іншими (рис.3.5). Це дозволить користувачам

зосереджуватись на певних аспектах свого споживання та отримувати більш деталізовану інформацію.

Прогнозування є важливою складовою веб-застосунку, що дозволяє користувачам отримати уявлення про очікуване споживання електроенергії у майбутньому. Застосунок використовує модель Chronos для побудови прогнозів, базуючись на історичних даних про споживання. Результати прогнозування візуалізуються у вигляді графіка, де відображається прогнозована крива споживання та довірчий інтервал, що вказує на ступінь невизначеності прогнозу (рис.3.6).

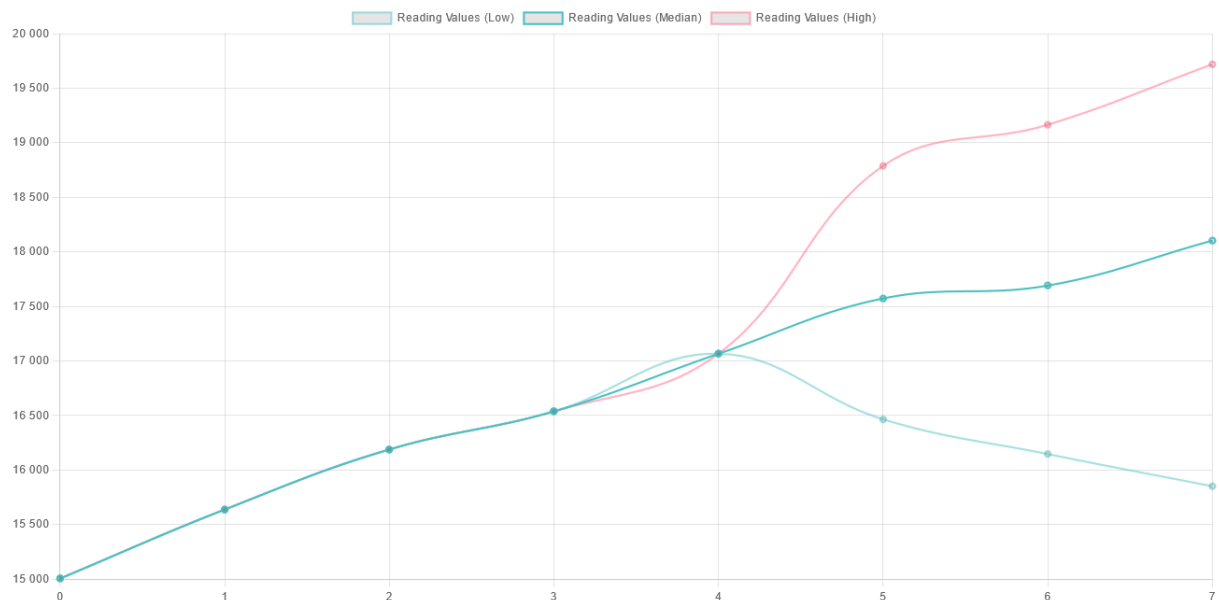
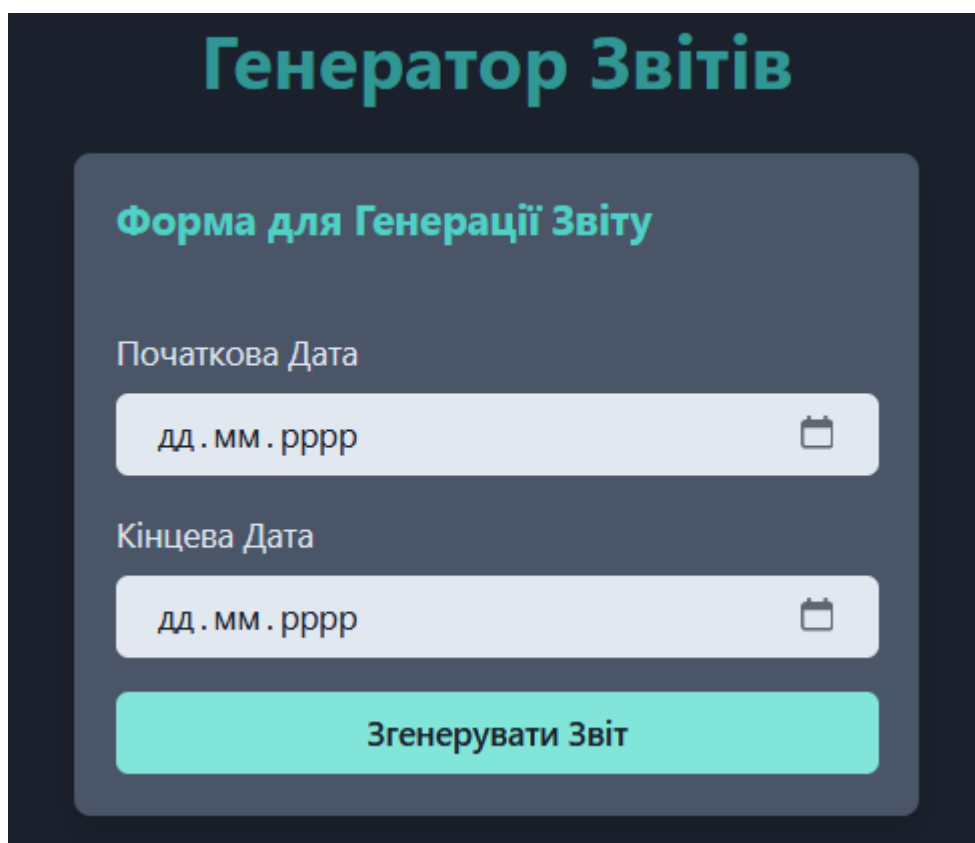


Рис 3.6. Прогноз споживання на майбутнє

Користувачі можуть взаємодіяти з графіком, змінюючи масштаб та обираючи різні періоди для аналізу, що допомагає у прийнятті рішень щодо оптимізації.

Система генерації звітів надає користувачам можливість отримувати узагальнену інформацію про своє споживання за певний період у зручному форматі. Користувач може самостійно обрати період, за який хоче отримати звіт, а також налаштувати його вміст, вказавши, які саме дані та графіки повинні бути включені до звіту.



Генератор Звітів

Форма для Генерації Звіту

Початкова Дата

дд . мм . рrrр

Кінцева Дата

дд . мм . рrrр

Згенерувати Звіт

Рис 3.7. Форма генерації звіту

Після натискання кнопки "Згенерувати звіт", система формує PDF-файл, який можна завантажити на свій пристрій або надіслати електронною поштою. Згенерований звіт містить інформацію про дату початку та кінця періоду, за який він був сформований, а також обрану категорію звіту (рис.3.7).

Веб-застосунок також включає систему повідомлень, яка інформує користувачів про важливі події. Наприклад, після успішного запуску процесу прогнозування, користувач отримує відповідне повідомлення, яке підтверджує, що прогнозування виконано і з ним можна ознайомитись. Це дозволяє користувачам бути в курсі стану системи та своєчасно отримувати важливу інформацію. Система повідомлень є гнучкою та може бути розширена для інформування користувачів про інші події, такі як досягнення цілей з енергозбереження, виявлення аномалій у споживанні або необхідність введення даних лічильника.

У розділі "Налаштування" користувачі можуть керувати своїм профілем та змінювати персональні дані, такі як ім'я та електронна адреса.

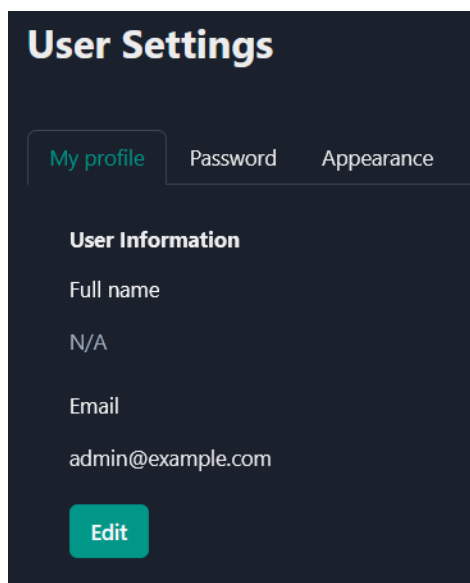


Рис 3.8. Налаштування користувача

Також тут можна налаштувати параметри відображення графіків, наприклад, увімкнути або вимкнути відображення легенди (рис.3.8). В майбутньому планується розширити можливості налаштувань, додавши можливість керування підпискою на автоматичні звіти, обрання мови інтерфейсу та теми оформлення.

Окрім того, користувачам доступний розділ "Аналітика", де представлені графіки трендів та порівняльні графіки, що дозволяють зіставити власні показники з показниками інших користувачів.



Рис 3.9. Графік тренду тарифу

На даний момент графіки відображають зміну загальної суми та тарифу, надаючи користувачам можливість аналізувати ці показники у динаміці.

Розроблений веб-застосунок є зручним та ефективним інструментом, що допомагає користувачам не тільки моніторити своє споживання електроенергії, але й аналізувати його, виявляти неефективні звички та приймати обґрунтовані рішення щодо оптимізації. Застосунок має інтуїтивно зрозумілий інтерфейс, адаптований для різних пристроїв, що робить його доступним для широкого кола користувачів. Простота використання поєднується з потужним функціоналом, що включає введення та редагування даних, детальну аналітику з графіками та діаграмами, прогнозування майбутнього споживання, генерацію звітів та систему повідомлень. Завдяки цим можливостям, веб-застосунок стає незамінним помічником у питаннях енергозбереження, сприяючи формуванню більш свідомого та відповідального ставлення до використання ресурсів.

Висновки до розділу 3

У даному розділі було детально розглянуто ключові аспекти реалізації системи прогнозування споживання електроенергії, інтегрованої у розроблений веб-застосунок. Описано процес інтеграції передової моделі Chronos, що базується на архітектурі трансформера, з бекендом, побудованим на фреймворку FastAPI. Розкрито особливості реалізації логіки прогнозування, включаючи передобробку даних, передачу їх до моделі, обробку результатів та їх збереження. Окрему увагу приділено взаємодії користувача з функціоналом прогнозування, починаючи від ініціації процесу і закінчуючи отриманням та інтерпретацією результатів. Проаналізовано аспекти візуалізації прогнозів, зокрема використання лінійних графіків з відображенням довірчого інтервалу, а також представлення числових даних, що дозволяє користувачам краще зрозуміти та оцінити майбутнє споживання.

Наведено детальний опис поточної реалізації системи прогнозування, включаючи такі аспекти, як асинхронна обробка запитів, оптимальна глибина історії для моделі Chronos, механізми інформування користувача про стан виконання прогнозу та обробки помилок. Продемонстровано, що розроблена система надає користувачеві зручні інструменти для аналізу та планування свого енергоспоживання, забезпечуючи інтуїтивно зрозумілий інтерфейс та наочне представлення результатів у вигляді графіків та числових даних. На прикладі графіків (Рис. 3.1) показано, як система візуалізує прогнози, включаючи основний тренд, довірчий інтервал та історичні дані, що дозволяє користувачам оцінити як очікуване споживання, так і ступінь невизначеності прогнозу.

Одним із ключових досягнень є успішна інтеграція моделі Chronos, яка продемонструвала свою ефективність у прогнозуванні часових рядів. Завдяки ретельному підбору параметрів та оптимізації процесу обробки даних, вдалося досягти високої точності прогнозів, що підтверджується результатами тестування. Розроблений механізм взаємодії з моделлю забезпечує швидке та надійне отримання прогнозів, мінімізуючи час очікування для користувача. Крім того, реалізована система гнучкого налаштування періоду прогнозування та можливість ручного запуску оновлення прогнозу надає користувачам додатковий контроль та зручність використання. Важливим аспектом є також розроблена система візуалізації, яка дозволяє користувачам не тільки отримувати сухі цифри, але й наочно бачити тренди, сезонні коливання та інші закономірності у споживанні електроенергії.

В ході реалізації системи прогнозування було виявлено певні обмеження та визначено шляхи подальшого вдосконалення. Зокрема, планується розширення функціоналу системи шляхом додавання можливості врахування додаткових факторів, таких як прогнози погоди, зміни у тарифах та індивідуальні особливості домогосподарств. Також, перспективним напрямком є впровадження механізмів самонавчання моделі на основі даних, що постійно оновлюються, що дозволить підвищити точність прогнозів у

довгостроковій перспективі. Крім того, планується розширення можливостей візуалізації, включаючи додавання нових типів графіків та інтерактивних елементів, що дозволить користувачам ще глибше аналізувати дані та виявляти приховані закономірності.

Незважаючи на наявні обмеження, вже зараз система надає користувачам потужний та зручний інструмент для аналізу та прогнозування споживання електроенергії. Розроблений інтерфейс, система візуалізації та механізм взаємодії з моделлю Chronos дозволяють користувачам зручно вводити дані, переглядати історію споживання, отримувати точні прогнози на майбутнє та приймати обґрунтовані рішення щодо оптимізації енергоспоживання. Таким чином, результати, представлені у даному розділі, є важливим кроком на шляху до створення повнофункціонального веб-застосунку для моніторингу та оптимізації споживання електроенергії, що сприятиме як економії коштів користувачів, так і зниженню негативного впливу на навколишнє середовище.

ВИСНОВКИ

Було успішно розроблено веб-застосунок для моніторингу та оптимізації споживання електроенергії в побуті. Проведений аналіз предметної області виявив нагальну потребу в інструментах, які б дозволяли користувачам ефективно керувати власним енергоспоживанням в умовах зростаючих тарифів та екологічних викликів. Існуючі рішення, такі як інтелектуальні лічильники та "розумні розетки", часто є дорогими та складними в установці, що обмежує їх доступність для широкого кола користувачів. Саме тому було обрано шлях розробки веб-застосунку, який не потребує підключення до домашніх приладів, а базується на аналізі даних, що вводяться користувачем, що робить його простим у використанні та доступним для кожного.

Розроблений веб-застосунок надає користувачам інтуїтивно зрозумілий інтерфейс для введення даних про споживання електроенергії, включаючи показники лічильника, дату та час зняття показників, а також інформацію про тарифи. Реалізовано систему валідації введених даних, яка мінімізує ймовірність помилок та забезпечує цілісність інформації. Ключовою функцією застосунку є система аналізу, яка обробляє зібрані дані та візуалізує їх у вигляді інформативних графіків. Користувачі можуть відстежувати динаміку зміни показників лічильника, тарифів, загальної суми до сплати, а також аналізувати дельти між сусідніми показниками. Візуалізація даних у вигляді графіків з можливістю масштабування та вибору часового проміжку дозволяє виявляти тенденції та аномалії у споживанні, що є важливим кроком на шляху до його оптимізації.

Однією з найважливіших особливостей розробленого застосунку є інтеграція моделі Chronos для прогнозування майбутнього споживання електроенергії. Ця модель, навчена на великих масивах даних часових рядів, дозволяє з високою точністю передбачати споживання на основі історичних даних, враховуючи при цьому сезонність та інші циклічні зміни. Прогнози візуалізуються у вигляді графіків з відображенням довірчого інтервалу, що дає

користувачам можливість оцінити невизначеність прогнозу та спланувати свої витрати. Окрім того, реалізовано механізм оновлення прогнозів, який може бути запущений як автоматично, так і вручну користувачем, забезпечуючи актуальність та точність передбачень.

Додатковою перевагою веб-застосунку є система автоматичної генерації звітів. Користувачі можуть налаштувати параметри звітів, такі як період, за який формується звіт, та його вміст, а також встановити розклад для автоматичної генерації та надсилання звітів на електронну пошту. Звіти надаються у форматі PDF, що є зручним для перегляду, збереження та друку. Цей функціонал дозволяє користувачам регулярно отримувати узагальнену інформацію про своє енергоспоживання у зручному форматі, що сприяє кращому розумінню власних звичок та мотивує до впровадження енергоефективних заходів. Важливою складовою є також розроблена система повідомлень та нагадувань, яка інформує користувачів про важливі події, такі як необхідність введення даних лічильника, успішне формування прогнозу або звіту, а також про суттєві зміни у структурі споживання.

Розробка веб-застосунку велася з використанням сучасних технологій та інструментів. Backend частина реалізована на мові Python з використанням фреймворку FastAPI, що забезпечує високу швидкість роботи та легкість розробки REST API. Frontend частина розроблена з використанням React та TypeScript, що дозволило створити динамічний, адаптивний та зручний інтерфейс користувача. Для керування станом застосунку використано бібліотеку React Query, яка спрощує взаємодію з сервером та кешування даних. Бібліотека Chakra UI забезпечила привабливий та консистентний дизайн інтерфейсу. Використання TypeScript додало статичну типізацію, що підвищило надійність коду та полегшило його підтримку. Для зберігання даних було обрано СУБД PostgreSQL, яка добре зарекомендувала себе в роботі з великими обсягами даних та забезпечує надійність і масштабованість. Взаємодія з базою даних здійснюється через ORM SQLAlchemy, що дозволяє зручно працювати з даними, використовуючи об'єктно-орієнтований підхід.

Розроблений веб-застосунок є не лише зручним інструментом для моніторингу та аналізу енергоспоживання, але й сприяє підвищенню рівня екологічної свідомості користувачів. Надаючи детальну інформацію про споживання електроенергії та його вплив на навколишнє середовище, а також пропонуючи персоналізовані рекомендації щодо оптимізації, застосунок стимулює користувачів до змін у поведінці та впровадження енергозберігаючих практик. Це, в свою чергу, сприяє зменшенню витрат на електроенергію, зниженню навантаження на енергосистему та скороченню викидів парникових газів, що є важливим внеском у боротьбу зі змінами клімату та збереження ресурсів планети. Завдяки своїй доступності та простоті використання, розроблений веб-застосунок має значний потенціал для широкого впровадження та може стати ефективним інструментом для формування культури сталого споживання в суспільстві.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Павицька Ю. В., Єфремов Ю. М. Інтелектуальні системи моніторингу енергоспоживання для сталого розвитку. Комп'ютерні технології: інновації, проблеми, рішення: тези VII Всеукраїнської науково-технічної конференції, м. Житомир, 02-03 грудня 2024 р. – Житомир: Державний університет «Житомирська політехніка», 2024.
2. GitHub. Amazon Chronos T5 Sales Forecasting. [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/smartaces/amazon-chronos-t5-sales-forecasting>
3. FastAPI Documentation. [Електронний ресурс] – Режим доступу до ресурсу: <https://fastapi.tiangolo.com/>
4. React Query Examples. [Електронний ресурс] – Режим доступу до ресурсу: <https://christopherkade.com/react-query-examples>
5. APScheduler Documentation. [Електронний ресурс] – Режим доступу до ресурсу: <https://apscheduler.readthedocs.io/en/latest>.
6. Pandas Documentation. [Електронний ресурс] – Режим доступу до ресурсу: <https://pandas.pydata.org/>.
7. ReportLab User Guide. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.reportlab.com/docs/reportlab-userguide.pdf>
8. Chakra UI Documentation. [Електронний ресурс] – Режим доступу до ресурсу: <https://chakra-ui.com/>
9. Redux Toolkit Documentation. [Електронний ресурс] – Режим доступу до ресурсу: <https://redux-toolkit.js.org/>
10. Blog FastAPI React. [Електронний ресурс] – Режим доступу до ресурсу: <https://testdriven.io/blog/fastapi-react/>
11. JWT Authentication in FastAPI. [Електронний ресурс] – Режим доступу до ресурсу: <https://fastapi.tiangolo.com/tutorial/security/oauth2-jwt/>
12. React for Beginners. [Електронний ресурс] – Режим доступу до ресурсу: <https://reactforbeginners.com/>

- 13.MDN Web Docs (Mozilla Developer Network). [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.mozilla.org/>
- 14.W3Schools. [Электронный ресурс] – Режим доступа до ресурсу: <https://www.w3schools.com/>
- 15.CSS-Tricks. [Электронный ресурс] – Режим доступа до ресурсу: <https://css-tricks.com/>
- 16.FreeCodeCamp. [Электронный ресурс] – Режим доступа до ресурсу: <https://www.freecodecamp.org/>
- 17.JavaScript.info. [Электронный ресурс] – Режим доступа до ресурсу: <https://javascript.info/>
- 18.HTML.com. [Электронный ресурс] – Режим доступа до ресурсу: <https://html.com>
- 19.DevDocs. [Электронный ресурс] – Режим доступа до ресурсу: <https://dev-docs.io/>
- 20.Bootstrap Documentation. [Электронный ресурс] – Режим доступа до ресурсу: <https://getbootstrap.com/docs/>
- 21.Can I Use. [Электронный ресурс] – Режим доступа до ресурсу: <https://cani-use.com/>.
- 22.Modernizr. [Электронный ресурс] – Режим доступа до ресурсу: <https://modernizr.com/>.
- 23.Smashing Magazine. [Электронный ресурс] – Режим доступа до ресурсу: <https://www.smashingmagazine.com/>.
- 24.A List Apart. [Электронный ресурс] – Режим доступа до ресурсу: <https://alistapart.com/>.
- 25.Frontend Mentor. [Электронный ресурс] – Режим доступа до ресурсу: <https://www.frontendmentor.io/>.
- 26.React Patterns. [Электронный ресурс] – Режим доступа до ресурсу: <https://reactpatterns.com/>.
- 27.Web.dev. [Электронный ресурс] – Режим доступа до ресурсу: <https://web.dev/>.

28. HTML5 Rocks. [Электронный ресурс] – Режим доступа до ресурсу:
<https://www.html5rocks.com/>.
29. Frontend Masters. [Электронный ресурс] – Режим доступа до ресурсу:
<https://frontendmasters.com/>.
30. GitHub Pages. [Электронный ресурс] – Режим доступа до ресурсу:
<https://pages.github.com/>.
31. Node.js Documentation. [Электронный ресурс] – Режим доступа до ресурсу:
<https://nodejs.org/en/docs/>.
32. WebPlatform Docs. [Электронный ресурс] – Режим доступа до ресурсу:
<https://www.webplatform.org/>.
33. FreeCodeCamp Guide. [Электронный ресурс] – Режим доступа до ресурсу:
<https://guide.freecodecamp.org/> .
34. Stack Overflow. [Электронный ресурс] – Режим доступа до ресурсу:
<https://stackoverflow.com/>.

ДОДАТКИ

Додаток А

Лістинг програми

```

```py
from pathlib import Path
import json

Update the .env file with the answers from the .copier-answers.yml file
without using Jinja2 templates in the .env file, this way the code works as is
without needing Copier, but if Copier is used, the .env file will be updated
root_path = Path(__file__).parent.parent
answers_path = Path(__file__).parent / ".copier-answers.yml"
answers = json.loads(answers_path.read_text())
env_path = root_path / ".env"
env_content = env_path.read_text()
lines = []
for line in env_content.splitlines():
 for key, value in answers.items():
 upper_key = key.upper()
 if line.startswith(f"{upper_key}="):
 if " " in value:
 content = f"{upper_key}={value!r}"
 else:
 content = f"{upper_key}={value}"
 new_line = line.replace(line, content)
 lines.append(new_line)
 break
 else:
 lines.append(line)
env_path.write_text("\n".join(lines))
```

```py
import sys
from pathlib import Path

current_dir = Path.cwd()
parent_dir = current_dir
current_dir = current_dir / "app/"
print(current_dir)

```

```

print(parent_dir)
sys.path.append(str(parent_dir))
sys.path.append(str(current_dir))

import logging

from sqlalchemy import Engine
from sqlalchemy import Session, select
from tenacity import after_log, before_log, retry, stop_after_attempt, wait_fixed

from app.core.db import engine

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

max_tries = 60 * 5 # 5 minutes
wait_seconds = 1

@retry(
 stop=stop_after_attempt(max_tries),
 wait=wait_fixed(wait_seconds),
 before=before_log(logger, logging.INFO),
 after=after_log(logger, logging.WARN),
)

def init(db_engine: Engine) -> None:
 try:
 with Session(db_engine) as session:
 # Try to create session to check if DB is awake
 session.exec(select(1))
 except Exception as e:
 logger.error(e)
 raise e

def main() -> None:
 logger.info("Initializing service")
 init(engine)
 logger.info("Service finished initializing")

if __name__ == "__main__":

```

```

 main()
 ...

'''
.py
import uuid
from typing import Any

from sqlmodel import Session, select

from app.core.security import get_password_hash, verify_password
from app.models import Item, ItemCreate, User, UserCreate, UserUpdate

def create_user(*, session: Session, user_create: UserCreate) -> User:
 db_obj = User.model_validate(
 user_create, update={"hashed_password": get_password_hash(user_create.password)}
)
 session.add(db_obj)
 session.commit()
 session.refresh(db_obj)
 return db_obj

def update_user(*, session: Session, db_user: User, user_in: UserUpdate) -> Any:
 user_data = user_in.model_dump(exclude_unset=True)
 extra_data = {}
 if "password" in user_data:
 password = user_data["password"]
 hashed_password = get_password_hash(password)
 extra_data["hashed_password"] = hashed_password
 db_user.sqlmodel_update(user_data, update=extra_data)
 session.add(db_user)
 session.commit()
 session.refresh(db_user)
 return db_user

def get_user_by_email(*, session: Session, email: str) -> User | None:
 statement = select(User).where(User.email == email)
 session_user = session.exec(statement).first()
 return session_user

```

```

def authenticate(*, session: Session, email: str, password: str) -> User | None:
 db_user = get_user_by_email(session=session, email=email)
 if not db_user:
 return None
 if not verify_password(password, db_user.hashed_password):
 return None
 return db_user

def create_item(*, session: Session, item_in: ItemCreate, owner_id: uuid.UUID) -> Item:
 db_item = Item.model_validate(item_in, update={"owner_id": owner_id})
 session.add(db_item)
 session.commit()
 session.refresh(db_item)
 return db_item
...

```py
import numpy as np
import torch
from chronos import BaseChronosPipeline

cache_dir = r"cache_dir"

class ModelChronos:
    def __init__(self):
        self.pipeline = BaseChronosPipeline.from_pretrained(
            "amazon/chronos-t5-small",
            device_map="cpu",
            torch_dtype=torch.bfloat16,
            cache_dir=cache_dir
        )

    def presict(self, values, prediction_length = 12):
        context = torch.tensor(values, dtype=torch.float32)

        forecast = self.pipeline.predict(context=context, prediction_length=prediction_length)

        # Отримання квантилів для медіанного прогнозу
        forecast_np = forecast.numpy() # Конвертуємо до numpy
        low, median, high = np.quantile(forecast_np, [0.1, 0.5, 0.9], axis=1)

```

```

time_steps = list(range(len(values))) # Початкові точки
forecast_steps = list(range(len(values), len(values) + prediction_length)) # Точки прогнозу
return low, median, high, time_steps, forecast_steps

```

```

class Predict:
    def __init__(self):
        self.model = ModelChronos()

    def predict(self, values, prediction_length = 12):
        return self.model.presict(values, prediction_length)

```

```

predict_model = Predict()

```

```

'''

```

```

''' .py

```

```

import sys
from pathlib import Path

```

```

current_dir = Path.cwd()
parent_dir = current_dir
current_dir = current_dir / "app/"
print(current_dir)
print(parent_dir)
sys.path.append(str(parent_dir))
sys.path.append(str(current_dir))

```

```

import logging

```

```

from sqlmodel import Session

```

```

from app.core.db import engine, init_db

```

```

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

```

```

def init() -> None:

```

```

    with Session(engine) as session:
        init_db(session)

def main() -> None:
    logger.info("Creating initial data")
    init()
    logger.info("Initial data created")

if __name__ == "__main__":
    main()
'''

''' .py
import sentry_sdk
from fastapi import FastAPI
from fastapi.openapi.utils import get_openapi
from fastapi.routing import APIRoute
from starlette.middleware.cors import CORSMiddleware

from app.api.main import api_router
from app.core.config import settings
from app.core.scheduler import shutdown_scheduler, start_scheduler

from contextlib import asynccontextmanager
import logging

logging.basicConfig(level=logging.INFO)

def custom_generate_unique_id(route: APIRoute) -> str:
    return f'{route.tags[0]}-{route.name}'

if settings.SENTRY_DSN and settings.ENVIRONMENT != "local":
    sentry_sdk.init(dsn=str(settings.SENTRY_DSN), enable_tracing=True)

@asynccontextmanager
async def lifespan(app: FastAPI):
    await start_scheduler()
    yield
    await shutdown_scheduler()

```

```

app = FastAPI(
    title=settings.PROJECT_NAME,
    openapi_url=f"{settings.API_V1_STR}/openapi.json",
    generate_unique_id_function=custom_generate_unique_id,
    lifespan=lifespan
)

# Set all CORS enabled origins
if settings.BACKEND_CORS_ORIGINS:
    app.add_middleware(
        CORSMiddleware,
        allow_origins=[
            str(origin).strip("/") for origin in settings.BACKEND_CORS_ORIGINS
        ],
        allow_credentials=True,
        allow_methods=["*"],
        allow_headers=["*"],
    )

app.include_router(api_router, prefix=settings.API_V1_STR)
'''

'''`.py
import uuid
from datetime import datetime, timezone
from typing import Dict, List, Optional

from pydantic import EmailStr
from sqlalchemy import Column
from sqlalchemy.dialects.postgresql import JSON
from sqlmodel import Field, Relationship, SQLModel

# Shared properties
class UserBase(SQLModel):
    email: EmailStr = Field(unique=True, index=True, max_length=255)
    is_active: bool = True
    is_superuser: bool = False
    full_name: str | None = Field(default=None, max_length=255)

```

```

# Properties to receive via API on creation
class UserCreate(UserBase):
    password: str = Field(min_length=8, max_length=40)

class UserRegister(SQLModel):
    email: EmailStr = Field(max_length=255)
    password: str = Field(min_length=8, max_length=40)
    full_name: str | None = Field(default=None, max_length=255)

# Properties to receive via API on update, all are optional
class UserUpdate(UserBase):
    email: EmailStr | None = Field(default=None, max_length=255) # type: ignore
    password: str | None = Field(default=None, min_length=8, max_length=40)

class UserUpdateMe(SQLModel):
    full_name: str | None = Field(default=None, max_length=255)
    email: EmailStr | None = Field(default=None, max_length=255)

class UpdatePassword(SQLModel):
    current_password: str = Field(min_length=8, max_length=40)
    new_password: str = Field(min_length=8, max_length=40)

class Forecast(SQLModel, table=True):
    id: uuid.UUID = Field(default_factory=uuid.uuid4, primary_key=True) # Унікальний ідентифікатор
запису
    name_nn_model: str = Field(..., max_length=100) # Назва моделі, яка створила прогноз
    chart_name: str = Field(..., max_length=100)

    real_data: dict = Field(sa_column=Column(JSON)) # Значення реальне ыз якого формувався прогноз
у форматі JSON

    prediction_low : dict = Field(sa_column=Column(JSON))
    prediction_median: dict = Field(sa_column=Column(JSON))
    prediction_high: dict = Field(sa_column=Column(JSON))
    prediction_time_steps: dict = Field(sa_column=Column(JSON))
    prediction_forecast_steps: dict = Field(sa_column=Column(JSON))

