НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

“До захисту допущено”
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

“___”_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
“Цифрові технології в енергетиці”
зі спеціальності *122 “Комп’ютерні науки”*

на тему: **Система організації щоденних справ з дошкою Kanban**

Виконав:
студент IV курсу, групи ТМ-02

ПІДГЕРСЬКИЙ Іван Олександрович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник: *доцент каф. цифрових технологій в енергетиці, доцент, к.т.н., Володимир ТИХОХОД* _____
(посада, вчене звання ,науковий ступінь, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент: _____
(посада, вчене звання, науковий ступінь, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль: *асистент Володимир РУДИК* _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА_

«__»_____2024 р.

ЗАВДАННЯ
на дипломну роботу студенту

ПІДГЕРСЬКОМУ Івану Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи **Система організації щоденних справ з дошкою Kanban**

керівник роботи ***Тихоход Володимир Олександрович, к.т.н., доцент***
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 202__ р. №_____

2. Термін подання роботи студентом **07.06.2024 року**

3. Вихідні дані до роботи мови програмування Python та JavaScript, фреймворки Flask та Vue.js 3, СКБД MongoDB, веб-сервер NGINX, система контролю версій Git, платформа Docker

4. Перелік питань, які потрібно розробити
1) провести аналіз серед наявних аналогів веб-систем для організації щоденних справ з дошкою Kanban
2) аргументувати вибрані інструменти, технології та алгоритми для реалізації програмного забезпечення

3) побудувати архітектури програмного забезпечення, баз даних та сховища файлів

4) створити прикладний програмний веб-інтерфейс

5) представити процес застосування веб-інтерфейсу

5. Орієнтований перелік ілюстративного матеріалу діаграми, що демонструють архітектуру системи та бази даних, взаємодію користувачів із системою; приклади роботи з програмним продуктом.

6. Дата видачі завдання «15» вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі	17-20.04.24	
3.	Розробка архітектури та загальної структури системи	21-25.04.24	
4.	Розробка частин окремих підсистем	25.04-02.05.24	
5.	Програмна реалізація системи	03-07.05.24	
6.	Оформлення пояснювальної записки	08-12.05.24	
7.	Захист програмного продукту	16.05.24	
8.	Передзахист	06.06.24	
9.	Подання готової роботи на кафедру	07.06.2024	
10.	Захист	17-21.06.2024	

Студент _____ **Іван ПІДГЕРСЬКИЙ**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ **Володимир ТИХОХОД**
(підпис) (ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 70 сторінках, містить 22 ілюстрації, 8 таблиць, 2 додатки, 20 джерел в переліку посилань.

Мета роботи – створення системи, що допоможе людям ефективніше організовувати свої щоденні завдання та краще розподіляти свій час з використанням дошок Kanban.

Методи та засоби: фреймворки Python Flask та Vue.js, веб-сервіс для хостингу репозиторіїв Github, текстовий редактор Sublime Text, платформа для контейнеризації Docker з утилітою Docker Compose, хмарний хостинг Contabo.

Результат – зручний та інтуїтивний веб-застосунок з приємним дизайном, що дозволяє користувачам ефективніше керувати своїми завданнями та проектами.

Ключові слова: ВЕБ-ЗАСТОСУНОК, ПРОДУКТИВНІСТЬ, UI/UX ДИЗАЙН, KANBAN ДОШКА, API, КОНТЕЙНЕРИЗАЦІЯ.

ABSTRACT

The thesis is made on 70 pages, contains 22 illustrations, 8 tables, 2 appendices, 20 sources in the list of references.

The purpose of the work is to create a system that will help people organize their daily tasks more efficiently and better allocate their time using Kanban boards.

Methods and tools: Python Flask and Vue.js frameworks, web service for hosting repositories Github, text editor Sublime Text, Docker containerization platform with Docker Compose utility, Contabo cloud hosting.

The result is a user-friendly and intuitive web application with a pleasing design that allows users to manage their tasks and projects more efficiently.

Keywords: WEB APPLICATION, PERFORMANCE, UI/UX DESIGN, KANBAN BOARD, API, CONTAINERIZATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Scrum – гнучка методологія розробки програмного забезпечення, яка базується на ітераційному підході та регулярних нарадах команди.

HTTP (Hypertext Transfer Protocol) – протокол передачі гіпертексту, який використовується для передачі даних в Інтернеті.

REST (Representational State Transfer) – архітектурний стиль для створення веб-служб, який базується на протоколі HTTP.

JSON (JavaScript Object Notation) – текстовий формат обміну даними, який легко читається людиною та машиною.

SQL (Structured Query Language) – мова структурованих запитів, яка використовується для взаємодії з реляційними базами даних.

NoSQL (Not only SQL) – бази даних, які не використовують табличну реляційну модель даних, як у традиційних SQL базах даних.

DevOps (Development and Operations) – об'єднана практика розробки програмного забезпечення та адміністрування IT-інфраструктури.

SSL/TLS (Secure Sockets Layer/Transport Layer Security) – криптографічні протоколи, які забезпечують безпечну передачу даних через Інтернет.

CRUD (Create, Read, Update, Delete) – основні операції, які виконуються з даними в базах даних та веб-застосунках.

SHA-256 (Secure Hash Algorithm 256-bit) – криптографічна геш-функція, яка використовується для створення цифрових відбитків та перевірки цілісності даних.

UUID v4 (Universally Unique Identifier version 4) – стандартний формат для створення унікальних ідентифікаторів, який широко використовується в програмуванні.

ВСТУП

В сучасному світі ефективне управління часом та організація щоденних справ є ключовими факторами успіху як для окремих людей, так і для організацій. Зі зростанням обсягу інформації та кількості завдань, з якими ми стикаємося кожного дня, стає все більш важливим мати надійні інструменти для планування та відстеження прогресу. Одним з таких інструментів є дошка Kanban – візуальний метод управління проектами та завданнями, який набув широкої популярності завдяки своїй простоті та ефективності.

Kanban – це гнучкий метод управління проектами, який фокусується на постійному покращенні робочого процесу. На відміну від Scrum, де роботу розбивають на спринти з фіксованою тривалістю, Kanban передбачає безперервний потік завдань [1]. Він дозволяє наочно представити робочий процес, розділивши його на етапи та відобразивши завдання у вигляді карток на дошці. Це допомагає визначити пріоритети, виявити вузькі місця та оптимізувати робочий потік. Проте, незважаючи на переваги використання дошки Kanban, все ще існує потреба в зручних та ефективних програмних рішеннях, які б дозволили інтегрувати цей метод в повсякденне життя та адаптувати його під індивідуальні потреби користувачів.

В останні роки спостерігається зростання попиту на програмні рішення для управління завданнями та організації робочого процесу. Згідно з даними звіту “Productivity Management Software Market Size – Global Industry, Share, Analysis, Trends and Forecast 2022 – 2030” розмір глобального ринку програмного забезпечення для управління продуктивністю оцінювався в 47,4 мільярда доларів США в 2021 році, і, за прогнозами, до 2030 року він займатиме ринок у 157,7 мільярда доларів США, зростаючи на 14,4% у період з 2022 по 2030 рік. [2]. Це свідчить про значний потенціал розвитку та впровадження інноваційних рішень в цій сфері.

Крім того, пандемія COVID–19 прискорила процес цифрової трансформації та переходу до віддаленої роботи, що ще більше підкреслило необхідність ефективних інструментів для організації та управління завданнями. За даними опитування, проведеного компанією Gartner, 74% керівників планували перевести частину своїх

співробітників на постійну віддалену роботу після закінчення пандемії [3]. Це створює додаткові виклики для забезпечення продуктивності та ефективної комунікації в розподілених командах, і саме тут системи організації щоденних справ можуть стати незамінним інструментом.

Серед існуючих систем, які використовують методологію Kanban для організації робочих процесів, можна виділити такі відомі рішення, як Trello, Asana та Notion. Ці платформи пропонують зручні інструменти для візуалізації завдань у вигляді канбан-дошок, проте мають певні обмеження та недоліки. Зокрема, безкоштовні або базові версії цих продуктів часто накладають значні обмеження на функціональність та дії користувачів без передплати на платні тарифи. Крім того, Trello критикують за відсутність розширених можливостей керування складними проектами, Asana може бути складною для освоєння, а Notion – надмірно комплексною для простих завдань. Таким чином, існуючі рішення нерідко вимагають компромісів або не задовольняють усіх вимог користувачів, відкриваючи простір для нових систем організації робочих процесів на основі Kanban.

