

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

«Сервіс для автоматизації розсилання повідомлень та
обліку робочого часу працівників університету»

Виконав студент групи ІПЗм-22-2

САВЧУК Іван Олександрович

Керівник роботи:

МОРОЗОВ Андрій Васильович

Рецензент:

ЄФРЕМОВ Юрій Миколайович

Житомир – 2023

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«18» вересня 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **САВЧУК Іван Олександрович**

Керівник роботи: **МОРОЗОВ Андрій Васильович**

Тема роботи: **«Сервіс для автоматизації розсилання повідомлень та обліку робочого часу працівників університету»**,

затверджена Наказом закладу вищої освіти від **«18» вересня 2023 р. № 479/с**

Вихідні дані для роботи: Об'єктом дослідження є процеси комунікації та обліку робочого часу в університетському середовищі.

Предметом дослідження є засоби реалізації та впровадження автоматизованих систем для підвищення ефективності процесів комунікації та обліку робочого часу в університеті.

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Локтікова Т.М.	18.09.2023р.	18.09.2023р.
2	Локтікова Т.М.	11.10.2023р.	11.10.2023р.
3	Локтікова Т.М.	24.11.2023р.	24.11.2023р.

Дата видачі завдання: **«18» вересня 2023 р.**

РЕФЕРАТ

Завдання кваліфікаційної роботи магістра полягає у науковому дослідженні та обґрунтуванні вибору моделей, методів та технологій, які будуть використовуватися для розробки сервісу, призначеного для автоматизації розсилання повідомлень та обліку робочого часу працівників університету. Дослідження включає аналіз та вибір оптимальних інструментів для реалізації веб-додатку, що слугуватиме для ефективного управління та моніторингу робочого часу, та чат-бота в Telegram, який буде використовуватися для оперативного інформування студентів.

Кваліфікаційна магістерська робота на тему «Сервіс для автоматизації розсилання повідомлень та обліку робочого часу працівників університету» складається з програмного комплексу сервісу та пояснювальної записки. Пояснювальна записка до кваліфікаційної магістерської роботи складається зі вступу, трьох розділів, списку літератури з 21 найменування, 45 рисунків, 3 таблиць та 3 додатків. Повний обсяг роботи становить 81 сторінка.

КЛЮЧОВІ СЛОВА: ВЕБДОДАТОК, ФРЕЙМВОРК, ЧАТ-БОТ, ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ, ОБЛІК, АСИНХРОННІСТЬ.

ABSTRACT

The task of the master's thesis involves scientific research and justification of the choice of models, methods, and technologies that will be used for the development of a service designed for the automation of message distribution and working time accounting for university employees. The study includes an analysis and selection of optimal tools for the implementation of a web application, which will serve for effective management and monitoring of working hours, and a chatbot in Telegram, which will be used for prompt communication with students.

The master's thesis on "Service for Automation of Message Distribution and Working Time Accounting for University Employees" consists of the software complex of the service and an explanatory note. The explanatory note to the master's thesis consists of an introduction, three chapters, a bibliography with 25 titles, 45 figures, 3 tables, and 3 appendices. The total volume of the work is 82 pages.

KEYWORDS: WEB APPLICATION, FRAMEWORK, CHATBOT, PROGRAMMING TECHNOLOGIES, ACCOUNTING, ASYNCHRONY.

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1 АНАЛІЗ НАПРЯМІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СЕРВІСУ ДЛЯ АВТОМАТИЗАЦІЇ РОЗСИЛАННЯ ПОВІДОМЛЕНЬ ТА ОБЛІКУ РОБОЧОГО ЧАСУ ПРАЦІВНИКІВ УНІВЕРСИТЕТУ	9
1.1 Постановка задачі	9
1.2 Огляд й аналіз аналогів	10
1.3 Обґрунтування вибору інструментальних засобів для розробки.....	21
1.4 Вибір архітектури	26
Висновки до першого розділу	27
РОЗДІЛ 2 ПРОЕКТУВАННЯ СЕРВІСУ ДЛЯ АВТОМАТИЗАЦІЇ РОЗСИЛАННЯ ПОВІДОМЛЕНЬ ТА ОБЛІКУ РОБОЧОГО ЧАСУ ПРАЦІВНИКІВ УНІВЕРСИТЕТУ	29
2.1 Розробка структури бази даних.....	29
2.2 Проектування та реалізація алгоритмів роботи сервісу для автоматизації розсилання повідомлень	35
2.3 Реалізація сервісу обліку робочого часу працівників університету	43
Висновки до другого розділу	51
РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СЕРВІСОМ.....	52
3.1 Структура вебдодатка.....	52
3.2 Структура чат-бота	56
3.3 Тестування роботи ПЗ	57
Висновки до третього розділу.....	66
ВИСНОВКИ.....	67
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	69
ДОДАТКИ.....	72

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

PHP – PHP: Hypertext Preprocessor

SQL – Structured Query Language

ВСТУП

У сучасному світі, де кожен аспект людського життя можна описати за допомогою статистики, значно зросла потреба у розробці та впровадженні ефективних інструментів для збору, аналізу та використання цих даних. Ця потреба стає особливо актуальною у сфері освіти, де обробка інформації та її оперативне поширення є критично важливими. Університети, як багатофункціональні освітні установи, зіштовхуються з низкою викликів, пов'язаних із управлінням інформацією та ефективною комунікацією, які включають в себе взаємодію як зі студентами, так і з персоналом.

В університетському середовищі, де кожен день генерується величезна кількість даних – від академічних досягнень студентів до адміністративних записів – наявність надійних систем для ефективного управління цією інформацією є надзвичайно важливою. Це включає в себе не тільки зберігання даних, але й їх аналіз, обробку та представлення у зручному та доступному форматі. Таким чином, університети можуть використовувати ці дані для покращення якості навчання, планування ресурсів, а також для забезпечення більш ефективної взаємодії між студентами, викладачами та адміністративним персоналом.

Однак, не дивлячись на наявність численних цифрових інструментів, багато університетів все ще зіштовхуються з проблемами неефективного використання та розподілу інформаційних ресурсів. Це може призводити до втрати важливої інформації, неправильного розподілу завдань та неефективної взаємодії між усіма учасниками освітнього процесу. У зв'язку з цим, існує важлива потреба у розробці інтегрованих систем, які б могли не тільки автоматизувати процеси збору та обробки інформації, але й забезпечувати її оперативне поширення серед відповідних груп університетської спільноти.

Мета роботи – розробка автоматизованої системи для розсилання повідомлень та обліку робочого часу працівників університету.

Визначена мета вирішується шляхом виконання завдань:

1. Аналіз сучасного стану комунікацій та систем обліку часу.

2. Розробка концепції та архітектури сервісу, з урахуванням специфіки роботи університету.
3. Сформулювати постановку задачі та розробити завдання.
4. Обрати технології та засоби реалізації проекту.
5. Реалізувати вебдодаток з використанням сучасних технологій та фреймворків та чат-бот в максимально зручному для користувачів середовищі.
6. Реалізувати інформаційну систему та протестувати її.

Об'єкт дослідження – процеси комунікації та обліку робочого часу в університетському середовищі.

Предмет дослідження – засоби реалізації та впровадження автоматизованих систем для підвищення ефективності процесів комунікації та обліку робочого часу в університеті.

Наукова новизна роботи полягає в аналізі потенціалу сучасних технологій у сфері освіти та розробка інноваційного рішення для вдосконалення внутрішніх процесів університету.

Практичне значення роботи полягає у впровадженні автоматизованої системи в університеті для поліпшення комунікації та обліку робочого часу, що сприяє підвищенню якості навчання, оптимізації ресурсів і ефективнішій взаємодії між учасниками освітнього процесу.

РОЗДІЛ 1 АНАЛІЗ НАПРЯМІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СЕРВІСУ ДЛЯ АВТОМАТИЗАЦІЇ РОЗСИЛАННЯ ПОВІДОМЛЕНЬ ТА ОБЛІКУ РОБОЧОГО ЧАСУ ПРАЦІВНИКІВ УНІВЕРСИТЕТУ

1.1 Постановка задачі

Сервіс буде складатися з двох складових частин – чат-бота для автоматичного розсилання повідомлень та вебдодатка для обліку робочого часу працівників університету.

Основними користувачами сервісу будуть адміністративний персонал, викладачі та студенти. Вебдодаток використовуватиметься для відстеження та обліку робочого часу, планування занять та координації зустрічей зі студентами. Адміністративний персонал зможе використовувати вебдодаток для моніторингу робочого часу своїх підлеглих, планування та розподілу робочих завдань. Також веб додаток може бути корисним для відділу кадрів для ведення обліку робочих годин і легшого доступу до даних про відвідуваність та ефективність працівників. Чат-бот використовуватиметься адміністративним персоналом для інформування студентів та викладачів про майбутні події та заходи. Викладачі та студенти матимуть змогу переглядати свій розклад на день чи свій повний розклад. Також, студенти матимуть змогу переглядати свій журнал успішності з предметів.

Вебдодаток має містити функціонал для планування та відстеження робочого часу, запису відпрацьованих робочих годин за день, за попередні дні, можливість перегляду та редагування попередніх записів. Також, за наявності робочого плану, має бути наявність відстеження виконання плану за запропонований проміжок часу. Адміністративний персонал має мати можливість налаштовувати та контролювати робочий час своїх підлеглих, вивантажувати звіти про роботу персоналу у форматі Excel. Також, користувачі вебдодатка повинні мати можливість планування відпусток, адміністративний персонал повинен мати можливість підтвердити чи відхилити відпустку

підлеглого. Для зручного користування, вебдодаток має містити календар, який буде містити інформації про робочий план, події та інше.

Чат-бот має містити функціонал для перегляду розкладу занять для викладачів та студентів. Це може бути розклад занять на конкретний день або повний розклад. До того ж, має бути наявною можливість увімкнення сповіщень про заняття, які будуть приходити за 5 хвилин до початку заняття. Також має бути наявною можливість перегляду розкладу сесії, журналу успішності, розкладу подій та заходів в університеті та іншої корисної інформації.

1.2 Огляд й аналіз аналогів

Аналіз аналогів є ключовим етапом у розробці будь-якого програмного забезпечення, оскільки він допомагає визначити сильні та слабкі сторони існуючих рішень, виявити потенційні можливості для інновацій та уникнути повторення типових помилок. При проведенні аналізу сервісів для обліку робочого часу працівників ми маємо враховувати такі важливі параметри як: основний функціонал, доступність цін, цільова аудиторія, можливість інтеграції з іншими додатками, платформи, на яких доступний сервіс та відгуки користувачів про нього. Важливість такого аналізу полягає у визначенні того, як подібні рішення імплементуються на практиці, які технології та підходи вони використовують, а також у виявленні потенційних недоліків, які можуть бути удосконалені у нашому проекті. При проведенні аналізу чат-ботів буде розглянуто чат-боти інших університетів країни для дослідження їх функціоналу та виділення переваг та недоліків.

Toggl Track [4] - це популярний інструмент для відстеження часу, який використовується різними компаніями та фрілансерами для підвищення продуктивності та прибутковості (рис. 1.1). Він пропонує функції, такі як відстеження проектів, управління завданнями, оцінки проектів, а також управління командою та даними. Користувачі відзначають його простоту використання та інтеграцію з іншими інструментами, що робить його ефективним для відстеження часу, особливо для малого бізнесу та фрілансерів.

Проте, Toggl Track має деякі недоліки. Користувачі вказують на складнощі з додаванням клієнтів, обмеження у звітуванні, особливо у безкоштовній версії, та проблеми з дизайном інтерфейсу. Інші недоліки включають проблеми з розширеннями для браузерів, складнощі з повторним використанням записів та високу вартість преміум-версії. Ці фактори можуть вплинути на користувацький досвід, особливо для тих, хто шукає більш сучасні функції або хто зіткнувся з технічними проблемами у використанні програми.

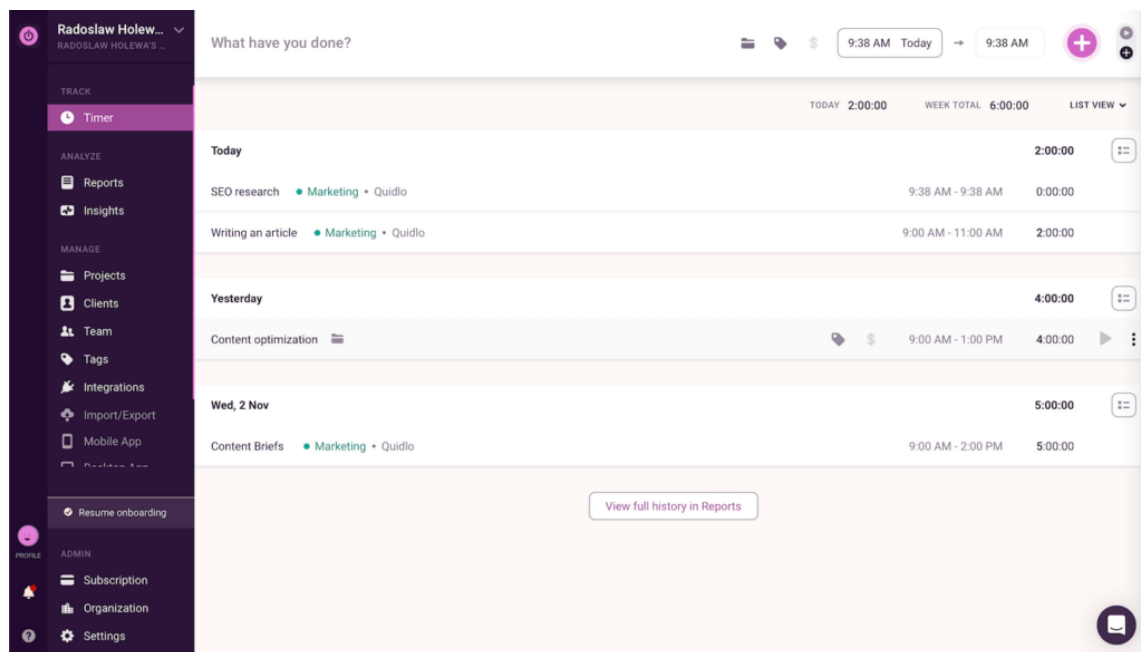


Рисунок 1.1 - Інтерфейс Toggl Track

Harvest [5] є популярним інструментом для відстеження часу, який ідеально підходить для різних видів бізнесу, включаючи фрілансерів, малі та середні компанії, а також великі підприємства (рис. 1.2). Його основні функції охоплюють відстеження часу, управління проектами, створення звітів. Однією з ключових особливостей Harvest є його здатність відстежувати витрати по проектах, дозволяючи користувачам отримувати детальні звіти та аналізувати витрати часу, що є важливим для планування майбутніх проектів.

Harvest має ряд недоліків. Одним з них є відсутність повної безкоштовної версії. Вебдодаток має тільки безкоштовний пробний період на місяць. Також, деякі користувачі відзначають, що знаходження необхідних функцій в інтерфейсі не завжди є інтуїтивно зрозумілим, і це може вимагати деякого часу для адаптації.

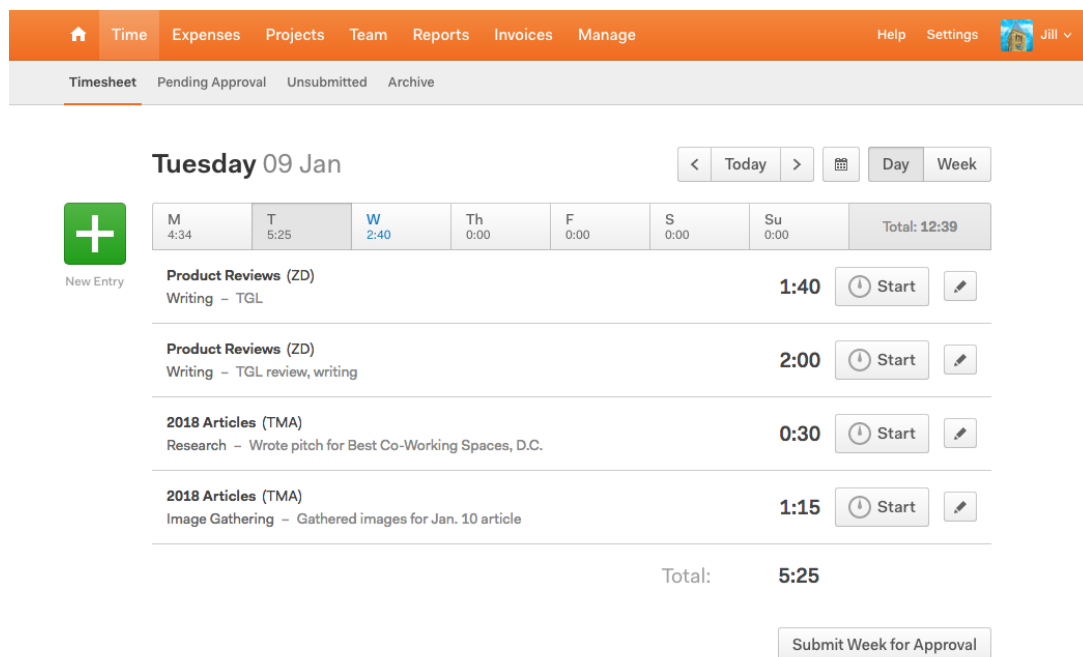


Рисунок 1.2 - Інтерфейс Harvest

Clockify [6] - це універсальне програмне забезпечення для відстеження часу, яке підходить для широкого кола користувачів, включаючи фрілансерів, малі та середні підприємства. Цей інструмент пропонує безкоштовну версію з базовими функціями, яка задовольняє основні потреби у відстеженні часу. Для більш складних завдань існують платні версії, які включають розширені можливості. Clockify дозволяє користувачам відстежувати робочі години по різних завданнях і проектах, створювати деталізовані звіти і аналізувати витрати часу, що є важливим для ефективного управління проектами та роботою. Тим не менш, деякі користувачі відзначають обмеження в мобільному додатку і побажання до більш розвинутих функцій і інтеграцій, особливо в плані управління проектами та клієнтами.

Загалом, Clockify оцінюється користувачами позитивно за простоту використання та ефективність у відстеженні часу, хоча деякі зазначають складнощі при навігації та розумінні деяких функцій на початкових етапах користування.

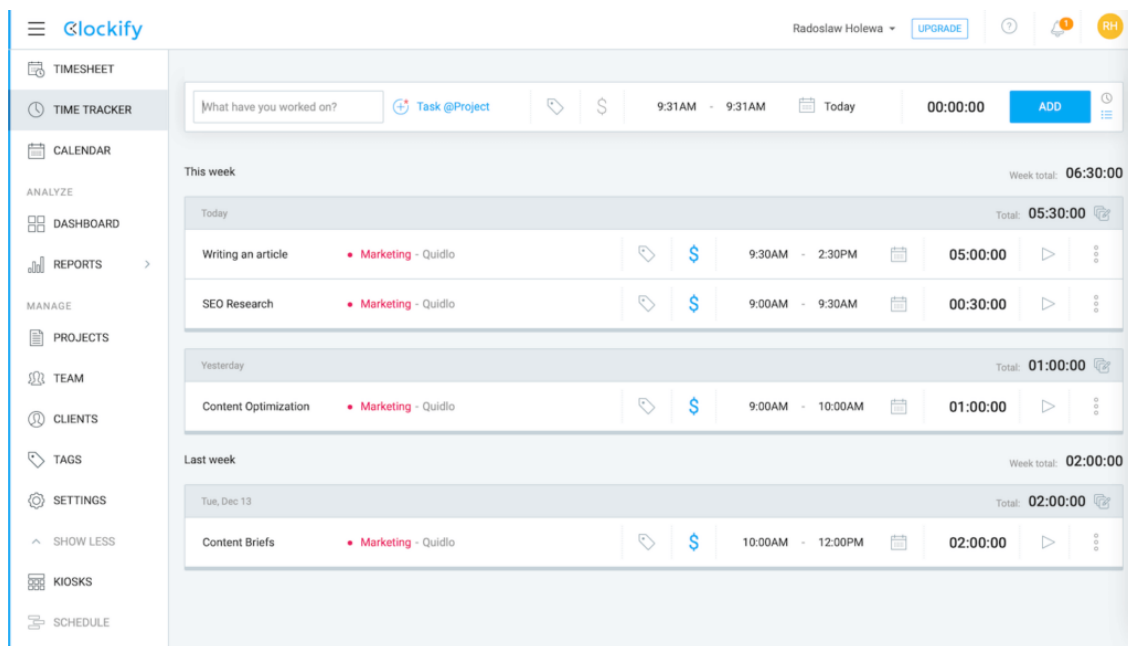


Рисунок 1.3 - Інтерфейс Clockify

RescueTime [7] - це програма для відстеження часу, яка спрямована на підвищення продуктивності та фокусу. Основною особливістю RescueTime є автоматичне відстеження активностей користувача на комп'ютері або мобільному пристрої. Ця програма надає докладні звіти про те, як використовується час, дозволяючи користувачам бачити, скільки часу вони проводять на різних вебсайтах, програмах та документах. RescueTime також пропонує функції, які допомагають зменшити кількість відволікаючих факторів, наприклад, блокування доступу до неефективних сайтів під час робочого часу.

Проте, користувачі зазначають певні недоліки RescueTime. Наприклад, RescueTime не завжди може точно розрізнити браузерні вкладки або ідентифікувати певні типи діяльності, що може спотворити аналіз продуктивності. Також, безкоштовна версія RescueTime має значно обмежені можливості порівняно з платною версією. Загалом, варто зазначити, що зважаючи на специфіку роботи працівників університету, особливо в умовах очного навчання, цей застосунок може бути малоефективним.

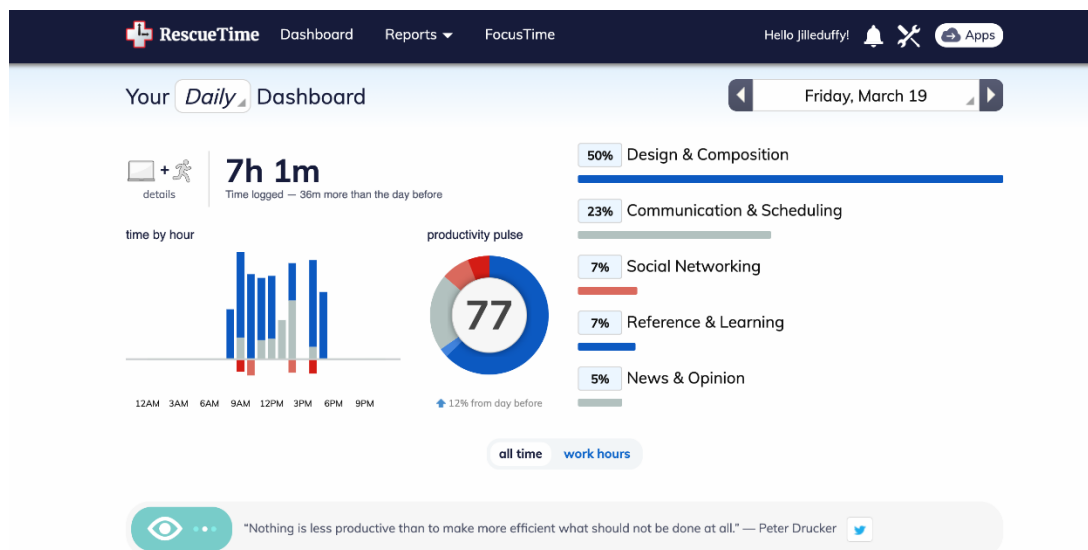


Рисунок 1.4 - Інтерфейс додатку RescueTime

TimeCamp [8] є програмним забезпеченням для відстеження часу, яке розроблено з метою допомогти користувачам краще розуміти та оптимізувати свій робочий час (рис. 1.5). Програма надає функції для автоматичного відстеження часу, здатну визначати, скільки часу витрачається на різні завдання, проекти чи вебсайти. Основною метою TimeCamp є поліпшення продуктивності шляхом надання користувачам детальної інформації про їхні робочі звички.