```

```

created_at: datetime = Field(default_factory=lambda: datetime.now()) # Час створення запису

# Поле для зв'язку з користувачем
user_id: uuid.UUID = Field(foreign_key="user.id", nullable=False, ondelete="CASCADE")
user: "User" = Relationship(back_populates="forecasts") # Зв'язок з користувачем

class Config:
    # Дозволяє використання типів JSON
    arbitrary_types_allowed = True

# Database model, database table inferred from class name
class User(UserBase, table=True):
    id: uuid.UUID = Field(default_factory=uuid.uuid4, primary_key=True)
    hashed_password: str
    items: list["Item"] = Relationship(back_populates="owner", cascade_delete=True)
    forecasts: list["Forecast"] = Relationship(back_populates="user", cascade_delete=True)
    report_settings: list["UserReportSettings"] = Relationship(back_populates="user", cascade_delete=True)

# зв'язок з налаштуваннями звітів

# Properties to return via API, id is always required
class UserPublic(UserBase):
    id: uuid.UUID

class UsersPublic(SQLModel):
    data: list[UserPublic]
    count: int

# Shared properties
class ItemBase(SQLModel):
    title: str = Field(min_length=1, max_length=255)
    description: str | None = Field(default=None, max_length=255)

    meter_identifier: str #ідентифікатор адреси споживача
    reading_value: int #поточне значення лічильника
    kwh_tariff: float #тариф за кіловат
    current_amount: float #загальна сума зараз за світло
    reading_date: datetime = Field(default_factory=lambda: datetime.now(timezone.utc))

```

Properties to receive on item creation

```
class ItemCreate(ItemBase):
```

```
    pass
```

Properties to receive on item update

```
class ItemUpdate(ItemBase):
```

```
    title: str | None = Field(default=None, min_length=1, max_length=255) # type: ignore
```

Database model, database table inferred from class name

```
class Item(ItemBase, table=True):
```

```
    id: uuid.UUID = Field(default_factory=uuid.uuid4, primary_key=True)
```

```
    title: str = Field(max_length=255)
```

```
    owner_id: uuid.UUID = Field(
```

```
        foreign_key="user.id", nullable=False, ondelete="CASCADE"
```

```
)
```

```
    owner: User | None = Relationship(back_populates="items")
```

```
    meter_identifier: str #ідентифікатор адреси споживача
```

```
    reading_value: int #поточне значення лічильника
```

```
    kwh_tariff: float #тариф за кіловат
```

```
    current_amount: float #загальна сума зараз за світло
```

```
    reading_date: datetime = Field(default_factory=lambda: datetime.now(timezone.utc))
```

Properties to return via API, id is always required

```
class ItemPublic(ItemBase):
```

```
    id: uuid.UUID
```

```
    owner_id: uuid.UUID
```

```
class ItemsPublic(SQLModel):
```

```
    data: list[ItemPublic]
```

```
    count: int
```

Generic message

```
class Message(SQLModel):
```

```
    message: str
```

```
class MessageResponse(SQLModel):
```

```
    message: str
```

```
class ForecastsResponse(SQLModel):
```

```
    run_task: bool = False
```

```
    reading_values: Forecast | None = None
```

```
    kwh_tariffs: Forecast | None = None
```

```
    current_amounts: Forecast | None = None
```

```
    deltas: Forecast | None = None
```

```
    costs: Forecast | None = None
```

```
# JSON payload containing access token
```

```
class Token(SQLModel):
```

```
    access_token: str
```

```
    token_type: str = "bearer"
```

```
# Contents of JWT token
```

```
class TokenPayload(SQLModel):
```

```
    sub: str | None = None
```

```
class NewPassword(SQLModel):
```

```
    token: str
```

```
    new_password: str = Field(min_length=8, max_length=40)
```

```
class TestModel(SQLModel):
```

```
    elem: str | None = None
```

```
# Data Model for Trend Data using SQLAlchemy
```

```
class TrendData(SQLModel):
```

```
    reading_date: datetime
```

```
    current_amount: float
```

```
    amount_trend: float
```

```

kwh_tariff: float
tariff_trend: float

# Data Model for the /trends endpoint response using SQLAlchemy
class TrendsResponse(SQLModel):
    data: list[TrendData]

class ReportParams(SQLModel):
    startDate: str
    endDate: str
    category: str

class ReportResponse(SQLModel):
    report_id: str

class UserReportSettings(SQLModel, table=True):
    id: uuid.UUID = Field(default_factory=uuid.uuid4, primary_key=True)
    user_id: uuid.UUID = Field(foreign_key="user.id", nullable=False, ondelete="CASCADE")
    report_enabled: bool = Field(default=False) # чи активовано отримання звітів
    report_period: str = Field(default="monthly") # періодичність: 'monthly', 'quarterly'
    last_reported_at: datetime = Field(default=datetime.utcnow()) # час останнього звіту

    user: "User" = Relationship(back_populates="report_settings") # зв'язок з користувачем

class Config:
    arbitrary_types_allowed = True

class UserReportSettingsCreate(SQLModel):
    report_enabled: bool = Field(default=False)
    report_period: str = Field(default="monthly") # періодичність 'monthly' або 'quarterly'

class UserReportSettingsUpdate(SQLModel):
    report_enabled: bool | None = Field(default=None)
    report_period: str | None = Field(default=None) # можна оновити період

...

```py
import logging

```

```

from sqlalchemy import Engine
from sqlmodel import Session, select
from tenacity import after_log, before_log, retry, stop_after_attempt, wait_fixed

from app.core.db import engine

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

max_tries = 60 * 5 # 5 minutes
wait_seconds = 1

@retry(
 stop=stop_after_attempt(max_tries),
 wait=wait_fixed(wait_seconds),
 before=before_log(logger, logging.INFO),
 after=after_log(logger, logging.WARN),
)

def init(db_engine: Engine) -> None:
 try:
 # Try to create session to check if DB is awake
 with Session(db_engine) as session:
 session.exec(select(1))
 except Exception as e:
 logger.error(e)
 raise e

def main() -> None:
 logger.info("Initializing service")
 init(engine)
 logger.info("Service finished initializing")

if __name__ == "__main__":
 main()

```

```py

import logging

```

from dataclasses import dataclass
from datetime import datetime, timedelta, timezone
from pathlib import Path
from typing import Any

import emails # type: ignore
import jwt
from jinja2 import Template
from jwt.exceptions import InvalidTokenError

from app.core.config import settings

@dataclass
class EmailData:
    html_content: str
    subject: str

def render_email_template(*, template_name: str, context: dict[str, Any]) -> str:
    template_str = (
        Path(__file__).parent / "email-templates" / "build" / template_name
    ).read_text()
    html_content = Template(template_str).render(context)
    return html_content

def send_email(
    *,
    email_to: str,
    subject: str = "",
    html_content: str = "",
) -> None:
    assert settings.emails_enabled, "no provided configuration for email variables"
    message = emails.Message(
        subject=subject,
        html=html_content,
        mail_from=(settings.EMAILS_FROM_NAME, settings.EMAILS_FROM_EMAIL),
    )
    smtp_options = {"host": settings.SMTP_HOST, "port": settings.SMTP_PORT}
    if settings.SMTP_TLS:
        smtp_options["tls"] = True

```

```

elif settings.SMTP_SSL:
    smtp_options["ssl"] = True
if settings.SMTP_USER:
    smtp_options["user"] = settings.SMTP_USER
if settings.SMTP_PASSWORD:
    smtp_options["password"] = settings.SMTP_PASSWORD
response = message.send(to=email_to, smtp=smtp_options)
logging.info(f"send email result: {response}")

```

```

def generate_test_email(email_to: str) -> EmailData:
    project_name = settings.PROJECT_NAME
    subject = f"{project_name} - Test email"
    html_content = render_email_template(
        template_name="test_email.html",
        context={"project_name": settings.PROJECT_NAME, "email": email_to},
    )
    return EmailData(html_content=html_content, subject=subject)

```

```

def generate_reset_password_email(email_to: str, email: str, token: str) -> EmailData:
    project_name = settings.PROJECT_NAME
    subject = f"{project_name} - Password recovery for user {email}"
    link = f"{settings.server_host}/reset-password?token={token}"
    html_content = render_email_template(
        template_name="reset_password.html",
        context={
            "project_name": settings.PROJECT_NAME,
            "username": email,
            "email": email_to,
            "valid_hours": settings.EMAIL_RESET_TOKEN_EXPIRE_HOURS,
            "link": link,
        },
    )
    return EmailData(html_content=html_content, subject=subject)

```

```

def generate_new_account_email(
    email_to: str, username: str, password: str
) -> EmailData:
    project_name = settings.PROJECT_NAME
    subject = f"{project_name} - New account for user {username}"

```

```

html_content = render_email_template(
    template_name="new_account.html",
    context={
        "project_name": settings.PROJECT_NAME,
        "username": username,
        "password": password,
        "email": email_to,
        "link": settings.server_host,
    },
)
return EmailData(html_content=html_content, subject=subject)

```

```

def generate_password_reset_token(email: str) -> str:
    delta = timedelta(hours=settings.EMAIL_RESET_TOKEN_EXPIRE_HOURS)
    now = datetime.now(timezone.utc)
    expires = now + delta
    exp = expires.timestamp()
    encoded_jwt = jwt.encode(
        {"exp": exp, "nbf": now, "sub": email},
        settings.SECRET_KEY,
        algorithm="HS256",
    )
    return encoded_jwt

```

```

def verify_password_reset_token(token: str) -> str | None:
    try:
        decoded_token = jwt.decode(token, settings.SECRET_KEY, algorithms=["HS256"])
        return str(decoded_token["sub"])
    except InvalidTokenError:
        return None
...

```

```

```py
...

```

```

```py
from collections.abc import Generator
from typing import Annotated

import jwt

```

```

from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer
from jwt.exceptions import InvalidTokenError
from pydantic import ValidationError
from sqlmodel import Session

from app.core import security
from app.core.config import settings
from app.core.db import engine
from app.models import TokenPayload, User

reusable_oauth2 = OAuth2PasswordBearer(
    tokenUrl=f"{settings.API_V1_STR}/login/access-token"
)

def get_db() -> Generator[Session, None, None]:
    with Session(engine) as session:
        yield session

SessionDep = Annotated[Session, Depends(get_db)]
TokenDep = Annotated[str, Depends(reusable_oauth2)]

def get_current_user(session: SessionDep, token: TokenDep) -> User:
    try:
        payload = jwt.decode(
            token, settings.SECRET_KEY, algorithms=[security.ALGORITHM]
        )
        token_data = TokenPayload(**payload)
    except (InvalidTokenError, ValidationError):
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Could not validate credentials",
        )
    user = session.get(User, token_data.sub)
    if not user:
        raise HTTPException(status_code=404, detail="User not found")
    if not user.is_active:
        raise HTTPException(status_code=400, detail="Inactive user")
    return user

```

```
CurrentUser = Annotated[User, Depends(get_current_user)]
```

```
def get_current_active_superuser(current_user: CurrentUser) -> User:
    if not current_user.is_superuser:
        raise HTTPException(
            status_code=403, detail="The user doesn't have enough privileges"
        )
    return current_user
...

```

```
...py

```

```
from app.api.routes.analytics import prediction
from fastapi import APIRouter

```

```
from app.api.routes import items, login, messages, users, utils, analytics, reports

```

```
api_router = APIRouter()
api_router.include_router(login.router, tags=["login"])
api_router.include_router(users.router, prefix="/users", tags=["users"])
api_router.include_router(utils.router, prefix="/utils", tags=["utils"])
api_router.include_router(items.router, prefix="/items", tags=["items"])
api_router.include_router(messages.router, prefix="/messages", tags=["messages"])
api_router.include_router(analytics.router, prefix="/analytics", tags=["analytics"])
api_router.include_router(reports.router, prefix="/reports", tags=["reports"])
...

```

```
...py
...

```

```
...py

```

```
from fastapi import APIRouter

```

```
from app.api.routes.analytics_route import prediction, trends, comparison

```

```
router = APIRouter()
router.include_router(prediction.router, tags=["prediction"])
router.include_router(trends.router, tags=["trends"])
router.include_router(comparison.router, tags=["comparison"])``

```

```

```py
import uuid
from typing import Any

from fastapi import APIRouter, HTTPException
from sqlmodel import func, select

from app.api.deps import CurrentUser, SessionDep
from app.models import Item, ItemCreate, ItemPublic, ItemsPublic, ItemUpdate, Message

router = APIRouter()

@router.get("/", response_model=ItemsPublic)
def read_items(
 session: SessionDep, current_user: CurrentUser, skip: int = 0, limit: int = 100
) -> Any:
 """
 Retrieve items.
 """

 if current_user.is_superuser:
 count_statement = select(func.count()).select_from(Item)
 count = session.exec(count_statement).one()
 statement = select(Item).offset(skip).limit(limit)
 items = session.exec(statement).all()
 else:
 count_statement = (
 select(func.count())
 .select_from(Item)
 .where(Item.owner_id == current_user.id)
)
 count = session.exec(count_statement).one()
 statement = (
 select(Item)
 .where(Item.owner_id == current_user.id)
 .offset(skip)
 .limit(limit)
)
 items = session.exec(statement).all()

 return ItemsPublic(data=items, count=count)

```

```

@router.get("/{id}", response_model=ItemPublic)
def read_item(session: SessionDep, current_user: CurrentUser, id: uuid.UUID) -> Any:
 """
 Get item by ID.
 """
 item = session.get(Item, id)
 if not item:
 raise HTTPException(status_code=404, detail="Item not found")
 if not current_user.is_superuser and (item.owner_id != current_user.id):
 raise HTTPException(status_code=400, detail="Not enough permissions")
 return item

```

```

@router.post("/", response_model=ItemPublic)
def create_item(
 *, session: SessionDep, current_user: CurrentUser, item_in: ItemCreate
) -> Any:
 """
 Create new item.
 """
 item = Item.model_validate(item_in, update={"owner_id": current_user.id})
 session.add(item)
 session.commit()
 session.refresh(item)
 return item

```

```

@router.put("/{id}", response_model=ItemPublic)
def update_item(
 *,
 session: SessionDep,
 current_user: CurrentUser,
 id: uuid.UUID,
 item_in: ItemUpdate,
) -> Any:
 """
 Update an item.
 """
 item = session.get(Item, id)
 if not item:

```

```

 raise HTTPException(status_code=404, detail="Item not found")
 if not current_user.is_superuser and (item.owner_id != current_user.id):
 raise HTTPException(status_code=400, detail="Not enough permissions")
 update_dict = item_in.model_dump(exclude_unset=True)
 item.sqlmodel_update(update_dict)
 session.add(item)
 session.commit()
 session.refresh(item)
 return item

```

```

@router.delete("/{id}")
def delete_item(
 session: SessionDep, current_user: CurrentUser, id: uuid.UUID
) -> Message:
 """
 Delete an item.
 """
 item = session.get(Item, id)
 if not item:
 raise HTTPException(status_code=404, detail="Item not found")
 if not current_user.is_superuser and (item.owner_id != current_user.id):
 raise HTTPException(status_code=400, detail="Not enough permissions")
 session.delete(item)
 session.commit()
 return Message(message="Item deleted successfully")

```

```

```py
from datetime import timedelta
from typing import Annotated, Any

from fastapi import APIRouter, Depends, HTTPException
from fastapi.responses import HTMLResponse
from fastapi.security import OAuth2PasswordRequestForm

from app import crud
from app.api.deps import CurrentUser, SessionDep, get_current_active_superuser
from app.core import security
from app.core.config import settings
from app.core.security import get_password_hash
from app.models import Message, NewPassword, Token, UserPublic

```

```

from app.utils import (
    generate_password_reset_token,
    generate_reset_password_email,
    send_email,
    verify_password_reset_token,
)

router = APIRouter()

@router.post("/login/access-token")
def login_access_token(
    session: SessionDep, form_data: Annotated[OAuth2PasswordRequestForm, Depends()]
) -> Token:
    """
    OAuth2 compatible token login, get an access token for future requests
    """
    user = crud.authenticate(
        session=session, email=form_data.username, password=form_data.password
    )
    if not user:
        raise HTTPException(status_code=400, detail="Incorrect email or password")
    elif not user.is_active:
        raise HTTPException(status_code=400, detail="Inactive user")
    access_token_expires = timedelta(minutes=settings.ACCESS_TOKEN_EXPIRE_MINUTES)
    return Token(
        access_token=security.create_access_token(
            user.id, expires_delta=access_token_expires
        )
    )

@router.post("/login/test-token", response_model=UserPublic)
def test_token(current_user: CurrentUser) -> Any:
    """
    Test access token
    """
    return current_user

@router.post("/password-recovery/{email}")
def recover_password(email: str, session: SessionDep) -> Message:

```

```

"""
Password Recovery
"""

user = crud.get_user_by_email(session=session, email=email)

if not user:
    raise HTTPException(
        status_code=404,
        detail="The user with this email does not exist in the system.",
    )
password_reset_token = generate_password_reset_token(email=email)
email_data = generate_reset_password_email(
    email_to=user.email, email=email, token=password_reset_token
)
send_email(
    email_to=user.email,
    subject=email_data.subject,
    html_content=email_data.html_content,
)
return Message(message="Password recovery email sent")

@router.post("/reset-password/")
def reset_password(session: SessionDep, body: NewPassword) -> Message:
    """
    Reset password
    """
    email = verify_password_reset_token(token=body.token)
    if not email:
        raise HTTPException(status_code=400, detail="Invalid token")
    user = crud.get_user_by_email(session=session, email=email)
    if not user:
        raise HTTPException(
            status_code=404,
            detail="The user with this email does not exist in the system.",
        )
    elif not user.is_active:
        raise HTTPException(status_code=400, detail="Inactive user")
    hashed_password = get_password_hash(password=body.new_password)
    user.hashed_password = hashed_password
    session.add(user)
    session.commit()

```

```

return Message(message="Password updated successfully")

@router.post(
    "/password-recovery-html-content/{email}",
    dependencies=[Depends(get_current_active_superuser)],
    response_class=HTMLResponse,
)
def recover_password_html_content(email: str, session: SessionDep) -> Any:
    """
    HTML Content for Password Recovery
    """
    user = crud.get_user_by_email(session=session, email=email)

    if not user:
        raise HTTPException(
            status_code=404,
            detail="The user with this username does not exist in the system.",
        )
    password_reset_token = generate_password_reset_token(email=email)
    email_data = generate_reset_password_email(
        email_to=user.email, email=email, token=password_reset_token
    )

    return HTMLResponse(
        content=email_data.html_content, headers={"subject": email_data.subject}
    )
"""

"""py
import asyncio
from typing import List

from fastapi import APIRouter, HTTPException

from app.api.deps import CurrentUser, SessionDep
from app.core.client_messages import Client, clients
from app.models import MessageResponse

router = APIRouter()

```

```

@router.get("/new_messages", response_model=list[MessageResponse])
def get_messages(session: SessionDep, current_user: CurrentUser) -> list[MessageResponse]:
    client_id = current_user.id
    if client_id in clients:
        client = clients[client_id]
    else:
        client = Client(client_id)
        clients[client_id] = client

    messages = client.get_messages()

    if not isinstance(messages, list):
        raise HTTPException(status_code=500, detail="Invalid response from Client.get_messages")

    return messages
'''

'''py
from fastapi import APIRouter

from app.api.routes.reports_route import automatic, report

router = APIRouter()
router.include_router(automatic.router, tags=["automatic"])
router.include_router(report.router, tags=["report"])

'''

'''py
import uuid
from typing import Any

from fastapi import APIRouter, Depends, HTTPException
from sqlmodel import col, delete, func, select

from app import crud
from app.api.deps import (
    CurrentUser,
    SessionDep,
    get_current_active_superuser,
)
from app.core.config import settings

```

```

from app.core.security import get_password_hash, verify_password
from app.models import (
    Item,
    Message,
    UpdatePassword,
    User,
    UserCreate,
    UserPublic,
    UserRegister,
    UsersPublic,
    UserUpdate,
    UserUpdateMe,
)
from app.utils import generate_new_account_email, send_email

router = APIRouter()

@router.get(
    "/",
    dependencies=[Depends(get_current_active_superuser)],
    response_model=UsersPublic,
)
def read_users(session: SessionDep, skip: int = 0, limit: int = 100) -> Any:
    """
    Retrieve users.
    """

    count_statement = select(func.count()).select_from(User)
    count = session.exec(count_statement).one()

    statement = select(User).offset(skip).limit(limit)
    users = session.exec(statement).all()

    return UsersPublic(data=users, count=count)

@router.post(
    "/", dependencies=[Depends(get_current_active_superuser)], response_model=UserPublic
)
def create_user(*, session: SessionDep, user_in: UserCreate) -> Any:
    """

```

Create new user.

"""

```
user = crud.get_user_by_email(session=session, email=user_in.email)
```

```
if user:
```

```
    raise HTTPException(
        status_code=400,
        detail="The user with this email already exists in the system.",
    )
```

```
user = crud.create_user(session=session, user_create=user_in)
```

```
if settings.emails_enabled and user_in.email:
```

```
    email_data = generate_new_account_email(
        email_to=user_in.email, username=user_in.email, password=user_in.password
    )
    send_email(
        email_to=user_in.email,
        subject=email_data.subject,
        html_content=email_data.html_content,
    )
```

```
return user
```

```
@router.patch("/me", response_model=UserPublic)
```

```
def update_user_me(
```

```
    *, session: SessionDep, user_in: UserUpdateMe, current_user: CurrentUser
```

```
) -> Any:
```

"""

Update own user.

"""

```
if user_in.email:
```

```
    existing_user = crud.get_user_by_email(session=session, email=user_in.email)
```

```
    if existing_user and existing_user.id != current_user.id:
```

```
        raise HTTPException(
            status_code=409, detail="User with this email already exists"
        )
```

```
user_data = user_in.model_dump(exclude_unset=True)
```

```
current_user.sqlmodel_update(user_data)
```

```
session.add(current_user)
```

```
session.commit()
```

```
session.refresh(current_user)
```

```
return current_user
```

```

@router.patch("/me/password", response_model=Message)
def update_password_me(
    *, session: SessionDep, body: UpdatePassword, current_user: CurrentUser
) -> Any:
    """
    Update own password.
    """
    if not verify_password(body.current_password, current_user.hashed_password):
        raise HTTPException(status_code=400, detail="Incorrect password")
    if body.current_password == body.new_password:
        raise HTTPException(
            status_code=400, detail="New password cannot be the same as the current one"
        )
    hashed_password = get_password_hash(body.new_password)
    current_user.hashed_password = hashed_password
    session.add(current_user)
    session.commit()
    return Message(message="Password updated successfully")

```

```

@router.get("/me", response_model=UserPublic)
def read_user_me(current_user: CurrentUser) -> Any:
    """
    Get current user.
    """
    return current_user

```

```

@router.delete("/me", response_model=Message)
def delete_user_me(session: SessionDep, current_user: CurrentUser) -> Any:
    """
    Delete own user.
    """
    if current_user.is_superuser:
        raise HTTPException(
            status_code=403, detail="Super users are not allowed to delete themselves"
        )
    statement = delete(Item).where(col(Item.owner_id) == current_user.id)
    session.exec(statement) # type: ignore
    session.delete(current_user)

```

```

session.commit()
return Message(message="User deleted successfully")

```

```

@router.post("/signup", response_model=UserPublic)
def register_user(session: SessionDep, user_in: UserRegister) -> Any:
    """
    Create new user without the need to be logged in.
    """
    user = crud.get_user_by_email(session=session, email=user_in.email)
    if user:
        raise HTTPException(
            status_code=400,
            detail="The user with this email already exists in the system",
        )
    user_create = UserCreate.model_validate(user_in)
    user = crud.create_user(session=session, user_create=user_create)
    return user

```

```

@router.get("/{user_id}", response_model=UserPublic)
def read_user_by_id(
    user_id: uuid.UUID, session: SessionDep, current_user: CurrentUser
) -> Any:
    """
    Get a specific user by id.
    """
    user = session.get(User, user_id)
    if user == current_user:
        return user
    if not current_user.is_superuser:
        raise HTTPException(
            status_code=403,
            detail="The user doesn't have enough privileges",
        )
    return user

```

```

@router.patch(
   ("/{user_id}",
    dependencies=[Depends(get_current_active_superuser)],
    response_model=UserPublic,

```

```

)
def update_user(
    *,
    session: SessionDep,
    user_id: uuid.UUID,
    user_in: UserUpdate,
) -> Any:
    """
    Update a user.
    """

    db_user = session.get(User, user_id)
    if not db_user:
        raise HTTPException(
            status_code=404,
            detail="The user with this id does not exist in the system",
        )
    if user_in.email:
        existing_user = crud.get_user_by_email(session=session, email=user_in.email)
        if existing_user and existing_user.id != user_id:
            raise HTTPException(
                status_code=409, detail="User with this email already exists"
            )

    db_user = crud.update_user(session=session, db_user=db_user, user_in=user_in)
    return db_user

@router.delete("/{user_id}", dependencies=[Depends(get_current_active_superuser)])
def delete_user(
    session: SessionDep, current_user: CurrentUser, user_id: uuid.UUID
) -> Message:
    """
    Delete a user.
    """

    user = session.get(User, user_id)
    if not user:
        raise HTTPException(status_code=404, detail="User not found")
    if user == current_user:
        raise HTTPException(
            status_code=403, detail="Super users are not allowed to delete themselves"
        )

```

```

statement = delete(Item).where(col(Item.owner_id) == user_id)
session.exec(statement) # type: ignore
session.delete(user)
session.commit()
return Message(message="User deleted successfully")

```

```

"""
.py
from fastapi import APIRouter, Depends
from pydantic.networks import EmailStr

from app.api.deps import get_current_active_superuser
from app.models import Message
from app.utils import generate_test_email, send_email

router = APIRouter()

@router.post(
    "/test-email/",
    dependencies=[Depends(get_current_active_superuser)],
    status_code=201,
)
def test_email(email_to: EmailStr) -> Message:
    """
    Test emails.
    """
    email_data = generate_test_email(email_to=email_to)
    send_email(
        email_to=email_to,
        subject=email_data.subject,
        html_content=email_data.html_content,
    )
    return Message(message="Test email sent")

```

```

"""
.py

```

```

"""
.py
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.ext.asyncio import AsyncSession

```

```

from sqlmodel import select, func
from typing import List, Optional
from datetime import date

from app.api.deps import CurrentUser, SessionDep
from app.models import Item, TrendData, TrendsResponse, User
from app.core.client_messages import Client, add_message, clients

router = APIRouter()

```

```

@router.get("/comparison", response_model=TrendsResponse)
def comparison( session: SessionDep, current_user: CurrentUser):
    """

```

Порівнює дані споживання поточного користувача з середніми показниками інших користувачів.

Args:

user: Поточний користувач.

db: Залежність для отримання сесії бази даних.

region: Опціональний регіон для порівняння. Якщо вказано, порівняння буде з користувачами з цього регіону.

Returns:

TrendsResponse: Об'єкт з даними для порівняльного аналізу.

Raises:

HTTPException: Якщо не знайдено даних споживання для користувача або інших користувачів.

"""

1. Отримуємо дані поточного користувача

```

user_items_query = (
    select(Item.reading_date, Item.current_amount, Item.kwh_tariff)
    .where(Item.owner_id == current_user.id)
    .order_by(Item.reading_date)
)
user_items_result = session.execute(user_items_query)
user_items_data = user_items_result.all()

if not user_items_data:
    raise HTTPException(
        status_code=404, detail="No consumption data found for the user"
    )

```

2. Отримуємо дані інших користувачів

```
other_users_query = select(
    func.date(Item.reading_date).label("reading_date"),
    func.avg(Item.current_amount).label("avg_current_amount"),
    func.avg(Item.kwh_tariff).label("avg_kwh_tariff"),
).where(Item.owner_id != current_user.id)
```

```
other_users_query = other_users_query.group_by(func.date(Item.reading_date))
other_users_result = session.execute(other_users_query)
other_users_data = other_users_result.all()
```

```
if not other_users_data:
    raise HTTPException(
        status_code=404,
        detail="No consumption data found for other users in the specified region"
    )
```

3. Створюємо словник з даними інших користувачів для швидкого доступу

```
other_users_dict = {
    row.reading_date: {
        "avg_current_amount": row.avg_current_amount,
        "avg_kwh_tariff": row.avg_kwh_tariff,
    }
    for row in other_users_data
}
```

4. Порівнюємо дані та формуємо результат

```
trend_data = []
for user_row in user_items_data:
    reading_date = user_row.reading_date.date()
    other_user_data = other_users_dict.get(reading_date)

    if other_user_data:
        amount_trend = user_row.current_amount - other_user_data["avg_current_amount"]
        tariff_trend = user_row.kwh_tariff - other_user_data["avg_kwh_tariff"]

    trend_data.append(
        TrendData(
            reading_date=user_row.reading_date,
            current_amount=user_row.current_amount,
            amount_trend=amount_trend,
            kwh_tariff=user_row.kwh_tariff,
```

```

        tariff_trend=tariff_trend,
    )
)

return TrendsResponse(data=trend_data)

'''

'''py
import asyncio
import uuid
from datetime import datetime, timedelta, timezone
from uuid import UUID

from fastapi import APIRouter, BackgroundTasks, HTTPException
from sqlalchemy import delete, desc
from sqlmodel import func, select

from app.api.deps import CurrentUser, SessionDep
from app.core.client_messages import Client, add_message, clients
from app.forecast import Predict, predict_model
from app.models import Forecast, ForecastsResponse, Item, MessageResponse, User

router = APIRouter()

MAX_CONCURRENT_TASKS = 5
semaphore = asyncio.Semaphore(MAX_CONCURRENT_TASKS)

def delete_forecast(
    session: SessionDep,
    user_id: uuid.UUID,
    chart_name: str,
    model_name: str
):
    """
    Видаляє передбачення для заданого користувача, графіка та моделі.

    :param session: Активна сесія SQLAlchemy
    :param user_id: ID користувача (uuid.UUID)
    :param chart_name: Назва графіка
    :param model_name: Назва моделі, яка створила передбачення
    """

```

```

delete_query = delete(Forecast).where(
    Forecast.user_id == user_id,
    Forecast.chart_name == chart_name,
    Forecast.name_nn_model == model_name
)
session.exec(delete_query)
session.commit()

def is_forecast_needed(
    session: SessionDep,
    user_id: uuid.UUID,
    chart_name: str="reading_values",
    model_name: str="chronos",
    interval_days: int = 7
) -> bool:
    """
    Перевіряє, чи потрібно створювати нове передбачення, базуючись на інтервалі.

    :param session: Активна сесія SQLAlchemy
    :param user_id: ID користувача (uuid.UUID)
    :param chart_name: Назва графіка
    :param model_name: Назва моделі
    :param interval_days: Інтервал в днях, за замовчуванням 7
    :return: True, якщо потрібно створити нове передбачення, False інакше
    """
    query = select(Forecast).where(
        Forecast.user_id == user_id,
        Forecast.chart_name == chart_name,
        Forecast.name_nn_model == model_name
    ).order_by(desc(Forecast.created_at)).limit(1)

    last_forecast = session.exec(query).first()

    if not last_forecast:
        # Якщо передбачення ще не існує, створюємо нове
        return True

    # Перевіряємо, чи минув заданий інтервал
    time_since_last = datetime.now() - last_forecast.created_at
    return time_since_last > timedelta(days=interval_days)

```

```
def get_forecasts(
    session: SessionDep,
    user_id: uuid.UUID,
    chart_name: str,
    model_name: str,
    interval_days: int = 7,
    limit: int = 1 # Ліміт на кількість передбачень
) -> list[Forecast]:
    """
```

Отримує список передбачень для користувача, графіка та моделі, які відповідають заданому інтервалу.