Отже, метою роботи є створити системи, що допоможе людям ефективніше організовувати свої щоденні завдання та краще розподіляти свій час з використанням дошок Kanban.

Для реалізації мети було поставлено наступні завдання:

1) проаналізувати існуючі застосунки з реалізацією Kanban-дошки;
2) спроектувати архітектуру застосунку та структуру БД;
3) розробити веб-застосунок для організації щоденних справ з дошкою Kanban;
4) провести комплексне тестування та публікацію застосунку в мережі.

Записка з переддипломної практики складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Робота містить 70 сторінок, 22 рисунки, 8 таблиць. Список використаних джерел налічує 20 найменувань.

1 ОРГАНІЗАЦІЯ ЩОДЕННИХ СПРАВ З ДОШКОЮ KANBAN

Метою розробки веб-застосунку “Система організації щоденних справ з дошкою Kanban” є надання користувачам зручного та ефективного інструменту для управління своїми завданнями та проектами.

Для досягнення поставленої мети було сформовано функціональні вимоги програмного продукту:

1) авторизація та аутентифікація користувачів (реєстрація, логін, логаут);
2) створення нових дошок з можливістю вказати назву, опис, перелік статусів;
3) запрошення інших користувачів до певної дошки (за поштою);
4) надання відкритого доступу до певної дошки за посиланням;
5) перегляд списку всіх дошок користувача (дошки, що прив'язані до певного дня – у вигляді календаря; інші дошки, створені користувачем; дошки, до яких запросили користувача);
6) відкриття дошки та перегляд всіх карток на ній у вигляді Kanban-дошки з розбиттям по статусах;
7) створення карток на дошці з можливістю вказати назву, додати теги картки, детальний опис, прикріпити файли будь-якого типу до картки (з обмеженням по розміру 10 МБ);
8) відкриття картки по натисканню на неї на дошці у модальному вікні з можливістю переглянути та редагувати назву, опис, перелік тегів, скачати або замінити прикріплені файли;
9) переміщення карток за допомогою drag-n-drop між стовпцями (статусами);
10) зміна порядку карток за допомогою drag-n-drop в межах одного статусу;
11) персоналізовані налаштування: редагування переліку статусів, які за замовчуванням додаються в новостворені дошки користувача; зміна імені, паролю, електронної пошти.

Нефункціональні вимоги:

1) зручний та інтуїтивно зрозумілий інтерфейс;
2) швидкодія та плавна робота;

3) безпека даних;

4) адаптивний дизайн та функціонал для різних типів пристроїв.

Потенційними користувачами цього програмного забезпечення є люди, які потребують організації своїх щоденних справ, команди, які працюють над спільними проектами, та будь-хто, хто шукає ефективний спосіб управління завданнями, серед яких:

– студенти (для планування навчальних завдань, проектів та особистих справ);

– фрілансери та віддалені працівники (для керування робочим потоком та відстеження статусу завдань);

– малі та середні підприємства (для управління проектами, завданнями та ресурсами);

– проектні менеджери та команди розробників (для організації робочого процесу та відстеження прогресу);

– домогосподарки та люди з активним способом життя (для структурування домашніх справ, покупок, зустрічей);

– викладачі та тренери (для планування навчальних курсів та відстеження прогресу студентів).

Реалізація системи організації щоденних справ з дошкою Kanban складається з декількох основних етапів:

1) аналіз вимог та проектування архітектури системи;

2) розробка серверного застосунку;

3) розробка клієнтського застосунку;

4) інтеграція клієнтської та серверної частин;

5) розгортання системи;

6) тестування та налагодження системи;

7) розробка документації та інструкцій для користувачів.

Вхідною інформацією для системи є дані користувача, завдання, зображення, описи завдань та налаштування. Вихідна інформація включає список створених раніше дошок Kanban користувача, дошки з візуалізацією завдань, деталі завдань та персоналізовані налаштування.

Було розроблено діаграму прецедентів для зображення переліку функціональних вимог (рисунок 1.1).

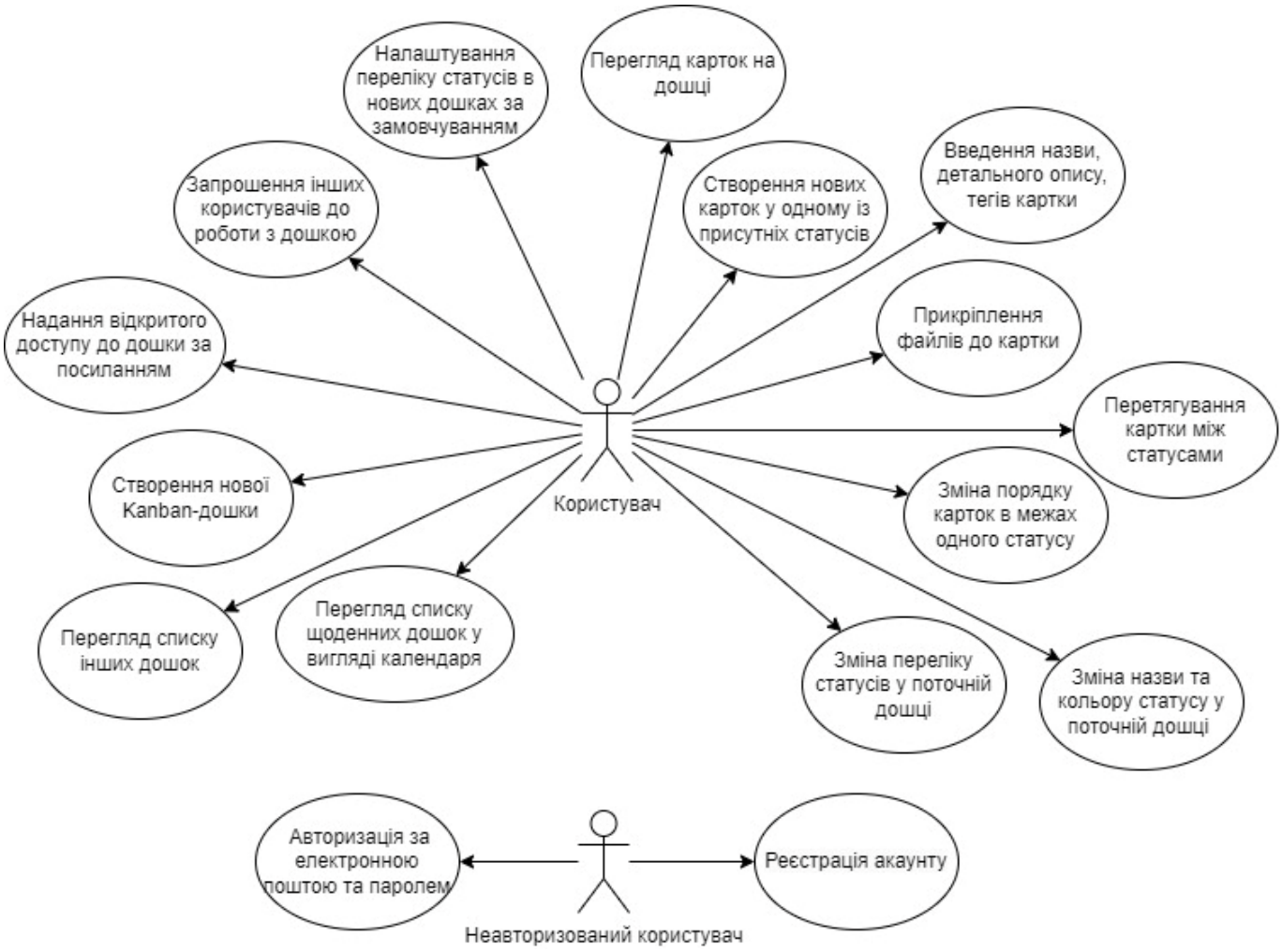

Рисунок 1.1 – Діаграма прецедентів

Для забезпечення безпеки та конфіденційності даних користувачів потрібно реалізувати систему аутентифікації та авторизації, а також забезпечити шифрування даних під час передачі та зберігання.

2 АНАЛІЗ ПРОБЛЕМИ РОЗРОБКИ СИСТЕМИ ОРГАНІЗАЦІЇ ЩОДЕННИХ СПРАВ З ДОШКОЮ KANBAN

Розробка сучасних веб-застосунків є складним та багатогранним процесом, що вимагає від фахівців різнобічних знань та навичок. Створення ефективної та надійної системи організації щоденних справ з дошкою Kanban не є винятком. Під час реалізації такого застосунку необхідно врахувати безліч аспектів, включаючи архітектуру та компоненти системи, інструменти та технології розробки, питання продуктивності, безпеки та зручності використання.