Загалом, відгуки про TimeCamp є позитивними, хоча деякі вказують на недоліки з роботою додатку, який може бути обмеженим у функціональності, не дозволяючи користувачам виконувати деякі дії, що вимагає перезавантаження додатку. Деякі користувачі повідомляють про технічні проблеми, такі як випадкові збої програми або втрата записів часу.

Використання TimeCamp в університетському середовищі може мати певні обмеження та виклики. Насамперед, TimeCamp зосереджений на відстеженні часу для конкретних проектів та завдань, що може не відповідати гнучкості та різноманітності завдань, з якими стикаються працівники університету.

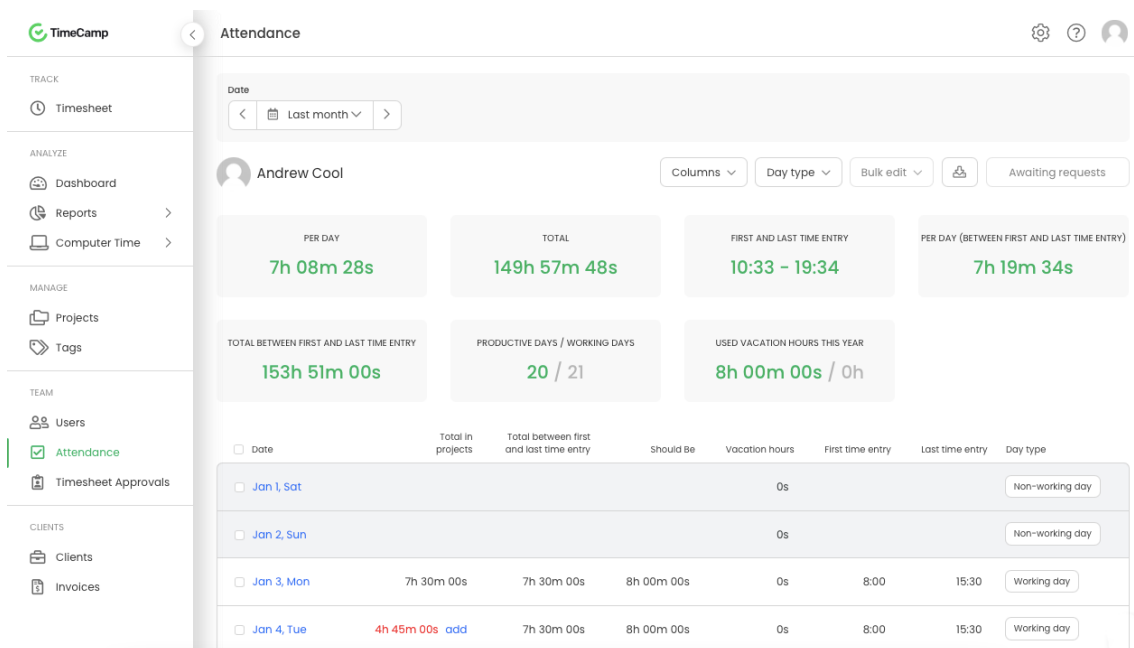


Рисунок 1.5 - Інтерфейс додатку TimeCamp

Asana [9] є популярним інструментом управління проектами, який використовується для організації, виконання та моніторингу процесу розвитку проектів. Це включає встановлення цілей проекту, делегування роботи членам команди та моніторинг розвитку проекту. Особливість Asana полягає в її інтуїтивному дизайні, простоті використання та комплексному наборі функцій, які роблять її ідеальним вибором для швидких проектів порівняно з іншими більш складними та функціонально багатими програмами.

Що стосується відстеження часу в Asana, програма не має вбудованої функції відстеження часу, але існує кілька способів відстежувати час у Asana, які включають інтеграцію з сторонніми інструментами відстеження часу або використання функції "Custom Fields" Asana. Користувачі можуть використовувати вбудовані таймери, які дозволяють точно і зручно відстежувати час, витрачений на конкретне завдання. Також можливе ручне введення часу, витраченого на завдання або проекти в Asana, що підходить для ситуацій, коли реальний час відстеження не є необхідним або можливим.

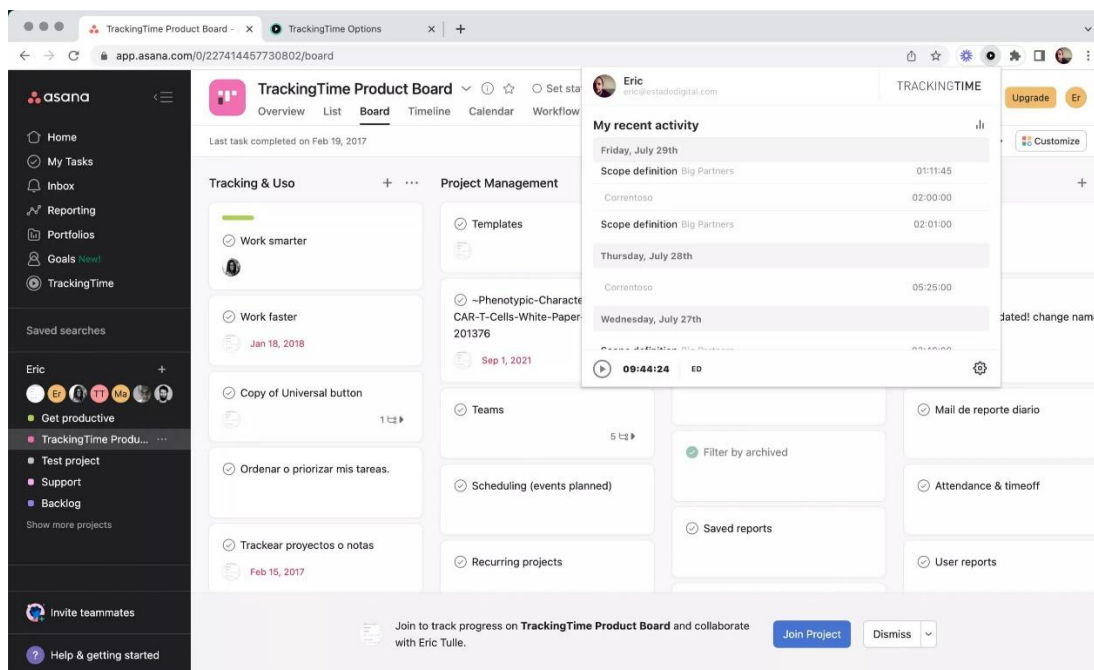


Рисунок 1.6 - Інтерфейс додатку Asana

Ознайомившись з вищезгаданими додатками можна зробити висновки про їх переваги та недоліки та скласти порівняльні таблиці (табл. 1.1 та табл. 1.2).

Таблиця 1.1

Порівняльна характеристика додатків для обліку робочого часу (Toggl Track, Harvest, Clockify)

Назва додатку	Toggl Track	Harvest	Clockify
Основні функції	Трекінг часу, звіти, аналітика	Трекінг часу, витрат, виставлення рахунків	Трекінг часу, створення звітів
Ціни	Безкоштовно, платні плани	Безкоштовний пробний період, платні плани	Безкоштовно, платні плани
Цільова аудиторія	Фрілансери, команди	Великі компанії	Фрілансери, малі команди
Інтеграція з іншими додатками	Обмежена	Так	Обмежена
Відгуки	Переважно позитивні	Змішані	Позитивні

Порівняльна характеристика додатків для обліку робочого часу
(RescueTime, TimeCamp, Asana)

Назва додатку	RescueTime	TimeCamp	Asana
Основні функції	Автоматичний трекінг, аналіз продуктивності	Автоматичний трекінг, облік проектів, виставлення рахунків	Управління проектами, трекінг часу через інтеграцію
Ціни	Безкоштовно, платні плани	Безкоштовно, платні плани	Безкоштовно, платні плани
Цільова аудиторія	Індивідуальні користувачі	Фрілансери, малі та середні команди	Команди, великі компанії
Інтеграція з іншими додатками	Обмежена	Обмежена	Обмежена
Відгуки	Позитивні	Позитивні	Змішані

Порівнявши існуючі додатки ми бачимо, що майже всі з них мають безкоштовні версії з обмеженим функціоналом. Також, більшість орієнтовані на індивідуального користувача або невелику команду, що не задовольняє нашу потребу використання вебдодатка великою кількістю користувачів.

Проведемо аналіз чат-ботів інших університетів. Для цього було знайдено чат-боти Житомирського державного університету імені Івана Франка, Університету Короля Данила та Національного університету «Полтавська політехніка імені Юрія Кондратюка»

Першим проаналізуємо чат-бот ЖДУ ім. Івана Франка [10]. Після початку роботи з ботом ми повинні ввести назву групи розклад якої ми хочемо побачити. Випадковим чином зі списку усіх груп було обрано одну та введено її назву боту, після чого відкрилось меню з можливістю перегляду розкладу на сьогодні,

завтра, тиждень (рис. 1.7). Було переглянуто розклад групи на сьогодні та на тиждень (рис. 1.8). Також наявна можливість переглянути розклад викладача за його ПІБ та інших груп за їх назвою.

Наступним проведемо дослідження роботи чат-бота Університету Короля Данила [11]. Початок роботи з ботом на рис. 1.9. Бот має можливість перегляду розкладу групи (рис. 1.10) або викладача, а також автоматичного розсилання щоранку розкладу на день.

Ще одним чат-ботом для аналізу було обрано чат-бот Національного університету «Полтавська політехніка імені Юрія Кондратюка» [12]. Особливістю цього боту є його захищеність від сторонніх користувачів, а саме потреба в авторизації за допомогою пошти університету. Роботу з цим чат-ботом зображено на рис. 1.11.

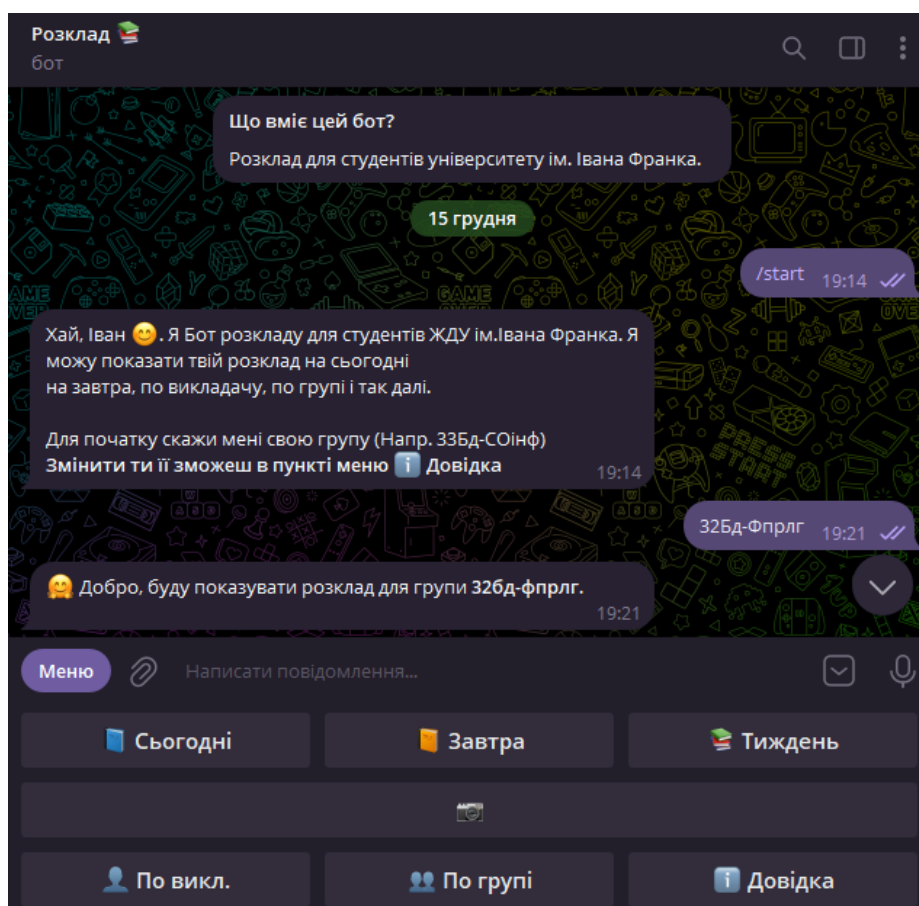


Рисунок 1.7 - Початок роботи з чат-ботом ЖДУ ім. Івана Франка

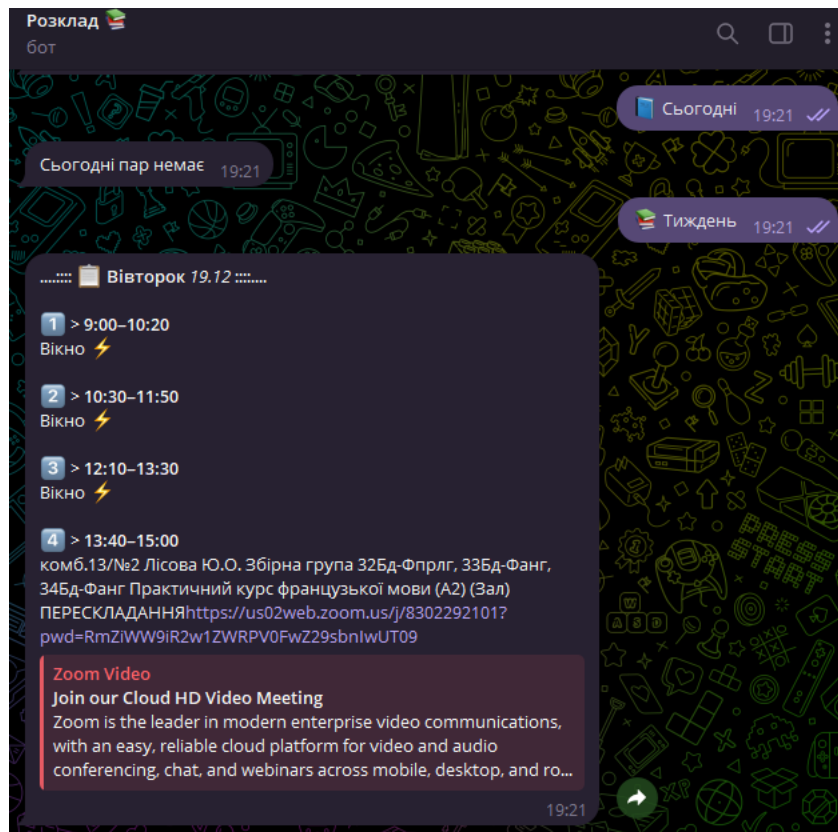


Рисунок 1.8 - Перегляд розкладу групи в чат-боті ЖДУ ім. Івана Франка

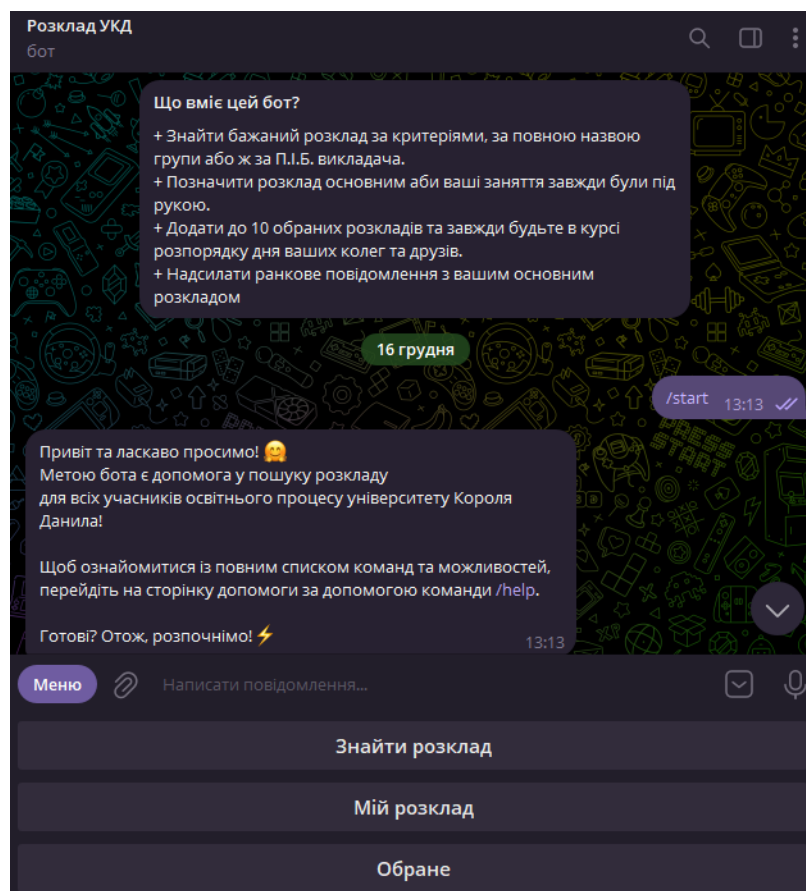


Рисунок 1.9 - Початок роботи з чат-ботом УКД

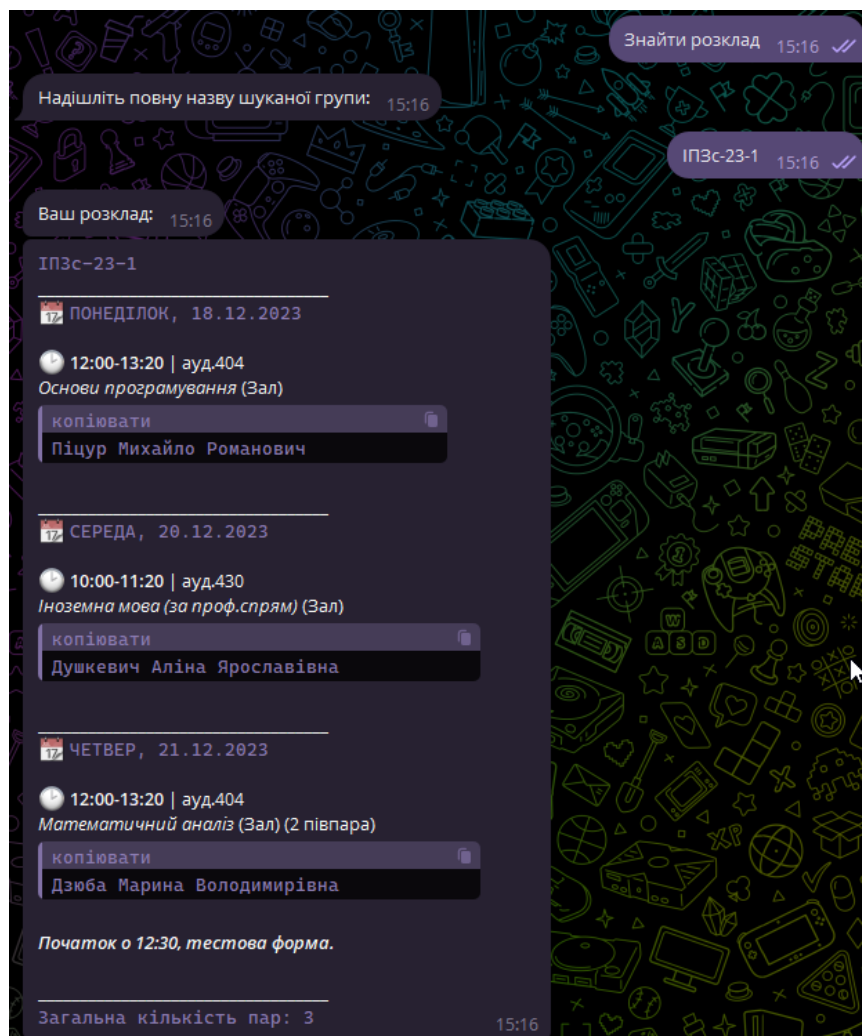


Рисунок 1.10 - Перегляд розкладу у чат-боті УКД

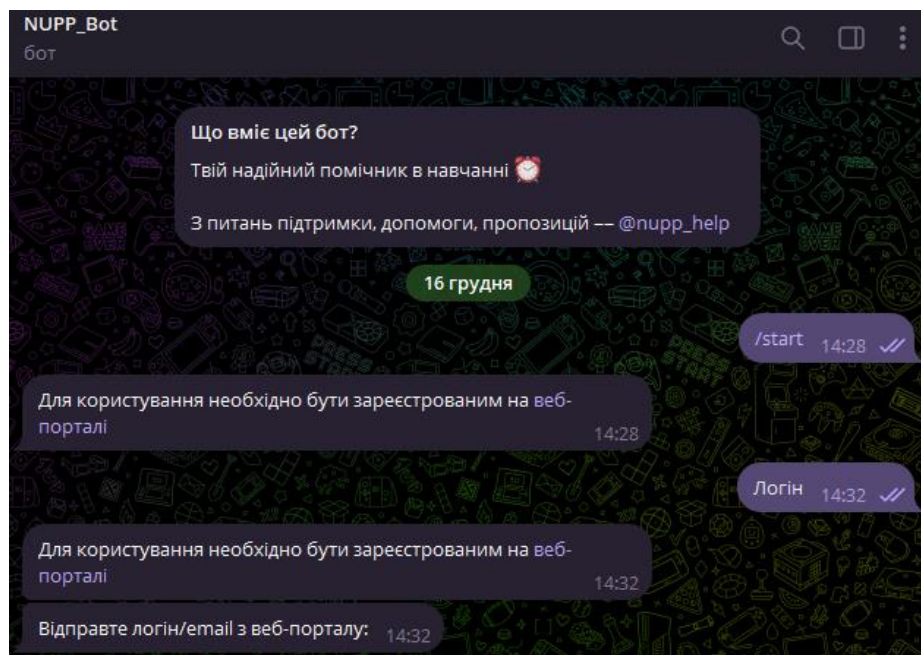


Рисунок 1.11 - Початок роботи з чат-ботом Національного університету
«Полтавська політехніка імені Юрія Кондратюка»

Ознайомившись з вищезгаданими чат-ботами можна зробити висновки про їх переваги та недоліки та скласти порівняльну таблицю (табл. 1.3).

