Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему:

Розробка автоматизованої системи управління проектами для оптимізації завдань на підприємстві

Виконав: студент 2 курсу, групи 1АКІТР-23м
спеціальності 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка
(шифр і назва спеціальності)
Роман ТИТОМИР
(Ім`я ПРІЗВИЩЕ)

Керівник: к.т.н., доцент кафедри АІІТ
Марія БАРАБАН
(Ім`я ПРІЗВИЩЕ)
«16» грудня 2024 р.

Опонент: к.т.н., доцент кафедри КН
Володимир ОЗЕРАНСЬКИЙ
(Ім`я ПРІЗВИЩЕ)
«16» грудня 2024 р.

Допущено до захисту
зав. кафедри АІІТ
Олег БІСІКАЛО
«16» 12 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти другий (магістерський)
Галузь знань - 17 - Електроніка, автоматизація та електронні комунікації
Спеціальність - 174 - Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Освітньо-професійна програма - Інтелектуальні комп'ютерні системи

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
"19" жовтня 2024 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Титомиру Роману Руслановичу

1. Тема роботи: «Розробка автоматизованої системи управління проектами для оптимізації завдань на підприємстві»
керівник роботи: Марія БАРАБАН, к.т.н. доцент кафедри АІІТ
затверджено наказом закладу вищої освіти від "17" вересня 2024 року №310
2. Термін подання студентом роботи 16.12.2024 р.
3. Вихідні дані до роботи:
   - операційна система Windows;
   - встановлений Django версії 3.0 та новіше;
   - мова програмування Python;
   - обсяг оперативної пам'яті: не менше 4 ГБ;
   - обсяг дискового простору для зберігання даних не менше 10 ГБ;
   - середній обсяг даних, які обробляються щодня до 1 ГБ;
   - частота процесора не менше 2 ГГц, 2 або більше ядер.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): дослідження предметної області, перелік літературних джерел, розробка програмного забезпечення, дослідження і порівняння технологій, фреймворків, мов програмування, створення моделей.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
Er діаграми сутностей зв'язків у базі даних, проміжні результати, коди запитів, результат виконаної сторінки.

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
4	Козловський В.О. – к.е.н., професор кафедри ЕПВМ	18.09.24	07.11.24

7. Дата видачі завдання 18.09.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз предметної області	08.10.2024	Виконано
2	Вибір оптимальних інформаційних технологій	15.10.2024	Виконано
3	Вибір мови програмування та середовища розробки	20.10.2024	Виконано
4	Програмна реалізація	22.10.2024	Виконано
5	Експериментальна перевірка роботи програми	04.11.2024	Виконано
6	Оформлення матеріалів до захисту МКР	05.11.2024	Виконано

Студент _____ Роман ТИТОМИ
(підпис) (Ім`я ПРІЗВИЩЕ)

Керівник роботи _____ Марія БАРАБАН
(підпис) (Ім`я ПРІЗВИЩЕ)

ЗМІСТ

АНОТАЦІЯ

УДК 004.4

Титомир Р. Р. Розробка автоматизованої системи управління проектами для оптимізації завдань на підприємстві. Магістерська кваліфікаційна робота зі спеціальності 174 – Автоматизація, комп’ютерно-інтегровані технології та робототехніка, освітня програма – Інтелектуальні комп`ютерні системи управління. Вінниця: ВНТУ, 2024. 130 с.

На укр. мові. Бібліогр.: 32 назв; Рисунок: 19; табл. 7.

Робота присвячена розробці автоматизованої системи управління проектами, що сприяє оптимізації роботи над завданнями на підприємстві. Основними компонентами системи є: Kanban-дошка для візуалізації та управління завданнями через веб-інтерфейс, Telegram бот для мобільного доступу до завдань і проектів, а також REST API, який забезпечує взаємодію між клієнтами, сервером і базою даних.

У процесі розробки використано сучасні технології - Django Rest Framework, PostgreSQL, Python-telegram-bot, що гарантують надійність і масштабованість. Проведено тестування функціональності системи, включаючи API, веб-додаток і бот, що підтвердило її працездатність.

Результатом роботи є готовий до впровадження інструмент для автоматизації управління проектами, який підвищує продуктивність команди, забезпечує зручність та ефективність у роботі з завданнями. Система підходить для малого та середнього бізнесу.

Ключові слова: автоматизована система, управління, Kanban, Django, веб-додаток, Python, API, телеграм бот.

ABSTRACT

Development of an automated project management system for optimization of tasks at the enterprise. Master's qualification work in specialty 174 - Automation, computer-integrated technologies and robotics, educational program - Intelligent computer control systems. Vinnytsia: VNTU, 2024. 130 p.

In Ukrainian. Bibliography: 32 titles; Figures: 19; Table 7.

The work is devoted to the development of an automated project management system that helps to optimize the work on tasks at the enterprise. The main components of the system are: Kanban board for visualization and management of tasks through a web interface, Telegram bot for mobile access to tasks and projects, and REST API that provides interaction between clients, server and database.

In the development process, we used modern technologies such as Django Rest Framework, PostgreSQL, Python-telegram-bot, which guarantee reliability and scalability. The system functionality, including API, web application, and bot, was tested, which confirmed its efficiency.

The result is a ready-to-implement project management automation tool that increases team productivity, provides convenience and efficiency in working with tasks. The system is suitable for small and medium-sized businesses.

Keywords: automated system, management, Kanban, Django, web application, Python, API, telegram bot.

ВСТУП

Актуальність теми. В сучасних умовах глобалізації та цифрової трансформації підприємства різного масштабу потребують ефективних інструментів для управління проектами. Впровадження автоматизованих систем управління проектами з розподілом завдань стає важливим елементом для досягнення ефективності, прозорості та оптимізації роботи команди. Більшість проектів мають складну структуру з великою кількістю задач, що вимагає чіткого планування, контролю та комунікації між учасниками. Традиційні методи управління проектами часто не здатні забезпечити достатній рівень координації та управління, що призводить до втрати часу, перевищення бюджету та зниження продуктивності [1].

Одним з ефективних рішень цієї проблеми є використання автоматизованих систем управління проектами, які дозволяють централізовано планувати та контролювати виконання завдань, відстежувати прогрес, розподіляти ресурси та забезпечувати взаємодію між учасниками проекту в реальному часі. Особливо актуальною така система є для великих підприємств, де одночасно виконується кілька проектів, а команди можуть бути розподілені географічно.

Сучасні автоматизовані системи також інтегруються з популярними месенджерами, такими як Telegram, що надає можливість оперативної комунікації та спрощує взаємодію між учасниками проекту. Використання ботів у месенджерах дозволяє автоматизувати рутинні процеси, такі як призначення завдань, відправлення нагадувань та формування звітів, що значно підвищує ефективність роботи команди.

Метою роботи є покращення ефективності управління проектами завдяки використанню розробленої системи автоматизації для оптимізації процесів розподілу, контролю та виконання завдань на підприємстві. Ця система дозволяє підвищити прозорість управлінських процесів, забезпечує своєчасне виконання завдань та покращує взаємодію між учасниками проектів.

Для досягнення мети потрібно вирішити наступні задачі:

- аналіз методів розробки веб-сайтів;

- вибір технологічних засобів для створення веб-сторінки;
- виконати проєктування моделі та розробити послідовність його розробки;
- розробити веб-сайту канбан-дошки;
- розробити бота на Телеграм.

Об’єктом дослідження є процес обробки та перетворення даних для візуалізації завдань та залучення кожного члена команди шляхом використання канбан-дошки на Python і бота на Телеграм.

Предмет дослідження – методи розробки веб-сайту канбан-дошки на Python та їх компонентів, розробка бота на Телеграм.

Методи дослідження. Аналіз для декомпозиції предметної області, синтез для розробки клієнт-серверної частини веб-сайту.

Науково-технічний результат полягає у тому, що на основі мова програмування Python та фреймворку Django розроблено веб-додаток канбан-дошки для управління проектами та організації роботи команди.

Практична цінність роботи полягає в розробці алгоритмічних та програмних засобів, які реалізують веб-додаток створення канбан-дошки, що може працювати на усіх доступних платформах: у веб-браузері, на IOS та Android.

Апробація наукової роботи: результати досліджень апробовано на LII науково-технічній конференції професорсько-викладацького складу, співробітників та студентів ВНТУ (березнь 2023 року) [2] та на LIII Всеукраїнській науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації (2024) [3].

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Сучасні підприємства все частіше стикаються з необхідністю автоматизації управлінських процесів для досягнення високого рівня ефективності. Однією з ключових сфер, що вимагає вдосконалення, є управління проектами. Успішне планування, координація та контроль над виконанням завдань є важливими складовими досягнення цілей підприємства. У зв'язку з цим все більш актуальними стають автоматизовані системи управління проектами з розподілом завдань.

Цей розділ присвячено аналізу предметної області та обґрунтуванню необхідності розробки автоматизованої системи управління проектами, що забезпечує розподіл завдань і контроль над їх виконанням на підприємстві [4].

1.1 Сутність управління проектами

Управління проектами — це дисципліна, яка спрямована на організацію, планування, управління ресурсами та виконання робіт для досягнення певних цілей. Вона включає широкий спектр процесів, що охоплюють управління часом, вартістю, ресурсами, ризиками, якістю, комунікаціями та змінами. Ефективне управління проектами дозволяє підприємствам досягати своїх стратегічних цілей, зменшувати ризики й уникати непередбачених затримок у виконанні завдань.

Основні процеси управління проектами включають планування проекту. Планування проекту — це процес визначення цілей, завдань і заходів, необхідних для досягнення кінцевого результату проекту. Цей процес включає створення чіткого плану робіт, визначення ключових етапів (міток) проекту та визначення ресурсів (людських, фінансових, технічних), необхідних для їх досягнення [5].

Основними складовими планування є:

- Мета проекту визначення конкретних результатів, які повинні бути досягнуті.

- Структура розподілу робіт (WBS) деталізація проекту на підзадачі і завдання, які можуть бути легко виконані та контрольовані.
- Оцінка термінів: визначення часу, необхідного для кожного етапу проекту, та формування графіку виконання.
- Оцінка ресурсів визначення кількості та типу ресурсів (люди, технології, фінанси), необхідних для виконання проекту.

Наступний процес розподіл завдань. Розподіл завдань включає делегування відповідальності за виконання конкретних завдань або етапів проекту відповідним членам команди. Для цього керівник проекту повинен володіти інформацією про компетенції учасників проекту, їхній робочий графік і доступність ресурсів.

Розподіл завдань передбачає:

- Призначення завдань конкретним виконавцям на основі їхніх навичок і досвіду.
- Визначення пріоритетів завдань та встановлення чітких термінів їх виконання.
- Забезпечення належної координації між різними членами команди для досягнення узгоджених цілей.

Контроль за виконанням проекту полягає у постійному відстеженні прогресу та оцінці відповідності результатів до плану. Це важливий процес для своєчасного виявлення можливих проблем або відхилень від запланованих термінів чи бюджету, що дозволяє вчасно вжити коригувальних заходів.

Ключові аспекти контролю та моніторингу:

- Оцінка прогресу порівняння фактичних результатів із запланованими показниками.
- Управління змінами визначення і впровадження змін у проект у разі необхідності.
- Відстеження ресурсів моніторинг використання ресурсів і корекція їхнього розподілу.
- Розробка звітів регулярне складання звітів для команди проекту та керівництва підприємства.

Завершення проекту. Фаза завершення проекту включає перевірку виконаних завдань, оцінку досягнутих результатів та підбиття підсумків. На цьому етапі проводиться аналіз того, наскільки проект досяг своїх цілей, оцінюється ефективність роботи команди, а також формуються рекомендації для покращення майбутніх проектів.

Основні завдання етапу завершення:

- Аналіз виконання плану та досягнення цілей проекту.
- Підготовка фінального звіту про результати проекту.
- Оцінка роботи команди та проведення фінальних зустрічей.
- Закриття проекту і передача результатів замовнику або внутрішнім відділам підприємства.

Управління проектами вимагає від керівників і команд проекту постійного вдосконалення підходів і методик роботи. У сучасних умовах часто застосовуються гнучкі методології управління проектами, такі як Agile або Scrum, які дозволяють швидше реагувати на зміни і забезпечують більш ефективну координацію між учасниками проекту. Це особливо важливо для великих і складних проектів, де виникає необхідність в частих коригуваннях плану через зміну зовнішніх або внутрішніх умов.

Таким чином, сутність управління проектами полягає у комплексному підході до координації діяльності та ресурсів, що сприяє досягненню стратегічних цілей підприємства. Автоматизація цього процесу значно підвищує його ефективність, знижуючи вплив людського фактору та полегшуючи управління великими обсягами інформації й завдань [6].

1.2 Класичні методи розробки

Класичні методи розробки веб-додатків включають в себе використа- ння стандартних мов та інструментів веб-розробки для створення фронтенду та бекенду додатка. Основні складові класичних методів розробки веб-додатків включають

HTML, CSS, JavaScript, серверні мови програмування та бази даних. HTML (HyperText Markup Language): використовується для створення структури та розмітки веб-сторінок. Він визначає, які елементи та контент будуть відображені на сторінці [5].

CSS (Cascading Style Sheets): використовується для задання зовні- шнього вигляду веб-сторінок, таких як кольори, шрифти, розміри, розташування елементів тощо. Він дозволяє розділити стиль від структури сторінки.

JavaScript: є мовою програмування, яка дозволяє надавати динамічну функціональність веб-сторінкам. Він використовується для взаємодії з користувачем, маніпуляції DOM (Document Object Model), обробки подій та виконання асинхронних запитів до сервера.

Серверні мови програмування: для реалізації бекенду веб-додатка мо- жуть використовуватись різні мови програмування, такі як PHP, Python, Ruby, Java, C тощо. Ці мови дозволяють обробляти запити користувачів, взаємодіяти з базою даних, обробляти бізнес-логіку та надавати відповіді на запити.

Бази даних: для зберігання та управління даними, які використовую- ться в веб-додатку, можуть використовуватись реляційні бази даних, такі як MySQL, PostgreSQL, або NoSQL бази даних, такі як MongoDB, Firebase. Бази даних забезпечують постійне зберігання і доступ до необхідної інформації.

Ці класичні методи розробки веб-додатків добре підходять для простих проектів, де немає потреби в складних фреймворках або специфічних інструментах. Вони є широко поширеними та мають велику спільноту розробників, що сприяє доступності допомоги та ресурсів для вирішення проблем [6].

1.2.1 Статичні та динамічні веб-сторінки

Статичні та динамічні веб-сторінки є двома різними підходами до веб-розробки, які використовуються для створення та відображення веб- контенту [7].

Статичні веб-сторінки:

- Статичні веб-сторінки складаються з фіксованого HTML, який вже наперед визначає структуру та вміст сторінки.
- Ці сторінки зберігаються на сервері у вигляді статичних файлів і не змінюються, якщо не внесені ручні зміни до HTML-коду.
- При відкритті статичної сторінки веб-браузер просто відображає вміст, який знаходиться у файлі, без необхідності обробки додаткового серверного коду або звернень до бази даних.
- Статичні веб-сторінки підходять для простих проектів, де змінюваний контент мінімальний, наприклад, промо-сторінки, веб-брошури, особисті блоги тощо.
- Динамічні веб-сторінки генеруються на льоту за допомогою серверного коду та бази даних;
- Серверний код, написаний на певній мові програмування (напри- клад, PHP, Python, Ruby), обробляє запити користувачів та генерує HTML-код в залежності від отриманих даних.

Обирання між статичними та динамічними веб-сторінками залежить від вимог та потреб проекту. Прості проекти зазвичай використовують статичні сторінки, тоді як більш складні додатки вимагають динамічного підходу для забезпечення більшої функціональності та взаємодії з користувачем. Порівняльні характеристики описуються в таблиці 1.1.

Таблиця 1.1 – порівняння статичних та динамічних веб-сторінок.

Статичні	Динамічні
Надають усім відвідувачам однакову інформацію	Вигляд і вміст сайту автоматично змінюється залежно від контексту, дій та уподобань користувача, часу та інших факторів
Веб-сторінки зберігаються веб-сервері в тому самому вигляді, у якому передаються браузеру	Веб-сторінки генеруються з використанням програмного коду та інформації з бази даних і передаються браузеру
Оновлюються періодично власниками вручну	Можуть оновлюватися часто, не обов’язково фахівцями і власниками

1.2.2 Фреймворки

Фреймворки веб-розробки є набором інструментів, бібліотек та правил, які допомагають розробникам створювати веб-додатки швидше та ефективніше. Вони надають структуру та організацію процесу розробки, а також пропонують готові рішення для типових завдань веб-розробки [8].

Різниця між бібліотекою та фреймворком. Бібліотека включає набір функцій, які призначені для вирішення конкретних завдань у певній області. Наприклад, можуть існувати бібліотеки для роботи з датою та часом, випадковими числами, файловою системою, HTTP-запитами та се- ріалізацією у різних форматах. Однак, бібліотека сама по собі не визначає структуру програми. При використанні бібліотеки користувач вирішує, які функції викликати і коли. У свою чергу, фреймворк визначає архітектуру програми і забезпечує взаємодію між її компонентами. Він включає різні бібліотеки і використовує їх для створення основи програми. При роботі з фреймворком відбувається зміна управління. Не користувач встановлює, коли викликати функції, а фреймворк сам визначає цей порядок.

Веб тримається на трьох китах: HTML, CSS та JavaScript. HTML визначає структуру веб-сторінки, CSS відповідає за її оформлення, а JavaScript забезпечує взаємодію з користувачем.

Веб-браузери «розуміють» лише ці три технології. Проблема в тому, що і ці технології, і браузери розвиваються окремо одне від одного. Щоб веб-сторінка виглядала однаково у всіх браузерах та однаково взаємодіяла з користувачами, під час написання коду потрібно врахувати особливості різних браузерів. Тож з’явилися бібліотеки. Вони являли собою набори функцій для вирішення типових завдань роботи з HTML-кодом сторінки. Прикладом є бібліотека JavaScript jQuery. Вона спрощує аналіз HTML-документа й керування його вмістом, обробку подій, анімацію та використання Ajax.

Згодом почали з’являтися бібліотеки для HTML і CSS. За допомогою цих бібліотек можна, наприклад, розбити сторінку на кілька колонок або закріпити заголовок. Це типові завдання, і їх краще вирішити один раз, а потім повторно

використовувати готовий код. Прикладом такого фреймворку є Bootstrap, створений компанією Twitter. Це фронтенд-фреймворк. Він працює на стороні клієнта – у браузері.

Класифікація фреймворків:

Бекенд-фреймворки. Працюють на боці сервера й забезпечують роботу програми веб або веб-сайту. За допомогою бекенд фреймворків можна створювати базові сторінки та форми, перевіряти вхідні дані та формувати вихідні. Прикладами бекенд-фреймворків є Laravel і CakePHP для PHP, Django і Flask для Python, Express.js для Node.js і Ruby on Rails для Ruby.

Фронтенд фреймворки. Відповідають за зовнішній вигляд вебсторінки. Забезпечують однакове відображення веб-сторінки у всіх браузерах. Спрощують роботу зі стилями, анімацією та дозволяють створювати привабливі інтерактивні інтерфейси. Приклади фронтенд фреймворкві є Vue.js, Bootstrap, Foundation, Angular, React.

Фуллстек фреймворки. Ці веб-фреймворки працюють і на боці клієнта, і на боці сервера. Прикладами фуллстек-фреймворків є Meteor, Next.js і Nuxt.

Основні переваги використання фреймворків веб-розробки включають:

- надають готові рішення та шаблони, що допомагають зменшити час розробки. Вони пропонують структуру проекту, задають правила та конвенції, що сприяють швидкому початку роботи;
- мають вбудовані механізми безпеки, які допомагають уникнути за- гроз, таких як хакерські атаки, вразливості, SQL-ін’єкції та інші. Вони надають вбудовані інструменти для автоматичного запобіга- ння таких проблем;
- надають механізми для масштабування веб-додатків. Вони допомагають забезпечити ефективну роботу з базами даних, кешуванням, обробкою запитів, що дозволяє легко розширювати та масштабувати проекти з ростом навантаження;
- надають інструменти для автоматичного тестування веб-додатків;
- це допомагає розробникам перевірити правильність роботи коду, виявити та виправити помилки швидше та ефективніше;

- мають активну спільноту розробників, яка готова надати допомогу, поради та ресурси. Також існує велика кількість документації, підручників та онлайн-ресурсів, що полегшують вивчення та використання фреймворків.

Недоліки:

- Фреймворк обмежує вашу свободу. Він диктує свої правила: угоди щодо іменування, структуру каталогів, накладає обмеження на функції, доступні у шаблонах сторінок тощо;
- На відміну від CMS, веб-фреймворки містять лише основні компо- ненти логіки, і для створення готового веб-сайту потрібно писати багато додаткового коду;
- Оскільки фреймворки надають безліч функцій і висувають безліч вимог, їх вивчення часто потребує значного часу.

1.2.3 Інтеграція ботів, месенджерів та інших систем

Інтеграція ботів, месенджерів та інших систем є важливим елементом для створення ефективного інструментарію управління проектами. Цей розділ розглядає методи та стратегії зв'язку між різними інструментами з метою полегшення комунікації та управління завданнями.

API (Application Programming Interface). Багато месенджерів та платформ ботів надають API для взаємодії з їхніми сервісами. Розробники можуть використовувати ці API для створення власних рішень та інтеграції ботів [9].

Вебхуки (Webhooks). Механізм, що дозволяє автоматично відправляти повідомлення або сповіщення в інші системи під час виникнення певних подій. Це ефективний спосіб отримувати оновлення в реальному часі [9].

Месенджери та Платформи - Telegram, Slack, Microsoft Teams, тощо: Багато месенджерів мають власні API для створення ботів та інтеграції з іншими сервісами.

Месенджерські платформи для розробників: Наприклад, Bot Framework від Microsoft або Bot API від Telegram, які дозволяють створювати універсальних ботів, що можуть працювати на різних платформах.

Інтеграція із системами управління проектами Trello, Jira, Asana, тощо. Багато інструментів для управління проектами також мають API для інтеграції з ботами та іншими системами.

Деякі компанії надають готові застосунки або додатки для інтеграції своїх продуктів з різними месенджерами та ботами.

Використання інтерфейсів для роботи з ботами UI для налаштування та керування. Розробники можуть створити графічні інтерфейси для користувачів, які дозволяють налаштовувати роботу ботів та інтеграцію з іншими системами.

Zapier, Integromat, Microsoft Power Automate: Ці платформи дозволяють створювати автоматизовані задачі між різними сервісами без програмування, шляхом налаштування правил та умов.

OpenAPI, GraphQL - використання стандартів інтеграцій може спростити розробку та підтримку, особливо при створенні універсальних інтерфейсів для ботів.

Завдяки цим методам і стратегіям можна створити потужний та гнучкий механізм інтеграції ботів, месенджерів та інших систем, що дозволяє командам ефективно управляти проектами та забезпечувати спрощення комунікацій та робочих процесів.

1.3 Боти зі штучним інтелектом

Боти зі штучним інтелектом (ШІ) відкривають нові можливості для покращення ефективності управління проектами, забезпечуючи автоматизацію, аналіз та персоналізовані рішення [10].

Автоматизовані чат-боти можуть використовувати розпізнавання мови для взаємодії з користувачами через чат, надавати інформацію, аналізувати завдання та відповідати на запитання.

Автоматична обробка запитань і відповідей. Боти можуть взаємодіяти з командою, автоматично обробляючи та відповідаючи на стандартні запитання. Прогнозування термінів завдань. Боти можуть використовувати аналіз даних та алгоритми для прогнозування термінів виконання завдань та подачі рекомендацій для оптимізації графіків проекту. Аналіз завдань та продуктивності. ШІ може допомагати в аналізі даних про виконання завдань, роблячи висновки щодо продуктивності та ідентифікації можливих вдосконалень.

Самонавчання та адаптація допоможе навчати ШІ на основі попередніх даних та досвіду використання, адаптуючись до змін у проекті та взаємодії з користувачами. Боти можуть виконувати рутинні завдання, такі як нагадування прострочених термінів, автоматична організація зустрічей та ведення статистики.

Автоматизований пошук даних - боти можуть швидко знаходити необхідну інформацію в базі даних проекту та надавати користувачам актуальні дані. ШІ може використовувати алгоритми для класифікації та фільтрації інформації, забезпечуючи користувачам лише необхідні дані.

Штучний інтелект в ботах додає рівень інтелекту та адаптивності до управління проектами. Вони можуть реагувати на зміни у проекті, аналізувати дані для прийняття рішень та забезпечувати ефективне взаємодії з користувачами.

Приклади ботів з штучним інтелектом в управлінні проектами:

- Zoom.ai. Використовує ШІ для надання асистентських послуг для планування зустрічей, автоматичної відправки запрошень та взаємодії з календарем.

  Функції:

  - Автоматичне планування зустрічей.
  - Взаємодія з Google Calendar та Outlook.
  - Розпізнавання мови для взаємодії з користувачами.

- AsanaBot для Slack. Інтегрується з Slack та використовує ШІ для полегшення управління завданнями та проектами безпосередньо в чаті.

  Функції:

  - Створення та призначення завдань з Slack.
  - Сповіщення про зміни в проектах.

- Відстеження статусу завдань.

- ChatGPT для управління проектами. ШІ-модель ChatGPT може бути використана для створення чат-бота для управління проектами, взаємодії з користувачами та аналізу потреб команди.

  Функції:

  - Відповіді на запитання користувачів щодо проектів.
  - Надання рекомендацій з управління завданнями.
  - Здатність навчатися на основі історії взаємодії.

- Wit.ai. Використовується для створення ботів із розумінням природної мови, що можуть інтегруватися з платформами для управління проектами.

  Функції:

  - Розпізнавання та інтерпретація текстових запитань.
  - Створення призначень та завдань на основі команд користувачів.
  - Здатність адаптуватися до нових команд та термінів.

- Talla. Використовує ШІ для автоматизації завдань управління проектами, включаючи створення звітів, призначення завдань та організацію комунікації в команді.

  Функції:

  - Автоматизоване створення та оновлення завдань.
  - Здатність відповідати на питання користувачів щодо проектів.
  - Аналіз даних для надання рекомендацій.

Ці приклади демонструють варіативність застосування ботів зі штучним інтелектом у сфері управління проектами, від чат-ботів для комунікації до ботів, які використовують аналітичні та прогнозуючі здібності.

1.4 Переваги використання автоматизованих систем управління проектами

На сьогодні існує безліч програмних рішень для автоматизації управління проектами, які охоплюють різноманітні потреби підприємств. Ці системи

допомагають спростити процеси планування, контролю і виконання проектів, забезпечуючи менеджерів інструментами для ефективної координації дій та моніторингу роботи. Одними з найпоширеніших платформ є Trello, Jira, Asana, Microsoft Project та інші. Кожна з них має свої переваги і особливості, а також різний рівень функціональних можливостей в залежності від типу та масштабу проектів.

Однією з найпопулярніших систем управління проектами є Trello. Це інструмент, що базується на принципах канбан-дошки, який дозволяє користувачам візуально відстежувати стан завдань та проектів. У Trello завдання представляються у вигляді карток, які можна переміщувати між різними статусами, такими як «Заплановано», «В роботі», «Виконано». Основною перевагою Trello є його простота у використанні та можливість швидкої адаптації під будь-який тип проекту. Завдяки цьому Trello користується популярністю у невеликих командах або при управлінні простими проектами, де немає потреби у складній аналітиці або глибокому контролі за ресурсами [11].

Інша популярна система — Jira, яка спеціалізується на управлінні проектами в сфері розробки програмного забезпечення. Вона є більш складною і багатофункціональною платформою, яка дозволяє керувати завданнями, відслідковувати помилки, планувати спринти, а також аналізувати ефективність роботи команди. Jira є невід'ємною частиною роботи багатьох команд, що працюють за методологіями Agile та Scrum, адже її функціонал дозволяє детально керувати кожною фазою розробки, від постановки завдань до тестування і випуску продукту. Окрім цього, Jira надає гнучкі можливості для інтеграції з іншими інструментами, що використовуються в IT-компаніях, такими як Git, Confluence та інші.

Asana — ще одна популярна система управління проектами, яка орієнтована на полегшення командної роботи. Asana дозволяє створювати завдання, призначати відповідальних осіб, встановлювати дедлайни та пріоритети. Інтерфейс Asana нагадує комбінацію списку справ і канбан-дошки, що робить її гнучкою у використанні для різних типів проектів. Однією з головних особливостей Asana є можливість створення проєктних цілей, що дозволяє більш структуровано планувати проект і забезпечувати досягнення кінцевого результату. Система пропонує багатий

набір функцій для моніторингу та аналізу прогресу, що робить її популярною серед середніх і великих компаній.

