

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 «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Онлайн робочий кабінет»

Виконав студент групи ПЗ-19-3
ЗАХАРЧУК Максим Сергійович

Керівник роботи:
ГРИШКУН Євгеній Олександрович

Консультант:
КРАВЧЕНКО Світлана Миколаївна

Рецензент:
ЄФРЕМОВ Юрій Миколайович

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

«ЗАТВЕРДЖУЮ»

В.о. зав. кафедри інженерії
програмного забезпечення
Андрій МОРОЗОВ
«14» лютого 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЗАХАРЧУК Максим Сергійович**

Керівник роботи: **ГРИШКУН Євгеній Олександрович**

Тема роботи: **«Онлайн робочий кабінет»,**

затверджена Наказом закладу вищої освіти від «14» лютого 2023 р., №70с

Вихідні дані для роботи: **використання мови програмування JavaScript, фреймворку NestJS, бібліотеки ReactJS і мови запитів GraphQL для створення веб-додатку для спрощення і/або автоматизації певних процесів всередині компанії.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Кравченко С.М., Гришкун Є. О.	14.02.2023р.	14.02.2023р.
2	Кравченко С.М., Гришкун Є. О.	29.03.2023р.	29.03.2023р.
3	Кравченко С.М., Гришкун Є. О.	25.05.2023р.	25.05.2023р.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі	2 березня 2023 – 12 березня 2023	Виконано
2	Пошук, огляд та аналіз аналогічних розробок	12 березня 2023 – 19 березня 2023	Виконано
3	Формулювання технічного завдання та опрацювання літературних джерел	19 березня 2023 – 28 березня 2023	Виконано
4	Проектування структури системи	28 березня 2023 – 5 квітня 2023	Виконано
5	Написання програмного коду	5 квітня 2023 – 23 травня 2023	Виконано
6	Тестування роботи додатку	23 травня 2023 – 30 травня 2023	Виконано
7	Попередній захист	5 травня 2023	Виконано

РЕФЕРАТ

Випускна робота бакалавра складається з програмного комплексу онлайн сервісу робочого кабінету та пояснювальної записки. Пояснювальна записка до випускної роботи містить 90 сторінок, 40 ілюстрацій та 15 таблиць.

Метою роботи є створення веб-додатку «Робочий кабінет» за допомогою фреймворку NestJS для серверної частини, бібліотеки ReactJS для клієнтської частини і мови запитів GraphQL.

В роботі визначені головні завдання для розробки додатку, проаналізовано методи розв'язання поставлених задач, обґрунтовано вибір структури додатку, наведено алгоритм його роботи, а також архітектуру та взаємодію його основних компонентів. Також була описана програмна реалізація програми.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, РОБОЧИЙ КАБІНЕТ, ВЕБ-САЙТ, JAVASCRIPT, GRAPHQL.

					ІПЗ.КР.Б-121-23-ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Онлайн робочий кабінет Пояснювальна записка	Літ.	Арк.	Аркуші
Розроб.		М. ЗАХАРЧУК						
Керівник		Є. ГРИШКУН					4	
						Житомирська політехніка, група ІПЗ-19-3		
н. контр.		О. ТИЧИНА						
Зав. каф.		А. МОРОЗОВ						

ABSTRACT

The bachelor's thesis consists of an online service software complex for a work cabinet and an explanatory memorandum. The explanatory memorandum for the thesis includes 90 pages, 40 illustrations, and 15 tables.

The aim of the project is to create a web application called "Work Cabinet" using the NestJS framework for the server-side, the ReactJS library for the client-side, and the GraphQL query language.

The thesis defines the main tasks for developing the application, analyzes the methods for solving the stated challenges, justifies the choice of the application's structure, provides an algorithm of its functioning, as well as the architecture and interaction of its main components. The software implementation of the program is also described.

KEYWORDS: Web application, Work Cabinet, website, JavaScript, GraphQL.

					ИП3.КР.Б-121-23-П3	
						5

ЗМІСТ

РЕФЕРАТ.....	6
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	7
ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ РОБОЧОГО ОНЛАЙН КАБІНЕТУ	10
1.1 ПОСТАНОВКА ЗАДАЧІ	10
1.2 АНАЛІЗ АНАЛОГІВ ПРОГРАМНОГО ПРОДУКТУ	11
1.3 ВИБІР АРХІТЕКТУРИ ВЕБ-ДОДАТКУ	15
1.4 ОБҐРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ТА ВИМОГИ ДО АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ.....	17
Висновки до першого розділу.	20
РОЗДІЛ 2. ПРОЕКТУВАННЯ РОБОЧОГО ОНЛАЙН КАБІНЕТУ	21
2.1 Визначення варіантів використання і структури додатку	21
2.2 Розробка бази даних системи.....	31
2.3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ АЛГОРИТМІВ РОБОТИ ДОДАТКУ	38
2.4 РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ ДОДАТКУ	41
Висновки до другого розділу	50
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ.....	51
3.1 Порядок встановлення та налаштування параметрів сервісу.....	51
3.2 Структура інтерфейсу додатку «Онлайн робочий кабінет».....	53
3.3 Тестування роботи додатку	66
Висновки до третього розділу	68
ВИСНОВКИ	69
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	71
ДОДАТКИ.....	74

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

JS – JavaScript

API – прикладний програмний інтерфейс

БД – база даних

ПЗ – програмне забезпечення

UI – інтерфейс користувача

ПК – персональний комп'ютер

СУБД – система управління базами даних

API – application programming interface – програмний інтерфейс додатка

DOM – document object model – об'єктна модель документа

SPA – single page application – одно сторінковий додаток

					ІПЗ.КР.Б-121-23-ПЗ	
						7

ВСТУП

Актуальність теми. В сучасному світі, де інформаційні технології займають центральне місце у нашому житті, розвиток інтернету та комп'ютерних систем відкриває безліч нових можливостей у різних сферах діяльності. Одним із напрямків, де ці технології мають особливу вагу, є управління бізнесом та робота з персоналом.

Робочий онлайн-кабінет стає все більш актуальним і важливим інструментом для ефективного управління бізнесом та взаємодії зі співробітниками. Він надає зручність і швидкість взаємодії, спрощує процеси роботи та сприяє збільшенню продуктивності.

Однією з основних переваг робочого онлайн-кабінету є можливість доступу до необхідної інформації з будь-якого місця та в будь-який час. Це особливо корисно в умовах, коли багато компаній працюють в розподіленому середовищі, де співробітники знаходяться на різних географічних об'єктах або працюють з віддалених локацій. Робочий онлайн-кабінет дозволяє злегка керувати процесами, надавати співробітникам необхідні інструменти та матеріали для ефективної роботи.

Ще однією вагомою перевагою робочого онлайн-кабінету є спрощення адміністративних процедур. Завдяки цифровізації документів та автоматизації бізнес-процесів, можна зменшити час, витрачений на виконання рутинних завдань, і зосередитися на стратегічних аспектах розвитку компанії. Робочий онлайн-кабінет дозволяє швидко обмінюватися даними, документами, спілкуватися з колегами та керівництвом, що прискорює прийняття рішень і покращує комунікацію всередині організації.

Крім того, використання робочого онлайн-кабінету сприяє збереженню даних та забезпеченню їх безпеки. Це особливо актуально в контексті зростаючої кількості кіберзагроз та необхідності захисту конфіденційної інформації. Робочий онлайн-кабінет дозволяє встановлювати рівні доступу та контролювати права користувачів, зменшуючи ризик витоку чи несанкціонованого використання даних.

Загалом, у сучасному світі, де час - це один із найцінніших ресурсів, робочий онлайн-кабінет стає незамінним інструментом для підвищення продуктивності та ефективності управління бізнесом. Його використання дозволяє прискорити процеси, забезпечити зручну комунікацію та співпрацю між співробітниками та керівництвом, а також зберегти і захистити важливу інформацію. Тому розробка та використання робочого онлайн-кабінету має велику актуальність для сучасних організацій у всіх галузях бізнесу.

Метою випускної роботи є розробка веб-додатку «Робочий кабінет».

Встановлена мета обумовлює наступні завдання:

- проведення аналізу вимог до онлайн робочого кабінету;
- аналіз аналогів програмного продукту;
- вибір архітектури компонентів системи;
- обґрунтування та вибір засобів реалізації;
- визначення варіантів використання;
- розробка веб-ресурсу та реалізація всіх визначених вимог до програмного забезпечення;
- проектування алгоритмів системи;
- опис структури інтерфейсу додатку;
- тестування веб-додатку.

Об'єктом дослідження є спрощення/автоматизація певних процесів всередині компанії.

Предметом дослідження є використання мови програмування JavaScript та мови запитів GraphQL для розробки робочого онлайн кабінету.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ РОБОЧОГО ОНЛАЙН КАБІНЕТУ

1.1 Постановка задачі

В випускній роботі необхідно розробити веб-додаток, що допоможе покращити взаємодію між робітниками в середині компанії, автоматизувати/спростити певні процеси, що займали б більше часу та зусиль.

Додаток має складатися із серверної і клієнтської частини.

Основними етапами розробки описаного сервісу є:

1. Розробка дизайну користувацького інтерфейсу.

1.1. Створення загальної концепції дизайну додатку.

1.2. Створення окремих елементів дизайну за допомогою графічних редакторів.

1.3. Розробка JSX-компонентів.

1.3.1. Макет шаблону.

1.3.2. Макет головної сторінки.

1.3.3. Макети форми входу.

1.3.4. Макет сторінки користувачів.

1.3.5. Макет сторінки відпусток.

1.3.6. Макет сторінки налаштувань.

2. Розробка серверного додатку.

2.1. Зв'язок з БД.

2.2. Реалізація функціоналу з сутностями.

2.3. Система авторизації і аутентифікації.

2.4. Налаштування відправки емейлів.

2.5. Налаштування GraphQL сервера.

2.6. Налаштування задач за розкладом.

3. Розробка клієнтського додатку.

3.1. Налаштування GraphQL.

- 3.2. Налаштування авторизації і аутентифікації.
- 3.3. Створення роутингу.
- 3.4. Автогенерація GraphQL типів.
- 3.5. Створення елементів підвантаження та помилок.
- 3.6. Створення необхідних хелперів і хуків.
- 3.7. Створення сторінок.
- 3.8. Налаштування meta даних.
- 3.9. Створення клієнтської валідації.

4. Розробка плану тестування системи:

- 4.1. Функціональне тестування.
- 4.2. Тестування графічного інтерфейсу.

Результатом виконання плану дій є створення додатку, який дозволить певній компанії інтегрувати свої внутрішні процеси всередину додатку і цим самим спростити робітникам певні дії.

1.2 Аналіз аналогів програмного продукту

Розробка конкурентоспроможного додатку, яким будуть користуватися багато клієнтів, потребує аналізу його аналогів. В процесі проведення аналізу були досліджені наступні продукти, схожі за своєю концепцією та функціоналом: HiBob, Huma, PeopleForce.

HiBob (Рисунок 1.1) – HRM платформа. Призначення для підвищення продуктивності, залученості та збереження персоналу.

Основними перевагами HiBob є:

- Досить широкий функціонал;
- Приємний інтерфейс;
- Можливість кастомізації;

Недоліками є:

- Англійська мова;
- Платний.

- Потребує багато часу для старту і ознайомлення;
- Незручне редагування профілю.

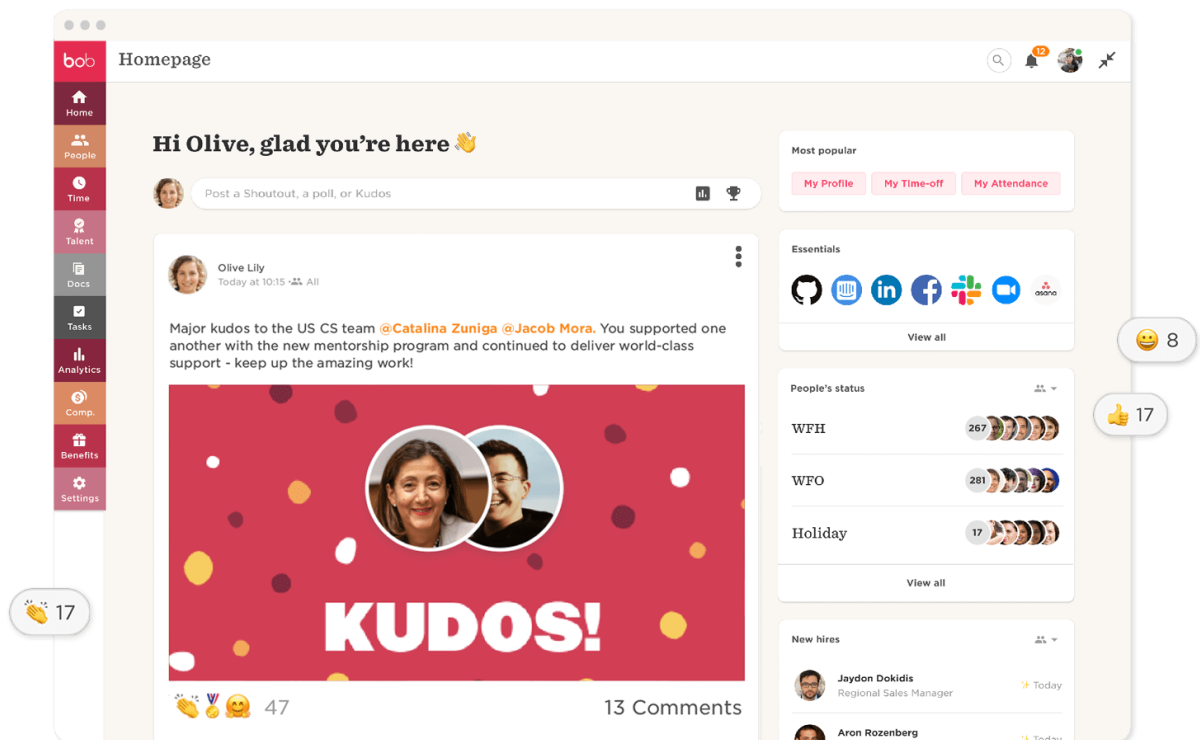


Рисунок 1.1. Головна сторінка HiBob

Hurma (Рисунок 1.2) – українська HRM система.

Основними перевагами Hurma є:

- Підтримка різних мов;
- Система рекрутингу;
- Система контролю робочого часу;

Недоліками є:

- Платний.
- Відсутність системи дописів.

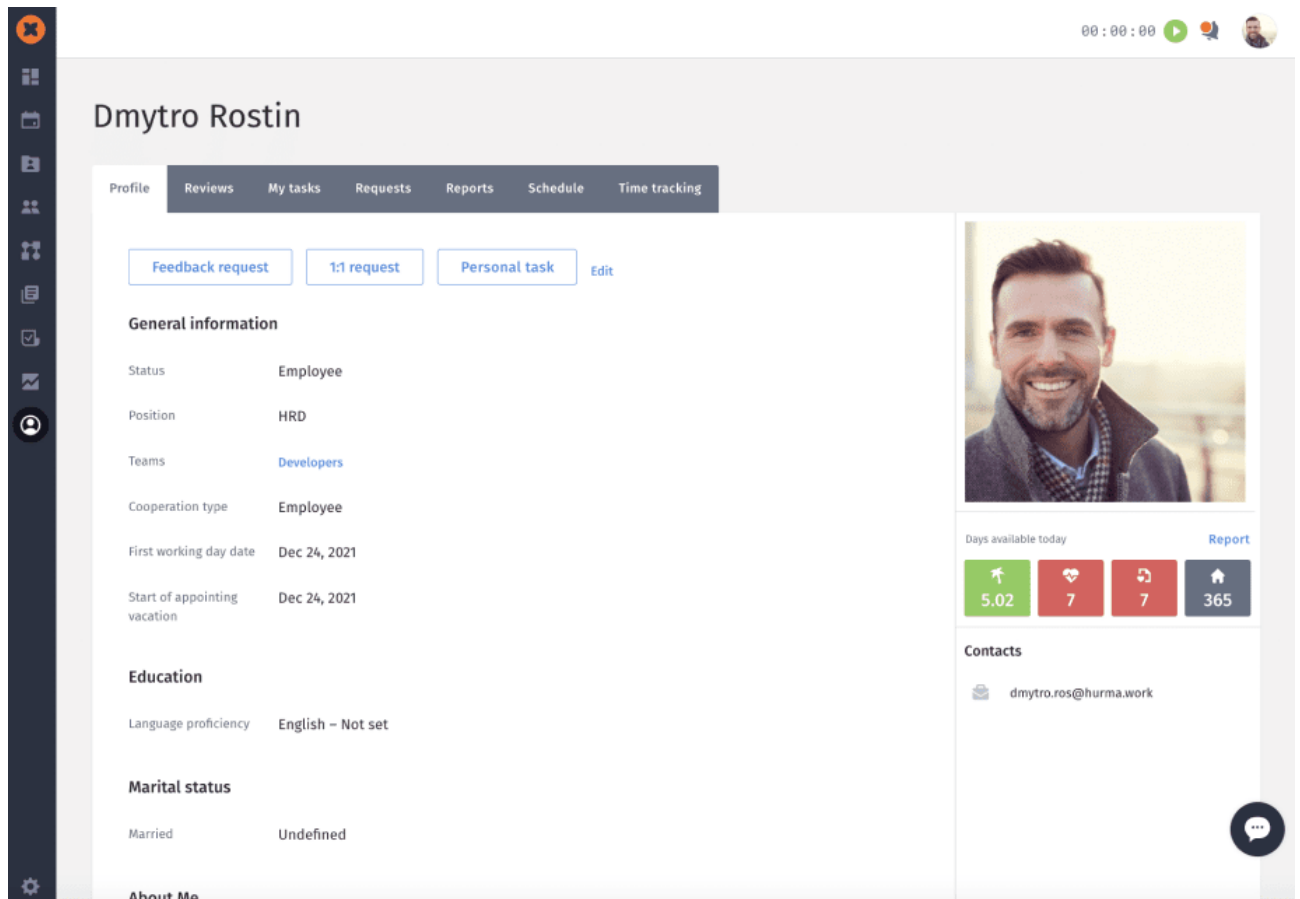


Рисунок 1.2 – Сторінка профілю в Hurma

PeopleForce (Рисунок 1.3) – це платформа управління персоналом, яка об'єднує та оптимізує завдання з управління персоналом та кадровим менеджментом в одній системі.

Основними перевагами PeopleForce є:

- Підтримка декількох мов;
- Наявність бази знань;
- Система запитів;

Недоліками є:

- Платний.

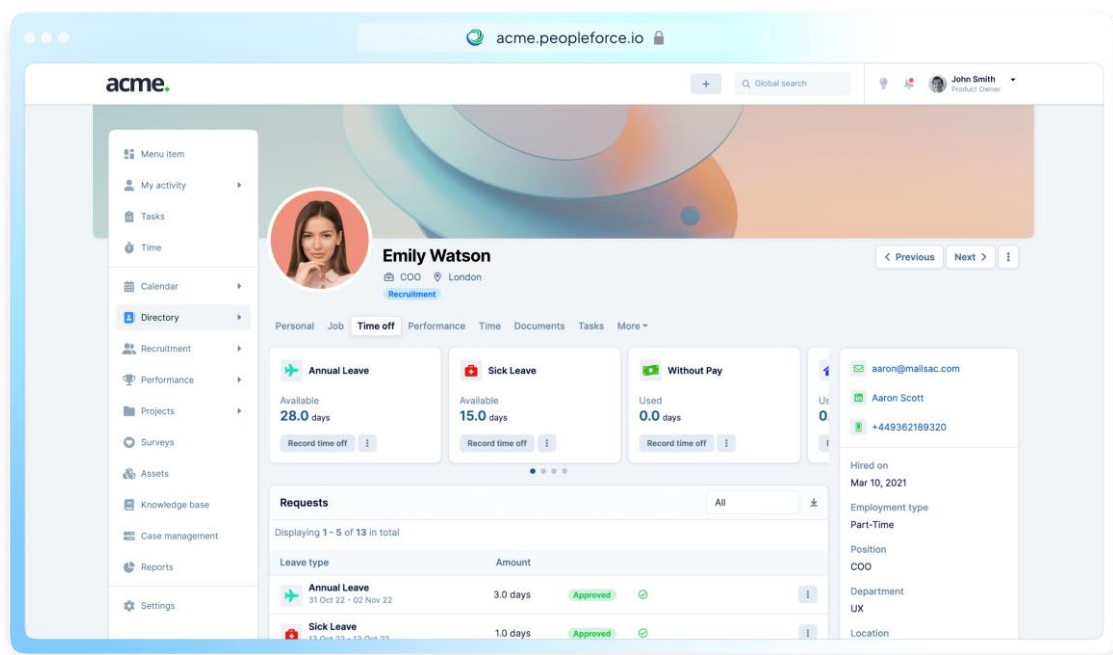


Рисунок 1.3 – Сторінка відпусток PeopleForce

Виходячи з наявних переваг та недоліків, складемо порівняльну таблицю вищезгаданих програмних аналогів (табл. 1.1).

Таблиця 1.1

Порівняння аналогів

Характеристика	HiBob	Hurma	PeopleForce
Ціна	Платний	Платний	Платний
Підтримка декількох мов	-	+	+
Наявність бази знань	-	-	-
Система дописів	+	-	-
Система рекрутингу	-	+	+

Переглянувши аналогічні сервіси, можна сказати що основний функціонал в кожного з них дуже схожий, відмінності полягають лише в мові, що підтримується і додаткових функціях.