Таблиця 1.3

Порівняльна характеристика чат-ботів

Назва чат-бота	Розклад (ЖДУ ім. Івана Франка)	Розклад УКД	NUPP_Bot
Функціонал	Перегляд розкладу групи за її назвою та розкладу викладачів	Перегляд розкладу групи за її назвою, надсилання щоденних автоматичних повідомлень з розкладом	-
Зрозумілість інтерфейсу	Інтуїтивний	Стандартний	-
Швидкість роботи	Середня	Середня	-
Захищеність	Відсутня	Відсутня	Для авторизації потрібна пошта університету

Порівнявши існуючі чат-боти ми бачимо, що тільки один з них є закритим для зовнішніх користувачів та доступний тільки для студентів та працівників університету. Також, усі з них мають можливість перегляду розкладу. Один з досліджених чат-ботів має можливість автоматичного розсилання розкладу.

1.3 Обґрунтування вибору інструментальних засобів для розробки

Враховуючи особливості розроблюваного сервісу, а саме те, що він складатиметься з вебдодатка для обліку робочого часу працівників університету

та чат-бота в Telegram, інструментами, які можуть підійти для розробки обох складових частин сервісу є: Node.js, Ruby, Python, PHP, ASP.NET. Розглянемо кожен з них детальніше.

Node.js - це потужне та гнучке середовище виконання для JavaScript, що дозволяє розробникам використовувати JS для створення серверної частини додатків. Node.js відрізняється високою продуктивністю та швидкістю виконання завдяки подієво-орієнтованій архітектурі. Це середовище підходить для розробки масштабованих мережових додатків, таких як вебсервери, API, а також для реалізації різних інструментів та утиліт. Завдяки своїй ефективності та гнучкості, Node.js швидко завоював популярність серед розробників по всьому світу [13].

Ruby - це високорівнева, інтерпретована мова програмування. Вона славиться своєю елегантністю та чистотою коду, надаючи особливу увагу продуктивності розробника та зрозумілості коду. Ruby підтримує різні парадигми програмування, включаючи об'єктно-орієнтовану, функціональну, імперативну та відбивну. Вона відома своєю гнучкістю, оскільки розробники можуть легко модифікувати її частини за своїми потребами. Ruby також активно використовується для створення вебдодатків, сценаріїв автоматизації, аналізу даних та інших завдань, де потрібна швидка розробка та легкість обслуговування коду. Для розробки вебдодатка та чат боту доцільно використовувати фреймворк Ruby on Rails.

Ruby on Rails, зазвичай відомий як Rails, є одним з найпопулярніших фреймворків для веброзробки на Ruby. Rails використовує патерн Model-View-Controller (MVC) для організації коду та спрощення процесу розробки. Особливістю Rails є його філософія "Convention over Configuration" та принцип "Don't Repeat Yourself" (DRY), які знижують кількість необхідного коду та збільшують ефективність розробки. Rails також надає багатий набір вбудованих функцій для роботи з базами даних, вебсервісами та іншими завданнями, що робить його ідеальним вибором для швидкого створення надійних та легко масштабованих вебдодатків [14].

Python - це гнучка та могутня мова програмування, яка завоювала популярність у різних областях, від веброзробки до науки про дані. Python має багато фреймворків, що дозволяють швидко та зручно створювати вебдодатки та інші сервіси. Одним з таких є Django [15]. Django є одним з найбільш використовуваних інструментів для створення вебдодатків. Його сила полягає в тому, що він дозволяє розробникам швидко створювати безпечні, надійні та масштабовані вебдодатки. Django слідує патерну MVC, що спрощує управління кодом та даними. Особливо привабливим Django робить його принцип "батарейки включені", що означає, що фреймворк вже має вбудовані рішення для багатьох стандартних задач веброзробки, як-от аутентифікація користувачів, адміністрування та робота з базами даних. Це, разом з міцною спільнотою та обширною документацією, робить Django відмінним вибором для розробки комплексних вебсервісів, особливо для тих, хто вже знайомий з Python.

PHP є однією з найпопулярніших мов програмування для веброзробки, відомою своєю простотою, гнучкістю та широким використанням у створенні вебсайтів та вебдодатків [16]. Як серверна мова, PHP дає змогу розробникам створювати динамічний контент, що взаємодіє з базами даних. Ця мова є основою для багатьох популярних систем управління контентом, таких як WordPress, Drupal та Joomla, а також використовується великими вебплатформами, такими як Facebook. Основні переваги PHP включають легкість інтеграції з HTML та іншими вебтехнологіями, велику кількість доступних бібліотек та фреймворків, таких як Laravel і Symfony, які спрощують розробку та надають додаткові можливості для безпеки, тестування та управління кодом. Завдяки своїй доступності та широкому спектру використання, PHP залишається одним найпопулярніших виборів для розробників, які бажають створювати надійні та масштабовані вебдодатки.

ASP.NET - це потужний веб-фреймворк, створений Microsoft для розробки динамічних вебсайтів, додатків та сервісів. Як частина платформи .NET, ASP.NET дозволяє розробникам створювати вебдодатки, використовуючи мови програмування, такі як C# та VB.NET. Він підтримує розробку за допомогою

патерну MVC, який сприяє чіткому розділенню логіки додатку, інтерфейсу користувача та управління даними. Однією з ключових переваг ASP.NET є його інтеграція з екосистемою Microsoft, що забезпечує легкий доступ до широкого спектру інструментів та сервісів, таких як Azure, для хмарних рішень, і Visual Studio для розробки [17].

ASP.NET відомий своєю високою продуктивністю, масштабованістю та безпекою, роблячи його ідеальним вибором для корпоративних вебдодатків і великих систем. Він також пропонує широкі можливості для автоматизованого тестування та підтримки, що є важливим для підтримки високої якості коду. Завдяки використанню знайомих мов програмування та звичних парадигм, ASP.NET є відмінним вибором для розробників, які шукають надійну, високопродуктивну платформу для створення вебдодатків.

Виділимо основні переваги й недоліки кожної з запропонованих технологій.

1. Node.js:

- Переваги: висока продуктивність завдяки асинхронній обробці подій, широка підтримка бібліотек та пакетів через npm.
- Недоліки: може бути менш зручним для розробки більших, складних за структурою застосунків.

2. Ruby on Rails:

- Переваги: швидке прототипування, конвенція над конфігурацією, що спрощує розробку.
- Недоліки: менша продуктивність порівняно з іншими мовами програмування.

3. Django (Python):

- Переваги: багато вбудованих функцій, безпека, широко використовується для веброботи.
- Недоліки: може бути менш гнучким у деяких сценаріях порівняно з іншими технологіями розробки.

4. PHP:

- Переваги: легко інтегрується з HTML, широко використовується для веброзробки, велика спільнота та підтримка.

- Недоліки: менша продуктивність та шкальованість порівняно з ASP.NET, може бути важчим у підтримці для великих проектів.

5. ASP.NET:

- Переваги: висока продуктивність, різноманітні вбудовані можливості безпеки, Сильна інтеграція з іншими сервісами Microsoft, використання C# забезпечує строгу типізацію, об'єктно-орієнтоване програмування та сучасні можливості мови.

- Недоліки: залежність від сервісів Microsoft, для тих, хто не знайомий з екосистемою .NET та C#, вивчення ASP.NET може бути складнішим.

Враховуючи усі ці аспекти для розробки сервісу було обрано ASP.NET. Значну роль зіграла висока продуктивність та ступінь безпеки, яку надає фреймворк, оскільки захист даних та систем є пріоритетним для будь-якого вебдодатка. Інтеграція з іншими продуктами Microsoft та сервісами, такими як Azure, забезпечить додаткову зручність та ефективність у роботі. Використання хмарних сервісів забезпечить зручність в розгортанні вебдодатка та чат-бота.

Для розробки клієнтської частини вебдодатка було обрано Java Script фреймворки React.js [18] та Redux [19]. Вибір цих технологій для розробки клієнтської частини додатку, що базується на ASP.NET, має низку переваг. Перш за все, React є однією з найпопулярніших JavaScript бібліотек для побудови інтерфейсів, яка забезпечує високу продуктивність та гнучкість завдяки своєму віртуальному DOM. Це дозволяє створювати швидкі та зручні веб-інтерфейси, що є важливим для забезпечення хорошого користувацького досвіду. Крім того, React підтримує компонентний підхід, що сприяє кращій організації коду, його перевикористанню та тестуванню. Також, React легко інтегрується з іншими бібліотеками та фреймворками, що робить його ідеальним вибором для проектів, які можуть розвиватися або змінюватися з часом [1].

Redux доповнює React, надаючи єдиний "store" для всього додатку, що полегшує управління даними у великих та складних додатках. Він забезпечує

передбачуваність та управління станом, які є критично важливими для динамічних додатків, де стан може змінюватися часто. З Redux, розробники можуть легко відстежувати, як стан змінюється у відповідь на дії користувачів, що спрощує дебагінг та тестування. Також, Redux допомагає уникнути "prop drilling" (прокидання пропсів через кілька рівнів компонентів), що робить структуру додатку більш організованою та зрозумілою. В поєднанні з ASP.NET, який забезпечує міцну серверну основу, React та Redux створюють потужний стек технологій для розробки сучасних вебдодатків.

1.4 Вибір архітектури

Об'єктно-орієнтоване програмування (ООП) в ASP.NET є ключовою концепцією, яка дозволяє розробникам створювати модульні, масштабовані та легко підтримувані вебдодатки. Використання ООП в ASP.NET включає створення класів та об'єктів, що відображають реальні або абстрактні сутності, та їх взаємодію через наслідування, інкапсуляцію, поліморфізм та абстракцію [20]. Це дозволяє розробникам керувати складністю вебдодатків, розділяючи їх на менші, взаємодіючі компоненти, що спрощує розробку та тестування.

MVC у контексті ASP.NET є популярним шаблоном проектування, що розділяє додаток на три основні компоненти: Модель, Вид і Контролер [21]. Модель відповідає за логіку даних і бізнес-правила, Вид відображає інформацію (інтерфейс користувача), а Контролер обробляє вхідні запити, маніпулює моделями та оновлює види. MVC сприяє чіткому розділенню відповідальності, що полегшує управління кодом, його розширення та тестування, а також дозволяє різним командам працювати над різними аспектами додатку одночасно.

Архітектура додатків, що використовують React у поєднанні з ASP.NET, часто базується на моделі клієнт-сервер, де ASP.NET відповідає за серверну частину (бекенд), а React - за клієнтську (фронтенд). Особливістю такої архітектури є чітке розділення відповідальностей: бекенд займається обробкою даних, взаємодією з базою даних, аутентифікацією та авторизацією, тоді як фронтенд відповідає за інтерфейс користувача та взаємодію з користувачем. Це

розділення сприяє модульності, полегшує тестування та підтримку коду.

Використання MVC патерну в архітектурі таких додатків є доцільним з кількох причин:

1. Розділення відповідальностей: MVC дозволяє розділити логіку додатку на три основні компоненти: модель (Model), відображення (View) та контролер (Controller). Модель відповідає за дані та бізнес-логіку, відображення - за представлення даних, а контролер - за обробку користувацьких запитів та взаємодію між моделлю та відображенням. Це сприяє легшій розробці, тестуванню та підтримці коду.

2. Гнучкість та масштабованість: MVC підтримує гнучкість, оскільки дозволяє розробникам змінювати відображення без зміни моделі, і навпаки. Це також полегшує масштабування додатку, оскільки можна окремо оптимізувати різні частини додатку.

3. Спрощення співпраці та розвитку: у великих командах, де різні розробники або команди працюють над різними частинами додатку, MVC допомагає уникнути конфліктів та спрощує розвиток продукту, оскільки кожен компонент може розвиватися незалежно.

4. Легкість інтеграції з іншими патернами: MVC легко інтегрується з іншими патернами та архітектурними підходами, такими як RESTful API, що часто використовуються в ASP.NET для розробки API.

5. Підтримка тестування: розділення відповідальностей у MVC сприяє тому, що різні компоненти системи можна тестувати окремо, що є особливо важливим для забезпечення якості та надійності великих та складних додатків.

Отже, використання MVC патерну в архітектурі додатків, які включають ASP.NET та React, є доцільним, оскільки воно сприяє чіткій організації коду, забезпечує високий рівень модульності та спрощує розвиток та підтримку додатку.

Висновки до першого розділу

В ході виконання першого розділу було виконано детальну постановку

задачі для розробки сервісу для автоматизації розсилання повідомлень та обліку робочого часу працівників університету. Розроблюваний сервіс буде складатися з двох складових частин – чат-бота для автоматичного розсилання повідомлень та вебдодатка для обліку робочого часу працівників університету. Вебдодаток буде зручним та відповідати сучасним вимогам, тобто таким, що матиме високу швидкість роботи та зручний та зрозумілий інтерфейс. Чат-бот матиме зрозуміле меню та широкий функціонал.

Провівши аналіз аналогів вебдодатків для обліку робочого часу та чат-ботів інших університетів країни та виділивши переваги і недоліки розглянутого ПЗ, було сформовано критерії до розроблюваної системи, такі як: швидкодія, інтуїтивна зрозумілість інтерфейсу, можливість автоматичного та ручного обліку часу, створення робочих планів, планування відпусток та інше. Чат-бот має бути доступним тільки для працівників та студентів університету, мати можливість перегляду розкладу та надсилання автоматичних сповіщень про розклад занять або заходів.

Для розробки системи було обрано такі інструментальні засоби: серверна частина вебдодатка для обліку робочого часу та чат-бота буде розроблятися засобами мови програмування C#, а саме фреймворку ASP.NET. Клієнтська частина вебдодатка розроблятиметься за допомогою Java Script фреймворків React.js та Redux.

РОЗДІЛ 2 ПРОЕКТУВАННЯ СЕРВІСУ ДЛЯ АВТОМАТИЗАЦІЇ РОЗСИЛАННЯ ПОВІДОМЛЕНЬ ТА ОБЛІКУ РОБОЧОГО ЧАСУ ПРАЦІВНИКІВ УНІВЕРСИТЕТУ

2.1 Розробка структури бази даних

База даних (БД) є критично важливою складовою будь-якої сучасної інформаційної системи, особливо у контексті вебдодатків та інших вебсервісів. Вона служить основою для зберігання, управління та доступу до даних, що є невід'ємною частиною функціональності системи. У даному випадку, для забезпечення надійності та ефективності обробки даних, обрано Microsoft SQL Server, що є широко визнаним рішенням для корпоративних додатків. Ця система управління базами даних не тільки забезпечує високу продуктивність та безпеку, але й гарантує масштабованість та надійність, що є ключовими для нашого проекту.

Для ефективної взаємодії з базою даних у даній системі використовувався Entity Framework (EF), ORM (Object-Relational Mapping) інструмент, що спрощує роботу з даними [22]. EF дозволяє взаємодіяти з базою даних через вищий рівень абстракції, використовуючи об'єктно-орієнтовані принципи, що істотно підвищує продуктивність розробки. Крім того, було інтегровано AutoMapper у систему для автоматизації процесу перетворення даних між об'єктами домену та об'єктами передачі даних. Це значно знижує кількість коду, потрібного для відображення одних об'єктів в інші, тим самим покращуючи читабельність коду та спрощуючи процес обслуговування.

Також, враховуючи особливості розроблюваної системи, матимемо дві бази даних. Одна для вебдодатка для обліку робочого часу працівників, інша для чат-бота. Враховуючи особливості архітектури вебдодатка, а саме серверну частину на ASP.NET та клієнтську на React та Redux, для взаємодії між ними та базою даних використовується GraphQL [20]. GraphQL пропонує ряд переваг у цьому контексті, особливо з огляду на його гнучкість та потужні можливості запитів [2].

З серверної сторони, ASP.NET ефективно обробляє запити GraphQL, що дозволяє нам точно визначати, які дані потрібні клієнтам, та забезпечувати їх у відповідності до цих запитів. Це виключає необхідність пересилати зайві дані, знижуючи тим самим навантаження на мережу та підвищуючи продуктивність застосунку.

З клієнтського боку, React разом із Redux створює інтуїтивно зрозумілий та ефективний інтерфейс, який інтегрований із GraphQL. Клієнти можуть формувати запити, що точно відповідають їхнім потребам у даних, що значно підвищує гнучкість та ефективність користувацького досвіду. Redux допомагає управляти станом додатку, забезпечуючи консистентність даних на клієнтській стороні та їхню синхронізацію з сервером.

У цілому, використання GraphQL у поєднанні з ASP.NET та React/Redux є сучасним підходом до розробки вебдодатків, що забезпечує високу продуктивність, гнучкість та ефективність у взаємодії між серверною та клієнтською частинами.

Також, для взаємодії з базою даних використовується патерн «Фабрика».

Factory Pattern, або патерн "Фабрика", — це один з найвідоміших патернів проектування у програмуванні. Він використовується для створення об'єктів без специфікації точного класу об'єкта, який буде створено. Цей патерн часто використовується у ситуаціях, коли система повинна залишатися незалежною від способу створення, композиції та представлення своїх об'єктів. Використання Factory Pattern у контексті взаємодії з базами даних може значно підвищити гнучкість, масштабованість та тестованість програмного забезпечення, особливо у складних системах з різноманітними вимогами до зберігання та обробки даних.

Схему бази даних чат-бота для перегляду розкладу та автоматичного надсилання повідомлень зображено на рис. 2.1.

Схему бази даних вебдодатка для обліку робочого часу працівників університету зображено на рис. 2.2.

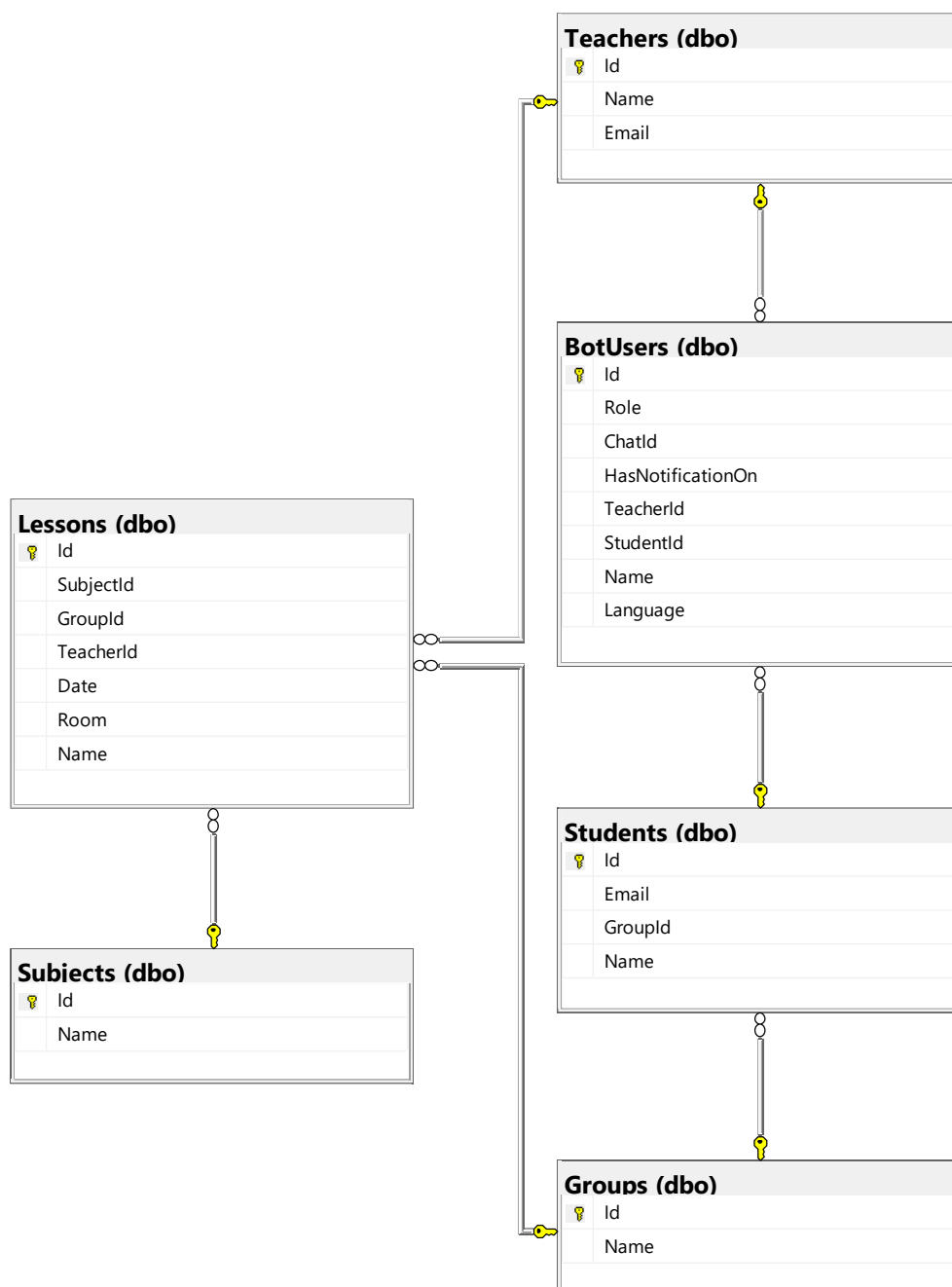


Рисунок 2.1 - Схема БД чат-бота

БД чат-бота включає різні таблиці та зв'язки між ними. Таблиця "Teachers" зберігає ідентифікатори, імена та електронні адреси викладачів. "BotUsers" містить інформацію про ролі користувачів та налаштування сповіщень, а також зв'язки з викладачами та студентами. Таблиця "Students" включає ідентифікатори, електронні адреси та імена студентів, а також зв'язок з групами. "Groups" просто містять ідентифікатори та імена груп. "Lessons" вказують на предмет, групу, викладача, дату, аудиторію та назву заняття. Остання таблиця "Subjects" зберігає ідентифікатори та імена предметів, що викладаються.