Ще одним важливим інструментом є Microsoft Project — професійна система для планування та управління проектами, що часто використовується у великих корпораціях і на складних проектах. Microsoft Project пропонує потужні функції для розробки детальних планів проектів, управління ресурсами, розподілу бюджетів та контролю термінів. Однією з ключових особливостей Microsoft Project є можливість створення діаграм Ганта — візуального інструменту для відображення взаємозв'язку між завданнями проекту та їхніх термінів. Ця система також добре інтегрується з іншими продуктами Microsoft, що робить її зручною для організацій, які вже використовують екосистему Microsoft.

Серед інших популярних систем варто згадати Basecamp, який відомий своїм простим інтерфейсом та широкими можливостями для організації командної роботи. Monday.com — це ще одна платформа, яка надає потужний інструментарій для візуалізації процесів і управління завданнями на рівні як малих, так і великих проектів. Wrike — інструмент, який поєднує можливості управління проектами з аналітичними функціями, що дозволяє детально відстежувати прогрес проекту і приймати обґрунтовані рішення на основі даних.

Отже, ринок пропонує широкий вибір систем для автоматизації управління проектами, кожна з яких може бути ефективною в залежності від специфіки підприємства та характеру проектів. Проте всі ці системи об'єднує одна важлива особливість — вони дозволяють значно підвищити продуктивність команд, забезпечують кращу організацію роботи, полегшують контроль за виконанням завдань і сприяють досягненню високих результатів в оптимальні строки. Вибір конкретної системи залежить від потреб підприємства, масштабів проектів, кількості залучених учасників, а також рівня складності завдань, які необхідно вирішити [12].

1.5 Особливості автоматизованих систем управління проектами

Автоматизовані системи управління проектами мають низку особливостей, які відрізняють їх від традиційних методів управління та роблять їх незамінними для сучасних підприємств. Ці особливості охоплюють технічні та функціональні аспекти, які дозволяють підвищити ефективність роботи команд, оптимізувати управлінські процеси та забезпечити успішне досягнення цілей проектів. Розглянемо основні з них.

Однією з ключових особливостей автоматизованих систем управління проектами є централізоване зберігання всієї інформації, пов’язаної з проектом. Це означає, що всі дані — від планів проекту до звітів про прогрес — зберігаються в одному місці і доступні всім учасникам проекту. Це не тільки зменшує ймовірність втрати інформації, але й спрощує доступ до неї, забезпечуючи кращу організацію роботи та координацію дій команди.

Автоматизація дозволяє звільнити час для виконання більш стратегічних завдань, таких як аналіз ризиків або прийняття важливих рішень щодо розподілу ресурсів. Крім того, автоматизовані системи забезпечують точність виконання завдань і зменшують ризики, пов’язані з людським фактором, що є особливо важливим у великих проектах із складною структурою.

Ще однією важливою особливістю автоматизованих систем є можливість організації ефективної командної роботи. Система дозволяє учасникам проекту взаємодіяти в реальному часі, обмінюватися повідомленнями, коментувати завдання та спільно працювати над документами. Відкритий доступ до інформації забезпечує прозорість процесів та допомагає кожному члену команди бути в курсі всіх важливих змін і оновлень.

Системи управління проектами підтримують функції для спільної роботи, такі як чати, форуми, дошки обговорень та можливість прикріплення файлів, що робить комунікацію більш ефективною і знижує потребу в тривалих зустрічах або листуванні по електронній пошті. Для віддалених або розподілених команд такі можливості є незамінними, оскільки вони дозволяють синхронізувати роботу, навіть коли учасники працюють у різних часових поясах.

Візуалізація є ще однією ключовою особливістю автоматизованих систем управління проектами. Використовуючи такі інструменти, як діаграми Ганта, канбан-дошки або інші засоби візуалізації, керівники проектів та учасники команди можуть отримати чітке уявлення про прогрес проекту, залежності між завданнями та потенційні проблеми, що можуть виникнути на різних етапах роботи.

Діаграми Ганта є особливо корисними для планування та моніторингу складних проектів з великою кількістю взаємопов'язаних завдань, оскільки вони дозволяють візуально бачити, які завдання є критичними для завершення проекту вчасно. Канбан-дошки надають можливість швидко переглядати поточний статус завдань, що корисно для команд, які працюють за методологіями Agile або Scrum [13].

Автоматизовані системи управління проектами часто підтримують інтеграцію з іншими корпоративними програмами та сервісами, що дозволяє об'єднати всі бізнес-процеси підприємства в єдине інформаційне середовище. Це можуть бути системи для управління фінансами, людськими ресурсами, CRM-системи або навіть зовнішні інструменти для розробки програмного забезпечення, такі як GitHub або Bitbucket.

Інтеграція дозволяє забезпечити безперебійний обмін даними між різними відділами підприємства, що спрощує координацію роботи та дозволяє уникнути дублювання інформації. Крім того, це дозволяє автоматизувати взаємодію між різними системами, що знижує необхідність ручного введення даних і мінімізує ймовірність помилок.

Отже, автоматизовані системи управління проектами пропонують широкий спектр можливостей, які дозволяють ефективніше управляти проектами, зменшувати витрати та підвищувати продуктивність команд. Їх функціонал охоплює різні аспекти роботи, від планування і моніторингу до аналізу та звітності, що робить їх універсальними рішеннями для підприємств різних галузей.

1.6 Об'єкт автоматизації

Об'єктом автоматизації в даній магістерській роботі є підприємство, яке займається розробкою програмного забезпечення. Це підприємство має розподілену команду, що складається з розробників, тестувальників, менеджерів проектів та інших спеціалістів, які працюють над кількома проектами одночасно. Процеси управління проектами, розподілом завдань та контролем за їх виконанням на підприємстві виконуються вручну або за допомогою окремих інструментів, що не забезпечують централізованого підходу до управління та автоматизації рутинних операцій.

Основною проблемою, з якою стикається підприємство, є складність координації роботи між різними командами і департаментами, що знижує ефективність виконання завдань. Процеси планування та контролю проектів виконуються неавтоматизовано, що ускладнює відстеження виконання завдань, своєчасне коригування планів і взаємодію між членами команди.

Автоматизована система управління проектами, яка включає канбан-дошку та інтеграцію з месенджером Telegram, покликана вирішити ці проблеми. Підприємству потрібна система, що забезпечить:

- Централізоване управління всіма проектами, що виконуються в компанії.
- Автоматизований розподіл завдань між учасниками проекту.
- Візуалізацію прогресу виконання завдань через канбан-дошку.
- Оперативну комунікацію між членами команди через Telegram-бота.
- Можливість отримати звіти про виконання в режимі реального часу.

Наприклад, підприємство може бути середньою ІТ-компанією, яка займається розробкою мобільних додатків і веб-сервісів. Компанія виконує кілька проектів паралельно, що ускладнює контроль над розподілом ресурсів і виконанням завдань у рамках кожного проекту. Впровадження автоматизованої системи дозволить не лише оптимізувати процеси управління, а й підвищити продуктивність роботи всієї компанії завдяки прозорій візуалізації задач, чіткому контролю над термінами і можливістю швидкої комунікації між членами команди.

Висновки до розділу

Автоматизовані системи управління проектами (АСУП) стають невід'ємною частиною сучасних підприємств, забезпечуючи їх конкурентоспроможність у швидкозмінному бізнес-середовищі. Ці системи дозволяють оптимізувати роботу команди, автоматизувати рутинні процеси, підвищити ефективність планування та контроль за виконанням проектів.

Аналіз предметної області підтвердив, що впровадження автоматизованих систем на конкретному підприємстві має значні переваги: від централізованого зберігання даних до покращеної взаємодії між членами команди, що особливо актуально у сучасних умовах віддаленої роботи. Це дозволяє зменшити ризики, пов'язані з людським фактором, і підвищити прозорість процесів.

У таблиці 1.2 наочно представлено порівняння традиційних методів управління проектами з автоматизованими системами. Ключові аспекти, як-от централізоване зберігання даних, автоматизація звітності, інтеграція з іншими системами та підтримка різних методологій (наприклад, Scrum, Kanban), підкреслюють ефективність АСУП.

Особливо важливо відзначити роль автоматизованих систем у безпеці даних: централізоване зберігання інформації, резервне копіювання та шифрування гарантують збереження конфіденційної інформації підприємства. Це створює додаткову цінність у порівнянні з традиційними підходами, які є менш захищеними та менш гнучкими.

Також вагомою перевагою є можливість інтеграції АСУП із зовнішніми системами, такими як CRM, ERP чи фінансові сервіси. Це дає змогу підприємствам отримувати консолідовані дані для прийняття оперативних управлінських рішень. Крім того, вбудовані засоби візуалізації, наприклад, Kanban-дошки або діаграми Ганта, спрощують сприйняття інформації та планування процесів.

Таким чином, впровадження автоматизованої системи управління проектами не лише спрощує процеси на підприємстві, а й сприяє досягненню стратегічних цілей. Сучасні інструменти автоматизації, які підтримують різні методології роботи, надають підприємствам універсальність і гнучкість, що є важливим фактором для підвищення продуктивності та адаптації до змін.

На основі проведеного аналізу можна зробити висновок, що автоматизовані системи управління проектами є ключовим інструментом для досягнення високої ефективності, забезпечення гнучкості в управлінні та підвищення конкурентоспроможності підприємства. Вони дозволяють не тільки зменшити витрати часу і ресурсів, але й створюють умови для покращення якості виконання проектів завдяки автоматизації процесів та інтеграції з іншими бізнес-інструментами.

Таблиця 1.2 Переваги автоматизованих систем управління

Параметр	Традиційні методи управління	Автоматизовані системи управління проектами
Зберігання даних	Розрізнене, локальне, можлива втрата	Централізоване, доступне в реальному часі
Автоматизація процесів	Немає, всі процеси виконуються вручну	Автоматизація планування, контролю та звітності
Командна робота	Складна, потребує постійних зустрічей	Легкість колаборації, можливість віддаленої роботи
Гнучкість у методологіях	Обмежена одним підходом	Підтримка різних методологій (Waterfall, Scrum, Kanban)
Візуалізація процесів	Обмежена, потребує додаткових інструментів	Вбудовані засоби візуалізації (діаграми Ганта, канбан- дошки)
Інтеграція з іншими системами	Обмежена або відсутня	Можлива інтеграція з CRM, ERP, фінансовими системами
Безпека та захист даних	Вразлива до втрат даних	Високий рівень безпеки, шифрування, резервне копіювання

2. ПРОЕКТУВАННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ ДЛЯ ОБРОБКИ, КЕРУВАННЯ ЗАВДАННЯМИ ТА ВИБІР ТЕХНОЛОГІЇ РОЗРОБКИ

2.1 Архітектура системи та її складові

Система управління проектами з інтегрованим ботом Telegram складається з кількох ключових компонентів: клієнтської частини (інтерфейсу користувача), серверної частини (логіка обробки запитів та взаємодії з базою даних) та Telegram-бота, який служить каналом для інтерактивної взаємодії з користувачем. Архітектура системи реалізована за допомогою підходу клієнт-сервер, де клієнтська частина надає можливість взаємодії з користувачем через веб-інтерфейс, а серверна обробляє запити та зберігає інформацію у базі даних [14].

Основні рівні архітектури. Клієнтська частина (Frontend). Клієнтська частина відповідає за інтерфейс взаємодії користувача із системою, який працює у веб-браузері. Вона включає всі елементи графічного інтерфейсу, дозволяє користувачам переглядати інформацію про проекти, створювати нові завдання та взаємодіяти з Telegram-ботом.

Основні завдання клієнтської частини:

- Забезпечення інтуїтивного та зручного користувацького інтерфейсу.
- Обробка дій користувача, таких як створення, редагування або видалення завдань.
- Відправлення запитів на сервер через REST API для отримання або оновлення даних.
- Реалізація динамічного оновлення інтерфейсу без перезавантаження сторінки (з використанням технології AJAX або WebSocket).

Серверна частина (Backend. Серверна частина відповідає за обробку запитів від клієнта та Telegram-бота, управління бізнес-логікою додатку, а також за доступ до бази даних. Сервер отримує запити через REST API, опрацьовує їх, взаємодіє з базою даних та повертає відповідь клієнтській частині або Telegram-боту.

Основні функції серверної частини:

- Реалізація REST API для обміну даними між клієнтською частиною та сервером.
- Управління бізнес-логікою додатку, зокрема обробка створення проектів, завдань, визначення пріоритетів тощо.
- Взаємодія з базою даних для збереження та отримання інформації про користувачів, проекти, завдання та інші дані.
- Забезпечення аутентифікації та авторизації користувачів за допомогою токенів або сесій.
- Інтеграція з Telegram-ботом для обробки повідомлень від користувачів через Telegram API.

База даних є основним компонентом для зберігання даних, таких як інформація про користувачів, проекти, завдання, статуси та інші метадані. Вибір бази даних базується на критеріях продуктивності, масштабованості та підтримки транзакцій.

Основні функції бази даних:

- Збереження інформації про проекти, завдання, користувачів та інші сутності.
- Забезпечення можливості виконання складних запитів для аналізу даних.
- Масштабованість для підтримки збільшення кількості користувачів та обсягу даних.

Telegram-бот забезпечує взаємодію користувачів із системою через месенджер Telegram. Він дозволяє користувачам виконувати основні операції, такі як перегляд поточних завдань, створення нових або оновлення існуючих завдань, без необхідності заходити у веб-інтерфейс.

Основні функції Telegram-бота:

- Обробка команд користувачів через текстові повідомлення або спеціальні кнопки в інтерфейсі Telegram.
- Взаємодія з серверною частиною додатку через API для отримання та оновлення інформації про завдання.

- Надання користувачам інформації про поточний статус проектів і завдань у режимі реального часу.

Взаємодія компонентів:

- Клієнтська частина взаємодіє з серверною через HTTP-запити до REST API. Запити можуть включати отримання списку завдань, зміну статусу завдань, створення нових проектів або завдань.
- Telegram-бот також надсилає запити до API для обробки запитів користувачів через Telegram. Він отримує від серверної частини відповіді та надсилає їх користувачам у вигляді повідомлень або інтерактивних елементів (кнопок).
- Серверна частина взаємодіє з базою даних через ORM (об'єктно-реляційне відображення), що дозволяє зберігати та витягати інформацію без необхідності написання SQL-запитів вручну.

Архітектурні шаблони - система реалізована на основі архітектурного шаблону MVC (Model-View-Controller), який забезпечує розділення логіки програми, даних і їх представлення. У контексті Django:

- Model відповідає за управління даними та взаємодію з базою даних через ORM.
- View обробляє запити користувача і повертає відповідні відповіді (веб-сторінки або API-відповіді).
- Controller (в Django) частково виконує функцію у вигляді "view functions", які обробляють логіку, а також у самому шаблоні управління URL.

Загалом, така архітектура забезпечує модульність, масштабованість та підтримку розвитку системи, дозволяючи легко інтегрувати нові функції та забезпечувати надійність роботи додатку [15].

2.2 Вибір мови програмування та середовища розробки

При розробці веб-додатку для управління проектами та інтеграції з Telegram-ботом, вибір технологій є ключовим етапом, оскільки від цього залежить

стабільність роботи системи, її масштабованість, безпека та зручність у використанні. Враховуючи специфіку задач, було обрано наступні інструменти та підходи для клієнтської та серверної частин.

2.2.1 Вибір мови програмування та середовища розробки

Для реалізації клієнтської частини веб-додатку використовується JavaScript. Це мова програмування, яка є основним інструментом для створення динамічного і інтерактивного контенту на стороні клієнта (у веб-браузері). Основні причини вибору JavaScript для клієнтської частини:

- JavaScript підтримується всіма сучасними браузерами, що гарантує однакову роботу додатку на різних платформах [16].
- Інтерактивність, можливість створення реактивних інтерфейсів з мінімальними затримками завдяки інтеграції з DOM (Document Object Model) браузера.
- Широкий вибір бібліотек та фреймворків, JavaScript має багату екосистему інструментів для фронтенд-розробки, таких як React, Vue.js, Angular, які спрощують розробку складних інтерфейсів.

Для реалізації графічного інтерфейсу у проекті можна використовувати один із сучасних фреймворків, таких як React. React забезпечує ефективне управління станом додатку, модульну структуру та високу продуктивність завдяки віртуальному DOM. Це дозволяє швидко оновлювати інтерфейс у відповідь на дії користувачів без необхідності повного перезавантаження сторінки.

2.2.2 Вибір мови програмування для серверної частини

Для серверної частини обрано мову Python. Python є одним із найпопулярніших мов програмування для веб-розробки через свою простоту,

потужність та багатофункціональність [17]. Вибір на користь Python обумовлений наступними факторами:

- Легкість освоєння та читабельність коду Python має простий і зрозумілий синтаксис, що прискорює процес розробки та полегшує підтримку проекту.
- Широка екосистема бібліотек, безліч бібліотек і фреймворків для роботи з веб-додатками, базами даних, а також для інтеграції з іншими сервісами (наприклад, для взаємодії з API Telegram).
- Стабільність та масштабованість Python добре підходить для створення масштабованих веб-додатків, особливо в поєднанні з фреймворком Django.
- Python також має підтримку асинхронної роботи через бібліотеки, такі як Asyncio, що може бути корисно для масштабування додатка при великій кількості одночасних запитів [18].

2.2.3 Вибір фреймворку

Основним фреймворком для розробки серверної частини обрано Django [19]. Це потужний веб-фреймворк, який спрощує розробку додатків за рахунок наявності готових рішень для часто використовуваних функцій. Переваги Django:

- Вбудовані компоненти Django надають автоматизовані рішення для управління базами даних (ORM), аутентифікації користувачів, обробки форм, і контролю доступу.
- Швидкий старт завдяки численним готовим компонентам, розробники можуть швидко створювати прототипи додатків і забезпечувати їх роботу.
- Безпека Django має вбудовані механізми для захисту від найбільш поширених веб-загроз, таких як SQL-ін'єкції, міжсайтові підробки запитів (CSRF) та атаки на основі міжсайтового сценаріювання (XSS).

Для реалізації REST API було вибрано Django Rest Framework (DRF). DRF дозволяє легко створювати RESTful API, яке є необхідним для взаємодії клієнтської частини з сервером, а також для інтеграції з Telegram-ботом [20].

Основні переваги DRF:

- Простота у використанні DRF дозволяє швидко створювати API з мінімальними зусиллями завдяки автоматичному створенню ендпоінтів і серіалізаторів для моделі даних.
- Масштабованість дозволяє легко додавати нові функціональні можливості та інтеграції.
- Документація API автоматично генерує документацію для всіх доступних ендпоінтів, що спрощує тестування та підтримку.

2.2.4 Вибір технологій для Telegram-бота

Для розробки Telegram-бота використовується Python у поєднанні з бібліотекою python-telegram-bot, яка надає зручний інтерфейс для інтеграції з Telegram API. Ця бібліотека дозволяє легко обробляти повідомлення користувачів та відповідати на них, створювати діалоги та інтерактивні інтерфейси всередині Telegram [21].

Python-telegram-bot вибрано через такі причини:

- Простота інтеграції бібліотек забезпечує зручні методи для роботи з усіма функціями Telegram API, включаючи обробку повідомлень, кнопок та inline-команд.
- Підтримка асинхронної роботи дозволяє ефективно обробляти великий потік повідомлень від користувачів.

Таким чином, вибір технологій для клієнтської та серверної частин, а також для Telegram-бота був здійснений з урахуванням вимог до продуктивності, безпеки та масштабованості системи.

2.3 Вибір бази даних

Одним із найважливіших етапів розробки веб-додатків є вибір бази даних, яка буде відповідати вимогам проекту щодо зберігання, доступу та управління даними. Для проекту управління проектами з інтеграцією Telegram-бота важливо забезпечити зручність роботи з великим обсягом даних, швидкий доступ та можливість масштабування. Вибір бази даних ґрунтується на різних критеріях, які будуть розглянуті нижче [22].

2.3.1 Критерії вибору бази даних

При виборі бази даних враховувались такі критерії:

- База даних повинна ефективно обробляти велику кількість запитів на читання та запис даних. Для системи управління проектами важливо, щоб зміни у завданнях, статусах або користувацьких даних відображались оперативно [23].
- Проект має потенціал для росту з точки зору кількості користувачів та даних. Важливо, щоб база даних підтримувала горизонтальне або вертикальне масштабування.
- Оскільки багато операцій у системі потребують зміни кількох пов'язаних даних одночасно (наприклад, зміна статусу завдання або оновлення кількох проектів), підтримка транзакцій є важливим аспектом для забезпечення цілісності даних.
- Важливою вимогою є підтримка обраної бази даних фреймворком Django та його ORM (Object-Relational Mapping), щоб забезпечити зручність розробки і управління базою.
- База даних повинна забезпечувати належні механізми безпеки для захисту даних від несанкціонованого доступу. Це включає механізми шифрування даних, контроль доступу і аудит.

2.3.2 Популярні варіанти баз даних

Для сучасних веб-додатків існує декілька популярних варіантів баз даних, кожен з яких має свої переваги та недоліки:

- PostgreSQL — високопродуктивна реляційна база даних з підтримкою складних запитів, індексів та транзакцій. Вона добре підходить для обробки великої кількості одночасних запитів, а також пропонує можливості розширення для роботи з неструктурованими даними, такими як JSON. PostgreSQL інтегрується з Django через вбудовану підтримку ORM.
- MySQL/MariaDB — легкі і швидкі реляційні бази даних, які є популярними у веб-розробці завдяки їхній продуктивності та легкості налаштування. Вони забезпечують хороший рівень продуктивності для додатків середнього масштабу, але їхні можливості щодо обробки складних транзакцій можуть бути менш гнучкими, ніж у PostgreSQL.
- SQLite — легка реляційна база даних, що використовується за замовчуванням у багатьох невеликих проєктах Django. Підходить для тестування та розробки, але через обмежені можливості зберігання даних і масштабованість не підходить для великих проектів.
- MongoDB — нереляційна база даних, яка використовує сховище документів. Вона підходить для додатків, де необхідно працювати з неструктурованими даними, але в проектах, де важлива консистентність і реляційність даних, MongoDB поступається реляційним базам даних [24].
- Oracle Database — це потужне корпоративне рішення з широким набором інструментів для роботи з даними. Вона забезпечує високу продуктивність і безпеку, але є дорогим рішенням, що робить її менш популярною для невеликих і середніх проектів [25].

2.3.3 Обґрунтування вибору бази даних

Для даного проекту було обрано PostgreSQL як основну базу даних з наступних причин:

- PostgreSQL забезпечує ефективну роботу як з малими, так і великими обсягами даних, що дозволяє розширювати систему з часом, не змінюючи основної бази даних [26].
- Потужна підтримка складних SQL-запитів та транзакцій, що є необхідним для забезпечення цілісності даних у системі управління проектами [27].
- PostgreSQL дозволяє зберігати не лише реляційні дані, але й JSON-дані, що може бути корисним для збереження динамічних структур даних, які не завжди підпадають під реляційну схему.
- Django має вбудовану підтримку PostgreSQL через ORM, що спрощує роботу з базою даних на рівні коду, дозволяючи розробникам легко виконувати запити та управляти даними.
- Підтримує різноманітні механізми безпеки, включаючи шифрування даних, автентифікацію за допомогою SSL, а також налаштування прав доступу для різних ролей користувачів.
- Підтримка хмарних платформ, легко інтегрується з провідними хмарними платформами, такими як AWS, Google Cloud та Azure, що робить її ідеальною для хмарних розгортань.

Таким чином, PostgreSQL поєднує в собі всі необхідні характеристики для масштабованої, безпечної та продуктивної бази даних для системи управління проектами з інтеграцією Telegram-бота.

2.4 Середовище розробки PyCharm

Середовище розробки є важливим інструментом для розробників, оскільки саме воно забезпечує комфортну та продуктивну роботу з кодом, управління проектом, тестування та відлагодження. Для створення веб-додатку управління

проектами з інтеграцією Telegram-бота було обрано середовище розробки PyCharm, яке є одним з найпопулярніших IDE (Integrated Development Environment) для Python-розробки. PyCharm забезпечує потужний набір інструментів для ефективної роботи, включаючи підтримку фреймворків, зручний інтерфейс та інтеграцію з системами контролю версій. У цьому розділі буде розглянуто основні можливості PyCharm, які використовувались під час розробки проекту, а також його переваги [28].

- пропонує автодоповнення коду, що дозволяє розробникам швидше писати код, уникаючи синтаксичних помилок. PyCharm аналізує код у реальному часі, надаючи рекомендації щодо використання методів, змінних та інших елементів на основі контексту. Це значно підвищує ефективність роботи, особливо при розробці великих проектів;
- має потужну інтеграцію з фреймворками для розробки веб-додатків, такими як Django, Flask та інші. У випадку даного проекту, Django був основним фреймворком для серверної частини, і PyCharm забезпечив зручний інтерфейс для роботи з його компонентами, включаючи створення моделей, шаблонів, форм та управління базами даних;
- підтримує інтеграцію з популярними системами контролю версій, такими як Git, що дозволяє зручно управляти версіями проекту, відстежувати зміни, створювати гілки та об'єднувати їх. У проекті використовувалась система Git для управління версіями, що допомагало організовувати спільну роботу над кодом та зберігати історію змін;
- має вбудовані інструменти для запуску та відлагодження тестів. Враховуючи, що тестування є важливою частиною будь-якого проекту, PyCharm надає можливість швидко запускати тести для перевірки окремих компонентів та виявлення помилок у коді. Це дозволило підтримувати високу якість коду під час розробки системи управління проектами;
- PyCharm забезпечує зручний відлагоджувач (debugger), який дозволяє зупиняти виконання коду на певних точках (breakpoints), переглядати значення змінних, стежити за виконанням циклів та функцій у реальному часі. Це стало

особливо корисним під час відлагодження інтеграції Telegram-бота з серверною частиною системи [29];

- PyCharm забезпечує просте створення та управління віртуальними середовищами. Це дозволяє ізолювати залежності проекту, уникаючи конфліктів між різними бібліотеками та пакетами. У даному проекті використовувалось віртуальне середовище для ізоляції бібліотек, таких як Django, Django Rest Framework, а також бібліотек для взаємодії з API Telegram-бота;
- PyCharm дозволяє підключати бази даних прямо в IDE та виконувати SQL-запити без необхідності використовувати окремі клієнти для баз даних. Це значно спрощує управління базою даних під час розробки. В даному проекті використовувалась база даних PostgreSQL, і PyCharm надав можливість підключатися до бази, переглядати структуру таблиць, редагувати дані та виконувати запити;
- Хоча PyCharm орієнтований на Python-розробку, він також підтримує роботу з HTML, CSS, JavaScript та іншими технологіями фронтенду. Це було корисно під час розробки клієнтської частини системи управління проектами, де використовувались HTML, CSS та JavaScript для інтерфейсу користувача;
- У випадках, коли потрібно розгорнути додаток у контейнерах, PyCharm підтримує інтеграцію з Docker, що дозволяє легко створювати та керувати контейнерами для додатка. Docker може бути корисним для хмарного розгортання проекту або тестування у різних середовищах.

PyCharm дозволяє працювати з кодом, тестувати, відлагоджувати, працювати з базами даних та управляти версіями проекту без необхідності використовувати окремі інструменти. Це значно спрощує робочий процес і дозволяє зосередитися на розробці функціоналу, не витрачаючи час на налаштування зовнішніх інструментів.

PyCharm забезпечує гнучкість у виборі фреймворків та бібліотек, що дозволяє використовувати його для різних типів проектів. Для даного проекту важливою була підтримка Django та бібліотек для інтеграції з Telegram API.

Інтелектуальне автодоповнення, рефакторинг коду та навігація по проекту значно спрощують роботу з великими кодовими базами. PyCharm дозволяє легко знаходити потрібні файли, методи та змінні, а також швидко вносити зміни до коду. Вбудовані інструменти для тестування та відлагодження допомагають розробникам швидко знаходити помилки та виправляти їх, що покращує якість коду та скорочує час на виявлення проблем. PyCharm дозволяє легко інтегрувати хмарні сервіси та використовувати Docker для тестування та розгортання проектів у хмарних середовищах, що робить його ідеальним для сучасних веб-додатків.