Враховуючи вищезгадані факти, було прийнято рішення розробити схожий за концепцією веб-додаток, з основним і головним функціоналом, що допоможе компаніям автоматизувати і спростити основні процеси.

Таким чином, даний додаток має бути дуже простим у використанні, з мінімалістичним дизайном; бути відмовостійким через потенційну велику навантаженість; містити невеликий функціонал для зворотнього зв'язку між водієм і пасажиром; бути повністю автоматизованим і незалежним задля оптимізації робочого штату і мінімізації бюрократичних рішень.

1.3 Вибір архітектури веб-додатку

В процесі розробки веб-додатку "Робочий кабінет" було проведено детальне обговорення та аналіз різних варіантів архітектурних рішень з метою вибору оптимальної моделі, що забезпечує ефективну взаємодію між серверною та клієнтською частинами додатку, використовуючи GraphQL як мову запитів [14].

З урахуванням вимог до проекту та переваг GraphQL, було прийнято рішення розділити серверний і клієнтський додатки.

Для серверного додатку було обрано модульну архітектуру. Модульна архітектура є одним з поширених підходів до розробки веб-додатків. Вона спрямована на розділення функціональності додатку на окремі модулі або компоненти, які можуть працювати незалежно один від одного, але взаємодіяти для досягнення загальної мети.

Основна ідея модульної архітектури полягає в тому, щоб розділити веб-додаток на логічні модулі, які виконують певні функції або відповідають за конкретні аспекти додатку. Кожен модуль може мати свою внутрішню структуру і бути відповідальним за власні задачі, такі як авторизація, обробка запитів, робота з базою даних, відображення інтерфейсу користувача тощо.

Переваги модульної архітектури веб-додатків:

Модульність: Модулі додатку можуть бути розроблені незалежно один від одного, що полегшує розподіл роботи між розробниками та підтримку коду.

- **Перевикористання:** Логічні модулі можуть бути використані в інших проектах або в різних частинах одного додатку, забезпечуючи ефективне перевикористання коду.

- **Складність:** Модульна архітектура дозволяє керувати складністю додатка шляхом розділення його на менші, легше управляємі модулі.
- **Тестування:** Кожен модуль може бути протестований окремо, що полегшує виявлення та виправлення помилок.
- **Масштабованість:** Завдяки модульності, можливо масштабувати лише певні модулі або компоненти додатку, не впливаючи на решту.

При розробці веб-додатку з модульною архітектурою, важливо ретельно спланувати структуру модулів, визначити їхні взаємозв'язки та правильно організувати комунікацію між ними. Також рекомендується використовувати популярні фреймворки або бібліотеки, які підтримують модульну архітектуру і надають інструменти для реалізації цього підходу, наприклад, NestJS.

Для створення клієнтського додатку було прийняте рішення використовувати архітектуру SPA(single page application).

SPA (Single Page Application) - це підхід до розробки веб-додатків, при якому весь контент додатку завантажується одним разом при першому завантаженні сторінки, а далі весь взаємодія з додатком відбувається без повторних перезавантажень сторінки.

Основні особливості SPA:

- **Асинхронна навігація:** SPA використовує асинхронну навігацію, що дозволяє переходити між сторінками без перезавантаження, використовуючи AJAX-запити або бібліотеки для маршрутизації.
- **Один HTML-документ:** Весь контент додатку завантажується одним разом, а змінюється лише внутрішній контент без повторного завантаження сторінки.
- **Більш ресурсозберігаючий:** SPA мінімізує кількість запитів до сервера, оскільки не потрібно завантажувати повністю нову сторінку при кожній взаємодії.

- Висока швидкість відгуку: SPA надає більш плавну та швидку взаємодію з додатком, оскільки більшість розрахунків відбувається на клієнтській стороні без потреби звертатися до сервера.
- Розширена можливість валідації та перевірки даних: SPA дозволяє валідувати та перевіряти дані на клієнтській стороні перед відправкою на сервер.
- Підтримка офлайн-режиму: Завдяки кешуванню ресурсів, SPA може підтримувати обмежену роботу в офлайн-режимі.

Для розробки SPA можна використовувати фреймворки та бібліотеки, такі як Angular, React або Vue.js, які надають потужні інструменти для створення таких додатків.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Розробка веб-додатку відбуватиметься у середовищі розробки VisualStudio Code. Для розробки додатку були обрані наступні засоби:

1. JavaScript є однією з найпоширеніших мов програмування, яка використовується для розробки веб-додатків. Вибір JavaScript забезпечує широкий вибір бібліотек, фреймворків та інструментів, які полегшують розробку та підтримку коду.

2. NestJS є фреймворком на основі Node.js, який надає структуру і стандартизацію для розробки серверної частини додатків. Він використовує TypeScript, що дозволяє писати код з типізацією і полегшує розробку та підтримку кодової бази [1].

3. ReactJS є одним з найпопулярніших фреймворків для розробки користувацького інтерфейсу. Він пропонує компонентний підхід, що дозволяє побудувати модульну та повторно використововану структуру додатку [2].

4. GraphQL - це запитова мова та середовище виконання, яке дозволяє ефективно запитувати та отримувати дані з сервера. Використання GraphQL дозволяє зменшити надлишковість даних та забезпечити гнучкість у виборі необхідних даних [14].

					ІПЗ.КР.Б-121-23-ПЗ	
						17

5. Apollo - це набір інструментів для розробки GraphQL-додатків. Він надає можливості для кешування, управління станом та побудови клієнтської частини додатку, спрощуючи роботу з GraphQL [19].

6. React Router - це бібліотека для навігації в додатку React. Вона дозволяє створювати динамічну маршрутизацію, що допомагає керувати переходами між сторінками та компонентами.

7. TypeScript: Вибір TypeScript має кілька переваг. Він є типізованою мовою програмування, що дозволяє виявляти помилки на етапі компіляції і полегшує розробку масштабних проектів. TypeScript також підтримує модульну архітектуру та покращує розуміння та підтримку коду.

8. Material-UI є популярною бібліотекою компонентів для React, яка реалізує концепцію Material Design. Використання Mui сприяє швидкій розробці інтерфейсу, оскільки надає готові та стильні компоненти, які можна легко налаштувати та використовувати.

9. PostgreSQL є потужною та розширеною системою керування базами даних з великою кількістю функцій і можливостей. Він підтримує реляційну модель даних, транзакції, індекси, географічні дані, текстовий пошук та багато іншого. Вибір PostgreSQL дозволяє ефективно управляти даними [32].

10. Об'єктно-реляційне відображення: TypeORM є одним з популярних ORM-фреймворків (Object-Relational Mapping), який дозволяє працювати з базою даних у зручній об'єктно-орієнтованій парадигмі. Він надає можливість використовувати класи та об'єкти для взаємодії з даними, замість написання складних SQL-запитів вручну. TypeORM допомагає зменшити кількість бойлерплейту і прискорює розробку додатків [31].

Вказані інструментальні засоби забезпечують чудову швидкодію сервісу при великих навантаженнях даних, Клієнтська частина вирізняється своїм привабливим та сучасним дизайном, реалізованим за допомогою допоміжних бібліотек.

Обґрунтування вимог до апаратного забезпечення:

Вимоги до апаратного забезпечення для веб-додатку "Онлайн робочий кабінет" можуть варіюватися в залежності від рівня складності додатку, його функціональності та навантаження. Однак, ось загальні рекомендації для веб-додатків:

1. Процесор: Рекомендується використовувати процесор з достатньою потужністю, щоб забезпечити швидку обробку запитів і взаємодію з базою даних. Більш потужні процесори з більшим числом ядер допоможуть розподілити навантаження більш ефективно.

2. Оперативна пам'ять: Великий обсяг оперативної пам'яті дозволить додатку ефективно працювати з багатими на дані операціями та обробляти багатокористувацькі запити. Рекомендується мати не менше 4 ГБ оперативної пам'яті, але точні вимоги можуть залежати від обсягу даних та навантаження.

3. Жорсткий диск: В залежності від обсягу даних, які додаток буде обробляти, може бути необхідно мати достатній обсяг простору на жорсткому диску для збереження даних та резервних копій.

4. Мережевий доступ: Надійне підключення до Інтернету з достатньою пропускною здатністю є важливим для забезпечення швидкого та безперебійного доступу до веб-додатку.

5. Безпека: Для захисту веб-додатку від злоумисників рекомендується використовувати фаєрволи, антивірусне програмне забезпечення та регулярно оновлювати системне програмне забезпечення.

6. Масштабованість: Якщо передбачається зростання обсягу користувачів або навантаження на додаток, слід розглянути можливості масштабування, такі як використання хмарних сервісів або розподіленого обчислення.

Висновки до першого розділу.

Проведений аналіз предметної області допоміг визначити основні вимоги до веб-додатку, основною задачею якого є автоматизація і спрощення процесів всередині компанії.

В процесі дослідження аналогічних і схожих за концепцією продуктів були проаналізовані та порівняні різні аналогічні платформи та сервіси. Крім цього, були створені основні вимоги для власно розробленого додатку.

Було проведено аналіз різних архітектур для створення програми та обрано найбільш підходящу та просту у використанні.

На основі сформованих вимог та критерій був сформований набір основних та допоміжних технологій і засобів реалізації для розробки веб-додатку.

					ІПЗ.КР.Б-121-23-ПЗ	
						20

РОЗДІЛ 2. ПРОЕКТУВАННЯ РОБОЧОГО ОНЛАЙН КАБІНЕТУ

2.1 Визначення варіантів використання і структури додатку

Веб-додаток поєднує в собі взаємодію користувачів різних ролей: робітник, помічник і адміністратор. Задля наглядності взаємодії між елементами додатку було створено діаграму варіантів використання (Рисунок 2.1).

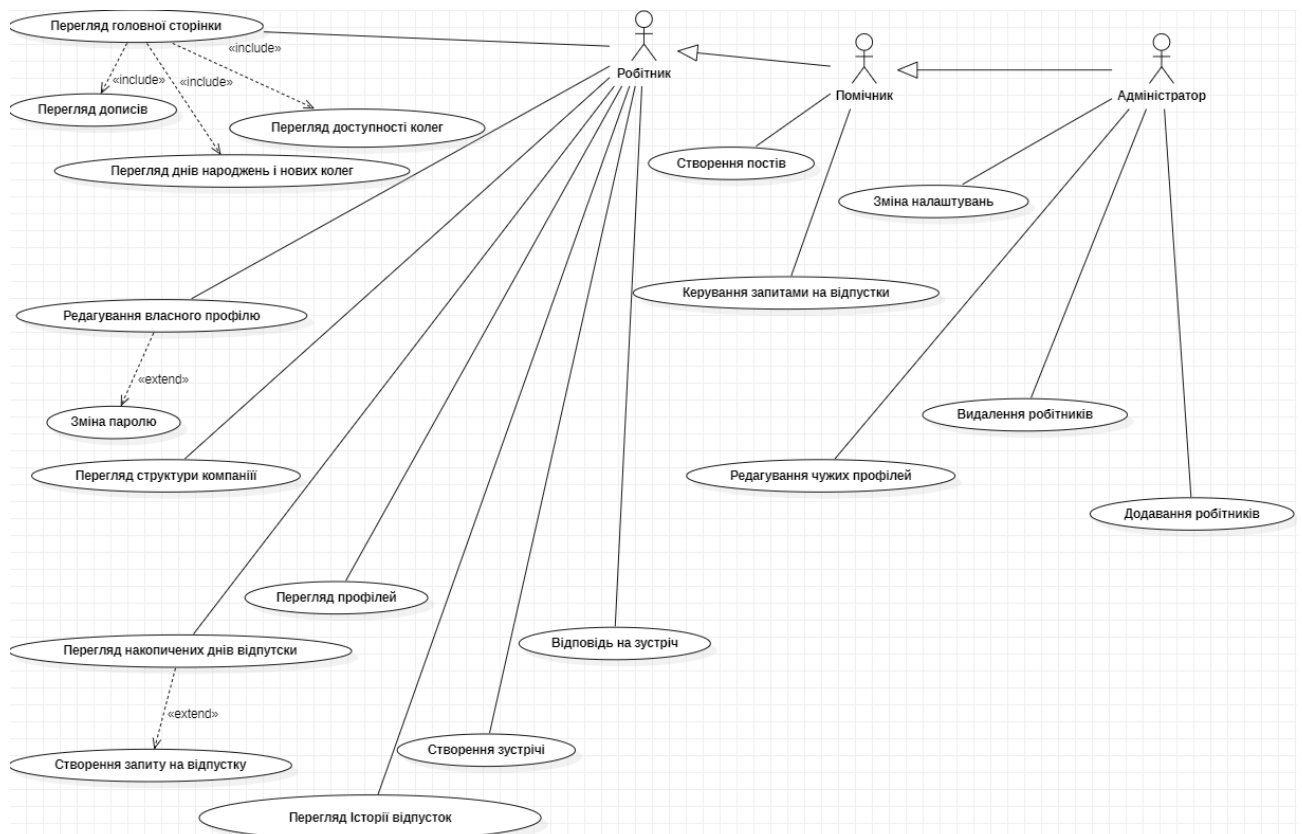


Рисунок 2.1 - Діаграма варіантів використання

Вимоги користувачів (робітники)

1. Можливість редагування власного профілю;
2. Можливість перегляду профілю співробітників;
3. Можливість запросити відпустку;
4. Можливість переглядати структуру компанії;
5. Можливість переглядати інформацію про накопичені дні відпустки;
6. Можливість перегляду доступності співробітників;
7. Можливість перегляду днів народжень співробітників

Вимоги користувачів (адміністрація і помічники):

1. Публікування новин
2. Створення, редагування та видалення профілів робітників
3. Підтвердження заяв на відпустки

Бізнес вимоги

1. Основні цілі: проект створюється з метою спрощення взаємодії між працівниками компанії та автоматизації певних процесів.
2. Представлення проекту: проект буде реалізовано у вигляді сайту, що містить структуровану інформацію по даній тематичній області.

Характеристика об'єкта комп'ютеризації:

Користувач додатку має достатньо можливостей та зручно розроблений інтерфейс для взаємодії з співробітниками та адміністрацією.

Функціональні вимоги:

1. Авторизація в системі: В системі повинна бути представлена можливість реєстрації користувача та присвоєння йому відповідної ролі (робітник, адміністратор).
2. Можливість збереження інформації: Система повинна зберігати інформацію і надавати можливість керувати нею.

Нефункціональні вимоги:

1. Сприйняття
 - Час, потрібний для навчання інструментами роботи з інформаційною системою для звичайних користувачів – 2-3 години, а для досвідчених – 1 година
 - Час відповіді системи для звичайних запитів не повинен перевищувати 5 секунд, а для більш складних запитів – 10 сек.
 - Інтерфейс представлення ІС повинен бути інтуїтивно зручним для користувача та не вимагати від нього додаткової підготовки.
2. Надійність
 - Доступність – час, потрібний для обслуговування системи не повинен перевищувати 10% від загального часу роботи.

- Середній час безперервної роботи – 20 робочих днів.
- Максимальна норма помилок та дефектів в роботі системи – 1 помилка на 1000 запитів користувача.

3. Продуктивність

- Система повинна підтримувати мінімум 100 одночасно працюючих користувачів, пов'язаних з спільною базою даних.

4. Можливість експлуатації

- Масштабування – система повинна мати можливість збільшувати потужності (продуктивність), зі збільшенням користувачів таким чином, щоб це аж ні як негативно не відобразилося на її роботі..

Аналіз функціональних вимог дозволив виділити наступні сутності, що забезпечать програмну реалізацію додатку. На рисунку 2.2. наведено діаграму основних класів, які використовуються для його роботи.

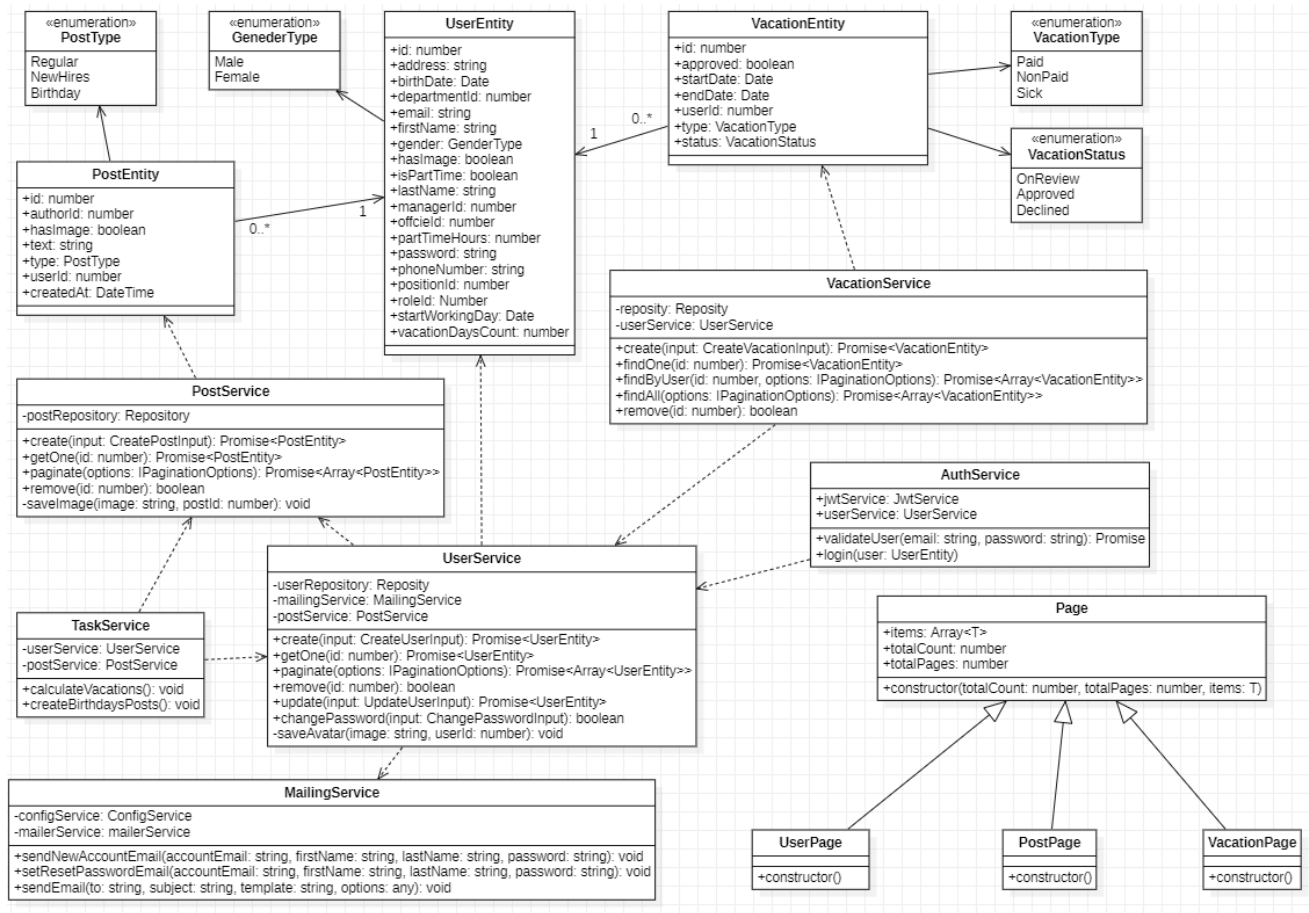


Рисунок 2.2 – Діаграма основних класів

UserEntity – клас, що описує сутність користувача:

id (тип: number): ідентифікатор користувача.

name (тип: string): ім'я користувача.

address (тип: string): адреса користувача.

birthDate (тип: Date): дата народження користувача.

departmentId (тип: number): ідентифікатор відділу, до якого належить користувач.

email (тип: string): електронна пошта користувача.

firstName (тип: string): ім'я користувача.

gender (тип: GenderType): стать користувача.

hasImage (тип: boolean): прапорець, що позначає, чи має користувач зображення.

isPartTime (тип: boolean): прапорець, що позначає, чи працює користувач на неповний робочий день.

lastName (тип: string): прізвище користувача.

managerId (тип: number): ідентифікатор керівника користувача.

officeId (тип: number): ідентифікатор офісу, в якому працює користувач.

partTimeHours (тип: number): кількість годин, які користувач працює на неповний робочий день.

password (тип: string): пароль користувача.

phoneNumber (тип: string): номер телефону користувача.

positionId (тип: number): ідентифікатор посади користувача.

roleId (тип: Number): ідентифікатор ролі користувача.