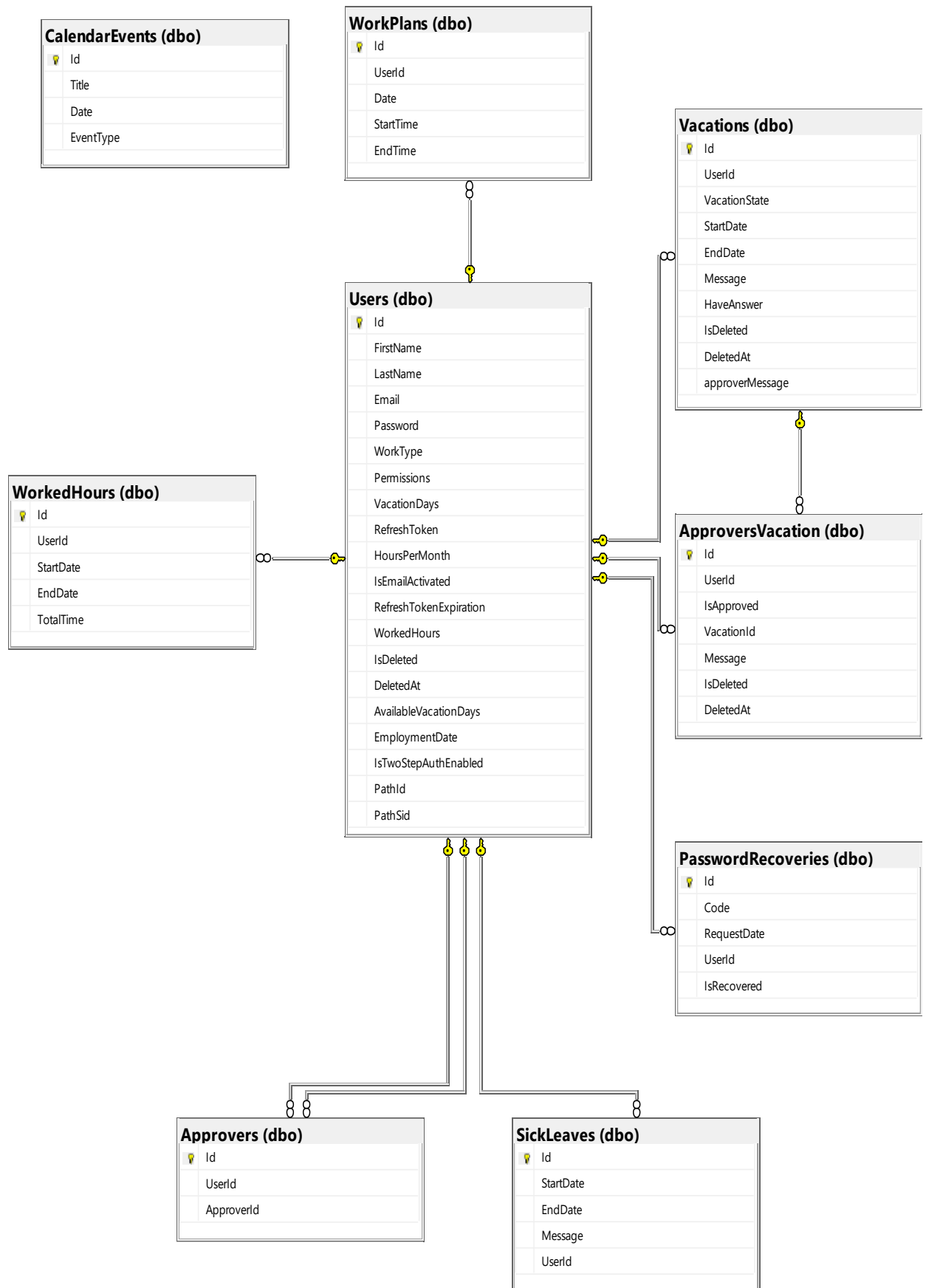


Рисунок 2.2 - Схема БД вебдодатка

БД містить таблиці для календарних подій, робочих планів, реєстрації відпрацьованих годин, керування відпустками та хворобами, а також для

відновлення паролів. Таблиця "Users" є центральною і пов'язана з багатьма іншими, що містить особисту інформацію користувачів, їхні робочі години, статус відпусток та інше. "WorkPlans" та "WorkedHours" відслідковують планування та відпрацювання часу працівниками, в той час як "Vacations" та "SickLeaves" управляють записами про відпустки та лікарняні. Додаткові таблиці, як-от "Approvers" та "PasswordRecoveries", забезпечують функціональність затвердження відпусток та відновлення доступу до системи.

Розглянемо приклад взаємодії між клієнтською та серверною частиною, враховуючи описані вище особливості роботи з базою даних для розроблюваного вебдодатка на основі входу користувача в систему.

```
Field<string>("login")
    .Argument<UserLoginInputType>("user")
    .ResolveAsync(async ctx =>
    {
        var args = ctx.GetArgument<User>("user");
        var searchUser = await uow.GenericRepository<User>()
            .FindAsync(u => u.Email == args.Email);
        if (searchUser == null || searchUser.IsDeleted)
        {
            throw new QueryError(Error.ERR_WRONG_CREDENTIALS);
        }
        if (!searchUser.IsEmailActivated) throw new
QueryError(Error.ERR_INACTIVE_ACCOUNT);
        var authService = ctx.RequestServices?.GetRequiredService<Authenticate>();
        return await AuthorizeConfirmedEmailAsync(searchUser, args.Password,
authService!);
    });
```

Функція "login" у GraphQL схемі - це ключова точка для аутентифікації користувачів у системі. Роботу функції "login" можна розбити на кілька основних етапів:

1. Прийом аргументів: функція "login" приймає об'єкт UserLoginInputType як аргумент. Цей тип даних визначений в класі

UserLoginInputType, який містить два поля: Email та Password. Обидва поля призначені для введення даних користувача: електронної пошти та пароля відповідно.

2. Пошук користувача: у функції виконується асинхронний запит до бази даних для пошуку користувача за електронною адресою. Це робиться за допомогою методу FindAsync у репозиторії GenericRepository<User>.

3. Перевірка користувача: після того, як користувач знайдений, функція перевіряє, чи існує такий користувач і чи не видалений він (перевірка `searchUser == null || searchUser.IsDeleted`). Якщо користувач не існує або видалений, викидається помилка з кодом `Error.ERR_WRONG_CREDENTIALS`.

4. Активація електронної пошти: якщо користувач існує, то перевіряється, чи активована його електронна пошта (`searchUser.IsEmailActivated`). Якщо пошта не активована, викидається помилка `Error.ERR_INACTIVE_ACCOUNT`.

5. Аутентифікація та створення токена: після успішної перевірки, функція звертається до сервісу аутентифікації `Authenticate`, що знаходиться у контексті запиту. Вона передає знайденого користувача та введений пароль у функцію `AuthorizeConfirmedEmailAsync`, яка виконує подальшу верифікацію та генерує токен доступу для користувача.

Таким чином, функція "login" тісно інтегрована з моделлю `User`, використовуючи її поля для аутентифікації та ідентифікації користувачів у системі. Лістинг коду моделі `User` наведено у додатку А.

Далі, для відправлення даних для входу (електронна пошта та пароль) від React-компоненту до сервера, використовується Redux функція `UserLoginQuery`.

```
export function UserLoginQuery(userData: { email: string, password: string }) {
  return AjaxQuery<{ userQuery: { login: string } }>(
    "query Login($user: UserLoginInputType!) {userQuery {login(user: $user)})",
    { user: userData }
  );
}
```

Функція приймає об'єкт `userData`, який містить два поля: `email` і `password`.

Ці дані зазвичай збираються з форми логіну на фронтенді. середині функції використовується GraphQL запит, який визначається як "query Login(\$user: UserLoginInputType!) {userQuery {login(user: \$user)}}". Цей запит відправляє дані користувача до GraphQL.

Цей процес забезпечує безпечний спосіб входу в систему, де користувачі можуть здійснити вхід у вебдодаток за допомогою своєї електронної пошти та пароля, а система перевіряє їхні дані та надає доступ за умови успішної перевірки.

2.2 Проектування та реалізація алгоритмів роботи сервісу для автоматизації розсилання повідомлень

Як платформу для реалізації чат-бота було обрано месенджер Telegram. Цей вибір зумовлено декількома ключовими факторами, такими як:

1. Широка функціональність API: Telegram пропонує один з найбільш потужних і гнучких API серед месенджерів. Це дозволяє розробникам створювати складні боти з різноманітними функціями.
2. Популярність серед розробників: завдяки своїй відкритості та легкості інтеграції, Telegram став популярним вибором серед розробників, особливо в сфері стартапів і технологій.
3. Безпека та приватність: Telegram відомий своїм акцентом на безпеку та приватність, що є важливим фактором для багатьох користувачів та розробників.
4. Велика користувацька база: месенджер має мільйони активних користувачів по всьому світу, включаючи Україну, де в умовах війни Telegram став одним з найрозповсюдженіших джерел інформації та способів комунікації.
5. Мультиплатформенність: Telegram доступний на різних платформах, включаючи iOS, Android, веб-версії, що робить його зручним для широкого спектру користувачів.
6. Цільова аудиторія: Telegram здебільшого популярний у молодого покоління, яке є основою користувачів розроблюваного чат-бота. Інші

месенджери, такі як WhatsApp та Viber, націлені на більш дорослу аудиторію користувачів.

Починається розробка чат-бота в Telegram з його реєстрації за допомогою чат-бота «BotFather» (рис. 2.3).

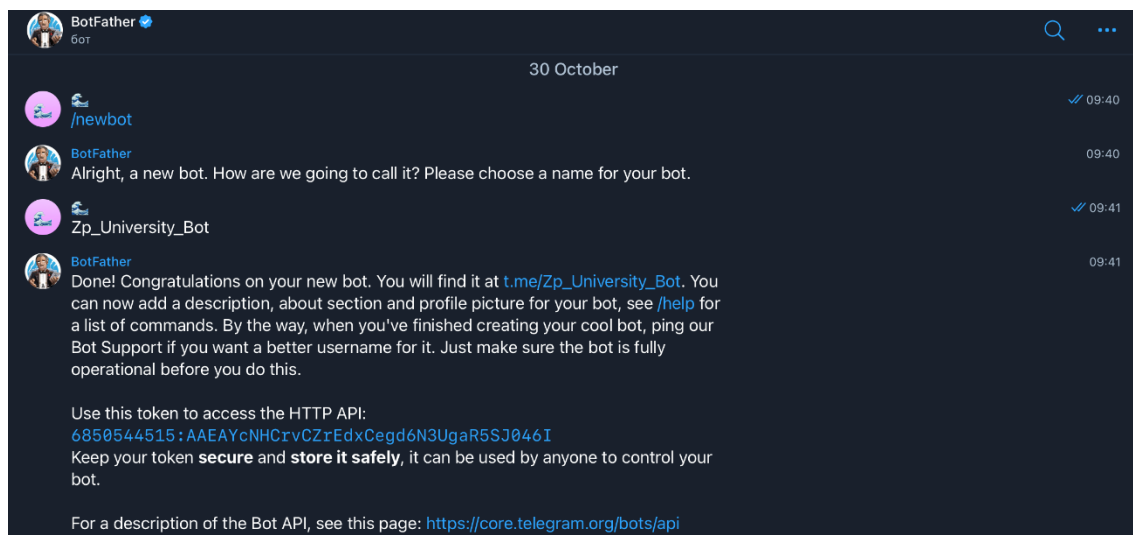


Рисунок 2.3 - Реєстрація чат-бота в Telegram

Далі, за допомогою команди «/setcommands» відбувається реєстрація команд, які міститиме чат-бот.

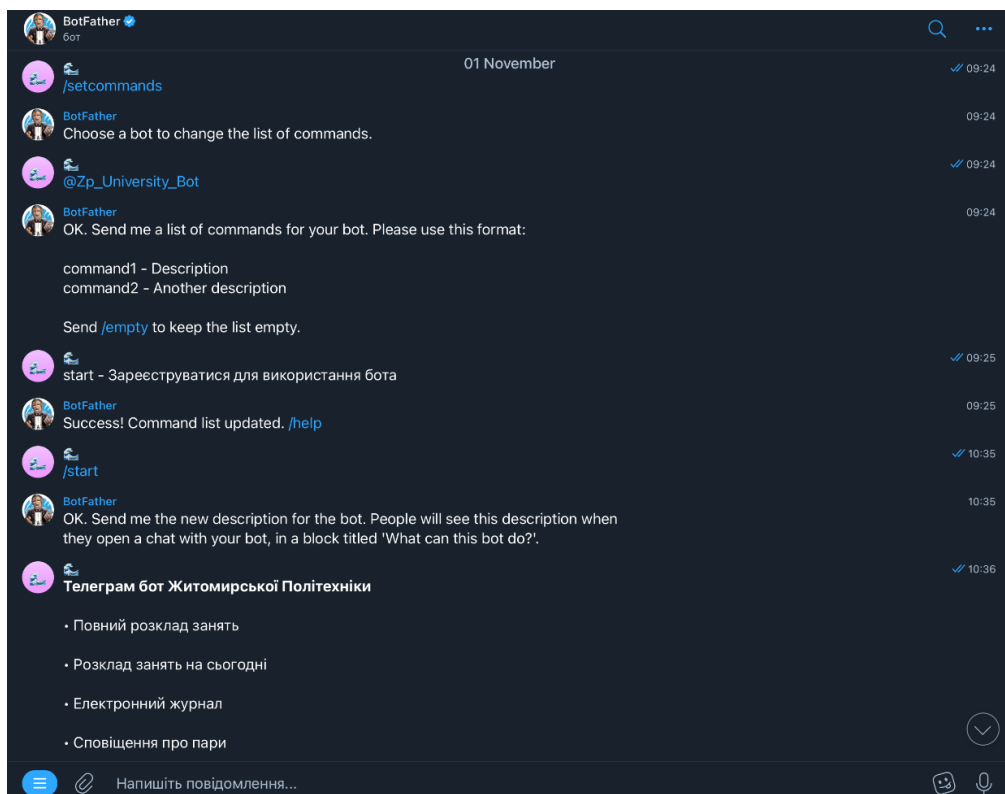


Рисунок 2.4 - Реєстрація команди чат-бота

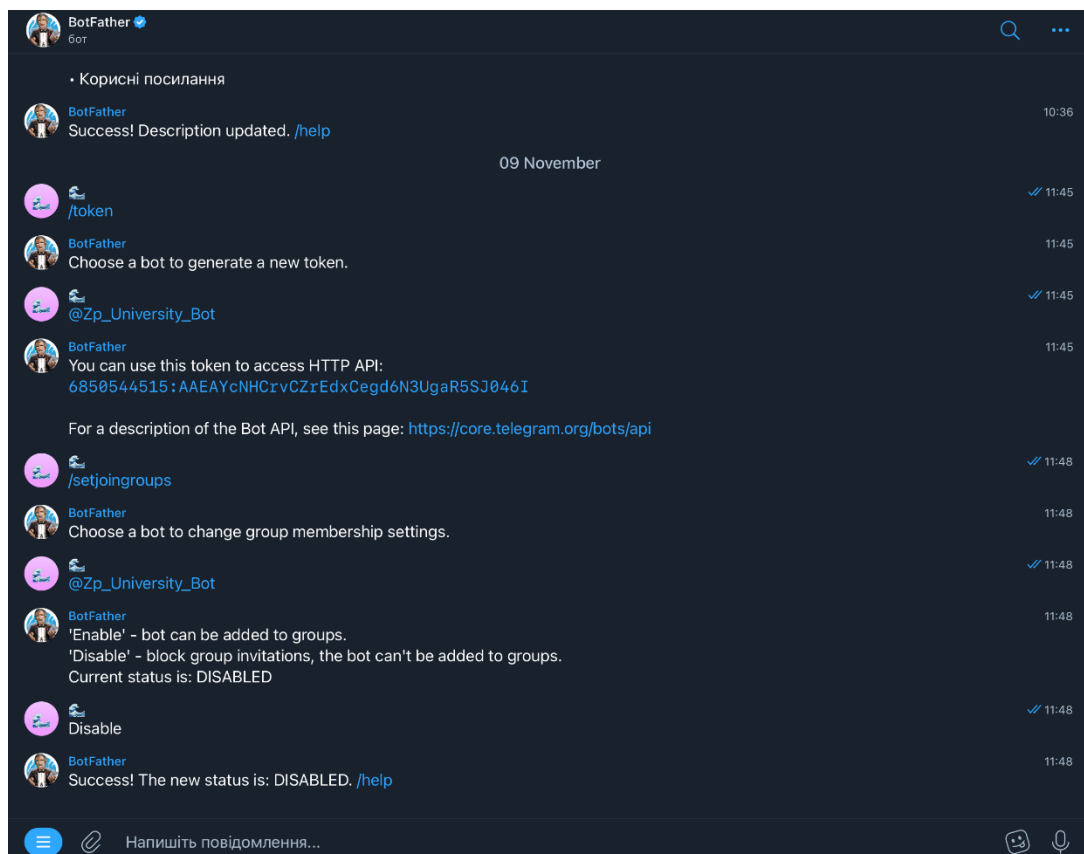


Рисунок 2.5 - Додаткові налаштування чат-бота

Далі приступаємо безпосередньо до написання коду. В першу чергу потрібно налаштувати параметри підключення до Telegram API та безпосередньо розроблюваного чат-бота.

```
"BotConfiguration": {
  "BotToken": "6850544515:AAEAYcNHCrVCZrEdxCegd6N3UgaR5SJ046I",
  "HostAddress": "https://zptelegrambot.azurewebsites.net",
  "Route": "/bot",
  "SecretToken": "ABCD4123adlf123VFcqd",
  "DropPendingUpdates": "True"
},
```

Вказана конфігурація включає «BotToken», який є унікальним ідентифікатором для доступу до API Telegram, «HostAddress» - URL-адресу сервера, де розміщений бот, «Route» - шлях для обробки оновлень від Telegram, «SecretToken» - додатковий секретний токен для забезпечення безпеки, та «DropPendingUpdates», що вказує, чи слід ігнорувати невиконані оновлення. Ці параметри є важливими для забезпечення правильної роботи та безпеки бота.

Для роботи з базою даних, як вказувалось у попередньому розділі, використовується Entity Framework. Розглянемо фрагмент коду, що представляє сутність EF.

```
using TelegramBot.Enums;
namespace TelegramBot.Entities;
public class BotUser : BaseEntity
{
    public UserRole Role { get; set; }
    public long ChatId { get; set; }
    public bool HasNotificationOn { get; set; }
    public long? TeacherId { get; set; }
    public Teacher Teacher { get; set; }
    public long? StudentId { get; set; }
    public Student Student { get; set; }
    public string Language { get; set; } = null!;
}
```

Цей код представляє собою сутність Entity Framework для класу «BotUser». Це означає, що він визначає структуру для запису в базі даних, де кожен екземпляр «BotUser» відповідає рядку в таблиці. Сутність містить властивості, такі як роль користувача, ID чату, статус сповіщень, ідентифікатори та зв'язки з об'єктами «Teacher» та «Student».

Для реєстрації та подальшого розпізнавання команд у нашому чат-боті створено клас «CommandNames».

```
public class CommandNames
{
    public const string StartCommand = "/s";
    public const string MainMenuCommand = @"🏠";
    public const string ScheduleMenuCommand = @"📅";
    public const string GetFullScheduleCommand = @"📋";
    public const string GetTodaysScheduleCommand = @"📅 31";
    public const string GetExamsScheduleCommand = @"📅";
    public const string EnableNotificationCommand = "✅";
}
```

```

public const string DisableNotificationCommand = @"✕";
public const string PinMessageInlineCommand = @"📌";
public const string ExtraInfoCommand = @"❓";
public const string NewsCommand = @"📰";
public const string GradebookCommand = @"📖";
}

```

Код, представлений вище, призначений для зберігання констант, які представляють різні команди для Telegram бота. Кожна команда має власну рядкову константу, яка може бути використана в інших частинах програми для ідентифікації та обробки цих команд. Наприклад, StartCommand визначається як "/s", що може використовуватися для розпізнавання команди запуску бота, тоді як інші константи представляють різні функції бота, такі як перегляд розкладу, управління сповіщеннями чи отримання додаткової інформації.

Далі розглянемо клас, що обробляє отриману команду на прикладі команди «/start».

```

public class StartCommand : BaseCommand
{
    public override string Name => CommandNames.StartCommand;
    private readonly IConfiguration _config;
    private readonly IRepositoryFactory _factory;
    public StartCommand(IRepositoryFactory factory, IConfiguration config)
    {
        _factory = factory;
        _config = config;
    }
    public override async Task ExecuteAsync(ITelegramBotClient client, Update upd)
    {
        long chatId = CommandsHelper.GetChatId(upd);
        var languageCode = upd.Message.From.LanguageCode!;
        var user = await _factory.Instance<BotUser>().FindAsync(user => user.ChatId ==
chatId);
        if (user != null)
            await CommandsHelper.SendWelcomeMessageAsync(client, chatId,

```

```

upd.Message!.From!.LanguageCode!);
    else
        await SendOAuthMessage(client, upd);
}
private async Task SendOAuthMessage(ITelegramBotClient client, Update upd)
{
    long chatId = CommandsHelper.GetChatId(upd);
    var languageCode = upd.Message.From.LanguageCode!;
    var scope = _config["OAuthSettings:Scope"];
    var redirectUrl = _config["OAuthSettings:RedirectUrl"];
    var oAuthEndpoint = _config["OAuthSettings:ServerEndpoint"];
    string state = chatId.ToString();
    var codeChallenge = Sha256Helper.ComputeHash(state);
    await client.SendChatActionAsync(chatId: chatId, chatAction: ChatAction.Typing);
    await client.SendTextMessageAsync(
        chatId: chatId,
        text:
            $"{LocalizationService.GetLocalizedResponse("StartResponse",
languageCode)}",
        parseMode: ParseMode.Markdown,
        replyMarkup: new InlineKeyboardMarkup(new[]
            { new[]
                { InlineKeyboardButton.WithUrl(
                    text: "\ud83d\udd10 Google",
                    url: $"{GoogleOAuthService.GenerateOAuthRequestUrl(
                        scope: scope!,
                        redirectUrl: redirectUrl!,
                        codeChallenge: codeChallenge,
                        oAuthEndpoint: oAuthEndpoint!,
                        state: state)}")
                },
            },
        )
    );
}
}

```

Властивість «Name» перевизначає базову властивість «Name», присвоюючи їй значення «StartCommand» із класу «CommandNames».

Конструктор класу приймає два параметри – «IRepositoryFactory» і «IConfiguration». Це дозволяє використовувати фабрику репозиторіїв для взаємодії з даними та конфігураційні налаштування для доступу до конфігураційних параметрів.

Метод «ExecuteAsync» викликається, коли користувач відправляє команду «/start». Він отримує ідентифікатор чату з об'єкта «Update», перевіряє, чи існує користувач у базі даних, і надсилає відповідне повідомлення: привітання, якщо користувач вже існує, або повідомлення для OAuth аутентифікації, якщо ні.

Метод «SendOAuthMessage» використовується для надсилання повідомлення з OAuth аутентифікацією. Він формує URL для OAuth запиту з використанням конфігураційних налаштувань та надсилає його користувачу для ініціації процесу аутентифікації.

Варто вказати, що для роботи з чат-ботом є обов'язковою аутентифікація з пошти університету. Для аутентифікації використовується вищезгаданий OAuth.

OAuth (Open Authorization) — це стандарт делегування доступу, який дозволяє користувачам безпечно надавати додаткам обмежений доступ до своїх ресурсів на сервері, не розкриваючи своїх основних паролів [24]. Це широко використовується для авторизації та обміну інформацією між додатками, вебсайтами та серверами. OAuth використовується багатьма великими компаніями, такими як Google, Facebook та Microsoft, для безпечного делегування доступу до своїх сервісів. Він забезпечує зручність та безпеку для користувачів, дозволяючи легко взаємодіяти з різними додатками та сервісами [3].