Для розробки системи управління проектами з інтеграцією Telegram-бота було обрано середовище PyCharm з наступних причин:

- Забезпечує глибоку інтеграцію з фреймворком Django, що полегшує роботу з моделями, представленнями, шаблонами та базами даних. Це стало основною причиною вибору PyCharm для розробки серверної частини проекту.
- Дозволив ефективно відлагоджувати проект, зупиняючи виконання на певних точках, переглядаючи значення змінних та слідкуючи за виконанням коду. Це було особливо корисно під час інтеграції Telegram-бота з сервером через API.
- Оскільки проект мав багато залежностей, PyCharm спростив роботу з ними, дозволяючи легко створювати та керувати віртуальними середовищами.
- Використання PyCharm спростило роботу з системою контролю версій Git, що забезпечило ефективне управління кодом проекту та співпрацю з іншими учасниками.

Завдяки таким можливостям, PyCharm став основним інструментом для розробки даного проекту, забезпечуючи високу ефективність роботи та комфортний процес розробки.

Висновки до розділу

У даному розділі ми розглянули проектування та реалізацію системи управління проектами з інтеграцією Telegram-бота, а також обґрунтували вибір

технологічного стеку. Для розробки серверної частини було обрано Django як основний фреймворк. Django забезпечує швидку розробку, високу масштабованість та безпеку, що є важливими аспектами для сучасних веб-додатків, орієнтованих на управління проектами. Використання Django Rest Framework спрощує створення API для взаємодії з клієнтськими додатками та Telegram-ботом.

На клієнтській частині проекту використовуються HTML, CSS та JavaScript. Ці технології забезпечують гнучкість і зручність у розробці інтерфейсів, дозволяючи створювати інтерактивні та швидкі користувацькі інтерфейси, що є важливими для зручної роботи з системою управління проектами.

Для бази даних було обрано PostgreSQL, що забезпечує надійність, підтримку складних транзакцій і хорошу масштабованість. Це дозволяє ефективно зберігати та обробляти великий обсяг даних у межах системи управління проектами, забезпечуючи цілісність та безпеку даних.

Особливу увагу було приділено вибору системи безпеки. Для аутентифікації користувачів обрано Django Allauth та JWT, що забезпечує безпечну ідентифікацію користувачів та захищає дані за допомогою шифрування і токенів. Використання HTTPS гарантує захищену передачу даних між сервером і клієнтами.

Середовище розробки PyCharm було обрано завдяки його інтеграції з Django, підтримці роботи з базами даних, системами контролю версій та віртуальними середовищами. Це дозволило організувати зручний процес розробки і тестування системи [30].

3. РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Структура проекту та основні налаштування

Для розробки автоматизованої системи управління проектами на Django було створено основний проект під назвою kanban, у межах якого реалізовано додаток board для керування основними функціями Kanban-системи. Також використовується додаток для Telegram-бота, який виконує інтерактивну взаємодію з користувачами.

Процес створення проекту розпочинається зі створення віртуального середовища для ізоляції залежностей:

```
python -m venv env
.\env\Scripts\activate
pip install django djangorestframework
```

Після активації середовища створюється проект:

```
django-admin startproject kanban
cd kanban
python manage.py startapp board
```

Структура проекту kanban містить такі основні файли та директорії (рисунок 3.1):

- kanban/settings.py – файл налаштувань, де задані параметри бази даних, сервіси аутентифікації та інші параметри проекту.
- kanban/urls.py – головний файл маршрутів, який налаштовує доступ до API і маршрутів.
- board/models.py – файл з визначенням моделей даних, включаючи Project, Task, User.
- board/serializers.py – модуль для налаштування серіалізаторів, які перетворюють моделі у формати JSON для API.
- board/views.py – представлення (views), де обробляються запити до API, наприклад, для створення або видалення завдань.

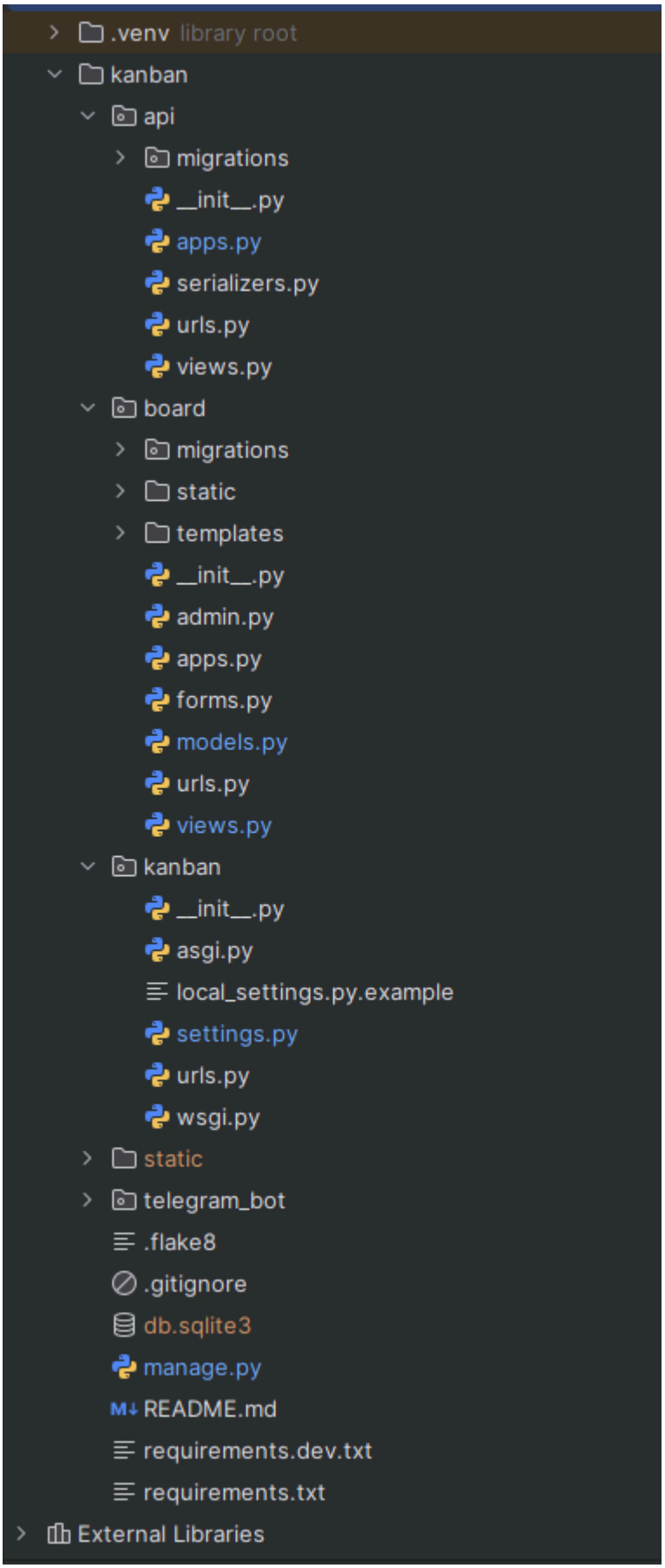

Рисунок 3.1 - Загальна структура проєкту

Додатки Django у секції INSTALLED_APPS додаємо rest_framework для побудови REST API, board для основних моделей та представлень, і rest_framework.authtoken для реалізації токенної аутентифікації.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'rest_framework.authtoken',
    'board',
]
```

Налаштування REST Framework, щоб забезпечити автентифікацію та обмежити доступ до API, вказуємо, що за замовчуванням API повинен використовувати токену аутентифікацію.

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ],}
```

Маршрутизація дозволяє налаштувати доступ до API та адміністративної панелі:

- kanban/urls.py: Цей файл керує маршрутами проекту і вказує на включення API маршрутів додатку board:
- board/urls.py: У цьому файлі визначено маршрути для додатку board, які дозволяють працювати з проектами, завданнями та іншими сутностями.
- Файл serializers.py містить серіалізатори, які перетворюють моделі в JSON-формат та навпаки, дозволяючи обмін даними між сервером і клієнтською частиною через API.

Файл models.py визначає основні моделі, що відповідають за структуру даних у базі даних. У нашому випадку це моделі Проекту (Project) та Завдання (Task). Опис моделей:

- Project – модель для збереження інформації про проекти.

  name – назва проекту.

  description – опис проекту.

  created_at – дата створення проекту.

  owner – користувач, який створив проект.

- Task – модель для завдань, пов’язаних з проектами.

  title – назва завдання.

  description – опис завдання.

  status – статус завдання з вибором між трьома варіантами: "Зробити", "В процесі", "На перевірці", "Зроблено".

  project – проект, до якого відноситься завдання.

  assigned_to – користувач, якому призначено завдання.

  created_at та updated_at – дати створення та останнього оновлення завдання.

3.2 Реалізація клієнтської частини (Frontend)

Даний розділ описує реалізацію клієнтської частини Kanban-системи, що забезпечує користувачам інтуїтивний інтерфейс для взаємодії з проєктами та завданнями. Клієнтська частина розроблена як веб-застосунок з акцентом на зручність та швидкість роботи, що забезпечується завдяки використанню асинхронної взаємодії з сервером. Це дозволяє зменшити навантаження на сервер та підвищити зручність роботи з інтерфейсом.

Для створення клієнтської частини використані наступні технології:

- HTML та CSS для структури та оформлення сторінок.
- JavaScript для динамічної взаємодії з елементами інтерфейсу та API.

- AJAX (асинхронний JavaScript та XML) для взаємодії з сервером без перезавантаження сторінки, що покращує роботу інтерфейсу.
- Bootstrap CSS-фреймворк, який надає готові компоненти та забезпечує адаптивність інтерфейсу.

Цей стек технологій вибраний з метою прискорити процес розробки, забезпечити високу продуктивність та створити сучасний та зручний інтерфейс для користувачів.

Інтерфейс системи організований у вигляді дошки Kanban, де завдання розподілені за статусами, що відображають їхній стан у процесі виконання. Кожна дошка включає колонки:

- Заплановано (To Do): завдання, які потребують виконання.
- В роботі (In Progress): завдання, що виконуються.
- На перевірці(Review): завдання, що тестуються
- Виконано (Done): завершені завдання.

Користувач може переміщати завдання між колонками за допомогою drag-and-drop або через інтерактивні кнопки управління, доступні для кожного завдання.

Основні компоненти клієнтської частини включають головна сторінку, що відображає список проектів користувача з можливістю переходу на сторінку кожного проекту. Дошка завдань - головний інтерфейс Kanban-дошки, де користувач може бачити усі завдання, а також їхній статус. Реалізована підтримка drag-and-drop для зміни статусу завдань. Форма додавання завдань - інтерфейс для створення нових завдань із зазначенням проєкту, назви та опису. Інтерактивні кнопки управління, кожне завдання має кнопки для редагування, видалення або зміни статусу, що спрощує управління процесами.

Структура HTML шаблонів:

- base.html: основний шаблон, який містить спільні елементи для всіх сторінок, такі як заголовок, шапка (header), футер (footer), підключення CSS/JS-файлів. Він використовується як основа для інших сторінок, що дозволяє уникнути дублювання коду.

- index.html: головна сторінка застосунку, яка зазвичай відображає основну інформацію або огляд проєктів і завдань. Цей шаблон може показувати Kanban-дошку з різними завданнями, зокрема ті, які потребують дій користувача (рисунок 3.2).

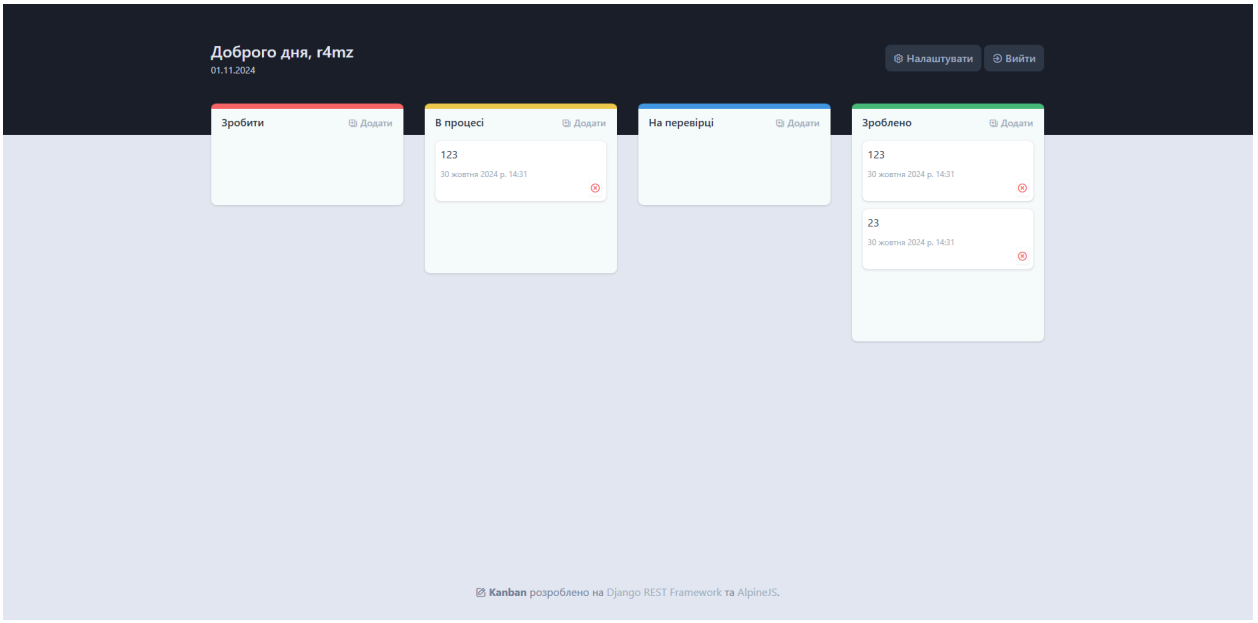

Рисунок 3.2 - Головна сторінка

- login.html: сторінка для входу користувача в систему. Вона зазвичай містить форму для введення імені користувача (або електронної пошти) та пароля (рисунок 3.3).

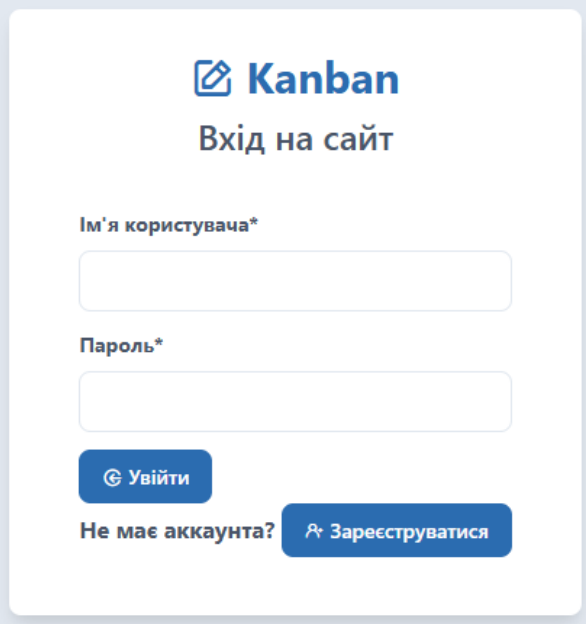

Рисунок 3.3 - Сторінка для входу

- messages.html: шаблон для відображення повідомлень користувачам, наприклад, про успішний вхід або помилки під час взаємодії із системою. Може бути включений в інші сторінки для відображення сповіщень.
- register.html: сторінка реєстрації нового користувача, де користувач може створити новий обліковий запис. Зазвичай містить форму для введення основних даних, таких як ім'я користувача, пароль і, можливо, електронну пошту (рисунок 3.4).

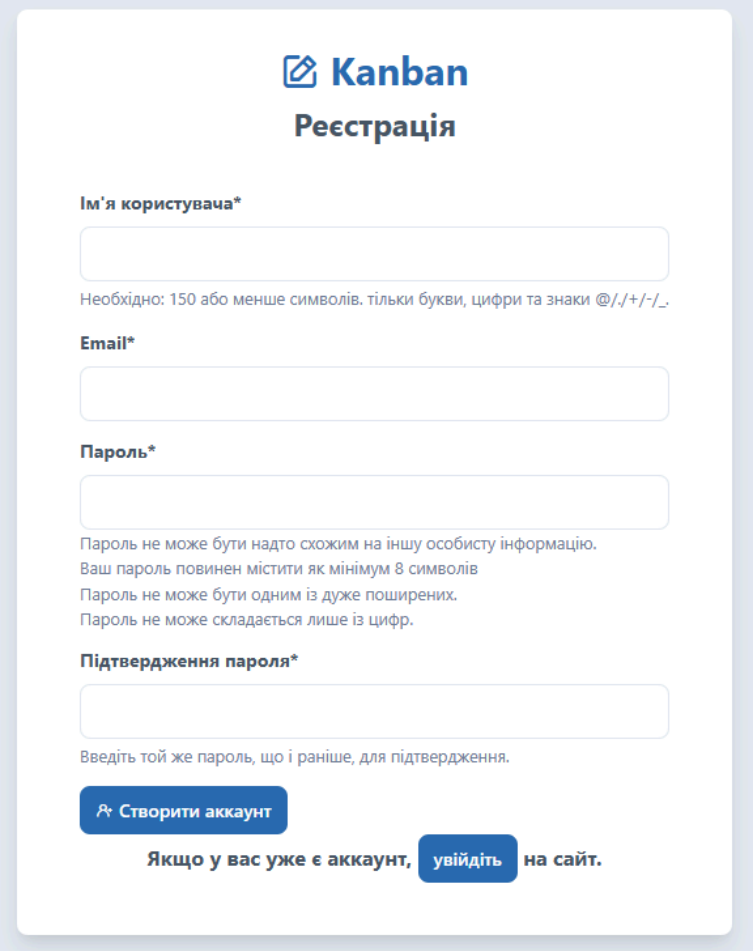

Рис 3.4 - Сторінка реєстрації

- settings.html: сторінка налаштувань користувача, яка дозволяє змінювати дизайн сторінки (рисунок 3.5).

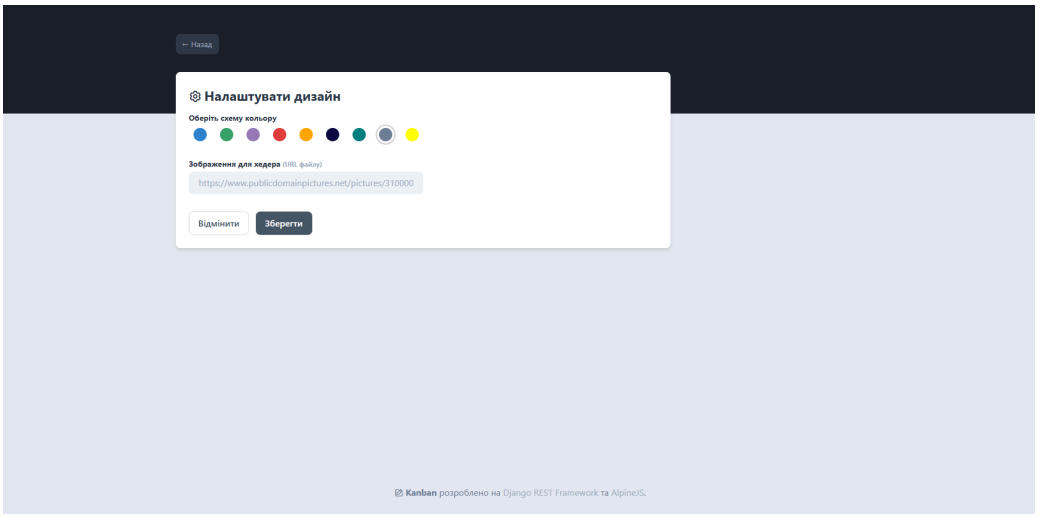

Рисунок 3.5 - Сторінка налаштувань

Взаємодія з API у нашому застосунку реалізована за допомогою Django REST framework, що забезпечує доступ до даних у форматі JSON та дозволяє легко здійснювати запити до серверної частини. API підтримує операції CRUD (створення, читання, оновлення та видалення) для управління завданнями (tasks) користувачів.

API складається з таких основних компонентів:

- Серіалізатори (serializers.py) — для перетворення даних моделей у формат JSON.
- Представлення (views.py) — для визначення логіки обробки HTTP-запитів.
- Маршрути (urls.py) — для визначення URL-адрес, які обробляє API.

У файлі serializers.py описані серіалізатори для моделей Task та User.

- TaskSerializer. Цей серіалізатор відповідає за перетворення об'єктів моделі Task у JSON-формат. Поле owner (власник) є доступним лише для читання, щоб уникнути його редагування користувачем напряму.

```
from board.models import Task
from django.contrib.auth.models import User
from rest_framework import serializers
class TaskSerializer(serializers.ModelSerializer):
    owner = serializers.ReadOnlyField(source='owner.username')
    class Meta:
        model = Task
```

```
        fields = ('uuid', 'name', 'boardName', 'date', 'owner')
```

- UserSerializer. Серіалізатор для моделі User, що включає ідентифікатор користувача, ім'я користувача та список завдань (tasks), які належать цьому користувачу.

```
class UserSerializer(serializers.ModelSerializer):
    tasks = serializers.PrimaryKeyRelatedField(many=True, read_only=True)
    class Meta:
        model = User
        fields = ['id', 'username', 'tasks']
```

У файлі views.py реалізовані два класи представлень для обробки запитів до завдань. Клас ListTask (успадковує ListCreateAPIView) обробляє запити на отримання списку завдань та створення нового завдання.

- get_queryset повертає тільки ті завдання, які належать автентифікованому користувачу.
- perform_create зберігає нове завдання з власником, який відповідає поточному користувачу.

```
from api.serializers import TaskSerializer
from board.models import Task
from rest_framework.authentication import SessionAuthentication
from rest_framework.permissions import IsAuthenticated
class ListTask(generics.ListCreateAPIView):
    queryset = Task.objects.all()
    serializer_class = TaskSerializer
    authentication_classes = [SessionAuthentication]
    permission_classes = [IsAuthenticated]
    def get_queryset(self):
        user = self.request.user
        return Task.objects.filter(owner=user)
    def perform_create(self, serializer):
        serializer.save(owner=self.request.user)
```

- клас DetailTask (успадковує RetrieveUpdateDestroyAPIView) обробляє запити для перегляду, оновлення та видалення конкретного завдання. Подібно до

ListTask, він фільтрує завдання, щоб надати доступ лише до завдань, які належать поточному користувачу.

```
class DetailTask(generics.RetrieveUpdateDestroyAPIView):
    queryset = Task.objects.all()
    serializer_class = TaskSerializer
    authentication_classes = [SessionAuthentication]
    permission_classes = [IsAuthenticated]
    def get_queryset(self):
        user = self.request.user
        return Task.objects.filter(owner=user)
```

Файл urls.py в папці api визначає маршрути, що дозволяють доступ до представлень ListTask та DetailTask. Наприклад, для отримання списку всіх завдань або доступу до певного завдання, використовуючи ідентифікатор.

```
from django.urls import path
from . import views
urlpatterns = [
    path('tasks/', views.ListTask.as_view(), name='list_tasks'),
    path('tasks/<int:pk>/', views.DetailTask.as_view(), name='detail_task'),
]
```

Обидва класи представлень використовують SessionAuthentication для автентифікації та IsAuthenticated для перевірки прав доступу. Це означає, що лише автентифіковані користувачі можуть отримати доступ до API. Користувачі можуть створювати, переглядати, оновлювати та видаляти тільки ті завдання, власниками яких вони є.

3.3 Розробка API для клієнтської частини

Метою створення API є забезпечення взаємодії між клієнтською частиною Kanban-дошки та сервером, що обробляє дані та зберігає інформацію про проекти, завдання, а також користувачів. API дозволяє клієнтській частині виконувати CRUD-операції, що включають створення, читання, оновлення та видалення

завдань, а також реалізує аутентифікацію та управління доступом до даних. API забезпечує такі основні функції:

- Доступ користувачів до інформації про проекти та завдання.
- Безпечне зберігання та оновлення даних.
- Захист інформації через аутентифікацію та авторизацію.

Для створення API використовується Django Rest Framework (DRF), що дозволяє швидко та зручно розробляти RESTful API на базі Django. Django Rest Framework підтримує функції, необхідні для налаштування маршрутизації, аутентифікації, серіалізації та обробки запитів, що полегшує створення масштабованого API.

Процес роботи API:

- Формування запиту. Клієнт формує HTTP-запит із параметрами, що відповідають запитуваній операції.
- Обробка на сервері.
- Сервер перевіряє автентичність користувача через токен.
- Проводить валідацію отриманих даних.
- Взаємодіє з ORM для доступу до бази даних.
- Відповідь сервера.
- Сервер повертає JSON-об’єкт із результатами операції (успіх або помилка).

Переваги API:

- Клієнтська частина не залежить від реалізації серверної логіки.
- Авторизація через токени забезпечує захист даних.
- Нові функції можуть бути легко додані через створення нових ендпоінтів.
- Взаємодія бота з базою даних

Проте є і недоліки. API працює за принципом "запит-відповідь", тому клієнтська частина не отримує оновлення автоматично, без ручного запиту. Це може призводити до затримок у відображенні змін, якщо кілька користувачів одночасно працюють із системою. При значному збільшенні кількості завдань або проектів час обробки запитів може зрости, що вплине на продуктивність. Немає оптимізованих механізмів для посторінкового завантаження (pagination) у стандартній реалізації.

API не дозволяє складних фільтрацій чи сортування завдань за специфічними параметрами без додаткової розробки. Це ускладнює роботу з великими наборами даних. API не включає автоматично згенерованої документації (наприклад, через Swagger), що ускладнює інтеграцію нових клієнтських додатків або розробку нових функцій. API не має вбудованих інструментів для відстеження запитів, часу їх виконання або частоти використання ендпоінтів, що ускладнює пошук та усунення проблем.

Для усунення більшості недоліків можна використати наступні впровадження:

- WebSocket або SignalR для оновлення даних у реальному часі.
- Додавання пагінації та розширеної фільтрації.
- Інтеграція Swagger або іншого генератора документації.
- Реалізація двофакторної аутентифікації для підвищення безпеки.
- Розширення підтримки сторонніх інтеграцій через стандартні протоколи (наприклад, OAuth2).

Для відображення структури програмного забезпечення на рівні фізичних модулів, необхідно створити діаграму компонентів. Вона відображає архітектуру системи, демонструючи взаємодію між її ключовими складовими, такими як клієнтська частина (Kanban-дошка), серверна частина, Telegram бот, база даних і API. Вона допомагає зрозуміти, як різні компоненти системи обмінюються даними та виконують свої функції (рисунок 3.6).

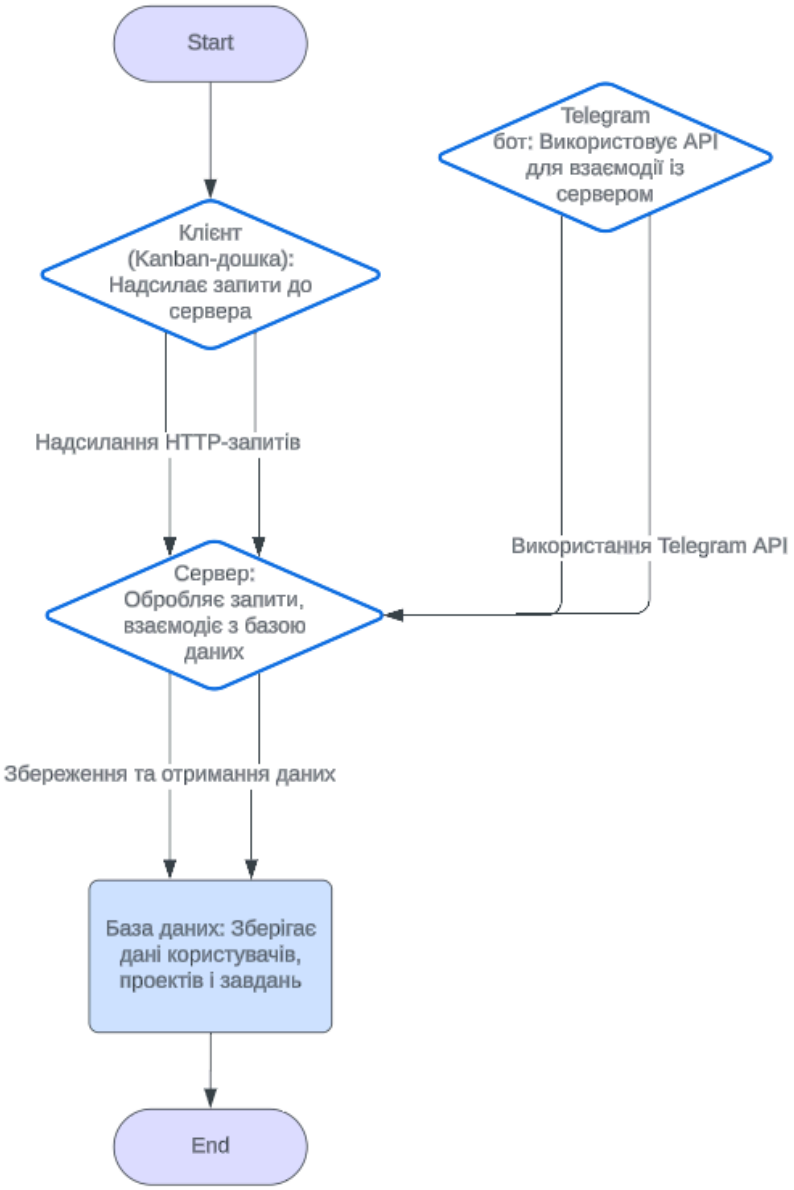

Рисунок 3.6 - Діаграма компонентів

Архітектура API та основні компоненти. Забезпечує реєстрацію, авторизацію та отримання інформації про користувачів. Використання токенів для аутентифікації дозволяє захистити дані та контролювати доступ. Кожен проект може містити декілька завдань і має певні атрибути, зокрема назву, опис та дату створення. За допомогою API користувач може створювати, переглядати, редагувати та видаляти проекти. Дозволяє додавати нові завдання до проектів, переглядати існуючі завдання, змінювати їхні атрибути (наприклад, назву, опис, статус) та видаляти завдання.