2.1 Архітектура веб-застосунку

Архітектура веб-застосунку визначає макет всіх програмних компонентів, таких як бази даних, проміжне програмне забезпечення та самі застосунки, а також їх взаємодію між собою. Вона встановлює, як дані передаються через HTTP протокол, гарантуючи, що вони можуть бути зрозумілими як для клієнтської, так і для серверної частини застосунку. Архітектура забезпечує наявність достовірних даних у всіх запитах користувачів, створення та керування записами, а також механізми доступу та автентифікації на основі дозволів.

Впровадження багаторівневої архітектури для сучасного веб-застосунку допомагає чітко визначити функціональність кожного компонента системи та полегшує внесення змін у відповідний рівень, не торкаючись решти частин програми. Такий підхід спрощує процеси написання, налагодження, керування та повторного використання коду [4]. Існує декілька загальних рівнів, які використовуються в різних видах сучасних архітектури веб-застосунків.

Перший – це, звісно ж, рівень подання – клієнтський компонент (зовнішній інтерфейс). Саме тут користувач бачить макет з дизайном UI/UX, інформаційні панелі та повідомлення і інтерактивно взаємодіє з застосунком. Тобто, користувачі

взаємодіють з сервером через браузер, який здійснює запити та відображає необхідну інформацію. Деякі з найпопулярніших інтерфейсних технологій представлені нижче.

HTML – стандартна мова розмітки, що дозволяє структурувати вміст веб-сторінки за допомогою різноманітних тегів.

CSS – це популярна мова стилів, що дозволяє розробникам описувати зовнішній вигляд сторінок веб-застосунків та їх складових, розроблених за допомогою мов розмітки. Використовуючи її, можна призначити стилі елементам і багаторазово перевикористовувати.

JavaScript – популярна мова програмування на стороні клієнта, що останнім часом використовується більш ніж на 90% веб-сайтів. Усі браузери містять компілятор для запуску коду JavaScript на будь-яких пристроях. Використання цієї мови на клієнті дозволяє розвантажити сервер, розподіляючи частину обчислювальних навантажень на сторону клієнта.

React – JS-фреймворк з відкритим кодом від Meta, що дозволяє створювати складні користувацькі інтерфейси за допомогою компонентної архітектури. Використовує концепцію Virtual DOM для оптимізації та прискорення роботи з DOM-деревом, що забезпечує високу продуктивність. Він має невелику криву навчання та широку екосистему сторонніх бібліотек і інструментів. Однак React зосереджений лише на представленні даних, тому для повноцінної розробки веб-застосунків потрібно використовувати додаткові бібліотеки, наприклад Redux для управління станом [5].

Vue – це прогресивний JS-фреймворк, який поєднує в собі легкість освоєння та гнучкість. Він пропонує широкий спектр можливостей, включаючи двостороннє зв'язування даних, віртуальний DOM, реактивну систему та модульність. Vue.js має чудову документацію та дружню спільноту розробників. Він легко інтегрується з існуючими проектами та бібліотеками, що робить його привабливим для поступової модернізації застарілих застосунків.

Angular – це повнофункціональний JS-фреймворк, розроблений командою Google. Він пропонує всі необхідні інструменти та функції для створення масштабних веб-застосунків, включаючи модульну структуру, систему маршрутизації, управління

станом, впровадження залежностей та інші. Angular має жорстку архітектуру та крутішу криву навчання в порівнянні з React та Vue.js, але водночас забезпечує високу продуктивність, кращу підтримку TypeScript та ефективну роботу з великими командами розробників [6].

Наступний рівень – серверний компонент (внутрішня частина). Це ключовий компонент архітектури веб-застосунку, що отримує запити від браузера, виконує основну бізнес-логіку та передає необхідні запитані дані. Для розробки цього рівня використовуються різні технології.

Node.js дозволяє виконувати код JavaScript на стороні сервера, що робить його ідеальним для створення швидких та масштабованих мережевих застосунків без потреби у вивченні додаткових мов програмування. Він використовує асинхронну модель введення/виведення, що забезпечує високу продуктивність [7].

Python – високорівнева мова програмування, є інтерпретованою динамічно типізованою мовою, яка відрізняється простотою синтаксису та читабельністю коду. Завдяки великій стандартній бібліотеці та численним фреймворкам (Django, Flask) Python широко використовується у веб-розробці, науковому програмуванні та автоматизації завдань.

Ruby відомий своїм елегантним і лаконічним синтаксисом, а також фреймворком Ruby on Rails, який прискорює розробку веб-застосунків, з дотриманням парадигми “Don't Repeat Yourself”.

Go (відомий також, як Golang) був розроблений в Google з метою створення компільованої статично типізованої мови, яка є ефективною, простою, та безпечною. Вона добре підходить для розробки систем з паралельним та розподіленим програмуванням.

Laravel – потужний фреймворк для PHP, який забезпечує зручний синтаксис, модульність, ефективну інтеграцію з базами даних та ще безліч функцій “з коробки”.

В межах серверного компонента зазвичай імплементують інтерфейс прикладного програмування (API). Це концепція, яка дозволяє отримувати доступ до певних даних та функцій програмного забезпечення. API є посередником, який дозволяє різним програмам та сервісам обмінюватися даними та функціональністю,

при цьому надаючи лише обмежений доступ, не розкриваючи всіх оригінальних даних.

Наприклад, коли користувач відкриває деяку сторінку, він викликає API для отримання даних, що мають бути відображені на ній. Застосунок зв’язується з відповідними серверами, щоб отримати потрібну інформацію і повернути ці дані. Після цього, користувач певними діями може спричиняти виклики в API уже не на отримання даних, а на доповнення, модифікацію або видалення певних ресурсів, доступних даному користувачу, на стороні сервера.

Однією з ключових переваг створення та використання API є відкриття доступу до функціональності та даних сторонніх сервісів без необхідності розробляти їх з нуля. Наприклад, веб-застосунки можуть інтегруватися з API соціальних мереж (Facebook, Twitter) для авторизації користувачів, публікації контенту чи отримання даних профілю. Подібним чином існують API для картографічних сервісів (Google Maps), платіжних систем, погодних сервісів тощо [8]. Скорочуючи зусилля розробки, API значно скорочують витрати. Це також покращує спільну роботу та зв'язок в екосистемі, підвищуючи якість обслуговування клієнтів.

RESTful API – найпопулярніший тип API у веб-розробці. Архітектурний стиль REST (Representational State Transfer) базується на принципах HTTP-протоколу та використовує легкі формати обміну даними, такі як JSON або XML. Він добре масштабується та забезпечує високу надійність і продуктивність, поширюючи способи для найкращої роботи з протоколом HTTP, що робить його найпопулярнішим API [9].

Останній з представлених рівнів – база даних, яка є ключовим компонентом веб-програми, що зберігає всю інформацію про користувачів та ресурси і дозволяє легко зчитувати їх та маніпулювати ними. Вона забезпечує доступ з врахуванням ролей та підтримує цілісність даних. При виборі бази даних потрібно уважно враховувати 4 аспекти: розмір, швидкість, масштабованість та структура.

Для структурованих даних гарним вибором є бази даних на основі SQL. Вони підходять для застосунків, в яких цілісність даних є ключовою вимогою, відповідно робота з такими даними буде максимально оптимізованою.

Для обробки неструктурованих даних прийнятним варіантом є NoSQL-бази даних. Вони підходять для застосунків, у яких характер вхідних даних може бути непередбачуваним або надто складним для структуризації.

У традиційній дворівневій архітектурі веб-застосунків присутні два компоненти: інтерфейс користувача та внутрішня система, яка зазвичай представлена сервером бази даних. Недоліком такого підходу є те, що вся бізнес-логіка вбудована або в інтерфейс користувача, або в сервер бази даних. Це призводить до зменшення продуктивності системи зі збільшенням кількості користувачів. Крім того, пряма взаємодія бази даних і клієнтського пристрою створює серйозні проблеми з безпекою.

Для вирішення цих недоліків використовується трирівнева архітектура веб-застосунків (рисунок 2.1).