VacationEntity – клас, що описує сутність відпустки:

id (тип: number) - ідентифікатор відпустки

approved (тип: boolean) - прапорець, що позначає, чи схвалена відпустка користувача

startDate (тип: Date) - дата початку відпустки

endDate (тип: Date) - дата завершення відпустки

userId (тип: number) - ідентифікатор користувача, пов'язаний з відпусткою
type (тип: VacationType) - тип відпустки
status (тип: VacationStatus) - статус відпустки

PostEntity – клас, що описує сутність поста:

id (тип: number) - ідентифікатор поста
authorId (тип: number) - ідентифікатор автора поста
hasImage (тип: boolean) - прапорець, що позначає наявність зображення в
пості
text (тип: string) - текст поста
type (тип: PostType) - тип поста
userId (тип: number) - ідентифікатор користувача, пов'язаний з постом
createdAt (тип: DateTime) - дата створення поста

UserService – клас для роботи з користувачами:

userRepository: об'єкт типу Repository - репозиторій, який забезпечує доступ
до даних користувача.

mailingService: об'єкт типу MailingService - сервіс для відправки
повідомлень користувачам.

postService: об'єкт типу PostService - сервіс для операцій з
повідомленнями.

create(input: CreateUserInput): Promise<UserEntity>: створює нового
користувача на основі вхідних даних і повертає об'єкт UserEntity.

getOne(id: number): Promise<UserEntity>: отримує користувача за його
ідентифікатором і повертає об'єкт UserEntity.

paginate(options: IPaginationOptions): Promise<Array<UserEntity>>:
здійснює посторінкове отримання списку користувачів залежно від наданих
опцій та повертає масив об'єктів UserEntity.

remove(id: number): boolean: видаляє користувача за його ідентифікатором
і повертає значення true, якщо операція успішна.

`update(input: UpdateUserInput): Promise<UserEntity>`: оновлює дані користувача на основі вхідних даних і повертає оновлений об'єкт `UserEntity`.

`changePassword(input: ChangePasswordInput): boolean`: змінює пароль користувача на основі вхідних даних і повертає значення `true`, якщо операція успішна.

`saveAvatar(image: string, userId: number): void`: зберігає аватар користувача на основі вхідних даних.

VacationService – клас для роботи з відпустками:

`repository` (тип: приватне поле): Представляє об'єкт типу `Repository` і використовується для збереження та отримання даних про відпустки.

`userService` (тип: приватне поле): Представляє об'єкт типу `UserService` і використовується для взаємодії з користувачем, пов'язаним з відпустками.

`create(input: CreateVacationInput): Promise<VacationEntity>` (тип: публічний метод): Створює новий об'єкт `VacationEntity` на основі вхідних даних `input` і повертає його у вигляді об'єкта `Promise`.

`findOne(id: number): Promise<VacationEntity>` (тип: публічний метод): Знаходить та повертає об'єкт `VacationEntity` за його ідентифікатором `id` у вигляді об'єкта `Promise`.

`findByUser(id: number, options: IPaginationOptions): Promise<Array<VacationEntity>>` (тип: публічний метод): Знаходить та повертає масив об'єктів `VacationEntity`, пов'язаних з користувачем за його ідентифікатором `id`, з додатковими опціями пагінації у вигляді об'єкта `IPaginationOptions`.

`findAll(options: IPaginationOptions): Promise<Array<VacationEntity>>` (тип: публічний метод): Знаходить та повертає всі об'єкти `VacationEntity` з додатковими опціями пагінації у вигляді об'єкта `IPaginationOptions`.

`remove(id: number): boolean` (тип: публічний метод): Видаляє об'єкт `VacationEntity` за його ідентифікатором `id` і повертає значення `true`, якщо

видалення пройшло успішно, або false, якщо виникла помилка під час видалення.

PostService – клас для роботи з постами:

postRepository (тип: Repository) - приватне поле, яке представляє репозиторій постів.

create(input: CreatePostInput): Promise<PostEntity> - публічний метод, який створює новий пост з використанням наданого вхідного об'єкту CreatePostInput і повертає об'єкт Promise, який містить створений пост (PostEntity).

getOne(id: number): Promise<PostEntity> - публічний метод, який отримує пост за його ідентифікатором (id) і повертає об'єкт Promise, який містить знайдений пост (PostEntity).

paginate(options: IPaginationOptions): Promise<Array<PostEntity>> - публічний метод, який здійснює пагінацію постів за допомогою наданих параметрів options і повертає об'єкт Promise, який містить масив постів (Array<PostEntity>).

remove(id: number): boolean - публічний метод, який видаляє пост за його ідентифікатором (id) і повертає значення boolean, яке показує, чи вдалося видалити пост.

saveImage(image: string, postId: number): void - приватний метод, який зберігає зображення (image) для вказаного поста з використанням його ідентифікатора (postId).

AuthService – клас для роботи з аутентифікацією:

jwtService: JwtService - сервіс JWT, який використовується для генерації та перевірки токенів.

userService: UserService - сервіс користувача, який надає доступ до даних користувачів.

`validateUser(email: string, password: string): Promise` - метод для перевірки користувача. Він приймає електронну пошту та пароль, перевіряє їх на відповідність збереженим даним користувача та повертає обіцянку (Promise).

`login(user: UserEntity)` - метод для входу користувача. Він приймає об'єкт користувача (UserEntity) і виконує процес авторизації.

MailingService – клас для роботи з електронними листами:

`configService` (приватне поле типу `ConfigService`): Представляє сервіс конфігурації, який використовується для отримання налаштувань поштового сервісу.

`mailerService` (приватне поле типу `mailerService`): Представляє поштовий сервіс, який використовується для надсилання електронних листів.

`sendNewAccountEmail(accountEmail: string, firstName: string, lastName: string, password: string): void`: Метод для надсилання електронного листа з інформацією про новий обліковий запис. Приймає електронну адресу облікового запису, ім'я, прізвище та пароль як аргументи.

`setResetPasswordEmail(accountEmail: string, firstName: string, lastName: string, password: string): void`: Метод для встановлення електронного листа для скидання пароля. Приймає електронну адресу облікового запису, ім'я, прізвище та новий пароль як аргументи.

`sendEmail(to: string, subject: string, template: string, options: any): void`: Метод для надсилання електронного листа. Приймає отримувача, тему, шаблон листа та додаткові параметри як аргументи.

TaskService – клас для роботи з періодичним діями:

`userService: UserService` (приватне поле) - представляє об'єкт сервісу користувачів, який використовується для взаємодії з користувачами в системі.

`postService: PostService` (приватне поле) - представляє об'єкт сервісу постів, який використовується для взаємодії з постами в системі.

`calculateVacations(): void` - публічний метод, який виконує розрахунок та обробку відпусток згідно з розкладом. Цей метод повинен перевіряти і оновлювати відпустки співробітників кожного першого числа місяця.

`createBirthdaysPosts(): void` - публічний метод, який створює та публікує вітальні повідомлення з нагоди дня народження користувачів згідно з розкладом. Цей метод повинен автоматично створювати та публікувати повідомлення з вітаннями, чия дата народження припадає на поточний день.

Page – клас, що описує сторінку з набором сутностей:

`items: Array<T>` (публічне): Масив об'єктів, які представляють елементи сторінки.

`totalCount: number` (публічне): Загальна кількість елементів, які доступні для даної операції.

`totalPages: number` (публічне): Загальна кількість сторінок, які доступні для даної операції.

`constructor(totalCount: number, totalPages: number, items: T)` (публічний): Конструктор класу `Page`, який приймає загальну кількість елементів (`totalCount`), загальну кількість сторінок (`totalPages`) і масив елементів (`items`). Цей конструктор ініціалізує відповідні поля класу.

На рис 2.3 наведено діаграму модулів системи:

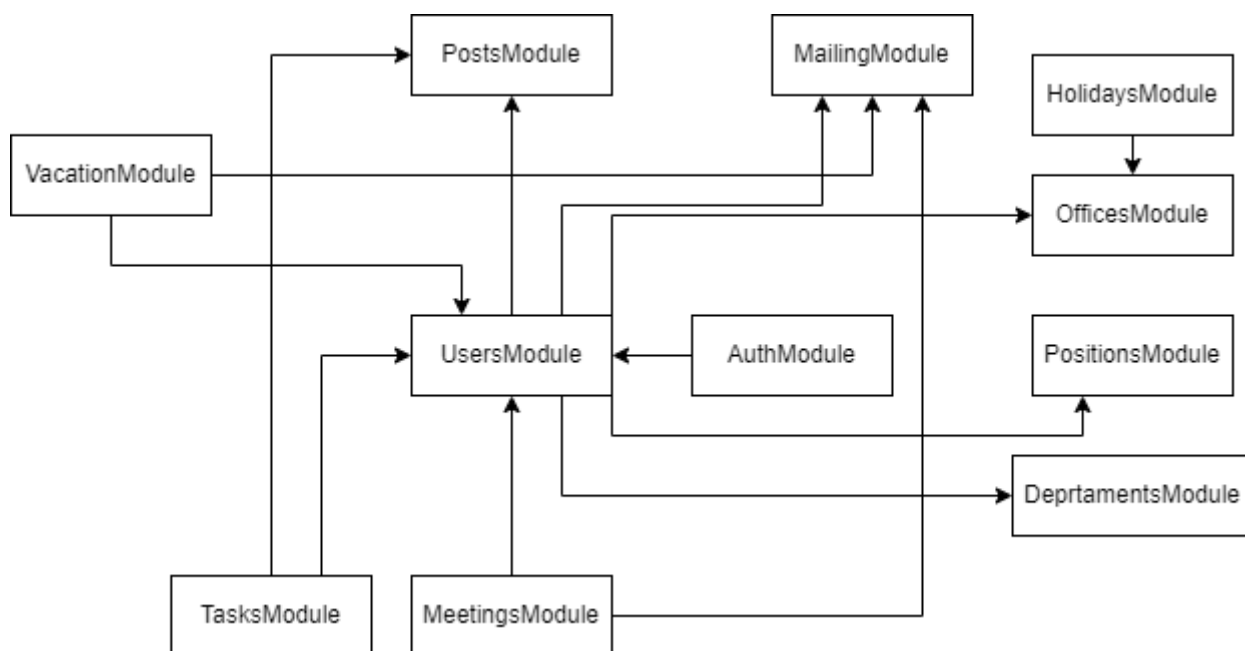


Рисунок 2.3. Діаграма модулів системи

На рисунках 2.3 – 2.4 наведено діаграму класів для окремих модулів системи: модуль користувачів і модуль аутентифікації.

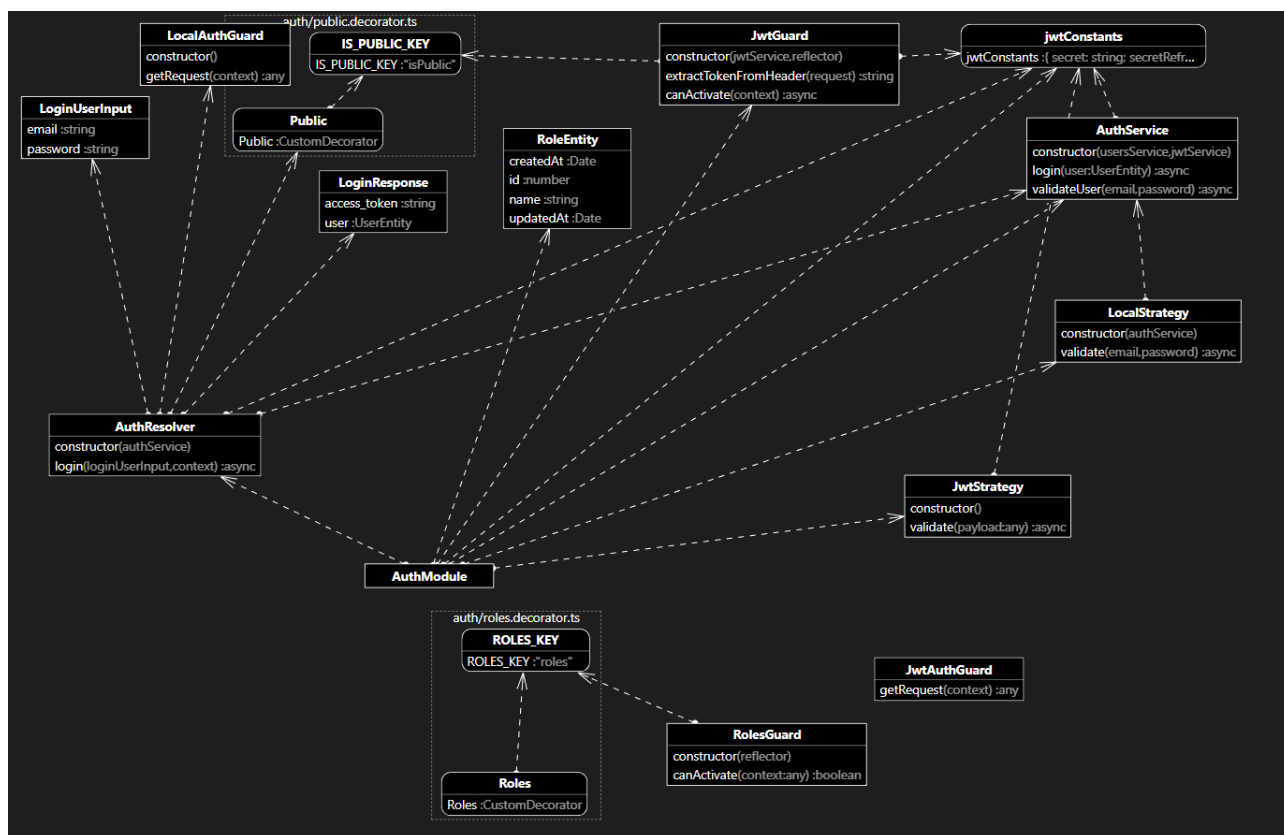


Рисунок 2.3 - Діаграма класів Auth модуля

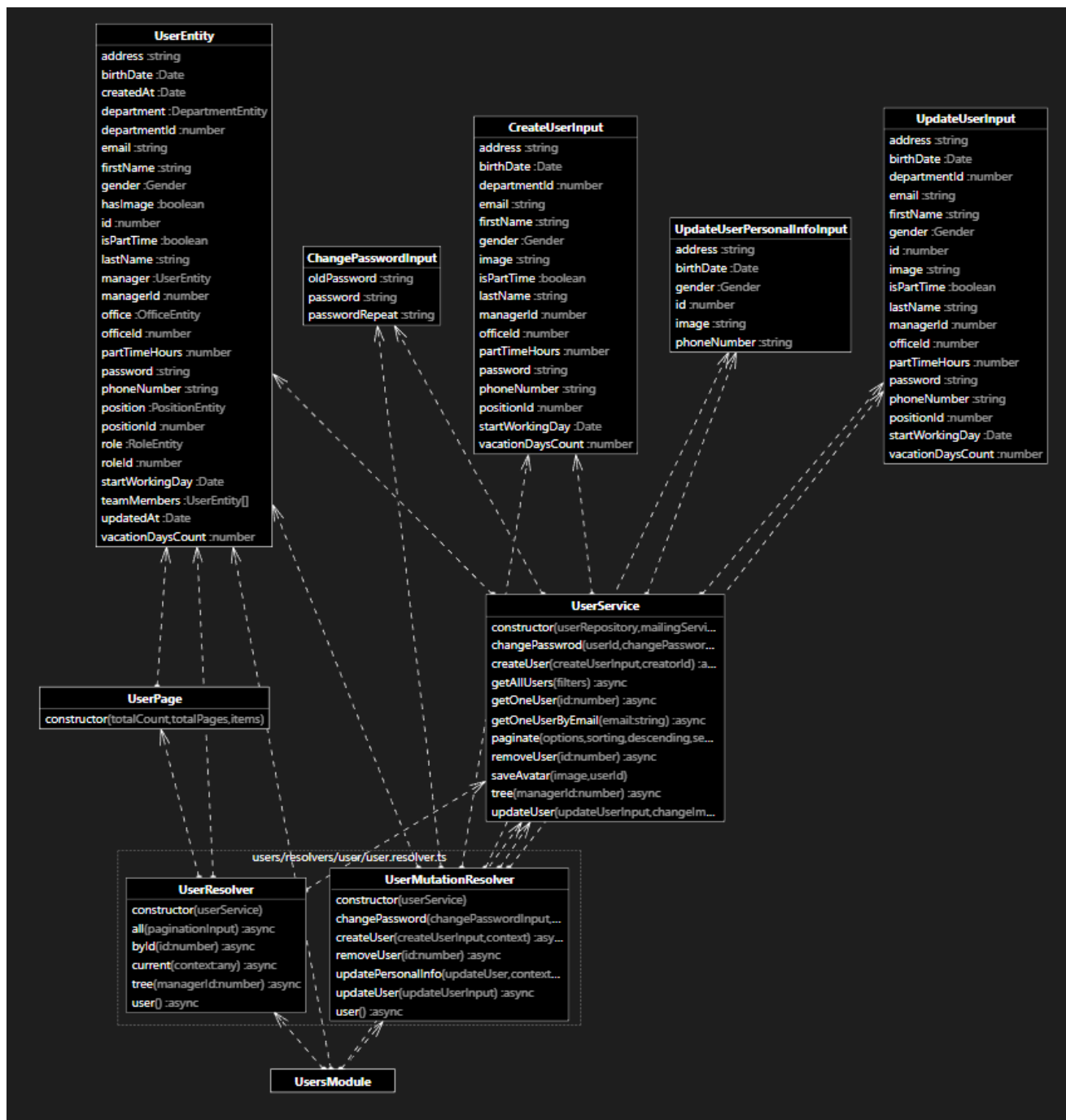


Рисунок 2.4 - Діаграма класів User модуля

2.2 Розробка бази даних системи

Для збереження файлів та інформації у додатку використовуються база даних PostgreSQL.

База даних системи складається з 8 таблиць:

1. users – таблиця, яка зберігає дані про користувачів;
2. vacation – таблиця, яка зберігає дані про відпустки;

3. roles – таблиця, яка зберігає дані про ролі;
4. posts – таблиця, яка зберігає дані про пости;
5. department – таблиця, яка зберігає дані про відділи;
6. holidays – таблиця, яка зберігає дані про свята;
7. offices – таблиця, яка зберігає дані про офіси;
8. positions – таблиця, яка зберігає дані про посади;
9. meetings – таблиця, що зберігає дані про зустрічі;
10. meetings-approvals – таблиця, що зберігає дані про підтвердження користувачами запрошень на зустрічі.

На рисунку 2.5 наведено діаграму бази даних:

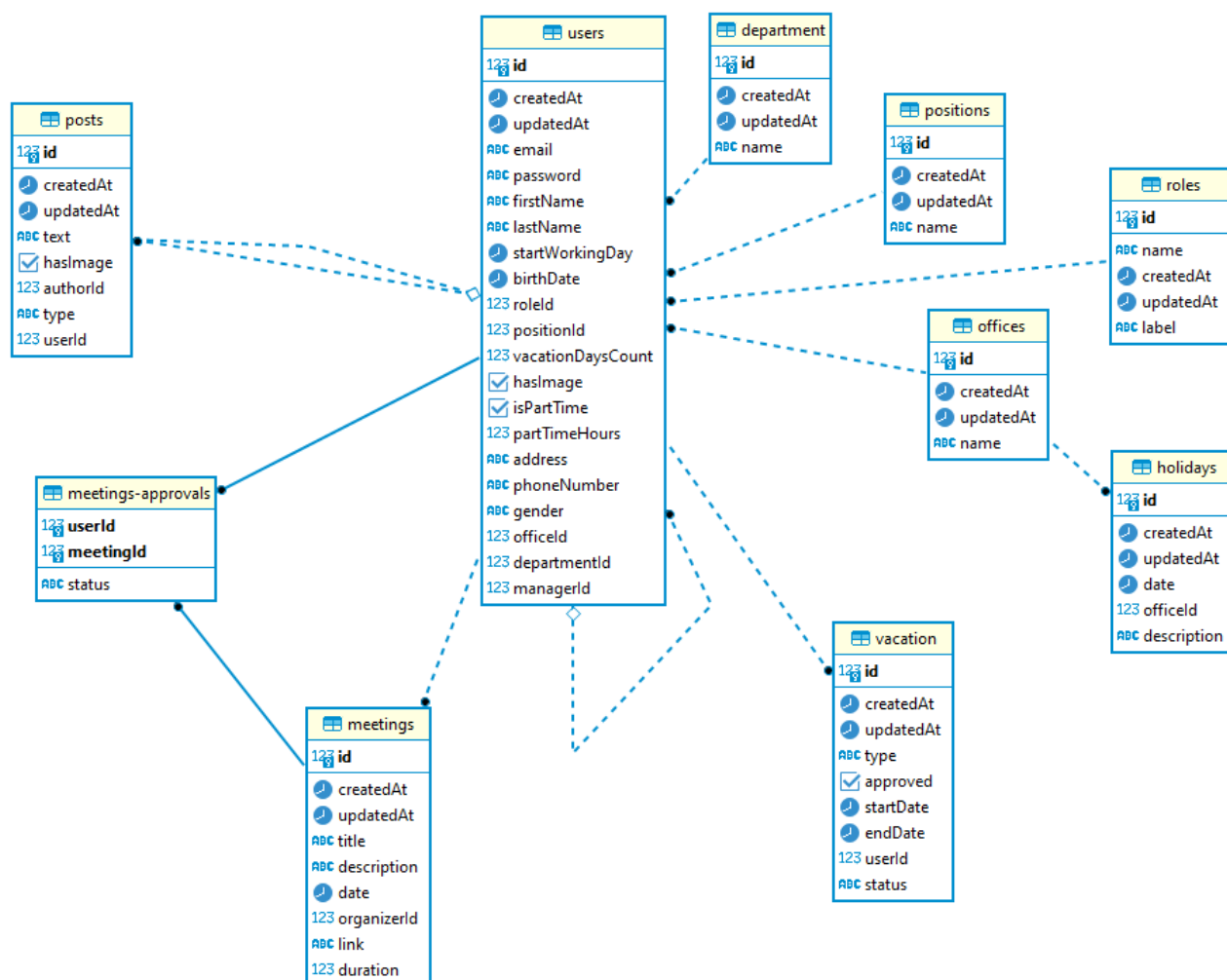


Рисунок 2.5. Діаграма бази даних.