Також, чат-бот містить дві мови інтерфейсу – українську та англійську. Це є важливою особливістю, зважаючи на велику кількість іноземних студентів в університеті. Зміна мови відбувається в залежності від мови інтерфейсу користувача. За локалізацію в розроблюваному чат-боті відповідає клас «LocalizationService»

```

public static class LocalizationService{
    public static string? GetLocalizedCommand(string key, string code)
    {
        var resourceManager = new ResourceManager($"TelegramBot.Resources.Commands",
typeof(Commands).Assembly);

        var command = resourceManager.GetString(key,
CultureInfo.GetCultureInfoByIetfLanguageTag(code))
        ?? resourceManager.GetString(key,
CultureInfo.GetCultureInfoByIetfLanguageTag("en"));
        return command;
    }
    public static string GetLocalizedResponse(string key, string code)
    {
        var resourceManager = new ResourceManager($"TelegramBot.Resources.Responses",
typeof(Responses).Assembly);

        var response = resourceManager.GetString(key,
CultureInfo.GetCultureInfoByIetfLanguageTag(code))
        ?? resourceManager.GetString(key,
CultureInfo.GetCultureInfoByIetfLanguageTag("en"));
        return response ?? key;
    }
}

```

Цей статичний клас містить два основних методи: «GetLocalizedCommand» і «GetLocalizedResponse», кожен з яких виконує важливу роль у забезпеченні мультимовності. Ці методи використовують «ResourceManager» для доступу до рядків, які зберігаються в окремих ресурсних файлах, що дозволяє боту відповідати користувачам з урахуванням мови інтерфейсу їх додатку Telegram.

Метод «GetLocalizedCommand» використовується для отримання локалізованої версії команди, заснованої на заданому мовному коді. Якщо для заданої мови команда відсутня, метод повертає англійську версію. З іншого боку, «GetLocalizedResponse» забезпечує отримання відповідей бота, також з урахуванням мовного коду. Цей метод повертає англійську версію або ключ відповіді, якщо локалізована версія не знайдена.

2.3 Реалізація сервісу обліку робочого часу працівників університету

Після постановки задачі, вибору інструментальних засобів та проектування бази даних починається процес розробки вебдодатка. У розділі 2.1. було розглянуто функціонал авторизації користувача в системі. Розглянемо фрагмент коду клієнтської частини форми входу в систему. Повний лістинг коду знаходиться в додатку Б.

```
const LoginFromFirstStep = () => {
  return (
    <>
      <div className="external-auth-wrapper" style={{width: "65%"}}>
        <a id="google" onClick={(e) => handleAuthClick(e)} className="external-
auth"
          href="/to-external-auth?authType=google">
            <span>Continue with Google</span>
            <img style={{width: "25px", height: "25px"}} src={googleImg} alt=""/>
          </a>
        <a id="github" onClick={(e) => handleAuthClick(e)} className="external-
auth"
          href="/to-external-auth?authType=github">
            <span>Continue with Github</span>
            <img style={{width: "25px", height: "25px"}} src={githubImg} alt=""/>
          </a>
      </div>
      <div className="or">
        <span className="or-text">OR</span>
      </div>
      <form onSubmit={handleSubmit(onSubmit)} className="login-form">
        <div className="login-inputs__wrapper">
          <TextInput name="email" placeholder="Enter your work email"
            register={register("email", {required: "Email can't be empty!"})}
            errors={errors.email}/>
          <div className="password-input__wrapper">
            <PasswordInput name="password" placeholder="Password"
```

```

register={register("password", {required: "Password can't be
empty!"})}}

errors={errors.password}/>
<InputTooltip description="Forgot your password?"
url="/passwordRecovery"
urlTitle="Click here"/>
</div>
</div>
<div className="tooltip-wrapper__bold" style={{display: 'none'}}>
<InputTooltip description="Don't have an account?" url="/auth" urlTitle="Get
started"/>
</div>
<div className='submit-wrapper'>
<LargeButton type="submit" value="Login"/>
</div>
</form>
</>
);
}
return (
<div className="login-form__wrapper">
<H1 value={stepMsg[step]}/>
<div className="login-form__messages-wrapper" style={{display: `${error ? "flex"
: "none"}`}}>
{loading ? <Loader/> : ""}
<H5 value={error ? error : ""}/>
</div>
{showTwoStep?<div className="login-form"
style={{gap:"15px",marginTop:"20px"}}>
<input onChange={e=>{
setVal(e.target.value);
}} value={val} className="text-input" type="text" placeholder="You code"/>
<LargeButton handleClick={authWithCode} type="submit" value="Login"/>
</div>:<LoginFromFirstStep />}
</div>

```

);

Форма містить такі основні елементи:

1. Авторизація за допомогою Google та GitHub: показує дві посилання для авторизації через Google і GitHub. Використовується обробник подій «onClick», який викликає функцію «handleAuthClick» при кліку на посилання. Для авторизації за допомогою цих сервісів використовується OAuth.

2. Форма для входу: включає поля для введення електронної пошти та пароля. Використовується компонент «TextInput» для поля електронної пошти та «PasswordInput» для поля пароля.

3. Кнопка входу: компонент «LargeButton» використовується як кнопка для подання форми.

Далі розглянемо безпосередньо процес запису робочого часу. Вебдодаток містить декілька варіантів ведення обліку робочих годин: запуск таймера та ручний ввід даних. Розглянемо детальніше роботу таймера.

```
export const TrackerTimer = ({ workedHour }: { workedHour?: WorkedHour }) => {
  if (workedHour) {
    const time = GetTimeFromString(workedHour.totalTime);
    return (
      <>
      <CurrentDateElement date={workedHour.startDate.toDate()} showFullDate =
{true} />
      <div className="tracker-content__wrapper">
        <div className="tracker-content__inner">
          <div className="timer-tracker">
            <div className="timer-tracker__wrapper worked-hours__timer">
              <Timer hours={time.hours} minutes={time.minutes}
seconds={time.seconds} />
            </div>
            <WorkedHours workedHour={workedHour} />
          </div></div></div>
      </>
    )
  }
}
```

```

const dispatch = useDispatch();
const { hours, minutes, seconds, isRunning, startedAt } = useTypedSelector((state) =>
state.timer);
const { user } = useTypedSelector(state => state.auth)
useEffect(() => {
  if (isRunning) {
    const intervalId = setInterval(() => {
      dispatch(tick());
    }, 1000);
    return () => {      clearInterval(intervalId);      }
  }
}, [dispatch, isRunning]);
const handleStartButton = () => { if (!isRunning) { dispatch(startTimer(startedAt)); } };
const handleStopButton = () => {
  const startDate = new Date(startedAt!);
  const stopDate = new Date();
  dispatch(resetTimer({
    userId: user!.id,
    startDate: moment(startDate).utc().format("YYYY-MM-DDTHH:mm:ss"),
    endDate: moment(stopDate).utc().format("YYYY-MM-DDTHH:mm:ss")
  }));
}
return (
  <>
    <CurrentDateElement date={new Date()} showFullDate={true} />
    <div className="tracker-content__wrapper">
      <div className="tracker-content__inner">
        <div className="timer-tracker">
          <div className="timer-tracker__wrapper">
            <Timer hours={hours} minutes={minutes} seconds={seconds}/>
          </div>
          <div className="tracker-btn__wrapper">
            { isRunning ? <SmallButton type="button" value="Stop"
handleClick={handleStopButton} />
              : <SmallButton type="button" value="Run"

```

```

handleClick={handleStartButton} /> }
      </div> </div></div></div>

</>
  );
};

```

Вищевказаний лістинг коду містить різні компоненти, розглянемо їх детальніше.

Кнопки управління таймером для його запуску та зупинки. Функції «handleStartButton» і «handleStopButton» викликаються при натисканні на кнопки і використовують дії Redux для управління станом таймера.

Компонент використовує Redux для управління станом таймера («useDispatch», «useTypedSelector»). «useEffect» використовується для створення інтервалу, який оновлює час кожну секунду, коли таймер працює. При зупинці таймера фіксується час початку та завершення, і ці дані відправляються через дію Redux. Цей компонент ефективно управляє відображенням та управлінням таймером у додатку, використовуючи Redux для управління станом і React Hooks для життєвого циклу компонента.

Далі, в «timerSlice» з Redux Toolkit визначає декілька дій («resetTimer», «resetTimerSuccess», «resetTimerFail», «startTimer», «tick») та їхні обробники. Розглянемо дії, що відповідають за зупинку таймера та передачі даних до метода, що відповідає за збереження відпрацьованого часу до БД.

```

resetTimer:                                createPendingReducerWithPayload<TimerSliceState,
CreateWorkedHourType>(),
resetTimerSuccess:                        createSuccessReducerWithPayload<TimerSliceState,
WorkedHour>(<
  (state: TimerSliceState, action: PayloadAction<WorkedHour>) => {
    state.startedAt = null,
    state.hours = 0,
    state.minutes = 0,
    state.seconds = 0,
    state.isRunning = false
  }

```

```

    ),
    resetTimerFail: createErrorReducer((state: TimerSliceState) => {
      state.isRunning = false;
    }),

```

Далі розглянемо асинхронний API запит, який використовується для збереження даних у базу даних.

```

export const resetTimerEpic: Epic = (action:
Observable<PayloadAction<CreateWorkedHourType>>, state) =>
  action.pipe(
    ofType("timer/resetTimer"),
    mergeMap(action =>
      CreateWorkedHoursQuery(action.payload).pipe(
        mergeMap(async resp => {
          if (resp.response.errors != null) {
            console.log(resp.response)
            const errorMessage = await
GetErrorMessage(resp.response.errors[0].message);
            return resetTimerFail(errorMessage)
          }
          return resetTimerSuccess(resp.response.data.workedHourMutations.create);
        }),
        catchError((e: Error) => {
          console.log(e)
          return of(resetTimerFail("Unexpected error"))
        })
      ),
    ),
  )
);

export const resetTimerSuccessEpic: Epic = (action:
Observable<PayloadAction<WorkedHour>>, state) =>
  action.pipe(
    ofType("timer/resetTimerSuccess"),
    mergeMap(action => {
      return of(createWorkedHourSuccess(action.payload));
    })
  )

```

)

«resetTimerEpic» - це epic - асинхронний обробник дій у Redux-Observable. Він «слухає» дії типу «timer/resetTimer».

У межах цього epic, викликається функція «CreateWorkedHoursQuery», передаючи їй payload з дії resetTimer. Ця функція робить API запит для створення нового запису про відпрацьовані години, що означає взаємодію з сервером, який потім може зберегти ці дані у базу даних. Якщо в процесі виконання запиту виникає помилка, використовується «catchError» для перехоплення помилки і генерування дії «resetTimerFail» з повідомленням про помилку. Далі, для збереження даних в БД використовується відповідна мутація.

Мутація у контексті GraphQL є способом зміни даних на сервері, наприклад, додавання, оновлення або видалення записів. Вона подібна до POST, PUT або DELETE запитів у REST API. GraphQL використовує мутації для визначення того, які операції можна виконувати на даних, і як ці дії впливають на стан сервера. Розглянемо мутацію, що відповідає за збереження запису з таймера в БД, а саме метод «create». Повний лістинг мутації «WorkedHourMutations» знаходиться в додатку В.

```
public sealed class WorkedHourMutations : ObjectGraphType
{
    public WorkedHourMutations(IUnitOfWorkRepository uow, IMapper mapper)
    {
        Field<WorkedHour>("create")
            .Argument<WorkedHourInputType>("workedHour")
            .ResolveAsync(async ctx => {
                var wh = ctx.GetArgument<WorkedHourInputDto>("workedHour");
                if (wh.StartDate > DateTime.UtcNow
                    || wh.EndDate > DateTime.UtcNow)
                { throw new ValidationError("Can't create worked hour for future"); }
                var exists = await uow.GenericRepository<WorkedHour>()
                    .FindAsync(p => p.UserId == wh.UserId && p.StartDate < wh.EndDate
                        && wh.StartDate < p.EndDate);
                if (exists is not null)
```

```

        throw new ValidationError("Worked hours intersect!");
    var created = await uow.GenericRepository<WorkedHour>()
        .CreateAsync(mapper.Map<WorkedHour>(wh));
    await uow.SaveAsync();
    return created;
});

```

Ця мутація приймає один параметр – «workedHour», який є об'єктом типу «WorkedHourInputType». Цей тип визначає структуру даних, які передаються у запиті.

```

public sealed class WorkedHourInputType: InputObjectGraphType<WorkedHourInputDto>
{
    public WorkedHourInputType()
    {
        Field(v => v.UserId).Description("user id");
        Field(x => x.StartDate).Description("start date");
        Field(x => x.EndDate).Description("end date");
    }
}

```

Продовжимо огляд функції «create». Після отримання даних, функція проводить ряд перевірок на їх валідність. Перевіряє, що дата початку та кінця не знаходиться у майбутньому. Якщо це так, генерує помилку валідації. Перевіряє, чи не перетинаються нові відпрацьовані години з вже існуючими записами. Якщо є перетин, також генерує помилку.

Якщо всі перевірки пройдені, використовується репозиторій для створення нового запису в базі даних (uow.GenericRepository<WorkedHour>().CreateAsync(mapper.Map<WorkedHour>(wh))).

В даному контексті uow є інстанцією класу, який реалізує патерн Unit of Work. Це патерн, який використовується для групування декількох операцій з базою даних у одну транзакцію, щоб забезпечити консистентність даних. Він дозволяє виконати серію операцій і зберегти або відмінити їх як одне ціле.

Командою «CreateAsync» ми ініціюємо асинхронне створення запису про відпрацьовані години, використовуючи патерни Unit of Work і Repository.

Висновки до другого розділу

В цьому розділі було розроблено структуру бази даних та описано засоби та технології для роботи з нею. Використанням фреймворку ASP.NET було зумовлено використання певних патернів, зокрема MVC та фабричний патерн, а також певних технологій та фреймворків, таких як Entity Framework, GraphQL.

Далі було проведено проектування та реалізацію алгоритмів роботи чат боту. Для реалізації та розгортання роботи чат-бота було обрано месенджер Telegram. Було описано команди, які може обробляти чат-бот та наведено та описано приклади їх реалізації.

Для сервісу обліку робочого часу було розроблено вебдодаток. Описано процес розробки клієнтської частини сервісу за допомогою React та описано реалізацію її зв'язку з серверною частиною за допомогою Redux та GraphQL.

РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СЕРВІСОМ

3.1 Структура вебдодатка

Для роботи з вебдодатком обов'язковим є вхід в систему. Форму авторизації зображено на рис. 3.1.

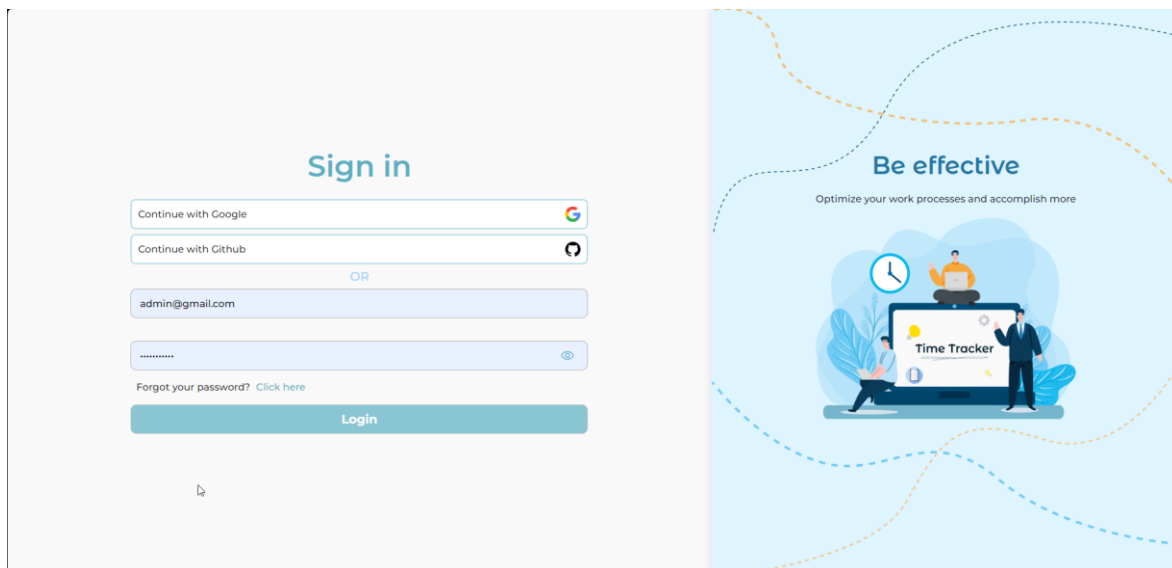


Рисунок 3.1 - Форма входу

На рисунку зображено веб-інтерфейс форми входу. Користувачі можуть увійти в систему використовуючи свої акаунти Google або GitHub, що вказано кнопками "Continue with Google" та "Continue with Github" відповідно. Нижче є традиційні поля для введення електронної адреси та пароля. Також є посилання для відновлення забутого пароля. Після авторизації в системі користувач переходить на головну сторінку.

Інтерфейс вебдодатка містить таку структуру:

1. Головна. Головна сторінка вебдодатка містить інформацію про відсоток відпрацьованих годин з запланованих на сьогодні або на місяць, робочі плани на поточний день, таймер для обліку робочого часу, розклад подій та планів, нещодавню активність та інформацію про відпустки.
2. Календар. Календар призначений для перегляду та додавання заходів чи планів.
3. Панель керування. На цій сторінці відбувається перегляд списку,

запис та редагування відпрацьованих годин. Адміністратор також має можливість дивитись статистику інших користувачів.

4. Мої відпустки. Сторінка зі списком відпусток та для їх створення. Список відпусток містить інформацію про дату відпустки та її статус (підтверджена/не підтверджена/скасована).

5. Запити. Сторінка з запитами інших користувачів на відпустки або лікарняні. Адміністратор бачить запити інших користувачів та може задовольнити запит або відмовити в ньому.

6. Команда. Сторінка де знаходиться список користувачів. Адміністратор може їх додавати, видаляти, редагувати. Також, адміністратор має можливість обирати кількість робочих годин працівника. Тобто, якщо людина влаштовується на роботу з частковою зайнятістю – адміністратор ставить їй відповідний відсоток запланованих годин, які вона має відпрацювати за місяць. Також на цій сторінці наявна можливість вивантаження звітів про відпрацьовані працівниками робочі години.

7. Налаштування. В налаштуваннях користувачі мають можливість увімкнути двоетапну аутентифікацію.

Розглянемо детальніше вигляд головної сторінки на рис. 3.2.

Зліва можна побачити статичне меню, яке містить описані вище пункти меню.

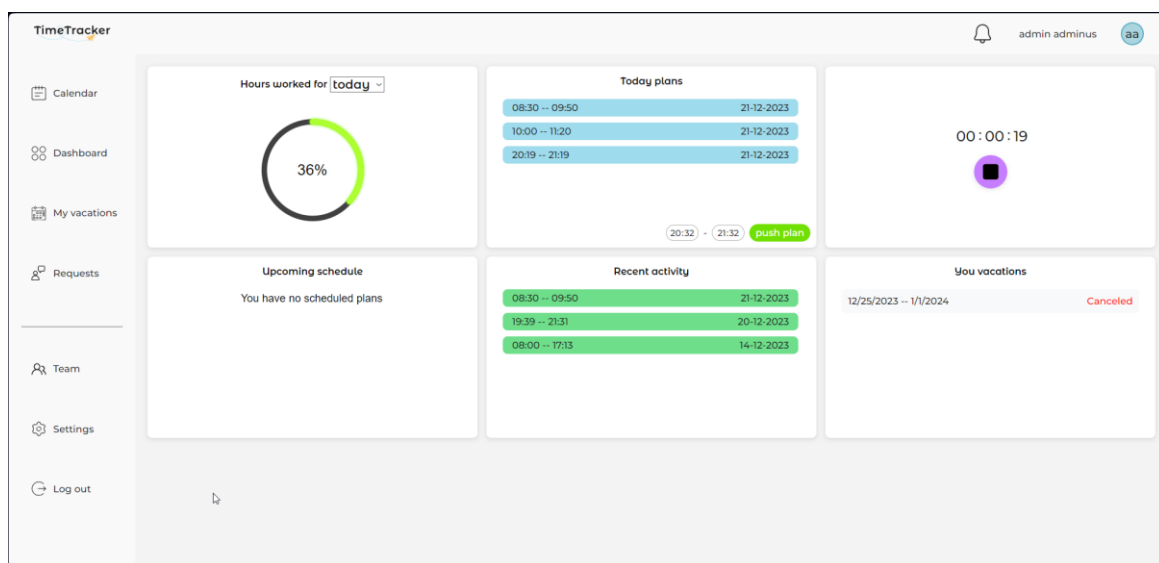


Рисунок 3.2 - Інтерфейс головної сторінки

Далі розглянемо інтерфейс панелі керування.

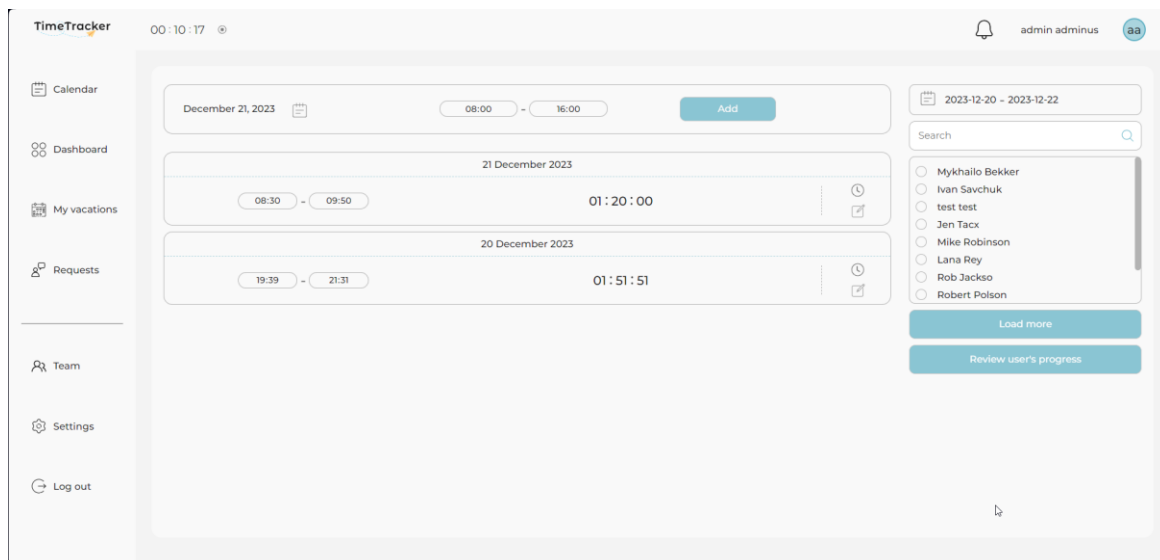


Рисунок 3.3 - Інтерфейс панелі керування

На рис. 3.3 можна побачити інтерфейс додавання запису обліку робочих годин, який містить вибір дати, часу початку та часу закінчення робочого відрізка. Нижче знаходиться список раніше зроблених записів та кнопки для їх редагування. З правої сторони знаходиться меню для вибору користувача та часового проміжку. Користувач за допомогою цього інструменту може бачити робочі години інших працівників.