Обмежує кількість отриманих передбачень лімітом.

```
:param session: Активна сесія SQLAlchemy
:param user_id: ID користувача (uuid.UUID)
:param chart_name: Назва графіка
:param model_name: Назва моделі
:param interval_days: Інтервал в днях, за замовчуванням 7
:param limit: Ліміт на кількість передбачень, за замовчуванням 1
:return: Список передбачень
"""
```

```
query = select(Forecast).where(
    Forecast.user_id == user_id,
    Forecast.chart_name == chart_name,
    Forecast.name_nn_model == model_name
).order_by(desc(Forecast.created_at)).limit(limit)
```

```
forecasts = session.exec(query).first()
```

```
# Фільтруємо передбачення за інтервалом часу
time_threshold = datetime.now(timezone.utc) - timedelta(days=interval_days)
filtered_forecasts = [forecast for forecast in forecasts if forecast.created_at > time_threshold]

return filtered_forecasts
```

```
def get_all_forecasts_by_chart(
    session: SessionDep,
    user_id: UUID,
    model_name: str,
    interval_days: int = 7
) -> dict[str, Forecast]:
    """
```

Отримує останнє передбачення для кожного графіка для заданого користувача та моделі.

```

:param session: Активна сесія SQLAlchemy
:param user_id: ID користувача (UUID)
:param model_name: Назва моделі
:param interval_days: Інтервал в днях, за замовчуванням 7
:return: Словник з графіками як ключами та останніми передбаченнями як значеннями
"""

query = select(Forecast).where(
    Forecast.user_id == user_id,
    Forecast.name_nn_model == model_name
).order_by(desc(Forecast.created_at))

forecasts = session.exec(query).all()

# Фільтруємо передбачення за інтервалом часу
time_threshold = datetime.now() - timedelta(days=interval_days)
filtered_forecasts = [forecast for forecast in forecasts if forecast.created_at > time_threshold]

# Групуємо передбачення за графіками
forecasts_by_chart = {}
for forecast in filtered_forecasts:
    chart_name = forecast.chart_name
    if chart_name not in forecasts_by_chart:
        forecasts_by_chart[chart_name] = forecast

return forecasts_by_chart

def get_all_forecasts(
    session: SessionDep,
    current_user: CurrentUser,
    model_name: str = "chronos",
    interval_days: int = 7
) -> ForecastsResponse:
    """
    Отримує останнє передбачення для кожного графіка для заданого користувача та моделі.

    :param session: Активна сесія SQLAlchemy
    :param current_user: Поточний користувач
    :param model_name: Назва моделі
    :param interval_days: Інтервал в днях, за замовчуванням Forecast]
    """

```

```

forecasts_by_chart = get_all_forecasts_by_chart(
    session=session,
    user_id=current_user.id,
    model_name=model_name,
    interval_days=interval_days
)

# Формуємо відповідь з окремими полями
response = ForecastsResponse(
    reading_values=forecasts_by_chart.get("reading_values"),
    kwh_tariffs=forecasts_by_chart.get("kwh_tariffs"),
    current_amounts=forecasts_by_chart.get("current_amounts"),
    deltas=forecasts_by_chart.get("deltas"),
    costs=forecasts_by_chart.get("costs")
)

return response

def test_prediction_length(values):
    if len(values) > 10:
        return 5
    else:
        return 3

async def chronos_task(session: SessionDep, client_id_uuid:UUID):
    async with semaphore:
        await asyncio.sleep(2)
        #current_reading_value_graph || lineData_reading_value || Графік поточного показника
        #kwh_tariff_graph || lineData_kwh_tariff || Графік тарифу
        #monthly_amount_graph || lineData_current_amount || Графік суми за місяць
        #per_month_delta_graph || lineData_per_m || Графік дельт між місяцями
        #monthly_cost_bar_graph || Вартість за місяць
        query = select(User).where(User.id == client_id_uuid)
        user = session.exec(query).first()
        if user is not None:
            query = (
                select(Item)
                .where(Item.owner_id == client_id_uuid)
                .order_by(desc(Item.reading_date)) # Сортуюмо за датою в спадному порядку
                .limit(100) # Обмежуємо 100 записами
            )
            items = session.exec(query).all()

```

```

if not items:
    raise HTTPException(status_code=404, detail="No items found")

items = list(reversed(items))

# Сортуємо записи за датою
items.sort(key=lambda x: x.reading_date)

# Масиви даних для графіків
labels = [
    item.reading_date.strftime("%B") for item in items
] # Місяці для осі X
reading_values = [
    item.reading_value for item in items
] # Поточні показники
kwh_tariffs = [
    item.kwh_tariff for item in items
] # Тариф за кіловат
current_amounts = [
    item.current_amount for item in items
] # Сума за місяць

# Обчислюємо дельти між місяцями
deltas = [
    items[i].reading_value - items[i - 1].reading_value
    for i in range(1, len(items))
] if len(items) > 1 else []
delta_labels = labels[1:] # Зсуваємо мітки для дельт

# Обчислюємо вартість за місяць
costs = [
    (items[i].reading_value - items[i - 1].reading_value) * items[i - 1].kwh_tariff
    for i in range(1, len(items))
] if len(items) > 1 else []

# Формуємо дані для фронтенду
data = {
    "labels": labels,
    "reading_values": reading_values,
    "kwh_tariffs": kwh_tariffs,
    "current_amounts": current_amounts,

```

```

    "delta_labels": delta_labels,
    "deltas": deltas,
    "costs": costs,
}

```

```

if is_forecast_needed(session, user.id, "reading_values", "chronos"):
    delete_forecast(session, user.id, "reading_values", "chronos")
    low, median, high, time_steps, forecast_steps = predict_model.predict(reading_values,
test_prediction_length(reading_values))
    low = low.tolist()
    median = median.tolist()
    high = high.tolist()
    reading_values_forecast = Forecast(
        real_data={"data":reading_values},
        name_nn_model="chronos",
        chart_name="reading_values",
        prediction_low={"data":low},
        prediction_median={"data":median},
        prediction_high={"data":high},
        prediction_time_steps={"data":time_steps},
        prediction_forecast_steps={"data":forecast_steps},
        user_id=user.id,
    )
    session.add(reading_values_forecast)
    session.commit()

```

```

if is_forecast_needed(session, user.id, "kwh_tariffs", "chronos"):
    delete_forecast(session, user.id, "kwh_tariffs", "chronos")
    low, median, high, time_steps, forecast_steps = predict_model.predict(kwh_tariffs,
test_prediction_length(kwh_tariffs))
    low = low.tolist()
    median = median.tolist()
    high = high.tolist()
    kwh_tariffs_forecast = Forecast(
        real_data={"data":kwh_tariffs},
        name_nn_model="chronos",
        chart_name="kwh_tariffs",
        prediction_low={"data":low},
        prediction_median={"data":median},
        prediction_high={"data":high},
        prediction_time_steps={"data":time_steps},

```

```

        prediction_forecast_steps={"data":forecast_steps},
        user_id=user.id,
    )
    session.add(kwh_tariffs_forecast)
    session.commit()
    add_message("test", client_id_uuid)

if is_forecast_needed(session, user.id, "current_amounts", "chronos"):
    delete_forecast(session, user.id, "current_amounts", "chronos")
    low, median, high, time_steps, forecast_steps = predict_model.predict(current_amounts,
test_prediction_length(current_amounts))
    low = low.tolist()
    median = median.tolist()
    high = high.tolist()
    current_amounts_forecast = Forecast(
        real_data={"data":current_amounts},
        name_nn_model="chronos",
        chart_name="current_amounts",
        prediction_low={"data":low},
        prediction_median={"data":median},
        prediction_high={"data":high},
        prediction_time_steps={"data":time_steps},
        prediction_forecast_steps={"data":forecast_steps},
        user_id=user.id,
    )
    session.add(current_amounts_forecast)
    session.commit()
    add_message("test", client_id_uuid)

if is_forecast_needed(session, user.id, "deltas", "chronos"):
    delete_forecast(session, user.id, "deltas", "chronos")
    low, median, high, time_steps, forecast_steps = predict_model.predict(deltas,
test_prediction_length(deltas))
    low = low.tolist()
    median = median.tolist()
    high = high.tolist()
    deltas_forecast = Forecast(
        real_data={"data":deltas},
        name_nn_model="chronos",
        chart_name="deltas",
        prediction_low={"data":low},
        prediction_median={"data":median},

```

```

        prediction_high={"data":high},
        prediction_time_steps={"data":time_steps},
        prediction_forecast_steps={"data":forecast_steps},
        user_id=user.id,
    )
    session.add(deltas_forecast)
    session.commit()
    add_message("test", client_id_uuid)

    if is_forecast_needed(session, user.id, "costs", "chronos"):
        delete_forecast(session, user.id, "costs", "chronos")
        low, median, high, time_steps, forecast_steps = predict_model.predict(costs,
test_prediction_length(costs))
        low = low.tolist()
        median = median.tolist()
        high = high.tolist()
        costs_forecast = Forecast(
            real_data={"data":costs},
            name_nn_model="chronos",
            chart_name="costs",
            prediction_low={"data":low},
            prediction_median={"data":median},
            prediction_high={"data":high},
            prediction_time_steps={"data":time_steps},
            prediction_forecast_steps={"data":forecast_steps},
            user_id=user.id,
        )
        session.add(costs_forecast)
        session.commit()
        add_message("Прогноз виконано", client_id_uuid)

    @router.post("/prediction", response_model=ForecastsResponse)
    async def start_task(session: SessionDep, current_user: CurrentUser, background_tasks: BackgroundTasks)
-> str:
        if is_forecast_needed(session=session, user_id=current_user.id):
            background_tasks.add_task(chronos_task, session, str(current_user.id))
            add_message("Задача chronos додана до черги", current_user.id)
            return ForecastsResponse(run_task=True)
        else:
            return get_all_forecasts(session=session, current_user=current_user)

```

```

...

```py
import asyncio
import uuid
from datetime import datetime, timedelta, timezone
from uuid import UUID

from fastapi import APIRouter, BackgroundTasks, HTTPException
from sqlalchemy import delete, desc
from sqlmodel import func, select, Field, Session, SQLModel

from app.api.deps import CurrentUser, SessionDep, get_db
from app.core.client_messages import Client, add_message, clients
from app.forecast import Predict, predict_model
from app.models import Forecast, ForecastsResponse, Item, MessageResponse, User, TrendData,
TrendsResponse
import pandas as pd
from typing import List

router = APIRouter()

Тестові дані (можна замінити на запит до бази даних)
sample_data = [
 {"reading_date": "2023-01-01", "kwh_tariff": 1.5, "current_amount": 50},
 {"reading_date": "2023-02-01", "kwh_tariff": 1.6, "current_amount": 60},
 {"reading_date": "2023-03-01", "kwh_tariff": 1.7, "current_amount": 70},
 {"reading_date": "2023-04-01", "kwh_tariff": 1.8, "current_amount": 80},
 {"reading_date": "2023-05-01", "kwh_tariff": 1.8, "current_amount": 90},
 {"reading_date": "2023-06-01", "kwh_tariff": 1.9, "current_amount": 95},
]

@router.get("/trends", response_model=TrendsResponse)
def get_trends(session: SessionDep, current_user: CurrentUser):
 # Завантаження даних
 df = pd.DataFrame(sample_data)
 df['reading_date'] = pd.to_datetime(df['reading_date'])
 df = df.sort_values(by='reading_date')

 # Розрахунок тренду для current_amount
 df['amount_trend'] = df['current_amount'].rolling(window=3, min_periods=1).mean()

```

```

Розрахунок тренду для kwh_tariff
df['tariff_trend'] = df['kwh_tariff'].rolling(window=3, min_periods=1).mean()

Перетворення DataFrame в список TrendData
trend_data_list = []
for index, row in df.iterrows():
 trend_data_list.append(
 TrendData(
 reading_date=row['reading_date'],
 current_amount=row['current_amount'],
 amount_trend=row['amount_trend'],
 kwh_tariff=row['kwh_tariff'],
 tariff_trend=row['tariff_trend']
)
)

return TrendsResponse(data=trend_data_list)``

``.py
import asyncio
import os
import uuid
from datetime import datetime, timedelta, timezone
from typing import List
from uuid import UUID

import pandas as pd
from fastapi import APIRouter, BackgroundTasks, HTTPException
from fastapi.responses import FileResponse
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
from sqlalchemy import delete, desc
from sqlmodel import Field, Session, SQLModel, func, select

from app.api.deps import CurrentUser, SessionDep, get_db
from app.core.client_messages import Client, add_message, clients
from app.forecast import Predict, predict_model
from app.models import (
 Forecast,
 ForecastsResponse,
 Item,

```

```

 MessageResponse,
 TrendData,
 TrendsResponse,
 User,
 UserReportSettings,
 UserReportSettingsCreate,
)

router = APIRouter()

@router.post("/set-report-settings")
def set_user_report_settings(settings: UserReportSettingsCreate, session: SessionDep, current_user:
CurrentUser):
 user_id = current_user.id
 # Перевірка, чи є користувач
 user = session.exec(select(User).where(User.id == user_id)).first()
 if not user:
 raise HTTPException(status_code=404, detail="User not found")

 # Перевірка чи вже є налаштування звітів для цього користувача
 user_settings = session.exec(select(UserReportSettings).where(UserReportSettings.user_id ==
user_id)).first()

 if user_settings:
 # Оновлюємо налаштування
 user_settings.report_enabled = settings.report_enabled
 user_settings.report_period = settings.report_period
 user_settings.last_reported_at = datetime.now() # Оновлюємо час останнього звіту
 else:
 # Створюємо нові налаштування
 user_settings = UserReportSettings(
 user_id=user_id,
 report_enabled=settings.report_enabled,
 report_period=settings.report_period,
 last_reported_at=datetime.now()
)
 session.add(user_settings)

 session.commit()
 session.refresh(user_settings)
 return {"message": "User report settings updated", "settings": user_settings}

```

```

@router.get("/get-report-settings")
def get_user_report_settings(session: SessionDep, current_user: CurrentUser):
 user_id = current_user.id
 user_settings = session.exec(select(UserReportSettings).where(UserReportSettings.user_id ==
user_id)).first()
 if not user_settings:
 raise HTTPException(status_code=404, detail="User report settings not found")
 return user_settings

...

```py
import os
import uuid

from fastapi import APIRouter, FastAPI, HTTPException, Response
from fastapi.responses import FileResponse
from pydantic import BaseModel
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas

from app.api.deps import CurrentUser, SessionDep
from app.core.generate_and_send_reports import generate_report as generate_report_file
from app.models import ReportParams, ReportResponse

router = APIRouter()

@router.post("/generate_report", response_model=ReportResponse)
def generate_report(params: ReportParams, session: SessionDep, current_user: CurrentUser, ):
    report_id = str(uuid.uuid4()) # Генеруємо унікальний ID
    file_path = f"reports/report_{report_id}.pdf" # Використовуємо ID у назві файлу

    # Перевірка та створення директорії reports, якщо її немає
    os.makedirs("reports", exist_ok=True)

    # Створення PDF файлу
    c = canvas.Canvas(file_path, pagesize=letter)
    c.drawString(100, 750, f"Звіт по споживанню електроенергії (ID: {report_id})")
    c.drawString(100, 730, f"Початкова дата: {params.startDate}")
    c.drawString(100, 710, f"Кінцева дата: {params.endDate}")
    c.drawString(100, 690, f"Категорія: {params.category}")
    # Додати більше інформації до звіту, використовуючи params

```

```

c.save()

if os.path.exists(file_path):
    return ReportResponse(report_id=report_id)
else:
    raise HTTPException(status_code=404, detail="Report not found")

@router.get("/view_report/{report_id}")
async def view_report(report_id: str):
    file_path = f"reports/report_{report_id}.pdf"
    if not os.path.exists(file_path):
        raise HTTPException(status_code=404, detail="Report not found")

    with open(file_path, "rb") as f:
        pdf_content = f.read()

    return Response(content=pdf_content, media_type="application/pdf", headers={"Content-Disposition":
f"filename=report_{report_id}.pdf"})

'''

'''py
import os
from datetime import datetime, timedelta

from apscheduler.jobstores.sqlalchemy import SQLAlchemyJobStore
from apscheduler.schedulers.asyncio import AsyncIOScheduler
from apscheduler.triggers.interval import IntervalTrigger
from sqlalchemy import select
from sqlmodel import Session

from app.api.deps import CurrentUser, SessionDep
from app.core.generate_and_send_reports import generate_report, send_report
from app.models import UserReportSettings
import logging

async def process_report(
    session: SessionDep,
    user_id: int,
    year: int,
    month: int | None,
    quarter: int | None,

```

```

email: str):
    """
    Обробляє звіт: генерує та відправляє.
    """

    print(f"[{datetime.now()}] Processing report for user {user_id} - Year: {year}, Month: {month},
Quarter: {quarter}, Email: {email}")

    file_path = generate_report(user_id, year, month, quarter)
    send_report(email, file_path)
    os.remove(file_path)

    # Оновлюємо час останньої відправки
    user_settings = session.exec(select(UserReportSettings).where(UserReportSettings.user_id ==
user_id)).first()
    if user_settings:
        user_settings.last_report_sent_at = datetime.now()
        session.add(user_settings)
        session.commit()
        session.refresh(user_settings)

async def check_and_process_reports(session: SessionDep):
    """
    Перевіряє налаштування звітів для всіх користувачів і запускає `process_report`, якщо потрібно.
    """

    print(f"[{datetime.now()}] Checking all users' report settings")
    now = datetime.now()
    try:
        result = session.exec(select(UserReportSettings).where(UserReportSettings.report_enabled == True))
        all_settings = result.scalars().all()
        for settings in all_settings:
            email = settings.report_email or settings.user.email
            if settings.report_period == "monthly":
                if settings.last_report_sent_at is None or (now - settings.last_report_sent_at) >=
timedelta(days=30):
                    if now.day == 1 and now.hour >= 9:
                        # Перевірка на нульову різницю часу
                        if settings.last_report_sent_at is not None and (now - settings.last_report_sent_at).days ==
0:
                            print(f"[{datetime.now()}] Monthly report for user {settings.user_id} was already sent
today")

                            continue
                        print(f"[{datetime.now()}] Time to send monthly report for user {settings.user_id}")
                        await process_report(settings.user_id, now.year, now.month, None, email, session)

```

```

elif settings.report_period == "quarterly":
    quarter = (now.month - 1) // 3 + 1
    if settings.last_report_sent_at is None or (now - settings.last_report_sent_at) >=
timedelta(days=90):
        if now.month in [1, 4, 7, 10] and now.day == 1 and now.hour >= 9:
            # Перевірка на нульову різницю часу
            if settings.last_report_sent_at is not None and (now - settings.last_report_sent_at).days ==
0:
                print(f"[{datetime.now()}] Quarterly report for user {settings.user_id} was already sent
today")

                continue
            print(f"[{datetime.now()}] Time to send quarterly report for user {settings.user_id}")
            await process_report(settings.user_id, now.year, None, quarter, email, session)
except Exception as e:
    print(f"[{datetime.now()}] Error during all users check: {e}")

def check_all_users_report_settings():
    print("call check_all_users_report_settings")
    logging.info("call check_all_users_report_settings")

...

```py
from uuid import UUID
from app.models import MessageResponse

class Client:
 def __init__(self, client_id_uuid: UUID):
 self.client_id_uuid = client_id_uuid
 self.list_messages = []

 def send_message(self, message: str):
 self.list_messages.append(MessageResponse(message=message))

 def get_messages(self):
 return self.list_messages

clients: dict[UUID, Client] = {}

def add_message(message_text:str, client_id_uuid:UUID):
 if client_id_uuid in clients:

```

```

 client = clients[client_id_uuid]
 else:
 client = Client(client_id_uuid)
 clients[client_id_uuid] = client
 client.send_message(message_text)

'''

'''py
import secrets
import warnings
from typing import Annotated, Any, Literal

from pydantic import (
 AnyUrl,
 BeforeValidator,
 HttpUrl,
 PostgresDsn,
 computed_field,
 model_validator,
)
from pydantic_core import MultiHostUrl
from pydantic_settings import BaseSettings, SettingsConfigDict
from typing_extensions import Self

def parse_cors(v: Any) -> list[str] | str:
 if isinstance(v, str) and not v.startswith("["):
 return [i.strip() for i in v.split(",")]
 elif isinstance(v, list | str):
 return v
 raise ValueError(v)

class Settings(BaseSettings):
 model_config = SettingsConfigDict(
 env_file=".env", env_ignore_empty=True, extra="ignore"
)
 API_V1_STR: str = "/api/v1"
 SECRET_KEY: str = secrets.token_urlsafe(32)
 # 60 minutes * 24 hours * 8 days = 8 days
 ACCESS_TOKEN_EXPIRE_MINUTES: int = 60 * 24 * 8

```

```

DOMAIN: str = "localhost"
ENVIRONMENT: Literal["local", "staging", "production"] = "local"

@computed_field # type: ignore[prop-decorator]
@property
def server_host(self) -> str:
 # Use HTTPS for anything other than local development
 if self.ENVIRONMENT == "local":
 return f"http://{self.DOMAIN}"
 return f"https://{self.DOMAIN}"

BACKEND_CORS_ORIGINS: Annotated[
 list[AnyUrl] | str, BeforeValidator(parse_cors)
] = []

PROJECT_NAME: str
SENTRY_DSN: HttpUrl | None = None
POSTGRES_SERVER: str
POSTGRES_PORT: int = 5432
POSTGRES_USER: str
POSTGRES_PASSWORD: str = "kkfIJDFKLd099373kkjhDjkod9"
POSTGRES_DB: str = "energy_monitoring"

@computed_field # type: ignore[prop-decorator]
@property
def SQLALCHEMY_DATABASE_URI(self) -> PostgresDsn:
 return MultiHostUrl.build(
 scheme="postgresql+psycopg2",
 username=self.POSTGRES_USER,
 password=self.POSTGRES_PASSWORD,
 host=self.POSTGRES_SERVER,
 port=self.POSTGRES_PORT,
 path=self.POSTGRES_DB,
)

SMTP_TLS: bool = True
SMTP_SSL: bool = False
SMTP_PORT: int = 587
SMTP_HOST: str | None = None
SMTP_USER: str | None = None
SMTP_PASSWORD: str | None = None
TODO: update type to EmailStr when sqlalchemy supports it

```

EMAILS\_FROM\_EMAIL: str | None = None

EMAILS\_FROM\_NAME: str | None = None

@model\_validator(mode="after")

def \_set\_default\_emails\_from(self) -> Self:

if not self.EMAILS\_FROM\_NAME:

self.EMAILS\_FROM\_NAME = self.PROJECT\_NAME

return self

EMAIL\_RESET\_TOKEN\_EXPIRE\_HOURS: int = 48

@computed\_field # type: ignore[prop-decorator]

@property

def emails\_enabled(self) -> bool:

return bool(self.SMTP\_HOST and self.EMAILS\_FROM\_EMAIL)

# TODO: update type to EmailStr when sqlmodel supports it

EMAIL\_TEST\_USER: str = "test@example.com"

# TODO: update type to EmailStr when sqlmodel supports it

FIRST\_SUPERUSER: str

FIRST\_SUPERUSER\_PASSWORD: str

def \_check\_default\_secret(self, var\_name: str, value: str | None) -> None:

if value == "changethis":

message = (

f'The value of {var\_name} is "changethis", '

"for security, please change it, at least for deployments."

)

if self.ENVIRONMENT == "local":

warnings.warn(message, stacklevel=1)

else:

raise ValueError(message)

@model\_validator(mode="after")

def \_enforce\_non\_default\_secrets(self) -> Self:

self.\_check\_default\_secret("SECRET\_KEY", self.SECRET\_KEY)

self.\_check\_default\_secret("POSTGRES\_PASSWORD", self.POSTGRES\_PASSWORD)

self.\_check\_default\_secret(

"FIRST\_SUPERUSER\_PASSWORD", self.FIRST\_SUPERUSER\_PASSWORD

)

return self

```

settings = Settings() # type: ignore
...

```py
from sqlmodel import Session, create_engine, select

from app import crud
from app.core.config import settings
from app.models import User, UserCreate

engine = create_engine(str(settings.SQLALCHEMY_DATABASE_URI))

# make sure all SQLAlchemy models are imported (app.models) before initializing DB
# otherwise, SQLAlchemy might fail to initialize relationships properly
# for more details: https://github.com/fastapi/full-stack-fastapi-template/issues/28

def init_db(session: Session) -> None:
    # Tables should be created with Alembic migrations
    # But if you don't want to use migrations, create
    # the tables un-commenting the next lines
    # from sqlmodel import SQLAlchemyModel

    # from app.core.engine import engine
    # This works because the models are already imported and registered from app.models
    # SQLAlchemyModel.metadata.create_all(engine)

    user = session.exec(
        select(User).where(User.email == settings.FIRST_SUPERUSER)
    ).first()
    if not user:
        user_in = UserCreate(
            email=settings.FIRST_SUPERUSER,
            password=settings.FIRST_SUPERUSER_PASSWORD,
            is_superuser=True,
        )
        user = crud.create_user(session=session, user_create=user_in)
...

```

```
```py
```

```
from datetime import datetime
from typing import Optional
```

```
import pandas as pd
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
from sqlalchemy import extract, or_
from sqlmodel import Session, select
```

```
from app.models import Item, User
```

```
def generate_report(
 user_id: int,
 year: int,
 month: Optional[int] = None,
 quarter: Optional[int] = None,
```

```
) -> str:
```

```
"""
```

Генерує звіт про споживання електроенергії для користувача за вказаний період.

Args:

user\_id: ID користувача.  
 year: Рік для звіту.  
 month: Місяць для звіту (1-12).  
 quarter: Квартал для звіту (1-4).

Returns:

Шлях до згенерованого PDF-файлу.

```
"""
```

```
with Session(engine) as session:
```

```
 user = session.exec(select(User).where(User.id == user_id)).first()
```

```
 if not user:
```

```
 raise ValueError(f"User with id {user_id} not found")
```

```
 # Формуємо запит до БД
```

```
 query = select(Item).where(Item.owner_id == user_id)
```

```

if month:
 # Фільтр за роком та місяцем
 query = query.where(
 extract("year", Item.reading_date) == year,
 extract("month", Item.reading_date) == month,
)
elif quarter:
 # Фільтр за роком та кварталом
 start_month = (quarter - 1) * 3 + 1
 end_month = start_month + 2
 query = query.where(
 extract("year", Item.reading_date) == year,
 or_(
 extract("month", Item.reading_date) == start_month,
 extract("month", Item.reading_date) == start_month + 1,
 extract("month", Item.reading_date) == end_month,
),
)
else:
 # Фільтр тільки за роком
 query = query.where(extract("year", Item.reading_date) == year)

Виконуємо запит та отримуємо дані
results = session.exec(query).all()

if not results:
 raise ValueError("No data found for the specified period")

Створюємо DataFrame з результатів запити
df = pd.DataFrame([item.model_dump() for item in results])

Перетворюємо поле reading_date на datetime
df["reading_date"] = pd.to_datetime(df["reading_date"])

Створюємо звіт у форматі PDF
file_name = f"report_{user_id}_{year}_{month or quarter}.pdf"
file_path = f"reports/{file_name}"
c = canvas.Canvas(file_path, pagesize=letter)

Додаємо заголовок
c.drawString(100, 750, "Звіт про споживання електроенергії")
c.drawString(100, 730, f"Користувач: {user.full_name or user.email}")

```

```

c.drawString(100, 710, f"Період: {year}-{month or quarter}")

Додаємо дані з DataFrame
y_position = 680
for index, row in df.iterrows():
 c.drawString(100, y_position, f"Дата: {row['reading_date'].strftime('%Y-%m-%d')}")
 c.drawString(300, y_position, f"Показник: {row['reading_value']}")
 c.drawString(400, y_position, f"Тариф: {row['kwh_tariff']}")
 y_position -= 20

Завершуємо створення PDF
c.showPage()
c.save()

return file_path

def send_report(email, file_path):
 pass

...

```py
import logging
import os
from datetime import datetime

from apscheduler.job import Job
from apscheduler.jobstores.sqlalchemy import SQLAlchemyJobStore
from apscheduler.schedulers.asyncio import AsyncIOScheduler

#from apscheduler.schedulers.background import BackgroundScheduler
from apscheduler.triggers.interval import IntervalTrigger
from sqlalchemy import select
from sqlmodel import Session

from app.api.deps import CurrentUser, SessionDep
from app.core.automatic_reports_sender import check_all_users_report_settings
from app.core.config import settings
from app.models import UserReportSettings

scheduler = AsyncIOScheduler()

```

```

async def start_scheduler():
    if not scheduler.running:
        print(f"[{datetime.now()}] Starting scheduler")
        scheduler.start()

    # Додаємо завдання для періодичної перевірки всіх користувачів
    scheduler.add_job(
        check_all_users_report_settings,
        trigger=IntervalTrigger(seconds=300),
        id="check_all_users",
        replace_existing=True,
    )
    logging.info("call check_all_users_report_settings")

async def shutdown_scheduler():
    if scheduler.running:
        print(f"[{datetime.now()}] Shutting down scheduler")
        scheduler.shutdown()

...

```py
from datetime import datetime, timedelta, timezone
from typing import Any

import jwt
from passlib.context import CryptContext

from app.core.config import settings

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

ALGORITHM = "HS256"

def create_access_token(subject: str | Any, expires_delta: timedelta) -> str:
 expire = datetime.now(timezone.utc) + expires_delta
 to_encode = {"exp": expire, "sub": str(subject)}
 encoded_jwt = jwt.encode(to_encode, settings.SECRET_KEY, algorithm=ALGORITHM)

```

```
return encoded_jwt
```

```
def verify_password(plain_password: str, hashed_password: str) -> bool:
 return pwd_context.verify(plain_password, hashed_password)
```

```
def get_password_hash(password: str) -> str:
 return pwd_context.hash(password)
...

```

```
```py
...