Нижче наведений опис кожної з наведених таблиць бази даних (див. табл. 2.8 – 2.15)

Таблиця 2.8

Опис таблиці users

users		
Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор користувача
createdAt	timestamp without time zone	Дата і час створення запису користувача
updatedAt	timestamp without time zone	Дата і час останнього оновлення запису користувача
email	character varying	Email користувача
password	character varying	Пароль користувача
firstName	character varying	Ім'я користувача
lastName	character varying	Прізвище користувача
startWorkingDay	timestamp without time zone	Дата початку робочого дня користувача
birthDate	timestamp without time zone	Дата народження користувача
roleId	integer	Ідентифікатор ролі користувача
positionId	integer	Ідентифікатор посади користувача
vacationDaysCount	double precision	Кількість днів відпустки користувача
hasImage	boolean	Прапорець, що вказує, чи є у користувача зображення
isPartTime	boolean	Прапорець, що вказує, чи працює користувач на неповний робочий день
partTimeHours	integer	Кількість годин, які користувач працює на неповний робочий день
address	character varying	Адреса користувача
phoneNumber	character varying	Номер телефону користувача
gender	character varying	Стать користувача

officeId	integer	Ідентифікатор офісу, до якого належить користувач
departmentId	integer	Ідентифікатор відділу, до якого належить користувач
managerId	integer	Ідентифікатор керівника користувача

Таблиця 2.9

Опис таблиці roles

roles		
Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор ролі (обов'язкове поле)
name	character varying	Назва ролі (обов'язкове поле)
createdAt	timestamp without time zone	Дата та час створення ролі (значення за замовчуванням: поточний час)
updatedAt	timestamp without time zone	Дата та час останнього оновлення ролі (значення за замовчуванням: поточний час)

Таблиця 2.10

Опис таблиці vacation

vacation		
Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор відпустки (обов'язкове поле)
createdAt	timestamp without time zone	Дата та час створення запису відпустки (значення за замовчуванням: поточний час)
updatedAt	timestamp without time zone	Дата та час останнього оновлення запису відпустки (значення за замовчуванням: поточний час)
type	vacation_type_enum	Тип відпустки (обов'язкове поле)

approved	boolean	Позначка про затвердження відпустки
startDate	timestamp without time zone	Дата та час початку відпустки (обов'язкове поле)
endDate	timestamp without time zone	Дата та час закінчення відпустки (обов'язкове поле)
userId	integer	Унікальний ідентифікатор користувача, який подав заявку на відпустку (обов'язкове поле)
status	vacation_status_enum	Статус відпустки (значення за замовчуванням: 'OnReview')

Таблиця 2.11

Опис таблиці posts

posts		
Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор поста (обов'язкове поле)
createdAt	timestamp without time zone	Дата та час створення запису поста (значення за замовчуванням: поточний час)
updatedAt	timestamp without time zone	Дата та час останнього оновлення запису поста (значення за замовчуванням: поточний час)
text	character varying COLLATE pg_catalog."default"	Текст поста (обов'язкове поле)
hasImage	boolean	Прапорець, що вказує, чи містить пост зображення
authorId	integer	Унікальний ідентифікатор автора поста
type	posts_type_enum	Тип поста (обов'язкове поле)
userId	integer	Унікальний ідентифікатор користувача, пов'язаного з постом

Таблиця 2.12

Опис таблиці department

department		
Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор відділу (обов'язкове поле)
createdAt	timestamp without time zone	Дата та час створення запису відділу (значення за замовчуванням: поточний час)
updatedAt	timestamp without time zone	Дата та час останнього оновлення запису відділу (значення за замовчуванням: поточний час)
name	character varying COLLATE pg_catalog."default"	Назва відділу (обов'язкове поле)

Таблиця 2.13

Опис таблиці positions

positions		
Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор посади (обов'язкове поле)
createdAt	timestamp without time zone	Дата та час створення запису посади (значення за замовчуванням: поточний час)
updatedAt	timestamp without time zone	Дата та час останнього оновлення запису посади (значення за замовчуванням: поточний час)
name	character varying COLLATE pg_catalog."default"	Назва посади (обов'язкове поле)

Таблиця 2.14

Опис таблиці offices

offices		
Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор офісу (обов'язкове поле)
createdAt	timestamp without time zone	Дата та час створення запису офісу (значення за замовчуванням: поточний час)
updatedAt	timestamp without time zone	Дата та час останнього оновлення запису офісу (значення за замовчуванням: поточний час)
name	character varying COLLATE pg_catalog."default"	Назва офісу (обов'язкове поле)

Таблиця 2.15

Опис таблиці holidays

holidays		
Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор свята (обов'язкове поле)
createdAt	timestamp without time zone	Дата та час створення запису про свято (значення за замовчуванням: поточний час)
updatedAt	timestamp without time zone	Дата та час останнього оновлення запису про свято (значення за замовчуванням: поточний час)
date	timestamp without time zone	Дата свята (обов'язкове поле)
officeId	integer	Унікальний ідентифікатор офісу, пов'язаного зі святом (обов'язкове поле)

description	character varying COLLATE pg_catalog."default"	Опис свята (обов'язкове поле)
-------------	--	-------------------------------

2.3 Проектування та реалізація алгоритмів роботи додатку

Додаток простий у використанні, тому має в більшості основні функції, необхідні для реалізації першочергової мети створення сервісу. При використанні додатка користувач має можливість взаємодіяти з різними модулями програми і виконувати різні дії. Ось їх список:

1. Взаємодія з головною сторінкою: перегляд постів, інформації про доступність людей, нових робітників та найближчі дні народження.
2. Взаємодія зі списком робітників.
3. Взаємодія зі сторінкою структури компанії.
4. Взаємодія зі сторінкою відпусток: створення запита на відпустку, перегляд історії відпусток.
5. Взаємодія зі сторінкою профіля (своєю або чужою).

Помічник адміністратора додатково має наступні можливості:

1. Взаємодія з сторінкою відпусток: підтвердження або відхилення.
2. Взаємодія з головною сторінкою: створення постів.

Адміністратор має наступні можливості:

1. Взаємодія з сторінкою профіля: створення, редагування, видалення користувачів.
2. Взаємодія з налаштуваннями: керування офісами, відділами, посадами, святами.

Враховуючи всю цю інформацію, було створено загальну діаграму активності додатку:

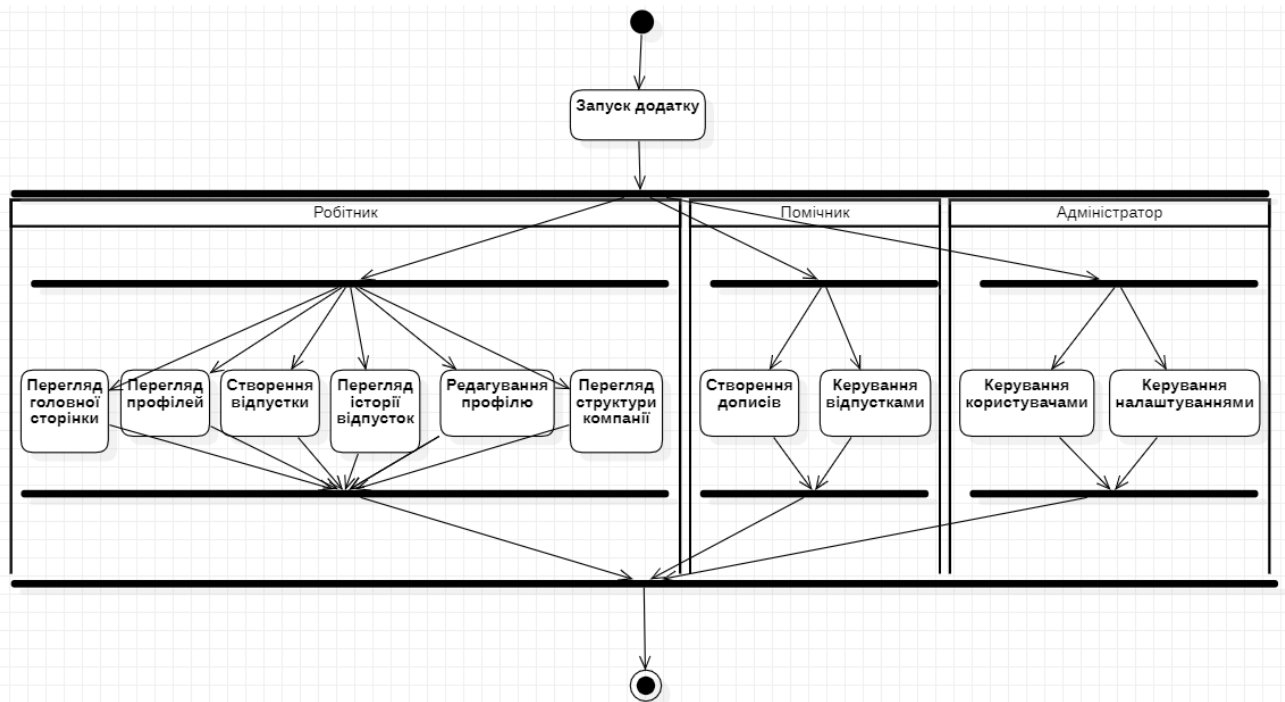


Рисунок 2. 5. Загальна діаграма активностей.

Один з основних модулів додатку – модуль відпусток. І один з найскладніших процесів, що вимагає втручання декількох ролей – це процес створення відпустки. На рисунку 2.6 – 2.7 наведено діаграму активностей і послідовності для цього процесу відповідно:

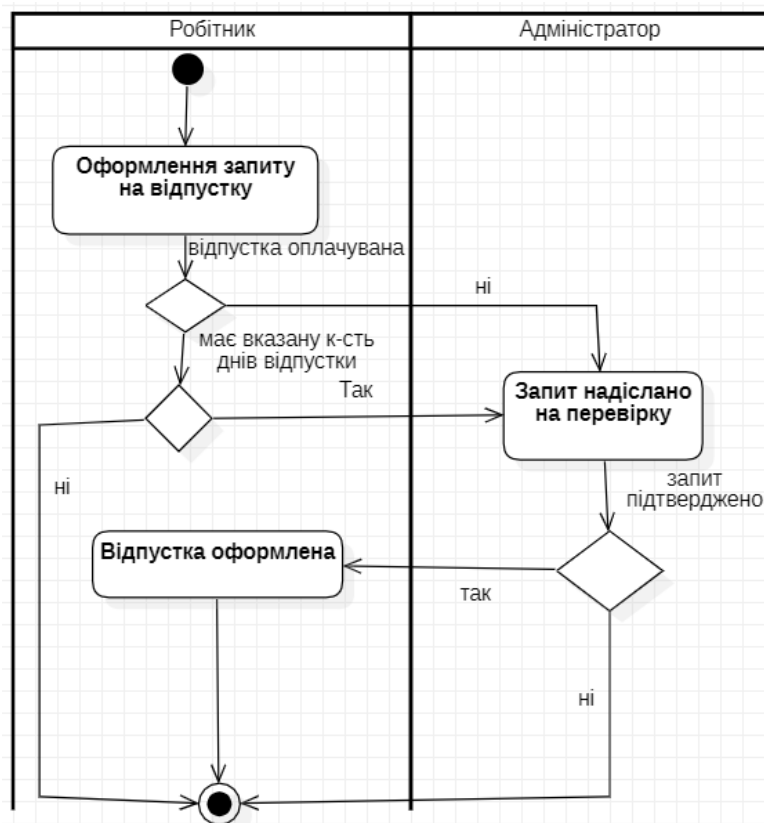


Рисунок 2.6. Діаграма активності процесу створення відпустки

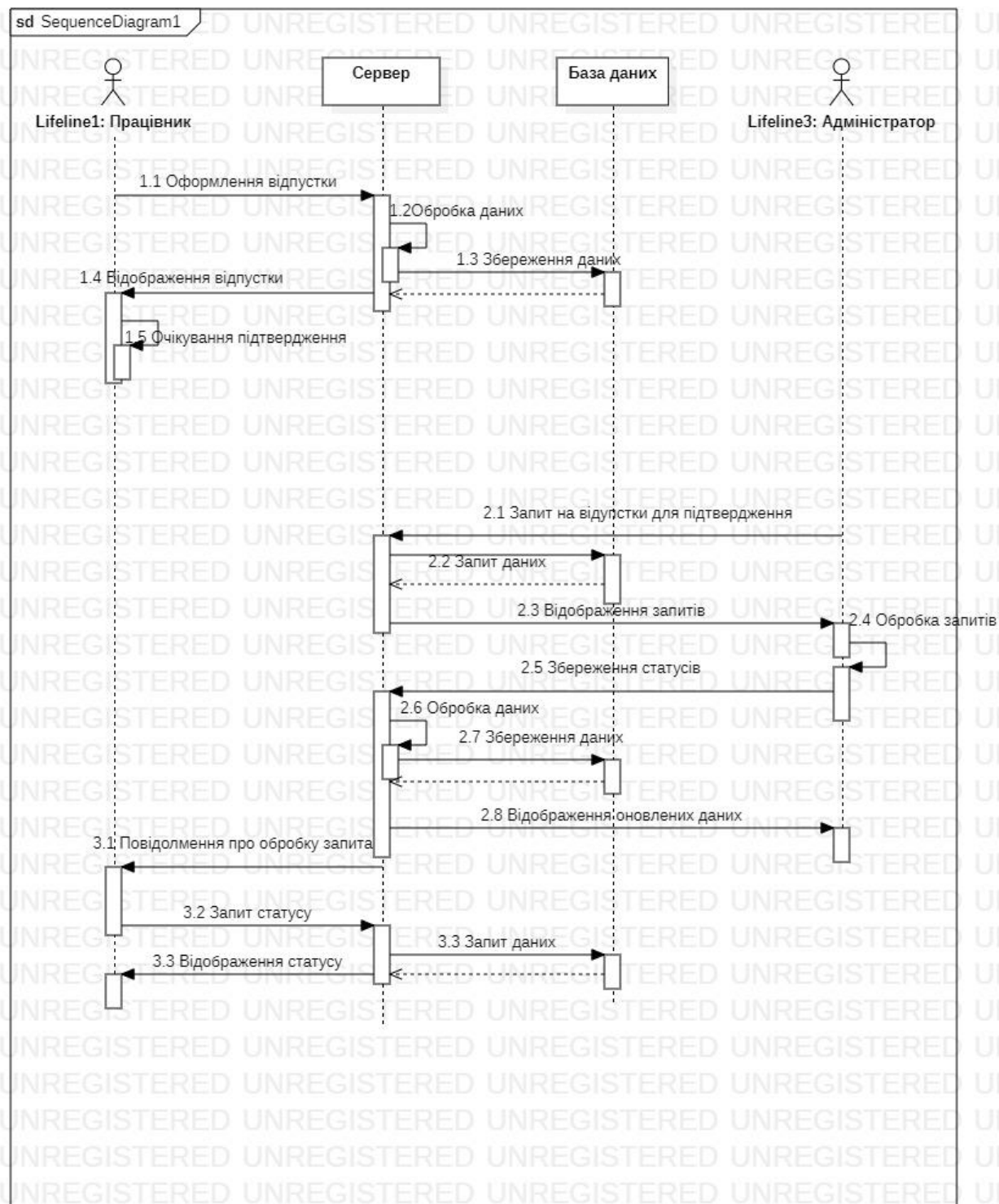


Рисунок 2.7. Діаграма послідовності для процесу оформлення відпустки.

Не менш важливим є модуль користувачів. Користувач може по різному взаємодіяти зі своїм і з чужим профілем. На рис 2.8 наведено діаграму активності для цього випадку:

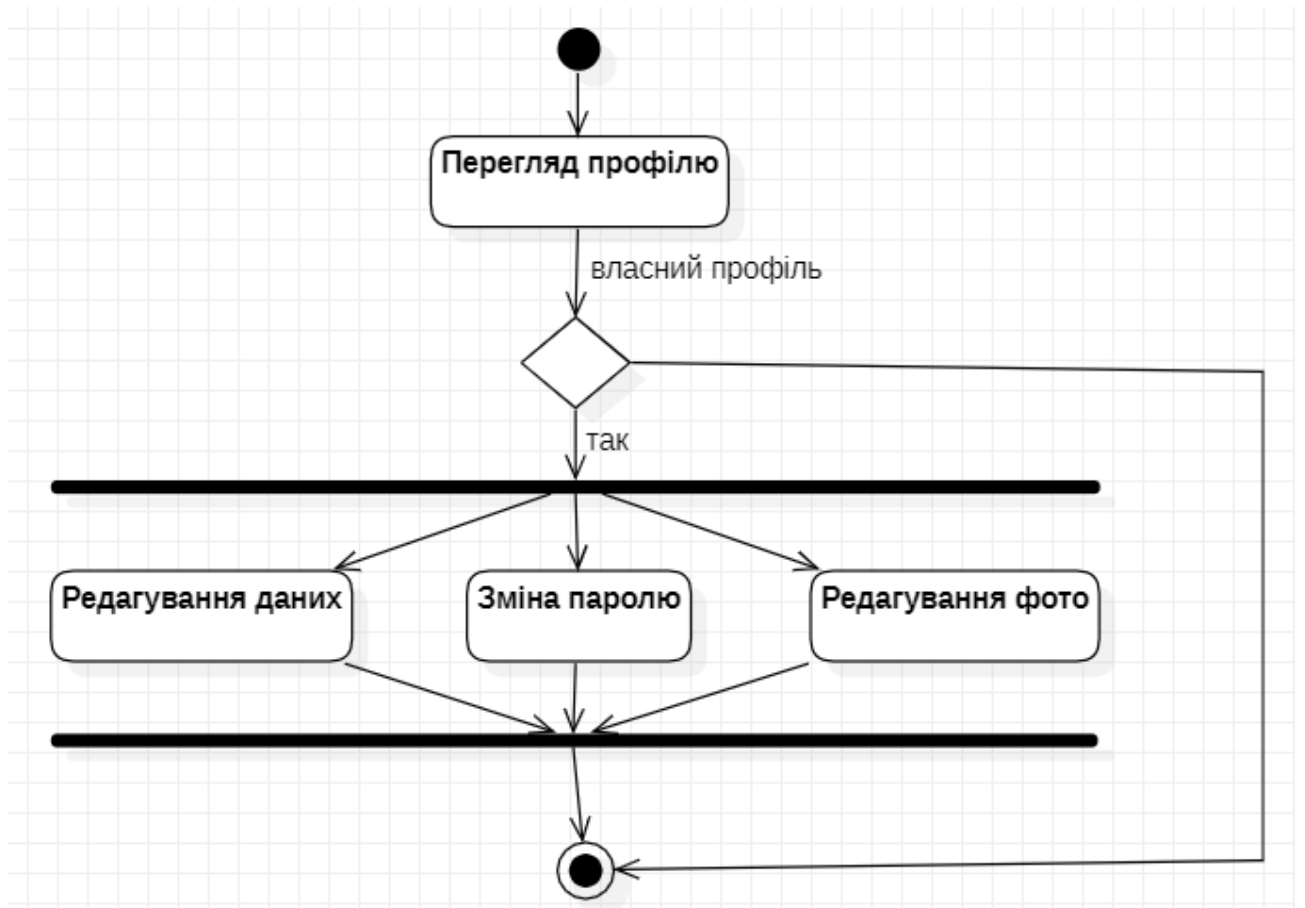


Рисунок 2.8. Діаграма активності процесу перегляду профілю

2.4 Реалізація функціоналу додатку

В наступному фрагменті коду реалізовано функціонал створення нового користувача:

```

async createUser(createUserInput: CreateUserInput, creatorId: number):
Promise<UserEntity> {
    if (!createUserInput.password)
        createUserInput.password = generator.generate({ length: 16, numbers:
true });
    const password = await bcrypt.hash(createUserInput.password, 10);

    const { image, ...user } = createUserInput;
    const savedUser = await this.userRepository.save(
        {
            ...user,
            password,
            roleId: 3,
            hasImage: !!image,
        }
    )
    if (image) {

```

```

        this.saveAvatar(image, savedUser.id);
    }

    this.mailingService.sendNewAccountMail(user.email, user.firstName,
user.lastName, createUserInput.password);
    if (createUserInput.startWorkingDay.toString() === (new
Date()).toString()) {
        this.postsService.create(
            {
                text: `Вітаємо нового члена нашої команди: ${user.lastName}
${user.firstName}`,
                image,
            },
            creatorId,
            PostTypes.NewHires
        );
    }

    return savedUser;
}

```

Дана функція генерує для користувача пароль, якщо він був не вказаний, дали пароль хешується. Користувачу автоматично присвоюється роль звичайного користувача, якщо вона не була вказана. Після цього, якщо було завантажено зображення, воно зберігається в окремій функції saveAvatar. Наступний крок – відправка емейла користувачу про створення його профілю з паролем. У випадку, якщо дата створення користувача співпадає з його датою прийняття на роботу, створюється пост про прийняття на роботу нового колегу.