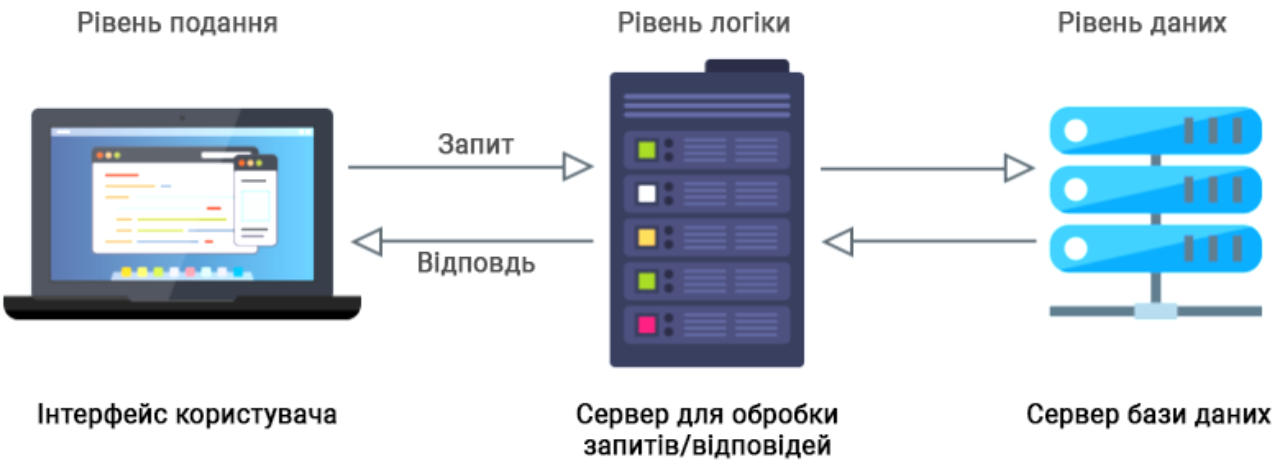

Рисунок 2.1 – Трирівнева архітектура веб-застосунку

У цій моделі існує проміжний рівень, який обробляє запити клієнтів, координуючи свої дії з базою даних та застосовуючи бізнес-логіку. Таким чином, проміжний рівень керує зв'язком між базою даних та клієнтом, надаючи клієнтам доступ до різних даних [10].

Така архітектура вважається безпечнішою, оскільки клієнти не мають прямого доступу до даних. Усі дані проходять через сервер, який вирішує, хто і як може отримати доступ до них, забезпечуючи цілісність даних. Загалом, трирівнева архітектура є більш безпечною, гнучкою та масштабованою порівняно з дворівневою.

2.2 Комплексність задачі створення веб-застосунку

Розробка сучасних веб-застосунків вимагає від фахівців різнобічних знань та навичок у різних сферах. Це пов'язано з тим, що веб-застосунок складається з кількох взаємопов'язаних компонентів, кожен з яких потребує спеціалізованих вмінь.

Перш за все, розробники повинні мати глибокі знання в програмуванні серверної та клієнтської частин застосунку. Для створення серверної частини необхідно володіти відповідними мовами програмування, фреймворками та бібліотеками, наприклад, Python з Flask або Node.js з Express.js. Для клієнтської частини потрібні навички роботи з JavaScript, HTML, CSS, а також фреймворками, такими як React, Angular або Vue.js. Ефективна взаємодія між цими двома компонентами є критичною для забезпечення належної функціональності та продуктивності застосунку.

Створення успішних веб-застосунків вимагає від розробників поєднання різнобічних компетенцій, що охоплюють програмування, веб-дизайн, кібербезпеку та DevOps. Лише завдяки комплексному підходу та безперервному навчанню можна забезпечити розробку якісних, безпечних та ефективних веб-застосунків, які відповідають сучасним вимогам.

2.2.1 Проектування зручного користувацького інтерфейсу

Привабливий та інтуїтивно зрозумілий інтерфейс є ключовим фактором успіху веб-застосунку, оскільки він визначає досвід взаємодії користувачів з системою. Розробники повинні мати знання з принципів UI/UX дизайну, верстки та адаптивної розмітки [11].

Дизайн UI та UX – це дві різні, але взаємопов'язані концепції в проектуванні цифрових продуктів.

Дизайн UI (user interface) відноситься до візуальних та інтерактивних елементів цифрового продукту, які користувачі бачать і з якими взаємодіють користувачі. Він включає в себе розробку макета, типографіки, кольорової гами, іконок, кнопок та інших візуальних елементів, які складають користувацький інтерфейс. Мета UI

дизайну є створення візуально привабливого, інтуїтивно зрозумілого та зручного інтерфейсу, який дозволяє користувачам легко орієнтуватися та взаємодіяти з продуктом.

З іншого боку, дизайн UX (user experience) відноситься до всього шляху користувача при взаємодії з цифровим продуктом, від першого знайомства з ним до підтримки після покупки. Він включає в себе проектування загального користувацького досвіду, включаючи потоки користувачів, інформаційну архітектуру, контент-стратегію та юзабіліті-тестування. Мета UX-дизайну – створити безперебійний, приємний і змістовний користувацький досвід, який відповідає потребам та очікуванням користувача.

По суті, в той час як UI дизайн фокусується на візуальних елементах продукту, UX дизайн фокусується на загальному користувацькому досвіді. Обидва ці напрямки є важливими в дизайні цифрових продуктів і працюють разом для створення успішного та цікавого користувацького досвіду [12].

Головні принципи UI/UX дизайну:

1) зручність використання (Usability) – інтерфейс має бути інтуїтивним, легким для навігації та виконання завдань;

2) зосередженість на користувачеві (User-centered design) – дизайн повинен враховувати потреби, звички та переваги цільової аудиторії;

3) візуальна ієрархія – використання різних розмірів, кольорів, відступів для виділення найважливішої інформації;

4) послідовність (Consistency) – однаковий стиль, розташування елементів по всьому інтерфейсу для передбачуваного досвіду;

5) зворотний зв'язок (Feedback) – підтвердження успішних дій користувачів за допомогою анімацій, кольорів тощо;

6) доступність (Accessibility) – врахування потреб людей з обмеженими можливостями [13].

Рекомендації для створення привабливого дизайну:

1) використовувати від 2 до 4 основних кольорів (яскраві кольори притягують увагу, нейтральні – підкреслюють інформацію);

2) комбінувати не більше 2-3 шрифтів однієї сім'ї;

3) контрастність та читабельність тексту;

4) залишати вільне місце між елементами для легшого сприйняття і меншого візуального навантаження;

5) використовувати іконки, мінімалістичні форми та милі ілюстрації для емоційного зв'язку з користувачами;

6) додавати плавні анімовані переходи та відгуки на дії користувачів для приємного досвіду;

7) забезпечити коректне відображення на різних пристроях та розмірах екрану.

Одним з ефективних інструментів для візуалізації та прототипування дизайну веб-застосунків є Figma – хмарний багатоплатформенний сервіс. Вона надає розробникам та дизайнерам зручне середовище для створення макетів сторінок, компонентів інтерфейсу та прототипів взаємодії. Перед початком верстки веб-сторінок, Figma дозволяє візуалізувати та протестувати шаблони сторінок, що полегшує процес розробки та забезпечує кращу комунікацію між командою.

Використання Figma в процесі розробки веб-застосунків забезпечує ефективну візуалізацію та прототипування дизайну, що сприяє кращому розумінню вимог, підвищує узгодженість інтерфейсу та полегшує комунікацію між членами команди. Це допомагає зосередитися на створенні привабливого та інтуїтивно зрозумілого інтерфейсу користувача [14].

2.2.2 Комп’ютерні мережі та кібербезпека

Для розробки сучасних, оптимізованих застосунків необхідно розумітися щонайменше на базових поняттях у сфері комп'ютерних мереж. Крім того, важливо усвідомлювати, як відбувається обмін даними між комп'ютерами, та знати, як зробити цей обмін безпечним, щоб треті особи не мали змоги отримати доступ до приватних даних користувачів.

Для забезпечення безпеки передачі даних слід використовувати протокол HTTPS (Hyper Text Transfer Protocol Secure). Цей протокол забезпечує захищене з'єднання між клієнтом та сервером, шифруючи весь трафік за допомогою

криптографічних протоколів, таких як SSL/TLS. Це унеможливлює перехоплення та зчитування даних зловмисниками під час їх передачі.

Окрім використання захищеного протоколу, важливо також зберігати паролі користувачів лише в захищеному вигляді, використовуючи надійні алгоритми хешування, такі як bcrypt або scrypt. Це запобігає витоку паролів у разі компрометації бази даних [15].

Нарешті, необхідно належним чином налаштувати доступ до управління комп'ютером чи віддаленим сервером, на якому функціонує застосунок та зберігаються дані. Це включає встановлення надійних паролів, обмеження доступу за допомогою брандмауерів, регулярне оновлення програмного забезпечення та використання інструментів для моніторингу та виявлення загроз.