```

```
```py
from collections.abc import Generator
```

```
import pytest
from fastapi.testclient import TestClient
from sqlmodel import Session, delete
```

```
from app.core.config import settings
from app.core.db import engine, init_db
from app.main import app
from app.models import Item, User
from app.tests.utils.user import authentication_token_from_email
from app.tests.utils.utils import get_superuser_token_headers
```

```
@pytest.fixture(scope="session", autouse=True)
def db() -> Generator[Session, None, None]:
 with Session(engine) as session:
 init_db(session)
 yield session
 statement = delete(Item)
 session.execute(statement)
 statement = delete(User)
 session.execute(statement)
 session.commit()
```

```
@pytest.fixture(scope="module")
```

```
def client() -> Generator[TestClient, None, None]:
 with TestClient(app) as c:
 yield c
```

```
@pytest.fixture(scope="module")
def superuser_token_headers(client: TestClient) -> dict[str, str]:
 return get_superuser_token_headers(client)
```

```
@pytest.fixture(scope="module")
def normal_user_token_headers(client: TestClient, db: Session) -> dict[str, str]:
 return authentication_token_from_email(
 client=client, email=settings.EMAIL_TEST_USER, db=db
)
'''
```

```
'''
.py
'''
```

```
'''
.py
'''
```

```
'''
.py
import uuid
```

```
from fastapi.testclient import TestClient
from sqlmodel import Session
```

```
from app.core.config import settings
from app.tests.utils.item import create_random_item
```

```
def test_create_item(
 client: TestClient, superuser_token_headers: dict[str, str]
) -> None:
 data = {"title": "Foo", "description": "Fighters"}
 response = client.post(
 f"{settings.API_V1_STR}/items/",
 headers=superuser_token_headers,
 json=data,
)
```

```

assert response.status_code == 200
content = response.json()
assert content["title"] == data["title"]
assert content["description"] == data["description"]
assert "id" in content
assert "owner_id" in content

```

```

def test_read_item(
 client: TestClient, superuser_token_headers: dict[str, str], db: Session
) -> None:
 item = create_random_item(db)
 response = client.get(
 f"{settings.API_V1_STR}/items/{item.id}",
 headers=superuser_token_headers,
)
 assert response.status_code == 200
 content = response.json()
 assert content["title"] == item.title
 assert content["description"] == item.description
 assert content["id"] == str(item.id)
 assert content["owner_id"] == str(item.owner_id)

```

```

def test_read_item_not_found(
 client: TestClient, superuser_token_headers: dict[str, str]
) -> None:
 response = client.get(
 f"{settings.API_V1_STR}/items/{uuid.uuid4()}",
 headers=superuser_token_headers,
)
 assert response.status_code == 404
 content = response.json()
 assert content["detail"] == "Item not found"

```

```

def test_read_item_not_enough_permissions(
 client: TestClient, normal_user_token_headers: dict[str, str], db: Session
) -> None:
 item = create_random_item(db)
 response = client.get(
 f"{settings.API_V1_STR}/items/{item.id}",

```

```

 headers=normal_user_token_headers,
)
 assert response.status_code == 400
 content = response.json()
 assert content["detail"] == "Not enough permissions"

def test_read_items(
 client: TestClient, superuser_token_headers: dict[str, str], db: Session
) -> None:
 create_random_item(db)
 create_random_item(db)
 response = client.get(
 f"{settings.API_V1_STR}/items/",
 headers=superuser_token_headers,
)
 assert response.status_code == 200
 content = response.json()
 assert len(content["data"]) >= 2

def test_update_item(
 client: TestClient, superuser_token_headers: dict[str, str], db: Session
) -> None:
 item = create_random_item(db)
 data = {"title": "Updated title", "description": "Updated description"}
 response = client.put(
 f"{settings.API_V1_STR}/items/{item.id}",
 headers=superuser_token_headers,
 json=data,
)
 assert response.status_code == 200
 content = response.json()
 assert content["title"] == data["title"]
 assert content["description"] == data["description"]
 assert content["id"] == str(item.id)
 assert content["owner_id"] == str(item.owner_id)

def test_update_item_not_found(
 client: TestClient, superuser_token_headers: dict[str, str]
) -> None:

```

```

data = {"title": "Updated title", "description": "Updated description"}
response = client.put(
 f"{settings.API_V1_STR}/items/{uuid.uuid4()}",
 headers=superuser_token_headers,
 json=data,
)
assert response.status_code == 404
content = response.json()
assert content["detail"] == "Item not found"

```

```

def test_update_item_not_enough_permissions(
 client: TestClient, normal_user_token_headers: dict[str, str], db: Session
) -> None:
 item = create_random_item(db)
 data = {"title": "Updated title", "description": "Updated description"}
 response = client.put(
 f"{settings.API_V1_STR}/items/{item.id}",
 headers=normal_user_token_headers,
 json=data,
)
 assert response.status_code == 400
 content = response.json()
 assert content["detail"] == "Not enough permissions"

```

```

def test_delete_item(
 client: TestClient, superuser_token_headers: dict[str, str], db: Session
) -> None:
 item = create_random_item(db)
 response = client.delete(
 f"{settings.API_V1_STR}/items/{item.id}",
 headers=superuser_token_headers,
)
 assert response.status_code == 200
 content = response.json()
 assert content["message"] == "Item deleted successfully"

```

```

def test_delete_item_not_found(
 client: TestClient, superuser_token_headers: dict[str, str]
) -> None:

```

```

response = client.delete(
 f"{settings.API_V1_STR}/items/{uuid.uuid4()}",
 headers=superuser_token_headers,
)
assert response.status_code == 404
content = response.json()
assert content["detail"] == "Item not found"

def test_delete_item_not_enough_permissions(
 client: TestClient, normal_user_token_headers: dict[str, str], db: Session
) -> None:
 item = create_random_item(db)
 response = client.delete(
 f"{settings.API_V1_STR}/items/{item.id}",
 headers=normal_user_token_headers,
)
 assert response.status_code == 400
 content = response.json()
 assert content["detail"] == "Not enough permissions"
...

```py
from unittest.mock import patch

from fastapi.testclient import TestClient
from sqlmodel import Session, select

from app.core.config import settings
from app.core.security import verify_password
from app.models import User
from app.utils import generate_password_reset_token

def test_get_access_token(client: TestClient) -> None:
    login_data = {
        "username": settings.FIRST_SUPERUSER,
        "password": settings.FIRST_SUPERUSER_PASSWORD,
    }
    r = client.post(f"{settings.API_V1_STR}/login/access-token", data=login_data)
    tokens = r.json()
    assert r.status_code == 200

```

```

assert "access_token" in tokens
assert tokens["access_token"]

```

```

def test_get_access_token_incorrect_password(client: TestClient) -> None:
    login_data = {
        "username": settings.FIRST_SUPERUSER,
        "password": "incorrect",
    }
    r = client.post(f"{settings.API_V1_STR}/login/access-token", data=login_data)
    assert r.status_code == 400

```

```

def test_use_access_token(
    client: TestClient, superuser_token_headers: dict[str, str]
) -> None:
    r = client.post(
        f"{settings.API_V1_STR}/login/test-token",
        headers=superuser_token_headers,
    )
    result = r.json()
    assert r.status_code == 200
    assert "email" in result

```

```

def test_recovery_password(
    client: TestClient, normal_user_token_headers: dict[str, str]
) -> None:
    with (
        patch("app.core.config.settings.SMTP_HOST", "smtp.example.com"),
        patch("app.core.config.settings.SMTP_USER", "admin@example.com"),
    ):
        email = "test@example.com"
        r = client.post(
            f"{settings.API_V1_STR}/password-recovery/{email}",
            headers=normal_user_token_headers,
        )
        assert r.status_code == 200
        assert r.json() == {"message": "Password recovery email sent"}

```

```

def test_recovery_password_user_not_exists(

```

```

        client: TestClient, normal_user_token_headers: dict[str, str]
    ) -> None:
        email = "jVgQr@example.com"
        r = client.post(
            f"{settings.API_V1_STR}/password-recovery/{email}",
            headers=normal_user_token_headers,
        )
        assert r.status_code == 404


def test_reset_password(
    client: TestClient, superuser_token_headers: dict[str, str], db: Session
) -> None:
    token = generate_password_reset_token(email=settings.FIRST_SUPERUSER)
    data = {"new_password": "changethis", "token": token}
    r = client.post(
        f"{settings.API_V1_STR}/reset-password/",
        headers=superuser_token_headers,
        json=data,
    )
    assert r.status_code == 200
    assert r.json() == {"message": "Password updated successfully"}

    user_query = select(User).where(User.email == settings.FIRST_SUPERUSER)
    user = db.exec(user_query).first()
    assert user
    assert verify_password(data["new_password"], user.hashed_password)


def test_reset_password_invalid_token(
    client: TestClient, superuser_token_headers: dict[str, str]
) -> None:
    data = {"new_password": "changethis", "token": "invalid"}
    r = client.post(
        f"{settings.API_V1_STR}/reset-password/",
        headers=superuser_token_headers,
        json=data,
    )
    response = r.json()

    assert "detail" in response
    assert r.status_code == 400

```

```

    assert response["detail"] == "Invalid token"
    ...

```py
import uuid
from unittest.mock import patch

from fastapi.testclient import TestClient
from sqlmodel import Session, select

from app import crud
from app.core.config import settings
from app.core.security import verify_password
from app.models import User, UserCreate
from app.tests.utils.utils import random_email, random_lower_string

def test_get_users_superuser_me(
 client: TestClient, superuser_token_headers: dict[str, str]
) -> None:
 r = client.get(f'/{settings.API_V1_STR}/users/me', headers=superuser_token_headers)
 current_user = r.json()
 assert current_user
 assert current_user["is_active"] is True
 assert current_user["is_superuser"]
 assert current_user["email"] == settings.FIRST_SUPERUSER

def test_get_users_normal_user_me(
 client: TestClient, normal_user_token_headers: dict[str, str]
) -> None:
 r = client.get(f'/{settings.API_V1_STR}/users/me', headers=normal_user_token_headers)
 current_user = r.json()
 assert current_user
 assert current_user["is_active"] is True
 assert current_user["is_superuser"] is False
 assert current_user["email"] == settings.EMAIL_TEST_USER

def test_create_user_new_email(
 client: TestClient, superuser_token_headers: dict[str, str], db: Session
) -> None:

```

```

with (
 patch("app.utils.send_email", return_value=None),
 patch("app.core.config.settings.SMTP_HOST", "smtp.example.com"),
 patch("app.core.config.settings.SMTP_USER", "admin@example.com"),
):
 username = random_email()
 password = random_lower_string()
 data = {"email": username, "password": password}
 r = client.post(
 f"{settings.API_V1_STR}/users/",
 headers=superuser_token_headers,
 json=data,
)
 assert 200 <= r.status_code < 300
 created_user = r.json()
 user = crud.get_user_by_email(session=db, email=username)
 assert user
 assert user.email == created_user["email"]

```

```

def test_get_existing_user(
 client: TestClient, superuser_token_headers: dict[str, str], db: Session
) -> None:
 username = random_email()
 password = random_lower_string()
 user_in = UserCreate(email=username, password=password)
 user = crud.create_user(session=db, user_create=user_in)
 user_id = user.id
 r = client.get(
 f"{settings.API_V1_STR}/users/{user_id}",
 headers=superuser_token_headers,
)
 assert 200 <= r.status_code < 300
 api_user = r.json()
 existing_user = crud.get_user_by_email(session=db, email=username)
 assert existing_user
 assert existing_user.email == api_user["email"]

```

```

def test_get_existing_user_current_user(client: TestClient, db: Session) -> None:
 username = random_email()
 password = random_lower_string()

```

```

user_in = UserCreate(email=username, password=password)
user = crud.create_user(session=db, user_create=user_in)
user_id = user.id

login_data = {
 "username": username,
 "password": password,
}
r = client.post(f"{settings.API_V1_STR}/login/access-token", data=login_data)
tokens = r.json()
a_token = tokens["access_token"]
headers = {"Authorization": f"Bearer {a_token}"}

r = client.get(
 f"{settings.API_V1_STR}/users/{user_id}",
 headers=headers,
)
assert 200 <= r.status_code < 300
api_user = r.json()
existing_user = crud.get_user_by_email(session=db, email=username)
assert existing_user
assert existing_user.email == api_user["email"]

def test_get_existing_user_permissions_error(
 client: TestClient, normal_user_token_headers: dict[str, str]
) -> None:
 r = client.get(
 f"{settings.API_V1_STR}/users/{uuid.uuid4()}",
 headers=normal_user_token_headers,
)
 assert r.status_code == 403
 assert r.json() == {"detail": "The user doesn't have enough privileges"}

def test_create_user_existing_username(
 client: TestClient, superuser_token_headers: dict[str, str], db: Session
) -> None:
 username = random_email()
 # username = email
 password = random_lower_string()
 user_in = UserCreate(email=username, password=password)

```

```

crud.create_user(session=db, user_create=user_in)
data = {"email": username, "password": password}
r = client.post(
 f'{settings.API_V1_STR}/users/',
 headers=superuser_token_headers,
 json=data,
)
created_user = r.json()
assert r.status_code == 400
assert "_id" not in created_user

```

```

def test_create_user_by_normal_user(
 client: TestClient, normal_user_token_headers: dict[str, str]
) -> None:
 username = random_email()
 password = random_lower_string()
 data = {"email": username, "password": password}
 r = client.post(
 f'{settings.API_V1_STR}/users/',
 headers=normal_user_token_headers,
 json=data,
)
 assert r.status_code == 403

```

```

def test_retrieve_users(
 client: TestClient, superuser_token_headers: dict[str, str], db: Session
) -> None:
 username = random_email()
 password = random_lower_string()
 user_in = UserCreate(email=username, password=password)
 crud.create_user(session=db, user_create=user_in)

 username2 = random_email()
 password2 = random_lower_string()
 user_in2 = UserCreate(email=username2, password=password2)
 crud.create_user(session=db, user_create=user_in2)

 r = client.get(f'{settings.API_V1_STR}/users/', headers=superuser_token_headers)
 all_users = r.json()

```

```

assert len(all_users["data"]) > 1
assert "count" in all_users
for item in all_users["data"]:
 assert "email" in item

```

```

def test_update_user_me(
 client: TestClient, normal_user_token_headers: dict[str, str], db: Session
) -> None:
 full_name = "Updated Name"
 email = random_email()
 data = {"full_name": full_name, "email": email}
 r = client.patch(
 f"{settings.API_V1_STR}/users/me",
 headers=normal_user_token_headers,
 json=data,
)
 assert r.status_code == 200
 updated_user = r.json()
 assert updated_user["email"] == email
 assert updated_user["full_name"] == full_name

 user_query = select(User).where(User.email == email)
 user_db = db.exec(user_query).first()
 assert user_db
 assert user_db.email == email
 assert user_db.full_name == full_name

```

```

def test_update_password_me(
 client: TestClient, superuser_token_headers: dict[str, str], db: Session
) -> None:
 new_password = random_lower_string()
 data = {
 "current_password": settings.FIRST_SUPERUSER_PASSWORD,
 "new_password": new_password,
 }
 r = client.patch(
 f"{settings.API_V1_STR}/users/me/password",
 headers=superuser_token_headers,
 json=data,
)

```

```

assert r.status_code == 200
updated_user = r.json()
assert updated_user["message"] == "Password updated successfully"

user_query = select(User).where(User.email == settings.FIRST_SUPERUSER)
user_db = db.exec(user_query).first()
assert user_db
assert user_db.email == settings.FIRST_SUPERUSER
assert verify_password(new_password, user_db.hashed_password)

Revert to the old password to keep consistency in test
old_data = {
 "current_password": new_password,
 "new_password": settings.FIRST_SUPERUSER_PASSWORD,
}
r = client.patch(
 f"{settings.API_V1_STR}/users/me/password",
 headers=superuser_token_headers,
 json=old_data,
)
db.refresh(user_db)

assert r.status_code == 200
assert verify_password(settings.FIRST_SUPERUSER_PASSWORD, user_db.hashed_password)

def test_update_password_me_incorrect_password(
 client: TestClient, superuser_token_headers: dict[str, str]
) -> None:
 new_password = random_lower_string()
 data = {"current_password": new_password, "new_password": new_password}
 r = client.patch(
 f"{settings.API_V1_STR}/users/me/password",
 headers=superuser_token_headers,
 json=data,
)
 assert r.status_code == 400
 updated_user = r.json()
 assert updated_user["detail"] == "Incorrect password"

def test_update_user_me_email_exists(

```

```

 client: TestClient, normal_user_token_headers: dict[str, str], db: Session
) -> None:
 username = random_email()
 password = random_lower_string()
 user_in = UserCreate(email=username, password=password)
 user = crud.create_user(session=db, user_create=user_in)

 data = {"email": user.email}
 r = client.patch(
 f"{settings.API_V1_STR}/users/me",
 headers=normal_user_token_headers,
 json=data,
)
 assert r.status_code == 409
 assert r.json()["detail"] == "User with this email already exists"

def test_update_password_me_same_password_error(
 client: TestClient, superuser_token_headers: dict[str, str]
) -> None:
 data = {
 "current_password": settings.FIRST_SUPERUSER_PASSWORD,
 "new_password": settings.FIRST_SUPERUSER_PASSWORD,
 }
 r = client.patch(
 f"{settings.API_V1_STR}/users/me/password",
 headers=superuser_token_headers,
 json=data,
)
 assert r.status_code == 400
 updated_user = r.json()
 assert (
 updated_user["detail"] == "New password cannot be the same as the current one"
)

def test_register_user(client: TestClient, db: Session) -> None:
 username = random_email()
 password = random_lower_string()
 full_name = random_lower_string()
 data = {"email": username, "password": password, "full_name": full_name}
 r = client.post(

```

```

 f'{settings.API_V1_STR}/users/signup',
 json=data,
)
 assert r.status_code == 200
 created_user = r.json()
 assert created_user["email"] == username
 assert created_user["full_name"] == full_name

 user_query = select(User).where(User.email == username)
 user_db = db.exec(user_query).first()
 assert user_db
 assert user_db.email == username
 assert user_db.full_name == full_name
 assert verify_password(password, user_db.hashed_password)

def test_register_user_already_exists_error(client: TestClient) -> None:
 password = random_lower_string()
 full_name = random_lower_string()
 data = {
 "email": settings.FIRST_SUPERUSER,
 "password": password,
 "full_name": full_name,
 }
 r = client.post(
 f'{settings.API_V1_STR}/users/signup',
 json=data,
)
 assert r.status_code == 400
 assert r.json()["detail"] == "The user with this email already exists in the system"

def test_update_user(
 client: TestClient, superuser_token_headers: dict[str, str], db: Session
) -> None:
 username = random_email()
 password = random_lower_string()
 user_in = UserCreate(email=username, password=password)
 user = crud.create_user(session=db, user_create=user_in)

 data = {"full_name": "Updated_full_name"}
 r = client.patch(

```

```

 f"{settings.API_V1_STR}/users/{user.id}",
 headers=superuser_token_headers,
 json=data,
)
 assert r.status_code == 200
 updated_user = r.json()

 assert updated_user["full_name"] == "Updated_full_name"

 user_query = select(User).where(User.email == username)
 user_db = db.exec(user_query).first()
 db.refresh(user_db)
 assert user_db
 assert user_db.full_name == "Updated_full_name"

def test_update_user_not_exists(
 client: TestClient, superuser_token_headers: dict[str, str]
) -> None:
 data = {"full_name": "Updated_full_name"}
 r = client.patch(
 f"{settings.API_V1_STR}/users/{uuid.uuid4()}",
 headers=superuser_token_headers,
 json=data,
)
 assert r.status_code == 404
 assert r.json()["detail"] == "The user with this id does not exist in the system"

def test_update_user_email_exists(
 client: TestClient, superuser_token_headers: dict[str, str], db: Session
) -> None:
 username = random_email()
 password = random_lower_string()
 user_in = UserCreate(email=username, password=password)
 user = crud.create_user(session=db, user_create=user_in)

 username2 = random_email()
 password2 = random_lower_string()
 user_in2 = UserCreate(email=username2, password=password2)
 user2 = crud.create_user(session=db, user_create=user_in2)

```

```

data = {"email": user2.email}
r = client.patch(
 f"{settings.API_V1_STR}/users/{user.id}",
 headers=superuser_token_headers,
 json=data,
)
assert r.status_code == 409
assert r.json()["detail"] == "User with this email already exists"

```

```

def test_delete_user_me(client: TestClient, db: Session) -> None:
 username = random_email()
 password = random_lower_string()
 user_in = UserCreate(email=username, password=password)
 user = crud.create_user(session=db, user_create=user_in)
 user_id = user.id

 login_data = {
 "username": username,
 "password": password,
 }
 r = client.post(f"{settings.API_V1_STR}/login/access-token", data=login_data)
 tokens = r.json()
 a_token = tokens["access_token"]
 headers = {"Authorization": f"Bearer {a_token}"}

 r = client.delete(
 f"{settings.API_V1_STR}/users/me",
 headers=headers,
)
 assert r.status_code == 200
 deleted_user = r.json()
 assert deleted_user["message"] == "User deleted successfully"
 result = db.exec(select(User).where(User.id == user_id)).first()
 assert result is None

 user_query = select(User).where(User.id == user_id)
 user_db = db.execute(user_query).first()
 assert user_db is None

```

```

def test_delete_user_me_as_superuser(

```

```

 client: TestClient, superuser_token_headers: dict[str, str]
) -> None:
 r = client.delete(
 f'{settings.API_V1_STR}/users/me',
 headers=superuser_token_headers,
)
 assert r.status_code == 403
 response = r.json()
 assert response["detail"] == "Super users are not allowed to delete themselves"

```

```

def test_delete_user_super_user(
 client: TestClient, superuser_token_headers: dict[str, str], db: Session
) -> None:
 username = random_email()
 password = random_lower_string()
 user_in = UserCreate(email=username, password=password)
 user = crud.create_user(session=db, user_create=user_in)
 user_id = user.id
 r = client.delete(
 f'{settings.API_V1_STR}/users/{user_id}',
 headers=superuser_token_headers,
)
 assert r.status_code == 200
 deleted_user = r.json()
 assert deleted_user["message"] == "User deleted successfully"
 result = db.exec(select(User).where(User.id == user_id)).first()
 assert result is None

```

```

def test_delete_user_not_found(
 client: TestClient, superuser_token_headers: dict[str, str]
) -> None:
 r = client.delete(
 f'{settings.API_V1_STR}/users/{uuid.uuid4()}',
 headers=superuser_token_headers,
)
 assert r.status_code == 404
 assert r.json()["detail"] == "User not found"

```

```

def test_delete_user_current_super_user_error(

```

```

 client: TestClient, superuser_token_headers: dict[str, str], db: Session
) -> None:
 super_user = crud.get_user_by_email(session=db, email=settings.FIRST_SUPERUSER)
 assert super_user
 user_id = super_user.id

 r = client.delete(
 f"{settings.API_V1_STR}/users/{user_id}",
 headers=superuser_token_headers,
)
 assert r.status_code == 403
 assert r.json()["detail"] == "Super users are not allowed to delete themselves"

```

```

def test_delete_user_without_privileges(
 client: TestClient, normal_user_token_headers: dict[str, str], db: Session
) -> None:
 username = random_email()
 password = random_lower_string()
 user_in = UserCreate(email=username, password=password)
 user = crud.create_user(session=db, user_create=user_in)

 r = client.delete(
 f"{settings.API_V1_STR}/users/{user.id}",
 headers=normal_user_token_headers,
)
 assert r.status_code == 403
 assert r.json()["detail"] == "The user doesn't have enough privileges"

```

```

'''
.py
'''

```

```

'''
.py
from fastapi.encoders import jsonable_encoder
from sqlmodel import Session

from app import crud
from app.core.security import verify_password
from app.models import User, UserCreate, UserUpdate
from app.tests.utils import random_email, random_lower_string

```

```
def test_create_user(db: Session) -> None:
```

```
 email = random_email()
 password = random_lower_string()
 user_in = UserCreate(email=email, password=password)
 user = crud.create_user(session=db, user_create=user_in)
 assert user.email == email
 assert hasattr(user, "hashed_password")
```

```
def test_authenticate_user(db: Session) -> None:
```

```
 email = random_email()
 password = random_lower_string()
 user_in = UserCreate(email=email, password=password)
 user = crud.create_user(session=db, user_create=user_in)
 authenticated_user = crud.authenticate(session=db, email=email, password=password)
 assert authenticated_user
 assert user.email == authenticated_user.email
```

```
def test_not_authenticate_user(db: Session) -> None:
```

```
 email = random_email()
 password = random_lower_string()
 user = crud.authenticate(session=db, email=email, password=password)
 assert user is None
```

```
def test_check_if_user_is_active(db: Session) -> None:
```

```
 email = random_email()
 password = random_lower_string()
 user_in = UserCreate(email=email, password=password)
 user = crud.create_user(session=db, user_create=user_in)
 assert user.is_active is True
```

```
def test_check_if_user_is_active_inactive(db: Session) -> None:
```

```
 email = random_email()
 password = random_lower_string()
 user_in = UserCreate(email=email, password=password, disabled=True)
 user = crud.create_user(session=db, user_create=user_in)
 assert user.is_active
```

```

def test_check_if_user_is_superuser(db: Session) -> None:
 email = random_email()
 password = random_lower_string()
 user_in = UserCreate(email=email, password=password, is_superuser=True)
 user = crud.create_user(session=db, user_create=user_in)
 assert user.is_superuser is True

def test_check_if_user_is_superuser_normal_user(db: Session) -> None:
 username = random_email()
 password = random_lower_string()
 user_in = UserCreate(email=username, password=password)
 user = crud.create_user(session=db, user_create=user_in)
 assert user.is_superuser is False

def test_get_user(db: Session) -> None:
 password = random_lower_string()
 username = random_email()
 user_in = UserCreate(email=username, password=password, is_superuser=True)
 user = crud.create_user(session=db, user_create=user_in)
 user_2 = db.get(User, user.id)
 assert user_2
 assert user.email == user_2.email
 assert jsonable_encoder(user) == jsonable_encoder(user_2)

def test_update_user(db: Session) -> None:
 password = random_lower_string()
 email = random_email()
 user_in = UserCreate(email=email, password=password, is_superuser=True)
 user = crud.create_user(session=db, user_create=user_in)
 new_password = random_lower_string()
 user_in_update = UserUpdate(password=new_password, is_superuser=True)
 if user.id is not None:
 crud.update_user(session=db, db_user=user, user_in=user_in_update)
 user_2 = db.get(User, user.id)
 assert user_2
 assert user.email == user_2.email
 assert verify_password(new_password, user_2.hashed_password)
...

```

```
```py
```
```

```
```py
```

```
from unittest.mock import MagicMock, patch
```

```
from sqlmodel import select
```

```
from app.backend_pre_start import init, logger
```

```
def test_init_successful_connection() -> None:
```

```
    engine_mock = MagicMock()
```

```
    session_mock = MagicMock()
```

```
    exec_mock = MagicMock(return_value=True)
```

```
    session_mock.configure_mock(**{"exec.return_value": exec_mock})
```

```
    with (
```

```
        patch("sqlmodel.Session", return_value=session_mock),
```

```
        patch.object(logger, "info"),
```

```
        patch.object(logger, "error"),
```

```
        patch.object(logger, "warn"),
```

```
    ):

```

```
        try:
```

```
            init(engine_mock)
```

```
            connection_successful = True
```

```
        except Exception:
```

```
            connection_successful = False
```

```
    assert (
```

```
        connection_successful
```

```
    ), "The database connection should be successful and not raise an exception."
```

```
    assert session_mock.exec.called_once_with(
```

```
        select(1)
```

```
    ), "The session should execute a select statement once."
```

```
```
```

```
```py
```

```
from unittest.mock import MagicMock, patch
```

```

from sqlmodel import select

from app.tests_pre_start import init, logger

def test_init_successful_connection() -> None:
    engine_mock = MagicMock()

    session_mock = MagicMock()
    exec_mock = MagicMock(return_value=True)
    session_mock.configure_mock(**{"exec.return_value": exec_mock})

    with (
        patch("sqlmodel.Session", return_value=session_mock),
        patch.object(logger, "info"),
        patch.object(logger, "error"),
        patch.object(logger, "warn"),
    ):
        try:
            init(engine_mock)
            connection_successful = True
        except Exception:
            connection_successful = False

    assert (
        connection_successful
    ), "The database connection should be successful and not raise an exception."

    assert session_mock.exec.called_once_with(
        select(1)
    ), "The session should execute a select statement once."
...

```py
...

```py
from sqlmodel import Session

from app import crud
from app.models import Item, ItemCreate

```

```

from app.tests.utils.user import create_random_user
from app.tests.utils.utils import random_lower_string

def create_random_item(db: Session) -> Item:
    user = create_random_user(db)
    owner_id = user.id
    assert owner_id is not None
    title = random_lower_string()
    description = random_lower_string()
    item_in = ItemCreate(title=title, description=description)
    return crud.create_item(session=db, item_in=item_in, owner_id=owner_id)
'''

'''py
from fastapi.testclient import TestClient
from sqlmodel import Session

from app import crud
from app.core.config import settings
from app.models import User, UserCreate, UserUpdate
from app.tests.utils.utils import random_email, random_lower_string

def user_authentication_headers(
    *, client: TestClient, email: str, password: str
) -> dict[str, str]:
    data = {"username": email, "password": password}

    r = client.post(f"{settings.API_V1_STR}/login/access-token", data=data)
    response = r.json()
    auth_token = response["access_token"]
    headers = {"Authorization": f"Bearer {auth_token}"}
    return headers

def create_random_user(db: Session) -> User:
    email = random_email()
    password = random_lower_string()
    user_in = UserCreate(email=email, password=password)
    user = crud.create_user(session=db, user_create=user_in)
    return user

```

```

def authentication_token_from_email(
    *, client: TestClient, email: str, db: Session
) -> dict[str, str]:
    """
    Return a valid token for the user with given email.

    If the user doesn't exist it is created first.
    """
    password = random_lower_string()
    user = crud.get_user_by_email(session=db, email=email)
    if not user:
        user_in_create = UserCreate(email=email, password=password)
        user = crud.create_user(session=db, user_create=user_in_create)
    else:
        user_in_update = UserUpdate(password=password)
        if not user.id:
            raise Exception("User id not set")
        user = crud.update_user(session=db, db_user=user, user_in=user_in_update)

    return user_authentication_headers(client=client, email=email, password=password)
"""

"""py
import random
import string

from fastapi.testclient import TestClient

from app.core.config import settings

def random_lower_string() -> str:
    return "".join(random.choices(string.ascii_lowercase, k=32))

def random_email() -> str:
    return f"{random_lower_string()}@{random_lower_string()}.com"

def get_superuser_token_headers(client: TestClient) -> dict[str, str]:

```

```

login_data = {
    "username": settings.FIRST_SUPERUSER,
    "password": settings.FIRST_SUPERUSER_PASSWORD,
}
r = client.post(f"{settings.API_V1_STR}/login/access-token", data=login_data)
tokens = r.json()
a_token = tokens["access_token"]
headers = {"Authorization": f"Bearer {a_token}"}
return headers
...