В системи наявні задачі, що повторюються з певною періодичністю. Одна з них – це нарахування відпустки:

```

@Cron(CronExpression.EVERY_DAY_AT_MIDNIGHT)
async calculateVacations() {
    const date = new Date();
    var day = date.toLocaleString("default", { day: "2-digit" });

    const users = await this.userService.getAllUsers([
        { field: "startWorkingDate", value: day }
    ]);
    users.forEach(user => {
        const updateInput = new UpdateUserInput();
        updateInput.id = user.id;
        updateInput.vacationDaysCount = user.vacationDaysCount + 2;
        this.userService.updateUser(updateInput, false)
        this.logger.debug(user.id);
    })
}

```

В даній функції спочатку знаходиться поточне число місяця і потім знаходяться користувачі, що почали працювати в цей день місяця. Після цього для кожного з них оновлюється кількість днів відпустки. Дана функція спрацьовує кожен день в 00:00.

Друга функція, що виконується періодично – це автоматичне створення дописів про день народження:

```
@Cron(CronExpression.EVERY_DAY_AT_10AM)
async createBirthdaysPosts() {
    const date = new Date();
    var day = date.toLocaleString("default", { day: "2-digit" });
    var month = date.toLocaleString("default", { month: "2-digit" });

    const users = await this.userService.getAllUsers([
        { field: "birthday", value: `${month}.${day}` }
    ]);
    users.forEach(user => {
        this.postService.create(
            {
                text: `${user.lastName} ${user.firstName}, вітаємо з днем народження!`,
                userId: user.id
            },
            undefined,
            PostTypes.Birthday
        );
    });
}
```

В даній функції спочатку вираховується поточні місяць і число. Після цього знаходяться користувачі, у яких день народження в цей день. Для кожного користувача створюється допис про день народження. Дана функція спрацьовує кожен день в 10:00.

Логіка зміни статусу відпустки(відхилення або підтвердження) реалізовано наступним чином:

```
async update(updateInput: UpdateVacationInput): Promise<VacationEntity> {
    await this.repository.update({ id: updateInput.id }, { ...updateInput })
    const vacation = await this.findOne(updateInput.id)

    if (vacation.type === VacationTypes.Paid && vacation.status ===
    VacationStatus.Approved) {
        const user = await this.userService.findOne(vacation.userId);

        const updatedUser = new UpdateUserInput();
        updatedUser.id = vacation.userId;

        const diffTime = Math.abs(+vacation.endDate - +vacation.startDate);
```

```

const diffDays = Math.ceil(diffTime / (1000 * 60 * 60 * 24)) + 1;

updatedUser.vacationDaysCount = user.vacationDaysCount - diffDays;

await this.userService.updateUser(updatedUser, false);
}

return vacation;
}

```

Спершу оновлюється лише статус відпустки, але далі, якщо відпустка є оплачуваною і вона була підтверджена, то оновлюється кількість днів відпустки користувача. Спочатку вираховується її тривалість і ця кількість днів віднімається від кількості днів відпустки користувача.

Функція, що повертає список користувачів, приймає багато параметрів і здійснює пошук за багатьма критеріями:

```

async paginate(options: IPaginationOptions, sorting: string, descending: boolean,
search: string, filters: EntityFilter[]): Promise<Pagination<UserEntity>> {
    const query = this.userRepository
        .createQueryBuilder('users')
        .orderBy(`users.${sorting}`, descending ? 'DESC' : 'ASC')
        .leftJoinAndSelect("users.position", "position")
        .leftJoinAndSelect("users.office", "office")
        .leftJoinAndSelect("users.department", "department")
        .where(
            new Brackets((qb) => {
                qb
                    .where('users.firstName || \' \' || users.lastName ILIKE
:search')
                    .orWhere('users.lastName || \' \' || users.firstName ILIKE
:search')
                    .orWhere('users.email ILIKE :search')
            }),
            { search: `%${!!search ? search : ''}%` }
        );

    if (filters) {
        filters.forEach((f, i) => {
            if (!f.value)
                return;

            const value = {}
            value[`val${i}`] = f.value

            if (f.field === "birthday") {
                query.andWhere(`to_char("birthDate", 'MM.dd') >= :val${i}`,
value);
                query.orderBy(`to_char("birthDate", 'MM.dd')`)
            }
        });
    }
}

```

					ІПЗ.КР.Б-121-23-ІЗ	44

```

        } else {
            query.andWhere(`users.${f.field} = :val${i}`, value)
        }
    })
}

const users = await paginate<UserEntity>(query, options);
return users
}

```

Спершу застосовується сортування і приєднуються потрібні таблиці: positions, offices, departments. Далі додається перший where вираз для пошук. Пошук здійснюється за комбінацією полів lastName і firstName, а також навпаки. Також пошук працює і з полем email. Далі перебираються всі фільтри. Для звичайного фільтра додається звичайний where вираз з умовою «=». Але для певних полів є виключення. В даному випадку це поле birthday. Якщо це поле наявне у фільтрах, то будуть шукатися користувачі, у яких день народження пізніше за вказаний день. Відповідно буде використовуватись умова «>=». Додатково буде здійснено сортування за цим полем.

В клієнтській частині один із головних компонентів, що часто повторюється – список з можливими сортуванням, фільтрацією та пошуком. Так виглядає код цього компоненту:

```

const PaginationList: FC<Props> = ({ query, extractData, renderItem,
sortingOptions, filters, defaultDesc, filtersComponent, showSearch }) => {
    let [searchParams, setSearchParams] = useSearchParams();
    const page = searchParams.get('page') || "1";
    const limit = searchParams.get('limit') || "20";
    const sorting = searchParams.get('sorting') || (sortingOptions &&
sortingOptions[0].key);
    const descending = searchParams.get('desc') ? searchParams.get('desc') === "true"
: defaultDesc;
    const searchString = searchParams.get('q') || '';

    const [expanded, setExpanded] = useState(false);

    useEffect(() => {
        setExpanded(!!filters?.length || !!searchString)
    }, [filters?.length, searchString])

    const handleExpandClick = () => {
        setExpanded(!expanded);
    };

    const handlePageChange = (event: React.ChangeEvent<unknown>, value: number) => {

```

```

    searchParams.set('page', value.toString());
    setSearchParams(searchParams);
  };

  const handleLimitChange = (event: SelectChangeEvent) => {
    searchParams.set('limit', event.target.value);
    searchParams.set('page', "1");
    setSearchParams(searchParams);
  };

  const handleSortingChange = (event: SelectChangeEvent) => {
    searchParams.set('sorting', event.target.value);
    searchParams.set('page', "1");
    setSearchParams(searchParams);
  };

  const handleDescendingChange = () => {
    searchParams.set('desc', `${!descending}`);
    searchParams.set('page', "1");
    setSearchParams(searchParams);
  };

  const handleSearchChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    searchParams.set('q', event.target.value.toString());
    setSearchParams(searchParams);
  };

  return (
    <>
      {(filters || showSearch) &&
        <ExpandMore
          expand={expanded}
          text="Фільтри"
          onClick={handleExpandClick}
          aria-expanded={expanded}
          aria-label="Показати фільтри"
        >
          <ExpandMoreIcon />
        </ExpandMore>
      }
      <Collapse in={expanded} timeout="auto" unmountOnExit>
        {showSearch &&
          <TextField label="Пошук" variant="outlined" margin="normal" fullWidth
            onChange={handleSearchChange} value={searchString} />
        }
        {filtersComponent}
      </Collapse>
      <RawQueryData
        query={query}
        variables={{
          paginationInput: {

```

```

        page: +page,
        limit: +limit,
        sorting,
        descending,
        filters: filters ? filters : undefined,
        search: searchString,
      }
    }}
  >
  {(queryData: any) => {
    const data = extractData(queryData);

    if (data.totalCount === 0)
      return <Typography variant="h6" component='p'>Нічого немає</Typography>

    return (
      <>
        ...
        {data.items.map((item: any) => renderItem(item))}
        <Stack spacing={2} marginBottom={3}>
          <Pagination count={data.totalPages} page={+page}
onChange={handlePageChange} />
        </Stack>
      </>
    )
  }}
</RawQueryData>
</>
);
}

```

Компонент працює з url строкою через searchParams. Тому при будь-якій зміні якогось параметру(сторінка, сортування, пошук, фільтрація), це відразу записується в строку url. І при перезавантаженні сторінки ми отримуємо ті ж самі результати. Компонент приймає можливі поля для сортування. Змінна showSearch відповідає за відображення поля пошуку, showFilters – фільтрів, а filtersComponent – це компонент, який містить фільтри. Також в компонент обов'язко маж бути передано параметри query, extractData і renderItem. Query – це назва GraphQL query, що використовується для отримання списку елементів. extractData – функція, що вилучає елементи з відповіді GraphQL. renderItem – функція, яка отримує елемент і відображає його.

Для використання будь-якої query було створено компонент-обгортку, що буде відображати індикатор завантаження під час завантаження і помилка при виникненні помилки:

					ІПЗ.КР.Б-121-23-ПЗ	
						47

```

export const RawQueryData: FC<RawQueryDataProps> = ({
  query,
  variables,
  children,
}) => {
  const { loading, error, data } = useQuery(query, { variables });

  if (loading || !data) {
    return <Loader />;
  }
  if (error) {
    return (
      <Error error={error.message} />
    );
  }

  if (!children) {
    return <p>'Error! You forgot to pass children'</p>;
  }

  if (typeof children === 'function') {
    return children(data);
  }

  return React.cloneElement(children, data);
};

```

При успішному виконанні буде викликано функцію, children, куди буде передано відповідь GraphQL.

Так виглядає функціонал видалення користувача:

```

export default function UserDeliting({ id }: Props) {
  const navigate = useNavigate();
  const [open, setOpen] = useState(false);
  const [removeUser, { loading }] = useMutation(REMOVE_USER);
  const onRemoveClick = () => {
    setOpen(false);
    removeUser({
      variables: {
        id,
      },
      onCompleted() {
        navigate("/profile/list");
      },
    })
  }

  return (
    <>
      <Backdrop
        sx={{ color: '#fff', zIndex: (theme) => theme.zIndex.drawer + 1 }}

```

```

        open={loading}
      >
      <CircularProgress color="inherit" />
    </Backdrop>
    <Button variant="contained" fullWidth color='error' onClick={() =>
setOpen(true)} sx={{ mt: 2 }}>
      Видалити користувача
    </Button>
    <RemoveDialog
      name='Видалення користувача'
      description='Ви впевнені, що хочете видалити цього користувача?'
      open={open}
      onClose={() => setOpen(false)}
      onRemove={onRemoveClick}
    />
  </>
)
}

```

При натисканні на кнопку «Видалити користувача» буде відкрито модальне вікно для підтвердження. Якщо дія підтвердилась, буде відправлена відповідна GraphQL мутація. Під час її виконання буде відображатись індикатор завантаження і всі елементи UI будуть недоступні для взаємодії. Після успішного видалення користувача, буде перенаправлено на сторінку всіх користувачів.

Для зручного використання функції фільтрації було створено кастомний хук useFilters:

```

export default function useFilters() {
  let [searchParams, setSearchParams] = useSearchParams();
  const filterString = searchParams.get('filters');
  const [filters, setFilters] = useState<EntityFilter[]>([]);

  const handleFilterChange = (event: any) => {
    const newFilter = { field: event.target.name, value:
event.target.value.toString() };

    let newFilters: EntityFilter[] = [];
    if (newFilter.value) {
      newFilters.push(newFilter);
    }
    filters.forEach(f => {
      if (f.field !== newFilter.field) {
        newFilters.push(f);
      }
    })

    searchParams.set('filters', newFilters.map(f =>
`${f.field}:${f.value}`).join('+'));
  }
}

```

```

    setSearchParams(searchParams);
  };

  useEffect(() => {
    if (filterString) {
      let filters: EntityFilter[] = [];

      const filterArray = filterString.split('+');
      filterArray.forEach(f => {
        const filterData = f.split(':');
        filters.push({ field: filterData[0], value: filterData[1] })
      })

      setFilters(filters);
    } else {
      setFilters([]);
    }
  }, [filterString])

  return { filters, handleFilterChange };
}

```

Даний хук слугує для того, щоб керувати поточними фільтрами та записувати або навпаки, вилучати їх, з url строки. Він використовує масив filters типу EntityFilter[], який синхронізується з url строкою. Даний хук повертає масив filters з поточними фільтрами і функцію handleFilterChange, яка обробляє зміну або додавання нового фільтру.

Висновки до другого розділу

В процесі проектування додатку були проаналізовані усі варіанти використання програмного продукту відповідно до вимог програмного забезпечення. На основі отриманого результату була спроектована структура сервісу. Проектування включало в себе моделювання структури проекту та взаємодії між їх складовими частинами.

Також було спроектовано базу даних для зберігання даних, що використовуватимете додаток.

Було описано алгоритм роботи з додатком та описані послідовні пункти варіантів його використання.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ

3.1 Порядок встановлення та налаштування параметрів сервісу

Встановлення та налаштування сервісу вимагає розгортання всіх його компонентів та налаштування як кожного елементу окремо, так і загальної системи в цілому.

Для опису архітектури сервісу та розгортання було створено діаграму розгортання (Рисунок 3.1).

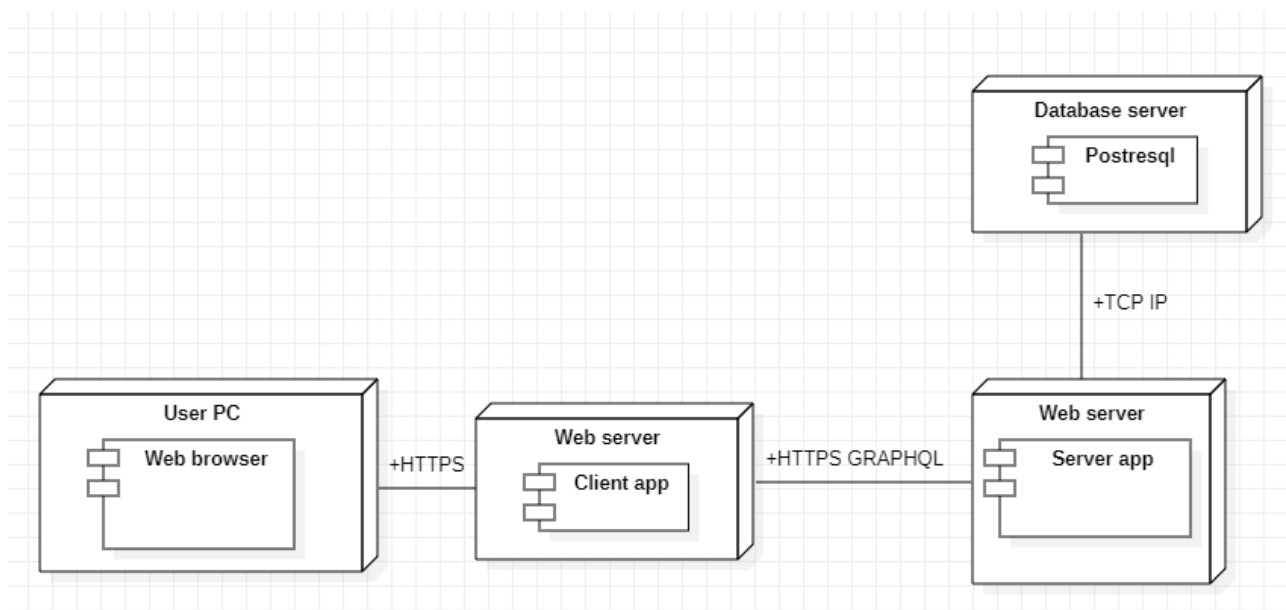


Рисунок 3.1 - Діаграма розгортання додатку «Онлайн робочий кабінет»

Система складається з 4 вузлів: веб-сервер з серверним додатком, веб-сервер з клієнтським додатком, сервер бази даних, браузер користувача. Браузер користувача обмінюються даними з клієнтським за допомогою протоколу HTTPS. В свою чергу клієнтський додаток обмінюється даними з серверним додатком теж за допомогою протоколу HTTPS з використанням мови запитів GraphQL.

Для коректної взаємодії клієнтського і серверного додатку мають бути правильно налаштовані cors (Cross-origin resource sharing). В клієнтському додатку при налаштуванні apollo клієнта необхідно вказати credential – «include»:

```
const httpLink = new HttpLink({ uri: 'http://localhost:3030/GraphQL', credentials:
'include' });
```

В серверному додатку cors потрібно налаштувати двічі: для всього додатку і для GraphQL сервера:

```
app.enableCors({
  credentials: true,
  origin: true,
});
```

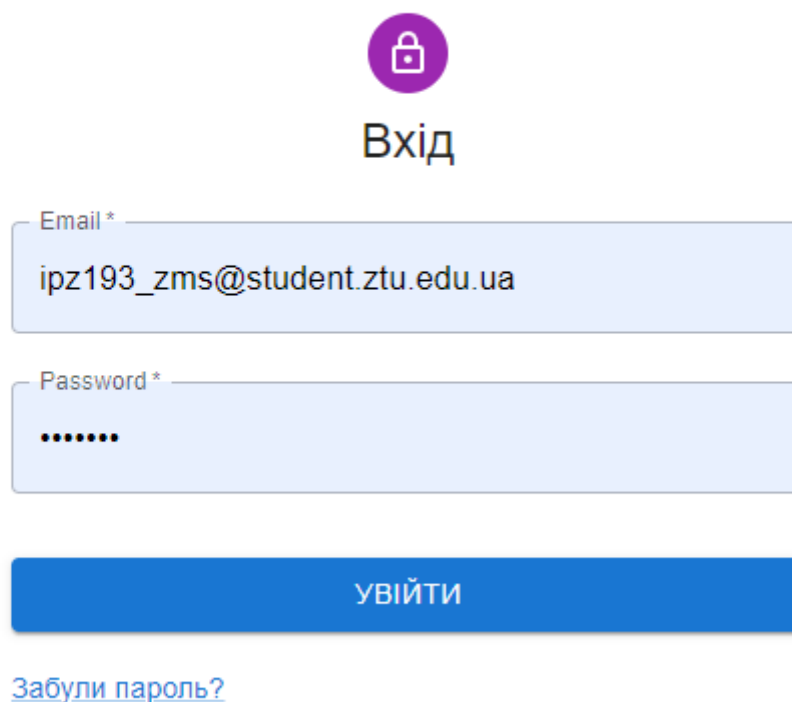
```
GraphQLModule.forRoot({
  ...
  cors: {
    credentials: true,
    origin: true,
  },
  ...
}),
```

На серверному додатку має бути правильно сконфігурований конфігураційний файл .env, що містить в собі дані для підключення до БД, дані для розсилки електронних листів і URL клієнтського додатку.

Після виконаних дій серверний додаток може бути запущений командою `npm run start: dev` у режимі розробки та командою `npm run start:prod` в режимі продакшена. Попередньо мають бути виконані команді `npm run build`. Для запуску клієнтського додатку в режимі розробки необхідно скористатися командою `npm start`. Для запуску в режимі продакшена потрібно спочатку зібрати проект командою `npm run build` і для отриманих файлів налаштувати статичний сервер.

3.2 Структура інтерфейсу додатку «Онлайн робочий кабінет»

Для того, щоб користуватися додатком, необхідно увійти в системи, використовуючи email і пароль:



Вхід

Email *

ipz193_zms@student.ztu.edu.ua

Password *

.....

УВІЙТИ

[Забули пароль?](#)

Рисунок 3.2. Форма входу.