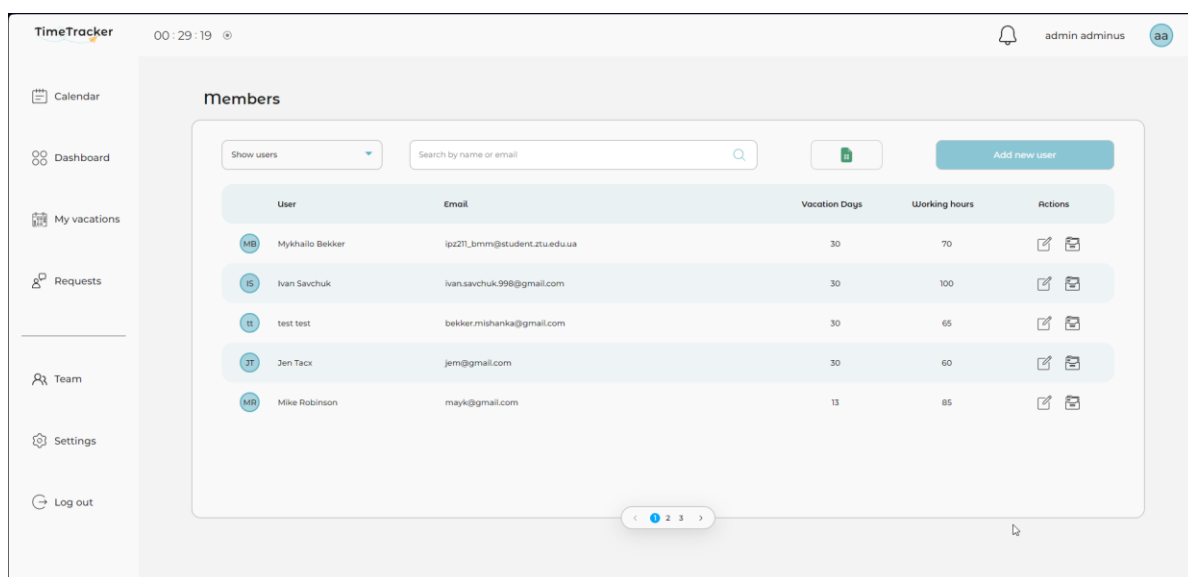


Рисунок 3.4 - Інтерфейс сторінки «Користувачі»

Сторінка «Користувачі» містить список працівників. Адміністратор має можливість додавати нових за допомогою кнопки «Add new user». Також є

фільтр для вибору користувачів для виводу за статусом – активний користувач чи заархівований(звільнений). Також є пошук користувачів за іменем чи електронною поштою. З правої сторони від пошуку знаходиться кнопка для вивантаження звіту про відпрацьовані користувачами години. Звіт зберігається у форматі Microsoft Excel.

Розглянемо цю сторінку для користувача, що не є адміністратором (рис. 3.5).

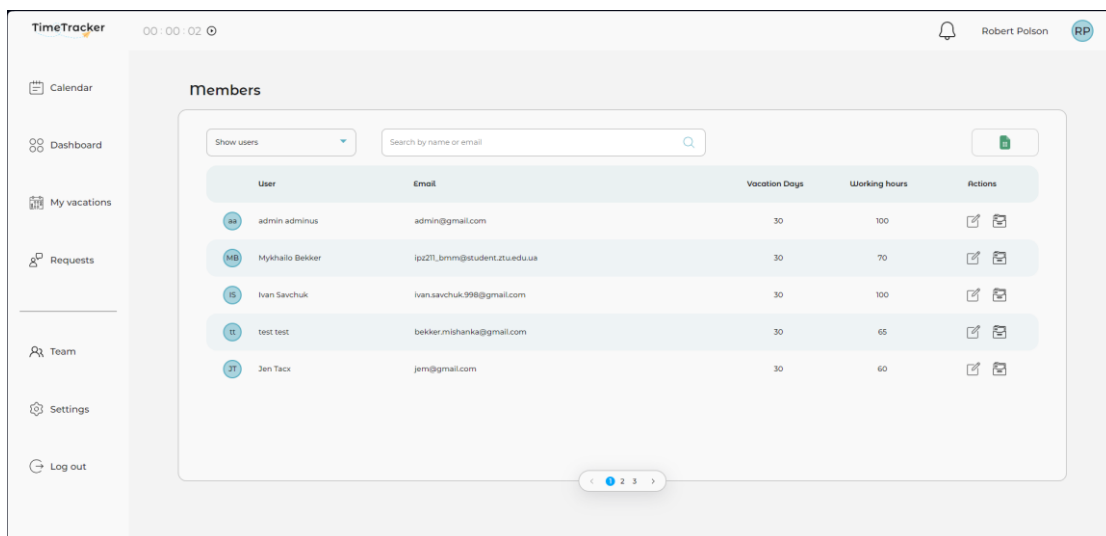


Рисунок 3.5 - Сторінка «Користувачі» для користувача, що не є адміністратором

Як можна побачити, у даного користувача відсутня кнопка додавання нових користувачів, а кнопки редагування чи архівування є неактивними. Якщо перейти на сторінку редагування користувача – буде отримано повідомлення про відсутність доступу (рис. 3.6).

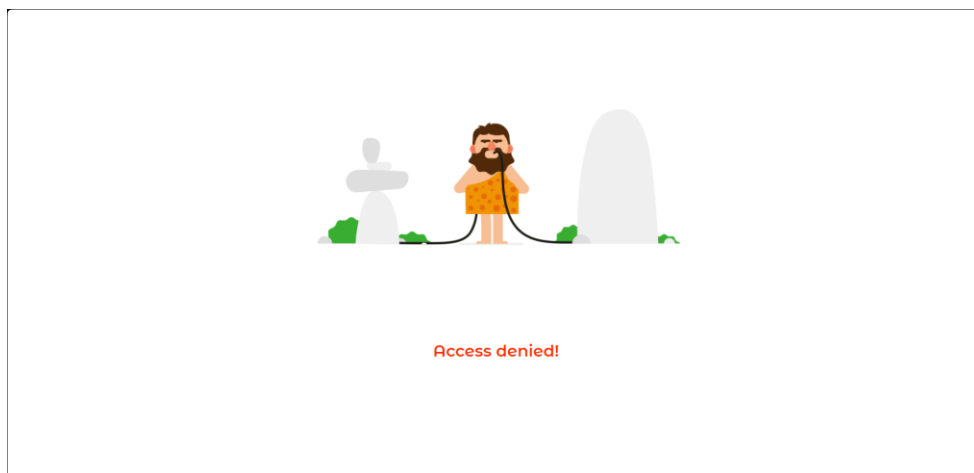


Рисунок 3.6 - Повідомлення про відсутність доступу

Також, якщо запущено таймер та користувач знаходиться не на головній сторінці – таймер розміщено у верхньому лівому кутку меню.

3.2 Структура чат-бота

Розроблений чат-бот містить багаторівневе меню для спрощення роботи з ним. Розглянемо головне меню (рис. 3.7).

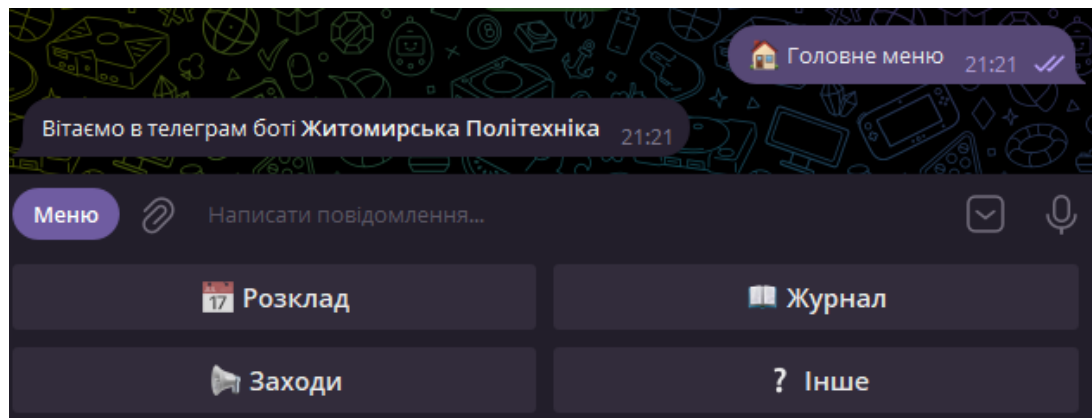


Рисунок 3.7 - Головне меню чат-бота

Головне меню містить такі розділи(команди):

1. «Розклад»;
2. «Журнал»;
3. «Заходи»;
4. «Інше».

«Розклад» містить своє внутрішнє меню з команд для відображення розкладу на сьогодні, на тиждень, розкладу сесії, повернення на головну сторінку, а також команду для увімкнення автоматичних сповіщень про заняття (рис. 3.8).

«Журнал» – це команда, що відправляє користувачу Excel файл з його оцінками за лабораторні роботи, отриманими в поточному семестрі.

«Заходи» виводять список дат для вибору. При виборі дати виводяться заходи, що заплановані на неї, або повідомлення, що заходів не заплановано.

«Інше» відправляє список корисних посилань, таких як: електронний кабінет студента, освітній портал, реквізити для оплати або скриньку добра.

Працюючи з чат-ботом на пристрої з англійською мовою інтерфейсу

змінюється мова інтерфейсу чат-бота (рис. 3.9).

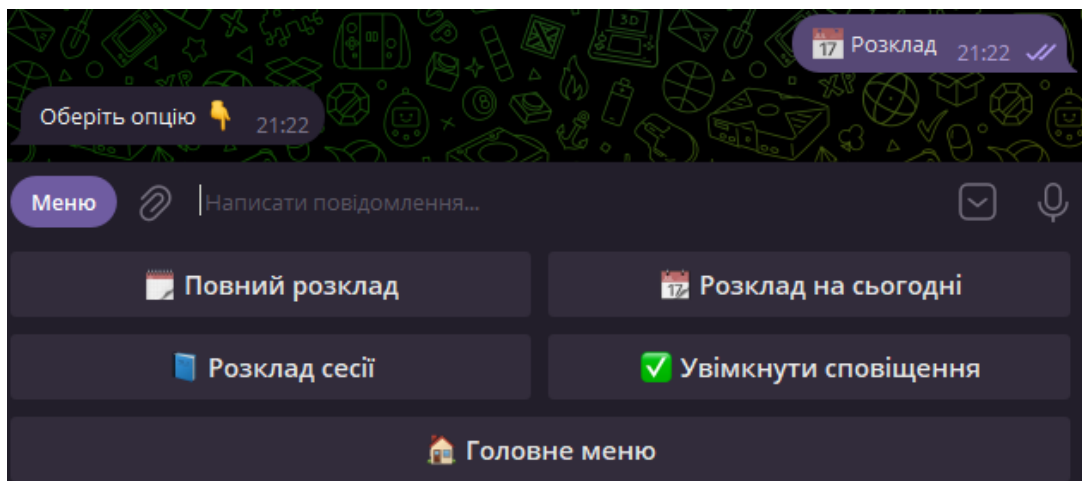


Рисунок 3.8 - Меню «Розклад»



Рисунок 3.9 - Інтерфейс чат-бота англійською мовою

3.3 Тестування роботи ПЗ

Почнемо тестування вебдодатка з авторизації в системі. Форму входу в систему зображено на рис. 3.1. Підтвердимо відправку форми та авторизуємось в вебдодатка. На рис. 3.10 відображено результат входу в систему в профіль адміністратора.

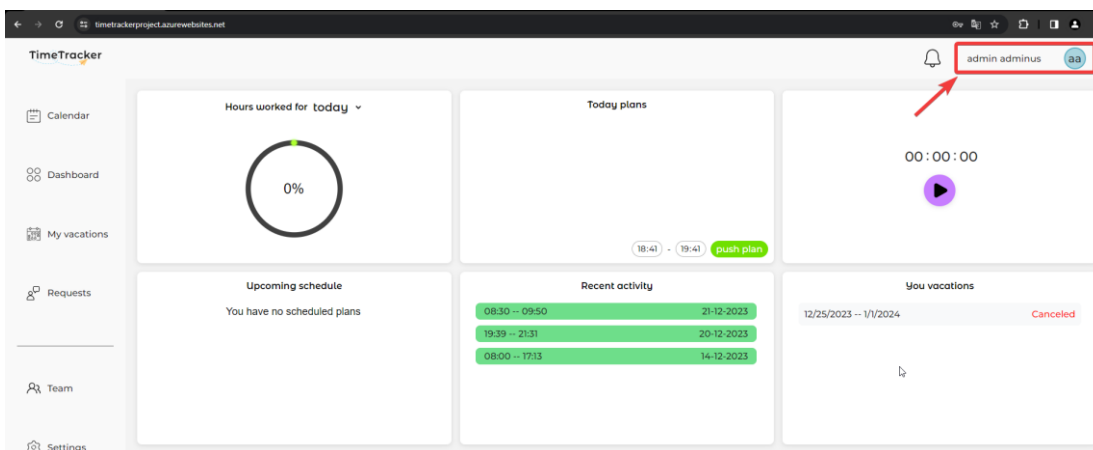


Рисунок 3.10 - Успішний вхід в систему

Далі, створимо робочий план на сьогоднішній день. Нехай це буде одна робоча година. Для цього оберемо потрібний нам час та натиснемо кнопку «push plan» (рис. 3.11). Результат створення плану зображено на рис. 3.12. Також, на цьому рисунку можна побачити відсоток відпрацьованих годин з запланованих на сьогодні та запущений таймер. Через 8 хвилин зупинимо його та зафіксуємо новий запис про відпрацьовані години, а також зміну відсотка відпрацьованих годин (рис. 3.13).

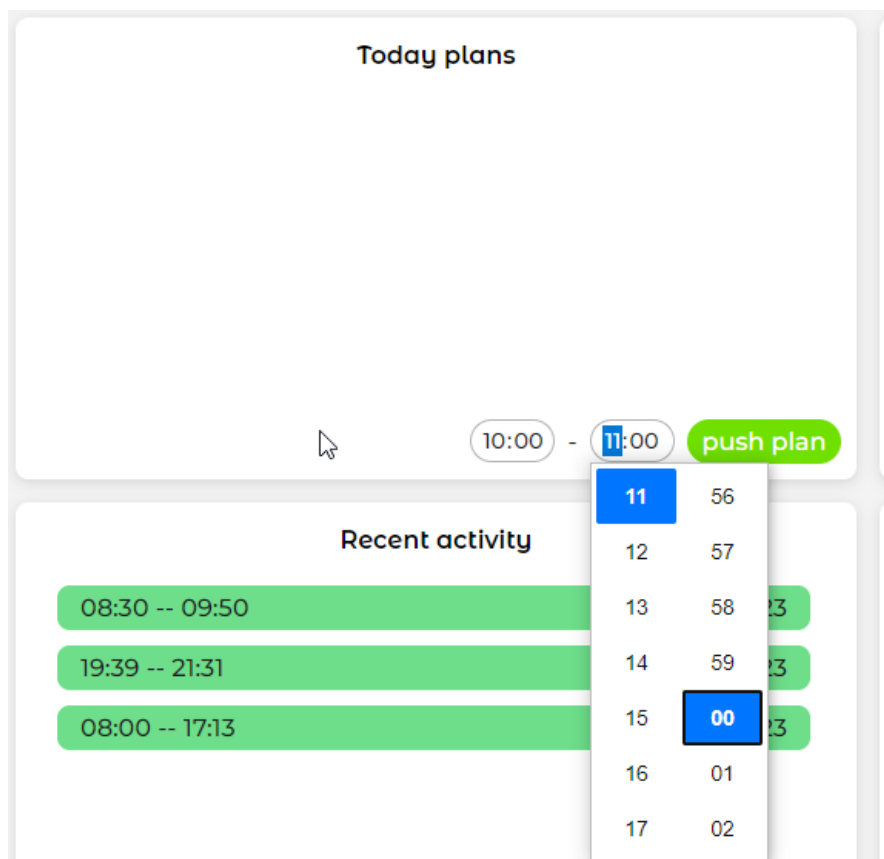


Рисунок 3.11 - Створення плану

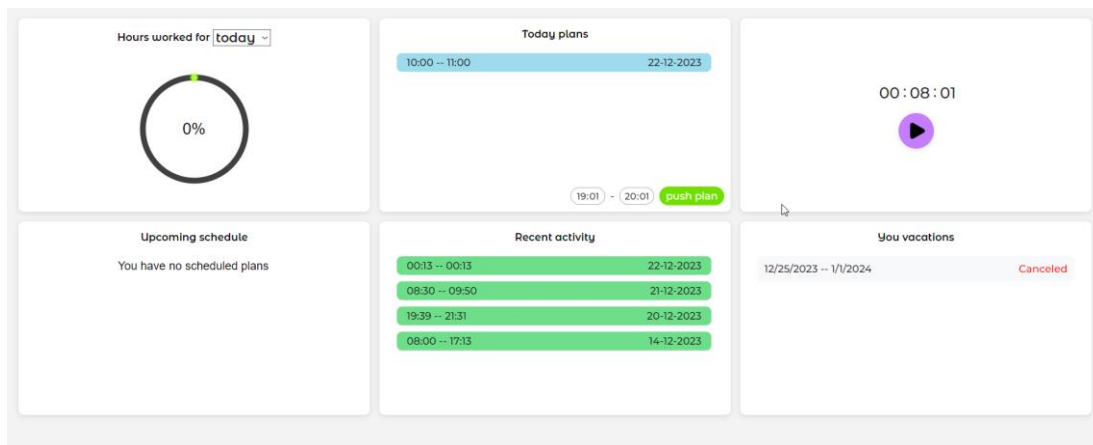


Рисунок 3.12 - Створений план та працюючий таймер

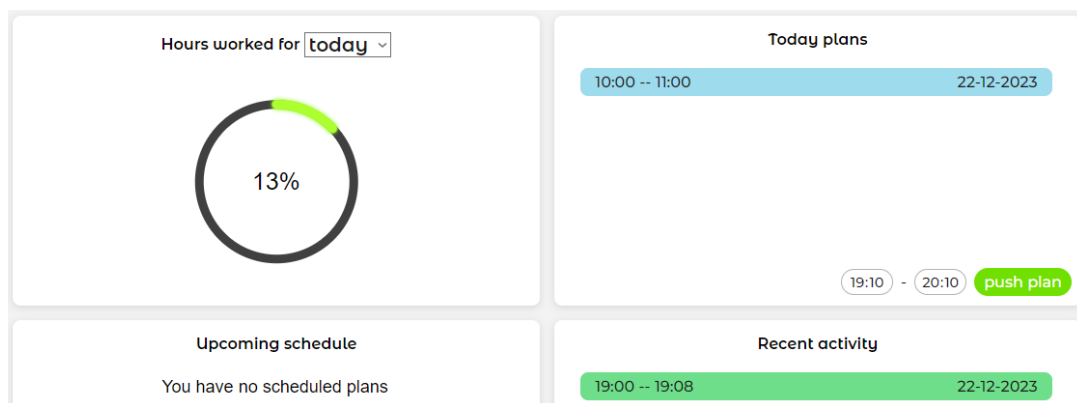


Рисунок 3.13 - Результат зупинки таймера

Як можна побачити, після зупинки таймера на позначці в 8 хвилин створився новий запис в списку останніх активностей, а також змінився відсоток відпрацьованих годин відносно плану. Додамо до плану ще 20 хвилин, щоб сумарно план на день складав 80 хвилин, а 8 відпрацьованих хвилин складало рівно 10%.

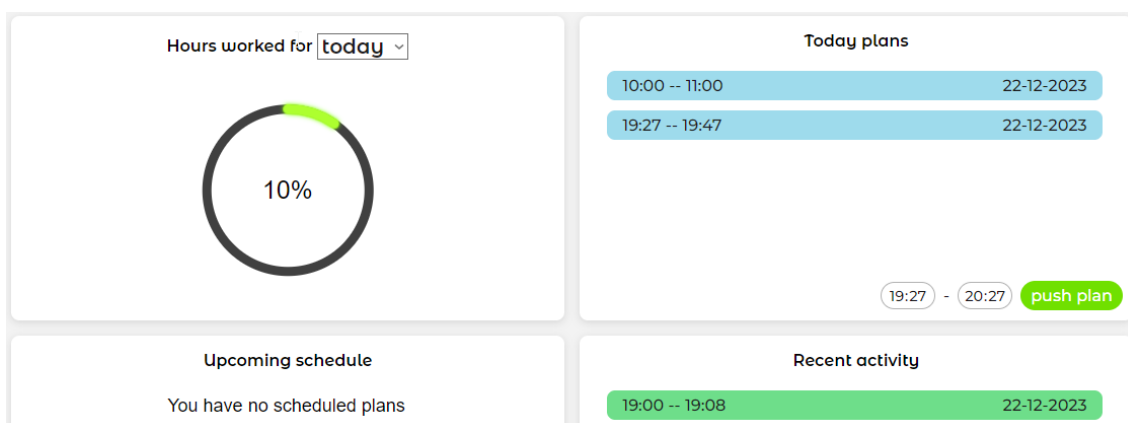


Рисунок 3.14 - Результат додавання 20 хв до плану на день

З рис. 3.14 видно, що розрахунок часу працює коректно. Далі протестуємо роботу ручного додавання відпрацьованого часу. Запишемо відпрацьовану годину та 12 хвилин, щоб в сумі отримати 100% виконання плану. Інтерфейс додавання та вже створений запис зображено на рис. 3.15.

The interface shows a date selector set to 'December 22, 2023'. Below it, a time range '09:00 - 10:12' is entered, with an 'Add' button to the right. A second section, titled '22 December 2023', displays the added entry as '09:00 - 10:12' and shows a total of '01 : 12 : 00' (1 hour and 12 minutes). There are icons for a clock and a list on the right side of this section.