```py
...

```js
import * as fs from "node:fs"

async function modifyOpenAPIFile(filePath) {
  try {
    const data = await fs.promises.readFile(filePath)
    const openapiContent = JSON.parse(data)

    const paths = openapiContent.paths
    for (const pathKey of Object.keys(paths)) {
      const pathData = paths[pathKey]
      for (const method of Object.keys(pathData)) {
        const operation = pathData[method]
        if (operation.tags && operation.tags.length > 0) {
          const tag = operation.tags[0]
          const operationId = operation.operationId
          const toRemove = `${tag}-`
          if (operationId.startsWith(toRemove)) {
            const newOperationId = operationId.substring(toRemove.length)
            operation.operationId = newOperationId
          }
        }
      }
    }
  }

  await fs.promises.writeFile(
    filePath,
    JSON.stringify(openapiContent, null, 2),
  )
}

```

```

    )
    console.log("File successfully modified")
  } catch (err) {
    console.error("Error:", err)
  }
}

const filePath = "./openapi.json"
modifyOpenAPIFile(filePath)
```

```tsx
import React, { Component, ReactNode } from "react";

interface ErrorBoundaryProps {
  children: ReactNode;
  fallback: ReactNode; // Компонент або JSX для відображення у разі помилки
}

interface ErrorBoundaryState {
  hasError: boolean;
}

class ErrorBoundary extends Component<ErrorBoundaryProps, ErrorBoundaryState> {
  constructor(props: ErrorBoundaryProps) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(_: Error): ErrorBoundaryState {
    // Оновлює стан для відображення fallback UI
    return { hasError: true };
  }

  componentDidCatch(error: Error, errorInfo: React.ErrorInfo) {
    // Логування помилки для аналітики
    console.error("ErrorBoundary caught an error:", error, errorInfo);
  }

  render() {
    if (this.state.hasError) {
      return this.props.fallback; // Відображення fallback-компонента
    }
  }
}

```

```

    }

    return this.props.children; // Відображення дочірніх компонентів
  }
}

export default ErrorBoundary;
```

```tsx
import { ChakraProvider } from "@chakra-ui/react"
import { QueryClient, QueryClientProvider } from "@tanstack/react-query"
import { RouterProvider, createRouter } from "@tanstack/react-router"
import ReactDOM from "react-dom/client"
import { routeTree } from "./routeTree.gen"
import ErrorBoundary from "./ErrorBoundary";
import { StrictMode } from "react"
import { OpenAPI } from "./client"
import theme from "./theme"

import store from './store';
import { Provider } from 'react-redux';
import { PollingProvider } from './contexts/PollingContext'

OpenAPI.BASE = import.meta.env.VITE_API_URL
OpenAPI.TOKEN = async () => {
  return localStorage.getItem("access_token") || ""
}

const queryClient = new QueryClient()

const router = createRouter({ routeTree })
declare module "@tanstack/react-router" {
  interface Register {
    router: typeof router
  }
}

ReactDOM.createRoot(document.getElementById("root")!).render(
  <StrictMode>
    <ErrorBoundary fallback={<h1>Щось пішло не так!</h1>}>

```

```

    <Provider store={store}>
      <ChakraProvider theme={theme}>
        <QueryClientProvider client={queryClient}>

          <PollingProvider>
            <RouterProvider router={router} />
          </PollingProvider>

          </QueryClientProvider>
        </ChakraProvider>
      </Provider>
    </ErrorBoundary>
  </StrictMode>
)
```

```tsx

import { Message } from './client';

import { createSlice, PayloadAction } from '@reduxjs/toolkit';

export interface SSEState {
  messages: Message[];
  isConnected: boolean;
  retries: number;
}

const initialState: SSEState = {
  messages: [],
  isConnected: false,
  retries: 0,
};

const sseSlice = createSlice({
  name: 'sse',
  initialState,
  reducers: {
    setMessages: (state, action: PayloadAction<Message[]>) => {
      state.messages = action.payload;
    },
  },
});

```

```

    setConnectionStatus: (state, action: PayloadAction<boolean>) => {
      state.isConnected = action.payload;
    },
    setRetries: (state, action: PayloadAction<number>) => {
      state.retries = action.payload;
    },
  },
});

export const { setMessages, setConnectionStatus, setRetries } = sseSlice.actions;

export default sseSlice.reducer;
```

```.tsx

import { configureStore } from '@reduxjs/toolkit'
import sseReducer from './reducers';

const store = configureStore({
  reducer: {
    sse: sseReducer,
  }
})

// Infer the `RootState` and `AppDispatch` types from the store itself
export type RootState = ReturnType<typeof store.getState>
// Inferred type: {posts: PostsState, comments: CommentsState, users: UsersState}
export type AppDispatch = typeof store.dispatch

export default store;```

```.tsx
import { extendTheme } from "@chakra-ui/react"

const disabledStyles = {
 _disabled: {
 backgroundColor: "ui.main",
 },
}
}

```

```

const theme = extendTheme({
 colors: {
 ui: {
 main: "#009688",
 secondary: "#EDF2F7",
 success: "#48BB78",
 danger: "#E53E3E",
 light: "#FAFAFA",
 dark: "#1A202C",
 darkSlate: "#252D3D",
 dim: "#A0AEC0",
 },
 },
 components: {
 Button: {
 variants: {
 primary: {
 backgroundColor: "ui.main",
 color: "ui.light",
 _hover: {
 backgroundColor: "#00766C",
 },
 _disabled: {
 ...disabledStyles,
 _hover: {
 ...disabledStyles,
 },
 },
 },
 danger: {
 backgroundColor: "ui.danger",
 color: "ui.light",
 _hover: {
 backgroundColor: "#E32727",
 },
 },
 },
 },
 Tabs: {
 variants: {
 enclosed: {
 tab: {

```



```

interface UserCreateForm extends UserCreate {
 confirm_password: string
}

const AddUser = ({ isOpen, onClose }: AddUserProps) => {
 const queryClient = useQueryClient()
 const showToast = useCustomToast()
 const {
 register,
 handleSubmit,
 reset,
 getValues,
 formState: { errors, isSubmitting },
 } = useForm<UserCreateForm>({
 mode: "onBlur",
 criteriaMode: "all",
 defaultValues: {
 email: "",
 full_name: "",
 password: "",
 confirm_password: "",
 is_superuser: false,
 is_active: false,
 },
 })

 const mutation = useMutation({
 mutationFn: (data: UserCreate) =>
 UsersService.createUser({ requestBody: data }),
 onSuccess: () => {
 showToast("Success!", "User created successfully.", "success")
 reset()
 onClose()
 },
 onError: (err: ApiError) => {
 handleError(err, showToast)
 },
 onSettled: () => {
 queryClient.invalidateQueries({ queryKey: ["users"] })
 },
 })
}

```

```

const onSubmit: SubmitHandler<UserCreateForm> = (data) => {
 mutation.mutate(data)
}

return (
 <>
 <Modal
 isOpen={isOpen}
 onClose={onClose}
 size={{ base: "sm", md: "md" }}
 isCentered
 >
 <ModalOverlay />
 <ModalContent as="form" onSubmit={handleSubmit(onSubmit)}>
 <ModalHeader>Add User</ModalHeader>
 <ModalCloseButton />
 <ModalBody pb={6}>
 <FormControl isRequired isInvalid={!!errors.email}>
 <FormLabel htmlFor="email">Email</FormLabel>
 <Input
 id="email"
 {...register("email", {
 required: "Email is required",
 pattern: emailPattern,
 })}
 placeholder="Email"
 type="email"
 />
 {errors.email && (
 <FormErrorMessage>{errors.email.message}</FormErrorMessage>
)}
 </FormControl>
 <FormControl mt={4} isInvalid={!!errors.full_name}>
 <FormLabel htmlFor="name">Full name</FormLabel>
 <Input
 id="name"
 {...register("full_name")}
 placeholder="Full name"
 type="text"
 />
 {errors.full_name && (

```

```

 <FormErrorMessage>{errors.full_name.message}</FormErrorMessage>
)}
</FormControl>
<FormControl mt={4} isRequired isValid={!errors.password}>
 <FormLabel htmlFor="password">Set Password</FormLabel>
 <Input
 id="password"
 {...register("password", {
 required: "Password is required",
 minLength: {
 value: 8,
 message: "Password must be at least 8 characters",
 },
 })}
 placeholder="Password"
 type="password"
 />
 {errors.password && (
 <FormErrorMessage>{errors.password.message}</FormErrorMessage>
)}
</FormControl>
<FormControl
 mt={4}
 isRequired
 isValid={!errors.confirm_password}
>
 <FormLabel htmlFor="confirm_password">Confirm Password</FormLabel>
 <Input
 id="confirm_password"
 {...register("confirm_password", {
 required: "Please confirm your password",
 validate: (value) =>
 value === getValues().password ||
 "The passwords do not match",
 })}
 placeholder="Password"
 type="password"
 />
 {errors.confirm_password && (
 <FormErrorMessage>
 {errors.confirm_password.message}
 </FormErrorMessage>
)}

```

```

 })
 </FormControl>
 <Flex mt={4}>
 <FormControl>
 <Checkbox {...register("is_superuser")} colorScheme="teal">
 Is superuser?
 </Checkbox>
 </FormControl>
 <FormControl>
 <Checkbox {...register("is_active")} colorScheme="teal">
 Is active?
 </Checkbox>
 </FormControl>
 </Flex>
</ModalBody>
<ModalFooter gap={3}>
 <Button variant="primary" type="submit" isLoading={isSubmitting}>
 Save
 </Button>
 <Button onClick={onClose}>Cancel</Button>
</ModalFooter>
</ModalContent>
</Modal>
</>
)
}

```

```
export default AddUser
```

```
```
```

```
```tsx
```

```

import {
 Button,
 Checkbox,
 Flex,
 FormControl,
 FormErrorMessage,
 FormLabel,
 Input,
 Modal,
 ModalBody,
 ModalCloseButton,

```

```

 ModalContent,
 ModalFooter,
 ModalHeader,
 ModalOverlay,
 } from "@chakra-ui/react"
import { useMutation, useQueryClient } from "@tanstack/react-query"
import { type SubmitHandler, useForm } from "react-hook-form"

import {
 type ApiError,
 type UserPublic,
 type UserUpdate,
 UsersService,
} from "../../client"
import useCustomToast from "../../hooks/useCustomToast"
import { emailPattern, handleError } from "../../utils"

interface EditUserProps {
 user: UserPublic
 isOpen: boolean
 onClose: () => void
}

interface UserUpdateForm extends UserUpdate {
 confirm_password: string
}

const EditUser = ({ user, isOpen, onClose }: EditUserProps) => {
 const queryClient = useQueryClient()
 const showToast = useCustomToast()

 const {
 register,
 handleSubmit,
 reset,
 getValues,
 formState: { errors, isSubmitting, isDirty },
 } = useForm<UserUpdateForm>({
 mode: "onBlur",
 criteriaMode: "all",
 defaultValues: user,
 })

```

```

const mutation = useMutation({
 mutationFn: (data: UserUpdateForm) =>
 UsersService.usersUpdateUser({ userId: user.id, requestBody: data }),
 onSuccess: () => {
 showToast("Success!", "User updated successfully.", "success")
 onClose()
 },
 onError: (err: ApiError) => {
 handleError(err, showToast)
 },
 onSettled: () => {
 queryClient.invalidateQueries({ queryKey: ["users"] })
 },
})

const onSubmit: SubmitHandler<UserUpdateForm> = async (data) => {
 if (data.password === "") {
 data.password = undefined
 }
 mutation.mutate(data)
}

const onCancel = () => {
 reset()
 onClose()
}

return (
 <>
 <Modal
 isOpen={isOpen}
 onClose={onClose}
 size={{ base: "sm", md: "md" }}
 isCentered
 >
 <ModalOverlay />
 <ModalContent as="form" onSubmit={handleSubmit(onSubmit)}>
 <ModalHeader>Edit User</ModalHeader>
 <ModalCloseButton />
 <ModalBody pb={6}>
 <FormControl isValid={!errors.email}>

```

```

<FormLabel htmlFor="email">Email</FormLabel>
<Input
 id="email"
 {...register("email", {
 required: "Email is required",
 pattern: emailPattern,
 })}
 placeholder="Email"
 type="email"
/>
{errors.email && (
 <FormErrorMessage>{errors.email.message}</FormErrorMessage>
)}
</FormControl>
<FormControl mt={4}>
 <FormLabel htmlFor="name">Full name</FormLabel>
 <Input id="name" {...register("full_name")} type="text" />
</FormControl>
<FormControl mt={4} isValid={!errors.password}>
 <FormLabel htmlFor="password">Set Password</FormLabel>
 <Input
 id="password"
 {...register("password", {
 minLength: {
 value: 8,
 message: "Password must be at least 8 characters",
 },
 })}
 placeholder="Password"
 type="password"
 />
 {errors.password && (
 <FormErrorMessage>{errors.password.message}</FormErrorMessage>
)}
</FormControl>
<FormControl mt={4} isValid={!errors.confirm_password}>
 <FormLabel htmlFor="confirm_password">Confirm Password</FormLabel>
 <Input
 id="confirm_password"
 {...register("confirm_password", {
 validate: (value) =>
 value === getValues().password ||

```

```

 "The passwords do not match",
)})
 placeholder="Password"
 type="password"
 />
 {errors.confirm_password && (
 <FormErrorMessage>
 {errors.confirm_password.message}
 </FormErrorMessage>
)}
</FormControl>
<Flex>
 <FormControl mt={4}>
 <Checkbox {...register("is_superuser")} colorScheme="teal">
 Is superuser?
 </Checkbox>
 </FormControl>
 <FormControl mt={4}>
 <Checkbox {...register("is_active")} colorScheme="teal">
 Is active?
 </Checkbox>
 </FormControl>
</Flex>
</ModalBody>

<ModalFooter gap={3}>
 <Button
 variant="primary"
 type="submit"
 isLoading={isSubmitting}
 isDisabled={!isDirty}
 >
 Save
 </Button>
 <Button onClick={onCancel}>Cancel</Button>
</ModalFooter>
</ModalContent>
</Modal>
</>
)
}

```

```

export default EditUser
...

```tsx
import React, { useState, useEffect } from 'react';
import { Box, Heading, Text, Center, Alert, AlertIcon } from '@chakra-ui/react';
import { ComparisonService } from '../client';
import { TrendsResponse } from '../client/models';

const Comparison = () => {
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [trendData, setTrendData] = useState<TrendsResponse | null>(null); // Використовуємо згенеровану модель TrendsResponse

  useEffect(() => {
    const fetchData = async () => {
      setLoading(true);
      setError(null);
      try {
        const response = await ComparisonService.analyticsComparison();
        setTrendData(response);
      } catch (err) {
        if (err instanceof Error) {
          // Обробка помилок, якщо це стандартний об'єкт Error
          setError(err.message);
        } else if (err.response && err.response.data && err.response.data.detail) {
          // Обробка помилок, якщо це помилка від API з полем detail
          setError(err.response.data.detail);
        } else {
          // Обробка інших невідомих помилок
          setError('An error occurred while fetching data.');
```

```

if (loading) {
  return (
    <Center height="100vh" bg="gray.100">
      <Text>Loading...</Text>
    </Center>
  );
}

if (error) {
  return (
    <Center height="100vh" bg="gray.100">
      <Alert status="error">
        <AlertIcon />
        {error}
      </Alert>
    </Center>
  );
}

return (
  <Center height="100vh" bg="gray.100">
    <Box textAlign="center" p={6} boxShadow="md" bg="white" borderRadius="md">
      <Heading as="h1" size="xl" mb={4} color="teal.500">
        Порівняльний аналіз
      </Heading>
      {/* Тут ви можете вивести trendData, наприклад, у вигляді таблиці */}
      {trendData && trendData.data.length > 0 ? (
        <Text fontSize="lg" color="gray.600">
          Дані завантажено. Кількість записів: {trendData.data.length}
        </Text>
      ) : (
        <Text fontSize="lg" color="gray.600">
          Немає даних для порівняння.
        </Text>
      )}
    </Box>
  </Center>
);
};

export default Comparison;``

```

```

```.tsx

import React, { useEffect, useState } from 'react';
import { Box, Heading, Text, Center, Spinner, VStack } from '@chakra-ui/react';
import { PredictionService } from '../client';
import { usePolling } from '../contexts/PollingContext';
import { ForecastsResponse } from '../client/models';
import { Line, Bar } from 'react-chartjs-2';
import { Chart as ChartJS, registerables, TooltipItem } from 'chart.js';
import zoomPlugin from 'chartjs-plugin-zoom';

ChartJS.register(...registerables, zoomPlugin);

const Prediction = () => {
 const [loading, setLoading] = useState(true);
 const [error, setError] = useState(null);
 const [forecasts, setForecasts] = useState<ForecastsResponse | null>(null);

 const { data: messages } = usePolling();

 useEffect(() => {
 const startTask = async () => {
 try {
 setLoading(true);
 const response = await PredictionService.analyticsStartTask();
 if (response !== null) {
 setForecasts(response);
 }
 console.log("Task started:", response);
 } catch (err) {
 setError(err);
 console.error("Error starting task:", err);
 } finally {
 setLoading(false);
 }
 };

 startTask();
 }, []);

 useEffect(() => {
 if (messages && messages.length > 0) {
 const forecastCompletedMessage = messages.find(

```

```

 (msg) => msg.message === 'Прогноз виконано'
);

 if (forecastCompletedMessage) {
 console.log('Прогноз виконано:', forecastCompletedMessage);
 const startTask = async () => {
 try {
 const response = await PredictionService.analyticsStartTask();
 if (response !== null) {
 setForecasts(response);
 }
 } catch (err) {
 setError(err);
 } finally {
 setLoading(false);
 }
 };

 startTask();
 }
}, [messages]);

const prepareChartData = (forecastData: any, label: string, color: string) => {
 if (!forecastData) return null;

 const real_data = forecastData.real_data.data;
 const timeSteps = forecastData.prediction_time_steps.data;
 const forecastSteps = forecastData.prediction_forecast_steps.data;
 const lowData = forecastData.prediction_low.data[0];
 const medianData = forecastData.prediction_median.data[0];
 const highData = forecastData.prediction_high.data[0];

 const labels = [...timeSteps, ...forecastSteps];
 return {
 labels: labels,
 datasets: [
 {
 label: `${label} (Low)`,
 data: timeSteps.map((val, index) => ({x: val, y: real_data[index]})).concat(forecastSteps.map((val,
index) => ({x: val, y: lowData[index]}))),
 borderColor: 'rgba(75, 192, 192, 0.5)',

```

```

 borderWidth: 2,
 fill: false,
 tension: 0.4,
 showLine: true
 },
 {
 label: `${label} (Median)`,
 data: timeSteps.map((val, index) => ({x: val, y: real_data[index]})).concat(forecastSteps.map((val,
index) => ({x: val, y: medianData[index]}))),
 borderColor: color,
 borderWidth: 2,
 fill: false,
 tension: 0.4,
 showLine: true
 },
 {
 label: `${label} (High)`,
 data: timeSteps.map((val, index) => ({x: val, y: real_data[index]})).concat(forecastSteps.map((val,
index) => ({x: val, y: highData[index]}))),
 borderColor: 'rgba(255, 99, 132, 0.5)',
 borderWidth: 2,
 fill: false,
 tension: 0.4,
 showLine: true
 },
],
 };
};

```

```

const lineOptions = {
 responsive: true,
 plugins: {
 tooltip: {
 callbacks: {
 label: function (tooltipItem: TooltipItem<'line'>) {
 return `Значения: ${tooltipItem.raw}`;
 },
 },
 },
 },
 interaction: {
 intersect: false,
 },
};

```

```

 mode: 'index',
 axis: 'x',
 },
 scales: {
 x: {
 ticks: {
 autoSkip: true,
 maxTicksLimit: 10,
 },
 },
 y: {},
 },
 zoom: {
 wheel: {
 enabled: true,
 modifierKey: 'ctrl',
 },
 pinch: {
 enabled: true,
 },
 drag: {
 enabled: true,
 },
 },
};

```

```

const readingValuesChartData = prepareChartData(
 forecasts?.reading_values,
 'Reading Values',
 'rgba(75, 192, 192, 1)'
);

const kwhTariffsChartData = prepareChartData(
 forecasts?.kwh_tariffs,
 'KWh Tariffs',
 'rgba(99, 34, 98, 1)'
);

const currentAmountsChartData = prepareChartData(
 forecasts?.current_amounts,
 'Current Amounts',
 'rgba(33, 192, 22, 1)'
);

const deltasChartData = prepareChartData(

```

```

forecasts?.deltas,
'Deltas',
'rgba(9, 66, 76, 1)'
);

const barData = forecasts?.costs && {
 labels: forecasts.costs.prediction_time_steps.data,
 datasets: [
 {
 label: "Вартість",
 data: [
 ...forecasts.costs.real_data.data, // Додаємо реальні дані
 ...forecasts.costs.prediction_median.data[0], // Додаємо передбачені дані
],
 backgroundColor: "rgba(75, 192, 192, 0.6)",
 borderColor: "rgba(75, 192, 192, 1)",
 borderWidth: 2,
 },
],
};

const barOptions = {
 responsive: true,
 scales: {
 y: {
 beginAtZero: true,
 },
 },
 plugins: {
 tooltip: {
 callbacks: {
 label: function (tooltipItem) {
 return `Вартість: ${tooltipItem.raw} грн`;
 },
 },
 },
 },
};

if (loading) {
 return (
 <Center height="100vh" bg="gray.100">

```

```

 <Spinner size="xl" color="teal.500" />
 </Center>
);
}

if (error) {
 return (
 <Center height="100vh" bg="gray.100">
 <Box textAlign="center" p={6} boxShadow="md" bg="white" borderRadius="md">
 <Heading as="h1" size="xl" mb={4} color="red.500">
 Помилка
 </Heading>
 <Text fontSize="lg" color="gray.600">
 Неможливо запустити завдання. Спробуйте пізніше.
 </Text>
 </Box>
 </Center>
);
}

return (
 <Box width="100%" textAlign="center" p={6} boxShadow="md" bg="white" borderRadius="md">
 <VStack spacing={8}>
 {!forecasts && (
 <>
 <Heading as="h1" size="xl" mb={4} color="teal.500">
 Завдання запущено
 </Heading>
 <Text fontSize="lg" color="gray.600">
 Завдання успішно запущено. Результат буде доступний пізніше.
 </Text>
 </>
)}
 {messages && messages.length > 0 && (
 <Box mt={4}>
 <Text fontSize="md" color="blue.500">
 Останні повідомлення:
 </Text>
 {messages.map((msg, index) => (
 <Text key={index} fontSize="sm" color="gray.600">
 {msg.message}
 </Text>
))}
 </Box>
)}
 </VStack>
 </Box>
);

```

```

 })
 </>
})
{forecasts && (
 <VStack mt={8} width="100%">
 <Heading as="h2" size="lg" mb={4} color="teal.500">
 Прогнози
 </Heading>

 {barData && (
 <Box width={{ base: "100%", md: "80%" }}>
 <Text mb={2}>Вартість електроенергії по днях</Text>
 <Bar data={barData} options={barOptions} />
 </Box>
)}

 {readingValuesChartData && (
 <Box width={{ base: "100%", md: "80%" }}>
 <Text mb={2}>Прогноз показників лічильника (кВт·год)</Text>
 <Line data={readingValuesChartData} options={lineOptions} />
 </Box>
)}

 {kwhTariffsChartData && (
 <Box width={{ base: "100%", md: "80%" }}>
 <Text mb={2}>Прогноз тарифів (грн/кВт·год)</Text>
 <Line data={kwhTariffsChartData} options={lineOptions} />
 </Box>
)}

 {/* інші графіки так само */}

 {currentAmountsChartData && (
 <Box width={{ base: "100%", md: "80%" }}>
 <Line data={currentAmountsChartData} options={lineOptions} />
 </Box>
)}

 {deltasChartData && (
 <Box width={{ base: "100%", md: "80%" }}>
 <Line data={deltasChartData} options={lineOptions} />
 </Box>
)}
 </VStack>
})
</VStack>

```

```

 </Box>
);
};

export default Prediction;``

``.tsx
import React, { useEffect, useState } from "react";
import { Line } from "react-chartjs-2";
import { Box, Spinner, Heading, Stack, Text } from "@chakra-ui/react";
import { TrendsService } from '../client';
import { TrendData, TrendsResponse } from '../client/models';
import { Chart as ChartJS, registerables } from 'chart.js';
ChartJS.register(...registerables);

const Trends = () => {
 const [state, setState] = useState<{
 data: TrendsResponse | null;
 isLoading: boolean;
 error: string | null;
 }>({
 data: null,
 isLoading: true,
 error: null,
 });

 useEffect(() => {
 const fetchTrends = async () => {
 try {
 const response: TrendsResponse = await TrendsService.analyticsGetTrends();
 setState({
 data: response,
 isLoading: false,
 error: null,
 });
 } catch (error) {
 console.error("Помилка завантаження даних: ", error);
 setState({
 data: null,
 isLoading: false,
 error: "Помилка завантаження даних",
 });
 }
 };
 });
};

```

```

 }
 };

 if (state.isLoading) {
 fetchTrends();
 }
}, [state.isLoading]);

if (state.isLoading) {
 return (
 <Stack align="center" justify="center" h="100vh">
 <Spinner size="xl" />
 <Heading mt={4}>Завантаження...</Heading>
 </Stack>
);
}

if (state.error) {
 return (
 <Box textAlign="center" mt={10}>
 <Heading as="h2" size="lg" color="red.500">
 {state.error}
 </Heading>
 </Box>
);
}

if (!state.data) {
 return (
 <Box textAlign="center" mt={10}>
 <Heading as="h2" size="lg" color="red.500">
 Дані відсутні
 </Heading>
 </Box>
);
}

const chartOptions = {
 responsive: true,
 maintainAspectRatio: false, // Додано для гнучкого розміру
 plugins: {
 legend: {

```

```

 position: "top" as const,
 },
},
};

const labels = state.data.data.map((item) =>
 new Date(item.reading_date).toLocaleDateString("uk-UA")
);

const currentAmountsData = {
 labels,
 datasets: [
 {
 label: "Загальна сума (Current Amount)",
 data: state.data.data.map((item) => item.current_amount),
 borderColor: "blue",
 backgroundColor: "rgba(0, 0, 255, 0.2)",
 fill: true,
 },
],
};

const amountTrendsData = {
 labels,
 datasets: [
 {
 label: "Тренд суми",
 data: state.data.data.map((item) => item.amount_trend),
 borderColor: "green",
 borderDash: [5, 5],
 fill: false,
 },
],
};

const kwhTariffsData = {
 labels,
 datasets: [
 {
 label: "Тариф за кіловат (kWh Tariff)",
 data: state.data.data.map((item) => item.kwh_tariff),
 borderColor: "red",

```

```

 backgroundColor: "rgba(255, 0, 0, 0.2)",
 fill: true,
 },
],
};

```

```

const tariffTrendsData = {
 labels,
 datasets: [
 {
 label: "Тренд тарифу",
 data: state.data.data.map((item) => item.tariff_trend),
 borderColor: "orange",
 borderDash: [5, 5],
 fill: false,
 },
],
};

```

```

return (
 <Box p={6} maxW="container.lg" mx="auto" w="100%">
 <Heading as="h2" size="lg" mb={6} textAlign="center">
 Тренди споживання та тарифів
 </Heading>

```

```

 { /* Chart for Current Amount */ }
 <Box bg="white" p={8} shadow="md" borderRadius="lg" mb={4} h="400px"> { /* Додано висоту
*/ }
 <Text mb={2}>Загальна сума (Current Amount)</Text>
 <Line data={currentAmountsData} options={chartOptions} />
 </Box>

```

```

 { /* Chart for Amount Trend */ }
 <Box bg="white" p={8} shadow="md" borderRadius="lg" mb={4} h="400px"> { /* Додано висоту
*/ }
 <Text mb={2}>Тренд суми</Text>
 <Line data={amountTrendsData} options={chartOptions} />
 </Box>

```

```

 { /* Chart for kWh Tariff */ }
 <Box bg="white" p={8} shadow="md" borderRadius="lg" mb={4} h="400px"> { /* Додано висоту
*/ }

```

```

<Text mb={2}>Тариф за кіловат (kWh Tariff)</Text>
 <Line data={kwhTariffsData} options={chartOptions} />
</Box>

 { /* Chart for Tariff Trend */ }
 <Box bg="white" p={8} shadow="md" borderRadius="lg" h="400px"> { /* Додано висоту */ }
 <Text mb={2}>Тренд тарифу</Text>
 <Line data={tariffTrendsData} options={chartOptions} />
 </Box>
</Box>

);
};

export default Trends;``

``.tsx
import {
 Button,
 Menu,
 MenuButton,
 MenuItem,
 MenuList,
 useDisclosure,
} from "@chakra-ui/react"
import { BsThreeDotsVertical } from "react-icons/bs"
import { FiEdit, FiTrash } from "react-icons/fi"

import type { ItemPublic, UserPublic } from "../../client"
import EditUser from "../Admin/EditUser"
import EditItem from "../Items/EditItem"
import Delete from "../DeleteAlert"

interface ActionsMenuProps {
 type: string
 value: ItemPublic | UserPublic
 disabled?: boolean
}

const ActionsMenu = ({ type, value, disabled }: ActionsMenuProps) => {
 const editUserModal = useDisclosure()
 const deleteModal = useDisclosure()

```

```

return (
 <>
 <Menu>
 <MenuButton
 isDisabled={disabled}
 as={Button}
 rightIcon={<BsThreeDotsVertical />}
 variant="unstyled"
 />
 <MenuList>
 <MenuItem
 onClick={editUserModal.onOpen}
 icon={<FiEdit fontSize="16px" />}
 >
 Edit {type}
 </MenuItem>
 <MenuItem
 onClick={deleteModal.onOpen}
 icon={<FiTrash fontSize="16px" />}
 color="ui.danger"
 >
 Delete {type}
 </MenuItem>
 </MenuList>
 {type === "User" ? (
 <EditUser
 user={value as UserPublic}
 isOpen={editUserModal.isOpen}
 onClose={editUserModal.onClose}
 />
) : (
 <EditItem
 item={value as ItemPublic}
 isOpen={editUserModal.isOpen}
 onClose={editUserModal.onClose}
 />
)}
 <Delete
 type={type}
 id={value.id}
 isOpen={deleteModal.isOpen}
 onClose={deleteModal.onClose}
 />
 </>
)

```

```

 />
 </Menu>
</>
)
}

export default ActionsMenu
```

```tsx
import {
 AlertDialog,
 AlertDialogBody,
 AlertDialogContent,
 AlertDialogFooter,
 AlertDialogHeader,
 AlertDialogOverlay,
 Button,
} from "@chakra-ui/react"
import { useMutation, useQueryClient } from "@tanstack/react-query"
import React from "react"
import { useForm } from "react-hook-form"

import { ItemsService, UsersService } from "../../client"
import useCustomToast from "../../hooks/useCustomToast"

interface DeleteProps {
 type: string
 id: string
 isOpen: boolean
 onClose: () => void
}

const Delete = ({ type, id, isOpen, onClose }: DeleteProps) => {
 const queryClient = useQueryClient()
 const showToast = useCustomToast()
 const cancelRef = React.useRef<HTMLButtonElement | null>(null)
 const {
 handleSubmit,
 formState: { isSubmitting },
 } = useForm()

```

```

const deleteEntity = async (id: string) => {
 if (type === "Item") {
 await ItemsService.itemsDeleteItem({ id: id })
 } else if (type === "User") {
 await UsersService.usersDeleteUser({ userId: id })
 } else {
 throw new Error(`Unexpected type: ${type}`)
 }
}

const mutation = useMutation({
 mutationFn: deleteEntity,
 onSuccess: () => {
 showToast(
 "Success",
 `The ${type.toLowerCase()} was deleted successfully.`,
 "success",
)
 onClose()
 },
 onError: () => {
 showToast(
 "An error occurred.",
 `An error occurred while deleting the ${type.toLowerCase()}.`,
 "error",
)
 },
 onSettled: () => {
 queryClient.invalidateQueries({
 queryKey: [type === "Item" ? "items" : "users"],
 })
 },
})

const onSubmit = async () => {
 mutation.mutate(id)
}

return (
 <>
 <AlertDialog
 isOpen={isOpen}

```

```

 onClose={onClose}
 leastDestructiveRef={cancelRef}
 size={{ base: "sm", md: "md" }}
 isCentered
 >
 <AlertDialogOverlay>
 <AlertDialogContent as="form" onSubmit={handleSubmit(onSubmit)}>
 <AlertDialogHeader>Delete {type}</AlertDialogHeader>

 <AlertDialogBody>
 {type === "User" && (

 All items associated with this user will also be{" "}
 permanently deleted.