Далі користувач потрапляє на головну сторінку, де бачить останні дописи, доступність робітників, нових робітників та найближчі дні народження:

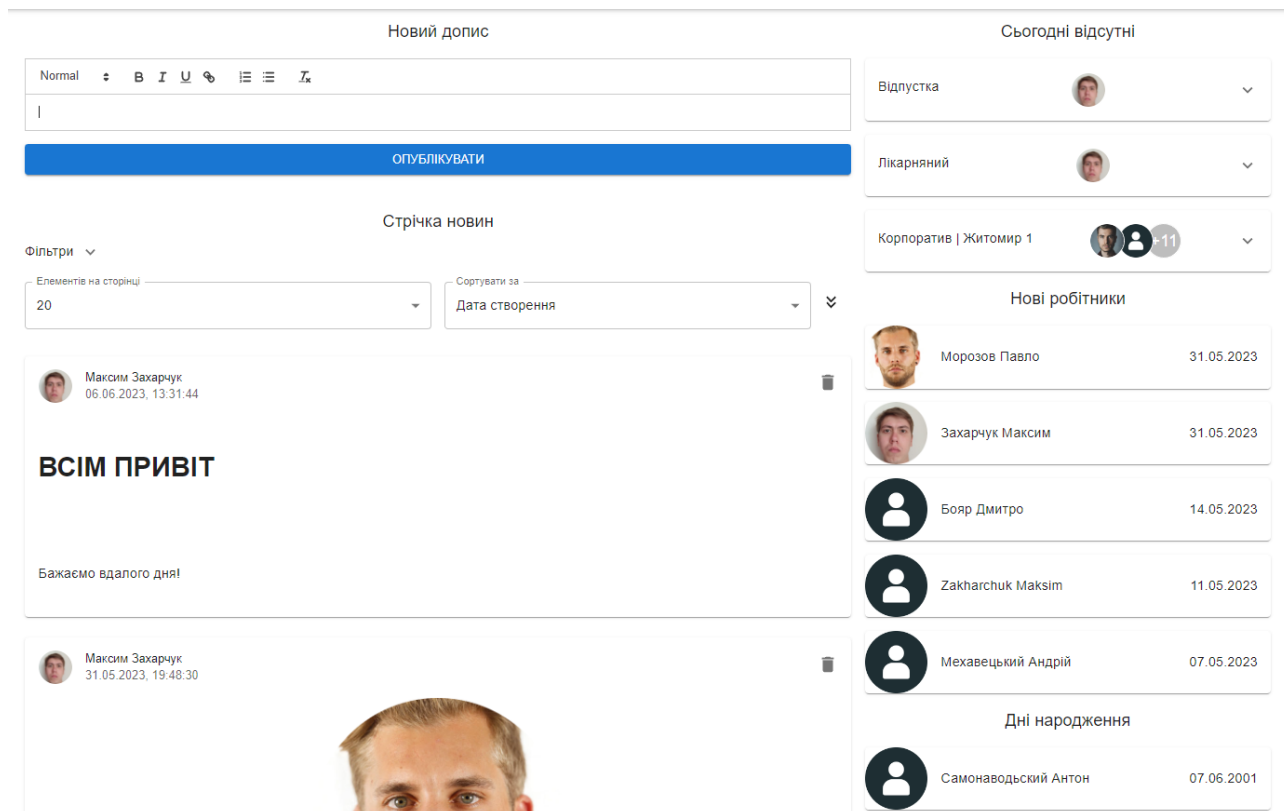


Рисунок 3.3. Головна сторінка.

Зліва є навігація, що дозволяє вибрати потрібну сторінку:

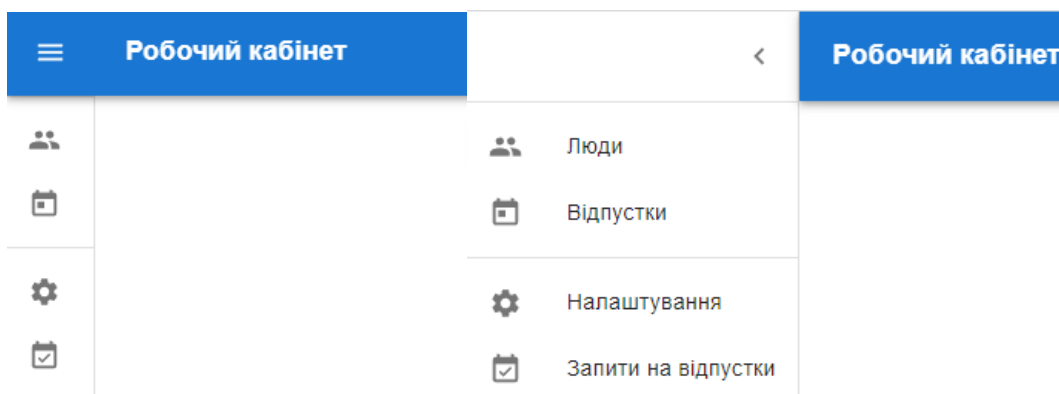


Рисунок 3.4. Навігація

Нижче наведено рисунок з виглядом сторінки «Люди». Оскільки вхід в систему було здійснено під користувачем з роллю адміністратора, доступна кнопка «Додати користувача»:

ДОДАТИ КОРИСТУВАЧА

Фільтри ^

Пошук

Офіс

Житомир 1

Відділ

Продукт

Посада

Стать

Елементів на сторінці

20

Сортувати за

Дата початку роботи



Панченко Лідія

Junior Software Developer

zaharfake55@gmail.com



Домбровський Ігор

HR

zahdasda@gmail.com



Козавецька Наталія

Junior Software Developer

zaharfake66@gmail.com

Рисунок 3.5. Список робітників

Для створення користувача використовується наступна форма:

Email *

ghotyup@mail.com

Ім'я *

Іван

Прізвище *

Морошов

Номер телефону

+380687405896

Офіс

Київ

Відділ

Продукт

Посада

Software Developer

Стать

☒ Чоловік
 ☐ Жінка

☐ Неповний робочий день

К-сть робочих годин *

4

Відповідальна особа

Захарчук Максим(ipz193_zms@student.ztu.edu.ua)

Рисунок 3.6. Форма для створення користувача

					ІПЗ.КР.Б-121-23-ПЗ	55

Дата народження *

01.02.1995

Дата початку роботи *

08.02.2018

Адреса

Дні відпустки

15



ЗБЕРЕГТИ

Рисунок 3.6. Форма створення користувача

Після цього на вказаний email буде відправлено лист-вітання зі згенерованим паролем:

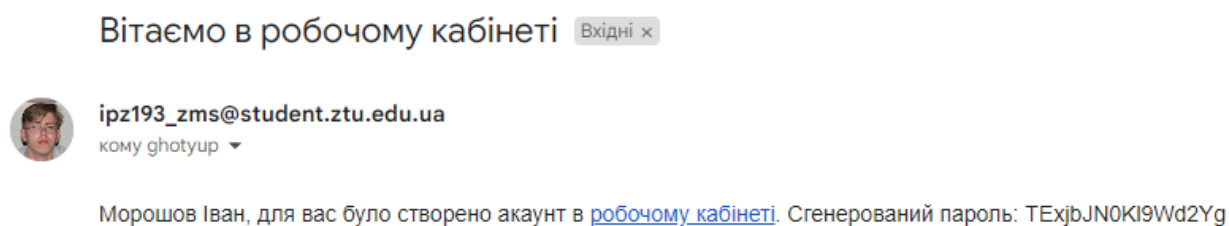


Рисунок 3.7. Лист-вітання

Так виглядає сторінки тільки що створеного користувача:



Морошов Іван

Київ | Продукт

Software Developer

Команда

Відповідальна особа

Захарчук Максим

Основна інформація

Email

ghotyup@mail.com

Номер телефону

+380687405896

Робочий день

Повний

Дата початку роботи

08.02.2018

Додаткова інформація

Дата народження

01.02.1995

Стать

Чоловік

ВИДАЛИТИ КОРИСТУВАЧА

Рисунок 3.8. Сторінка користувача.

Здійснимо вхід під щойно створеним користувачем і перейдемо на сторінку відпусток:

К-сть днів оплачуваної відпустки:

15

Запит на відпустку

Дата початку *

ДД.ММ.ГГГГ

Дата кінця *

ДД.ММ.ГГГГ

Тип

СТВОРИТИ ЗАПИТ

Останні відпустки

Нічого немає

Рисунок 3.9. Сторінка відпусток.

Створимо нову відпустку і побачимо, що вона відразу буде відображена в списку:

К-сть днів оплачуваної відпустки:

15

Запит на відпустку

Дата початку *

06.06.2023

Дата кінця *

07.06.2023

Тип

Оплачувана

СТВОРИТИ ЗАПИТ

Останні відпустки

2023-06-06 - 2023-06-07

На розгляді

Оплачувана

ДИВИТИСЯ ВСІ

Рисунок 3.10. Вигляд сторінки відпусток після створення відпустки.

Тепер знову увійдемо в систему як адміністратор і перейдемо на сторінку управління відпустками:

Відпустки

Фільтри ^

Статус

Тип

З

ДД.ММ.ГГГГ

По

ДД.ММ.ГГГГ

Робітник

Морошов Іван

Елементів на сторінці

20

Сортувати за

Дата створення

Іван Морошов

2023-06-06 - 2023-06-07

На розгляді

Оплачувана

✓

✗

< 1 >

Рисунок 3.11. Сторінка управління відпустками.

Тепер спробуємо підтвердити відпустку:

Іван Морошов

2023-06-06 - 2023-06-07

Підтверджена

Оплачувана

Рисунок 3.12. Підтверджена відпустка.

Зараз увійдемо під цим користувачем і знову зайдемо на сторінку відпусток. Побачимо, що кількість днів відпустки і історія змінились:

К-сть днів оплачуваної відпустки:

13

Запит на відпустку

Дата початку *

ДД.ММ.ГГГГ

Дата кінця *

ДД.ММ.ГГГГ

Тип

СТВОРИТИ ЗАПИТ

Останні відпустки

2023-06-06 - 2023-06-07

Підтверджена

Оплачувана

ДИВИТИСЯ ВСІ

Рисунок 3.13. Сторінка відпусток після підтвердження.

Після цього дана відпустка буде відображена на головній сторінці, оскільки припадає на поточну дату:

Сьогодні відсутні

Відпустка







Захарчук Максим

06.06.2023 - 06.06.2023



Морошов Іван

06.06.2023 - 07.06.2023

Лікарняний





Корпоратив | Житомир 1



+11



Рисунок 3.14. Список відсутніх робітників.

Для адміністратора доступне меню налаштувань. Тут можна керувати святами, посадами, офісами і відділами:

Налаштування

ЗАГАЛЬНІ

ПОСАДИ

ОФІСИ

ВІДДІЛИ

ВИХІДНІ

HR

Junior Software Developer

Software Developer

Senior Software Developer

Project Manager

QA Enginer

E-Commerce consultant

Нова посада *

















Рисунок 3.15. Сторінка налаштувань

При редагуванні або створенні користувача офіси, відділи і посади будуть доступні у відповідних полях для вибору:

Рисунок 3.16. Вибір відділу

На деяких сторінках при взаємодії користувача з інтерфейсом показуються повідомлення внизу сторінки, що зникають через певний час:

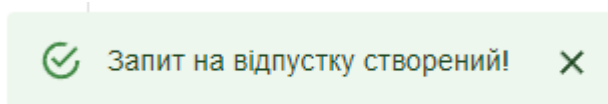


Рис 3. 17. Спливаюче повідомлення.

При спробі видалити користувача або пост з'являється спливаюче вікно, що вимагає підтвердження дії:

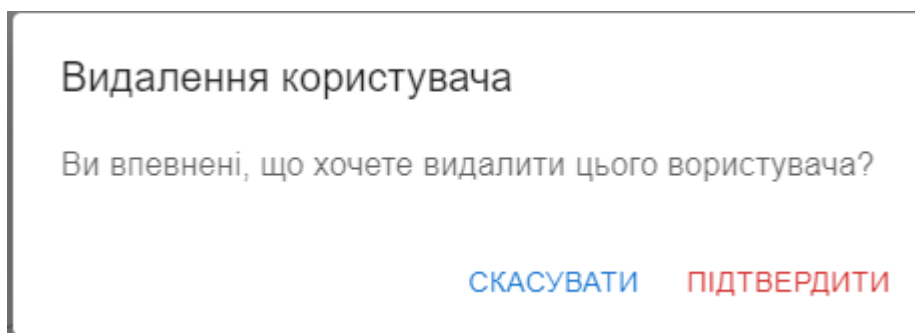


Рисунок 3.18. Спливаюче вікно для підтвердження видалення.

У системі автоматично створюються дописи про прийом нового робітника на роботу та про день народження:

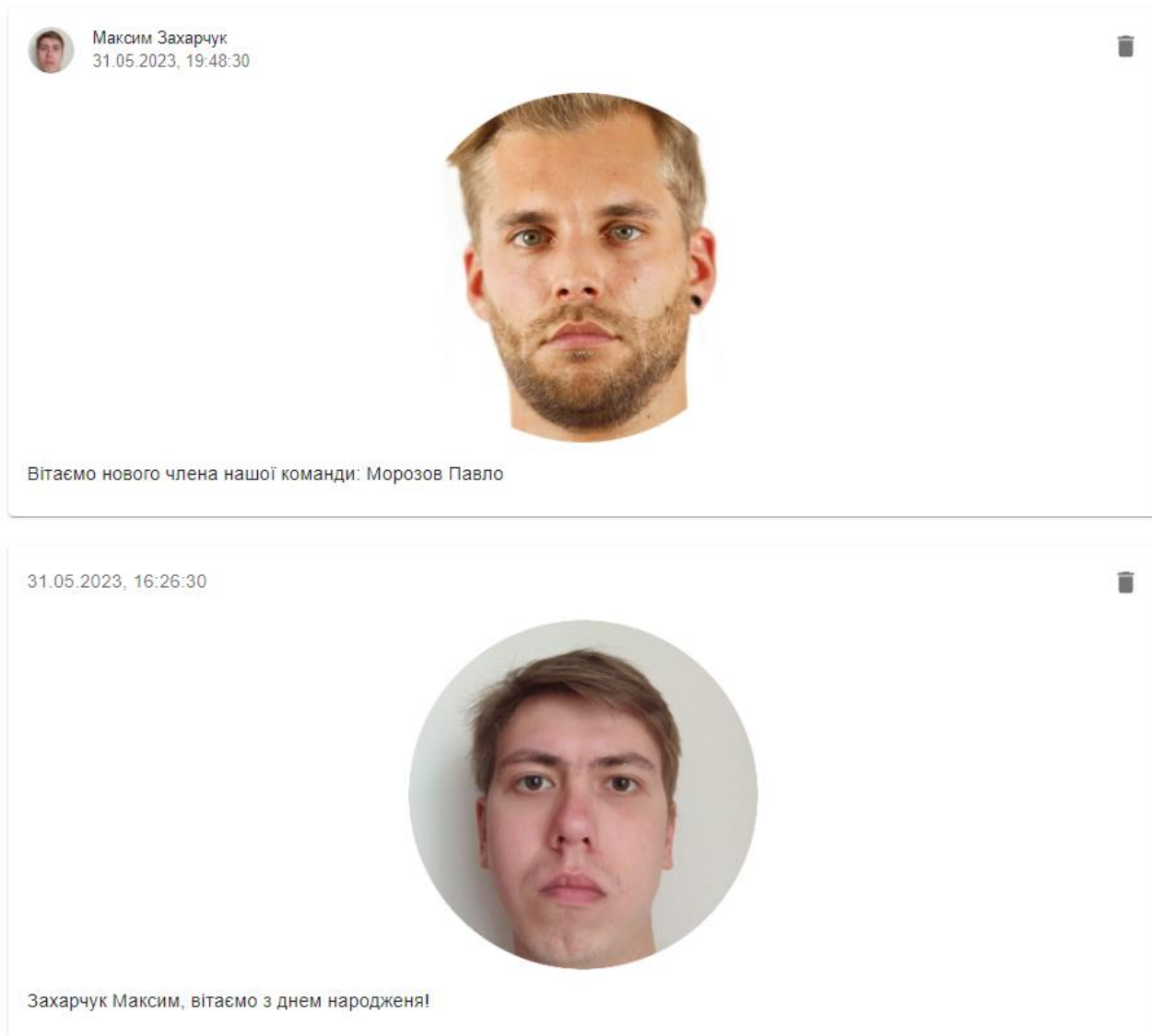


Рисунок 3.19. Автоматично створені дописи

Нижче наведено вигляд повної історії відпусток. Дана сторінка працює з фільтрами, сортуванням та пагінацією. Всі ці параметри є прив'язані до URL, що забезпечує отримання того ж самого результату при оновленні сторінки або при переході по посиланню:

Історія відпусток

Фільтри ^

Статус Підтверджена	Тип	З 06.06.2023	По 06.06.2023
Елементів на сторінці 20	Сортувати за Дата створення		

2023-06-06 - 2023-06-06 Підтверджена Лікарняний
2023-06-06 - 2023-06-06 Підтверджена Не оплачувана

< 1 >

Рисунок 3.20. Історія відпусток

При спробі перейти на сторінку, на перегляд якої у поточного користувача немає прав, буде показане відповідне повідомлення:

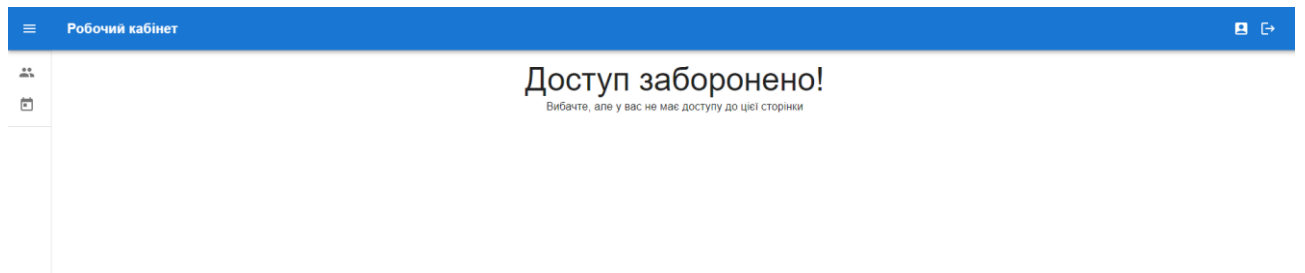


Рисунок 3.21. Помилка 403. Доступ заборонено.

При спробі перейти на неіснуючу сторінку користувач також буде бачити повідомлення про помилку:

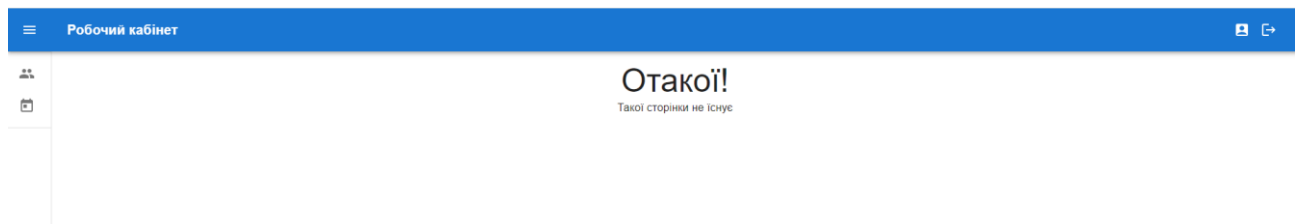


Рисунок 3.22. Помилка 404. Сторінку не знайдено.

Під час обробки запитів на сторінці відображається елемент підвантаження і весь інший UI стає недоступним для взаємодії:

					ІПЗ.КР.Б-121-23-ПЗ	
						62

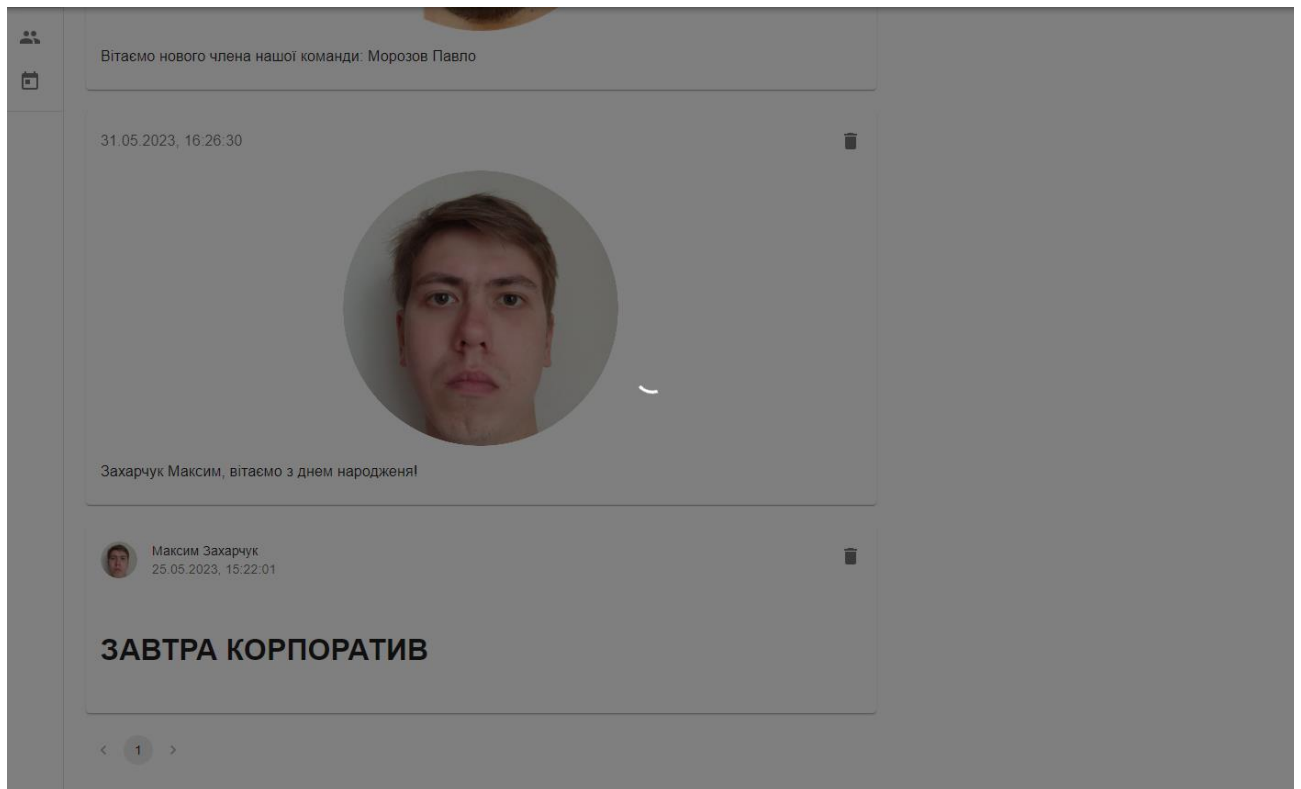


Рисунок 3.22. Підвантаження під час обробки запиту на видалення допису.