Рисунок 3.15 - Додавання запису відпрацьованого часу вручну

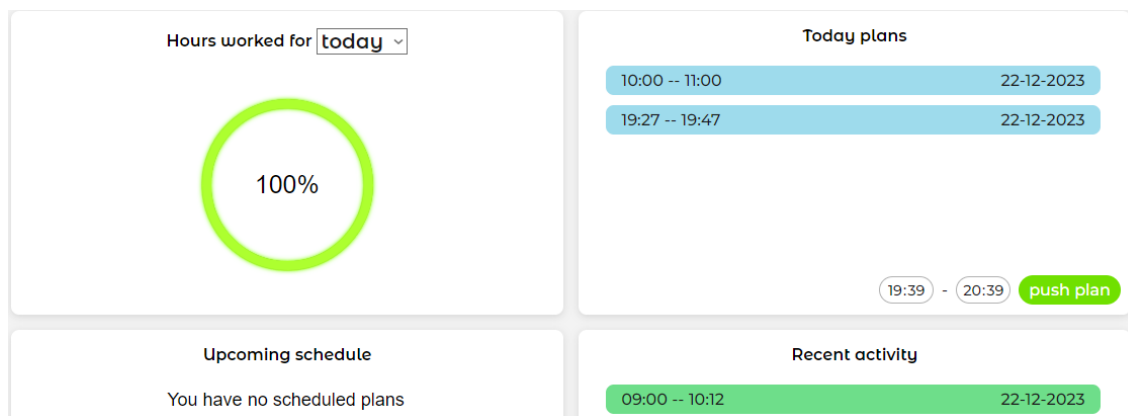


Рисунок 3.16 - Перевірка роботи ручного додавання робочих годин

Перевіримо чи є можливість додати два записи на один проміжок часу. Спробуємо створити новий запис з 9 до 10 години.

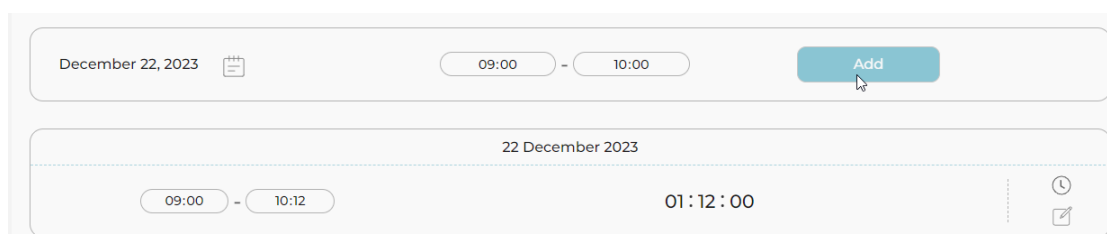


Рисунок 3.17 - Спроба створити запис, що перетинається з іншим

Після виконання цієї дії новий запис не створюється, а в панелі інструментів розробника в браузері, на вкладці "Мережа" (Network), можна побачити повідомлення про помилку "Worked hours intersect", це означає, що є проблема з перетином робочих годин.

Спробуємо переглянути прогрес інших працівників. Для цього оберемо проміжок часу з 30.11.2023 до 22.12.2023 та потрібного нам користувача (рис. 3.18).

На рис. 3.18 зображено записи робочих годин користувача Paul Tibit за вищевказаний проміжок часу. Запис містить в собі дату, часовий проміжок та загальну кількість відпрацьованого часу. Функціонал працює коректно.

Далі перевіримо роботу вивантаження звітів про відпрацьовані працівниками години. Для цього перейдемо на сторінку «Користувачі» та натиснемо відповідну кнопку (рис. 3.19).

Після натискання кнопки на комп'ютер завантажується файл у форматі Microsoft Excel (рис. 3.20).

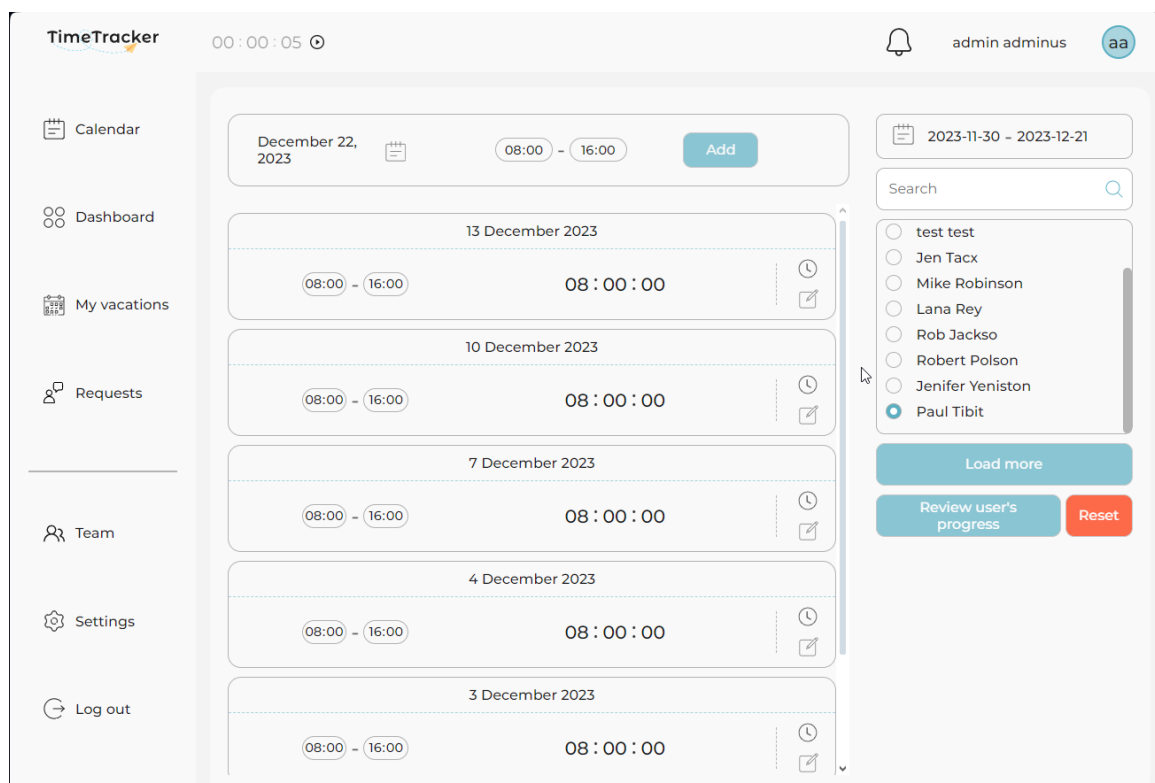


Рисунок 3.18 - Перегляд робочих годин інших працівників

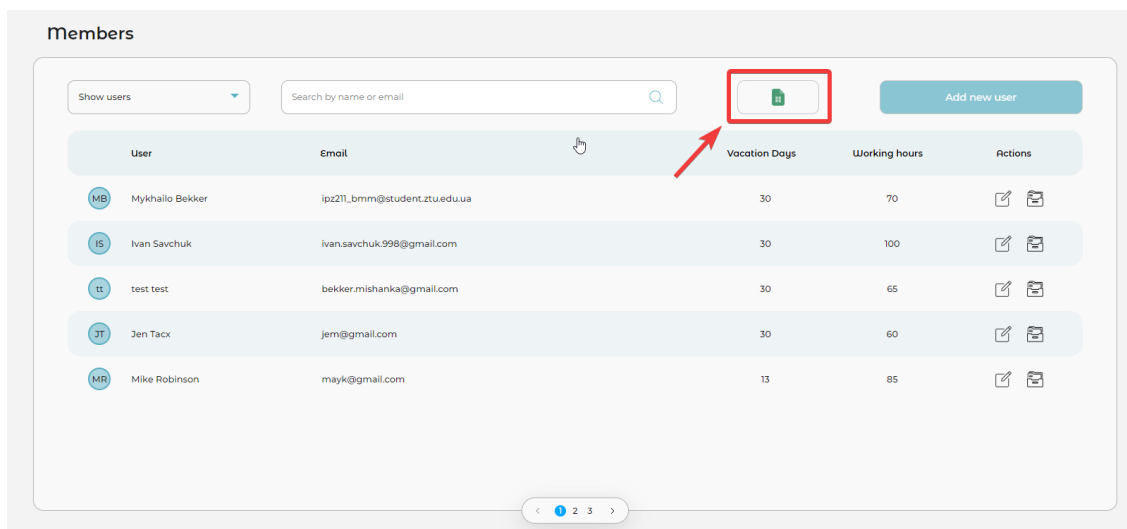


Рисунок 3.19 - Кнопка вивантаження звіту

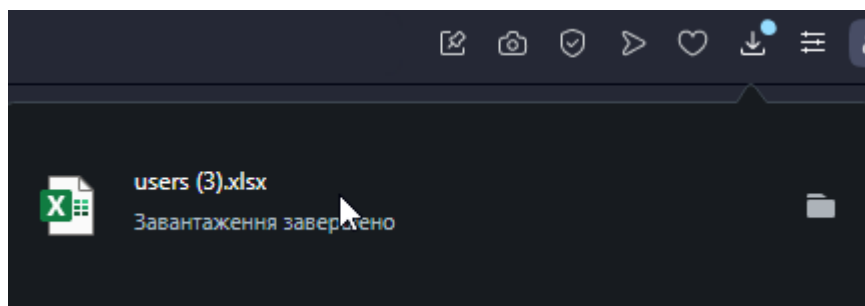
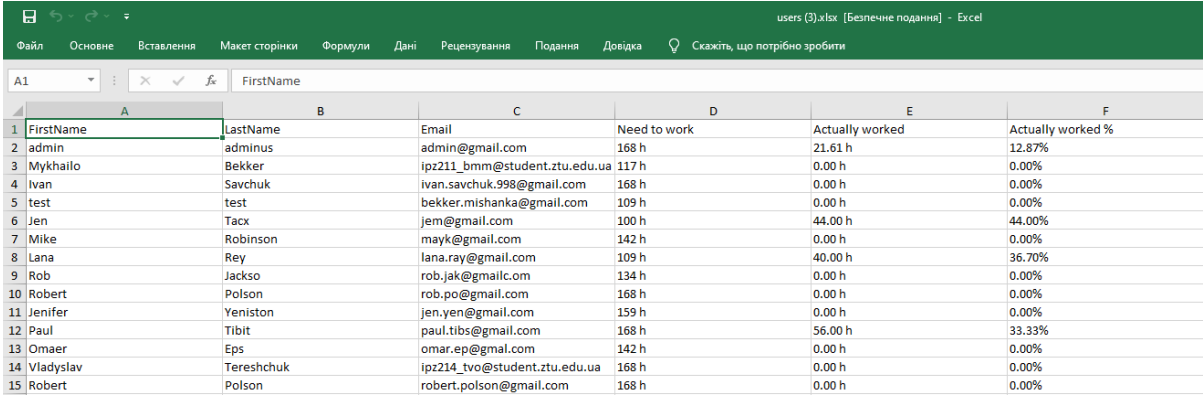


Рисунок 3.20 - Завантажений файл

Відкриємо завантажений файл та переглянемо його зміст. Він має містити такі дані:

- Ім'я та Прізвище користувача;
- Електронну адресу користувача;
- Кількість годин, що мають бути відпрацьованими за робочий місяць;
- Фактичну кількість годин, відпрацьованих користувачами;
- Відсоток відпрацьованих годин від запланованих на місяць.



	A	B	C	D	E	F
	FirstName	LastName	Email	Need to work	Actually worked	Actually worked %
1	admin	adminus	admin@gmail.com	168 h	21.61 h	12.87%
2	Mykhailo	Bekker	ip211_bmm@student.ztu.edu.ua	117 h	0.00 h	0.00%
3	Ivan	Savchuk	ivan.savchuk.998@gmail.com	168 h	0.00 h	0.00%
4	test	test	bekker.mishanka@gmail.com	109 h	0.00 h	0.00%
5	Jen	Tacx	jem@gmail.com	100 h	44.00 h	44.00%
6	Mike	Robinson	mayk@gmail.com	142 h	0.00 h	0.00%
7	Lana	Rey	lana.ray@gmail.com	109 h	40.00 h	36.70%
8	Rob	Jackso	rob.jak@gmail.com	134 h	0.00 h	0.00%
9	Robert	Polson	rob.po@gmail.com	168 h	0.00 h	0.00%
10	Jenifer	Yeniston	jen.yen@gmail.com	159 h	0.00 h	0.00%
11	Paul	Tibit	paul.tibs@gmail.com	168 h	56.00 h	33.33%
12	Omaer	Eps	omar.ep@gmail.com	142 h	0.00 h	0.00%
13	Vladyslav	Tereshchuk	ip214_two@student.ztu.edu.ua	168 h	0.00 h	0.00%
14	Robert	Polson	robert.polson@gmail.com	168 h	0.00 h	0.00%

Рисунок 3.21 - Звіт про робочі години працівників

Зміст файлу відповідає поставленим вимогам та містить коректну інформацію про відпрацьовані працівниками години.

Загалом, вебдодаток пройшов тести та відповідає поставленим вимогам.

Далі, перевіримо роботу чат-бота.

Робота з чат ботом починається з команди «/start» після чого він надсилає посилання для авторизації за допомогою пошти університету використовуючи OAuth (рис. 3.22). Після успішної авторизації чат-бот з вами вітається та стає доступним головне меню (рис. 3.23).

Далі перевіримо роботу функціоналу, що відповідає за надсилання розкладу. Надішлемо запит на розклад на сьогоднішній день та на повний розклад. Враховуючи, що у студентів 2 курсу магістратури заняття вже не проводяться, чат-бот має відповісти, що пари на сьогодні відсутні, але повний розклад на два тижні має бути надіслано. Перевірку цього зображено на рис. 3.24.

Перевірка вищезгаданого функціоналу пройшла успішно. Чат-бот відправив очікувані дані, а саме повідомлення про відсутність пар у відповідний день, а також файл у форматі pdf з розкладом на два тижні. Зміст файлу

зображено на рис. 3.25.

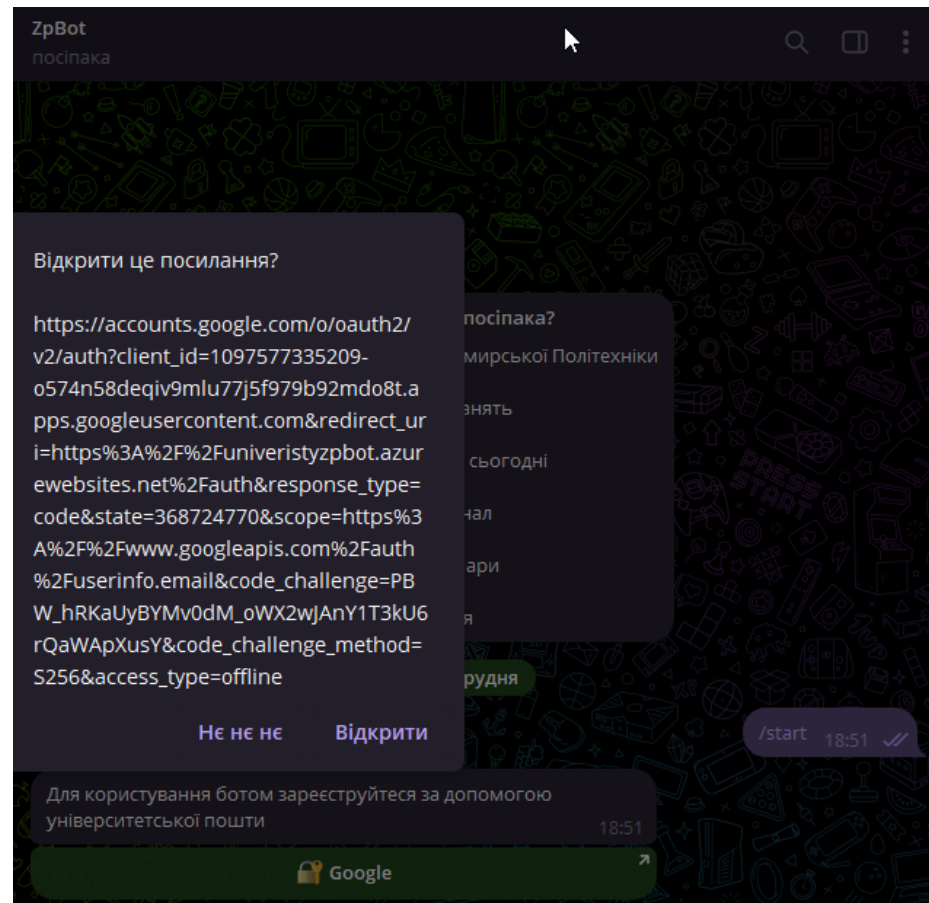


Рисунок 3.22 - Посилання для авторизації в чат-боті

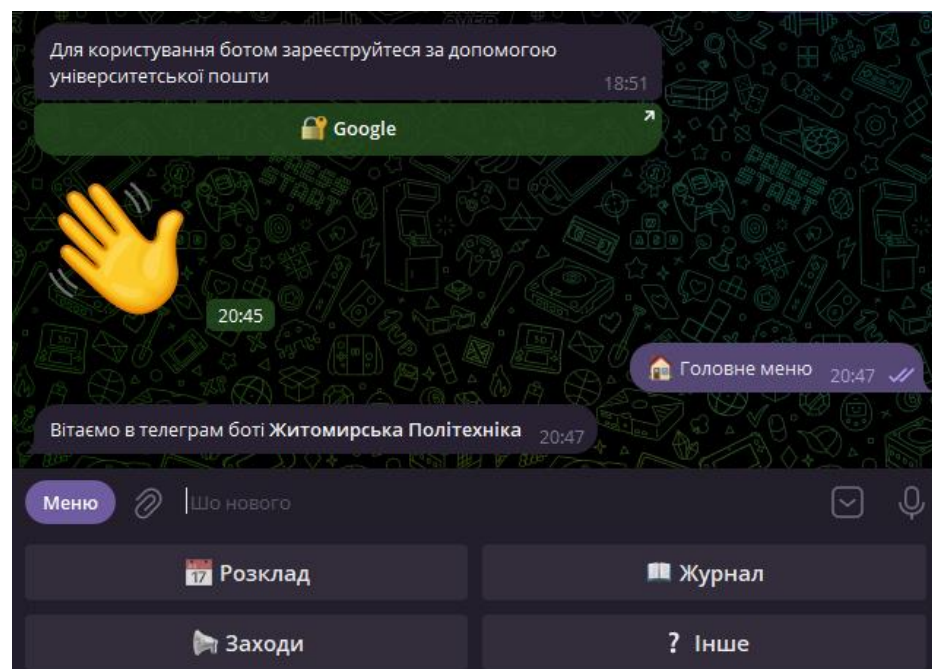


Рисунок 3.23 - Успішна авторизація в чат-боті

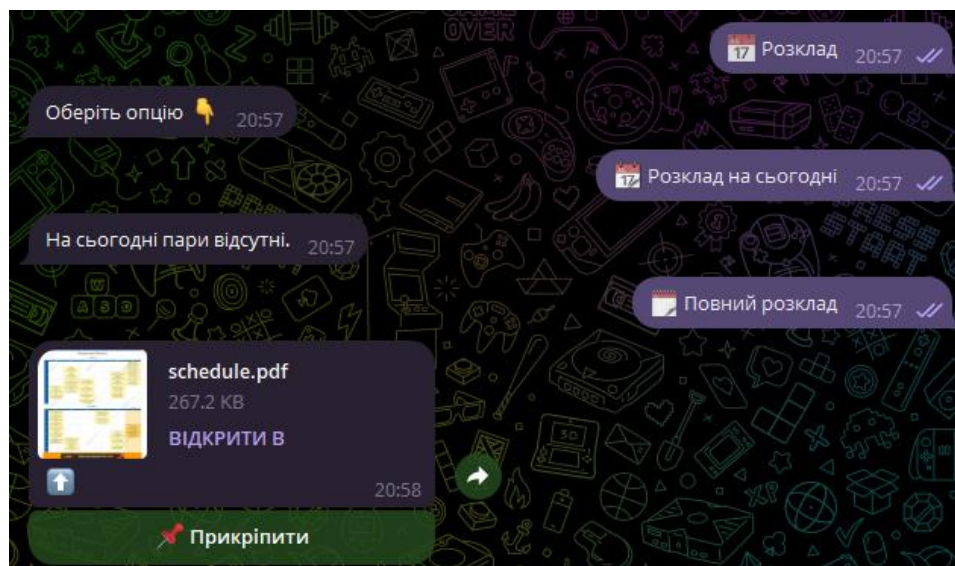


Рисунок 3.24 - Відповідь чат-бота на команди з розкладом

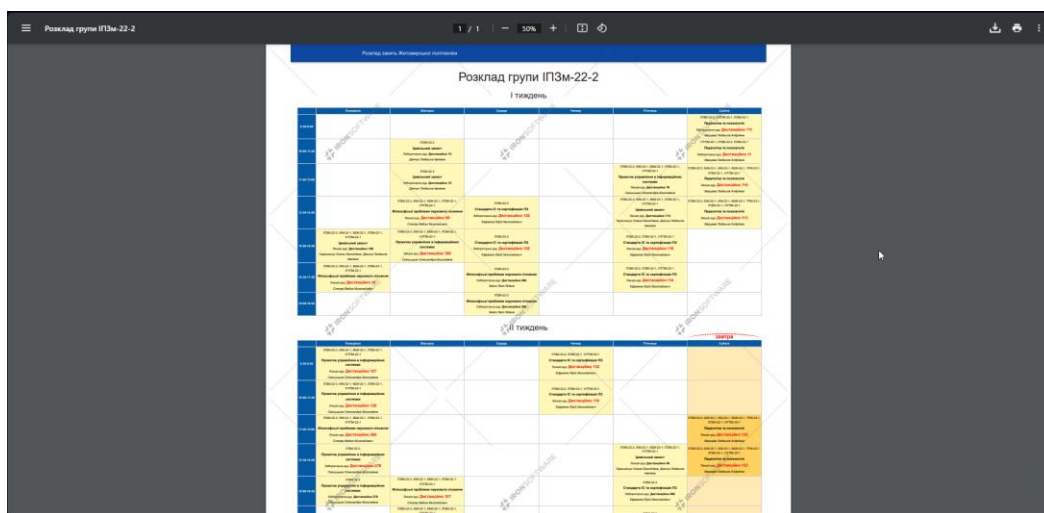


Рисунок 3.25 - Розклад на два тижні у pdf файлі

Далі перевіримо роботу розкладу заходів. Після відправки відповідної команди чат-бота він надсилає повідомлення з вибором дати, на яку ми хочемо побачити розклад заходів (рис. 3.26). Після того, як ми обираємо дату – чат-бот надсилає повідомлення з подіями на цей день. Якщо подій більше однієї, повідомлення має стрілочки для перемикавання між подіями. На рис. 3.27 зображено події, що було заплановано на 11.12.2023.

Тестування даного функціоналу пройшло успішно.

Далі перевіримо роботу команди «Журнал». Ця команда надсилає Excel таблицю з предметами та оцінками з лабораторних робіт з цих предметів. Результат виконання команди зображено на рис. 3.28. Зміст надісланого файлу зображено на рис. 3.29.