)}
 Are you sure? You will not be able to undo this action.
 </AlertDialogBody>

 <AlertDialogFooter gap={3}>
 <Button variant="danger" type="submit" isLoading={isSubmitting}>
 Delete
 </Button>
 <Button
 ref={cancelRef}
 onClick={onClose}
 isDisabled={isSubmitting}
 >
 Cancel
 </Button>
 </AlertDialogFooter>
 </AlertDialogContent>
 </AlertDialogOverlay>
</AlertDialog>
</>
)
}

export default Delete
```
  


```

```tsx

```


```

```

// src/components/Messages.tsx
import { Message } from "../../client";
import {
 IconButton,
 Menu,
 MenuButton,
 MenuItem,
 Box,
 Badge,
 Text,
 HStack,
} from "@chakra-ui/react";
import { FaEnvelope, FaCircle } from "react-icons/fa";
import { Link } from "@tanstack/react-router";
import React from 'react';
import { RootState } from "../../store";
import { useAppSelector, useAppDispatch } from "../../hooks/storeHooks";

const Messages = () => {
 const messages = useAppSelector((state: RootState) => state.sse.messages);

 return (
 <Box position="relative">
 <Menu>
 <MenuButton
 as={IconButton}
 aria-label="Messages"
 icon={<FaEnvelope />}
 variant="ghost"
 data-testid="messages-menu"
 position="relative"
 >
 { /*hasNewMessages && (
 <Badge
 colorScheme="red"
 borderRadius="full"
 position="absolute"
 top="-2px"
 right="-2px"
 fontSize="xs"
 >

```

```

 {messages.filter((msg) => msg.isNew).length}
 </Badge>
)*/}
 </MenuButton>
 <MenuList>
 {
 messages.map((msg: Message, index) => (
 <MenuItem key={index} as={Link} to={`/${messages/${index}}`} >
 <HStack>
 <FaCircle color="red" fontSize="8px" style={{ marginRight: "8px" }} />
 <Text>{msg.message}</Text>
 </HStack>
 </MenuItem>
))
 }
 {messages.length === 0 && (
 <Text p={2} textAlign="center">
 No messages
 </Text>
)}
 </MenuList>
</Menu>
</Box>

);
};

export default Messages;
```

```tsx
import type { ComponentType, ElementType } from "react"

import { Button, Flex, Icon, useDisclosure } from "@chakra-ui/react"
import { FaPlus } from "react-icons/fa"

interface NavbarProps {
 type: string
 addModalAs: ComponentType | ElementType
}

const Navbar = ({ type, addModalAs }: NavbarProps) => {
 const addModal = useDisclosure()

```

```

const AddModal = addModalAs
return (
 <>
 <Flex py={8} gap={4}>
 /* TODO: Complete search functionality */
 /* <InputGroup w={{ base: '100%', md: 'auto' }}>
 <InputLeftElement pointerEvents='none'>
 <Icon as={FaSearch} color='ui.dim' />
 </InputLeftElement>
 <Input type='text' placeholder='Search' fontSize={{ base: 'sm', md: 'inherit' }}
borderRadius='8px' />
 </InputGroup> */
 <Button
 variant="primary"
 gap={1}
 fontSize={{ base: "sm", md: "inherit" }}
 onClick={addModal.onOpen}
 >
 <Icon as={FaPlus} /> Add {type}
 </Button>
 <AddModal isOpen={addModal.isOpen} onClose={addModal.onClose} />
 </Flex>
 </>
)
}

```

```
export default Navbar
```

```
```
```

```
```tsx
```

```

import { Button, Container, Text } from "@chakra-ui/react"
import { Link } from "@tanstack/react-router"

```

```
const NotFound = () => {
```

```
 return (
```

```
 <>
```

```
 <Container
```

```
 h="100vh"
```

```
 alignItems="stretch"
```

```
 justifyContent="center"
```

```
 textAlign="center"
```

```

 maxW="sm"
 centerContent
 >
 <Text
 fontSize="8xl"
 color="ui.main"
 fontWeight="bold"
 lineHeight="1"
 mb={4}
 >
 404
 </Text>
 <Text fontSize="md">Oops!</Text>
 <Text fontSize="md">Page not found.</Text>
 <Button
 as={Link}
 to="/"
 color="ui.main"
 borderColor="ui.main"
 variant="outline"
 mt={4}
 >
 Go back
 </Button>
</Container>
</>
)
}

```

```
export default NotFound
```

```
```
```

```
```tsx
```

```

import {
 Box,
 Drawer,
 DrawerBody,
 DrawerCloseButton,
 DrawerContent,
 DrawerOverlay,
 Flex,
 IconButton,

```

```

Image,
Text,
useColorModeValue,
useDisclosure,
} from "@chakra-ui/react"
import { useQueryClient } from "@tanstack/react-query"
import { FiLogout, FiMenu } from "react-icons/fi"

import Logo from "/assets/images/fastapi-logo.svg"
import type { UserPublic } from "../client"
import useAuth from "../hooks/useAuth"
import SidebarItems from "./SidebarItems"

const Sidebar = () => {
 const queryClient = useQueryClient()
 const bgColor = useColorModeValue("ui.light", "ui.dark")
 const textColor = useColorModeValue("ui.dark", "ui.light")
 const secBgColor = useColorModeValue("ui.secondary", "ui.darkSlate")
 const currentUser = queryClient.getQueryData<UserPublic>(["currentUser"])
 const { isOpen, onOpen, onClose } = useDisclosure()
 const { logout } = useAuth()

 const handleLogout = async () => {
 logout()
 }

 return (
 <>
 { /* Mobile */}
 <IconButton
 onClick={onOpen}
 display={{ base: "flex", md: "none" }}
 aria-label="Open Menu"
 position="absolute"
 fontSize="20px"
 m={4}
 icon={<FiMenu />}
 />
 <Drawer isOpen={isOpen} placement="left" onClose={onClose}>
 <DrawerOverlay />
 <DrawerContent maxW="250px">
 <DrawerCloseButton />

```

```

<DrawerBody py={8}>
 <Flex flexDir="column" justify="space-between" pt={4}>
 <Box>
 { /*<Image src={Logo} alt="logo" p={6} />*/ }
 <SidebarItems onClose={onClose} />
 <Flex
 as="button"
 onClick={handleLogout}
 p={2}
 color="ui.danger"
 fontWeight="bold"
 alignItems="center"
 >
 <FiLogOut />
 <Text ml={2}>Log out</Text>
 </Flex>
 </Box>
 {currentUser?.email && (
 <Text color={textColor} noOfLines={2} fontSize="sm" p={2}>
 Logged in as: {currentUser.email}
 </Text>
)}
 </Flex>
</DrawerBody>
</DrawerContent>
</Drawer>

{ /* Desktop */ }
<Box
 bg={bgColor}
 p={3}
 h="100vh"
 position="sticky"
 top="0"
 display={{ base: "none", md: "flex" }}
>
 <Flex
 flexDir="column"
 justify="space-between"
 bg={secBgColor}
 p={4}
 borderRadius={12}

```

```

 >
 <Box>
 { /*<Image src={Logo} alt="Logo" w="200px" maxW="2xs" p={6} />*/ }
 <SidebarItems />
 </Box>
 {currentUser?.email && (
 <Text
 color={textColor}
 noOfLines={2}
 fontSize="sm"
 p={2}
 maxW="200px"
 >
 Logged in as: {currentUser.email}
 </Text>
)}
 </Flex>
</Box>
</>
)
}

export default Sidebar
```

```

```

```tsx
import React from "react";
import {
 Box,
 Flex,
 Icon,
 Text,
 useColorModeValue,
 Collapse,
 useDisclosure,
} from "@chakra-ui/react";
import { Link } from "@tanstack/react-router";
import {
 FiBriefcase,
 FiHome,
 FiSettings,
 FiDatabase,
}

```

```

 FiClipboard,
 FiActivity,
 FiChevronDown,
 FiChevronUp
 } from "react-icons/fi";

const items = [
 { icon: FiHome, title: "Dashboard", path: "/dashboard" },
 { icon: FiBriefcase, title: "Items", path: "/items" },
 { icon: FiSettings, title: "User Settings", path: "/settings" },
 {
 icon: FiClipboard,
 title: "Reports",
 path: "/reports",
 subItems: [
 { title: "Generate And Export Reports", path: "/reports/generate_and_export" },
 { title: "Automatic Reports", path: "/reports/automatic" },
],
 },
 {
 icon: FiActivity,
 title: "Analytics",
 path: "/analytics",
 subItems: [
 { title: "Trends", path: "/analytics/trends" },
 { title: "Prediction", path: "/analytics/prediction" },
 { title: "Comparison", path: "/analytics/comparison" },
],
 },
];

interface SidebarItemsProps {
 onClose?: () => void;
}

const SidebarItems = ({ onClose }: SidebarItemsProps) => {
 const textColor = useColorModeValue("ui.main", "ui.light");
 const bgActive = useColorModeValue("#E2E8F0", "#4A5568");

 // Використовуємо useDisclosure для конкретних пунктів меню
 const addDataDisclosure = useDisclosure();
 const reportsDisclosure = useDisclosure();

```

```

const analyticsDisclosure = useDisclosure();

const listItems = items.map(({ icon, title, path, subItems }) => {
 let disclosure = null;
 if (title == "Add Data"){
 disclosure = addDataDisclosure;
 }
 else if (title == "Reports"){
 disclosure = reportsDisclosure;
 }
 else if (title == "Analytics"){
 disclosure = analyticsDisclosure;
 }

 return (
 <Box key={title}>
 <Flex
 as={Link}
 to={path}
 w="100%"
 p={2}
 onClick={() => (subItems ? disclosure?.onToggle() : onClose?.())}
 color={textColor}
 cursor="pointer"
 activeProps={{
 style: {
 background: bgActive,
 borderRadius: "12px",
 },
 }}
 >
 <Icon as={icon} alignSelf="center" />
 <Text ml={2} whiteSpace="nowrap">
 {title}
 </Text>
 {subItems && (
 <Flex ml="auto" align="center" justify="center" w="auto" h="100%">
 <Icon as={disclosure?.isOpen ? FiChevronUp : FiChevronDown} ml="auto" />
 </Flex>
)}
 </Flex>
 </Box>
)
})

```

```

 {subItems && (
 <Collapse in={disclosure?.isOpen}>
 <Box pl={6} pt={2} pb={2}>
 {subItems.map(({ title: subTitle, path }) => (
 <Flex
 as={Link}
 to={path}
 w="100%"
 p={2}
 key={subTitle}
 color={textColor}
 onClick={onClose}
 activeProps={{
 style: {
 background: bgActive,
 borderRadius: "12px",
 },
 }}
 >
 <Text ml={2} whiteSpace="nowrap">
 {subTitle}
 </Text>
 </Flex>
)))}
 </Box>
 </Collapse>
)}
 </Box>
);
});

return <Box>{listItems}</Box>;
};

export default SidebarItems;
```
  


```

`.tsx

import {
 Box,
 IconButton,

```


```

```

Menu,
MenuButton,
MenuItem,
MenuList,
HStack
} from "@chakra-ui/react"
import { Link } from "@tanstack/react-router"
import { FaUserAstronaut } from "react-icons/fa"
import { FiLogOut, FiUser, FiHome } from "react-icons/fi"
import Messages from "./Messages"

import useAuth from "../../hooks/useAuth"

const UserMenu = () => {
  const { logout } = useAuth()

  const handleLogout = async () => {
    logout()
  }

  return (
    <>
      { /* Desktop */ }
      <Box
        display={{ base: "none", md: "block" }}
        position="fixed"
        top={4}
        right={4}
      >
        <HStack>
          <Messages/>
          <Menu>
            <MenuButton
              as={IconButton}
              aria-label="Options"
              icon={<FaUserAstronaut color="white" fontSize="18px" />}
              bg="ui.main"
              isRound
              data-testid="user-menu"
            />
            <MenuList>
              <MenuItem icon={<FiHome fontSize="18px" />} as={Link} to="/">

```

```

        Homepage
      </MenuItem>
      <MenuItem icon={<FiUser fontSize="18px" />} as={Link} to="settings">
        My profile
      </MenuItem>
      <MenuItem
        icon={<FiLogout fontSize="18px" />}
        onClick={handleLogout}
        color="ui.danger"
        fontWeight="bold"
      >
        Log out
      </MenuItem>
    </MenuItem>
  </Menu>
</HStack>
</Box>
</>
)
}

```

```
export default UserMenu
```

```
```
```

```
```tsx
```

```

import {
  Button,
  FormControl,
  FormErrorMessage,
  FormLabel,
  Input,
  Modal,
  ModalBody,
  ModalCloseButton,
  ModalContent,
  ModalFooter,
  ModalHeader,
  ModalOverlay,
} from "@chakra-ui/react"
import { useMutation, useQueryClient } from "@tanstack/react-query"
import { type SubmitHandler, useForm } from "react-hook-form"

```

```

import { type ApiError, type ItemCreate, ItemsService } from "../../client"
import useCustomToast from "../../hooks/useCustomToast"
import { handleError } from "../../utils"

interface AddItemProps {
  isOpen: boolean
  onClose: () => void
}

const AddItem = ({ isOpen, onClose }: AddItemProps) => {
  const queryClient = useQueryClient()
  const showToast = useCustomToast()
  const {
    register,
    handleSubmit,
    reset,
    formState: { errors, isSubmitting },
  } = useForm<ItemCreate>({
    mode: "onBlur",
    criteriaMode: "all",
    defaultValues: {
      title: "",
      description: "",
    },
  })

  const mutation = useMutation({
    mutationFn: (data: ItemCreate) =>
      ItemsService.itemsCreateItem({ requestBody: data }),
    onSuccess: () => {
      showToast("Success!", "Item created successfully.", "success")
      reset()
      onClose()
    },
    onError: (err: ApiError) => {
      handleError(err, showToast)
    },
    onSettled: () => {
      queryClient.invalidateQueries({ queryKey: ["items"] })
    },
  })
}

```

```

const onSubmit: SubmitHandler<ItemCreate> = (data) => {
  mutation.mutate(data)
}

return (
  <>
    <Modal
      isOpen={isOpen}
      onClose={onClose}
      size={{ base: "sm", md: "md" }}
      isCentered
    >
      <ModalOverlay />
      <ModalContent as="form" onSubmit={handleSubmit(onSubmit)}>
        <ModalHeader>Add Item</ModalHeader>
        <ModalCloseButton />

        <ModalBody pb={6}>

          <FormControl isRequired isInvalid={!errors.title}>
            <FormLabel htmlFor="title">Title</FormLabel>
            <Input
              id="title"
              {...register("title", {
                required: "Title is required.",
              })}
              placeholder="Title"
              type="text"
            />
            {errors.title && (
              <FormErrorMessage>{errors.title.message}</FormErrorMessage>
            )}
          </FormControl>

          <FormControl mt={4}>
            <FormLabel htmlFor="description">Description</FormLabel>
            <Input
              id="description"
              {...register("description")}
              placeholder="Description"
              type="text"
            />

```

```
</FormControl>
```

```
<FormControl isRequired isInvalid={!errors.meter_identifier}>
  <FormLabel htmlFor="meter_identifier">Meter Identifier</FormLabel>
  <Input
    id="meter_identifier"
    {...register("meter_identifier", {
      required: "Meter Identifier is required.",
    })}
    placeholder="Meter Identifier"
    type="text"
  />
  {errors.meter_identifier && (
    <FormErrorMessage>{errors.meter_identifier.message}</FormErrorMessage>
  )}
</FormControl>
```

```
<FormControl isRequired isInvalid={!errors.reading_value}>
  <FormLabel htmlFor="reading_value">Reading Value</FormLabel>
  <Input
    id="reading_value"
    {...register("reading_value", {
      required: "Reading Value is required.",
    })}
    placeholder="Reading Value"
    type="text"
  />
  {errors.reading_value && (
    <FormErrorMessage>{errors.reading_value.message}</FormErrorMessage>
  )}
</FormControl>
```

```
<FormControl isRequired isInvalid={!errors.kwh_tariff}>
  <FormLabel htmlFor="kwh_tariff">Kilowatt-Hour Tariff</FormLabel>
  <Input
    id="kwh_tariff"
    {...register("kwh_tariff", {
      required: "Kilowatt-Hour Tariff is required.",
    })}
    placeholder="Kilowatt-Hour Tariff"
    type="text"
  />
```

```

    {errors.kwh_tariff && (
      <FormErrorMessage>{errors.kwh_tariff.message}</FormErrorMessage>
    )}
  </FormControl>

  <FormControl isRequired isInvalid={!errors.current_amount}>
    <FormLabel htmlFor="current_amount">Current Amount</FormLabel>
    <Input
      id="current_amount"
      {...register("current_amount", {
        required: "Current Amount is required.",
      })}
      placeholder="Current Amount"
      type="text"
    />
    {errors.current_amount && (
      <FormErrorMessage>{errors.current_amount.message}</FormErrorMessage>
    )}
  </FormControl>

</ModalBody>

<ModalFooter gap={3}>
  <Button variant="primary" type="submit" isLoading={isSubmitting}>
    Save
  </Button>
  <Button onClick={onClose}>Cancel</Button>
</ModalFooter>
</ModalContent>
</Modal>
</>
)
}

export default AddItem
```



```

```.tsx
import {
 Button,
 FormControl,
 FormErrorMessage,

```


```

```

    FormLabel,
    Input,
    Modal,
    ModalBody,
    ModalCloseButton,
    ModalContent,
    ModalFooter,
    ModalHeader,
    ModalOverlay,
  } from "@chakra-ui/react"
import { useMutation, useQueryClient } from "@tanstack/react-query"
import { type SubmitHandler, useForm } from "react-hook-form"

import {
  type ApiError,
  type ItemPublic,
  type ItemUpdate,
  ItemsService,
} from "../../client"
import useCustomToast from "../../hooks/useCustomToast"
import { handleError } from "../../utils"

interface EditItemProps {
  item: ItemPublic
  isOpen: boolean
  onClose: () => void
}

const EditItem = ({ item, isOpen, onClose }: EditItemProps) => {
  const queryClient = useQueryClient()
  const showToast = useCustomToast()
  const {
    register,
    handleSubmit,
    reset,
    formState: { isSubmitting, errors, isDirty },
  } = useForm<ItemUpdate>({
    mode: "onBlur",
    criteriaMode: "all",
    defaultValues: item,
  })

```

```

const mutation = useMutation({
  mutationFn: (data: ItemUpdate) =>
    ItemsService.itemsUpdateItem({ id: item.id, requestBody: data }),
  onSuccess: () => {
    showToast("Success!", "Item updated successfully.", "success")
    onClose()
  },
  onError: (err: ApiError) => {
    handleError(err, showToast)
  },
  onSettled: () => {
    queryClient.invalidateQueries({ queryKey: ["items"] })
  },
})

const onSubmit: SubmitHandler<ItemUpdate> = async (data) => {
  mutation.mutate(data)
}

const onCancel = () => {
  reset()
  onClose()
}

return (
  <>
    <Modal
      isOpen={isOpen}
      onClose={onClose}
      size={{ base: "sm", md: "md" }}
      isCentered
    >
      <ModalOverlay />
      <ModalContent as="form" onSubmit={handleSubmit(onSubmit)}>
        <ModalHeader>Edit Item</ModalHeader>
        <ModalCloseButton />
        <ModalBody pb={6}>

          <FormControl isInvalid={!!errors.title}>
            <FormLabel htmlFor="title">Title</FormLabel>
            <Input
              id="title"

```

```

    {...register("title", {
      required: "Title is required",
    })}
    type="text"
  />

  {errors.title && (
    <FormErrorMessage>{errors.title.message}</FormErrorMessage>
  )}
</FormControl>

<FormControl mt={4}>
  <FormLabel htmlFor="description">Description</FormLabel>
  <Input
    id="description"
    {...register("description")}
    placeholder="Description"
    type="text"
  />
</FormControl>

<FormControl isRequired isInvalid={!errors.meter_identifier}>
  <FormLabel htmlFor="meter_identifier">Meter Identifier</FormLabel>
  <Input
    id="meter_identifier"
    {...register("meter_identifier", {
      required: "Meter Identifier is required.",
    })}
    placeholder="Meter Identifier"
    type="text"
  />
  {errors.meter_identifier && (
    <FormErrorMessage>{errors.meter_identifier.message}</FormErrorMessage>
  )}
</FormControl>

<FormControl isRequired isInvalid={!errors.reading_value}>
  <FormLabel htmlFor="reading_value">Reading Value</FormLabel>
  <Input
    id="reading_value"
    {...register("reading_value", {
      required: "Reading Value is required.",
    })}
  />
</FormControl>

```

```

        placeholder="Reading Value"
        type="text"
      />
      {errors.reading_value && (
        <FormErrorMessage>{errors.reading_value.message}</FormErrorMessage>
      )}
    </FormControl>

    <FormControl isRequired isInvalid={!errors.kwh_tariff}>
      <FormLabel htmlFor="kwh_tariff">Kilowatt-Hour Tariff</FormLabel>
      <Input
        id="kwh_tariff"
        {...register("kwh_tariff", {
          required: "Kilowatt-Hour Tariff is required.",
        })}
        placeholder="Kilowatt-Hour Tariff"
        type="text"
      />
      {errors.kwh_tariff && (
        <FormErrorMessage>{errors.kwh_tariff.message}</FormErrorMessage>
      )}
    </FormControl>

    <FormControl isRequired isInvalid={!errors.current_amount}>
      <FormLabel htmlFor="current_amount">Current Amount</FormLabel>
      <Input
        id="current_amount"
        {...register("current_amount", {
          required: "Current Amount is required.",
        })}
        placeholder="Current Amount"
        type="text"
      />
      {errors.current_amount && (
        <FormErrorMessage>{errors.current_amount.message}</FormErrorMessage>
      )}
    </FormControl>

  </ModalBody>
  <ModalFooter gap={3}>
    <Button
      variant="primary"

```

```

        type="submit"
        isLoading={isSubmitting}
        isDisabled={!isDirty}
      >
        Save
      </Button>
      <Button onClick={onCancel}>Cancel</Button>
    </ModalFooter>
  </ModalContent>
</Modal>
</>
)
}

export default EditItem
```

```tsx
import React, { useState } from 'react';
import {
  Box,
  Heading,
  FormControl,
  FormLabel,
  Select,
  Button,
  VStack,
  Text,
  Alert,
  AlertIcon,
  AlertTitle,
  AlertDescription,
  Checkbox,
  Input,
  Stack,
  RadioGroup,
  Radio,
} from '@chakra-ui/react';

const AutomaticReportsSettings = () => {
  const [reportEnabled, setReportEnabled] = useState(false);
  const [reportPeriod, setReportPeriod] = useState("");

```

```

const [email, setEmail] = useState("");

const [namer, setNamer] = useState("");
const [reportContent, setReportContent] = useState({
  totalConsumption: true,
  averageDailyConsumption: true,
  cost: true,
  charts: true,
});
const [isSubmitting, setIsSubmitting] = useState(false);
const [showSuccess, setShowSuccess] = useState(false);
const [showError, setShowError] = useState(false);

const handlePeriodChange = (event) => {
  setReportPeriod(event.target.value);
};

const handleEmailChange = (event) => {
  setEmail(event.target.value);
};

const handleNamerChange = (event) => {
  setNamer(event.target.value);
};

const handleContentChange = (event) => {
  setReportContent({
    ...reportContent,
    [event.target.name]: event.target.checked,
  });
};

const handleReportToggle = () => {
  setReportEnabled(!reportEnabled);
};

const handleSubmit = (event) => {
  event.preventDefault();
  setIsSubmitting(true);

  // Simulate saving settings
  setTimeout(() => {

```

```

setIsSubmitting(false);
if (
  reportEnabled &&
  (!reportPeriod || !email || !reportTime)
) {
  setShowError(true);
} else {
  setShowSuccess(true);
}

// Automatically hide the success/error message after 5 seconds
setTimeout(() => {
  setShowSuccess(false);
  setShowError(false);
}, 5000);
}, 1000);
};

return (
  <Box p={5} width="full" maxWidth="md" margin="auto" mt={10}>
    <Heading as="h1" size="xl" mb={6} textAlign="center" color="teal.500">
      Налаштування автоматичних звітів
    </Heading>

    <Alert
      status={showSuccess ? "success" : "error"}
      display={showSuccess || showError ? "block" : "none"}
      borderRadius="md"
    >
      <AlertIcon />
      <AlertTitle>{showSuccess ? "Успіх!" : "Помилка!"}</AlertTitle>
      <AlertDescription>
        {showSuccess
          ? "Налаштування успішно збережені."
          : "Будь ласка, заповніть всі обов'язкові поля."}
      </AlertDescription>
    </Alert>

    <VStack spacing={5} as="form" onSubmit={handleSubmit}>
      <FormControl>
        <Checkbox
          isChecked={reportEnabled}

```

```

onChange={handleReportToggle}
colorScheme="teal"
size="lg"
>
  <Text fontSize="xl">Увімкнути автоматичні звіти</Text>
</Checkbox>
</FormControl>

{reportEnabled && (
  <>
    <FormControl isRequired>
      <FormLabel htmlFor="reportPeriod">Періодичність:</FormLabel>
      <Select
        id="reportPeriod"
        placeholder="Оберіть період"
        value={reportPeriod}
        onChange={handlePeriodChange}
        size="lg"
      >
        <option value="monthly">Щомісяця</option>
        <option value="quarterly">Щокварталу</option>
      </Select>
    </FormControl>

    <FormControl isRequired>
      <FormLabel htmlFor="email">Email:</FormLabel>
      <Input
        id="email"
        type="email"
        placeholder="Введіть email"
        value={email}
        onChange={handleEmailChange}
        size="lg"
      />
    </FormControl>

    <FormControl isRequired>
      <FormLabel htmlFor="email">Назва</FormLabel>
      <Input
        id="namer"
        type="text"
        placeholder="Введіть назву"

```

```

    value={ namer}
    onChange={handleNamerChange}
    size="lg"
  />
</FormControl>

<FormControl>
  <FormLabel>Вміст звіту:</FormLabel>
  <VStack align="start">
    <Checkbox
      name="totalConsumption"
      isChecked={reportContent.totalConsumption}
      onChange={handleContentChange}
      colorScheme="teal"
      size="lg"
    >
      Загальне споживання
    </Checkbox>
    <Checkbox
      name="averageDailyConsumption"
      isChecked={reportContent.averageDailyConsumption}
      onChange={handleContentChange}
      colorScheme="teal"
      size="lg"
    >
      Середньодобове споживання
    </Checkbox>
    <Checkbox
      name="cost"
      isChecked={reportContent.cost}
      onChange={handleContentChange}
      colorScheme="teal"
      size="lg"
    >
      Витрати
    </Checkbox>
    <Checkbox
      name="charts"
      isChecked={reportContent.charts}
      onChange={handleContentChange}
      colorScheme="teal"
      size="lg"
    >

```

```

    >
    

Графіки


  </Checkbox>
  <Checkbox
    name="charts"
    isChecked={reportContent.charts}
    onChange={handleContentChange}
    colorScheme="teal"
    size="lg"
  >
    

Порівняння


  </Checkbox>
</VStack>
</FormControl>
</>
)}

<Button
  type="submit"
  colorScheme="teal"
  size="lg"
  isLoading={isSubmitting}
  width="full"
  mt={4}
>
  Зберегти налаштування
</Button>
</VStack>
</Box>
);
};

export default AutomaticReportsSettings;``

``.tsx
import React, { useState, useEffect } from 'react';
import {
  Box,
  Button,
  FormControl,
  FormLabel,
  VStack,

```

```

Select,
Spinner,
Heading,
Center,
Card,
CardBody,
CardHeader,
useToast,
Input,
Text
} from '@chakra-ui/react';
import { ReportService } from '../client';

const Generator = () => {
  const [startDate, setStartDate] = useState("");
  const [endDate, setEndDate] = useState("");
  const [category, setCategory] = useState("");
  const [reportId, setReportId] = useState<string | null>(null);
  const [reportContent, setReportContent] = useState<string | null>(null);
  const [isLoading, setIsLoading] = useState<boolean>(false);
  const toast = useToast();

  const handleStartDateChange = (event) => setStartDate(event.target.value);
  const handleEndDateChange = (event) => setEndDate(event.target.value);
  const handleCategoryChange = (event) => setCategory(event.target.value);

  const generateReport = async () => {
    try {
      const response = await ReportService.reportsGenerateReport({
        requestBody: {
          startDate,
          endDate,
          category,
        },
      });

      if (response && typeof response === 'object' && 'report_id' in response) {
        setReportId(response.report_id);

        toast({
          title: 'Звіт згенеровано',
          status: 'success',

```

```

        duration: 5000,
        isClosable: true,
    });
} else {
    throw new Error('Помилка при генерації звіту: Некоректна відповідь від сервера');
}
} catch (error) {
    console.error(error);
    setReportId(null);
    toast({
        title: 'Помилка',
        description: error.message || 'Помилка при генерації звіту',
        status: 'error',
        duration: 5000,
        isClosable: true,
    });
}
};

useEffect(() => {
    const fetchReport = async () => {
        if (reportId) {
            setIsLoading(true);
            try {
                const response = await ReportService.reportsViewReport({ reportId });

                // Створюємо data URL з отриманого вмісту
                const dataUrl = `data:application/pdf;base64,${btoa(unescape(encodeURIComponent(response)))}`;
                setReportContent(dataUrl);
                setIsLoading(false);
            } catch (error) {
                console.error('Error fetching report:', error);
                setIsLoading(false);
                toast({
                    title: 'Помилка',
                    description: 'Не вдалося завантажити звіт',
                    status: 'error',
                    duration: 5000,
                    isClosable: true,
                });
            }
        }
    };
}

```

```

    };
    fetchReport();
  }, [reportId]);

  return (
    <Box p={5} width="100%" bg="gray.700" color="gray.200"> {/* Змінено колір фону та тексту */}
      <Center mb={4}>
        <Heading color="teal.300">Генератор звітів</Heading> {/* Змінено колір заголовка */}
      </Center>

      <Card maxW="4xl" mx="auto" bg="gray.600" shadow="lg" rounded="lg"> {/* Змінено колір фону
картки */}
        <CardHeader>
          <Heading size="md" color="teal.300">Форма для генерації звіту</Heading> {/* Змінено колір
заголовка */}
        </CardHeader>
        <CardBody>
          <VStack spacing={4} align="stretch">
            <FormControl>
              <FormLabel htmlFor="startDate" color="gray.200">Початкова дата</FormLabel> {/* Змінено
колір тексту */}
              <Input
                type="date"
                id="startDate"
                value={startDate}
                onChange={handleStartDateChange}
                color="gray.800"
                bg="gray.200"
              />
            </FormControl>

            <FormControl>
              <FormLabel htmlFor="endDate" color="gray.200">Кінцева дата</FormLabel> {/* Змінено ко-
лір тексту */}
              <Input
                type="date"
                id="endDate"
                value={endDate}
                onChange={handleEndDateChange}
                color="gray.800"
                bg="gray.200"
              />
            </FormControl>
          </VStack>
        </CardBody>
      </Card>
    </Box>
  );