API містить набір ендпоінтів, які забезпечують доступ до даних проекту (рисунок 3.7).

Ось приклади ендпоінтів для основних CRUD-операцій:

- Управління користувачами
    - POST /api/register/: Реєстрація нового користувача.
    - POST /api/login/: Авторизація користувача та отримання токена доступу.
    - GET /api/users/<user_id>/: Отримання детальної інформації про користувача.
- Управління проектами
    - GET /api/projects/: Отримання списку проектів, доступних користувачу.
    - POST /api/projects/: Створення нового проекту.
    - GET /api/projects/<project_id>/: Отримання інформації про конкретний - проект.
    - PUT /api/projects/<project_id>/: Оновлення даних проекту.
    - DELETE /api/projects/<project_id>/: Видалення проекту.
- Управління завданнями
    - GET /api/tasks/: Отримання списку всіх завдань.
    - POST /api/tasks/: Створення нового завдання.
    - GET /api/tasks/<task_id>/: Отримання детальної інформації про конкретне завдання.
    - PUT /api/tasks/<task_id>/: Оновлення інформації про завдання.
    - DELETE /api/tasks/<task_id>/: Видалення завдання.

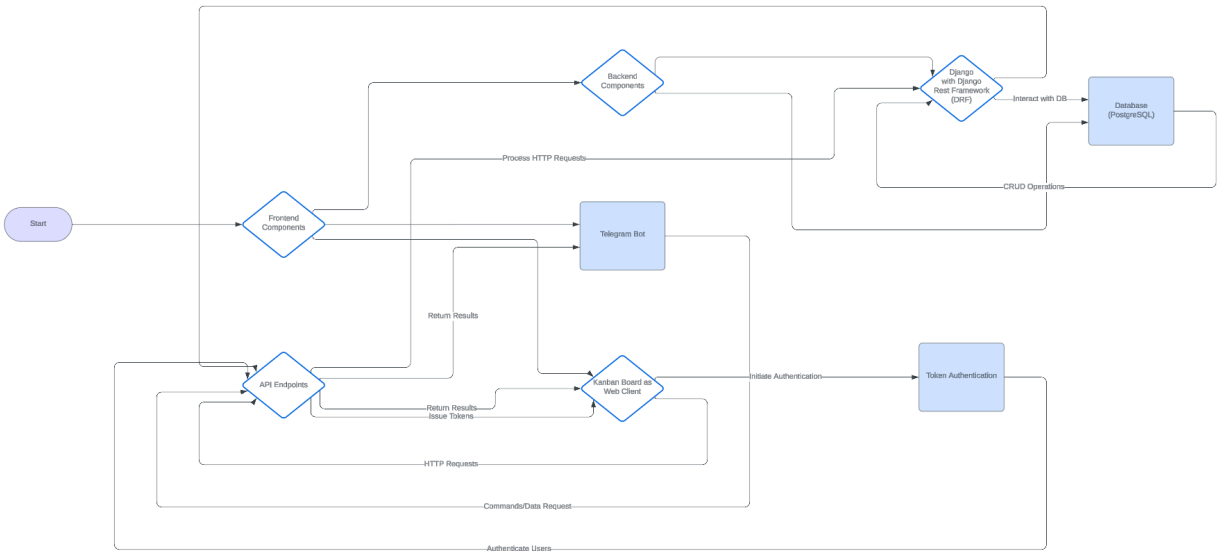

Рисунок 3.7 - Архітектурна діаграма API

Основні ендпоінти API:

- Ендпоінти для аутентифікації та управління користувачами.
    - Реєстрація користувачів - дозволяє новим користувачам зареєструватися в системі.
    - Авторизація користувачів - забезпечує видачу токенів для аутентифікації, що дає користувачам можливість виконувати дії у системі.
    - Перегляд профілю - користувачі можуть отримати доступ до інформації про свій профіль.
- Ендпоінти для управління проектами.
    - Отримання списку проектів, користувач може переглянути всі доступні йому проекти.
    - Створення нового проекту, надає можливість додати новий проект із вказаними параметрами.
    - Оновлення та видалення проектів, дозволяє вносити зміни в існуючі проекти або видаляти їх, якщо вони більше не потрібні.
- Ендпоінти для управління завданнями.

- Отримання списку завдань - дозволяє переглянути всі завдання, доступні користувачу.
- Додавання нового завдання - дає можливість створити нове завдання, визначивши його назву, опис, статус і зв'язок з проектом.
- Оновлення та видалення завдань - підтримує редагування існуючих завдань і видалення непотрібних.

Серіалізатори виступають важливою частиною API, оскільки вони відповідають за форматування даних перед передачею клієнту. Вони перетворюють об'єкти з бази даних у зрозумілий формат (наприклад, JSON), який потім використовується клієнтською частиною.

Представлення обробляють логіку запитів API. Вони приймають запити від клієнта (наприклад, отримати список завдань), взаємодіють із базою даних та повертають відповідь клієнту у потрібному форматі. Представлення також можуть обробляти бізнес-логіку, наприклад, перевірку прав користувачів для певних операцій.

API використовує систему маршрутів для визначення доступу до кожного ендпоінту. Кожен ендпоінт відповідає за конкретну функцію (наприклад, створення проекту або редагування завдання) і має свій унікальний шлях.

Для захисту даних і запобігання несанкціонованого доступу використовується токен-аутентифікація. Після успішного входу користувач отримує токен, який використовує для доступу до захищених ендпоінтів. Токени можуть оновлюватись автоматично, що забезпечує безперервність сесії користувача без необхідності повторної аутентифікації.

Для забезпечення зручного зберігання та управління даними у розробленій системі використовується реляційна база даних. Основні дані, такі як інформація про користувачів, проекти та завдання, організовані в таблиці, що мають чітко визначені зв’язки між собою.

На рисунку 3.8 представлено ER-діаграму бази даних, яка відображає структуру даних і взаємозв’язки між основними сутностями системи:

- User (Користувач). Таблиця містить інформацію про зареєстрованих користувачів, включаючи їх логін, email та пароль.
- Project (Проект). Таблиця зберігає інформацію про проекти, такі як назва, опис і власник (зв’язок із користувачем).
- Task (Завдання). Таблиця містить дані про завдання, включаючи назву, опис, статус та проект, до якого воно належить.

ER-діаграма є основою для моделювання бази даних і забезпечує чітке розуміння її структури. Вона полегшує процес розробки та інтеграції нових функцій, дозволяючи уникнути конфліктів у даних.

Рисунок 3.8 демонструє ці зв’язки у вигляді схеми, де кожна сутність представлена прямокутником із атрибутами, а зв’язки між ними позначені лініями.

Клієнт (Kanban-дошка):

- Відповідає за інтерфейс користувача.
- Надсилає HTTP-запити до серверу для отримання чи збереження даних про завдання, колонки та інші елементи.

Telegram-бот:

- Використовує Telegram API для обміну інформацією із сервером.
- Дозволяє користувачам взаємодіяти із системою через команди в Telegram (створення, редагування, видалення завдань).

Сервер:

- Приймає HTTP-запити від клієнта та Telegram-бота.
- Виконує обробку запитів та взаємодіє з базою даних.
- Забезпечує обробку бізнес-логіки: оновлення даних завдань, проектів, користувачів тощо.

База даних:

- Зберігає дані користувачів, проекти та завдання.
- Підтримує збереження, отримання та оновлення інформації.

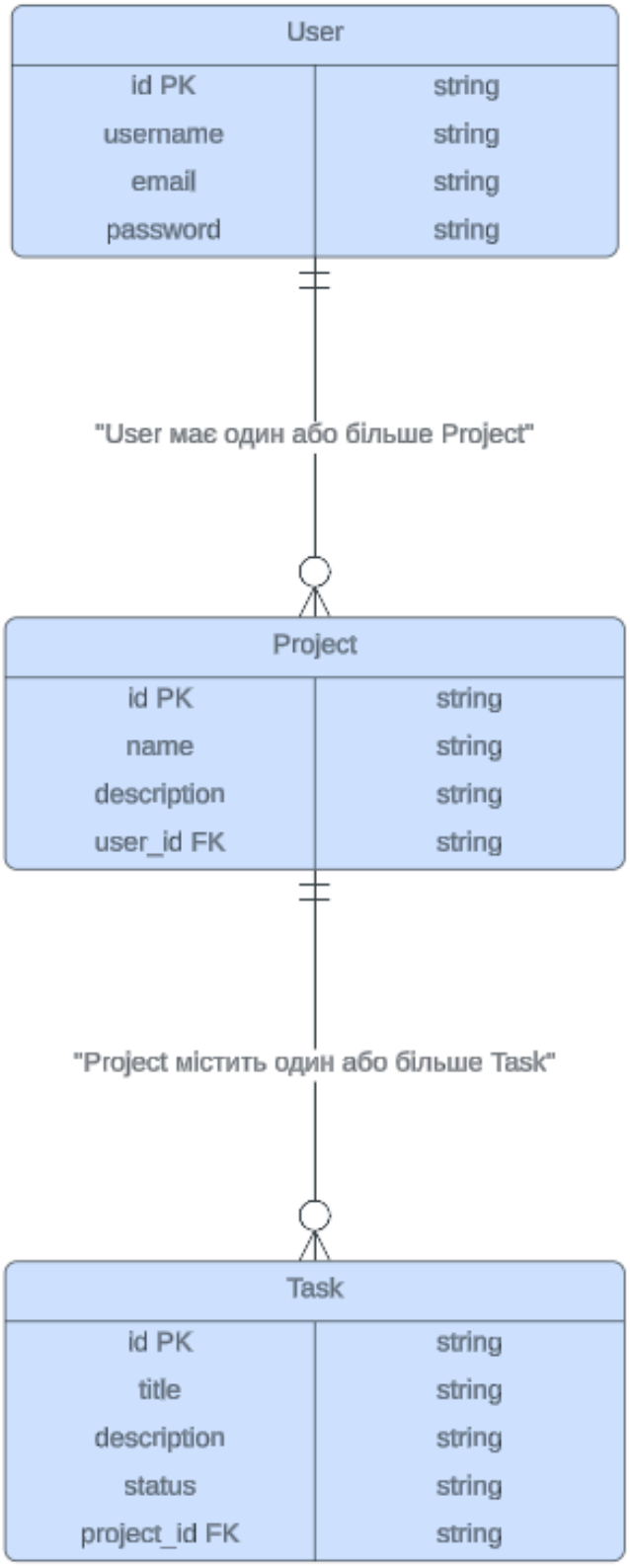

Рисунок 3.8 - ER-діаграма для бази даних

API підтримує кілька типів запитів для основних CRUD-операцій (рисунок 3.9). Нижче описані приклади запитів та відповідей для деяких операцій.

- Клієнт надсилає запит для отримання всіх проектів, доступних йому. У відповідь сервер повертає JSON-структуру зі списком проектів, які містять основну інформацію, як-от назву, опис і дату створення.
- Клієнт надсилає запит з інформацією про нове завдання (назва, опис, статус та проект). Сервер обробляє запит, створює завдання в базі даних і повертає інформацію про щойно створене завдання.
- Користувач надсилає свої облікові дані (логін і пароль), а сервер перевіряє їх і повертає токен доступу, який зберігається на стороні клієнта для подальших запитів.
- Клієнт (Kanban-дошка або Telegram-бот) надсилає HTTP-запити до сервера для виконання певних дій, таких як отримання списку завдань або створення нового проекту.
- Сервер отримує запит, обробляє його (перевіряє аутентифікацію, валідує дані), взаємодіє з базою даних для отримання чи збереження інформації та формує відповідь у форматі JSON.
- Дані про користувачів, проекти та завдання зберігаються в базі даних. Сервер звертається до бази для отримання та запису інформації відповідно до запитів клієнта.
- Коли клієнт надсилає запит до захищеного ендпоінту, він додає токен до заголовків запиту. Сервер перевіряє токен, і, якщо він дійсний, дозволяє доступ до запитуваного ресурсу.

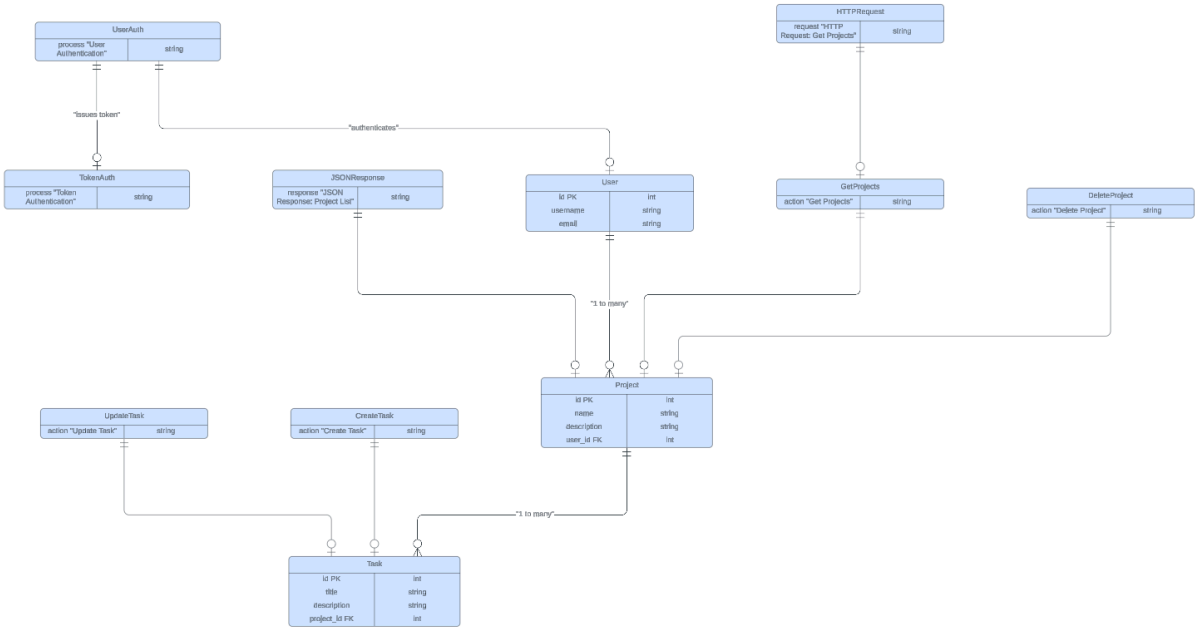

Рисунок 3.9 - Діаграма CRUD-операцій

Розроблене API є важливим компонентом системи управління проектами. Воно забезпечує взаємодію між клієнтською частиною та сервером, дозволяє керувати проектами та завданнями, а також захищає дані користувачів за допомогою токенів. Це API гнучке та масштабоване, що робить його зручним для розширення функціональності та інтеграції з іншими компонентами, такими як Telegram-бот.

3.4 Розробка Telegram бота

Розробка Telegram бота для Kanban-системи є надання користувачам зручного способу управління завданнями безпосередньо з мобільного пристрою або комп'ютера через Telegram. Бот дозволяє здійснювати основні операції, як-от авторизацію, створення, перегляд, переміщення та видалення завдань, а також надає інструменти для ефективної інтеграції із системою управління завданнями на основі Django.

Основні компоненти:

- Команди та обробники, відповідають за авторизацію, перегляд завдань, створення нових завдань.
- Сеанси користувачів: Використовуються для зберігання інформації про авторизованих користувачів.
- Асинхронна обробка: Бот використовує асинхронні функції для забезпечення швидкої обробки запитів і взаємодії з базою даних.

Основні функціональні можливості Telegram бота:

- Авторизація користувачів. Команда /login: Користувачі вводять свої облікові дані (логін та пароль), які перевіряються на сервері Django. У разі успішної авторизації сесія користувача зберігається, що дозволяє йому працювати із завданнями.
- Зберігання сеансів. Використовується словник sessions, де ключем є Telegram ID користувача, а значенням — об'єкт користувача.
- Створення завдань. Команда /add: Користувач може додати нове завдання, вказавши його назву після команди. Бот зберігає нове завдання в базі даних із прив'язкою до користувача та проєкту.
- Перегляд завдань за колонками. Команда /list: Запитує у користувача вибір колонки, завдання з якої слід показати (наприклад, "Заплановано", "В роботі", "На перевірці", "Виконано"). Після вибору колонки користувач отримує список завдань, що містяться в ній.
- Переміщення завдань між колонками.
- Обробка натискання кнопок. Після вибору завдання користувач може вибрати колонку для переміщення. Бот переміщує завдання до обраної колонки та оновлює запис у базі даних.
- Видалення завдань. Бот надає можливість видалення завдань через інтерфейс із кнопками. Користувач може вибрати завдання та натиснути кнопку для його видалення.

- Налаштування нагадувань. Команда /setreminder: Користувач може встановити дату і час для нагадування про завдання, і бот надішле повідомлення користувачу в заданий час.

Діаграма активності показана на рисунку 3.10 ілюструє процес авторизації користувача в системі. Вона демонструє послідовність дій, починаючи з отримання запиту клієнта, введення облікових даних та завершуючи відповіддю від сервера.

Процес виглядає наступним чином:

- початок — система отримує запит від клієнта;
- введення облікових даних — користувач вводить свій логін і пароль;
- передача даних — введені облікові дані надсилаються на сервер;
- перевірка в базі даних — сервер перевіряє, чи існує користувач із такими обліковими даними;
- якщо користувача не знайдено, сервер повертає повідомлення про помилку авторизації;
- якщо облікові дані є коректними, сервер генерує токен доступу;
- відповідь клієнту — сервер відправляє відповідь клієнту:

у разі успішної авторизації користувач отримує доступ до системи;

у разі помилки користувач отримує сповіщення про невдалу спробу авторизації;

- кінець — процес завершується.

Ця діаграма дозволяє візуалізувати ключові етапи авторизації користувача, а також описати сценарії успішної та невдалої авторизації. Вона чітко демонструє, як клієнтська частина та сервер взаємодіють між собою, забезпечуючи безпеку та точність доступу до системи.

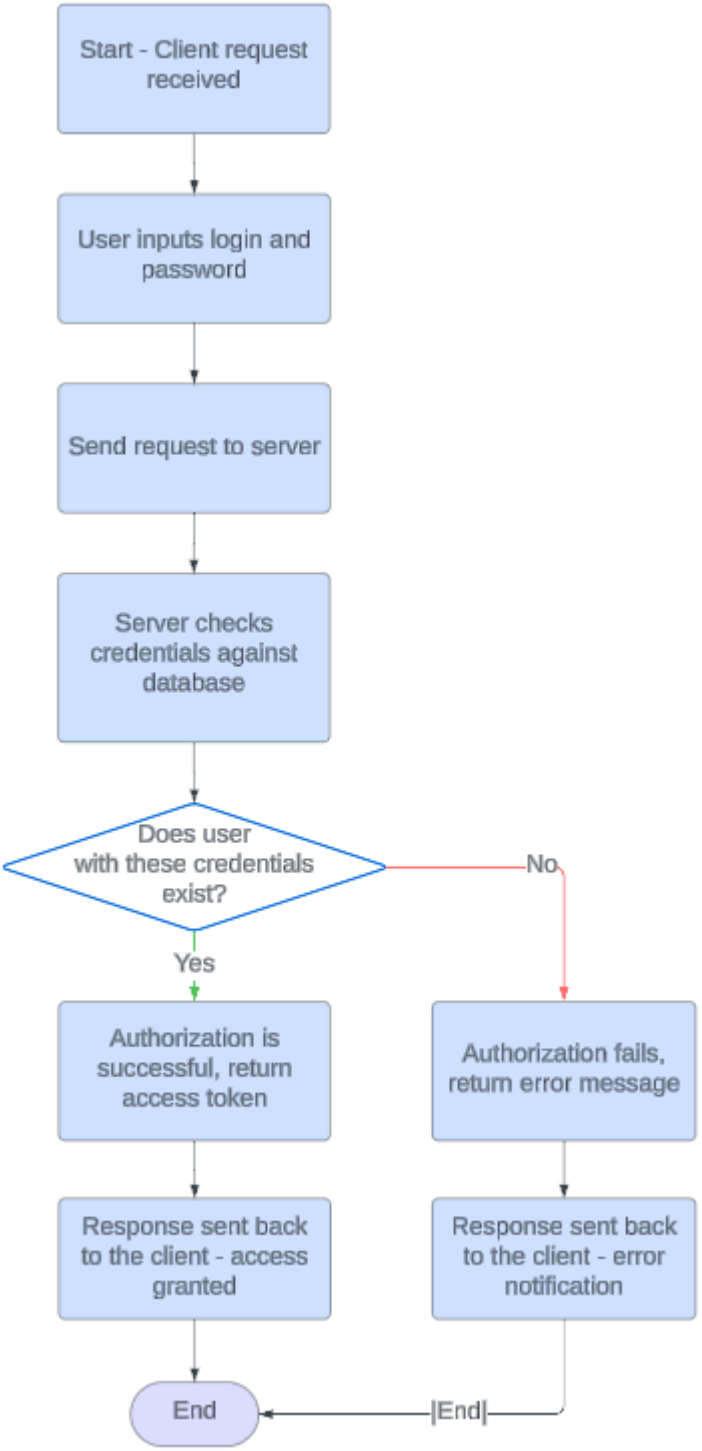

Рисунок 3.10 - Діаграма активності

Опис основних методів:

- Команда start - привітання користувача та пояснення основних команд бота. Це початкова точка для нового користувача.

- Команда login - авторизація користувача в системі. Користувач вводить ім'я та пароль, які перевіряються через Django. У разі успіху бот повідомляє про успішний вхід.
- Команда add_task - створення нового завдання. Користувач вводить команду разом із назвою завдання. Бот додає завдання в базу даних і повертає підтвердження з унікальним ідентифікатором завдання.
- Команда list_tasks - відображення завдань за обраною колонкою. Бот запитує вибір колонки, після чого відображає завдання, що в ній містяться, і пропонує дії (наприклад, переміщення або видалення).
- Метод handle_column_selection - обробляє вибір користувача та виводить список завдань із обраної колонки. Якщо завдання відсутні, бот надсилає повідомлення про це.
- Метод move_task - змінює колонку для завдання, що обирає користувач, і зберігає нове значення в базі даних. Після успішного переміщення користувач отримує повідомлення про оновлений статус.
- Метод delete_task - видаляє завдання, яке вибрав користувач, із бази даних. Якщо завдання знайдено, бот видаляє його та надсилає підтвердження користувачеві.
- Метод set_task_reminder - встановлює нагадування для завдання. Користувач задає дату та час, і бот надсилає повідомлення в зазначений час, нагадуючи про завдання.

Бот пов'язаний із Django через моделі та аутентифікацію. Основні запити до бази даних обробляються асинхронно за допомогою sync_to_async для зменшення затримок при взаємодії з базою даних. Це забезпечує ефективну роботу та швидкий обмін даними.

Telegram бот є інтерактивним інструментом для роботи із завданнями, який використовує сервер і базу даних для обробки запитів користувачів. Уся інформація, яку бот отримує чи змінює, проходить через базу даних за допомогою ORM Django.

Процес взаємодії - отримання даних, коли користувач надсилає команду, наприклад, /list, бот надсилає запит до бази даних для отримання списку завдань у

конкретній колонці. Через ORM Django виконується SQL-запит до таблиці Task, і результати передаються боту. Створення записів, після команди /add <назва завдання> бот формує новий запис у таблиці Task, використовуючи ORM. Оновлення записів при переміщенні завдання між колонками команда /move оновлює поле boardName у таблиці Task. Видалення записів, команда /delete видаляє завдання з бази даних.

У коді використовується модуль logging для запису подій роботи бота. Це дозволяє відстежувати будь-які помилки, включаючи проблеми з підключенням до бази даних або невірні запити користувачів. Усі критичні помилки обробляються асинхронно, щоб уникнути збоїв у роботі.

Telegram бот для Kanban-системи надає користувачам можливість керувати завданнями без необхідності доступу до веб-інтерфейсу, що підвищує мобільність і зручність використання. Реалізація асинхронної обробки запитів та інтеграція з Django забезпечує надійну роботу бота й ефективну взаємодію з базою даних.

3.5 Огляд та тестування додатку

Для перевірки коректної роботи додатку було проведено тестування на декількох рівнях: API, клієнтська частина (Kanban-дошка), Telegram бот, і база даних.

Розроблений додаток є системою управління завданнями, що включає такі компоненти:

- Kanban-дошка для візуалізації завдань у категоріях.
- Telegram бот для інтерактивної роботи користувачів із системою.
- REST API, що забезпечує взаємодію між клієнтською частиною, ботом і сервером.
- База даних, у якій зберігаються всі дані про користувачів, завдання та проекти.

Основні функціональні можливості:

- Авторизація користувачів через веб-інтерфейс (Рисунок 3.11).

- Реєстрація через веб-інтерфейс (Рисунок 3.13).
- Управління проектами створення, редагування, видалення проектів (Рисунок 3.12).
- Управління завданнями, додавання нових завдань, зміна їх статусу, переміщення між категоріями та налаштування дизайну (Рисунок 3.14).
- Нагадування можливість встановлювати нагадування про завдання.
- Telegram бот дозволяє виконувати більшість функцій через мобільний пристрій.