Дотримання цих практик кібербезпеки забезпечить надійний захист даних користувачів та захистить застосунок від потенційних зловмисників, гарантуючи безпечний обмін даними та конфіденційність інформації.

2.2.3 Розгортання

Ще одним важливим аспектом є ефективне розгортання, моніторинг та управління веб-застосунками вимагає знань та навичок у сфері DevOps. Розробники повинні вміти налаштовувати середовища розробки, тестування та продакшн, автоматизувати процеси розгортання, проводити моніторинг роботи застосунку та ефективно масштабувати його відповідно до потреб.

Для забезпечення ефективного розгортання та перенесення застосунків на різні платформи часто використовується принцип контейнеризації. Контейнеризація – це технологія віртуалізації на рівні операційної системи, яка дозволяє запускати кілька ізольованих екземплярів застосунків на одному фізичному сервері.

Одним з найпопулярніших інструментів для реалізації контейнеризації є платформа Docker. Docker дозволяє створювати ізольовані віртуальні середовища, відомі як контейнери, кожен з яких має власну самодостатню файлову систему, бібліотеки та конфігурації. Ця ізольованість забезпечує незалежність контейнерів від

операційної системи хоста та інших контейнерів, що полегшує перенесення застосунків між різними середовищами.

Основною перевагою контейнеризації є портативність та сумісність. Контейнери Docker можна легко переносити та розгортати на будь-якій платформі, що підтримує Docker, без необхідності адаптації застосунку до різних операційних систем чи середовищ. Це забезпечує однакову поведінку застосунку незалежно від того, де він запущений.

Для полегшення керування та розгортання складних застосунків, що складаються з кількох модулів, Docker пропонує інструмент Docker Compose. Він дозволяє описати та налаштувати взаємодію між контейнерами в одному файлі конфігурації. Кожен модуль застосунку оформлюється у вигляді окремого контейнера, а Docker Compose забезпечує їх взаємодію в рамках одного середовища. Це спрощує розгортання та масштабування складних систем, таких як трирівневі веб-застосунки, що можуть складатися з контейнерів для клієнтської частини, серверної частини та серверу бази даних [16].

Контейнеризація з використанням Docker та Docker Compose забезпечує ефективне розгортання, портативність та масштабованість застосунків, полегшуючи їх перенесення між різними середовищами та серверами. Це сприяє підвищенню продуктивності розробки, зменшенню часу на розгортання та забезпеченню однакової поведінки застосунку в різних умовах.

2.3 Забезпечення максимальної продуктивності

У сучасному динамічному світі, де час є найціннішим ресурсом, а кількість завдань та обов'язків невпинно зростає, люди постійно прагнуть максимізувати свою продуктивність. Адже лише ефективно розпоряджаючись своїм часом та зусиллями, можна досягати поставлених цілей, реалізовувати амбітні проекти та знаходити баланс між професійним та особистим життям. Розробка застосунків, орієнтованих на підвищення продуктивності користувачів, стає невід'ємною частиною цифрової

екосистеми, адже вони слугують потужними помічниками в досягненні поставлених цілей.

Створюючи програмний продукт, покликаний максимізувати продуктивність користувачів, необхідно керуватися найновішими та найефективнішими принципами і технологіями. Лише інтегруючи передові методики управління часом, візуалізації завдань та безперервної оптимізації процесів, можна забезпечити користувачам найкращий досвід та максимальну віддачу від використання застосунку.

2.3.1 Оптимізація процесів з використанням Kanban

Канбан-метод базується на трьох основних принципах: візуалізації, концентрації на одному завданні та управлінні потоком роботи для постійного вдосконалення процесів. Цю методику можна ефективно застосовувати не лише в командній роботі, але й для підвищення особистої продуктивності в повсякденних справах.

Ключовим інструментом є канбан-дошка, розділена на три секції: “Зробити” (список майбутніх справ), “В процесі” (поточні завдання) та “Зроблено” (виконані справи). Візуалізація завдань на дошці допомагає мозку швидше обробляти інформацію та забезпечує належну мотивацію для виконання наступних справ.

Обмеження кількості завдань у роботі є критично важливим для максимальної концентрації та продуктивності. Дослідження показують, що людина працює повільніше, намагаючись виконувати кілька справ одночасно. Тому рекомендується встановити ліміт, наприклад, не більше 3 завдань в процесі для однієї людини. Це дозволить зосередитися та уникнути розпорошення уваги.

Третій принцип – постійне спостереження за дошкою та вдосконалення особистих робочих процесів на основі візуального зворотного зв'язку. Відкрита візуалізація допоможе виявляти проблемні ділянки, визначати пріоритети та оптимізувати порядок виконання завдань для максимальної ефективності [17].

Використання канбан-дошки для планування щоденних справ забезпечує кілька ключових переваг для підвищення особистої продуктивності:

1) наочне бачення всіх поточних та майбутніх справ на одному полотні;

2) чітке розуміння пріоритетів та фокусування на найважливіших завданнях;

3) зменшення ризику забуття чи хаотичного виконання справ;

4) постійний аналіз та вдосконалення власних робочих шаблонів;

5) відчуття задоволення від виконаних завдань на колонці “Зроблено”.

Таким чином, канбан-дошка є потужним інструментом візуального планування, який допомагає максимізувати особисту продуктивність у повсякденних справах через візуалізацію, концентрацію та безперервне вдосконалення процесів.

2.3.2 Продуктивний застосунок для продуктивних людей

Оскільки цільовою аудиторією системи організації щоденних справ є люди, які прагнуть підвищити свою продуктивність, застосунок повинен максимально допомагати їм у цьому. Тому під час розробки постає проблема забезпечення високої швидкодії, зручності та інтуїтивно зрозумілого інтерфейсу застосунку.

Застосунок має бути оптимізований для швидкого завантаження, реагування на дії користувача та виконання операцій з даними. Навіть незначні затримки чи зависання можуть негативно вплинути на досвід користувача та знизити його продуктивність.

Крім того, важливо забезпечити простий та інтуїтивний інтерфейс, який дозволить користувачам легко орієнтуватися в системі та виконувати необхідні дії без зайвих зусиль. Зрозумілість та зручність використання є ключовими факторами при виборі інструментів для організації щоденних завдань.

Водночас, застосунок не повинен обмежуватися лише найпростішим функціоналом, оскільки це може негативно вплинути на його корисність та цінність для користувачів. Необхідно знайти баланс між простотою та потужністю, забезпечуючи необхідний набір функцій для ефективного управління завданнями та організації робочого процесу [18].

Ще одним важливим аспектом, який потребує реалізації, є забезпечення гнучкості та масштабованості застосунку, щоб він міг адаптуватися до зростаючих вимог користувачів і підтримувати розширений функціонал для максимально ефективної організації щоденних справ.

Процес розробки застосунку повинен орієнтуватися на оптимізацію, лаконічність та зручність використання, при цьому не обмежуючи користувачів лише базовим функціоналом. Це вимагає ретельного аналізу вимог користувачів, вивчення кращих практик та впровадження новітніх технологій і підходів для забезпечення високої продуктивності та задоволеності користувачів.

2.4 Функціональні можливості сучасних програмних засобів для організації щоденних справ з дошкою Kanban, порівняння розроблюваної системи з іншими

На ринку існує низка програмних рішень, які використовують методологію Kanban для організації щоденних справ та управління завданнями. Розглянемо деякі з них.

1. Trello – одна з найпопулярніших платформ для управління завданнями з використанням дошки Kanban. Вона має інтуїтивно зрозумілий інтерфейс, що дозволяє легко створювати дошки, додавати завдання у вигляді карток, призначати співробітників, встановлювати терміни виконання та додавати прикріплені файли. Перевагами Trello є простота використання, можливість інтеграції з багатьма іншими сервісами та наявність безкоштовного тарифного плану. Однак, для більш розширених функцій потрібно придбати платну підписку.

2. Asana – потужна система управління проектами, яка також підтримує використання дошки Kanban. Asana пропонує різноманітні способи візуалізації завдань, включаючи традиційні списки, календарі та дошки Kanban. Вона має багатий набір функцій, таких як можливість встановлювати залежності між завданнями, відстежувати часові витрати, інтегруватися з іншими інструментами та генерувати звіти. Asana добре підходить для командної роботи, але може бути складнішою у використанні для індивідуальних користувачів.