```

```

</FormControl>

<FormControl>
  <FormLabel htmlFor="category" color="gray.200">Категорія</FormLabel> {/* Змінено колір
тексту */}
  <Select
    id="category"
    value={category}
    onChange={handleCategoryChange}
    placeholder="Виберіть категорію"
    color="gray.800"
    bg="gray.200"
  >
    <option value="food" style={{ backgroundColor: '#A0AEC0', color: '#000' }}>Їжа</option>
    <option value="transport" style={{ backgroundColor: '#A0AEC0', color: '#000' }}>Транс-
порт</option>
    <option value="utilities" style={{ backgroundColor: '#A0AEC0', color: '#000' }}>Комунальні
послуги</option>
    {/* Додайте інші категорії */}
  </Select>
</FormControl>

<Button colorScheme="blue" onClick={generateReport} width="full">
  Згенерувати звіт
</Button>
</VStack>
</CardBody>
</Card>

{isLoading && (
  <Center mt={5}>
    <Spinner size="lg" color="teal.500" />
  </Center>
)}

{!isLoading && reportContent && (
  <Card maxW="4xl" mx="auto" mt={5} bg="gray.600" shadow="lg" rounded="lg"> {/* Змінено ко-
лір фону картки */}
    <CardHeader>
      <Heading size="md" color="teal.300">Згенерований звіт</Heading> {/* Змінено колір заголо-
вка */}
    </CardHeader>

```

```

    <CardBody>
      <Box width="100%">
        <iframe src={reportContent} width="100%" height="600px" title="Report" />
      </Box>
    </CardBody>
  </Card>
)}
{reportId && (
  <Text mt={2} textAlign="center">
    Звіт згенеровано. ID звіту: {reportId}
  </Text>
)}
</Box>
);
};

```

```
export default Generator;``
```

```
```.tsx
```

```

import {
 Badge,
 Container,
 Heading,
 Radio,
 RadioGroup,
 Stack,
 useColorMode,
} from "@chakra-ui/react"

```

```

const Appearance = () => {
 const { colorMode, toggleColorMode } = useColorMode()

 return (
 <>
 <Container maxW="full">
 <Heading size="sm" py={4}>
 Appearance
 </Heading>
 <RadioGroup onChange={toggleColorMode} value={colorMode}>
 <Stack>
 {/* TODO: Add system default option */}
 <Radio value="light" colorScheme="teal">

```

```

 Light Mode
 <Badge ml="1" colorScheme="teal">
 Default
 </Badge>
 </Radio>
 <Radio value="dark" colorScheme="teal">
 Dark Mode
 </Radio>
</Stack>
</RadioGroup>
</Container>
</>
)
}
export default Appearance
```

```tsx
import {
 Box,
 Button,
 Container,
 FormControl,
 FormErrorMessage,
 FormLabel,
 Heading,
 Input,
 useColorModeValue,
} from "@chakra-ui/react"
import { useMutation } from "@tanstack/react-query"
import { type SubmitHandler, useForm } from "react-hook-form"

import { type ApiError, type UpdatePassword, UsersService } from "../../client"
import useCustomToast from "../../hooks/useCustomToast"
import { confirmPasswordRules, handleError, passwordRules } from "../../utils"

interface UpdatePasswordForm extends UpdatePassword {
 confirm_password: string
}

const ChangePassword = () => {
 const color = useColorModeValue("inherit", "ui.light")

```

```

const showToast = useCustomToast()

const {
 register,
 handleSubmit,
 reset,
 getValues,
 formState: { errors, isSubmitting },
} = useForm<UpdatePasswordForm>({
 mode: "onBlur",
 criteriaMode: "all",
})

const mutation = useMutation({
 mutationFn: (data: UpdatePassword) =>
 UsersService.usersUpdatePasswordMe({ requestBody: data }),
 onSuccess: () => {
 showToast("Success!", "Password updated successfully.", "success")
 reset()
 },
 onError: (err: ApiError) => {
 handleError(err, showToast)
 },
})

const onSubmit: SubmitHandler<UpdatePasswordForm> = async (data) => {
 mutation.mutate(data)
}

return (
 <>
 <Container maxW="full">
 <Heading size="sm" py={4}>
 Change Password
 </Heading>
 <Box
 w={{ sm: "full", md: "50%" }}
 as="form"
 onSubmit={handleSubmit(onSubmit)}
 >
 <FormControl isRequired isValid={!errors.current_password}>
 <FormLabel color={color} htmlFor="current_password">
 Current Password

```

```

</FormLabel>
<Input
 id="current_password"
 {...register("current_password")}
 placeholder="Password"
 type="password"
 w="auto"
/>
{errors.current_password && (
 <FormErrorMessage>
 {errors.current_password.message}
 </FormErrorMessage>
)}
</FormControl>
<FormControl mt={4} isRequired isValid={!errors.new_password}>
 <FormLabel htmlFor="password">Set Password</FormLabel>
 <Input
 id="password"
 {...register("new_password", passwordRules())}
 placeholder="Password"
 type="password"
 w="auto"
 />
 {errors.new_password && (
 <FormErrorMessage>{errors.new_password.message}</FormErrorMessage>
)}
</FormControl>
<FormControl mt={4} isRequired isValid={!errors.confirm_password}>
 <FormLabel htmlFor="confirm_password">Confirm Password</FormLabel>
 <Input
 id="confirm_password"
 {...register("confirm_password", confirmPasswordRules(getValues))}
 placeholder="Password"
 type="password"
 w="auto"
 />
 {errors.confirm_password && (
 <FormErrorMessage>
 {errors.confirm_password.message}
 </FormErrorMessage>
)}
</FormControl>

```

```

 <Button
 variant="primary"
 mt={4}
 type="submit"
 isLoading={isSubmitting}
 >
 Save
 </Button>
 </Box>
 </Container>
 </>
)
}
export default ChangePassword
```

```tsx
import {
 Button,
 Container,
 Heading,
 Text,
 useDisclosure,
} from "@chakra-ui/react"

import DeleteConfirmation from "../DeleteConfirmation"

const DeleteAccount = () => {
 const confirmationModal = useDisclosure()

 return (
 <>
 <Container maxW="full">
 <Heading size="sm" py={4}>
 Delete Account
 </Heading>
 <Text>
 Permanently delete your data and everything associated with your
 account.
 </Text>
 <Button variant="danger" mt={4} onClick={confirmationModal.onOpen}>
 Delete
 </Button>
 </Container>
 </>
)
}

```

```

 </Button>
 <DeleteConfirmation
 isOpen={confirmationModal.isOpen}
 onClose={confirmationModal.onClose}
 />
 </Container>
</>
)
}
export default DeleteAccount
```

```tsx
import {
 AlertDialog,
 AlertDialogBody,
 AlertDialogContent,
 AlertDialogFooter,
 AlertDialogHeader,
 AlertDialogOverlay,
 Button,
} from "@chakra-ui/react"
import { useMutation, useQueryClient } from "@tanstack/react-query"
import React from "react"
import { useForm } from "react-hook-form"

import { type ApiError, UsersService } from "../../client"
import useAuth from "../../hooks/useAuth"
import useCustomToast from "../../hooks/useCustomToast"
import { handleError } from "../../utils"

interface DeleteProps {
 isOpen: boolean
 onClose: () => void
}

const DeleteConfirmation = ({ isOpen, onClose }: DeleteProps) => {
 const queryClient = useQueryClient()
 const showToast = useCustomToast()
 const cancelRef = React.useRef<HTMLButtonElement | null>(null)
 const {
 handleSubmit,

```

```

 formState: { isSubmitting },
 } = useForm()
 const { logout } = useAuth()

 const mutation = useMutation({
 mutationFn: () => UsersService.usersDeleteUserMe(),
 onSuccess: () => {
 showToast(
 "Success",
 "Your account has been successfully deleted.",
 "success",
)
 logout()
 onClose()
 },
 onError: (err: ApiError) => {
 handleError(err, showToast)
 },
 onSettled: () => {
 queryClient.invalidateQueries({ queryKey: ["currentUser"] })
 },
 })

 const onSubmit = async () => {
 mutation.mutate()
 }

 return (
 <>
 <AlertDialog
 isOpen={isOpen}
 onClose={onClose}
 leastDestructiveRef={cancelRef}
 size={{ base: "sm", md: "md" }}
 isCentered
 >
 <AlertDialogOverlay>
 <AlertDialogContent as="form" onSubmit={handleSubmit(onSubmit)}>
 <AlertDialogHeader>Confirmation Required</AlertDialogHeader>

 <AlertDialogBody>
 All your account data will be{" "}

```

```

 permanently deleted. If you are sure, please
 click "Confirm" to proceed. This action cannot be
 undone.
 </AlertDialogBody>

 <AlertDialogFooter gap={3}>
 <Button variant="danger" type="submit" isLoading={isSubmitting}>
 Confirm
 </Button>
 <Button
 ref={cancelRef}
 onClick={onClose}
 isDisabled={isSubmitting}
 >
 Cancel
 </Button>
 </AlertDialogFooter>
</AlertDialogContent>
</AlertDialogOverlay>
</AlertDialog>
</>
)
}

export default DeleteConfirmation
...

```.tsx
import {
    Box,
    Button,
    Container,
    Flex,
    FormControl,
    FormErrorMessage,
    FormLabel,
    Heading,
    Input,
    Text,
    useColorModeValue,
} from "@chakra-ui/react"
import { useMutation, useQueryClient } from "@tanstack/react-query"

```

```

import { useState } from "react"
import { type SubmitHandler, useForm } from "react-hook-form"

import {
  type ApiError,
  type UserPublic,
  type UserUpdateMe,
  UsersService,
} from "../../client"
import useAuth from "../../hooks/useAuth"
import useCustomToast from "../../hooks/useCustomToast"
import { emailPattern, handleError } from "../../utils"

const UserInformation = () => {
  const queryClient = useQueryClient()
  const color = useColorModeValue("inherit", "ui.light")
  const showToast = useCustomToast()
  const [editMode, setEditMode] = useState(false)
  const { user: currentUser } = useAuth()
  const {
    register,
    handleSubmit,
    reset,
    getValues,
    formState: { isSubmitting, errors, isDirty },
  } = useForm<UserPublic>({
    mode: "onBlur",
    criteriaMode: "all",
    defaultValues: {
      full_name: currentUser?.full_name,
      email: currentUser?.email,
    },
  })

  const toggleEditMode = () => {
    setEditMode(!editMode)
  }

  const mutation = useMutation({
    mutationFn: (data: UserUpdateMe) =>
      UsersService.usersUpdateUserMe({ requestBody: data }),
    onSuccess: () => {

```

```

    showToast("Success!", "User updated successfully.", "success")
  },
  onError: (err: ApiError) => {
    handleError(err, showToast)
  },
  onSettled: () => {
    queryClient.invalidateQueries()
  },
})

const onSubmit: SubmitHandler<UserUpdateMe> = async (data) => {
  mutation.mutate(data)
}

const onCancel = () => {
  reset()
  toggleEditMode()
}

return (
  <>
    <Container maxW="full">
      <Heading size="sm" py={4}>
        User Information
      </Heading>
      <Box
        w={{ sm: "full", md: "50%" }}
        as="form"
        onSubmit={handleSubmit(onSubmit)}
      >
        <FormControl>
          <FormLabel color={color} htmlFor="name">
            Full name
          </FormLabel>
          {editMode ? (
            <Input
              id="name"
              {...register("full_name", { maxLength: 30 })}
              type="text"
              size="md"
              w="auto"
            />

```

```

): (
  <Text
    size="md"
    py={2}
    color={!currentUser?.full_name ? "ui.dim" : "inherit"}
    isTruncated
    maxWidth="250px"
  >
    {currentUser?.full_name || "N/A"}
  </Text>
)}
</FormControl>
<FormControl mt={4} isInvalid={!errors.email}>
  <FormLabel color={color} htmlFor="email">
    Email
  </FormLabel>
  {editMode ? (
    <Input
      id="email"
      {...register("email", {
        required: "Email is required",
        pattern: emailPattern,
      })}
      type="email"
      size="md"
      w="auto"
    />
  ) : (
    <Text size="md" py={2} isTruncated maxWidth="250px">
      {currentUser?.email}
    </Text>
  )}
  {errors.email && (
    <FormErrorMessage>{errors.email.message}</FormErrorMessage>
  )}
</FormControl>
<Flex mt={4} gap={3}>
  <Button
    variant="primary"
    onClick={toggleEditMode}
    type={editMode ? "button" : "submit"}
    isLoading={editMode ? isSubmitting : false}
  >

```

```

        isDisabled={editMode ? !isDirty || !getValues("email") : false}
      >
        {editMode ? "Save" : "Edit"}
      </Button>
      {editMode && (
        <Button onClick={onCancel} isDisabled={isSubmitting}>
          Cancel
        </Button>
      )}
    </Flex>
  </Box>
</Container>
</>
)
}

export default UserInformation
...

```.tsx
import React, { createContext, useContext, useEffect, useState, ReactNode } from 'react';
import { useAppSelector, useAppDispatch } from '../hooks/storeHooks';
import { setMessages, setConnectionStatus, setRetries } from '../reducers';
import { RootState } from '../store';
import { MessagesService } from '../client';
import { MessageResponse } from '../client/models';

interface PollingContextType {
 isConnected: boolean;
 data: Array<MessageResponse>; // Замініть на потрібний тип для ваших даних
 retries: number;
}

const PollingContext = createContext<PollingContextType | null>(null);

export const usePolling = () => {
 return useContext(PollingContext);
};

// Тип для пропсів дочірнього компонента
interface PollingProviderProps {
 children: ReactNode;

```

```

}

export const PollingProvider: React.FC<PollingProviderProps> = ({ children }) => {
 const dispatch = useAppDispatch();
 const retries = useAppSelector((state: RootState) => state.sse.retries);
 const [isConnected, setIsConnected] = useState(false);
 const [data, setData] = useState<Array<MessageResponse>>([]); // Замініть на правильний тип

 useEffect(() => {

 const fetchData = async () => {
 try {
 // Використовуємо MessagesService для отримання повідомлень
 const result = await MessagesService.messagesGetMessages(); // викликаємо метод
 if (result != null){
 setData(result);
 }

 dispatch(setMessages(result)); // Відправляємо в Redux
 setIsConnected(true);
 dispatch(setConnectionStatus(true));
 } catch (error) {
 console.error('Polling error:', error);
 setIsConnected(false);
 dispatch(setConnectionStatus(false));
 setTimeout(() => {
 dispatch(setRetries(retries + 1));
 }, 15000); // Повторюємо запит через 3 секунди при помилці
 }
 };

 const intervalId = setInterval(fetchData, 15000); // Пулінг кожні 30 секунд

 return () => {
 clearInterval(intervalId); // Очищаємо інтервал при відміні компонента
 };
 }, [dispatch, retries]);

 return (
 <PollingContext.Provider value={{ isConnected, data, retries }}>
 {children}
 </PollingContext.Provider>
)
}

```

```

);
 };
 ...

  ```.tsx
  import { ViewIcon, ViewOffIcon } from "@chakra-ui/icons"
  import {
    Button,
    Container,
    FormControl,
    FormErrorMessage,
    Icon,
    Image,
    Input,
    InputGroup,
    InputRightElement,
    Link,
    Text,
    useBoolean,
  } from "@chakra-ui/react"
  import {
    Link as RouterLink,
    createFileRoute,

  } from "@tanstack/react-router"
  import { useState, useEffect } from "react";

  import { Box, Heading, Stack, SimpleGrid, GridItem } from "@chakra-ui/react";

  import { Flex, Spacer } from "@chakra-ui/react";

  import useAuth from "../hooks/useAuth";

  export const Route = createFileRoute("/")({
    component: IndexPageSite,
  })

  function IndexPageSite() {
    const { logout } = useAuth();
    const [loggedIn, setLoggedIn] = useState(() => localStorage.getItem("access_token") !== null);

```

```

const handleLogout = async () => {
  logout()
  setLoggedIn(false);
}

return (
  <>

  <Box bg="gray.50" minH="100vh">
    {/ * Меню з логотипом і навігацією */}
    <Box bg="white" boxShadow="md" py={4} px={6}>
      <Flex align="center">
        <Heading as="h1" size="lg" color="teal.500">
          ЄБорд
        </Heading>
        <Spacer />

        {loggedIn ? (
          <>
            <Stack direction="row" spacing={4}>
              <Button as={RouterLink} to="/dashboard" colorScheme="teal">
                Дашборд
              </Button>
              <Button onClick={handleLogout} colorScheme="teal">
                Вийти
              </Button>
            </Stack>
          </>
        ) : (
          <>
            <Stack direction="row" spacing={4}>
              <Button as={RouterLink} to="/login" colorScheme="teal">
                Увійти
              </Button>
              <Button as={RouterLink} to="/signup" colorScheme="teal">
                Реєстрація
              </Button>
            </Stack>
          </>
        )}
      </Flex>

```

```
</Box>
```

```
{/* Слайдер з основними показниками або графіками */}
```

```
<Container maxW="container.xl" centerContent mt={6}>
```

```
  <Box bg="white" p={4} borderRadius="md" boxShadow="lg" mb={8}>
```

```
    <Heading as="h2" size="xl" mb={4}>Головний слайдер</Heading>
```

```
    <Text fontSize="lg" mb={4}>
```

Тут можна буде відображати графіки, що показують дані про споживання електроенергії, за певний період.

```
  </Text>
```

```
  {/* Прості місця для слайдерів/графіків */}
```

```
  <Box bg="gray.200" h="300px" mb={4}>Графік 1</Box>
```

```
  <Box bg="gray.200" h="300px">Графік 2</Box>
```

```
  </Box>
```

```
</Container>
```

```
{/* Блоки можливостей */}
```

```
<Container maxW="container.lg" centerContent mb={12}>
```

```
  <Heading as="h2" size="xl" textAlign="center" mb={8}>
```

```
    Основні можливості
```

```
  </Heading>
```

```
  <SimpleGrid columns={{ base: 1, md: 2, lg: 3 }} spacing={10}>
```

```
    <GridItem>
```

```
      <Box p={5} bg="white" boxShadow="md" borderRadius="md" textAlign="center">
```

```
        <Heading size="md" mb={3}>Дашборд</Heading>
```

```
        <Text mb={4}>Переглядайте статистику споживання, отримуйте аналітику по періо-
```

дах.</Text>

```
        <Button as={RouterLink} to="/dashboard" colorScheme="teal">Переглянути даш-
```

борд</Button>

```
      </Box>
```

```
    </GridItem>
```

```
    <GridItem>
```

```
      <Box p={5} bg="white" boxShadow="md" borderRadius="md" textAlign="center">
```

```
        <Heading size="md" mb={3}>Аналітика</Heading>
```

```
        <Text mb={4}>Дізнайтесь про тенденції та прогнозуйте споживання за допомогою ана-
```

літичних інструментів.</Text>

```
        <Button as={RouterLink} to="/analytics" colorScheme="blue">Переглянути аналі-
```

тику</Button>

```
      </Box>
```

```
    </GridItem>
```

```
</GridItem>
```

```

    <Box p={5} bg="white" boxShadow="md" borderRadius="md" textAlign="center">
      <Heading size="md" mb={3}>Звіти</Heading>
      <Text mb={4}>Створюйте звіти про споживання та експоруйте їх у PDF або
Excel.</Text>
      <Button as={RouterLink} to="/reports" colorScheme="purple">Генерувати звіт</Button>
    </Box>
  </GridItem>
</SimpleGrid>
</Container>

{/* Загальна презентація */}
<Container maxW="container.xl" centerContent mb={12}>
  <Box bg="white" p={6} boxShadow="lg" borderRadius="md">
    <Heading as="h2" size="xl" mb={4}>Про наш проєкт</Heading>
    <Text fontSize="lg" mb={4}>
      Ми надаємо платформу для автоматизації обрахунку витрат електроенергії, збору даних,
аналізу та формування звітів. Завдяки нашому інтерфейсу, ви зможете знизити витрати на електроенергію і
ефективно управляти своїм споживанням.
    </Text>
    <Button as={RouterLink} to="/about" colorScheme="teal" size="lg">Дізнатися бі-
льше</Button>
  </Box>
</Container>

{/* Футер */}
<Box bg="teal.500" color="white" py={4}>
  <Container maxW="container.xl" centerContent>
    <Text>© 2024 Всі права захищено</Text>
    <Stack direction="row" spacing={6} mt={4}>
      <Link as={RouterLink} to="/privacy" color="white">Політика конфіденційності</Link>
      <Link as={RouterLink} to="/terms" color="white">Умови використання</Link>
    </Stack>
  </Container>
</Box>
</Box>

</>
)
}
...

...tsx

```

```

import { ViewIcon, ViewOffIcon } from "@chakra-ui/icons"
import {
  Button,
  Container,
  FormControl,
  FormErrorMessage,
  Icon,
  Image,
  Input,
  InputGroup,
  InputRightElement,
  Link,
  Text,
  useBoolean,
  Spinner
} from "@chakra-ui/react"
import {
  Link as RouterLink,
  createFileRoute,
  redirect,
} from "@tanstack/react-router"
import { type SubmitHandler, useForm } from "react-hook-form"

import Logo from "/assets/images/fastapi-logo.svg"
import type { Body_login_login_access_token as AccessToken } from "../client"
import useAuth, { isLoggedIn } from "../hooks/useAuth"
import { emailPattern } from "../utils"
import { Flex, Box, Heading, Stack, SimpleGrid, GridItem } from "@chakra-ui/react";

export const Route = createFileRoute("/login")({
  component: Login,
  beforeLoad: async () => {
    if (isLoggedIn()) {
      console.log("to /")
      throw redirect({
        to: "/",
      })
    }
  },
})

```

```

function Login() {
  const [show, setShow] = useBoolean()
  const { loginMutation, error, resetError } = useAuth()
  const {
    register,
    handleSubmit,
    formState: { errors, isSubmitting },
  } = useForm<AccessToken>({
    mode: "onBlur",
    criteriaMode: "all",
    defaultValues: {
      username: "",
      password: "",
    },
  })

  const onSubmit: SubmitHandler<AccessToken> = async (data) => {
    if (isSubmitting) return

    resetError()

    try {
      await loginMutation.mutateAsync(data)
    } catch {
      // error is handled by useAuth hook
    }
  }

  return (
    <>
      <Container
        as="form"
        onSubmit={handleSubmit(onSubmit)}
        h="100vh"
        maxW="sm"
        display="flex" // Додаємо Flex до контейнера
        flexDirection="column" // Це дозволить елементам бути у вертикальному порядку
        alignItems="center" // Вирівнювання по горизонталі
        justifyContent="center" // Вирівнювання по вертикалі
        gap={4}
      >
        <Link href="/">

```

```

<Heading as="h1" size="lg" color="teal.500">
  €Борд
</Heading>
</Link>
<FormControl id="username" isInvalid={!!errors.username || !!error}>
  <Input
    id="username"
    {...register("username", {
      required: "Username is required",
      pattern: emailPattern,
    })}
    placeholder="Email"
    type="email"
    required
  />
  {errors.username && (
    <FormErrorMessage>{errors.username.message}</FormErrorMessage>
  )}
</FormControl>
<FormControl id="password" isInvalid={!!error}>
  <InputGroup>
    <Input
      {...register("password", {
        required: "Password is required",
      })}
      type={show ? "text" : "password"}
      placeholder="Password"
      required
    />
    <InputRightElement
      color="ui.dim"
      _hover={{
        cursor: "pointer",
      }}
    >
      <Icon
        as={show ? ViewOffIcon : ViewIcon}
        onClick={setShow.toggle}
        aria-label={show ? "Hide password" : "Show password"}
      >
        {show ? <ViewOffIcon /> : <ViewIcon />}
      </Icon>
    </InputRightElement>
  </InputGroup>
</FormControl>

```

```

        </InputRightElement>
      </InputGroup>
      {error && <FormErrorMessage>{error}</FormErrorMessage>}
    </FormControl>
    <Link as={RouterLink} to="/recover-password" color="blue.500">
      Forgot password?
    </Link>
    <Button
      variant="primary"
      type="submit"
      display="block"
      mx="auto"
      width="full"
      justifyContent="center"
      alignItems="center"
    >
      {isSubmitting ? (
        <Spinner size="md" color="ui.light" />
      ) : (
        "Log In"
      )}
    </Button>
    <Text>
      Don't have an account?{" "}
      <Link as={RouterLink} to="/signup" color="blue.500">
        Sign up
      </Link>
    </Text>
  </Container>
</>
)
}
...

```.tsx
import {
 Button,
 Container,
 FormControl,
 FormErrorMessage,
 Heading,
 Input,

```

```

Text,
} from "@chakra-ui/react"
import { useMutation } from "@tanstack/react-query"
import { createFileRoute, redirect } from "@tanstack/react-router"
import { type SubmitHandler, useForm } from "react-hook-form"

import { type ApiError, LoginService } from "../client"
import { isLoggedIn } from "../hooks/useAuth"
import useCustomToast from "../hooks/useCustomToast"
import { emailPattern, handleError } from "../utils"

interface FormData {
 email: string
}

export const Route = createFileRoute("/recover-password")({
 component: RecoverPassword,
 beforeLoad: async () => {
 if (isLoggedIn()) {
 throw redirect({
 to: "/",
 })
 }
 },
})

function RecoverPassword() {
 const {
 register,
 handleSubmit,
 reset,
 formState: { errors, isSubmitting },
 } = useForm<FormData>()
 const showToast = useCustomToast()

 const recoverPassword = async (data: FormData) => {
 await LoginService.loginRecoverPassword({
 email: data.email,
 })
 }

 const mutation = useMutation({

```

```

mutationFn: recoverPassword,
onSuccess: () => {
 showToast(
 "Email sent.",
 "We sent an email with a link to get back into your account.",
 "success",
)
 reset()
},
onError: (err: ApiError) => {
 handleError(err, showToast)
},
})

const onSubmit: SubmitHandler<FormData> = async (data) => {
 mutation.mutate(data)
}

return (
 <Container
 as="form"
 onSubmit={handleSubmit(onSubmit)}
 h="100vh"
 maxW="sm"
 alignItems="stretch"
 justifyContent="center"
 gap={4}
 centerContent
 >
 <Heading size="xl" color="ui.main" textAlign="center" mb={2}>
 Password Recovery
 </Heading>
 <Text align="center">
 A password recovery email will be sent to the registered account.
 </Text>
 <FormControl isValid={!errors.email}>
 <Input
 id="email"
 {...register("email", {
 required: "Email is required",
 pattern: emailPattern,
 })}
 >

```

```

 placeholder="Email"
 type="email"
 />
 {errors.email && (
 <FormErrorMessage>{errors.email.message}</FormErrorMessage>
)}
 </FormControl>
 <Button variant="primary" type="submit" isLoading={isSubmitting}>
 Continue
 </Button>
 </Container>
)
}
```

```tsx
import {
 Button,
 Container,
 FormControl,
 FormErrorMessage,
 FormLabel,
 Heading,
 Input,
 Text,
} from "@chakra-ui/react"
import { useMutation } from "@tanstack/react-query"
import { createFileRoute, redirect, useNavigate } from "@tanstack/react-router"
import { type SubmitHandler, useForm } from "react-hook-form"

import { type ApiError, LoginService, type NewPassword } from "../client"
import { isLoggedIn } from "../hooks/useAuth"
import useCustomToast from "../hooks/useCustomToast"
import { confirmPasswordRules, handleError, passwordRules } from "../utils"

interface NewPasswordForm extends NewPassword {
 confirm_password: string
}

export const Route = createFileRoute("/reset-password")({
 component: ResetPassword,
 beforeLoad: async () => {

```

```

 if (isLoggedIn()) {
 throw redirect({
 to: "/",
 })
 }
 },
})

function ResetPassword() {
 const {
 register,
 handleSubmit,
 getValues,
 reset,
 formState: { errors },
 } = useForm<NewPasswordForm>({
 mode: "onBlur",
 criteriaMode: "all",
 defaultValues: {
 new_password: "",
 },
 })

 const showToast = useCustomToast()
 const navigate = useNavigate()

 const resetPassword = async (data: NewPassword) => {
 const token = new URLSearchParams(window.location.search).get("token")
 if (!token) return
 await LoginService.loginResetPassword({
 requestBody: { new_password: data.new_password, token: token },
 })
 }

 const mutation = useMutation({
 mutationFn: resetPassword,
 onSuccess: () => {
 showToast("Success!", "Password updated successfully.", "success")
 reset()
 navigate({ to: "/login" })
 },
 onError: (err: ApiError) => {
 handleError(err, showToast)
 }
 })
}

```

```

 },
 })

```

```

const onSubmit: SubmitHandler<NewPasswordForm> = async (data) => {
 mutation.mutate(data)
}

```

```

return (
 <Container
 as="form"
 onSubmit={handleSubmit(onSubmit)}
 h="100vh"
 maxW="sm"
 alignItems="stretch"
 justifyContent="center"
 gap={4}
 centerContent
 >
 <Heading size="xl" color="ui.main" textAlign="center" mb={2}>
 Reset Password
 </Heading>
 <Text textAlign="center">
 Please enter your new password and confirm it to reset your password.
 </Text>
 <FormControl mt={4} isValid={!errors.new_password}>
 <FormLabel htmlFor="password">Set Password</FormLabel>
 <Input
 id="password"
 {...register("new_password", passwordRules())}
 placeholder="Password"
 type="password"
 />
 {errors.new_password && (
 <FormErrorMessage>{errors.new_password.message}</FormErrorMessage>
)}
 </FormControl>
 <FormControl mt={4} isValid={!errors.confirm_password}>
 <FormLabel htmlFor="confirm_password">Confirm Password</FormLabel>
 <Input
 id="confirm_password"
 {...register("confirm_password", confirmPasswordRules(getValues))}
 placeholder="Password"
 />
 </FormControl>
 </Container>
)

```

```

 type="password"
 />
 {errors.confirm_password && (
 <FormErrorMessage>{errors.confirm_password.message}</FormErrorMessage>
)}
 </FormControl>
 <Button variant="primary" type="submit">
 Reset Password
 </Button>
 </Container>
)
}
```

```

```

```.tsx
import {
 Button,
 Container,
 Flex,
 FormControl,
 FormErrorMessage,
 FormLabel,
 Image,
 Input,
 Link,
 Text,
 Spinner
} from "@chakra-ui/react"
import {
 Link as RouterLink,
 createFileRoute,
 redirect,
} from "@tanstack/react-router"
import { type SubmitHandler, useForm } from "react-hook-form"

import Logo from "/assets/images/fastapi-logo.svg"
import type { UserRegister } from "../client"
import useAuth, { isLoggedIn } from "../hooks/useAuth"
import { confirmPasswordRules, emailPattern, passwordRules } from "../utils"
import { Box, Heading, Stack, SimpleGrid, GridItem } from "@chakra-ui/react";

```

```

export const Route = createFileRoute("/signup")({
 component: SignUp,
 beforeLoad: async () => {
 if (isLoggedIn()) {
 throw redirect({
 to: "/",
 })
 }
 },
})

interface UserRegisterForm extends UserRegister {
 confirm_password: string
}

function SignUp() {
 const { signUpMutation } = useAuth()
 const {
 register,
 handleSubmit,
 getValues,
 formState: { errors, isSubmitting },
 } = useForm<UserRegisterForm>({
 mode: "onBlur",
 criteriaMode: "all",
 defaultValues: {
 email: "",
 full_name: "",
 password: "",
 confirm_password: "",
 },
 })

 const onSubmit: SubmitHandler<UserRegisterForm> = (data) => {
 signUpMutation.mutate(data)
 }

 return (
 <>
 <Flex flexDir={{ base: "column", md: "row" }} justify="center" h="100vh">
 <Container
 as="form"

```