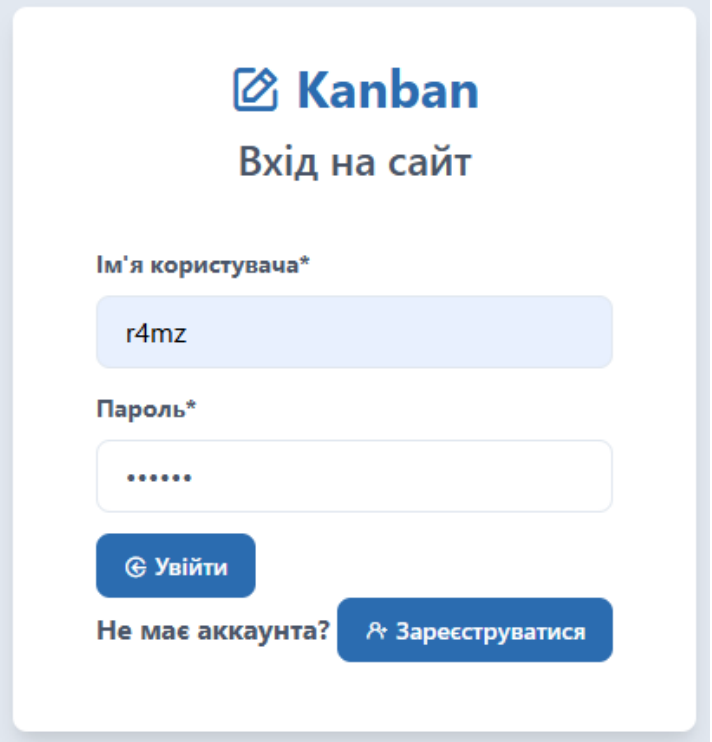

Рисунок 3.11 - Перевірка входу на сайт

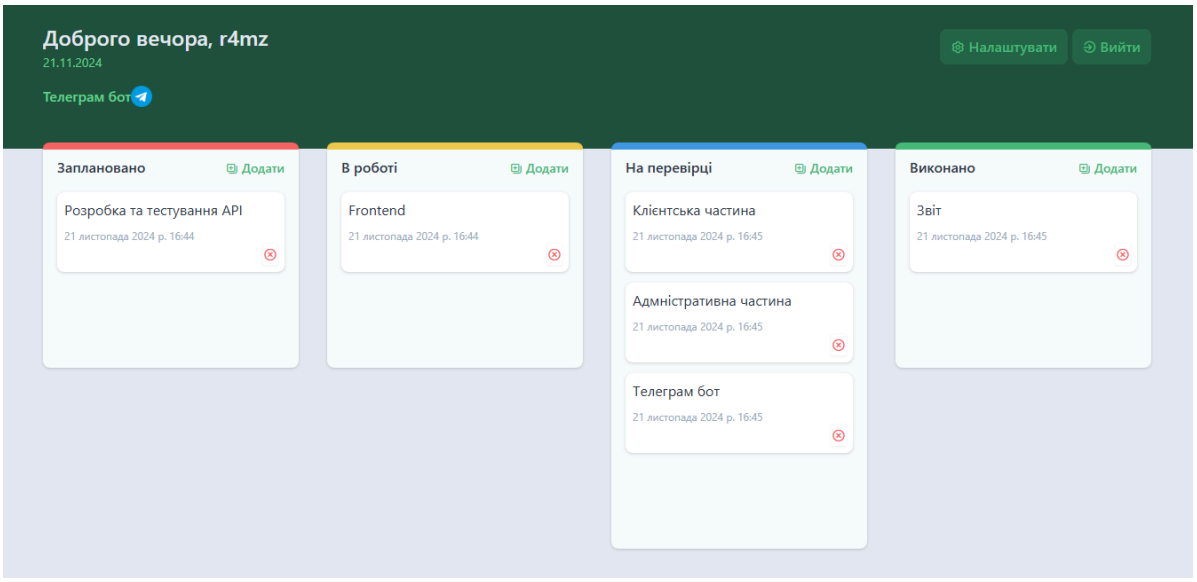

Рисунок 3.12 - Головна сторінка для керування поточними завданнями

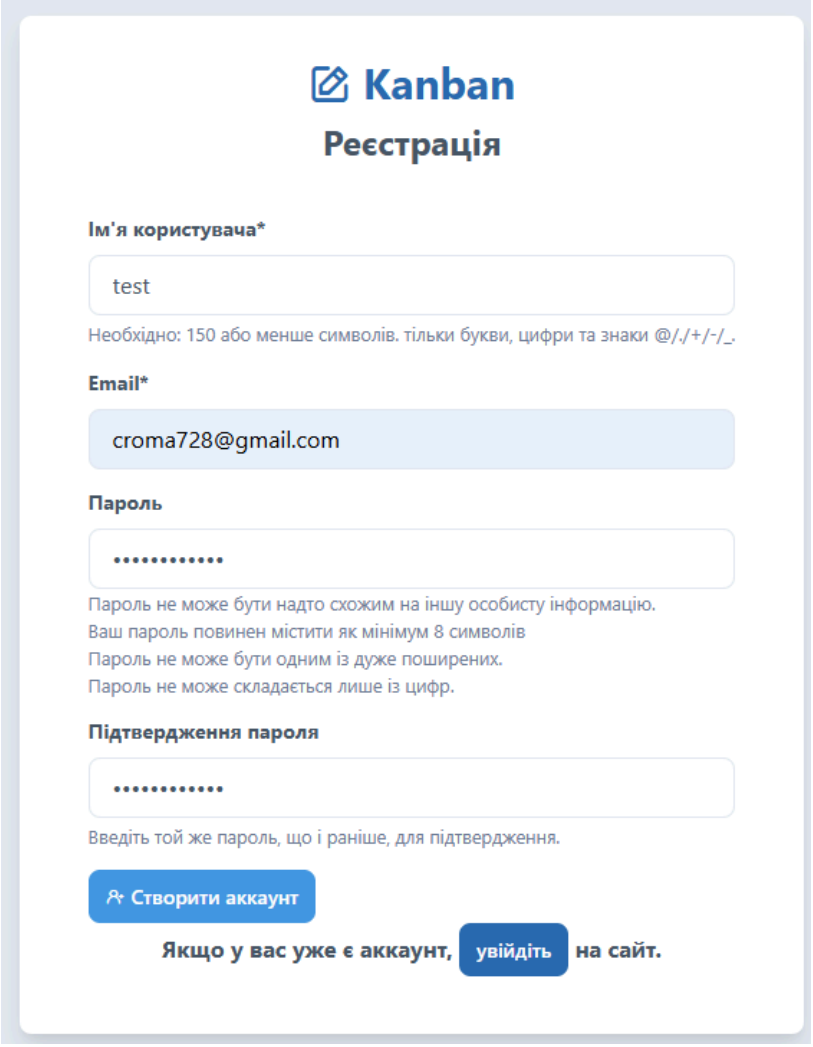

Рисунок 3.13 - Тестування сторінки реєстрації

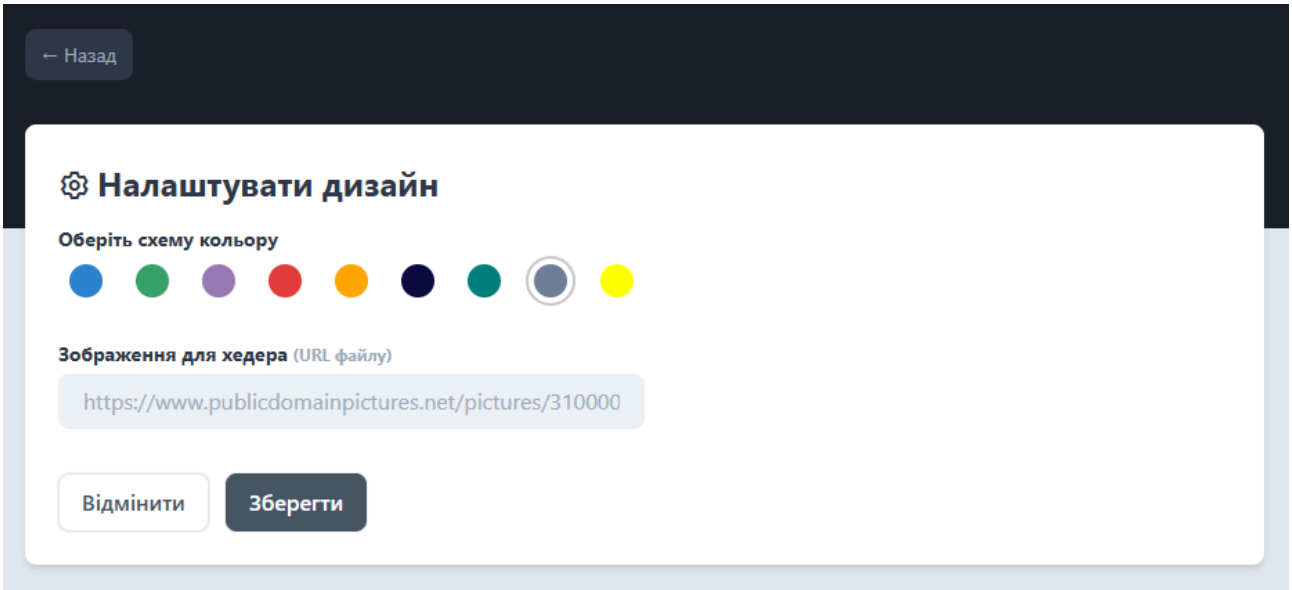

Рисунок 3.14 - Сторінка налаштування дизайну

Тестування включало наступні етапи:

- Тестування API. API було перевірено на відповідність основним вимогам:
  - Створення, отримання, редагування та видалення даних працюють коректно.
  - Токени аутентифікації видаються правильно, і доступ до захищених ресурсів обмежений неавторизованим користувачам.
  - API коректно обробляє помилкові дані, повертаючи відповідні повідомлення про помилку.
  - Додатково було проведено тестування адміністративної частини Django (Рисунок 3.15).
- Тестування Kanban-дошки. Kanban-дошка була протестована з акцентом на зручність користувача:
  - Всі елементи розташовані логічно, завдання створюються та переміщуються без затримок.
  - Дані на дошці оновлюються в реальному часі після змін через бот або API.
  - Завдання групуються відповідно до їх статусу (категорії).

- Тестування Telegram бота. Telegram бот був перевірений на функціональність через реальні сценарії використання:
    - Авторизація та збереження сеансу користувача.
    - Створення нових завдань через команду /add.
    - Перегляд завдань за колонками за допомогою команди /list.
    - Видалення та переміщення завдань через кнопки.
- Тестування бази даних. База даних була перевірена на цілісність і продуктивність:
    - Усі зв’язки між таблицями працюють коректно.
    - Дані зберігаються, оновлюються і видаляються відповідно до запитів (Рисунок 3.16).
- Інтеграційне тестування. Було перевірено взаємодію між усіма компонентами системи:
    - Telegram бот, Kanban-дошка та API працюють у єдиній екосистемі.
    - Дані, введені через один компонент (наприклад, бот), синхронно відображаються в інших.

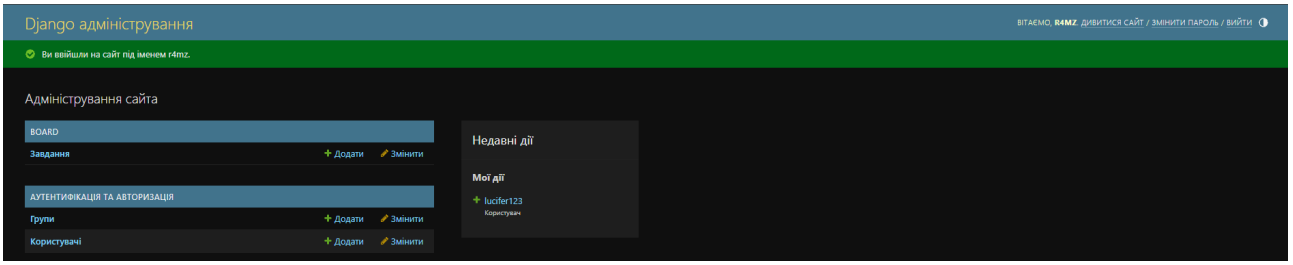

Рисунок 3.15 - Сторінка адміністрування

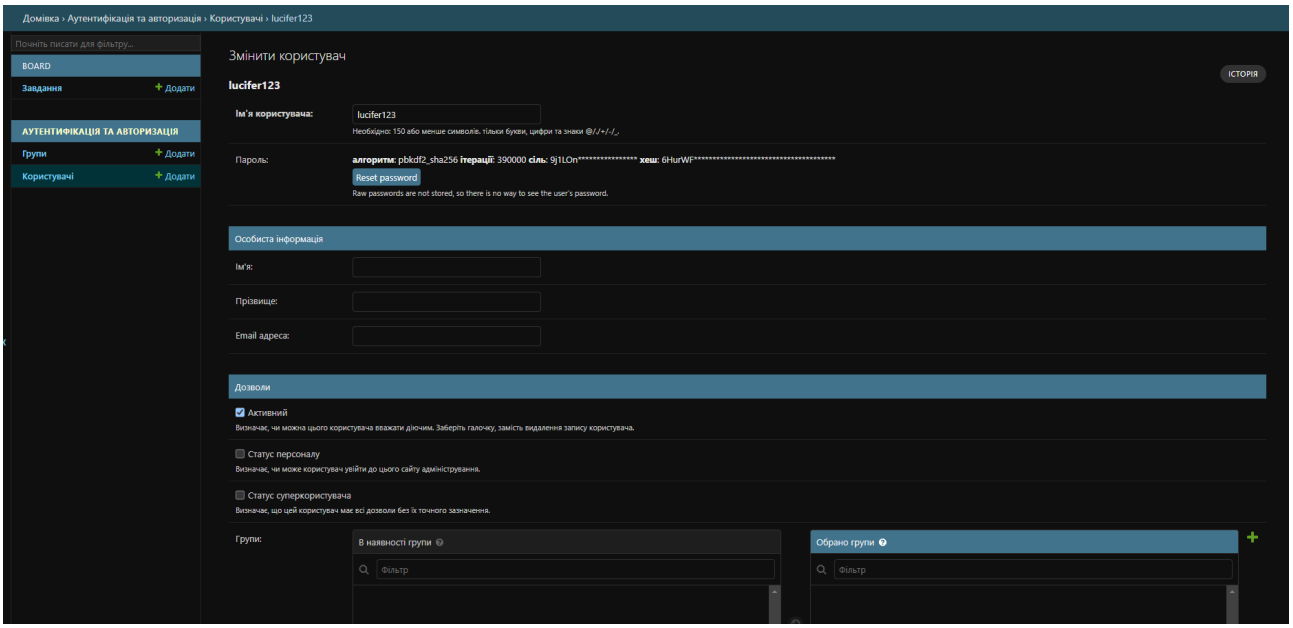

Рисунок 3.16 - Керування користувачами у системі

Результати тестування.

API:

- Усі CRUD-операції виконуються коректно.
- Захищені ендпоінти доступні лише для авторизованих користувачів.
- Токен-аутентифікація працює стабільно.
- Адмін-панель працює належним чином.

Kanban-дошка:

- Відображає всі завдання відповідно до їх статусів.
- Графічний інтерфейс інтуїтивно зрозумілий і зручний для користувача.
- Дані синхронізуються миттєво з сервером.

Telegram бот:

- Відповідає на команди швидко та коректно (Рисунок 3.17).
- Всі функції (авторизація, додавання, перегляд, переміщення, видалення завдань) працюють без помилок (Рисунок 3.18).
- Користувач отримує зрозумілі повідомлення у разі помилок (наприклад, некоректний логін).

База даних:

- Зберігає всі дані в узгодженій структурі.

- Відсутні проблеми із збереженням або видаленням даних.
- Використання тестових сценаріїв

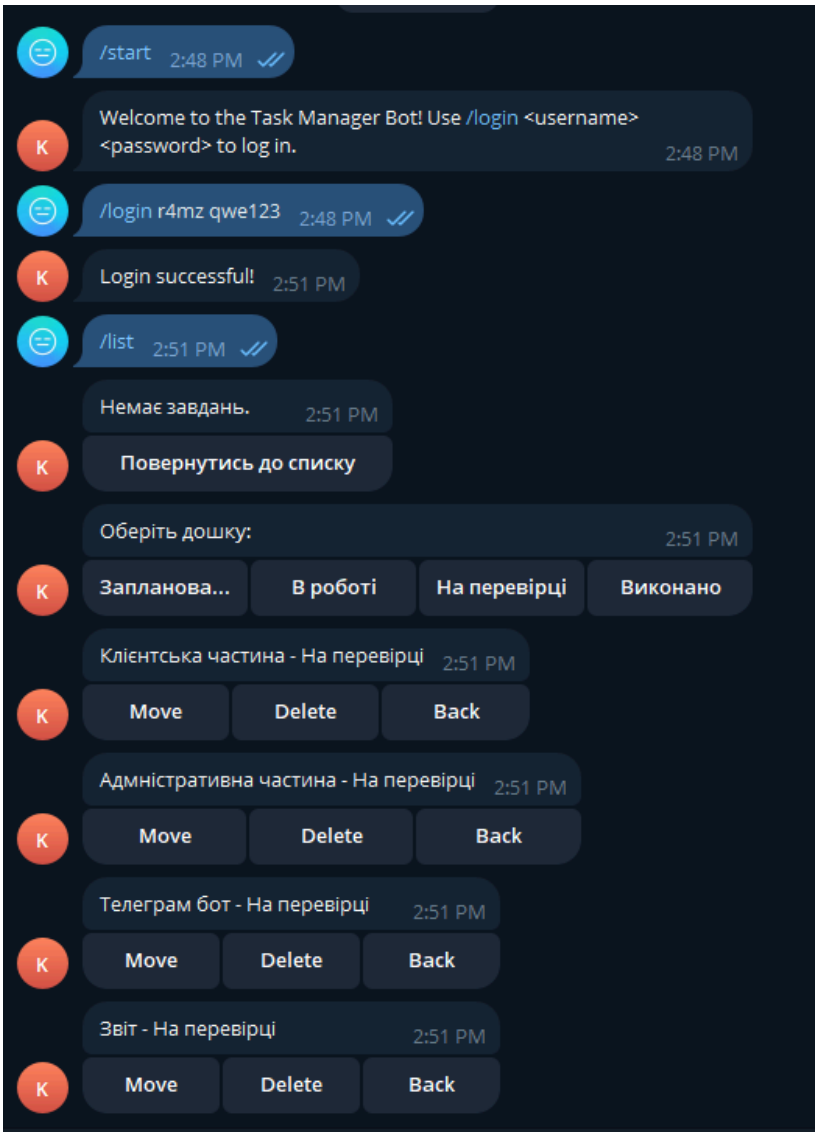

Рисунок 3.17 - Управління завданнями через Телеграм бота

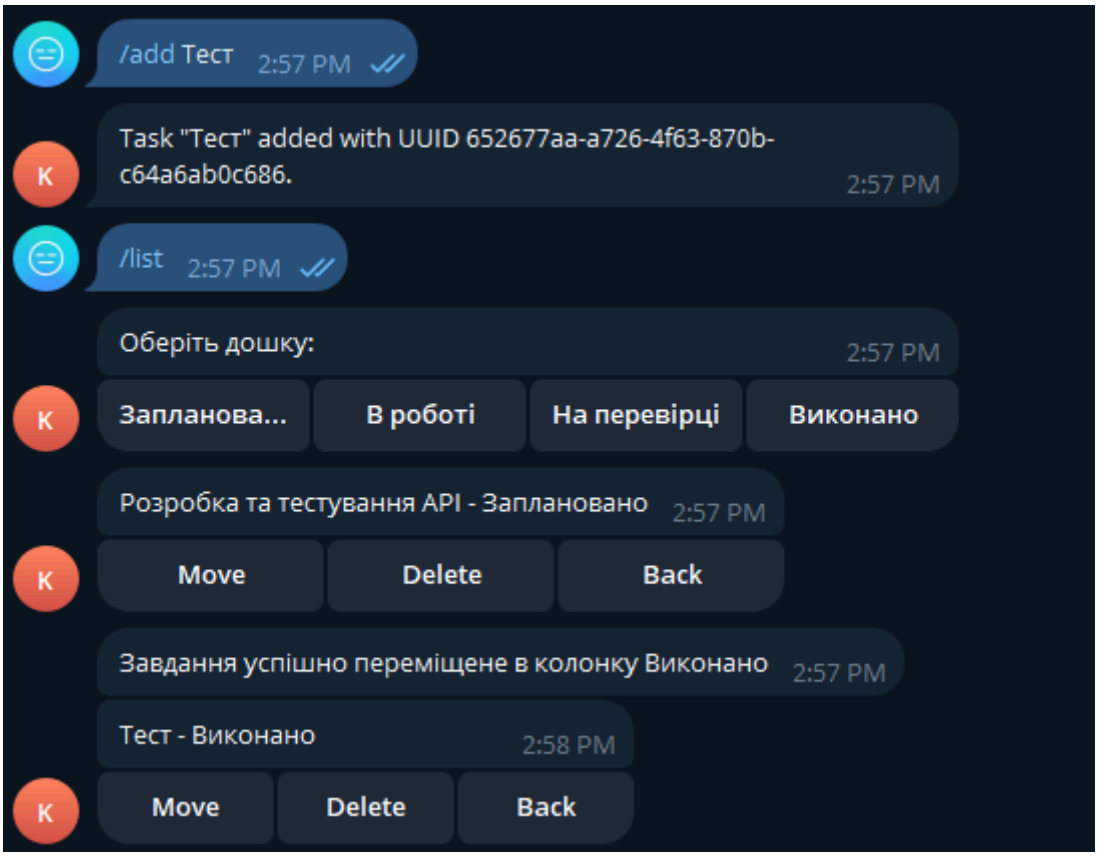

Рисунок 3.18 - Тестування функцій додавання та переміщення завдань

Висновки до розділу

Кожен компонент системи було перевірено за попередньо підготовленими тестовими сценаріями.

- Сценарій 1. Авторизація користувача в Telegram боті, створення нового завдання, його переміщення в іншу категорію.
- Сценарій 2. Додавання завдання через Kanban-дошку та перевірка його доступності через бот.
- Сценарій 3. Видалення проекту через API та перевірка видалення всіх пов’язаних завдань.

Тестування підтвердило, що всі функції додатку працюють коректно, а система відповідає вимогам до надійності та продуктивності. Виявлені під час тестування недоліки були оперативно усунуті. Система готова до впровадження, забезпечуючи користувачам зручність управління проектами через веб-інтерфейс і Telegram бот.

4. ЕКОНОМІЧНИЙ РОЗДІЛ

4.1 Технологічний аудит розробленої автоматизованої системи управління проєктами для оптимізаці завдань на підприємстві

Загальновідомо, що впровадження автоматизованих систем управління проєктами з розподілом завдань між виконавцями стало важливим елементом для досягнення ефективності, прозорості та оптимізації роботи команди. Оскільки більшість сучасних проєктів мають складну структуру і передбачають розв’язання великої кількості задач, то це вимагає від керівників налагодження чіткого планування, контролю та дієвої комунікації між учасниками. Разом з тим, традиційні методи управління проєктами часто не здатні забезпечити достатній рівень координації та управління, що призводить до втрати часу, перевищення бюджету та зниження продуктивності.

Одним з ефективних шляхів цієї проблеми є використання автоматизованих систем управління проєктами, які дозволяють централізовано планувати та контролювати виконання завдань, відстежувати хід процесу, розподіляти ресурси та забезпечувати активну взаємодію між учасниками проєкту в реальному часі. Особливо актуальною є така система управління для великих підприємств, де одночасно виконується кілька проєктів, а команди, що їх виконують, можуть бути розподілені по різних підрозділах і навіть розділені територіально.

Тому метою нашої кваліфікаційної роботи і було розроблення такої автоматизованої системи управління проєктами, яку можна було б використовувати для оптимізації процесів розподілу, контролю та виконання завдань, пов’язаних з розробкою та реалізацією цих проєктів.

Для досягнення поставленої мети нами було: проаналізовано методи розробки веб-сайтів; вибрано технологічні засоби для створення веб-сторінки; виконано проєктування моделі та обгрунтовано послідовність її розробки; розроблено веб-сайт канбан-дошки та Telegram-бот.

В результаті нами було розроблено автоматизовану систему управління проєктами, що поєднує веб-додаток канбан-дошки та Telegram-бот, забезпечує оперативну взаємодію всіх членів команди, дозволяє централізовано розподіляти завдання між членами команди, забезпечує їх візуалізацію та контроль виконання на різних етапах.

Для встановлення комерційного потенціалу розробленої нами автоматизованої системи управління проєктами було запрошено 3-х висококваліфікованих експертів: к.т.н., доцента кафедри АІІТ Я.А. Кулика, к.т.н., доцента кафедри АІІТ В.В. Гармаша та к.т.н., доцента кафедри АІІТ Р.В. Маслія.

Запрошені експерти зробили експертне оцінювання комерційного потенціалу розробленої ними автоматизованої системи управління проєктами за критеріями, наведеними в таблиці 5.1.

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу будь-якої розробки і їх бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
Ринкові перспективи					
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційні витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів

Продовження таблиці 5.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Експерти оцінили комерційний потенціал розробленої нами автоматизованої системи управління проєктами і зробили висновки, які зведено в таблицю 5.2:

Таблиця 5.2 – Результати оцінювання комерційного потенціалу розробленої нами автоматизованої системи управління проєктами
(за 5-ти бальною шкалою оцінювання 0 - 1 – 2 – 3 - 4)

Критерії	Ініціали, прізвище експертів		
	Я.А. Кулик	В.В. Гармаш	Р.В. Маслій
	Бали, що їх виставили експерти:		
1	3	2	3
2	3	3	2
3	3	2	3
4	3	3	3
5	3	2	3
6	3	3	3
7	3	3	3
8	4	3	3
9	3	3	3
10	3	3	3
11	3	3	3
12	3	3	3
Сума балів	$СБ_1 = 37$	$СБ_2 = 33$	$СБ_3 = 35$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{3} = \frac{37+33+35}{3} = \frac{105}{3} = 35,00$		

Для встановлення комерційного потенціалу розробленої нами автоматизованої системи управління проєктами були використані рекомендації, які наведено в таблиці 5.3 [31].

Таблиця 5.3 – Рівні комерційного потенціалу будь-якої наукової розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Оскільки середньоарифметична сума балів, що їх виставили експерти, складає 35,00 бал (із максимально можливих 48-ми балів), то це свідчить, що розроблена нами автоматизована система управління проєктами має рівень комерційного потенціалу, який можна вважати «вище середнього».

Це пояснюється тим, що а) наша система дозволяє більш ефективно розподі-ляти завдання, контролювати їх виконання і пріоритетність, що сприяє зменшен-ню витрат часу і ресурсів; б) завдяки доступності інформації в реальному часі доз-воляє відслідковувати хід виконання проєктів та забезпеченість команди потріб-ними ресурсами; в) інтегрує інструменти, які дозволяють користувачам отриму-вати швидкий зворотний зв'язок та обмінюватися інформацією без необхідності залучення сторонніх сервісів, таких як месенджери чи електронна пошта; г) збирає дані про продуктивність команди проєкту, що дозволяє керівникам аналі-зувати показники ефективності та вносити зміни до стратегії управління проєк-тами; д) надає зручний інтерфейс для віддаленого доступу до завдань, що важливо для забезпечення мобільності та оперативності в сучасному робочому середовищі.

4.2 Розрахунок витрат на розроблення автоматизованої системи управління проєктами

При розробленні автоматизованої системи управління проєктами було зроблено низку основних витрат:

а). Основна заробітна плата $З_о$ розробників, які здійснювали розроблення цієї системи. Величина цієї зарплати визначається за формулою:

$$З_о = \frac{М}{Т_р} \cdot t \quad \text{грн,} \qquad (5.1)$$

де М – місячний посадовий оклад розробника (дослідника), грн;

Для 2024 року приймемо, що:

М = (8000…28000) грн/місяць;

$Т_р$ – число робочих днів в місяці; приймемо $Т_р$ = 25 днів;

t – число днів роботи розробників (дослідників тощо).

Зроблені розрахунки основної заробітної плати розробників зведемо до таблиці 5.4:

Таблиця 5.4 – Основна заробітна плата розробників

Найменування посади виконавця	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів (годин) роботи	Витрати на оплату праці, грн
1. Науковий керівник магістерської роботи	24000	960	20 годин	(960 / 6) × 20 = 3200 (при 6-годинному робочому дні)
2. Здобувач-магістрант (виконавець)	2000 грн (беремо 8000 грн)	320	80 днів	25600
3. Консультант з економічної частини	20000	800	1,5 години	(800 /6) × 1,5 ≈ 200 грн (при 6-годинному робочому дні)
Загалом				$З_о$ = 29000 грн

Б). Додаткова заробітна плата $З_д$ розробників розраховується як (10…12)% від величини їх основної заробітної плати, тобто:

$$З_д = \alpha \cdot З_о = (0{,}1...0{,}12) \cdot З_о . \quad (5.2)$$

Приймемо, що α = 0,1. Тоді для нашого випадку отримаємо:

$$З_д = 0{,}1 \times 29000 = 2900 \text{ грн.}$$

В). Нарахування на заробітну плату $НЗП_{зп}$ розробників (дослідників) розраховуються за формулою:

$$НЗП_{зп} = (З_о + З_д) \cdot \frac{\beta}{100}, \quad (5.3)$$

де β – ставка обов'язкового єдиного внеску на державне соціальне страхування, %. В 2024 році ставка β = 22%. Тоді:

$$НЗН_{зп} = (29000 + 2900) \times 0{,}22 = 7018 \text{ грн.}$$

Г). Амортизація основних засобів А, які використовувались під час виконання магістерської кваліфікаційної роботи:

$$A = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12} \text{ грн,} \qquad (5.4)$$

де Ц – загальна балансова вартість основних засобів, грн;

$Н_а$ – річна норма амортизаційних відрахувань.

Приймемо, що $Н_а$ = (2,5...25)%;

Т – термін використання основних засобів, місяці.

Зроблені розрахунки зведено в таблицю 5.5.

Таблиця 5.5 – Розрахунок амортизаційних відрахувань

Найменування обладнання, приміщень тощо	Балансова вартість, грн.	Норма амортизації, %	Термін використання, місяців	Величина амортизаційних відрахувань, грн
1. Комп'ютерна техніка, обладнання тощо	80000	25	3,0 (при 90% використанні)	4500
2. Приміщення університету, факультету, кафедри	40000	5,0	3,0 (при 50% використанні)	250
Всього				А = 4750 грн

Д). Витрати на матеріали М розраховуються за формулою:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_в \text{ грн,} \qquad (5.5)$$

де $Н_i$ – витрати матеріалу *i*-го найменування, кг; $Ц_i$ – вартість матеріалу *i*-го най-менування; $К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15); $В_i$ – маса відходів матеріалу *i*-го найменування; $Ц_в$ – ціна відходів матеріалу *i*-го найменування; n – кількість видів матеріалів.