3. Notion – це всеохоплююча платформа для ведення нотаток, документації та управління завданнями. Notion дозволяє створювати дошки Kanban, а також

підтримує різноманітні типи контенту, такі як тексти, зображення, таблиці та бази даних. Перевагою Notion є її гнучкість та можливість налаштування під індивідуальні потреби користувачів. Окрім створення класичних канбан-дошок, Notion дозволяє розміщувати на одному робочому просторі тексти, таблиці, бази даних, вкладені сторінки, мультимедійний контент та багато іншого. Однак, вона може мати крутішу криву навчання порівняно з іншими рішеннями. Notion пропонує безкоштовний тарифний план, але він має певні обмеження щодо сховища даних та інтеграцій. Для повноцінного користування всіма можливостями може знадобитися підписка на платний план, що може бути дорогим для деяких користувачів [19].

Кожне з цих рішень має свої переваги та недоліки, а вибір найбільш підходящого варіанту залежить від конкретних потреб користувачів, розміру команди, бюджету та рівня гнучкості, який потрібен для організації щоденних справ.

Розроблюваний веб-застосунок має на меті надати користувачам альтернативне рішення з унікальними функціями та персоналізованими налаштуваннями, при цьому залишаючи всі функції безкоштовними та відкриваючи доступ до вихідного коду проєкту, дозволяючи іншим людям доповнювати проект бажаними доповненнями та виправленнями, потреба в яких виникає у них як у реальних користувачів застосунку, а також даючи іншим розробникам приклад використання та взаємодії обраних технологій.

Однією з переваг веб-застосунку є можливість швидкої реєстрації нових користувачів. Для того, щоб зареєструватися в системі, потрібно ввести лише мінімальний набір даних, таких як електронна пошта, ім'я користувача та бажаний пароль. Веб-застосунок не вимагає зайвої інформації від користувачів, що забезпечує зручність та економить їхній час. Весь процес реєстрації відбувається швидко та інтуїтивно, дозволяючи новим користувачам легко долучитися до використання програмного продукту та розпочати роботу з мінімальними затримками. Така мінімалістична реєстрація є додатковою перевагою порівняно з деякими конкуруючими рішеннями, які можуть вимагати більше особистих даних на етапі реєстрації.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

Загалом проект складається з трьох основних компонентів: модуль Flask, модуль Vue.js та база даних MongoDB. Кожен з цих модулів представлено у вигляді окремого Docker-контейнера, що забезпечує гнучкість та незалежність кожного компонента. Для налаштування зв'язку між контейнерами та їх об'єднання в одну мережу було створено файл конфігурації docker-compose.yml, який описує налаштування та взаємодію між різними частинами системи.

3.1 Обґрунтування засобів реалізації системи

Для реалізації програмного продукту було ретельно обрано сучасні та ефективні засоби розробки, які забезпечують гнучкість, масштабованість та зручність у використанні. Розглянемо детальніше обґрунтування вибору кожного з них.

3.1.1 Вибір мови програмування та фреймворку для серверної частини

При виборі мови програмування для розробки серверної частини веб-застосунку варто звертати увагу на такі аспекти, як легкість у вивченні та швидкість розробки, швидкодія, надійність і наявні розширення (бібліотеки), що полегшать розробку прикладного програмного інтерфейсу. Серед найпопулярніших варіантів є Python з фреймворками Django та flask, JavaScript (Node.js) із Express.js, NestJS та PHP з його Laravel.

Врахувавши вищезазначені критерії та проаналізувавши існуючі рішення, було обрано мову програмування Python із фреймворком Flask. Ця комбінація дозволяє просто та швидко будувати надійні та масштабовані застосунки, при цьому забезпечуючи дуже обширні можливості та легкість у інтеграції з будь-якими іншими технологіями.

3.1.2 Вибір фреймворку для клієнтської частини

Клієнтська частина є лицем застосунку та основним постачальником користувацького досвіду, тому свідомий та виважений вибір технологій його розробки є критично важливим. Існує три головні відомі JavaScript-фреймворки для розробки клієнтського застосунку, серед яких найчастіше обирають програмісти: React, Angular та Vue.js. Кожен із них є чудовим вибором та зазвичай підійде для розробки будь-якого застосунку, проте між ними є деякі відмінності, на які варто звернути увагу перед тим, як зробити свій вибір.

Враховуючи необхідність у легкому для вивчення та застосування веб-фреймворку, який при цьому забезпечить максимальну продуктивність та найкращий користувацький досвід, було обрано Vue.js. Він вважається найпрогресивнішим, активно розвивається, має обширні можливості та використовує віртуальний DOM, що забезпечує його підвищену продуктивність.

3.1.3 Вибір бази даних

У процесі вибору системи керування базами даних для системи були розглянуті різні варіанти, включаючи реляційні (PostgreSQL, MySQL) та NoSQL бази даних (Cassandra, MongoDB). Враховуючи орієнтування на легкість у вивченні та впровадженні, при цьому сучасність та широкий функціонал під будь-які потреби, а також необхідність забезпечення високої масштабованості та продуктивності, було обрано MongoDB – популярну документно-орієнтовану NoSQL базу даних. Також, взаємодії з MongoDB існує офіційна Python-бібліотека PyMongo, яка надає зручний та інтуїтивно зрозумілий API для виконання операцій CRUD.

Використання MongoDB забезпечує ефективне та гнучке зберігання даних, дозволяючи обробляти більш складні структури та забезпечуючи високу продуктивність та масштабованість розроблюваної системи.

3.1.4 Контейнеризація та розгортання системи

Для того, щоб забезпечити легке та надійне перенесення і розгортання застосунку на сторонній сервер було прийняте рішення використовувати технологію

контейнеризації Docker. Вона гарантує, що застосунок працюватиме однаковим чином у будь-якому середовищі, де його запущено за рахунок упакування сервісів у ізольовані середовища з їхніми залежностями та власною незалежною файловою системою.

Крім цього, було використано утиліту Docker Compose, яка дозволяє налаштувати декілька сервісів, таких як клієнтський, серверний застосунки та MongoDB-сервер, у вигляді окремих контейнерів, які функціонують в межах одного проекту та взаємодіють між собою.

3.2 Архітектура системи

Застосунок було розроблено на основі трирівневої архітектури, при якій застосунок складається з трьох основних сервісів: клієнтський застосунок, з яким взаємодіє користувач через браузер; серверний застосунок, який надає прикладний програмний інтерфейс та виконує основну бізнес-логіку; система управління базою даних, де зберігаються всі дані. Така архітектура є найбільш звичною та популярною у розробці сучасних веб-застосунків. Таким чином, певні дії користувача у клієнтському застосунку викликають надсилання HTTP-запитів до серверної сторони, яка в свою чергу ідентифікує користувача та визначає, до яких даних та дій має доступ цей користувач.

При цьому, клієнтська та серверна частини можуть не залежати одне від одного та розроблятись паралельно різними людьми чи командами. У розроблюваній системі, їх було реалізовано в окремих підкаталогах відповідно “frontend” та “backend”. У кореневому каталозі проекту, крім них міститься також файл конфігурації “docker-compose.yaml”, який описує три вищезазначені сервіси у вигляді Docker-контейнерів “frontend”, “backend” та “db”. Контейнери “backend” та “db” при цьому розміщені у спільну закриту мережу, що відкриває для Python-застосунку можливість здійснювати безпечні запити до MongoDB-сервера, не відкриваючи

зовнішній доступ до нього. З допомогою однієї команди “# docker-compose up -d” всі три сервіси починають працювати.

У контейнері із серверною частиною при запуску оновлюються всі необхідні бібліотеки (залежності) та запускається Flask-сервер на 5000-му порті. Після цього кінцеві точки стають доступні на хості http://backend:5000/.

Контейнер із клієнтською частиною при запуску оновлює всі необхідні пакети та із вихідного коду Vue-застосунка компілює статичні файли, такі як index.html, index.css, main.js. Після цього, ці статичні файли обслуговуються на кореневому маршруті хоста за допомогою веб-сервера NGINX. Також, за допомогою функції проксі у NGINX, відбувається переадресація зовнішніх запитів на підмаршрут /api до 5000-го порту із Flask-сервером.

Контейнер “db” відповідає за запуск серверу MongoDB. Він використовує офіційний образ MongoDB 4.4, а дані зберігаються у named volume mongodb_data для забезпечення персистентності даних.