Коли користувач, що не є адміністратором, переходить на власний профіль, він може редагувати лише обмежену інформацію:

ОСНОВНІ

БЕЗПЕКА

Номер телефону

+380687405896

Стать

☒ Чоловік
 ☐ Жінка

Дата народження *

01.02.1995

Адреса

ЗБЕРЕГТИ

Рисунок 3.23. Редагування власного профілю користувачем, що не має ролі адміністратора

Також при редагуванні власного профілю користувач має можливість скинути пароль:

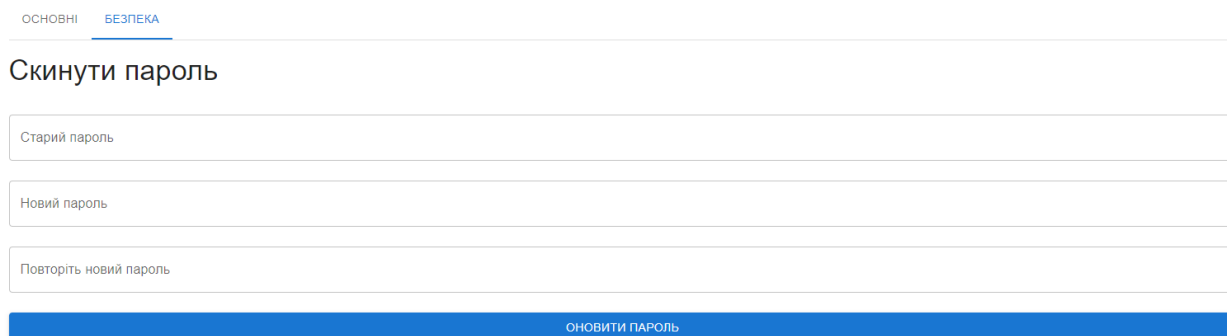


Рисунок 3.24. Форма для оновлення паролю.

Якщо користувач не пам'ятає свій пароль, він може його поновити, перейшовши за посиланнями «Забули пароль» на сторінці входу:

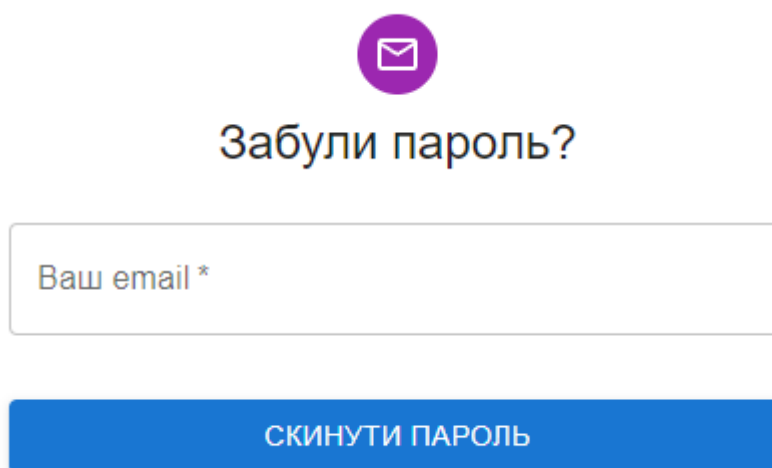


Рисунок 3.25. Сторінка «Забули пароль?»

На сторінці «Структура компанії» робітника має можливість переглянути ієрархію компанії(хто за кого відповідає). Спочатку відображаються користувачі, за яких ніхто не відповідає. Якщо хтось із них відповідає за іншим то його картку можна розгорнути і побачити список тих, за кого він відповідає.

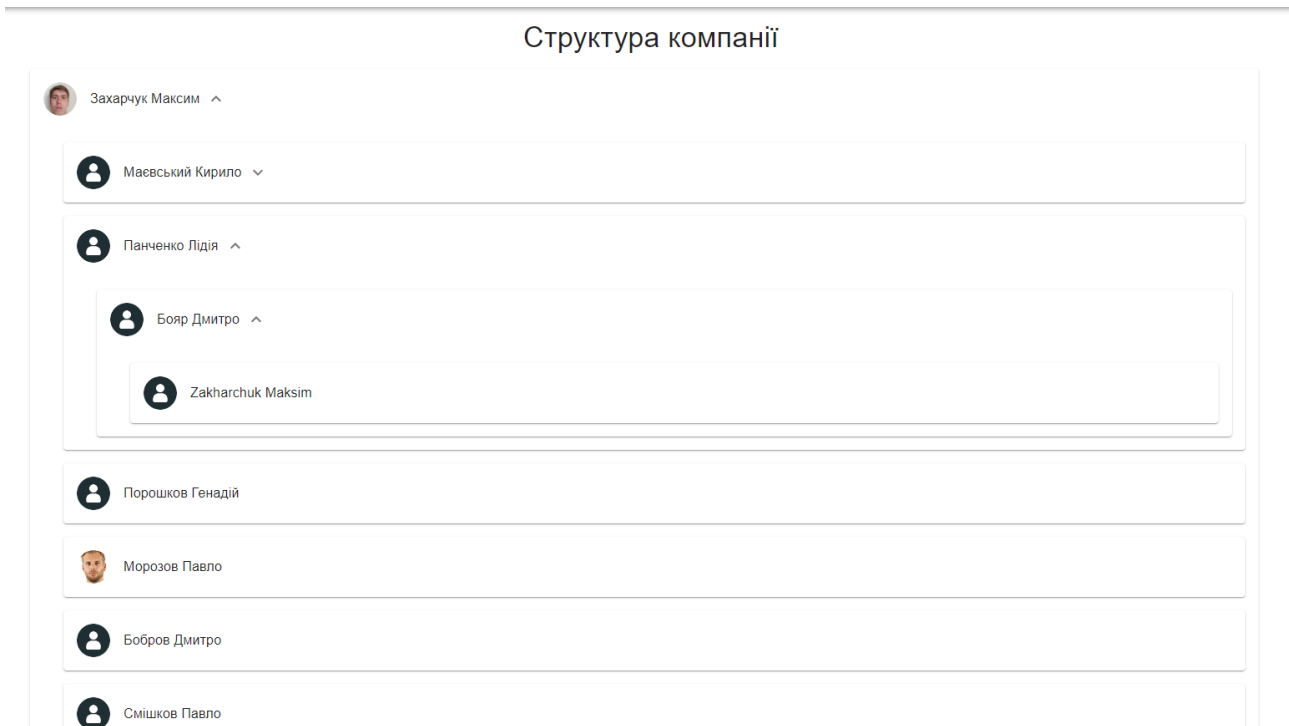


Рисунок 3.26. Сторінка «Структура компанії»

В системі наявна системи зустрічей. Зустріч – це запланована подія, що займає певний проміжок часу, яку організовує певний робітник і запрошує туди інших робітників. Сторінка зустрічей підтримує фільтрацію, де можна обирати дату, з якої показувати зустрічі (за замовчанням – поточний день), та робітника, що приймає участь в зустрічі. Фільтр для робітника працює як за організатором, так і за учасниками зустрічі.

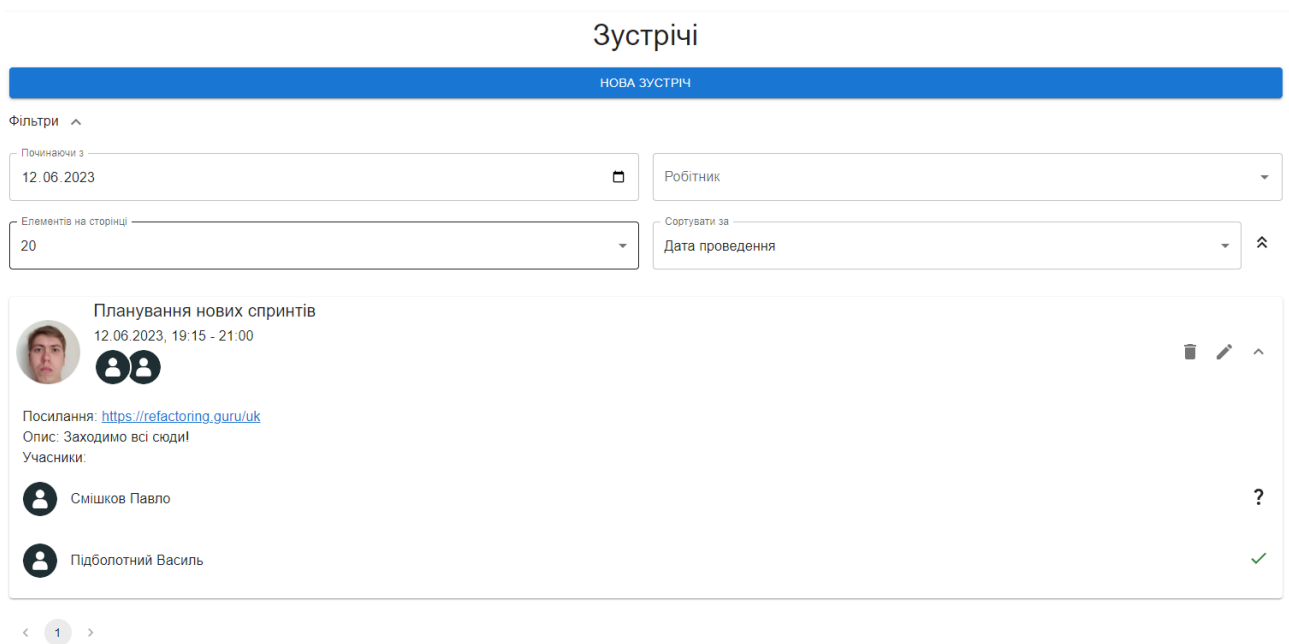


Рисунок 3.27. Список зустрічей

Кожен з робітників може створити зустріч та запросити туди інших робітників, натиснувши кнопку «Нова зустріч».

Нова зустріч

Назва *

Просто зустріч

Опис *

Просто збираємося

Посилання

https://meet.google.com/wcn-qwzy-qct

Учасники

Підболотний Василь(vasya228@gmail.com)

Масєвський Кирило(testaga@mail.com)

Дата та час *

06/14/2023 19:50

Тривалість (год.)

1,25

ЗБЕРЕГТИ

Рисунок 3.28. Форма для створення зустрічі

Робітників, яких запросили на зустріч. Можуть прийняти це запрошення або відхилити. Інші робітники будуть бачити відповідь цього учасника в деталях зустрічі.

Просто зустріч

14.06.2023, 16:30 - 17:45

Рисунок 3.29. Можливість прийняти або відхилити зустріч

Організатор зустрічі має можливість її відредагувати або видалити.

Просто зустріч

14.06.2023, 19:50 - 21:05

Рисунок 3.30. Керування зустріччю.

3.3 Тестування роботи додатку

Важливим етапом розробки програмного забезпечення є процес його тестування та виправлення дефектів. Для перевірки коректності роботи сервісу виконаємо процес створення заняття та перевіримо результат обробки відеозапису.

1. Вхід у обліковий запис:

передумови: користувач не увійшов в систему;

кроки: заповнення облікових даних, натиснути кнопку «Увійти»

очікуваний результат: користувач увійде в систему і буде перенаправлений на головну сторінку;

фактичний результат: користувач увійде в систему і буде перенаправлений на головну сторінку.

2. Створення оплачуваної відпустки, що перевищує кількість днів відпустки:

передумови: користувач знаходиться на сторінці відпусток;

кроки: заповнити форму запиту на відпустку, використовуючи тип «Оплачувана» і тривалістю, що перевищує кількість днів відпустки користувача

очікуваний результат: користувачу буде показне повідомлення про неможливість створити відпустку через те, що вона перевищує кількість днів оплачуваної відпустки;

фактичний результат: користувачу буде показне повідомлення про неможливість створити відпустку через те, що вона перевищує кількість днів оплачуваної відпустки.

3. Оновлення паролю:

передумови: користувач знаходиться на сторінці редагування профілю, на вкладці скидання паролю;

кроки: вписати старий пароль у відповідне поле, новий пароль вписати в поле «Новий пароль» і «Повторіть пароль»

очікуваний результат: користувач отримує повідомлення, що пароль було успішно змінено, вхід в систему працює за новим паролем;

фактичний результат: користувач отримує повідомлення, що пароль було успішно змінено, вхід в систему працює за новим паролем.

4. Створення відпустки з датою початку ранішою за поточний день:

передумови: користувач знаходиться на сторінці відпусток;

кроки: вибрати в полі «дата початку» дату, що є ранішою за поточний день.

очікуваний результат: дата, раніша за сьогоднішній день, буде недоступна для вибору;

фактичний результат: дата, раніша за сьогоднішній день, буде недоступна для вибору.

5. Зміна зображення профілю:

передумови: користувач знаходиться на сторінці редагування профілю;

кроки: користувач натискає видаляє старе фото, завантажує нове та обрізає його, натискає кнопку «зберегти».

очікуваний результат: фото профілю зміниться на щойно обране і відредаговане.

фактичний результат: фото профілю змінилося на щойно обране і відредаговане.

Висновки до третього розділу

В даному розділі було наведено схему розгортання додатку. Був проведений опис кожного структурного елемента та його взаємодію з іншими частинами системи.

Був проведений опис інтерфейсу розробленого програмного продукту. Опис включав в себе розгляд всіх функціональних можливостей додатку в залежності від ролі користувача.

Окрім цього, було проведене тестування додатку за допомогою тестового прикладу. Тестування додатку було здійснено покроково з метою перевірки коректності роботи на кожному етапі за допомогою порівняння очікуваних та фактичних результатів.

ВИСНОВКИ

Під час виконання випускної роботи бакалавра було спроектовано та реалізовано додаток «Онлайн робочий кабінет». Під час роботи над даною системою біли вирішені наступні завдання:

1. Проведений аналіз предметної області дозволив визначити основні вимоги до сервісу. На основі проведеного аналізу було визначено набір функціональних можливостей додатку.

2. Було проведено порівняння схожих за концепцією та функціоналом додатків. Визначено їх переваги та недоліки.

3. Проаналізовані наявні архітектурні підходи. Обрано модульну архітектуру для реалізації додатку, так як вона є найбільш оптимальною для реалізації поставленої задачі. Визначено бібліотеки NestJS та ReactJS як основні засоби для реалізації веб-додатку, а також мову програмування TypeScript (JavaScript).

4. Сформовані та проаналізовані функціональні та нефункціональні вимоги до системи, визначені варіанти використання. Визначені модулі, з яких складатиметься система та зв'язки між ними, визначена орієнтовна структура модуля. Наведені сутності системи, що є представленням даних, які зберігатимуться в системі.

5. Було описано структуру бази даних системи.

6. Було розглянуто алгоритми основних функцій додатку та проаналізовано взаємодію класів системи.

7. Було наведено детальний розгляд реалізації деяких основних функцій системи, як в серверній частині, так і в клієнтській.

8. Сформований порядок встановлення та налаштування веб-додатку.

9. Було описано інтерфейс додатку та наведено порядок роботи із системою.

10. Також було проведено тестування додатку з метою перевірки правильності його роботи.

					ІПЗ.КР.Б-121-23-ПЗ	
						69

Реалізований сервіс готовий до використання. Додаток може бути використано для спрощення і автоматизації певних процесів всередині певної компанії.

В подальшому даний програмний продукт може удосконалюватися шляхом введення нових функцій: коментування постів, реакція на пости, додавання до постів фото, створення задач робітникам, аналітика для адміністраторів, тощо.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. NestJS. "Official Documentation." [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.nestjs.com/>
2. React. "Official Documentation." [Електронний ресурс] – Режим доступу до ресурсу: <https://reactjs.org/docs/>
3. Node.js. "Official Documentation." [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/en/docs/>
4. TypeScript. "Official Documentation." [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/>
5. Apollo GraphQL. "Official Documentation." [Електронний ресурс] – Режим доступу до ресурсу: <https://www.apollographql.com/docs/>
6. JSON Web Tokens (JWT). "Official Website." [Електронний ресурс] – Режим доступу до ресурсу: <https://jwt.io/>
7. Material-UI (Mui). "Official Documentation." [Електронний ресурс] – Режим доступу до ресурсу: <https://mui.com/getting-started/installation/>
8. React Final Form. "Official Documentation." [Електронний ресурс] – Режим доступу до ресурсу: <https://final-form.org/react>
9. "JavaScript | MDN." [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>
10. "TypeScript Deep Dive." [Електронний ресурс] – Режим доступу до ресурсу: <https://basarat.gitbook.io/typescript/>
11. "The Modern JavaScript Tutorial." [Електронний ресурс] – Режим доступу до ресурсу: <https://javascript.info/>
12. "Node.js API Documentation." [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/api/>
13. "React Router Documentation." [Електронний ресурс] – Режим доступу до ресурсу: <https://reactrouter.com/>
14. "GraphQL | A query language for your API." [Електронний ресурс] – Режим доступу до ресурсу: <https://graphql.org/>

15. "MDN Web Docs." [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.mozilla.org/>
16. "React Hooks | A Complete Guide." [Электронный ресурс] – Режим доступа до ресурсу: <https://reactjs.org/docs/hooks-intro.html>
17. "React Context API." [Электронный ресурс] – Режим доступа до ресурсу: <https://reactjs.org/docs/context.html>
18. "Redux - A predictable state container for JavaScript apps." [Электронный ресурс] – Режим доступа до ресурсу: <https://redux.js.org/>
19. "Apollo Client | Apollo Client." [Электронный ресурс] – Режим доступа до ресурсу: <https://www.apollographql.com/docs/react/>
20. "JWT.io - JSON Web Tokens Introduction." [Электронный ресурс] – Режим доступа до ресурсу: <https://jwt.io/introduction/>
21. "Material-UI Examples." [Электронный ресурс] – Режим доступа до ресурсу: <https://mui.com/getting-started/example-projects/>
22. "React Final Form Examples." [Электронный ресурс] – Режим доступа до ресурсу: <https://final-form.org/docs/react-final-form/examples>
23. "TypeScript Handbook." [Электронный ресурс] – Режим доступа до ресурсу: <https://www.typescriptlang.org/docs/handbook/>
24. "React Patterns." [Электронный ресурс] – Режим доступа до ресурсу: <https://reactpatterns.com/>
25. "Node.js Best Practices." [Электронный ресурс] – Режим доступа до ресурсу: <https://github.com/goldbergonyi/nodebestpractices>
26. "The Complete React Developer Course (w/ Hooks and Redux)." [Электронный ресурс] – Режим доступа до ресурсу: <https://www.udemy.com/course/react-2nd-edition/>
27. "Fullstack React: The Complete Guide to ReactJS and Friends." [Электронный ресурс] – Режим доступа до ресурсу: <https://www.fullstackreact.com/>
28. "Testing JavaScript." [Электронный ресурс] – Режим доступа до ресурсу: <https://testingjavascript.com/>

29. "Practical TypeScript." [Электронный ресурс] – Режим доступа до ресурсу: <https://www.typescriptlang.org/docs/handbook/TypeScript-in-5-minutes.html>
30. "RESTful API Design - Best Practices in API Design." [Электронный ресурс] – Режим доступа до ресурсу: <https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design>
31. "TypeORM - Amazing ORM for TypeScript and JavaScript." [Электронный ресурс] – Режим доступа до ресурсу: <https://typeorm.io/>
32. "PostgreSQL: The World's Most Advanced Open Source Relational Database." [Электронный ресурс] – Режим доступа до ресурсу: <https://www.postgresql.org/>
33. "The Modern React Bootcamp (Hooks, Context, NextJS, Router)." [Электронный ресурс] – Режим доступа до ресурсу: <https://www.udemy.com/course/modern-react-bootcamp/>
34. "Full-Stack React with GraphQL and Apollo Boost." [Электронный ресурс] – Режим доступа до ресурсу: <https://www.udemy.com/course/full-stack-react-with-graphql-and-apollo-boost/>

ДОДАТКИ

					ІІЗ.КР.Б-121-23-ІІЗ	
						74

Лістинг user.service.ts:

```

import { HttpException, HttpStatus, Injectable } from '@NestJS/common';
import { InjectRepository } from '@NestJS/typeorm';
import { UserEntity } from 'src/users/entities/user.entity';
import { CreateUserInput } from 'src/users/dto/create-user.input';
import { UpdateUserInput } from 'src/users/dto/update-user.input';
import { Brackets, FindOperator, ILike, Like, Repository, TreeRepository,
createQueryBuilder } from 'typeorm';
import * as bcrypt from 'bcrypt';
import { Role } from 'src/auth/role.enum';
import { IPaginationOptions, Pagination, paginate } from 'NestJS-typeorm-
paginate';
import { createWriteStream, mkdirSync } from 'fs';
import path, { join } from 'path';
import * as fs from 'fs';
import { error } from 'console';
import { MailingService } from 'src/mailling/mailling.service';
import * as generator from "generate-password";
import { UpdateUserPersonalInfoInput } from 'src/users/dto/update-user-personal-
info.input';
import { ChangePasswordInput } from 'src/users/dto/change-password.input';
import { PostService } from 'src/posts/posts.service';
import { PostTypes } from 'src/posts/posts-types.enum';
import { EntityFilter } from 'src/common/dto/filter.type';

@Injectable()
export class UserService {
  constructor(
    @InjectRepository(UserEntity)
    private readonly userRepository: Repository<UserEntity>,
    private readonly mailingService: MailingService,
    private readonly postsService: PostService,
  ) { }

  async createUser(createUserInput: CreateUserInput, creatorId: number):
  Promise<UserEntity> {
    if (!createUserInput.password)
      createUserInput.password = generator.generate({ length: 16, numbers:
true });
    const password = await bcrypt.hash(createUserInput.password, 10);

    const { image, ...user } = createUserInput;
    const savedUser = await this.userRepository.save(
      {
        ...user,
        password,
        roleId: 3,
        hasImage: !!image,
      }
    )
    if (image) {
      this.saveAvatar(image, savedUser.id);
    }

    this.mailingService.sendNewAccountMail(user.email, user.firstName,
user.lastName, createUserInput.password);
    if (createUserInput.startWorkingDay.toString() === (new
Date()).toString()) {
      this.postsService.create(
        {