Е). Витрати на комплектуючі К розраховуються за формулою:

$$К = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i \text{ грн,} \qquad (5.6)$$

де $Н_i$ – кількість комплектуючих *i*-го виду, шт.; $Ц_i$ – ціна комплектуючих *i*-го виду; $К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15); n – кількість видів комплектуючих.

Під час виконання магістерської кваліфікаційної роботи загальні витрати на матеріали та комплектуючі склали укрупнено приблизно 500 грн.

Ж). Витрати на силову електроенергію $В_е$ розраховуються за формулою:

$$В_е = \frac{В \cdot П \cdot Ф \cdot К_п}{К_д}, \quad (5.7)$$

де В – вартість 1 кВт-год. електроенергії, в 2024 р. В ≈ 4,8 грн/кВт;

П – установлена потужність обладнання, кВт; П = 1,15 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

Приймемо, що Ф = 200 годин;

$К_п$ – коефіцієнт використання потужності; $К_п$ < 1 = 0,78.

$К_д$ – коефіцієнт корисної дії, $К_д$ = 0,69.

Тоді витрати на силову електроенергію будуть дорівнювати:

$$В_е = \frac{В \cdot Ф \cdot К_п}{К_д} = \frac{4,8 \cdot 1,15 \cdot 200 \cdot 0,78}{0,69} = 1248 \text{ грн.}$$

И). Інші витрати $В_{інш}$ можна прийняти як (50…300)% від основної заробітної плати розробників, тобто:

$$В_{інш} = (0,5....3) \times З_о. \quad (5.8)$$

Для нашого випадку отримаємо:

$$В_{інш} = 0,5 \times 29000 = 14500 \text{ грн.}$$

К). Сума всіх попередніх статей витрат складає витрати на виконання магістерської роботи безпосередньо розробником-магістрантом – В.

$$В = 29000 + 2900 + 7018 + 4750 + 500 + 1248 + 14500 = 59916 \text{ грн.}$$

Л). Загальні витрати на розробку та можливу комерціалізацію розробленої нами автоматизованої системи управління проєктами будуть становити:

$$В_{заг} = \frac{В}{\beta}, \quad (5.9)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання цієї роботи.

Оскільки наша розробка на цей момент часу має високий ступінь готовності, то можна умовно прийняти, що, $\beta \approx 0,9$.

Тоді: $В_{заг} = \frac{59916}{0,9} = 66573,33$ грн або приблизно 67 тисяч грн.

Тобто прогнозовані загальні витрати на розробку та можливу комерціалізацію розробленої автоматизованої системи управління проєктами становлять приблизно 67 тисяч грн.

4.3 Розрахунок економічного ефекту від можливої комерціалізації розробленої автоматизованої системи управління проєктами

Проведене дослідження ринку показало, що розроблена нами автоматизована система управління проєктами орієнтована на широкий спектр користувачів і може стати в нагоді різним підприємствам, організаціям, установам, навчальним закладам тощо, а також індивідуальним користувачам. Аналіз місткості ринку також показує, що на сьогодні в Україні кількість реальних користувачів подібних автоматизована система управління проєктами може становити до 1000 користувачів. Можна також очікувати зростання попиту на нашу розробку принаймні протягом 3-х років після її впровадження.

Тобто, якщо наша розробка буде впроваджена з 1 січня 2025 року, то її результати будуть виявлятися протягом 2025-го, 2026-го та 2027-го років.

Прогноз зростання попиту на нашу розробку може складати по роках:

а) 2025 р. – приблизно + 20 шт. (відносно базового року);

б) 2026 р. – +30 шт. (відносно базового року);

в) 2027 р. – +40 шт. (відносно базового року).

Економічний ефект від можливої комерціалізації розробленої нами автоматизована система управління проєктами пояснюється її значно кращими функціональними можливостями. Аналіз ринку показує, що сьогодні середня вартість подібних систем перебуває в межах $400 або 19 тисяч грн. А оскільки розроблена нами автоматизована система управління проєктами має значно кращі функціональні можливості, то її можна буде реалізовувати на ринку дещо дорожче,

ніж аналогічні розробки, наприклад, реалізовувати в середньому за 20 тисяч грн, тобто на 1 тисячу грн дорожче.

Тоді можливе збільшення чистого прибутку $\Delta П_i$, що його може отримати потенційний інвестор від комерціалізації нашої розробки, тобто виведення її на ринок, становитиме:

$$\Delta П_i = \sum_{1}^{n} (\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\upsilon}{100}), \qquad (5.10)$$

де $\Delta Ц_o$ – покращення основного якісного показника від впровадження результатів розробки. Для нашого випадку це є збільшення ціни реалізації нашої розробки $\Delta Ц_o$ = 20 - 19 = + 1 тисяч грн;

N – основний кількісний показник, який визначає обсяг діяльності у році до впровадження результатів розробки; N = 1000 шт.;

Δ N – покращення основного кількісного показника від впровадження результатів розробки. Таке покращення основного кількісного показника становитиме по роках, у 2025 році – + 20 шт., у 2026 році + 30 шт., у 2027 році +40 шт. (відносно базового 2024 року);

$Ц_o$ – основний якісний показник (тобто ціна), який являє собою ціну реалізації нашої розробки після її виведення на ринок, грн; $Ц_o$= 20 тисяч грн;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки; для нашого випадку n = 3;

λ – коефіцієнт, який враховує сплату податку на додану вартість; $\lambda = 0{,}8333$;

ρ – коефіцієнт, який враховує рентабельність продукту. Рекомендується приймати ρ = (0,2...0,5); візьмемо ρ = 0,5;

υ – ставка податку на прибуток. У 2024 році υ = 18%.

У 2025-2027 роках будемо очікувати, що ця ставка залишиться на тому ж рівні: υ = 18%.

Тоді можливе зростання чистого прибутку $\Delta П_1$ для потенційного інвестора протягом першого року від можливої комерціалізації нашої розробки (2025 р.) становитиме:

$$\Delta П_1 = [1 \cdot 1000 + 20 \cdot 20] \cdot 0,8333 \cdot 0,5 \cdot (1 - \frac{18}{100}) = 478,31 \approx 479$$ тисяч грн.

Можливе зростання чистого прибутку $\Delta П_2$ для потенційного інвестора від можливої комерціалізації нашої розробки протягом другого (2026 р.) року становитиме:

$$\Delta П_2 = [1 \cdot 1000 + 20 \cdot 30] \cdot 0,8333 \cdot 0,5 \cdot (1 - \frac{18}{100}) = 546,64 \approx 547$$ тисяч грн.

Можливе зростання чистого прибутку $\Delta П_3$ для потенційного інвестора від можливої комерціалізації нашої розробки протягом третього (2027 р.) року становитиме:

$$\Delta П_3 = [1 \cdot 1000 + 20 \cdot 40] \cdot 0,8333 \cdot 0,5 \cdot (1 - \frac{18}{100}) = 614,97 \approx 615$$ тис. грн.

Приведена вартість зростання для потенційного інвестора всіх чистих прибутків від можливої комерціалізації нашої розробки становитиме:

$$ПП = \sum_{1}^{т} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої нашої роботи, грн;

т – період часу, протягом якого виявляються результати впровадженої роботи, роки. Для нашого випадку т = 3 роки;

τ – ставка дисконтування (рівень інфляції). Приймемо τ = 0,10 (10%);

t – період часу від моменту початку розроблення автоматизованої системи управління проєктами до моменту отримання можливих чистих прибутків від її впровадження і виведення на ринок.

Тоді приведена вартість зростання всіх можливих чистих прибутків ПП, що їх може отримати потенційний інвестор від комерціалізації нашої розробки, складе:

$$ПП = \frac{479}{(1+0,1)^2} + \frac{547}{(1+0,1)^3} + \frac{615}{(1+0,1)^4} \approx 396 + 411 + 420 = 1227$$ тисяч грн.

Теперішня вартість інвестицій PV, що мають бути вкладені в комерціалізацію нашої розробки: $PV = К \times В_{заг} = (1,0…5,0) \times В_{заг}$.

Для нашого випадку приймемо, що:

$PV = (1,0…5,0) \times 67 = 3,0 \times 67 = 201$ тисяч грн.

Розраховуємо абсолютний ефект, який може отримати потенційний інвестор від можливих вкладених інвестицій $Е_{абс}$.

$$Е_{абс} = ПП – PV, \qquad (5.12)$$

де ПП – приведена вартість збільшення всіх чистих прибутків для інвестора від можливої комерціалізації нашої розробки, грн.

Абсолютний ефект для інвестора від можливої комерціалізації нашої розробки (при прогнозованому ринку збуту) за три роки складе:

$$Е_{абс} = 1227 – 201 = 1026 \text{ тисяч грн.}$$

Оскільки $Е_{абс} > 0$, то комерціалізація нашої розробки може бути доцільною.

Далі розрахуємо внутрішню дохідність $Е_в$ вкладених інвестицій (коштів):

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1, \qquad (5.13)$$

де $Е_{абс}$ – абсолютний ефект вкладених інвестицій; $Е_{абс} = 1026$ тисяч грн;

PV –теперішня вартість початкових інвестицій PV = 201 тисяч грн;

$Т_ж$ – життєвий цикл розробки, роки.

$Т_ж = 4$ роки (2024-й, 2025-й, 2026-й, 2027-й роки)

Для нашого випадку отримаємо:

$$Е_в = \sqrt[4]{1 + \frac{1026}{201}} - 1 = \sqrt[4]{1 + 5,1045} - 1 = \sqrt[4]{6,1045} - 1 = 1,57 - 1 = 0,57 \approx 57,0\%.$$

Далі визначимо ту мінімальну дохідність, нижче за яку потенційному інвестору не вигідно буде займатися комерціалізацією нашої розробки.

Мінімальна дохідність $\tau_{мін}$ визначається за формулою:

$$\tau_{мін} = d + f, \qquad (5.14)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d = (0,10...0,18). Приймемо, що $\tau = 15\%$.

f – показник, що характеризує ризикованість вкладень; f = (0,05...0,30).

Приймемо, що f = 30%, тобто f = 0,3.

Тоді для нашого випадку отримаємо:

$$\tau_{\text{мін}} = 0,15 + 0,30 = 0,45 \text{ або } \tau_{\text{мін}} = 45\%.$$

Оскільки величина $E_в = 57,0\% > \tau_{\text{мін}} = 45\%$, то потенційний інвестор у принципі може бути зацікавлений у комерціалізації розробленої нами автоматизованої системи управління проєктами.

Далі розраховуємо термін окупності коштів, вкладених у можливу комерціалізацію розробленої нами автоматизованої системи управління проєктами.

Термін окупності $T_{ок}$ розраховується за формулою:

$$T_{ок} = \frac{1}{E_в} = \frac{1}{0,57} = 1,75 \text{ років} < 3 \text{ років} \qquad (5.15)$$

що свідчить про потенційну доцільність комерціалізації розробленої нами автоматизованої системи управління проєктами.

Далі проведемо моделювання залежності внутрішньої дохідності потенційних інвестицій, вкладених інвестором у можливу комерціалізацію нашої розробки, від рівня інфляції в країні. На жаль, в наступні роки через військову агресію росії рівень інфляції в Україні може суттєво зрости.

Прийнявши рівень інфляції у 20%, отримаємо:

$$\text{ПП} = \frac{479}{(1+0,2)^2} + \frac{547}{(1+0,2)^3} + \frac{615}{(1+0,2)^4} \approx 333 + 317 + 297 = 947 \text{ тисяч грн.}$$

Тоді абсолютний економічний ефект від можливого впровадження нашої розробки за три роки складе:

$$E_{абс} = 947 - 201 = 746 \text{ тисяч грн.}$$

Внутрішня дохідність $E_в$ вкладених інвестицій становитиме:

$$E_в = \sqrt[4]{1 + \frac{746}{201}} - 1 = \sqrt[4]{1 + 3,7114} - 1 = \sqrt[4]{4,7114} - 1 = 1,475 - 1 = 0,473 \approx 47,3\%.$$

Оскільки величина $E_в = 47,3\% > \tau_{мін} = 45\%$, то потенційний інвестор також може бути зацікавлений у комерціалізації розробленої нами автоматизованої системи управління проєктами.

Прийнявши рівень інфляції у 30%, отримаємо:

$$ПП = \frac{479}{(1+0,3)^2} + \frac{547}{(1+0,3)^3} + \frac{615}{(1+0,3)^4} \approx 283 + 249 + 215 = 747 \text{ тисяч грн.}$$

Тоді абсолютний економічний ефект від можливого впровадження нашої розробки за три роки складе:

$$E_{абс} = 747 - 201 = 546 \text{ тисяч грн.}$$

Внутрішня дохідність $E_в$ вкладених інвестицій становитиме:

$$E_в = \sqrt[4]{1+\frac{646}{201}} - 1 = \sqrt[4]{1+2,7164} - 1 = \sqrt[4]{3,7164} - 1 = 1,388 - 1 = 0,388 \approx 38,8\%.$$

Оскільки величина $E_в = 38,8\% < \tau_{мін} = 45\%$, то потенційний інвестор може бути не зацікавлений у комерціалізації розробленої нами автоматизованої системи управління проєктами. Але для прийняття остаточного рішення потрібно провести додаткові обгрунтування і розрахунки (наприклад, знизити рівень прийнятого ризику, підняти ціну реалізації нашої розробки, збільшити попит на нашу розробку тощо).

Зроблені розрахунки у вигляді діаграм наведено на Рисунок 5.1.

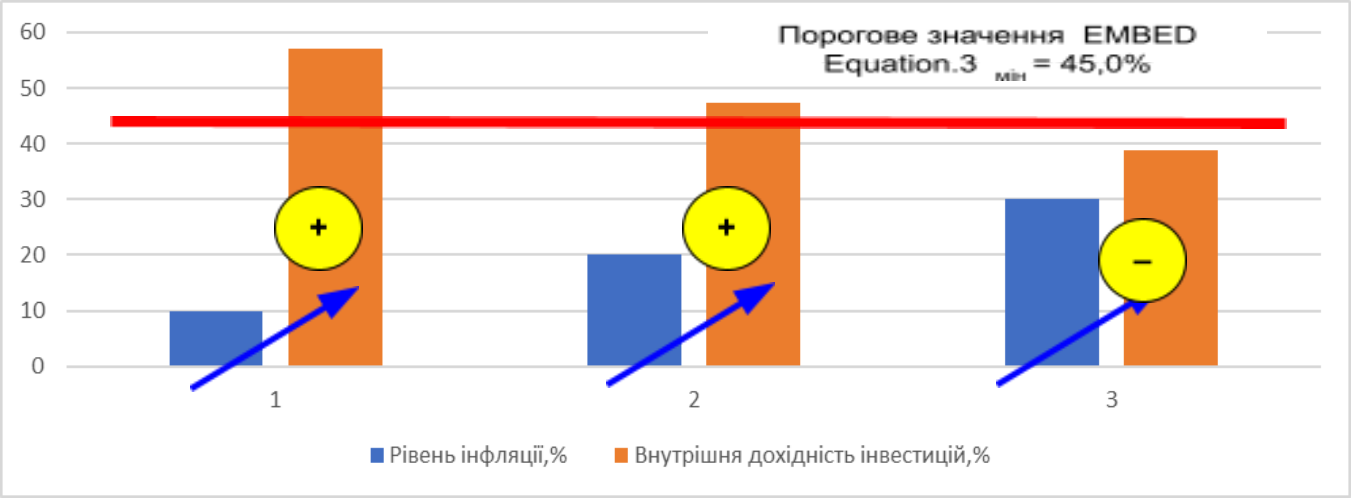

Рисунок 5.1 – Моделювання залежності величини внутрішньої дохідності потенційних інвестицій, які можуть бути вкладені у комерціалізацію нашої розробки, від рівня інфляції в країні

Результати виконаної економічної частини магістерської кваліфікаційної роботи зведено у таблицю:

Показники	Задані у ТЗ	Досягнуті у магістерській кваліфікаційній роботі	Висновок
1. Витрати на розробку	Не більше 70 тисяч грн	67 тисяч грн	Досягнуто
2. Абсолютний ефект від впровадженн розробки, тисяч грн	Не менше 1000 тисяч грн (за три роки)	1026 тисячі грн (при 10% інфляції)	Виконано
3. Внутрішня дохідність інвестиц (коштів), %	не менше 45,0%	57,0%	Виконано
4. Термін окупнос інвестицій (коштів), роки	до 3-ти років	1,75 років	Виконано

Таким чином, основні техніко-економічні показники розробленої нами автоматизованої системи управління проєктами, визначені у технічному завданні, повністю виконані.

ВИСНОВКИ

У ході виконання роботи було розроблено автоматизовану систему управління проектами, яка включає Kanban-дошку, Telegram бот та REST API. Ця система забезпечує зручне управління завданнями, покращує організацію роботи команди та підвищує ефективність процесів.

Реалізовано всі основні функції: авторизацію користувачів, створення та редагування завдань, їх переміщення між дошками, а також інтерактивну взаємодію через Telegram бот. REST API став ключовим елементом, що забезпечив взаємодію між компонентами системи.

Система створена із застосуванням сучасних технологій, таких як Django Rest Framework, PostgreSQL та Python-telegram-bot, що гарантує її надійність і масштабованість. Проведене тестування підтвердило коректність роботи всіх компонентів і відповідність функціональності початковим вимогам.

Проте виявлено кілька недоліків, таких як відсутність оновлень у реальному часі та обмежена фільтрація даних через API. Це відкриває можливості для подальшого вдосконалення системи.

Розроблена система готова до впровадження і може бути ефективно використана для автоматизації управління проектами у малих і середніх підприємствах. Її адаптивність дозволяє розширювати функціональність залежно від потреб користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кіндрацька Г. І. Сучасні інформаційні технології в управлінні проектами // Менеджмент та підприємництво в Україні. 2021. Вип. 1. С. 123–132.
2. Титомир Р.Р. РОЗРОБКА ВЕБ-ДОДАТКУ УПРАВЛІННЯ ПРОЕКТАМИ З БОТОМ НА ТЕЛЕГРАМ. LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації [Електронний ресурс]. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2023/paper/view/17869 (дата звернення 01.04.2024).
3. Титомир Р.Р. Розробка веб-сайту канбан-дошки за допомогою python фреймворку Django. LII науково-технічнічної конференції професорсько-викладацького складу, співробітників та студентів ВНТУ [Електронний ресурс]. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/20504 (дата звернення 10.06.2023).
4. Балабанов І. Т. Управління проектами: навчальний посібник. Київ: Кондор, 2020. 256 с.
5. HTML. [Електронний ресурс] // uk.wikipedia.org. – 2023 – URL: https://uk.wikipedia.org/wiki/HTML (дата звернення 10.01.2024).
6. Методи розробки сайтів. [Електронний ресурс] // webstudio2u.net. – 2018 – URL: https://webstudio2u.net/ua/webdesign/354-site-develop-methods.html (дата звернення 10.01.2024).
7. Статичні та динамічні веб-сторінки. [Електронний ресурс] // webstudio2u.net. – 2018 – URL: https://webstudio2u.net/ua/design-web/391-static-or-dynamic.html (дата звернення 10.01.2024).
8. Денисенко Андрій. Фреймворки у веб-розробці. [Електронний ресурс] // highload.today. – 2022 – URL: https://highload.today/uk/frejmvorki-u-veb-rozrobtsi-shho-tse-yaki-isnuyut-i-dlya-chogo-potribni/ (дата звернення 10.01.2024).

9. Прикладний програмний інтерфейс. [Електронний ресурс] // uk.wikipedia.org. – 2023 – URL: https://uk.wikipedia.org/wiki/Прикладний програмний інтерфейс (дата звернення 10.01.2024).
10. Kumar, A., & Rajasekaran, M. "Chatbots: The New Wave of Artificial Intelligence." International Journal of Applied Engineering Research. 2019. Vol. 14(5), P. 350–356.
11. Василенко В. А., Сивак Л. В. Системи управління проектами: особливості та перспективи розвитку // Економічний простір. 2021. Вип. 10. С. 14–22.
12. Maxim, T., & Sanchez, J. Modern Project Management Approaches for IT Companies // International Journal of Project Management. 2020. Vol. 38(4). P. 234–242.
13. Павленко І. П. Канбан-дошка як інструмент управління проектами в сучасних умовах // Проектний менеджмент і розробка програмного забезпечення. 2022. Вип. 12. С. 44–53.
14. L. Bass, P. Clements, and R. Kazman, “Software architecture in practice,” Addison-Wesley Professional, vol. 3, no. 1, pp. 1-50, 2013.
15. M. Fowler, “Patterns of enterprise application architecture,” Addison-Wesley, vol. 1, no. 1, pp. 1-400, 2002.
16. D. Crockford, “JavaScript: The good parts,” O'Reilly Media, vol. 1, no. 1, pp. 1-150, 2008.
17. D. Beazley and B. K. Jones, “Python cookbook: Recipes for mastering Python 3,” O'Reilly Media, vol. 3, no. 1, pp. 1-800, 2013.
18. M. Summerfield, “Programming in Python 3: A complete introduction to the Python language,” Addison-Wesley Professional, vol. 2, no. 1, pp. 1-520, 2010.
19. A. Holovaty and J. Kaplan-Moss, “The definitive guide to Django: Web development done right,” Apress, vol. 2, no. 1, pp. 1-600, 2009.
20. D. Greenfeld and A. Roy, “Two scoops of Django 3.x: Best practices for the Django web framework,” Two Scoops Press, vol. 3, no. 1, pp. 1-550, 2020.
21. Telegram. Telegram API documentation. [Електронний ресурс] // core.telegram.org. – 2024 – URL: https://core.telegram.org/bots/api (дата звернення 09.10.2024)
22. M. Kleppmann, “Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems,” O'Reilly Media, vol. 1, no. 1, pp. 1-600, 2017.

23. M. Fowler, “NoSQL distilled: A brief guide to the emerging world of polyglot persistence,” Addison-Wesley, vol. 1, no. 1, pp. 1-200, 2003.
24. K. Chodorow, “MongoDB: The definitive guide,” O'Reilly Media, vol. 3, no. 1, pp. 1-400, 2013.
25. E. Hewitt, “Cassandra: The definitive guide,” O'Reilly Media, vol. 1, no. 1, pp. 1-500, 2010.
26. B. Albahari and J. Albahari, “PostgreSQL for developers: A practical guide,” Pragmatic Bookshelf, vol. 1, no. 1, pp. 1-200, 2021.
27. B. Forta, “SQL in 10 minutes, Sams teach yourself,” Sams Publishing, vol. 5, no. 1, pp. 1-240, 2018.
28. B. Bruegge and A. Dutoit, “Object-oriented software engineering using UML, patterns, and Java,” Prentice Hall, vol. 3, no. 1, pp. 1-700, 2010.
29. JetBrains. PyCharm documentation. [Електронний ресурс] // www.jetbrains.com. – 2024 – URL: https://www.jetbrains.com/pycharm/features/ (дата звернення 09.10.2024).
30. M. Potthoff, “Mastering PyCharm: Best practices and productivity tips for experienced Python developers,” Packt Publishing, vol. 1, no. 1, pp. 1-330, 2019.
31. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. / Укладачі В.О. Козловський, О.Й. Лесько, В.В.Кавецький. Вінниця : ВНТУ, 2021. 42 с.
32. Методичні вказівки до виконання магістерських кваліфікаційних робіт для студентів спеціальностей: 174 «Автоматизація, комп’ютерно-інтегровані технології та робототехніка». / Уклад.: О.В. Бісікало, Ю.Ю. Іванов, Р.В. Маслій. – Вінниця : ВНТУ, 2023, 64 с.

ДОДАТКИ

91

Додаток А

(Обов'язковий)

Технічне завдання

ВНТУ

ЗАТВЕРДЖЕНО

зав. кафедри АІІТ ВНТУ, д.т.н., проф.

_____ Олег БІСІКАЛО

"14" жовтня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

робка автоматизованої системи управління проектами для оптимізації завдань на підприємстві

08-31.МКР.010.02.000 ТЗ

Студент групи 1АКІТР-23м

_____ Роман ТИТОМИР

Підпис *Ім'я ПРІЗВИЩЕ*

Керівник: к.т.н., доцент кафедри АПТ

_____ Марія БАРАБАН

Підпис *Ім'я ПРІЗВИЩЕ*

1. Назва та галузь застосування

1.1. Назва - розробка автоматизованої системи управління проектами для оптимізації завдань на підприємстві.

1.2. Галузь застосування – електронна комерція.

2. Підстава для проведення розробки.

Тема магістерської кваліфікаційної роботи затверджена наказом по ВНТУ №310 від “9” вересня 2024 р.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є покращення ефективності управління проектами завдяки використанню розробленої системи автоматизації для оптимізації процесів розподілу, контролю та виконання завдань на підприємстві.

4. Джерела розробки.

Магістерська кваліфікаційна робота виконується вперше. В ході проведення розробки повинні використовуватись такі документи:

1. Кіндрацька Г. І. Сучасні інформаційні технології в управлінні проектами // Менеджмент та підприємництво в Україні. 2021. Вип. 1. С. 123–132.
2. HTML. 2023. URL: https://uk.wikipedia.org/wiki/HTML (дата звернення 10.01.2024).
3. Kumar, A., & Rajasekaran, M. "Chatbots: The New Wave of Artificial Intelligence." International Journal of Applied Engineering Research. 2019. Vol. 14(5), P. 350–356.
4. D. Beazley and B. K. Jones, “Python cookbook: Recipes for mastering Python 3,” O'Reilly Media, vol. 3, no. 1, pp. 1-800, 2013.

5. Вимоги до розробки.

5.1.Перелік головних функцій:

- додавання, видалення, зміна контенту веб-додатку за допомогою розробленої системи;
- адміністрування користувачами та групами;
- інтегрований телеграм бот.

5.2.Основні технічні вимоги до розробки.

5.2.1. Вимоги до програмної платформи:

- WINDOWS 10/11;
- PyCharm;
- обсяг оперативної пам’яті: не менше 4 ГБ;
- обсяг дискового простору для зберігання даних не менше 10 ГБ;
- середній обсяг даних, які обробляються щодня до 1 ГБ;
- частота процесора не менше 2 ГГц, 2 або більше ядер.

5.2.2. Умови експлуатації системи:

- робота на мобільних пристроях, веб-браузерах та через Telegram;
- можливість цілодобового функціонування системи;
- дані оновлюються і є актуальними.

6. Стадії та етапи розробки.

6.1 Пояснювальна записка:

1. Аналіз методів, принципів, підходів і засобів реалізації задачі автоматизації процесами в об’єкті управління відповідно до теми дипломної роботи. Постановка задач дослідження «03»_09__ 2024 р.
2. Вибір оптимальних інформаційних технологій «15»жовтня 2024 р.
3. Огляд аналогів існуючих автоматизованих систем управління проектами «20»жовтня 2024 р.
4. Аналіз та вибір технологічного стеку та технологій для розробки автоматизованого веб-додатку. «22»жовтня 2024 р.
5. Програмна реалізація «4»листопада 2024р.

6.2 Графічні матеріали:

1. Er діаграми сутностей зв’язків у базі даних «1»листопада 2024 р
2. Розробка діаграм системи «4»листопада 2024 р
3. Проміжні результати, коди запитів, результат виконаної сторінки «10»листопада 2024 р

7. Порядок контролю і приймання.

7.1. Хід виконання роботи контролюється керівником роботи. Рубіжний контроль провести до «15» 11 2024 р.

7.2. Атестація проєкту здійснюється на попередньому захисті. Попередній захист магістерської кваліфікаційної роботи провести до «1» грудня 2024 р.

7.3. Підсумкове рішення щодо оцінки якості виконання роботи приймається на засіданні ЕК. Захист магістерської кваліфікаційної роботи провести до «20» грудня 2024 р.