Із такою конфігурацією забезпечується надійна та безпечна робота веб-застосунку з трирівневої архітектурою, при цьому його можна без проблем перенести та запустити на віддаленому сервері, і за допомогою технології Docker буде забезпечено цілісність та належне функціонування системи

3.3 Процес ідентифікації користувача

Для забезпечення належного рівня безпеки та контролю доступу у веб-застосунку реалізовано систему авторизації та аутентифікації користувачів. Цей процес відбувається за наступним алгоритмом:

1) авторизація (логін);
2) генерація та зберігання токена;
3) використання токена;
4) аутентифікація.

Під час спроби авторизації користувача (логіну) клієнтський застосунок надсилає введені облікові дані (email та пароль) на спеціальну кінцеву точку. На стороні серверу пароль користувача не зберігається у відкритому вигляді з міркувань безпеки. Замість цього, при реєстрації для кожного користувача було згенеровано унікальну “сіль” (salt) – випадкове значення, яке конкатенується з паролем перед хешуванням. Під час логіну сервер виконує цю ж операцію: конкатенує введений пароль з сіллю користувача, хешує їх за допомогою криптографічного алгоритму SHA-256 та порівнює отриманий хеш зі збереженим хешем дійсного пароля користувача. Якщо хеші збігаються, то пароль вважається правильним.

Якщо авторизація успішна, сервер генерує новий унікальний токен у форматі UUID.v4 (наприклад, “d9f68c8e-f7c4-4b9c-b3d1-e6c63b8db3f2”). Цей токен зберігається у базі даних, пов’язуючись із ідентифікатором авторизованого користувача. Згенерований токен надсилається у тілі відповіді на запит авторизації до клієнтської частини застосунку.

Після отримання токена від серверу, браузер зберігає його у localStorage. Надалі цей токен використовується у всіх наступних запитах до кінцевих точок, які вимагають авторизації, шляхом додавання його у заголовок Authorization у форматі “Bearer <токен>” (наприклад, “Bearer d9f68c8e-f7c4-4b9c-b3d1-e6c63b8db3f2”).

На стороні серверного застосунку, для кожного захищеної кінцевої точки, що вимагає авторизації, реалізовано перевірку наявності та коректності токена у заголовку Authorization запиту. Якщо токен є коректним, сервер знаходить у базі даних користувача, якому належить цей токен, і таким чином аутентифікує його, надаючи доступ до запитуваного ресурсу або виконання дії.

Такий підхід з використанням токенів забезпечує безпечну авторизацію та аутентифікацію користувачів, запобігаючи передачі облікових даних у відкритому вигляді та зберігаючи конфіденційність паролів за допомогою хешування та використання солей.

3.4 База даних

Система опирається на три основні сутності: Користувачі, Дошки та Завдання на дошках (рисунок 3.1).

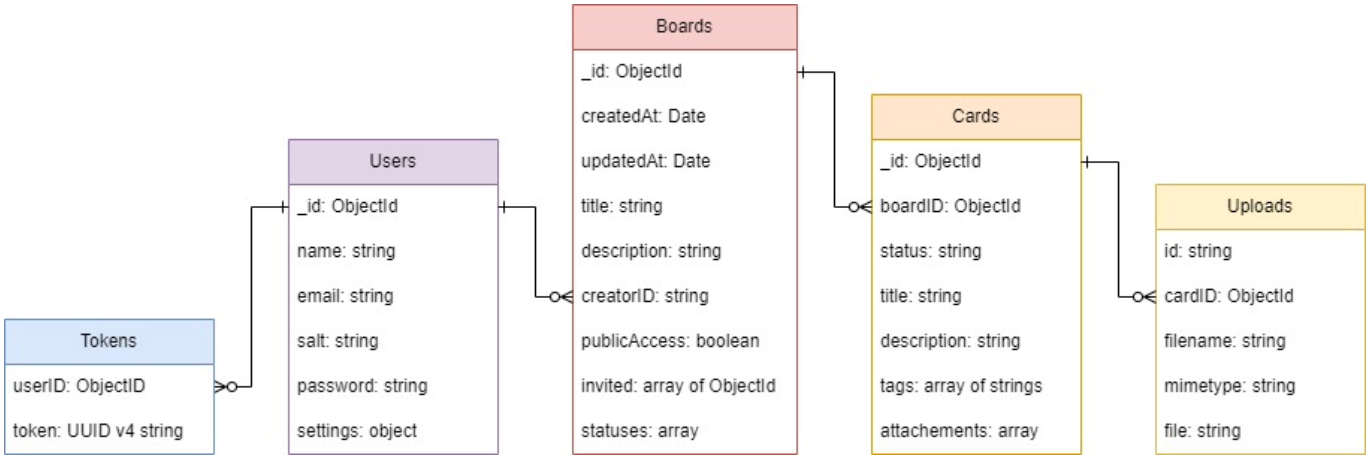

Рисунок 3.1 – Структура бази даних

Відповідно, база даних складається з наступних колекцій:

– Users зберігає інформацію про користувачів системи;
– Tokens зберігає авторизаційні токени користувачів;
– Boards зберігає інформацію про канбан-дошки;
– Cards зберігає інформацію про завдання (картки) на канбан-дошках;
– Uploads зберігає завантажені користувачами файли.

Детальний опис структури колекції Users можна побачити в таблиці 3.1.

Таблиця 3.1 – Колекція Users

Назва поля	Тип поля	Опис поля
_id	ObjectId	унікальний ідентифікатор користувача
name	string	ім'я чи псевдонім користувача
email	string	електронна пошта користувача; вона повинна бути унікальною і вона використовуватиметься для авторизації

Продовження таблиці 3.1

salt	string	“сіль”, рядок з випадкових символів, що конкатується до пароля перед його хешуванням
password	string	захешований пароль користувача(перед цим конкатенований з сіллю) з алгоритмом SHA-256
settings	object	об'єкт, який містить різноманітні налаштування користувача з різними ключами, має такий формат:{ “defaultStatuses”: array – масив назв статусів, які з'являтимуться в новостворених дошках поточного користувача за замовчуванням. При реєстрації записується список “To Do”, “В роботі”, “Виконано”}

Поля та їх опис колекції Tokens можна побачити в таблиці 3.2.

Таблиця 3.2 – Колекція Tokens

Назва поля	Тип поля	Опис поля
userID	ObjectId	ідентифікатор користувача
token	UUID v4 string	токен; для авторизації запитів користувача, заголовок Authorization порівнюватиметься із “Bearer” + token

У таблиці 3.3 описані поля колекції Boards.

Таблиця 3.3 – Колекція Boards

Назва поля	Тип поля	Опис поля
_id	ObjectId	унікальний ідентифікатор дошки
createdAt	Date	дата створення дошки
updatedAt	Date	дата останнього оновлення дошки
title	string	опціональна назва дошки, яку може ввести користувач

Продовження таблиці 3.2

description	string	опціональний опис дошки, який може ввести користувач
date	string	дата в форматі “YYYY-MM-DD”, до якої прив'язана дошка; якщо дошка не є прив'язаною до дати, значенням буде null
creatorID	ObjectId	ідентифікатор користувача, що створив цю дошку
publicAccess	boolean	чи може будь-який зареєстрований користувач, маючи посилання, отримати доступ до дошки
invited	array of ObjectId	масив користувачів, запрошених до роботи з дошкою
statuses	array	масив переліку статусів, що присутні саме на цій дошці

Детальний опис структури колекції Cards можна побачити в таблиці 3.4.

Таблиця 3.4 – Колекція Cards

Назва поля	Тип поля	Опис поля
_id	ObjectId	унікальний ідентифікатор картки
boardID	ObjectId	ідентифікатор дошки, на якій знаходиться картка
status	string	назва статусу, у якому знаходиться картка
title	string	назва картки (основна суть завдання)
description	string	детальний опис картки
tags	array of strings	масив “тегів” картки – набір будь-яких слів, що є асоціаціями/позначками до картки, вводиться користувачем
attachments	array	масив об'єктів з ключами fileID та name – прикріпленими до картки файлами

Структуру останньої з колекцій можна побачити в таблиці 3.5.

Таблиця 3.5 – Колекція Uploads

Назва поля	Тип поля	Опис поля
_id	ObjectId	унікальний ідентифікатор файла
cardID	ObjectId	ідентифікатор картки, до якої прикріплено файл
filename	string	назва файлу
mimetype	string	MIME-тип файлу
file	string	вміст файлу, закодований в формат base64

Така структура бази даних забезпечує зручне зберігання та маніпулювання даними, дозволяє легко розширювати функціональність системи в майбутньому шляхом додавання нових полів або колекцій.