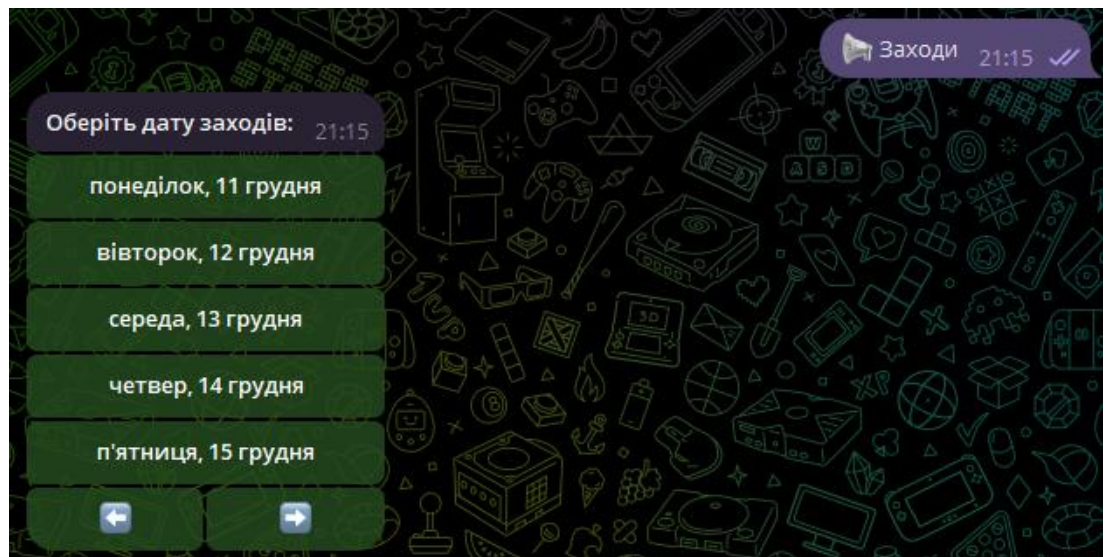


Рисунок 3.26 - Вибір дати для відображення заходів

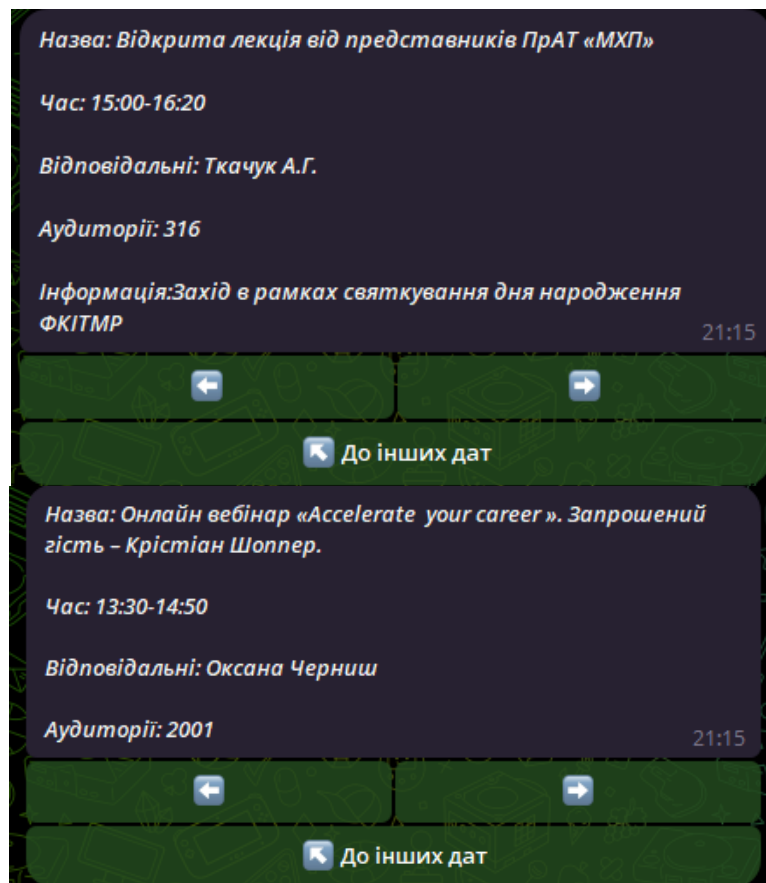


Рисунок 3.27 - Події заплановані на 11.12.2023

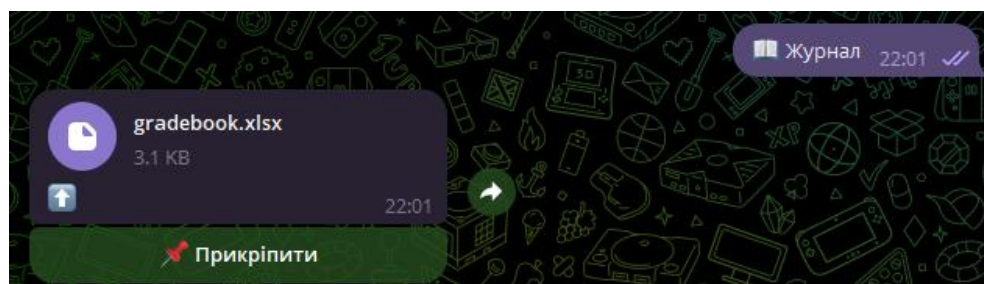


Рисунок 3.28 - Результат виконання команди «Журнал»

Савчук Іван Олександрович, ІПЗм-22-2				
Цивільний захист	9/10	4/5	6/6	3/5
Філософські проблеми наукового пізнання	4/5	3/5		
Проектне управління в інформаційних системах	11/15	2/2	7/10	
Стандарти ІС та сертифікація ПЗ	3/5	4/5	7/10	2/3
Педагогіка та психологія	6/8	5/5	9/10	

Рисунок 3.29 - Зміст файлу з оцінками

Тестування функціоналу пройшло успішно. Чат-бот відповідає поставленим до нього вимогам

Висновки до третього розділу

Під час виконання третього розділу кваліфікаційної роботи спочатку було описано структуру вебдодатка, описано його меню та сторінки. Вебдодаток містить головну сторінку, календар, панель керування, запити, список користувачів, налаштування.

Також, було описано команди розробленого чат-бота, структуру його меню та логіку роботи.

Було проведено тестування роботи вебдодатка та чат-бота. Усі тести пройшли успішно – дозволяє автоматично робити облік робочого часу та самостійно заповнювати інформацію про робочі години, створювати робочі плани, відслідковувати виконання плану на день або на місяць, а також планувати відпустки або лікарняні та вивантажувати звітність. Тести чат-бота в Telegram також пройшли успішно. Чат-бот надсилає розклад на різні періоди часу, оцінки з предметів та дозволяє переглядати список заходів та подій, що відбулись або відбудуться в університеті.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було реалізовано сервіс для автоматизації розсилання повідомлень та обліку робочого часу працівників університету.

В ході виконання першого розділу було виконано детальну постановку задачі для розробки сервісу для автоматизації розсилання повідомлень та обліку робочого часу працівників університету. Система складається з двох частин – вебдодатка для обліку робочого часу працівників та чат-бота для автоматичного надсилання повідомлень. Вебдодаток призначено для використання працівниками університету для обліку свого робочого часу. Також, адміністративний персонал зможе використовувати вебдодаток для контролю виконання роботи працівниками університету. Чат-бот призначений для використання студентами та викладачами університету для перегляду свого розкладу занять, а також для отримання інформації про свою успішність та майбутні події.

Провівши аналіз аналогів системи та виділивши переваги і недоліки розглянутого ПЗ, було сформовано критерій до розроблюваного вебдодатка та чат-бота. Вебдодаток має мати зручний та інтуїтивно зрозумілий інтерфейс, бути розрахованим на велику кількість персоналу, надавати можливість для вивантаження звітів. Чат-бот має бути закритим від зовнішніх користувачів, мати різні мови інтерфейсу, надавати розклад на різні проміжки часу, а також автоматично сповіщати студентів про заняття на день.

Для розробки системи було обрано такі інструментальні засоби: серверна частина вебдодатка для обліку робочого часу та чат-бота була розроблена засобами мови програмування C#, а саме фреймворку ASP.NET. Клієнтська частина вебдодатка розроблена за допомогою Java Script фреймворків React.js та Redux.

В другому розділі була створена структура бази даних та описано інструменти та технології для її використання. Завдяки фреймворку ASP.NET, були використані такі патерни як MVC та фабричний патерн, а також технології

і фреймворки, включаючи Entity Framework та GraphQL.

Після цього відбулося проектування і розробка алгоритмів для чат-бота. Чат-бот було вирішено реалізувати та розгорнути у месенджері Telegram. Були детально описані команди, які може виконувати чат-бот, і наведені приклади їх реалізації.

Для сервісу обліку робочого часу був розроблений вебдодаток. Клієнтська частина цього сервісу була реалізована за допомогою React, і описано як вона взаємодіє з серверною частиною через Redux та GraphQL.

Під час виконання третього розділу кваліфікаційної роботи спочатку було описано меню вебдодатка. Меню складається з таких сторінок: Головна, Календар, Панель керування, Мої відпустки, Запити, Команда, Налаштування. Було детально описано кожну зі сторінок. Також було описано меню та команди чат-бота

Було проведено тестування роботи вебдодатка та чат-бота. Усі тести пройшли успішно – система відповідає вимогам, поставленим до неї у першому розділі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. E. Malokhvii, V. Buhai, H. Molchanov, O. Chernykh. Analytical overview and comparison of modern JavaScript solutions for web application development. Control, navigation and communication systems, Poltava, Ukraine, 2021. <http://journals.nupp.edu.ua/sunz/article/download/2421/1891>.
2. Sri Lakshmi Vadlamani, Benjamin Emdon, Joshua Arts, Olga Baysal. Can GraphQL Replace REST? A Study of Their Efficiency and Viability. IEEE/ACM 8th International Workshop on Software Engineering Research and Industrial Practice, Madrid, Spain, June 04–04, 2021. https://olgabaysal.com/pdf/Vadlamani_SERIP2021.pdf.
3. Feng Yang, Sathiamoorthy Manoharan. A security analysis of the OAuth protocol. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, Victoria, BC, Canada, August 27–29, 2013. <https://doi.org/10.1109/PACRIM.2013.6625487>.
4. Toggl: Time Tracking Software, Project Planning, Hiring Tools [Електронний ресурс] – Режим доступу до ресурсу: <https://toggl.com>.
5. Time Tracking Software With Invoicing | Harvest [Електронний ресурс] – Режим доступу до ресурсу: <https://www.getharvest.com>.
6. Clockify - Time Tracking Software [Електронний ресурс] – Режим доступу до ресурсу: <https://clockify.me>.
7. RescueTime: Fully Automated Time Tracking Software [Електронний ресурс] – Режим доступу до ресурсу: <https://www.rescuetime.com>.
8. Time Tracking Software - Automated Time Tracker TimeCamp [Електронний ресурс] – Режим доступу до ресурсу: <https://www.timecamp.com>.
9. Manage your team's work, projects, & tasks online • Asana [Електронний ресурс] – Режим доступу до ресурсу: <https://asana.com>.
10. Розклад для студентів університету ім. Івана Франка. [Електронний ресурс] – Режим доступу до ресурсу: https://t.me/ZDU_bot.
11. Офіційний телеграм-бот УКД для пошуку розкладу [Електронний ресурс] – Режим доступу до ресурсу: https://t.me/KDU_scheduler_bot.

12. NUPP_Bot [Електронний ресурс] – Режим доступу до ресурсу: <https://t.me/NuppBot>.
13. Що таке Node.js та для чого він потрібен? [Електронний ресурс] – Режим доступу до ресурсу: <https://hyperhost.ua/info/uk/shho-take-nodejs-ta-dlya-cogo-vin-potriben>.
14. Ruby on Rails Explained [Електронний ресурс] – Режим доступу до ресурсу: <https://www.freecodecamp.org/news/ruby-on-rails-explained/>.
15. Django introduction [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction>.
16. What is PHP for Web Development and Why Should You Use It? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.netsolutions.com/insights/what-is-php/>.
17. Modern Web Application Development With Asp.Net [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@evincedvelop/modern-web-application-development-with-asp-net-1e7fc7b07cce>.
18. Minnick C. ReactJS Foundations Building User Interfaces with ReactJS AN APPROACHABLE GUIDE / Chris Minnick. – Hoboken, New Jersey: John Wiley & Sons, Inc., 2022. – 508 с.
19. What is Redux? A Complete Beginner's Guide [Електронний ресурс] – Режим доступу до ресурсу: <https://careerfoundry.com/en/blog/web-development/what-is-redux/>.
20. Putz J. Maximizing Asp.net: Real World, Object-oriented Development / Jeffrey Putz., 2005. – 309 с.
21. What is ASP.NET Core MVC? – The Complete Guide 2024 [Електронний ресурс] – Режим доступу до ресурсу: <https://positiwise.com/blog/introduction-asp-net-core-mvc>.
22. What is Entity Framework? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.entityframeworktutorial.net/entityframework6/what-is-entityframework.aspx>.

23. GraphQL API Integration In Asp.Net Core Application [Электронный ресурс] – Режим доступа до ресурсу: <https://www.learnmoreseekmore.com/2019/11/graphql-api-integration-in-aspnet-core.html>.

24. OAuth 2.0 [Электронный ресурс] – Режим доступа до ресурсу: <https://oauth.net/2/>.

ДОДАТКИ

Лістинг моделі User

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using TimeTracker.Enums;
using TimeTracker.Utils.SoftDelete;

namespace TimeTracker.Models;

public class User:SoftDeleteBase
{
    public int Id { get; set; }

    public static int FullTimeValue => 100;

    [Required] public string FirstName { get; set; } = string.Empty;

    [Required] public string LastName { get; set; } = string.Empty;

    [Required] public string Email { get; set; } = string.Empty;

    [Required] public string Password { get; set; } = string.Empty;

    [Required] public WorkType WorkType { get; set; }

    public bool? IsTwoStepAuthEnabled { get; set; }

    public string? PathId { get; set; }

    public string? PathSid { get; set; }
```

```
public int AvailableVacationDays { get; set; }
```

```
public string? RefreshToken { get; set; }
```

```
public DateTime? RefreshTokenExpiration { get; set; }
```

```
public bool IsEmailActivated { get; set; }
```

```
[Column(TypeName = "int")]
```

```
[EnumDataType(typeof(Permissions))]
```

```
public Permissions Permissions { get; set; }
```

```
public int HoursPerMonth { get; set; }
```

```
public int VacationDays { get; set; }
```

```
public DateOnly EmploymentDate { get; set; }
```

```
public List<Vacation> Vacations { get; } = new();
```

```
public List<UserApprover> Approvers { get; } = new();
```

```
public List<UserApprover> Senders { get; } = new();
```

```
public List<ApproverVacation> ApproverVacations { get; } = new();
```

```
public List<WorkedHour> WorkedHours { get; } = new();
```

```
public List<WorkPlan> WorkPlans { get; } = new();
```

```

public List<SickLeave> SickLeaves { get; } = new();
public User()
{
    VacationDays = 30;
    IsDeleted = false;
    EmploymentDate = DateOnly.FromDateTime(DateTime.Now);
}
}

```

ДОДАТОК Б

Форма входу в систему

```

import {LargeButton} from '@components/UI/Buttons/LargeButton';
import {PasswordInput} from '@components/UI/Inputs/PasswordInput';
import {TextInput} from '@components/UI/Inputs/TextInput';
import {Loader} from '@components/UI/Loaders/Loader';
import {InputTooltip} from '@components/UI/Tooltips/InputTooltip';
import {SubmitHandler, useForm} from 'react-hook-form';
import {useAppDispatch, useTypedSelector} from "../../hooks";
import {auth, login, loginSuccess, loginWithCode, verifyUserLogin} from
"../../redux";

import {H1, H5} from "../../Headings";
import "../../LoginForm.css";
import googleImg from '../../assets/images/search.png'
import githubImg from '../../assets/images/github.png'
import {useEffect, useState} from "react";
export type Inputs = {
    email: string
    password: string
}
export const LoginForm = () => {
    const dispatch = useAppDispatch();

```

```

const {loading, error,twoStepCodeStatus,userToken, user, verifiedUser,
currentAuth} = useTypedSelector(state => state.auth);
const [step, setStep] = useState<number>(0);
const [val,setVal] = useState<string>("");
const [showTwoStep, setShowTwoStep] = useState<boolean>(false);
const [data,setData] = useState<Inputs>({
  email:"",
  password:""
})
const stepMsg = ['Sign in', 'Enter the code']
const {
  register, handleSubmit,
  formState: {errors}, reset
} = useForm<Inputs>({
  mode: 'onBlur',
  defaultValues: {
    email: "",
    password: ""
  }
});
useEffect(() => {
  if(verifiedUser===null){
    return;
  }
  if(verifiedUser.isTwoStepAuthEnabled){
    setShowTwoStep(true);
    setStep(1);
  }else{
    dispatch(login(data));
  }
}

```

```

    }, [verifiedUser]);
    useEffect(() => {
      if(userToken!==""){
        setStep(0);
        setShowTwoStep(false);
        dispatch(loginSuccess(userToken))
      }
    }, [userToken]);
    const onSubmit: SubmitHandler<Inputs> = (data) => {
      dispatch(verifyUserLogin(data));
      setData(data);
    }
    function handleAuthClick(e: React.MouseEvent<HTMLAnchorElement>) {
      e.preventDefault();
      const authType = e.currentTarget.id;
      localStorage.setItem("authType", authType);
      window.location.href = `/to-external-auth?authType=${authType}`;
    }
    function authWithCode(){
      dispatch(loginWithCode({
        id:verifiedUser?.id!,
        code:val
      })))
    }
    const LoginFromFirstStep = () => {
      return (
        <>
        <div className="external-auth-wrapper" style={{width: "65%"}}>
          <a id="google" onClick={(e) => handleAuthClick(e)}
            className="external-auth"

```

```

      href="/to-external-auth?authType=google">
        <span>Continue with Google</span>
        <img style={{width: "25px", height: "25px"}}
src={googleImg} alt=""/>
      </a>
      <a id="github" onClick={(e) => handleAuthClick(e)}
className="external-auth"
      href="/to-external-auth?authType=github">
        <span>Continue with Github</span>
        <img style={{width: "25px", height: "25px"}} src={githubImg}
alt=""/>
      </a>
    </div>
    <div className="or">
      <span className="or-text">OR</span>
    </div>
    <form onSubmit={handleSubmit(onSubmit)} className="login-
form">
      <div className="login-inputs__wrapper">
        <TextInput name="email" placeholder="Enter your work email"
          register={register("email", {required: "Email can't be
empty!"})}
          errors={errors.email}/>

        <div className="password-input__wrapper">
          <PasswordInput name="password" placeholder="Password"
            register={register("password", {required: "Password
can't be empty!"})}
            errors={errors.password}/>

```

```

        <InputTooltip description="Forgot your password?"
url="/passwordRecovery"
        urlTitle="Click here"/>
    </div>
</div>

<div className="tooltip-wrapper__bold" style={{display:
'none'}}>
    <InputTooltip description="Don't have an account?" url="/auth"
urlTitle="Get started"/>
    </div>
    <div className='submit-wrapper'>
        <LargeButton type="submit" value="Login"/>
    </div>
</form>
</>
);
}

return (
    <div className="login-form__wrapper">
        <H1 value={stepMsg[step]}/>
        <div className="login-form__messages-wrapper" style={{display:
`${error ? "flex" : "none"}`}}>
            {loading ? <Loader/> : ""}
            <H5 value={error ? error : ""}/>
        </div>
        {showTwoStep?<div className="login-form"
style={{gap:"15px",marginTop:"20px"}}>
            <input onChange={e=>{

```

```

        setVal(e.target.value);
    }} value={val} className="text-input" type="text"
placeholder="You code"/>
    <LargeButton handleClick={authWithCode} type="submit"
value="Login"/>
    </div>:<LoginFromFirstStep />}
</div>
);
}

```

ДОДАТОК В

Лістинг мутації «WorkedHourMutations»

```

using AutoMapper;
using GraphQL;
using GraphQL.Types;
using TimeTracker.Absctration;
using GraphQL.Validation;
using TimeTracker.GraphQL.Types.InputTypes.WorkedHourInput;
using TimeTracker.Models;
using TimeTracker.Models.Dtos;
using TimeTracker.Utils.Errors;
using TimeTracker.Enums;
namespace TimeTracker.GraphQL.Mutations;
public sealed class WorkedHourMutations : ObjectGraphType
{
    public WorkedHourMutations(IUnitOfWorkRepository uow, IMapper
mapper)
    {
        Field<WorkedHour>("create")
            .Argument<WorkedHourInputType>("workedHour")
            .ResolveAsync(async ctx =>

```

```

{
    var wh = ctx.GetArgument<WorkedHourInputDto>("workedHour");
    if (wh.StartDate > DateTime.UtcNow
        || wh.EndDate > DateTime.UtcNow)
    {
        throw new ValidationException("Can't create worked hour for future");
    }
    var exists = await uow.GenericRepository<WorkedHour>()
        .FindAsync(p => p.UserId == wh.UserId
            && p.StartDate < wh.EndDate
            && wh.StartDate < p.EndDate);
    if (exists is not null)
        throw new ValidationException("Worked hours intersect!");
    var created = await uow.GenericRepository<WorkedHour>()
        .CreateAsync(mapper.Map<WorkedHour>(wh));
    await uow.SaveChangesAsync();
    return created;
});

Field<int>("delete")
    .Argument<int>("id")
    .ResolveAsync(async ctx =>
    {
        var id = ctx.GetArgument<int>("id");
        var wh = await
uow.GenericRepository<WorkedHour>().FindAsync(u => u.Id == id)
        ?? throw new ValidationException("Record not found"); // Temp error
        var result = await
uow.GenericRepository<WorkedHour>().DeleteAsync(wh);
        await uow.SaveChangesAsync();
        return result ? id : throw new ValidationException("Failed to delete");
    });

```

```

    });
    Field<WorkedHour>("update")
        .Argument<UpdateWorkedHourInputType>("workedHour")
        .ResolveAsync(async ctx =>
        {
            var wh =
ctx.GetArgument<WorkedHourUpdateDto>("workedHour");
            var currentValue = await
uow.GenericRepository<WorkedHour>().FindAsync(w => w.Id == wh.Id)
                ?? throw new ValidationException("Record not found"); // Temp
error
            if (wh.StartDate > DateTime.UtcNow || wh.EndDate >
DateTime.UtcNow)
            {
                throw new ValidationException("Can't create worked hour for future");
            }
            var exists = await uow.GenericRepository<WorkedHour>()
                .FindAsync(p => p.Id != currentValue.Id
                    && p.UserId == currentValue.UserId
                    && p.StartDate < wh.EndDate
                    && wh.StartDate < p.EndDate);
            if (exists is not null)
                throw new ValidationException("Worked hours intersect!");
            var updated = await uow.GenericRepository<WorkedHour>()
                .UpdateAsync(mapper.Map(wh, currentValue));
            await uow.SaveAsync();
            return updated;
        });
    }
}

```