```

```

        text: `Вітаємо нового члена нашої команди: ${user.lastName}
        ${user.firstName}`,
        image,
    },
    creatorId,
    PostTypes.NewHires
    );
    }

    return savedUser;
}

async getOneUser(id: number): Promise<UserEntity> {
    const user = await this.userRepository.findOne({ where: { id },
relations: { manager: true, teamMembers: true } });
    return user;
}

async getOneUserByEmail(email: string): Promise<UserEntity> {
    return await this.userRepository.findOneBy({ email });
}

async tree(managerId?: number): Promise<UserEntity[]> {
    const user = await this.userRepository.find({ where: { managerId },
relations: { teamMembers: true } });
    return user;
}

async getAllUsers(filters: EntityFilter[]): Promise<UserEntity[]> {
    const query = this.userRepository
        .createQueryBuilder('users')

    if (filters) {
        filters.forEach((f, i) => {
            if (!f.value)
                return;

            const value = {}
            value[`val${i}`] = f.value

            if (f.field === "startWorkingDate") {
                query.andWhere(`to_char("startWorkingDay",'dd') = :val${i}`,
value);
            } else if (f.field === "birthday") {
                query.andWhere(`to_char("birthDate",'MM.dd') = :val${i}`,
value);
            } else {
                query.andWhere(`users.${f.field} = :val${i}`, value)
            }
        })
    }

    return await query.getMany();
}

async paginate(options: IPaginationOptions, sorting: string, descending:
boolean, search: string, filters: EntityFilter[]):
Promise<Pagination<UserEntity>> {
    const query = this.userRepository
        .createQueryBuilder('users')
        .orderBy(`users.${sorting}`, descending ? 'DESC' : 'ASC')
        .leftJoinAndSelect("users.position", "position")
        .leftJoinAndSelect("users.office", "office")
        .leftJoinAndSelect("users.department", "department")

```

```

        .where(
            new Brackets((qb) => {
                qb
                    .where('users.firstName || \' \' || users.lastName ILIKE
:search')
                    .orWhere('users.lastName || \' \' || users.firstName
ILIKE :search')
                    .orWhere('users.email ILIKE :search')
            }),
            { search: `_${!!search ? search : ''}%` }
        );

    if (filters) {
        filters.forEach((f, i) => {
            if (!f.value)
                return;

            const value = {}
            value[`val${i}`] = f.value

            if (f.field === "birthday") {
                query.andWhere(`to_char("birthDate",'MM.dd') >= :val${i}`,
value);

                query.orderBy(`to_char("birthDate",'MM.dd')`)
            } else {
                query.andWhere(`users.${f.field} = :val${i}`, value)
            }
        })
    }

    const users = await paginate<UserEntity>(query, options);
    return users
}

async removeUser(id: number): Promise<number> {
    await this.userRepository.delete({ id })
    return id;
}

async updateUser(updateUserInput: UpdateUserInput |
UpdateUserPersonalInfoInput, changeImage: boolean = true): Promise<UserEntity> {
    const { image, ...user } = updateUserInput;
    let hasImage = changeImage ? false : true;

    if (image && changeImage) {
        this.saveAvatar(image, user.id);
        hasImage = true
    }

    await this.userRepository.update({ id: updateUserInput.id }, { ...user,
hasImage });
    return await this.findOneUser(updateUserInput.id);
}

async changePasswrod(userId: number, changePasswordInput:
ChangePasswordInput): Promise<boolean> {
    const { oldPassword, password: newPassword, passwordRepeat:
newPasswordRepeat } = changePasswordInput;

    if (newPassword !== newPasswordRepeat)
        return false;

    const user = await this.findOneUser(userId);

```

```

        const isOldPasswordValid = await bcrypt.compare(oldPassword,
user?.password);
        if (!isOldPasswordValid)
            return false;

        const password = await bcrypt.hash(newPassword, 10);
        await this.userRepository.update({ id: userId }, { password });
        return true;
    }

    saveAvatar(image: string, userId: number): void {
        mkdirSync(join(process.cwd(), `./files/photos/${userId}`), { recursive:
true });

        const base64Data = image.replace(/^data:image\/png;base64/, "");
        const buf = Buffer.from(base64Data, 'base64');

        fs.writeFile(join(process.cwd(), `./files/photos/${userId}/avatar.png`),
buf, (err) => {
            if (err)
                throw err;
            return true;
        });
    }
}

```

Лістинг app.module.ts:

```

import { ApolloDriver, ApolloDriverConfig } from '@NestJS/apollo';
import { Module } from '@NestJS/common';
import { ConfigModule, ConfigService } from '@NestJS/config';
import { GraphQLModule } from '@NestJS/GraphQL';
import { TypeOrmModule } from '@NestJS/typeorm';
import { join } from 'path';
import { UsersModule } from '../users/users.module';
import { AuthModule } from '../auth/auth.module';
import { VacationModule } from '../vacation/vacation.module';
import { PositionModule } from '../position/position.module';
import { ServeStaticModule } from '@NestJS/serve-static';
import { PostsModule } from '../posts/posts.module';
import { MailingModule } from '../mailing/mailing.module';
import { MailerModule } from '@NestJS-modules/mailer';
import { HandlebarsAdapter } from '@NestJS-
modules/mailer/dist/adapters/handlebars.adapter';
import { auth } from 'googleapis/build/src/apis/abusiveexperiencereport';
import { OfficeModule } from '../office/office.module';
import { DepartmentModule } from '../department/department.module';
import { HolidaysModule } from '../holidays/holidays.module';
import { ScheduleModule } from '@NestJS/schedule';
import { TasksModule } from '../tasks/tasks.module';

@Module({
  imports: [
    ScheduleModule.forRoot(),
    TasksModule,
    UsersModule,
    ConfigModule.forRoot({ isGlobal: true }),
    GraphQLModule.forRoot({
      context: ({ req, res }) => ({ req, res }),
      driver: ApolloDriver,
      cors: {
        credentials: true,
        origin: true,
      },
      autoSchemaFile: 'schema.gql',
      sortSchema: true,
      playground: true,
      fieldResolverEnhancers: ['guards'],
    }),
    TypeOrmModule.forRootAsync({
      imports: [ConfigModule],
      useFactory: (configService: ConfigService) => ({
        type: 'postgres',
        host: configService.get('DB_HOST'),
        port: +configService.get<number>('DB_PORT'),
        username: configService.get('DB_USERNAME'),
        password: configService.get('DB_PASSWORD'),
        database: configService.get('DB_NAME'),
        entities: [__dirname + 'dist/**/*.entity{.ts,.js}'],
        synchronize: true,
        autoLoadEntities: true,
        logging: true,
      }),
      inject: [ConfigService],
    }),
    AuthModule,
    VacationModule,
    PositionModule,
  ],
})

```

```

    ServeStaticModule.forRoot({
      rootPath: join(__dirname, '..', 'files'),
      serveRoot: '/files/'
    }),
    PostsModule,
    MailingModule,
    ConfigModule.forRoot({
      isGlobal: true,
      envFilePath: ['.env', '.env.development'],
    }),
    MailerModule.forRoot({
      transport: {
        host: 'smtp.gmail.com',
        auth: {
          user: process.env.EMAIL,
          pass: process.env.PASS,
        }
      },
      template: {
        dir: process.cwd() + '/templates/',
        adapter: new HandlebarsAdapter(),
        options: {
          strict: true,
        },
      },
    }),
    OfficeModule,
    DepartmentModule,
    HolidaysModule,
  ],
  controllers: [],
})
export class AppModule { }

```

Лістинг user.resolver.ts:

```

import { UseGuards } from '@NestJS/common';
import { Args, Context, Mutation, ObjectType, Query, ResolveField, Resolver }
from '@NestJS/GraphQL';
import { Role } from 'src/auth/role.enum';
import { Roles } from 'src/auth/roles.decorator';
import { RolesGuard } from 'src/auth/roles.guard';
import { PaginationInput } from 'src/common/dto/paginate.input';
import { ChangePasswordInput } from 'src/users/dto/change-password.input';
import { CreateUserInput } from 'src/users/dto/create-user.input';
import { UpdateUserPersonalInfoInput } from 'src/users/dto/update-user-personal-
info.input';
import { UpdateUserInput } from 'src/users/dto/update-user.input';
import { UserPage } from 'src/users/dto/user-page.type';
import { UserEntity } from 'src/users/entities/user.entity';
import { UserService } from 'src/users/services/user/user.service';

@ObjectType()
class UserQueryType { }

@Resolver(() => UserQueryType)
export class UserResolver {
  constructor(
    private readonly userService: UserService,
  ) { }

  @Query(returns => UserQueryType)
  async user() {
    return new UserQueryType();
  }

  @ResolveField(() => UserEntity)
  async byId(@Args('id') id: number): Promise<UserEntity> {
    return await this.userService.getOneUser(id);
  }

  @ResolveField(() => UserEntity)
  async current(@Context() context): Promise<UserEntity> {
    return await this.userService.getOneUser(context.req.user.id);
  }

  @ResolveField(() => [UserEntity])
  async tree(@Args('managerId', { nullable: true }) managerId?: number):
  Promise<UserEntity[]> {
    return await this.userService.tree(managerId);
  }

  @ResolveField(() => UserPage)
  async all(@Args('paginationInput') paginationInput: PaginationInput) {
    const pageItem = await this.userService.paginate(
      { limit: paginationInput.limit, page: paginationInput.page },
      paginationInput.sorting || 'id',
      paginationInput.descending,
      paginationInput.search,
      paginationInput.filters,
    );
    return new UserPage(pageItem.meta.totalItems, pageItem.meta.totalPages,
      pageItem.items);
  }
}

```

```

    /*@ResolveField(() => UserPage)
    async managers(@Args('paginationInput') paginationInput: PaginationInput) {
        const pageItem = await this.userService.paginate(
            { limit: paginationInput.limit, page: paginationInput.page },
            paginationInput.sorting || 'id',
            paginationInput.descending,
            paginationInput.search,
        );
        return new UserPage(pageItem.meta.totalItems, pageItem.meta.totalPages,
            pageItem.items);
    }*/

    @ObjectType()
    class UserMutationType { }

    @Resolver(() => UserMutationType)
    export class UserMutationResolver {
        constructor(
            private readonly userService: UserService,
        ) { }

        @Mutation(returns => UserMutationType)
        async user() {
            return new UserMutationType();
        }

        @ResolveField(() => UserEntity)
        @Roles(Role.Admin)
        @UseGuards(RolesGuard)
        async createUser(@Args('createUser') createUserInput: CreateUserInput,
            @Context() context): Promise<UserEntity> {
            return await this.userService.createUser(createUserInput,
                context.req.user.id)
        }

        @ResolveField(() => UserEntity)
        @Roles(Role.Admin)
        @UseGuards(RolesGuard)
        async updateUser(@Args('updateUser') updateUserInput: UpdateUserInput):
            Promise<UserEntity> {
            return await this.userService.updateUser(updateUserInput)
        }

        @ResolveField(() => UserEntity)
        async updatePersonalInfo(@Args('updateUser') updateUser:
            UpdateUserPersonalInfoInput, @Context() context): Promise<UserEntity> {
            updateUser.id = context.req.user.id;
            return await this.userService.updateUser(updateUser)
        }

        @ResolveField(() => Boolean)
        async changePassword(@Args('changePasswordInput') changePasswordInput:
            ChangePasswordInput, @Context() context): Promise<boolean> {
            return await this.userService.changePasswrod(context.req.user.id,
                changePasswordInput)
        }

        @ResolveField(() => Number)
        async removeUser(@Args('id') id: number): Promise<Number> {
            return await this.userService.removeUser(id)
        }
    }

```

Лістинг user.entity.ts:

```

import { Field, Float, ID, Int, ObjectType } from '@NestJS/GraphQL';
import { RoleEntity } from 'src/auth/entities/role.entity';
import { DepartmentEntity } from 'src/department/entities/department.entity';
import { OfficeEntity } from 'src/office/entities/office.entity';
import { PositionEntity } from 'src/position/entities/position.entity';
import {
  Column,
  CreateDateColumn,
  Entity,
  ManyToOne,
  OneToMany,
  PrimaryGeneratedColumn,
  UpdateDateColumn
} from 'typeorm';
import { Gender } from '../enums/gender.enum';

@ObjectType()
@Entity('users')
export class UserEntity {
  @Field(() => ID)
  @PrimaryGeneratedColumn()
  id: number;

  @Field()
  @CreateDateColumn()
  createdAt: Date;

  @Field()
  @UpdateDateColumn()
  updatedAt: Date;

  @Field()
  @Column({ unique: true })
  email: string;

  @Column()
  password: string;

  @Field()
  @Column()
  firstName: string;

  @Field()
  @Column()
  lastName: string;

  @Field()
  @Column()
  startWorkingDay: Date;

  @Field()
  @Column()
  birthDate: Date;

  @Field(() => Float)
  @Column({ default: 0, type: 'float' })
  vacationDaysCount: number;

  @Field()

```

```

    @ManyToOne(() => RoleEntity, (role) => role.name, { cascade: true, eager: true
    })
    role: RoleEntity

    @Column()
    roleId: number;

    @Field()
    @ManyToOne(() => PositionEntity, { cascade: true, eager: true })
    position: PositionEntity

    @Field()
    @Column()
    positionId: number;

    @Field()
    @ManyToOne(() => OfficeEntity, (office) => office.users, { cascade: true,
    eager: true })
    office: OfficeEntity

    @Field()
    @Column()
    officeId: number;

    @Field()
    @ManyToOne(() => DepartmentEntity, { cascade: true, eager: true })
    department: DepartmentEntity

    @Field()
    @Column()
    departmentId: number;

    @Field({ defaultValue: false })
    @Column({ default: false })
    hasImage: boolean;

    @Field({ nullable: true })
    @Column({ nullable: true })
    address: string;

    @Field()
    @Column({ default: false })
    isPartTime: boolean;

    @Field(() => Int, { nullable: true })
    @Column({ nullable: true })
    partTimeHours: number;

    @Field({ nullable: true })
    @Column({ nullable: true })
    phoneNumber: string;

    @Field(() => Gender)
    @Column({ default: "Male" })
    gender: Gender;

    @Field({ nullable: true })
    @Column({ nullable: true })
    managerId: number;

    @Field({ nullable: true })
    @ManyToOne(() => UserEntity, (user) => user.teamMembers)
    manager: UserEntity

```

```
@Field(() => [UserEntity], { nullable: true })
@OneToMany(() => UserEntity, (user) => user.manager, { cascade: true })
teamMembers: UserEntity[]
}
```

Лістинг App.tsx:

```
import { ApolloProvider } from '@apollo/client';
import { CssBaseline, ThemeProvider } from "@mui/material";
import UserContextProvider from "../../contexts/user/userContext";
import Router from "../../Router";
import { apolloClient } from '../apolloClient';
import { mainTheme } from '../theme';

function App() {
  return (
    <>
      <CssBaseline />
      <ApolloProvider client={apolloClient}>
        <UserContextProvider>
          <ThemeProvider theme={mainTheme}>
            <Router />
          </ThemeProvider>
        </UserContextProvider>
      </ApolloProvider>
    </>
  );
}

export default App;
```

Лістинг apolloClient.tsx:

```

import { ApolloClient, HttpLink, InMemoryCache } from "@apollo/client";
import { setContext } from "@apollo/client/link/context";
import { onError } from "@apollo/client/link/error";
import { createUploadLink } from 'apollo-upload-client';
import { redirect } from "react-router-dom";
import { AUTH, URLS } from "../../constants";
import { history } from "../../helpers/history";

const httpLink = new HttpLink({ uri: 'http://localhost:3030/GraphQL',
credentials: 'include' });

/*const uploadLink = createUploadLink({
  uri: URLS.server + '/GraphQL',
  credentials: 'include',
  headers: {'Apollo-Require-Preflight': 'true'},
})*//

const authLink = setContext((_, { headers }) => {
  const token = localStorage.getItem(AUTH.token);
  return {
    headers: {
      ...headers,
      authorization: token ? `Bearer ${token}` : "",
    }
  }
});

const errorLink = onError(({ GraphQLErrors, networkError, response }) => {
  console.log(networkError);
  if (GraphQLErrors) {
    GraphQLErrors.forEach(({ message, locations, path, extensions }) => {
      console.log(
        `[GraphQL error]: Message: ${message}, Location: ${locations}, Path: ${path}`
      );
      const statusCode = (extensions.originalError as any)?.statusCode;
      if (statusCode === 401) {
        if (history.navigate)
          history.navigate('/login');
        localStorage.removeItem(AUTH.token);
        localStorage.removeItem(AUTH.user);
        localStorage.removeItem(AUTH.role);
      }
    });
  }
  if (networkError) {
    console.log(`[Network error]: ${networkError}`);
    if ('statusCode' in networkError && networkError.statusCode === 401) {
      redirect("/login");
    }
  }
});

export const apolloClient = new ApolloClient({
  link: authLink.concat(errorLink.concat(httpLink)),
  cache: new InMemoryCache(),
});

```

Лістинг Router.tsx:

```

import { createBrowserRouter, Outlet, RouterProvider } from "react-router-dom";
import Layout from "../../layout";
import Error404 from "../../pages/Error404";
import Home from "../../pages/Home";
import Login from "../../pages/Login";
import Profile from "../../pages/Profile";
import ProtectedRoute from "../ProtectedRoute";
import AddEmployee from "../../pages/Profile/AddEmployee";
import { ROLES } from "../../constants";
import UserContextProvider from "../../contexts/user/userContext";
import ProfileList from "../../pages/Profile/List";
import EditEmployee from "../../pages/Profile/EditEmployee";
import EditOwn from "../../pages/Profile/EditOwn";
import SettingsLayout from "../../layout/settings";
import Positions from "../../pages/Settings/Positions";
import Vacation from "../../pages/Vacation";
import VacationList from "../../pages/Vacation/List";
import VacationHistory from "../../pages/Vacation/History";
import Offices from "../../pages/Settings/Offices";
import Departments from "../../pages/Settings/Departments";
import Holidays from "../../pages/Settings/Holidays";

const router = createBrowserRouter([
  {
    path: "/",
    element:
      <Layout />
    ,
    children: [
      {
        path: "*",
        element: <Error404 />
      },
      {
        path: "/",
        element:
          <ProtectedRoute>
            <Home />
          </ProtectedRoute>,
        index: true,
      },
      {
        path: "profile",
        children: [
          {
            path: ":userId?",
            element:
              <ProtectedRoute>
                <Profile />
              </ProtectedRoute>
          },
          {
            path: "list/:q?",
            element:
              <ProtectedRoute>
                <ProfileList />
              </ProtectedRoute>,
          },
        ],
      },
    ],
  },
]);

```

```

        path: "add",
        element:
          <ProtectedRoute roles={ [ROLES.admin]} >
            <AddEmployee />
          </ProtectedRoute>,
      },
      {
        path: "edit",
        element:
          <ProtectedRoute >
            <EditOwn />
          </ProtectedRoute>,
      },
      {
        path: "edit/:id",
        element:
          <ProtectedRoute roles={ [ROLES.admin]} >
            <EditEmployee />
          </ProtectedRoute>,
      },
    ],
  },
  {
    path: "vacation",
    children: [
      {
        path: '',
        element:
          <ProtectedRoute>
            <Vacation />
          </ProtectedRoute>
      },
      {
        path: "list",
        element:
          <ProtectedRoute roles={ [ROLES.admin]} >
            <VacationList />
          </ProtectedRoute>
      },
      {
        path: "history",
        element:
          <ProtectedRoute>
            <VacationHistory />
          </ProtectedRoute>
      },
    ],
  },
  {
    path: "settings",
    element:
      <ProtectedRoute roles={ [ROLES.admin]} >
        <SettingsLayout />
      </ProtectedRoute>,
    children: [
      {
        path: "",
        element: <Positions />,
        index: true,
      },
      {
        path: "positions",
        element: <Positions />,
      },
    ],
  },

```

```

        {
            path: "offices",
            element: <Offices />,
        },
        {
            path: "departments",
            element: <Departments />,
        },
        {
            path: "holidays",
            element: <Holidays />,
        }
    ]
},
],
},
{
    path: "/login",
    element: <Login />,
}
]);

export default function Router() {
    return (
        <RouterProvider router={router} />
    )
}

```