Додаток Б
(Обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

Автоматизована система управління проектами для оптимізації завдань на підприємстві

Студент групи 1АКІТР-23м
Роман ТИТОМИР
Ім'я ПРІЗВИЩЕ

Керівник: к.т.н., доцент кафедри АІІТ

Марія БАРАБАН
Ім'я ПРІЗВИЩЕ

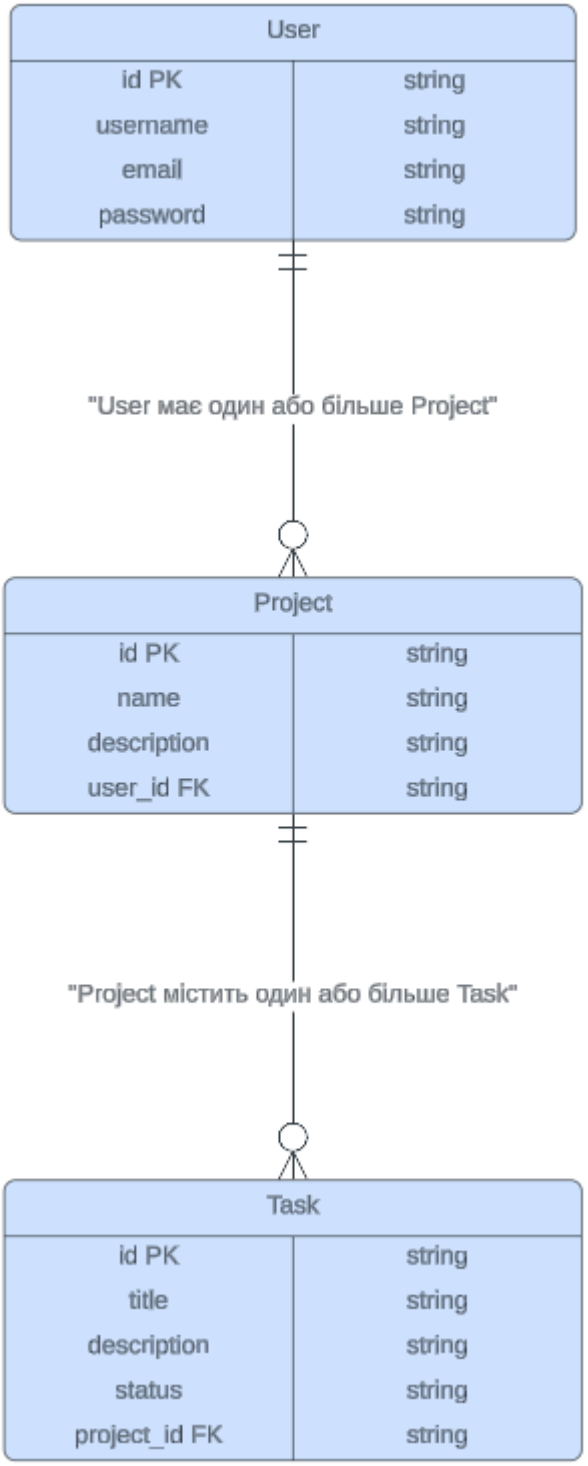

Рисунок Б.1 - Er діаграми сутностей зв’язків у базі даних

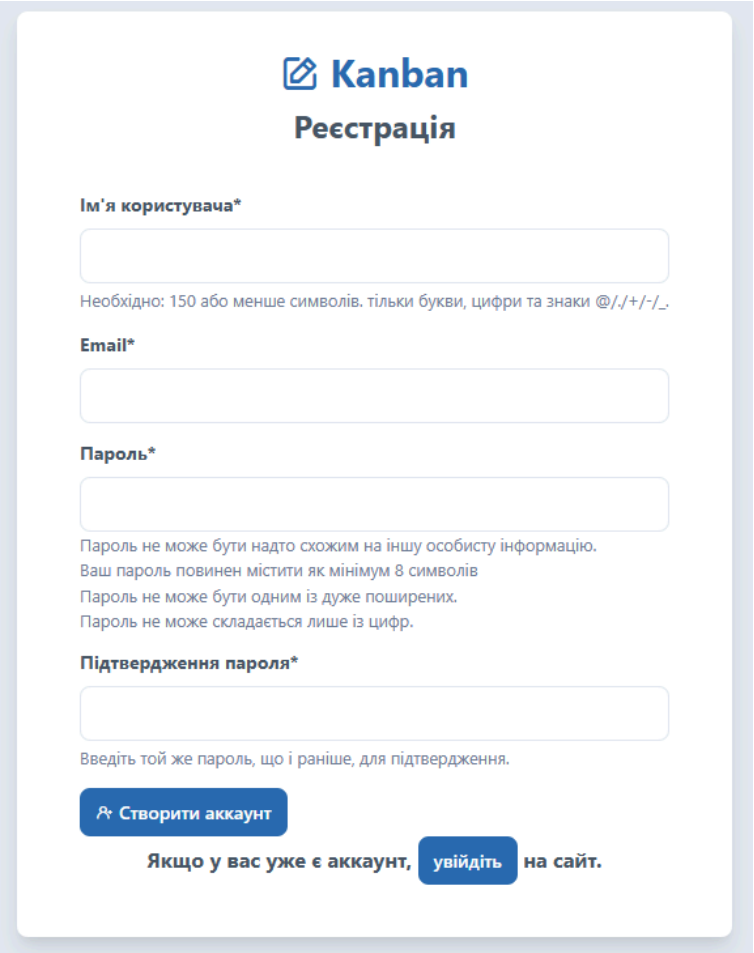

Рисунок Б.2 - Сторінка реєстрації

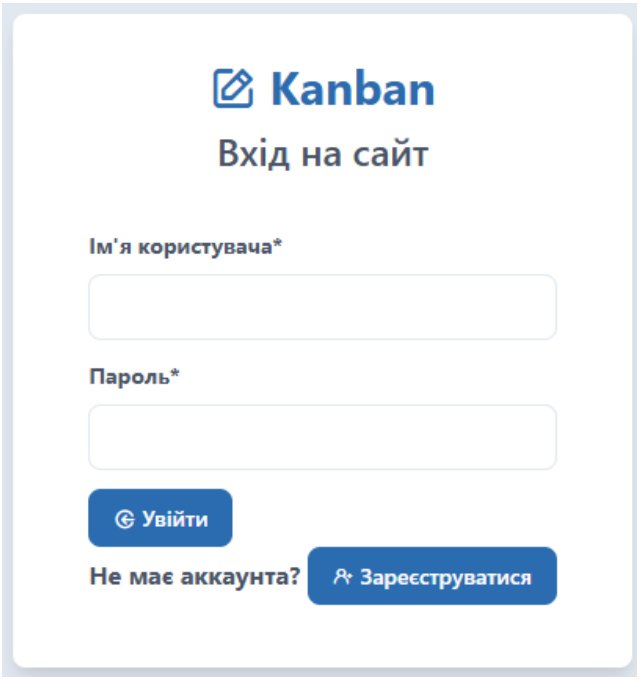

Рисунок Б.3 - Сторінка входу

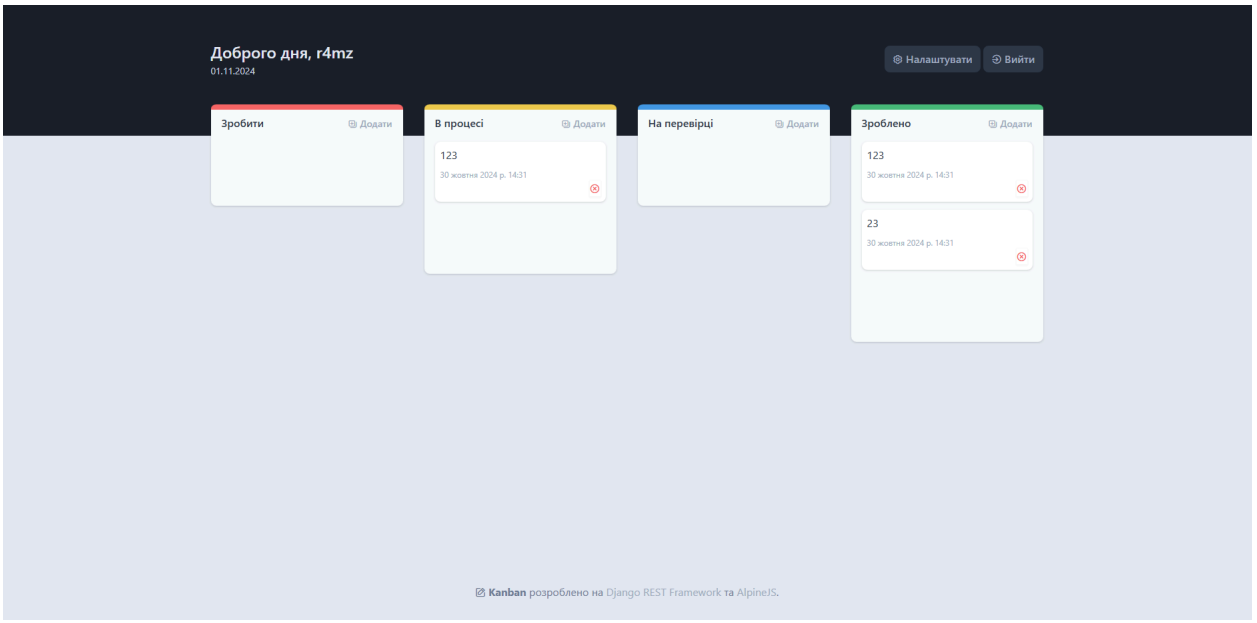

Рисунок Б.4 - Головна сторінка

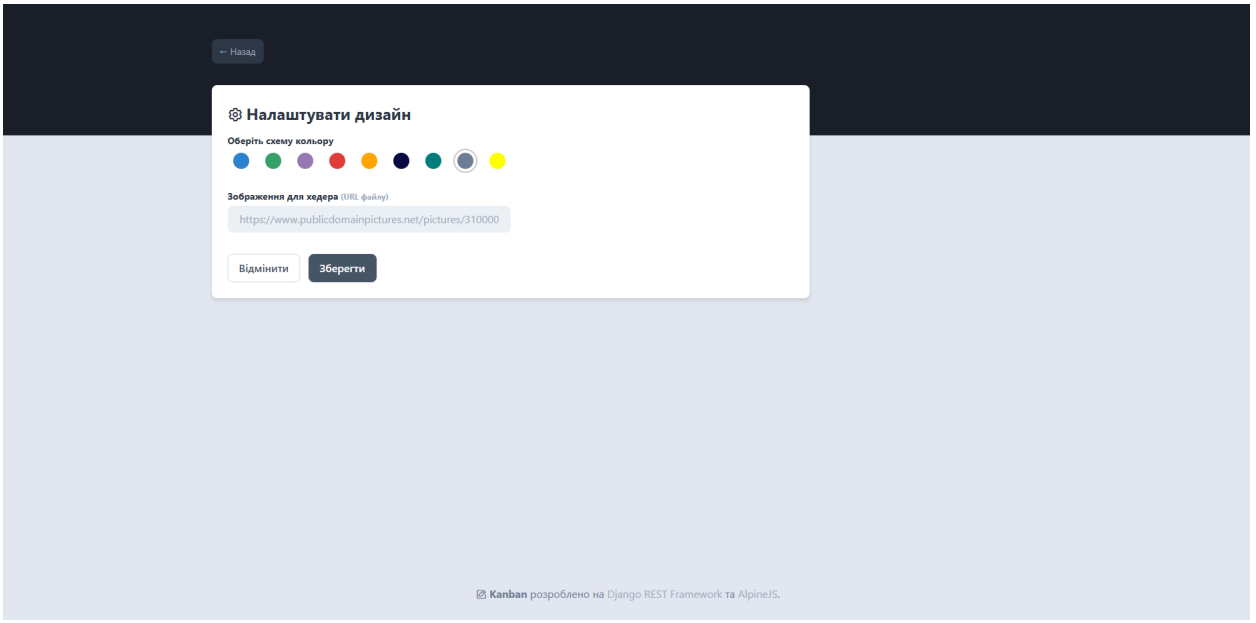

Рисунок Б.5 - Сторінка налаштувань

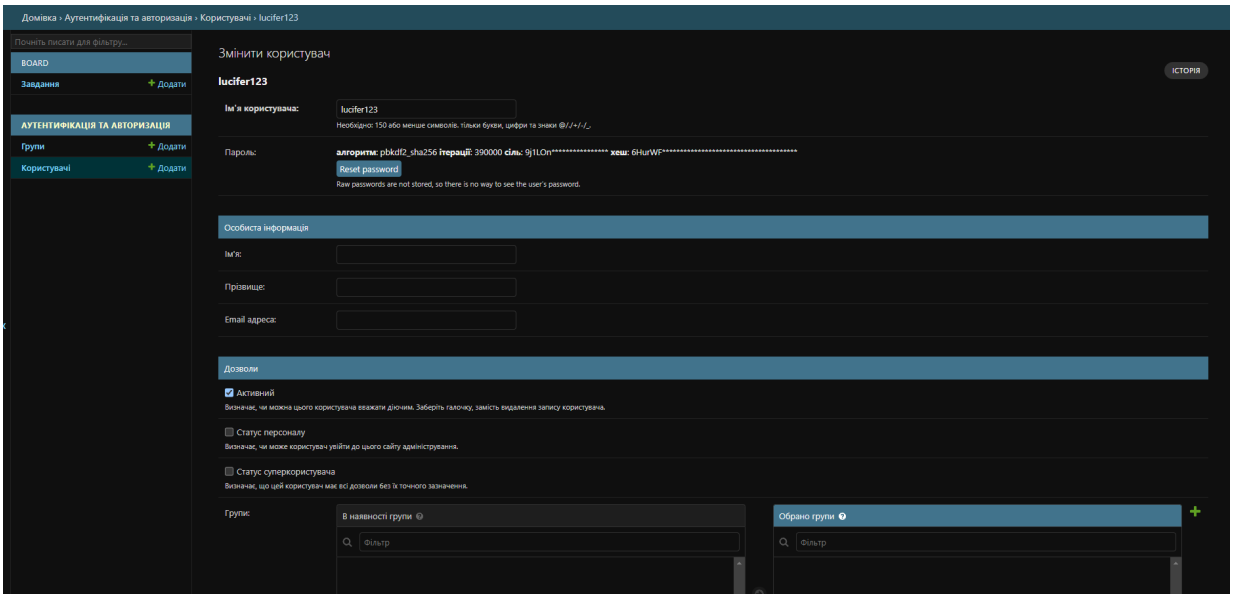

Рисунок Б.6 - Сторінка адміністрування

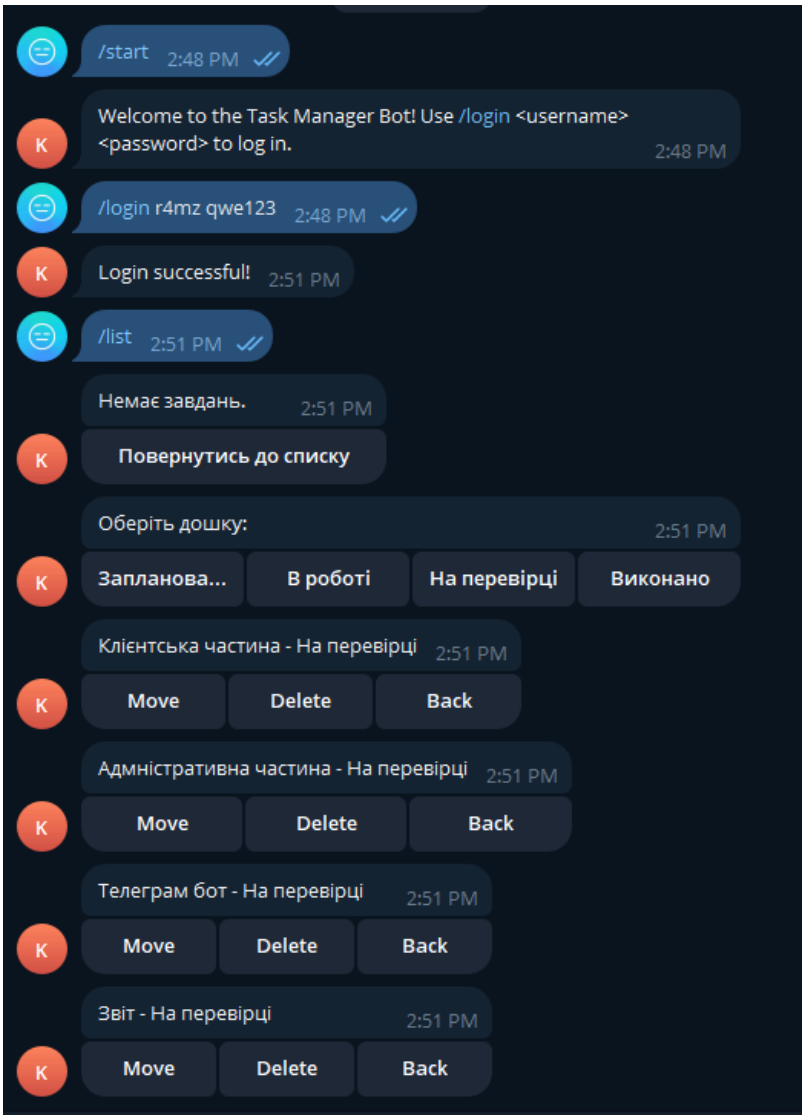

Рисунок Б.7 - Телеграм бот

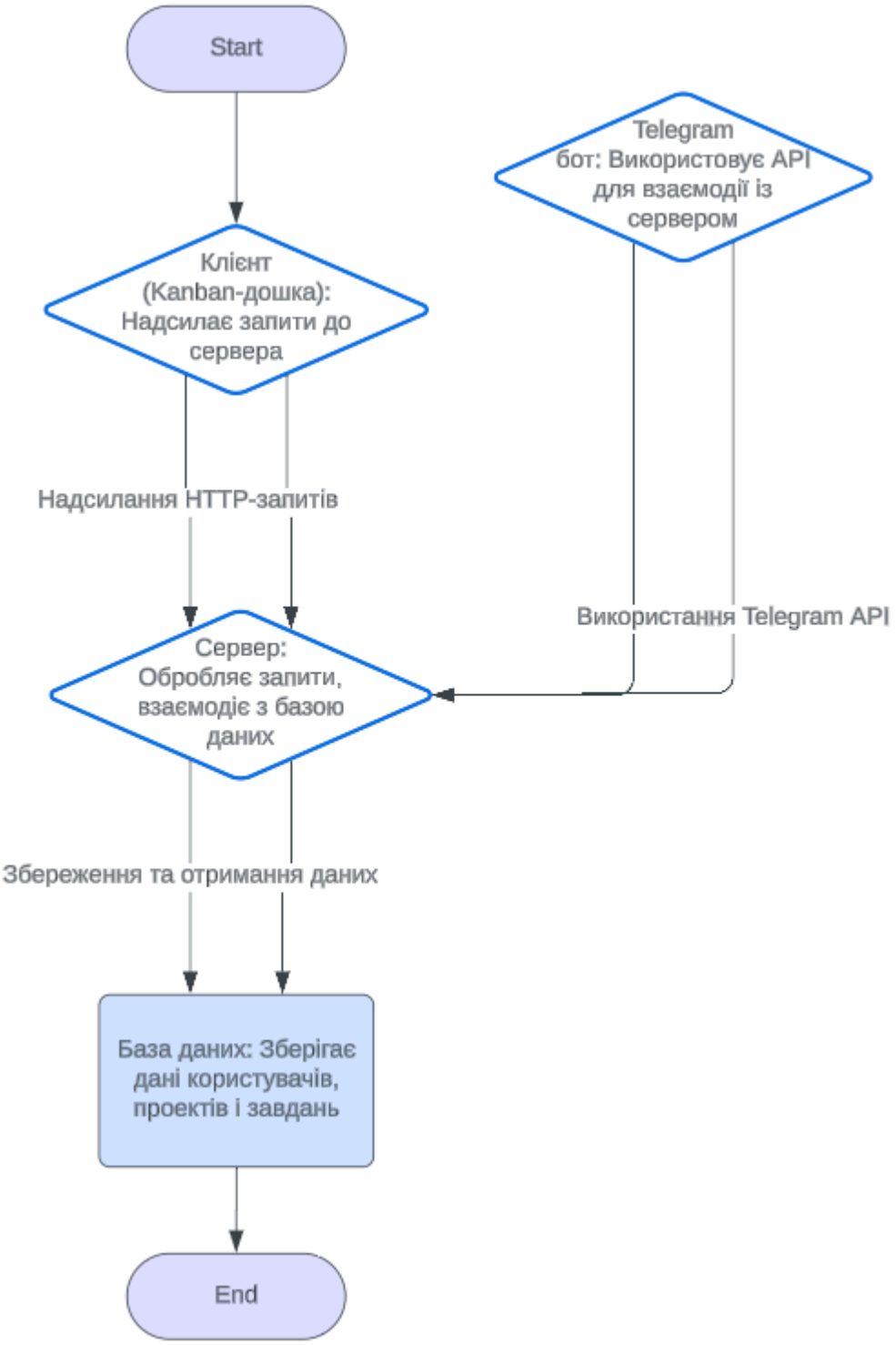

Рисунок Б.8 - Діаграма компонентів

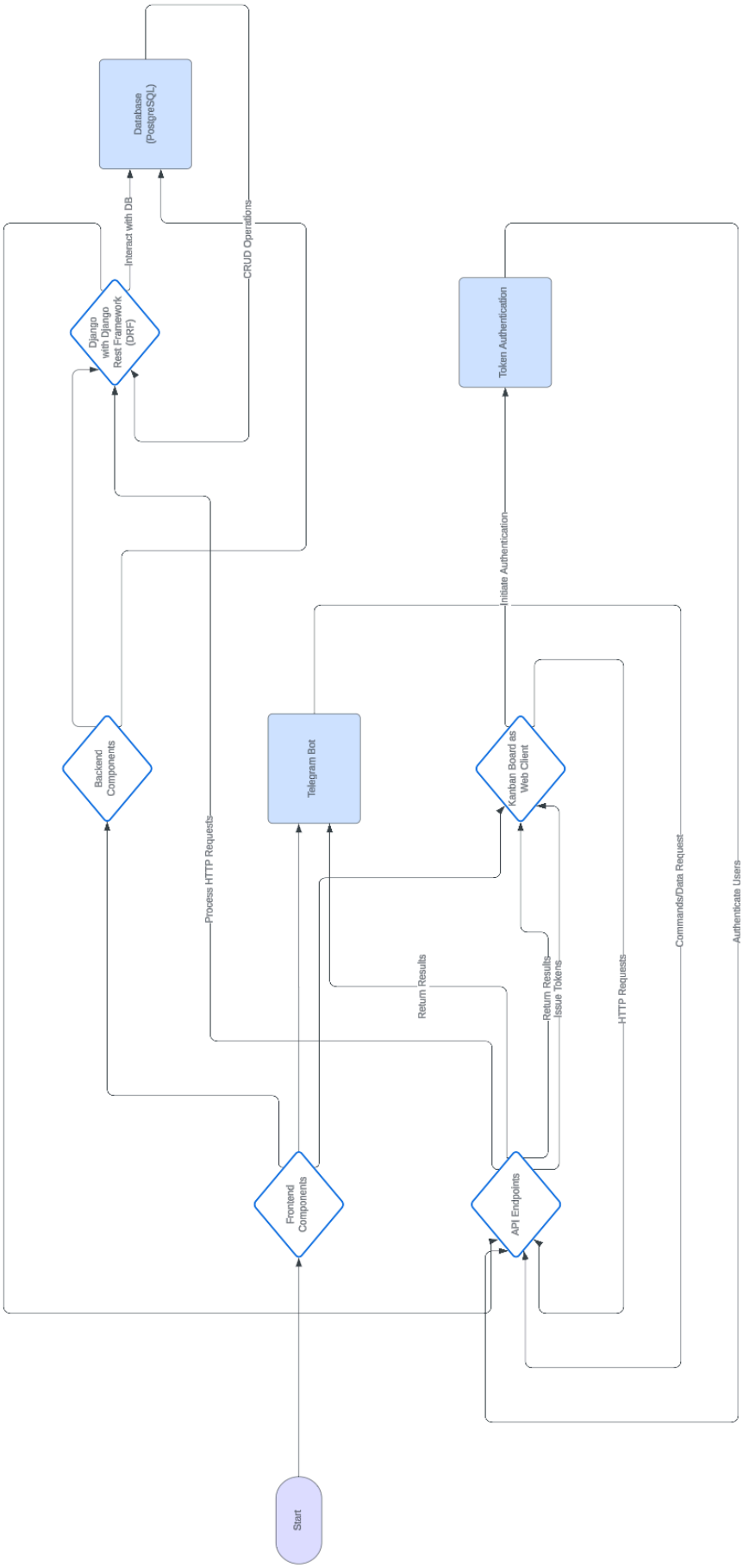

Рисунок Б.9 - Архітектурна діаграма

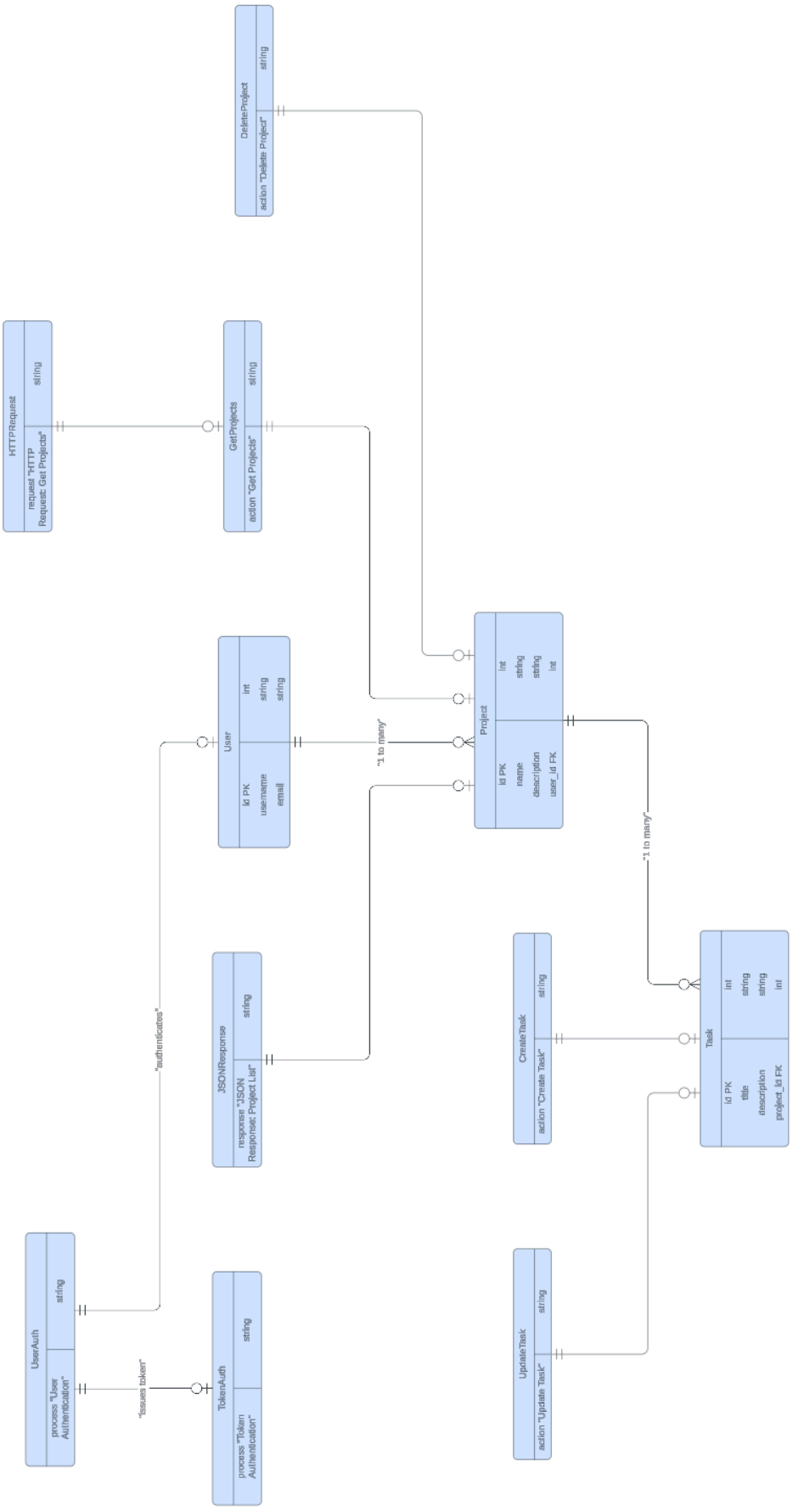

Рисунок Б.10 - Діаграма CRUD-операцій

Додаток В
(Вибірковий)
Лістинг програми

Файл bot.py

```
import logging

import os

import sys

import tempfile

import urllib.parse

import asyncio

from datetime import datetime

import errno

from channels.db import database_sync_to_async

from filelock import FileLock

from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup

from telegram.ext import ApplicationBuilder, CommandHandler, CallbackQueryHandler,
ContextTypes

from telegram import CallbackQuery

from asgiref.sync import sync_to_async

from django.contrib.auth import authenticate

import django

# Setup logging

logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
level=logging.INFO)

logger = logging.getLogger(__name__)

# Define constants

PID_FILE = os.path.join(tempfile.gettempdir(), 'bot.pid')

LOCK_FILE = f"{PID_FILE}.lock"
```

```
TOKEN = '7437196256:AAHY-Ir0oNudmjkDSC6FvqYMDTS0IVoSHII'  # Replace with your actual bot token

sessions = {}

e = None

# Ensure the Django settings module is correctly defined before setup

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'kanban.settings')

django.setup()

try:

    from board.models import Task

except ModuleNotFoundError:

    logger.error(

        "The 'board' module was not found. Ensure this app is present in your project and listed in 'INSTALLED_APPS' in 'settings.py'.")

    sys.exit(1)

async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:

    await update.message.reply_text('Welcome to the Task Manager Bot! Use /login <username> <password> to log in.')

async def login(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:

    if len(context.args) != 2:

        await update.message.reply_text('Usage: /login <username> <password>')

        return

    username, password = context.args

    user = await sync_to_async(authenticate)(username=username, password=password)

    if user is not None:

        telegram_user_id = update.message.from_user.id

        sessions[telegram_user_id] = user

        await update.message.reply_text('Login successful!')

    else:

        await update.message.reply_text('Invalid credentials. Please try again.')
```

```
def sanitize_callback_data(data):
    """Sanitize callback data."""
    return urllib.parse.quote(data, safe='')
async def add_task(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    telegram_user_id = update.message.from_user.id
    if telegram_user_id not in sessions:
        await update.message.reply_text('Please log in first using /login <username> <password>')
        return
    user = sessions[telegram_user_id]
    task_name = ' '.join(context.args)
    if not task_name:
        await update.message.reply_text('Please provide a task name. Usage: /add <task_name>')
        return
    task = await sync_to_async(Task.objects.create)(owner=user, name=task_name)
    await update.message.reply_text(f'Task "{task_name}" added with UUID {task.uuid}.')
async def list_tasks(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """List tasks and prompt for column selection."""
    columns = ['Заплановано', 'В роботі', 'На перевірці', 'Виконано']
    keyboard = [InlineKeyboardButton(col, callback_data=f"list_{col}") for col in columns]
    reply_markup = InlineKeyboardMarkup([keyboard])

    await update.message.reply_text("Оберіть дошку:", reply_markup=reply_markup)
async def handle_back_button(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Handle the BACK button and return to the column selection menu."""
    query = update.callback_query
    if query:
        await query.answer()
    columns = ['Заплановано', 'В роботі', 'На перевірці', 'Виконано']
```

```
    keyboard = [InlineKeyboardButton(col, callback_data=f"list_{col}") for col in columns]
    reply_markup = InlineKeyboardMarkup([keyboard])
    await query.edit_message_text("Оберіть дошку:", reply_markup=reply_markup)
async def handle_column_selection(update: Update, context: ContextTypes.DEFAULT_TYPE, column: str) -> None:
    query = update.callback_query
    if query:
        await query.answer()
    telegram_user_id = query.from_user.id
    if telegram_user_id not in sessions:
        await query.edit_message_text('Please log in first using /login <username> <password>')
        return
    user = sessions[telegram_user_id]
    tasks = await sync_to_async(list)(Task.objects.filter(owner=user, boardName=column))

    if not tasks:
        await query.edit_message_text(
            "Немає завдань.",
            reply_markup=InlineKeyboardMarkup([[InlineKeyboardButton("Повернутись до списку", callback_data='back')]])
        )
        return
    for task in tasks:
        task_text = f"{task.name} - {task.boardName}"
        buttons = [
            InlineKeyboardButton("Move", callback_data=f"move_{task.uuid}"),
            InlineKeyboardButton("Delete", callback_data=f"delete_{task.uuid}"),
            InlineKeyboardButton("Back", callback_data=f"back")
        ]
```

```
        reply_markup = InlineKeyboardMarkup([buttons])
        await query.message.reply_text(task_text, reply_markup=reply_markup)
def parse_callback(callback_data):
    try:
        action, column, task_uuid = callback_data.split('_')
        column = column.replace('_', ' ')  # Відновлення пробілів
        return action, column, task_uuid
    except ValueError as e:
        logger.error(f"Помилка розбору callback: {e}")
        return None, None, None
@database_sync_to_async
def get_task_by_uuid(task_uuid):
    try:
        task = Task.objects.get(uuid=task_uuid)
        return task
    except Task.DoesNotExist:
        return None
async def move_task(query: CallbackQuery, task_uuid: str, new_boardName: str):
    logger.info(f"Початок переміщення завдання {task_uuid} у колонку {new_boardName}")
    logger.debug(
        f"query type: {type(query)}, task_uuid type: {type(task_uuid)}, new_boardName type:
{type(new_boardName)}")
    try:
        # Перевірка типу query
        if isinstance(query, CallbackQuery):
            user_id = query.from_user.id
            message = query.message
        else:
```

```
        logger.error(f"Неправильний тип query: Очікується CallbackQuery, отримано {type(query)}")
        raise ValueError(f"Invalid query type: {type(query)}")


    task = await sync_to_async(Task.objects.get)(uuid=task_uuid)
    if task:
        logger.info(f"Завдання знайдено: {task}")
        # Поточна колонка перед змінами
        old_boardName = task.boardName
        logger.info(f"Поточна колонка: {old_boardName}")
        # Зміна колонки
        task.boardName = new_boardName
        await sync_to_async(task.save)()
        logger.info(f"Переміщуємо завдання {task.uuid} з колонки {task.boardName} у колонку {new_boardName}")
        # Перевірка після збереження
        task_updated = await sync_to_async(Task.objects.get)(uuid=task_uuid)
        logger.info(f"Колонка завдання після збереження: {task_updated.boardName}")
        if task_updated.boardName == new_boardName:
            logger.info(f"Завдання {task_uuid} успішно переміщене в колонку {new_boardName}")
            await query.edit_message_text(f"Завдання успішно переміщене в колонку {new_boardName}")
            # Оновлення списку завдань
            context = {}  # Змінити за необхідності
            tasks = await list_tasks(user_id, context)
            formatted_tasks = format_tasks(tasks)
            logger.debug(f"Зформатований список завдань: {formatted_tasks}")
            # Оновлення повідомлення в чаті
            await message.edit_reply_markup(reply_markup=formatted_tasks)
```

```
            await query.answer('Завдання переміщене.', show_alert=True)
        else:
            logger.error(f"Не вдалося змінити колонку для завдання {task_uuid}")
            await query.answer('Не вдалося перемістити завдання. Спробуйте ще раз.',
show_alert=True)
    else:
        logger.error(f"Завдання з UUID {task_uuid} не знайдено")
        await query.answer('Завдання не знайдено.', show_alert=True)
except Task.DoesNotExist:
    logger.error(f"Завдання з UUID {task_uuid} не знайдено")
    await query.answer('Завдання не знайдено.', show_alert=True)
except ValueError as ve:
    logger.error(str(ve))
    await query.answer(f'Помилка: {str(ve)}', show_alert=True)
@database_sync_to_async
def delete_task_by_uuid(task_uuid):
    try:
        task = Task.objects.get(uuid=task_uuid)
        task.delete()
        return True
    except Task.DoesNotExist:
        return False
async def delete_task(query, task_uuid):
    """Handle task deletion."""
    success = await delete_task_by_uuid(task_uuid)
    if success:
        logger.info(f"Завдання {task_uuid} видалено.")
        await query.answer('Завдання успішно видалено.', show_alert=True)
    else:
```

```
        logger.error(f"Завдання {task_uuid} не знайдено.")
        await query.answer('Завдання не знайдено.', show_alert=True)
async def show_move_options(query, task_uuid):
    columns = ['Заплановано', 'В роботі', 'На перевірці', 'Виконано']
    buttons = []
    for column in columns:
        callback_data = f'mv_{column}_{task_uuid}'
        logger.info(f"Callback data для кнопки: {callback_data}")
        buttons.append(InlineKeyboardButton(column, callback_data=callback_data))
        logger.info(f"Created button with callback_data: {callback_data}")

    reply_markup = InlineKeyboardMarkup([buttons])
    logger.info(f"Generated InlineKeyboardMarkup: {reply_markup}")
    try:
        await query.edit_message_reply_markup(reply_markup=reply_markup)
    except Exception as e:
        logger.error(f"Не вдалося оновити повідомлення через: {e}")
        await query.answer('Не вдалося оновити повідомлення.', show_alert=True)
async def button(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Handle button press."""
    global column
    query = update.callback_query
    await query.answer()
    callback_data = query.data
    logger.info(f"Отримано callback запит з даними: {callback_data}")

    action = None
    task_uuid = None
```

```
    new_column = None
    try:
        if callback_data.startswith('delete_task_'):
            action = 'delete_task'
            task_uuid = callback_data[len('delete_task_'):]
        elif callback_data.startswith('mv_'):
            action = 'move_to'
            parts = callback_data.split('_', 2)
            if len(parts) == 3:
                new_column = parts[1]
                task_uuid = parts[2]
        elif callback_data.startswith('list_'):
            action = 'list_tasks'
            column = callback_data.split('_', 1)[1]
        elif callback_data.startswith('move_'):
            action = 'show_move_options'
            task_uuid = callback_data[len('move_'):]
        elif callback_data == 'back':
            action = 'back'
        else:
            raise ValueError("Невідома дія")
        logger.info(
            f"Розбір дії: {action}, Нова колонка: {new_column}, UUID завдання: {task_uuid},
callback дані: {callback_data}")
    except ValueError as e:
        await query.answer('Невірний формат callback даних.', show_alert=True)
        logger.warning(f"Не вдалося розпарсити callback дані: {callback_data}, помилка: {e}")
        return
    if action == 'delete_task':
```

```
        await delete_task(query, task_uuid)
    elif action == 'move_to':
        if new_column:
            logger.info(f"Переміщення завдання {task_uuid} у колонку {new_column}")
            await move_task(query, task_uuid, new_column)
        else:
            await query.answer('Не вдалося визначити дошку.', show_alert=True)
    elif action == 'list_tasks':
        await handle_column_selection(update, context, column)
    elif action == 'show_move_options':
        logger.info(f"Showing move options for task: {task_uuid}")
        await show_move_options(query, task_uuid)
    elif action == 'back':
        await handle_back_button(update, context)
async def set_task_reminder(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    if len(context.args) != 2:
        await update.message.reply_text('Usage: /set_task_reminder <task_uuid> <YYYY-MM-DD
HH:MM:SS>')
        return
    task_uuid, reminder_time = context.args
    telegram_user_id = update.message.from_user.id
    if telegram_user_id not in sessions:
        await update.message.reply_text('Please log in first using /login <username> <password>')
        return
    try:
        reminder_datetime = datetime.strptime(reminder_time, '%Y-%m-%d %H:%M:%S')
    except ValueError:
        await update.message.reply_text('Invalid datetime format. Use YYYY-MM-DD HH:MM:SS.')
        return
```

```
    task = await sync_to_async(Task.objects.get)(uuid=task_uuid)
    if not task:
        await update.message.reply_text('Task not found.')
        return
    task.reminder = reminder_datetime
    await sync_to_async(task.save)()
    await update.message.reply_text(f'Reminder set for task "{task.name}" at {reminder_datetime}.')
    # Wait for the reminder
    await asyncio.sleep((reminder_datetime - datetime.now()).total_seconds())
    await update.message.reply_text(f'Reminder: {task.name}')
def task_to_string(task):
    return f"{task.name} - {task.boardName}"
def format_tasks(tasks):
    keyboard = []
    for task in tasks:
        task_button = [
            InlineKeyboardButton(task.name, callback_data=f"show_task_{task.uuid}")
        ]
        delete_button = [
            InlineKeyboardButton("Delete", callback_data=f"delete_task_{task.uuid}")
        ]
        move_button = [
            InlineKeyboardButton("Move", callback_data=f"move_task_{task.uuid}")
        ]
        keyboard.append(task_button + delete_button + move_button)
    return InlineKeyboardMarkup(keyboard)
async def error(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    logger.warning('Update "%s" caused error "%s"', update, context.error)
```

```
def setup():
    application = ApplicationBuilder().token(TOKEN).build()
    application.add_handler(CommandHandler("start", start))
    application.add_handler(CommandHandler("login", login))
    application.add_handler(CommandHandler("add", add_task))
    application.add_handler(CommandHandler("list", list_tasks))
    application.add_handler(CommandHandler("set_task_reminder", set_task_reminder))
    application.add_handler(CallbackQueryHandler(button))
    application.add_error_handler(error)
    application.run_polling()
def check_pidfile():
    if os.environ.get('RUN_MAIN') == 'true':
        logger.info("Detected RUN_MAIN environment, skipping PID check for auto-reloader.")
        return
    logger.info("Checking for existing PID file...")
    if os.path.isfile(PID_FILE):
        try:
            with open(PID_FILE, 'r') as pidfile:
                pid_content = pidfile.read().strip()
                logger.info(f"PID file content: {pid_content}")
                if pid_content.isdigit():
                    pid = int(pid_content)
                    logger.info(f"Found existing PID file with PID: {pid}")
                    try:
                        os.kill(pid, 0)
                        logger.warning("Another instance of the bot is already running. Exiting...")
                        sys.exit(1)
                    except OSError as e:
```

```
                if e.errno == errno.ESRCH:
                    logger.info("Previous bot instance is not running, removing stale PID file.")
                    os.remove(PID_FILE)
                else:
                    logger.error("An error occurred while checking the current process status.",
exc_info=e)
                    raise
            else:
                logger.error(f"PID file contains non-numeric content: {pid_content}")
                os.remove(PID_FILE)
        except Exception as general_e:
            logger.error("An error occurred while reading the PID file.", exc_info=general_e)
            os.remove(PID_FILE)
    logger.info("No existing PID file found or previous PID not running. Writing new PID file.")
    with open(PID_FILE, 'w') as pidfile:
        pidfile.write(str(os.getpid()))
def cleanup_pidfile():
    if os.path.isfile(PID_FILE):
        logger.info("Cleaning up PID file...")
        os.remove(PID_FILE)
    else:
        logger.warning("PID file not found during cleanup.")
if __name__ == '__main__':
    try:
        with FileLock(LOCK_FILE, timeout=10):
            check_pidfile()
            setup()
    except Exception as e:
        logger.error(f"An error occurred: {e}", exc_info=e)
```

```
    finally:
        cleanup_pidfile()
```

Файл models.py

```
import uuid
from django.contrib.auth.models import User
from django.db import models
class Task(models.Model):
    class boardNames(models.TextChoices):
        ToDo = 'Заплановано'
        InProgress = 'В роботі'
        Review = 'На перевірці'
        Done = 'Виконано'
    owner = models.ForeignKey('auth.User', on_delete=models.CASCADE,
                              related_name='tasks')
    uuid = models.UUIDField(default=uuid.uuid4, unique=True, primary_key=True,
                            editable=False)
    name = models.CharField(max_length=500)
    boardName = models.CharField(max_length=12, choices=boardNames.choices,
                                 default=boardNames.ToDo)
    date = models.DateTimeField(auto_now_add=True)
    reminder = models.DateTimeField(null=True, blank=True)
    class Meta:
        verbose_name = 'Завдання'
        verbose_name_plural = 'Завдання'
    def __str__(self):
        return str(self.name)
class TelegramUser(models.Model):
```

```
    user = models.OneToOneField(User, on_delete=models.CASCADE,
related_name="telegram_profile")

    telegram_id = models.CharField(max_length=64, unique=True)

    def __str__(self):

        return f"{self.user.username} ({self.telegram_id})"
```

Файл index.html

```
{% extends "base.html" %}

{% block title %}

Канбан-дошка на Django, DRF & Alpine.js

{% endblock %}

{% block content %}

<body class="antialiased sans-serif bg-gray-300">

  <div x-data="board()" x-init="getData()" x-cloak class="flex flex-col min-h-screen border-t-8"
:class="`border-${colorSelected.value}-700`">

    <div class="flex-1">

      <div class="bg-cover bg-center bg-no-repeat" :class="`bg-${colorSelected.value}-900`"
:style="`background-image: url(${bannerImage})`">

        <div class="container mx-auto px-4 pt-4 md:pt-10 pb-40"></div>

      </div>

      <div class="container mx-auto px-4 py-4 -mt-40">

{% include 'settings.html' %}

        <div x-show.immediate="showSettingsPage == false">

          <div x-show.transition="showSettingsPage == false">

            <div class="flex justify-between items-center mb-2">

              <div>

                <h1 class="text-xl md:text-2xl text-gray-300 font-semibold">

                  {% now "H" as current_time %}

                  {% if current_time|add:"0" < 12 %}
```

```
            Доброго ранку, {{ request.user.username }}
            {% elif current_time|add:"0" > 17 %}
            Доброго вечора, {{ request.user.username }}
            {% else %}
            Доброго дня, {{ request.user.username }}
            {% endif %}
          </h1>
          <div class="text-sm" :class="`text-${colorSelected.value}-400`">{% now
'SHORT_DATE_FORMAT' %}</div>
        </div>
        <div>
          <a @click.prevent="showSettingsPage = !showSettingsPage" href="#" class="rounded-lg
px-3 py-2 font-medium inline-flex" :class="`text-${colorSelected.value}-500
bg-${colorSelected.value}-800 hover:bg-${colorSelected.value}-700`">
            <i class="uil uil-setting text-left mr-1"></i> Налаштувати</a>
          <a href="/logout" class="rounded-lg px-3 py-2 font-medium inline-flex items-center"
:class="`text-${colorSelected.value}-500 bg-${colorSelected.value}-800
hover:bg-${colorSelected.value}-700`">
            <i class="uil uil-sign-out-alt mr-1"></i>Вийти</a>
        </div>
      </div>
<a class="rounded-lg px-0 py-2 font-medium inline-flex items-center" class="text-sm"
:class="`text-${colorSelected.value}-400`" href="https://t.me/KanbanHelp_bot"
target="_blank">Телеграм бот <img
src="https://cdn-icons-png.flaticon.com/512/2111/2111646.png" width="24" height="24"
alt="Telegram Bot"></a>
      <div class="py-4 md:py-8">
      <form id="task-form">
        {% csrf_token %}
      </form>
        <div class="flex -mx-4 block overflow-x-auto pb-2">
          <template x-for="board in boards" :key="board">
```

```
<div class="w-1/2 md:w-1/4 px-4 flex-shrink-0">

 <div class="bg-gray-100 pb-4 rounded-lg shadow overflow-y-auto overflow-x-hidden border-t-8" style="min-height: 100px" :class="{

  'border-red-500': board === boards[0],

  'border-yellow-500': board === boards[1],

  'border-blue-500': board === boards[2],

  'border-green-500': board === boards[3],

 }">

 <div class="flex justify-between items-center px-4 py-2 bg-gray-100 sticky top-0">

  <h2 x-text="board" class="font-medium text-gray-800"></h2>

  <a @click.prevent="showModal(board)" href="#" class="inline-flex items-center text-sm font-medium" :class="`text-${colorSelected.value}-500 hover:text-${colorSelected.value}-600`">

  <i class="uil uil-book-medical mr-1"></i>Додати

  </a>

 </div>

 <div class="px-4">

  <div @dragover="onDragOver(event)" @drop="onDrop(event, board)" @dragenter="onDragEnter(event)" @dragleave="onDragLeave(event)" class="pt-2 pb-20 rounded-lg">

   <template x-for="(t, taskIndex) in tasks.filter(t => t.boardName === board)" :key="taskIndex">

    <div :id="t.uuid">

     <div x-show="t.edit == false">

      <div x-show="t.edit == false" class="bg-white rounded-lg shadow mb-3 p-2" draggable="true" @dragstart="onDragStart(event, t.uuid)" @dblclick="t.edit = true; setTimeout(() => $refs[t.uuid].focus())">

       <div x-text="t.name" class="text-gray-800"></div>

       <div x-text="t.date" class="text-gray-500 text-xs mt-2"></div>

       <div class="text-right">

       <button type="button" class="font-semibold text-red-500 focus:outline-none hover:text-red-700 shadow-sm" @click="removeTask(t.uuid)">
```

```
                    <i class="uil uil-times-circle"></i></button></div>
                    </div>
                  </div>
                  <div x-show="t.edit == true" class="bg-white rounded-lg p-4 shadow mb-4">
                    <div class="mb-4">
                      <label class="text-gray-800 block mb-1 font-bold text-sm">Завдання</label>
                      <input :x-ref="t.uuid" class="bg-gray-200 appearance-none border-2
border-gray-200 rounded-lg w-full py-2 px-2 text-gray-700 leading-tight focus:outline-none
focus:bg-white focus:border-blue-500" type="text" x-model="t.name"
@keydown.enter="updateTask(t)">
                    </div>
                    <div class="text-right">
                      <button type="button" class="bg-white hover:bg-gray-100 focus:outline-none
text-gray-700 font-semibold py-1 px-2 text-xs border border-gray-300 rounded-lg shadow-sm mr-2"
@click="t.edit = false">
                        Відмінити
                      </button>
                      <button type="button" class="text-white font-semibold focus:outline-none py-1
px-2 text-sm border border-transparent rounded-lg shadow-xs" @click="updateTask(t.uuid, t.name)"
:class="`bg-${colorSelected.value}-700 hover:bg-${colorSelected.value}-800`">
                        Змінити
                      </button>
                    </div>
                  </div>
                </div>
              </template>
            </div>
          </div>
        </div>
      </template>
```

```
          </div>
        </div>
      </div>
    </div>
  </div>
  <div class="container mx-auto px-4 py-10">
    <p class="text-gray-600 text-center"><strong><i class="uil uil-edit mr-1"></i>Kanban</strong> розроблено на <a href="https://www.django-rest-framework.org/" :class="`text-${colorSelected.value}-500 hover:text-${colorSelected.value}-600`">Django REST Framework</a> та <a href="https://alpinejs.dev/" :class="`text-${colorSelected.value}-500 hover:text-${colorSelected.value}-600`">AlpineJS</a>.</p>
  </div>
  <div class="fixed inset-0 flex h-screen w-full items-end md:items-center justify-center z-10" x-show.transition.opacity="openModal">
    <div class="absolute inset-0 bg-black opacity-50"></div>
    <div class="md:p-4 md:max-w-lg mx-auto w-full flex-1 relative overflow-hidden">
      <div class="md:shadow absolute right-0 top-0 w-10 h-10 rounded-full bg-white text-gray-500 hover:text-gray-800 inline-flex items-center justify-center cursor-pointer" x-on:click="openModal = !openModal">
        <i class="uil uil-times"></i>
      </div>
      <div class="w-full rounded-t-lg md:rounded-lg bg-white p-8">
        <h2 class="font-bold text-2xl mb-6 text-gray-800">Нове завдання для <span class="leading-normal border-b-2" :class="`text-${colorSelected.value}-600 border-${colorSelected.value}-200`" x-text="task.boardName"></span></h2>
        <div class="mb-4">
          <label class="text-gray-800 block mb-1 font-bold text-sm">Опис завдання</label>
          <input class="bg-gray-200 appearance-none border-2 border-gray-200 rounded-lg w-full py-2 px-4 text-gray-700 leading-tight focus:outline-none focus:bg-white focus:border-blue-500" type="text" x-model="task.name" x-ref="taskName" autofocus @keydown.enter="addTask()">
        </div>
```

```
        <div class="mt-8 text-right">
          <button type="button" class="bg-white hover:bg-gray-100 text-gray-700 font-semibold focus:outline-none py-2 px-4 border border-gray-300 rounded-lg shadow-sm mr-2" @click="openModal = !openModal">
            Відмінити
          </button>
          <button type="button" class="text-white font-semibold py-2 px-4 border border-transparent focus:outline-none rounded-lg shadow-sm" @click="addTask(task.name, task.boardName)" :class="`bg-${colorSelected.value}-700 hover:bg-${colorSelected.value}-800`">
            Зберегти
          </button>
        </div>
      </div>
    </div>
  </div>
  <!-- конец формы -->
 </div>
<script>
const csrftoken = document.querySelector('#task-form > input').value;
const api_client = axios.create({
  baseURL: '/api',
  headers: { 'X-CSRFToken': csrftoken },
});

const addTask = async (name, board) => {
  try {
    const res = await api_client.post('/tasks/', {name, boardName: board});
    location.reload();
  } catch (e) {
    console.error(e);
```

```
    }


};
const removeTask = async taskUuid => {
    try {
        const res = await api_client.delete(`/task/${taskUuid}`);
        location.reload();
    } catch (e) {
        console.error(e);
    }
};
const updateTask = async (taskUuid, taskName) => {
    try {
        const res = await api_client.patch(`/task/${taskUuid}`,
            {uuid: taskUuid, name: taskName}
        );
        location.reload();
    } catch (e) {
        console.error(e);
    }
};
function board() {
     return {
        showSettingsPage: false,
        openModal: false,
        bannerImage: '',
        colors: [{
            label: '#3182ce',
```

```
  value: 'blue'
},
{
  label: '#38a169',
  value: 'green'
},
{
  label: '#967bb6',
  value: 'purple'
},
{
  label: '#e53e3e',
  value: 'red'
},
{
  label: '#ffa500',
  value: 'orange'
},
{
  label: '#0B0B45',
  value: 'indigo'
},
{
  label: '#008080',
  value: 'teal'
},
{
  label: '#718096',
```

```
    value: 'gray'
  },
  {
    label: '#ffff00',
    value: 'yellow'
  }
],
colorSelected: {
  label: '#3182ce',
  value: 'blue'
},
boards: [
  'Заплановано',
  'В роботі',
  'На перевірці',
  'Виконано'
],
task: {
  name: '',
  boardName: '',
  date: new Date()
},
tasks: [],
showModal(board) {
  this.task.boardName = board;
  this.openModal = true;
  setTimeout(() => this.$refs.taskName.focus(), 200);
},
```

```
getData() {
  const themeFromLocalStorage = JSON.parse(localStorage.getItem('KB-theme'));
  this.bannerImage = localStorage.getItem('KB-bannerImage') || '';
  this.colorSelected = themeFromLocalStorage || {
    label: '#3182ce',
    value: 'blue'
  };
   const tasksFromDjango = {{ tasks | safe }};
   this.tasks = tasksFromDjango.map(t => {
      return {
        uuid: t.uuid,
        name: t.name,
        boardName: t.boardName,
        date: t.date,
        edit: false
      }
    });
},
saveSettings() {
  const theme = JSON.stringify(this.colorSelected);
  localStorage.setItem('KB-theme', theme);
  localStorage.setItem('KB-bannerImage', this.bannerImage);
  this.showSettingsPage = false;
},
onDragStart(event, uuid) {
  event.dataTransfer.setData('text/plain', uuid);
  event.target.classList.add('opacity-5');
},
```

```
        onDragOver(event) {
          event.preventDefault();
          return false;       },
        onDragEnter(event) {
          event.target.classList.add('bg-gray-200');       },
        onDragLeave(event) {
          event.target.classList.remove('bg-gray-200');       },
        onDrop(event, boardName) {
          event.stopPropagation();
          event.preventDefault();
          event.target.classList.remove('bg-gray-200');
          const id = event.dataTransfer.getData('text');
          const draggableElement = document.getElementById(id);
          const dropzone = event.currentTarget;
          dropzone.appendChild(draggableElement);
          api_client.patch(`/task/${id}`, { boardName: boardName});
          event.dataTransfer.clearData();       },
        saveDataToLocalStorage(data, keyName) {
          var a = [];
          a = JSON.parse(localStorage.getItem(keyName)) || [];
          a.push(data);
          localStorage.setItem(keyName, JSON.stringify(a));
        },
       }
     }
</script>
</body>
 {% endblock %}
```

128

Додаток Г (обов'язковий)

Протокол перевірки магістерської кваліфікаційної роботи на наявність текстових запозичень

Назва роботи: «Розробка автоматизованої системи управління проектами для оптимізації завдань на підприємстві»
Тип роботи: магістерська кваліфікаційна робота
Підрозділ: кафедра АІІТ

Показники звіту подібності Turnitin

Оригінальність 93,0 % Схожість 7,0 %

Аналіз звіту подібності (відмітити потрібне)

- Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і самостійності її автора. Роботу направити на розгляд експертної комісії кафедри.
- Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Роман МАСЛІЙ
(підпис)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Роман ТИТОМИР
(підпис)

Керівник роботи _____ Марія БАРАБАН
(підпис)