```

onSubmit={handleSubmit(onSubmit)}
h="100vh"
maxW="sm"
display="flex" // Додаємо Flex до контейнера
flexDirection="column" // Це дозволить елементам бути у вертикальному порядку
alignItems="center" // Вирівнювання по горизонталі
justifyContent="center" // Вирівнювання по вертикалі
gap={4}
>
 <Link href="/">
<Heading as="h1" size="lg" color="teal.500">
 СБорд
</Heading>
</Link>
 <FormControl id="full_name" isInvalid={!!errors.full_name}>
 <FormLabel htmlFor="full_name" srOnly>
 Full Name
 </FormLabel>
 <Input
 id="full_name"
 minLength={3}
 {...register("full_name", { required: "Full Name is required" })}
 placeholder="Full Name"
 type="text"
 />
 {errors.full_name && (
 <FormErrorMessage>{errors.full_name.message}</FormErrorMessage>
)}
 </FormControl>
 <FormControl id="email" isInvalid={!!errors.email}>
 <FormLabel htmlFor="username" srOnly>
 Email
 </FormLabel>
 <Input
 id="email"
 {...register("email", {
 required: "Email is required",
 pattern: emailPattern,
 })}
 placeholder="Email"
 type="email"
 />

```

```

 {errors.email && (
 <FormErrorMessage>{errors.email.message}</FormErrorMessage>
)}
 </FormControl>
 <FormControl id="password" isInvalid={!errors.password}>
 <FormLabel htmlFor="password" srOnly>
 Password
 </FormLabel>
 <Input
 id="password"
 {...register("password", passwordRules())}
 placeholder="Password"
 type="password"
 />
 {errors.password && (
 <FormErrorMessage>{errors.password.message}</FormErrorMessage>
)}
 </FormControl>
 <FormControl
 id="confirm_password"
 isInvalid={!errors.confirm_password}
 >
 <FormLabel htmlFor="confirm_password" srOnly>
 Confirm Password
 </FormLabel>
 <Input
 id="confirm_password"
 {...register("confirm_password", confirmPasswordRules(getValues))}
 placeholder="Repeat Password"
 type="password"
 />
 {errors.confirm_password && (
 <FormErrorMessage>
 {errors.confirm_password.message}
 </FormErrorMessage>
)}
 </FormControl>

 <Button
 variant="primary"
 type="submit"

```

```

 display="block"
 mx="auto"
 width="full"
 justifyContent="center"
 alignItems="center"
 >
 {isSubmitting ? (
 <Spinner size="md" color="ui.light" />
) : (
 "Sign Up"
)}
 </Button>

 <Text>
 Already have an account?{" "}
 <Link as={RouterLink} to="/login" color="blue.500">
 Log In
 </Link>
 </Text>
 </Container>
</Flex>
</>
)
}

export default SignUp
```

```tsx
import { Flex, Spinner } from "@chakra-ui/react"
import { Outlet, createFileRoute, redirect } from "@tanstack/react-router"

import Sidebar from "../components/Common/Sidebar"
import UserMenu from "../components/Common/UserMenu"
import useAuth, { isLoggedIn } from "../hooks/useAuth"

export const Route = createFileRoute("/_layout")({
 component: Layout,
 beforeLoad: async () => {
 if (!isLoggedIn()) {
 console.log("to login")
 throw redirect({

```

```

 to: "/login",
 })

 /*throw redirect({
 to: "/",
 })*/
}
},
})

function Layout() {
 const { isLoading } = useAuth()

 return (
 <Flex maxW="large" h="auto" position="relative">
 <Sidebar />
 {isLoading ? (
 <Flex justify="center" align="center" height="100vh" width="full">
 <Spinner size="xl" color="ui.main" />
 </Flex>
) : (
 <Outlet />
)}
 <UserMenu />
 </Flex>
)
}

...

```.tsx

import { Outlet, createRootRoute } from "@tanstack/react-router"
import React, { Suspense } from "react"

import NotFound from "../components/Common/NotFound"

const loadDevtools = () =>
  Promise.all([
    import("@tanstack/router-devtools"),
    import("@tanstack/react-query-devtools"),
  ]).then(([routerDevtools, reactQueryDevtools]) => {
    return {
      default: () => (

```

```

    <>
      <routerDevtools.TanStackRouterDevtools />
      <reactQueryDevtools.ReactQueryDevtools />
    </>
  ),
}
})

const TanStackDevtools =
  process.env.NODE_ENV === "production" ? () => null : React.lazy(loadDevtools)

export const Route = createRootRoute({
  component: () => (
    <>
      <Outlet />
      <Suspense>
        <TanStackDevtools />
      </Suspense>
    </>
  ),
  notFoundComponent: () => <NotFound />,
})
```


```tsx



```

import { Flex, Spinner, Input, Button, VStack, FormLabel, Text, useToast } from "@chakra-ui/react";
import { createFileRoute } from "@tanstack/react-router";
import { useState } from "react";
import type { ChangeEvent } from "react";

// Створення маршруту
export const Route = createFileRoute("/_layout/add_data")({
 component: ImportPage,
});

function ImportPage() {
 const [file, setFile] = useState<File | null>(null);
 const [counterValue, setCounterValue] = useState("");
 const [isLoading, setIsLoading] = useState(false);
 const toast = useToast();

 const handleFileChange = (event: ChangeEvent<HTMLInputElement>) => {

```


```

```

    setFile(event.target.files?.[0] || null);
  };

  const handleCounterChange = (event: ChangeEvent<HTMLInputElement>) => {
    setCounterValue(event.target.value);
  };

  const handleImport = () => {
    if (!file) {
      toast({ title: "Будь ласка, виберіть файл.", status: "warning", duration: 3000 });
      return;
    }
    setIsLoading(true);
    setTimeout(() => {
      setIsLoading(false);
      toast({ title: "Файл успішно імпортовано!", status: "success", duration: 3000 });
    }, 2000);
  };

  const handleSubmit = () => {
    toast({ title: `Введено значення: ${counterValue}`, status: "info", duration: 3000 });
  };

  return (
    <Flex direction="column" width="100%" align="center" justify="center" height="100vh"
    bg="gray.500" p={5}>
      <VStack
        spacing={6}
        width="100%"
        maxWidth="800px"
        p={6}
        bg="gray.1000"
        boxShadow="md"
        borderRadius="md"
      >

        <Text fontSize="2xl" fontWeight="bold">Імпорт та введення даних</Text>

        <FormLabel>Імпорт файлу</FormLabel>
        <Input type="file" onChange={handleFileChange} />

        <FormLabel>Значення лічильника</FormLabel>

```

```

    <Input type="number" value={counterValue} onChange={handleCounterChange} placeholder="Введіть значення" />

```

```

    {isLoading ? (
      <Spinner size="lg" color="blue.500" />
    ) : (
      <>
        <Button colorScheme="blue" onClick={handleImport} width="full">Імпортувати</Button>
        <Button colorScheme="green" onClick={handleSubmit} width="full">Зберегти</Button>
      </>
    )}
  </VStack>
</Flex>
);
}

```

```

export default ImportPage;
...

```

```

```.tsx
import {
 Badge,
 Box,
 Button,
 Container,
 Flex,
 Heading,
 SkeletonText,
 Table,
 TableContainer,
 Tbody,
 Td,
 Th,
 Thead,
 Tr,
} from "@chakra-ui/react"
import { useQuery, useQueryClient } from "@tanstack/react-query"
import { createFileRoute, useNavigate } from "@tanstack/react-router"
import { useEffect } from "react"
import { z } from "zod"

import { type UserPublic, UsersService } from "../../client"

```

```

import AddUser from "../../components/Admin/AddUser"
import ActionsMenu from "../../components/Common/ActionsMenu"
import Navbar from "../../components/Common/Navbar"

const usersSearchSchema = z.object({
 page: z.number().catch(1),
})

export const Route = createFileRoute("/_layout/admin")({
 component: Admin,
 validateSearch: (search) => usersSearchSchema.parse(search),
})

const PER_PAGE = 5

function getUsersQueryOptions({ page }: { page: number }) {
 return {
 queryFn: () =>
 UsersService.usersReadUsers({ skip: (page - 1) * PER_PAGE, limit: PER_PAGE }),
 queryKey: ["users", { page }],
 }
}

function UsersTable() {
 const queryClient = useQueryClient()
 const currentUser = queryClient.getQueryData<UserPublic>(["currentUser"])
 const { page } = Route.useSearch()
 const navigate = useNavigate({ from: Route.fullPath })
 const setPage = (page: number) =>
 navigate({ search: (prev) => ({ ...prev, page }) })

 const {
 data: users,
 isPending,
 isPlaceholderData,
 } = useQuery({
 ...getUsersQueryOptions({ page }),
 placeholderData: (prevData) => prevData,
 })

 const hasNextPage = !isPlaceholderData && users?.data.length === PER_PAGE
 const hasPreviousPage = page > 1

```

```

useEffect(() => {
 if (hasNextPage) {
 queryClient.prefetchQuery(getUsersQueryOptions({ page: page + 1 }))
 }
}, [page, queryClient, hasNextPage])

return (
 <>
 <TableContainer>
 <Table size={{ base: "sm", md: "md" }}>
 <Thead>
 <Tr>
 <Th width="20%">Full name</Th>
 <Th width="50%">Email</Th>
 <Th width="10%">Role</Th>
 <Th width="10%">Status</Th>
 <Th width="10%">Actions</Th>
 </Tr>
 </Thead>
 {isPending ? (
 <Tbody>
 <Tr>
 {new Array(4).fill(null).map((_, index) => (
 <Td key={index}>
 <SkeletonText noOfLines={1} paddingBlock="16px" />
 </Td>
))}
 </Tr>
 </Tbody>
) : (
 <Tbody>
 {users?.data.map((user) => (
 <Tr key={user.id}>
 <Td
 color={!user.full_name ? "ui.dim" : "inherit"}
 isTruncated
 maxWidth="150px"
 >
 {user.full_name || "N/A"}
 {currentUser?.id === user.id && (
 <Badge ml="1" colorScheme="teal">

```

```

 You
 </Badge>
)}
 </Td>
 <Td isTruncated maxWidth="150px">
 {user.email}
 </Td>
 <Td>{user.is_superuser ? "Superuser" : "User"}</Td>
 <Td>
 <Flex gap={2}>
 <Box
 w="2"
 h="2"
 borderRadius="50%"
 bg={user.is_active ? "ui.success" : "ui.danger"}
 alignSelf="center"
 />
 {user.is_active ? "Active" : "Inactive"}
 </Flex>
 </Td>
 <Td>
 <ActionsMenu
 type="User"
 value={user}
 disabled={currentUser?.id === user.id ? true : false}
 />
 </Td>
</Tr>
)))
</Tbody>
)}
</Table>
</TableContainer>
<Flex
 gap={4}
 alignItems="center"
 mt={4}
 direction="row"
 justifyContent="flex-end"
>
 <Button onClick={() => setPage(page - 1)} isDisabled={!hasPreviousPage}>
 Previous

```

```

 </Button>
 Page {page}
 <Button isDisabled={!hasNextPage} onClick={() => setPage(page + 1)}>
 Next
 </Button>
 </Flex>
</>
)
}

```

```

function Admin() {
 return (
 <Container maxW="full">
 <Heading size="lg" textAlign={{ base: "center", md: "left" }} pt={12}>
 Users Management
 </Heading>

 <Navbar type="User" addModalAs={AddUser} />
 <UsersTable />
 </Container>
)
}

```

```

```tsx

```

```

import { createFileRoute, Outlet } from "@tanstack/react-router";

```

```

// Створення маршруту

```

```

export const Route = createFileRoute("/_layout/analytics")({
    component: AnalyticsPage
});

```

```

function AnalyticsPage() {
    return (
        <Outlet />
    );
}

```

```

export default AnalyticsPage;```

```

```

```tsx

```

```

import { Heading, VStack, HStack, Box, Container, Text, Button, Center } from "@chakra-ui/react"

```

```

import { createFileRoute } from "@tanstack/react-router"
import { Line, Bar, Doughnut } from "react-chartjs-2";
import { Chart as ChartJS, registerables, TooltipItem } from "chart.js";
import useAuth from "../../hooks/useAuth";
import { ItemsService } from "../../client";
import zoomPlugin from "chartjs-plugin-zoom"; // Імпортуємо плагін zoom
import { useQuery, useQueryClient } from "@tanstack/react-query";
import { useNavigate } from "@tanstack/react-router";
import { useEffect } from "react";
import { z } from "zod";
import { parse, format } from "date-fns";
import { uk } from 'date-fns/locale';
import { stringify } from "querystring";

const itemsSearchSchema = z.object({
 page: z.number().catch(1),
})

export const Route = createFileRoute("/_layout/dashboard")({
 component: Dashboard,
 validateSearch: (search) => itemsSearchSchema.parse(search),
})

ChartJS.register(...registerables);

ChartJS.register(...registerables, zoomPlugin); // Реєструємо плагін

const PER_PAGE = 5

function getItemsQueryOptions({ page }: { page: number }) {
 return {
 queryFn: () =>
 ItemsService.itemsReadItems({ skip: (page - 1) * PER_PAGE, limit: PER_PAGE }),
 queryKey: ["items", { page }],
 }
}

function Dashboard() {
 const { user: currentUser } = useAuth()

```

```

const queryClient = useQueryClient()
const { page } = Route.useSearch()
const navigate = useNavigate({ from: Route.fullPath })
const setPage = (page: number) =>
 navigate({ search: (prev) => ({ ...prev, page }) })

const {
 data: items = { data: [] },
 isPending,
 isPlaceholderData,
} = useQuery({
 ...getItemsQueryOptions({ page }),
 placeholderData: (prevData) => prevData,
})

const hasNextPage = !isPlaceholderData && items?.data.length > PER_PAGE && items?.data.length >
PER_PAGE + 1
const hasPreviousPage = page > 1

useEffect(() => {
 if (hasNextPage) {
 queryClient.prefetchQuery(getItemsQueryOptions({ page: page + 1 }))
 }
}, [page, queryClient, hasNextPage])

let sortedItems = {"data" : items?.data.slice().sort((a, b) => new Date(a.reading_date) - new
Date(b.reading_date))}

const lineData_reading_value = {
 labels: sortedItems?.data.map((item) =>
 (new Intl.DateTimeFormat('uk-UA', { month: 'long' })).format(new Date(item.reading_date))
),
 datasets: [
 {
 label: "Графік поточного показника",
 data: sortedItems?.data.map((item) => item.reading_value),
 borderColor: "rgba(75, 192, 192, 1)",
 borderWidth: 2,
 },
],

```

```

],
 };

 const lineData_kwh_tariff = {
 labels: sortedItems?.data.map((item) => (new Intl.DateTimeFormat('uk-UA', { month: 'long'
 })).format(new Date(item.reading_date))),
 datasets: [
 {
 label: "Графік тарифу",
 data: sortedItems?.data.map((item) => item.kwh_tariff),
 borderColor: "rgba(99, 34, 98, 1)",
 borderWidth: 2,
 },
],
 };

 const lineData_current_amount = {
 labels: sortedItems?.data.map((item) => (new Intl.DateTimeFormat('uk-UA', { month: 'long'
 })).format(new Date(item.reading_date))),
 datasets: [
 {
 label: "Графік суми за місяць",
 data: sortedItems?.data.map((item) => item.current_amount),
 borderColor: "rgba(33, 192, 22, 1)",
 borderWidth: 2,
 },
],
 };

 const deltas = sortedItems?.data.map((item, index, array) => {
 if (index === 0) return null; // Для першого елемента немає попереднього значення
 return item.reading_value - array[index - 1].reading_value;
 }).filter(value => value !== null); // Видаляємо `null` значення

 const lineData_per_m = {
 labels: sortedItems?.data.slice(1).map((item) => // Зсуваємо на один місяць вперед
 (new Intl.DateTimeFormat('uk-UA', { month: 'long' })).format(new Date(item.reading_date))
),
 datasets: [
 {
 label: "Графік дельт між місяцями",

```

```

 data: deltas, // Додаємо обчислені дельти
 borderColor: "rgba(9, 66, 76, 1)",
 borderWidth: 2,
 },
],
};

```

```

let lineOptions = {
 responsive: true,
 plugins: {
 tooltip: {
 callbacks: {
 label: function (tooltipItem: TooltipItem<'line'>) {
 return `Значення: ${tooltipItem.raw}`;
 },
 },
 },
 },
},
// Налаштування для zoom та панорування
interaction: {
 intersect: false,
 mode: "index",
 axis: 'x'
},
scales: {
 x: {
 // Встановлюємо дозволений діапазон для зуму по осі X
 ticks: {
 autoSkip: true,
 maxTicksLimit: 10,
 },
 },
 y: {
 // Встановлюємо дозволений діапазон для зуму по осі Y
 },
},
zoom: {
 // Параметри для зуму
 wheel: {
 enabled: true, // Дозволити зум за допомогою колеса миші
 modifierKey: "ctrl", // Для зуму можна натискати Ctrl

```

```

 },
 pinch: {
 enabled: true, // Дозволити зум на мобільних пристроях за допомогою пінчу
 },
 drag: {
 enabled: true, // Дозволити панорамування графіка за допомогою перетягування
 },
 }
};

```

```

const barData = {
 labels: sortedItems?.data.slice(1).map((item, index) => {
 return (new Intl.DateTimeFormat('uk-UA', { month: 'long' })).format(new Date(item.reading_date))
 }),
 datasets: [
 {
 label: "Вартість за місяць",
 data: sortedItems?.data.slice(1).map((item, index) => {

 const prevItem = sortedItems?.data[index];
 const delta = item.reading_value - prevItem.reading_value;

 // Множення на тариф
 const cost = delta * prevItem.kwh_tariff;

 return cost;

 }),
 backgroundColor: "rgba(75, 192, 192, 0.6)",
 borderColor: "rgba(75, 192, 192, 1)",
 borderWidth: 2,
 },
],
};

```

```

const barOptions = {
 responsive: true,
 scales: {
 y: {
 beginAtZero: true, // Почати від 0 на осі Y

```

```

 },
 },
 plugins: {
 tooltip: {
 callbacks: {
 label: function (tooltipItem) {
 return `Вартість: ${tooltipItem.raw} грн`;
 },
 },
 },
 },
};

```

```

return (
 <>
 <Container maxW="full" m={4}>
 { /*<Box pt={12} m={4}>
 <Text fontSize="2xl">
 Hi, {currentUser?.full_name || currentUser?.email} 🤖
 </Text>
 <Text>Welcome back, nice to see you again!</Text>
 </Box>* / }

 <VStack spacing={8}>

 <Heading size="lg" textAlign={{ base: "center", md: "left" }}>
 Графіки показників
 </Heading>

 <VStack spacing={8} justifyContent="center" alignItems="center" width="100%">
 <Box width={{ base: "100%", md: "80%" }}>
 <Bar data={barData} options={barOptions} />
 </Box>

 <Box width={{ base: "100%", md: "80%" }} >
 <Line data={lineData_reading_value} options={lineOptions}/>
 </Box>

 <Box width={{ base: "100%", md: "80%" }}>
 <Line data={lineData_kwh_tariff} options={lineOptions}/>
 </Box>

```

```

 <Box width={{ base: "100%", md: "80%" }}>
 <Line data={lineData_current_amount} options={lineOptions}/>
 </Box>

 <Box width={{ base: "100%", md: "80%" }}>
 <Line data={lineData_per_m} options={lineOptions}/>
 </Box>
 </VStack>

</VStack>

<Box pt={12} m={4} textAlign="center">
 <HStack spacing={8} justifyContent="center" alignItems="center">
 <Button onClick={() => setPage(page - 1)} isDisabled={!hasPreviousPage}>
 Previous
 </Button>
 Page {page}
 <Button isDisabled={!hasNextPage} onClick={() => setPage(page + 1)}>
 Next
 </Button>
 </HStack>
</Box>

</Container>
</>
)
}
```

```tsx
import {
 Container,
 Heading,
 Text,
 Table,
 Thead,
 Tbody,
 Tr,
 Th,
 Td,
 Tfoot,
 Box,

```

```

VStack,
Modal,
ModalOverlay,
ModalContent,
ModalHeader,
ModalBody,
ModalCloseButton,
Spinner,
} from "@chakra-ui/react";
import { createFileRoute } from "@tanstack/react-router";
import { useState, useEffect } from "react";
import InfiniteScroll from "react-infinite-scroll-component";

// Функція для генерації випадкових даних
const generateRandomData = (numRows: number) => {
 return Array.from({ length: numRows }, (_, index) => ({
 id: index + 1,
 name: `Item ${index + 1}`,
 value: Math.floor(Math.random() * 100), // Випадкове значення від 0 до 99
 }));
};

export const Route = createFileRoute("/_layout/data")({
 component: DataPage,
});

function DataPage() {
 const [data, setData] = useState(generateRandomData(10)); // Ініціалізація даних з 10 рядками
 const [hasMore, setHasMore] = useState(true); // Прапорець для перевірки наявності більше даних
 const [isLoading, setIsLoading] = useState(false); // Прапорець для показу модалки

 // Функція для підвантаження нових даних
 const loadMoreData = () => {
 if (!hasMore) return; // Якщо немає більше даних, виходимо

 setIsLoading(true); // Включаємо модалку
 const newData = generateRandomData(10);
 setTimeout(() => {
 setData((prevData) => [...prevData, ...newData]); // Додаємо нові дані до вже існуючих
 setIsLoading(false); // Вимикаємо модалку

 // Якщо нових даних менше 10, вважаємо, що це остання партія
 }, 1000);
 };
}

```

```

 if (newData.length < 10) {
 setHasMore(false);
 }
 }, 1000); // Симуляція затримки для завантаження
};

useEffect(() => {
 // Перший раз завантажуюємо дані
 loadMoreData();
}, []);

return (
 <Container maxW="full" py={10}>
 <VStack spacing={8}>
 <Heading size="lg" textAlign={{ base: "center", md: "left" }}>
 Data
 </Heading>
 <Text>Hello world</Text>

 <Box
 width="100%"
 overflowY="auto" // Дозволяємо вертикальний скрол
 height="calc(100vh - 200px)" // Встановлюємо висоту контейнера для скролінгу
 id="scrollableDiv"
 >
 <InfiniteScroll
 dataLength={data.length} // Кількість елементів
 next={loadMoreData} // Функція для підвантаження нових даних
 hasMore={hasMore} // Чи є ще дані для підвантаження
 loader={null} // Вимикаємо стандартний loader
 scrollableTarget="scrollableDiv" // Ідентифікатор для контейнера
 >
 <Table variant="striped" colorScheme="teal" width="100%"> { /* Задаємо ширину для таблиці
*/}

 <Thead>
 <Tr>
 <Th minWidth="100px">ID</Th>
 <Th minWidth="150px">Name</Th>
 <Th isNumeric minWidth="100px">Value</Th>
 </Tr>
 </Thead>
 <Tbody>

```

```

 {data.map((item) => (
 <Tr key={item.id}>
 <Td>{item.id}</Td>
 <Td>{item.name}</Td>
 <Td isNumeric>{item.value}</Td>
 </Tr>
))}
 </Tbody>
 <Tfoot>
 <Tr>
 <Th minWidth="100px">ID</Th>
 <Th minWidth="150px">Name</Th>
 <Th isNumeric minWidth="100px">Value</Th>
 </Tr>
 </Tfoot>
 </Table>
</InfiniteScroll>
</Box>

 {/* Модалка для завантаження */}
 <Modal isOpen={isLoading} onClose={() => setIsLoading(false)}>
 <ModalOverlay />
 <ModalContent>
 <ModalHeader>Loading...</ModalHeader>
 <ModalCloseButton />
 <ModalBody>
 <Spinner
 thickness="4px"
 speed="0.65s"
 emptyColor="gray.200"
 size="xl"
 display="block"
 margin="0 auto"
 />
 </ModalBody>
 </ModalContent>
 </Modal>
</VStack>
</Container>

);
}

```

```

export default DataPage;
...

```.tsx
import {
  Button,
  Container,
  Flex,
  Heading,
  SkeletonText,
  Table,
  TableContainer,
  Tbody,
  Td,
  Th,
  Thead,
  Tr,
} from "@chakra-ui/react"
import { useQuery, useQueryClient } from "@tanstack/react-query"
import { createFileRoute, useNavigate } from "@tanstack/react-router"
import { useEffect } from "react"
import { z } from "zod"

import { ItemsService } from "../../client"
import ActionsMenu from "../../components/Common/ActionsMenu"
import Navbar from "../../components/Common/Navbar"
import AddItem from "../../components/Items/AddItem"

const itemsSearchSchema = z.object({
  page: z.number().catch(1),
})

export const Route = createFileRoute("/_layout/items")({
  component: Items,
  validateSearch: (search) => itemsSearchSchema.parse(search),
})

const PER_PAGE = 5

function getItemsQueryOptions({ page }: { page: number }) {
  return {
    queryFn: () =>

```

```

    ItemsService.itemsReadItems({ skip: (page - 1) * PER_PAGE, limit: PER_PAGE }),
    queryKey: ["items", { page }],
  }
}

```

```

function ItemsTable() {
  const queryClient = useQueryClient()
  const { page } = Route.useSearch()
  const navigate = useNavigate({ from: Route.fullPath })
  const setPage = (page: number) =>
    navigate({ search: (prev) => ({ ...prev, page }) })

  const {
    data: items,
    isPending,
    isPlaceholderData,
  } = useQuery({
    ...getItemsQueryOptions({ page }),
    placeholderData: (prevData) => prevData,
  })

  const hasNextPage = !isPlaceholderData && items?.data.length === PER_PAGE
  const hasPreviousPage = page > 1

  useEffect(() => {
    if (hasNextPage) {
      queryClient.prefetchQuery(getItemsQueryOptions({ page: page + 1 }))
    }
  }, [page, queryClient, hasNextPage])

  return (
    <>
    <TableContainer>
      <Table size={{ base: "sm", md: "md" }}>
        <Thead>
          <Tr>
            <Th>ID</Th>
            <Th>Title</Th>
            <Th>Description</Th>
            <Th>Reading Value</Th>
            <Th>Kilowatt-Hour Tariff</Th>
            <Th>Current Amount</Th>

```

```

<Th>Meter Identifier</Th>
<Th>Date</Th>
<Th>Actions</Th>
</Tr>
</Thead>
{isPending ? (
  <Tbody>
    <Tr>
      {new Array(4).fill(null).map((_, index) => (
        <Td key={index}>
          <SkeletonText noOfLines={1} paddingBlock="16px" />
        </Td>
      ))}
    </Tr>
  </Tbody>
) : (
  <Tbody>
    {items?.data.map((item) => (
      <Tr key={item.id} opacity={isPlaceholderData ? 0.5 : 1}>
        <Td>{item.id}</Td>
        <Td isTruncated maxWidth="150px">
          {item.title}
        </Td>
        <Td
          color={!item.description ? "ui.dim" : "inherit"}
          isTruncated
          maxWidth="150px"
        >
          {item.description || "N/A"}
        </Td>
        <Td>{item.reading_value}</Td>
        <Td>{item.kwh_tariff}</Td>
        <Td>{item.current_amount}</Td>
        <Td>{item.meter_identifier}</Td>
        <Td>{item.reading_date}</Td>
        <Td>
          <ActionsMenu type={"Item"} value={item} />
        </Td>
      </Tr>
    ))}
  </Tbody>
)}

```

```

    </Table>
  </TableContainer>
  <Flex
    gap={4}
    alignItems="center"
    mt={4}
    direction="row"
    justifyContent="flex-end"
  >
    <Button onClick={() => setPage(page - 1)} isDisabled={!hasPreviousPage}>
      Previous
    </Button>
    <span>Page {page}</span>
    <Button isDisabled={!hasNextPage} onClick={() => setPage(page + 1)}>
      Next
    </Button>
  </Flex>
</>
)
}

```

```

function Items() {
  return (
    <Container maxW="full">
      <Heading size="lg" textAlign={{ base: "center", md: "left" }} pt={12}>
        Items Management
      </Heading>

      <Navbar type={"Item"} addModalAs={AddItem} />
      <ItemsTable />
    </Container>
  )
}

```

```

```tsx

```

```

import { createFileRoute, Outlet } from "@tanstack/react-router";

```

```

// Створення маршруту

```

```

export const Route = createFileRoute("/_layout/reports")({
 component: ReportsPage
});

```

```

function ReportsPage() {

 return (
 <Outlet />
);
}

export default ReportsPage;
...

```.tsx
import {
  Container,
  Heading,
  Tab,
  TabList,
  TabPanel,
  TabPanels,
  Tabs,
} from "@chakra-ui/react"
import { useQueryClient } from "@tanstack/react-query"
import { createFileRoute } from "@tanstack/react-router"

import type { UserPublic } from "../../client"
import Appearance from "../../components/UserSettings/Appearance"
import ChangePassword from "../../components/UserSettings/ChangePassword"
import DeleteAccount from "../../components/UserSettings/DeleteAccount"
import UserInformation from "../../components/UserSettings/UserInformation"

const tabsConfig = [
  { title: "My profile", component: UserInformation },
  { title: "Password", component: ChangePassword },
  { title: "Appearance", component: Appearance },
  { title: "Danger zone", component: DeleteAccount },
]

export const Route = createFileRoute("/_layout/settings")({
  component: UserSettings,
})

function UserSettings() {

```

```

const queryClient = useQueryClient()
const currentUser = queryClient.getQueryData<UserPublic>(["currentUser"])
const finalTabs = currentUser?.is_superuser
  ? tabsConfig.slice(0, 3)
  : tabsConfig

return (
  <Container maxW="full">
    <Heading size="lg" textAlign={{ base: "center", md: "left" }} py={12}>
      User Settings
    </Heading>
    <Tabs variant="enclosed">
      <TabList>
        {finalTabs.map((tab, index) => (
          <Tab key={index}>{tab.title}</Tab>
        ))}
      </TabList>
      <TabPanels>
        {finalTabs.map((tab, index) => (
          <TabPanel key={index}>
            <tab.component />
          </TabPanel>
        ))}
      </TabPanels>
    </Tabs>
  </Container>
)
}
```

```tsx
import { createFileRoute } from '@tanstack/react-router'
import Comparison from '../..../components/Analytics/Comparison';

export const Route = createFileRoute('/_layout/analytics/comparison')({
  component: Comparison
})
```

```tsx
import { createFileRoute } from '@tanstack/react-router'
import Prediction from '../..../components/Analytics/Prediction';

```

```
export const Route = createFileRoute('/_layout/analytics/prediction')({
  component: Prediction
})``
```

```
```.tsx
```

```
import { createFileRoute } from '@tanstack/react-router';
import Trends from '../..../components/Analytics/Trends';
```

```
export const Route = createFileRoute('/_layout/analytics/trends')({
 component: Trends
})``
```

```
```.tsx
```

```
import { createFileRoute } from '@tanstack/react-router'
import Automatic from '../..../components/Reports/Automatic'
```

```
export const Route = createFileRoute('/_layout/reports/automatic')({
  component: Automatic
})``
```

```
```.tsx
```

```
import { createFileRoute } from "@tanstack/react-router"
import Generate from "../..../components/Reports/GenerateAndExport"
```

```
export const Route = createFileRoute("/_layout/reports/generate_and_export")({
 component: Generate,
})
``
```

```
```.py
```

```
from pathlib import Path
```

```
path: Path
```

```
for path in Path(".").glob("**/*.sh"):
    data = path.read_bytes()
    lf_data = data.replace(b"\r\n", b"\n")
    path.write_bytes(lf_data)
``
```