3.5 Структура клієнтської частини

У розробленому застосунку клієнтська частина побудована з використанням фреймворку Vue.js та деяких допоміжних до нього бібліотек, що разом дозволяє створювати інтерактивні користувацькі інтерфейси. Застосунки з Vue.js працюють за принципом Single-Page Application, тобто сторінка завантажується лише один раз, після чого весь вміст на сторінках завантажується та оновлюється динамічно без перезавантаження сторінки.

Архітектура застосунку базується на компонентному підході, який є основою Vue.js. Це означає, що код складається з різноманітних модулів та компонентів, кожен з яких може описувати свою унікальну розмітку та логіку, пов'язану з певною функціональністю. Такі компоненти можна багаторазово перевикористовувати на різних сторінках, а також вбудовувати один компонент всередині іншого, забезпечуючи ієрархічність, читабельність та масштабованість.

Для маршрутизації між сторінками та компонентами використовується бібліотека Vue Router. Вона дозволяє співставляти компоненти застосунку із URL-адресами, автоматично виконуючи навігацію без перезавантаження сторінки. Це реалізовано за допомогою елементу “router-view”, який автоматично підміняється на компонент, що прив'язаний до поточного відкритого маршруту. З використанням таких механізмів, у розроблюваному застосунку створено компоненти з ієрархічною структурою, що відповідають за різні сторінки та функції (рисунок 3.2).

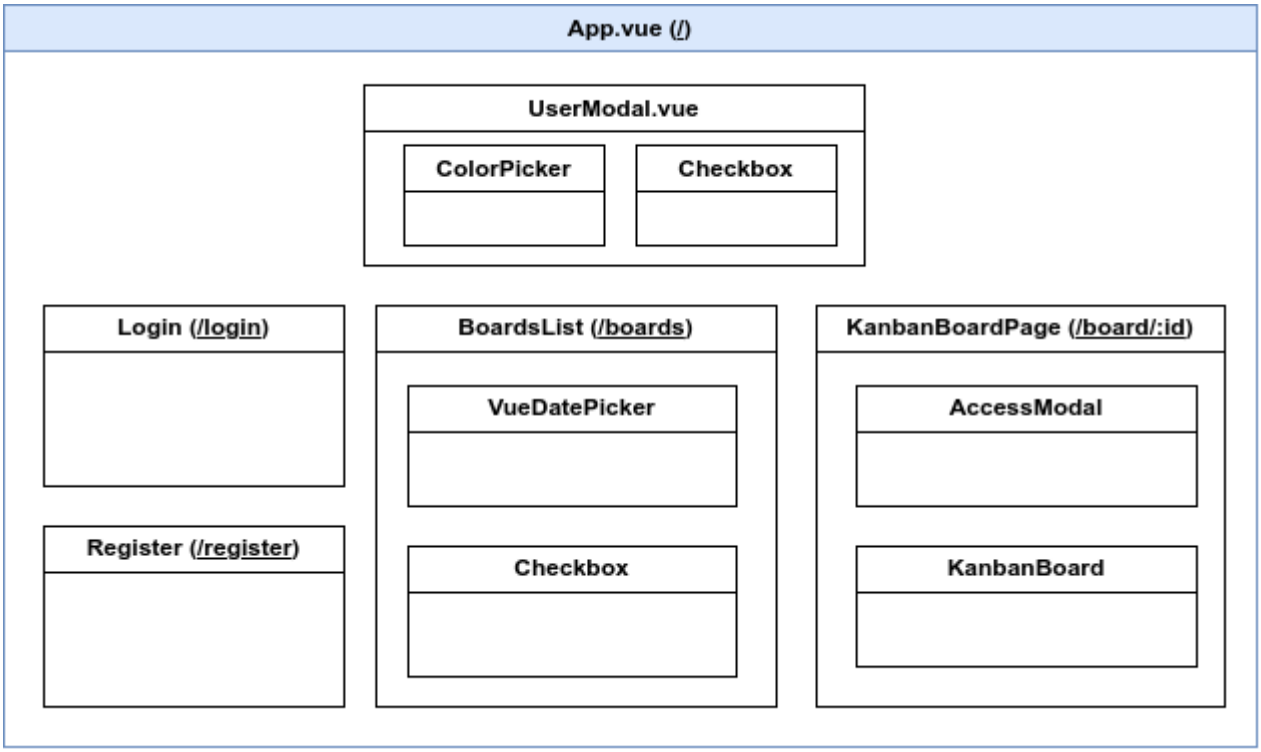

Рисунок 3.2 – Основні компоненти клієнтської частини

У кореневому компоненті “App.vue”, який служить точкою входу, вгорі розміщено верхню панель з логотипом застосунку та іконкою користувача, натискання на яку відкриває модальне вікно для перегляду та зміни інформації профіля та налаштувань – “UserModal.vue”. Під нею, розташовано елемент “router-view”, що відповідає за маршрутизацію. Таким чином, на всіх сторінках вгорі буде присутня відповідна панель, а нижче – весь вміст поточної відкритої сторінки, що забезпечує зручність та сталий дизайн.

Автоматично при відкритті кореневого маршруту застосунку, відбувається перевірка, чи є поточний користувач авторизованим. Якщо ні, він автоматично перенаправляється на маршрут /login, що відповідає Vue-компоненту “Login”. Якщо користувач ще не зареєстрований, звідси він може після натискання на спеціальну кнопку потрапити на сторінку реєстрації – /register.

Після успішної авторизації або реєстрації користувач перенаправляється на головну сторінку застосунку за маршрутом /boards – компонент “BoardsList.vue”. Він призначений для відображення існуючих дошок користувача, прив'язаних та не прив'язаних до дат, а також створення нових дошок. Сюди також автоматично потрапляють уже авторизовані користувачі, що відкрили кореневий маршрут застосунку або сторінку /login чи /register.

Коли користувач обирає одну з існуючих дошок або створює нову, відбувається переадресація на маршрут “/boards/:id”, де “:id” – URL-параметр, що відповідає унікальному ідентифікатору конкретної дошки. У відповідному компоненті “KanbanBoardPage.vue” вгорі знаходиться панель для управління назвою, описом та доступом до дошки, а під нею – сама Kanban-дошка. Вона реалізована через окремий модуль “KanbanBoard.vue” (рисунок 3.3).

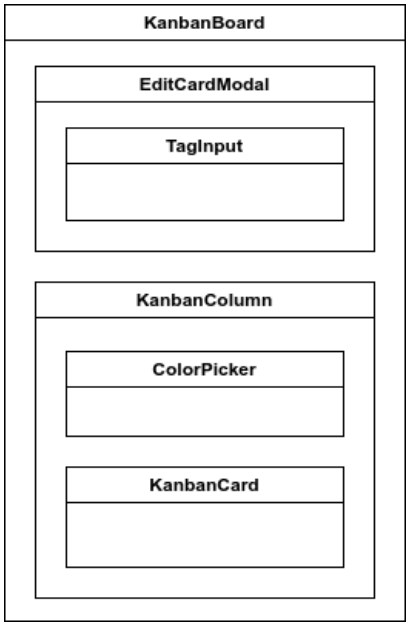

Рисунок 3.3 – Декомпозиція компонента KanbanBoard

Він використовується для контролю над усіма змінами на дошці, такими як зміни статусів та редагування карток, відстежуючи події від вбудованих в нього компонентів “KanbanColumn” для рендерингу колонок-статусів, з можливістю редагування їх назви, кольору фону та видалення, а також “KanbanCard” для рендерингу карток з завданнями на стовпцях.

Також, при натисканні на одну з карток на дошці, відкривається модальне вікно по центру екрану для редагування інформації про картку, реалізоване компонентом “EditCardModal.vue”. При натисканні на іконку поділитись у верхній панелі з'являється вікно для управлінням доступу до даної дошки – компонент “AccessModal.vue”. Крім цього, присутні декілька додаткових модулів: “TagInput.vue” для введення текстових тегів для карток, “Checkbox.vue” для перевикористання спеціальних чекбоксів з дизайном, “ColorPicker.vue” для вибору кольору із заданого переліку (для зміни кольору фону стовпця в налаштуваннях та на самій дошці).

Така архітектура дозволяє легко підтримувати та розширювати застосунок, а також забезпечує високу швидкодію завдяки механізмам віртуального DOM та Single-Page Application фреймворку Vue.js.

3.6 Структура серверної частини

Для забезпечення кращої читабельності та модульності серверної частини застосунку було використано механізм flask.Blueprint, який є вбудованою функціональністю фреймворку Flask. Цей механізм дозволяє виносити певні маршрути з їхньою імплементацією в окремі файли, а потім імпортувати їх в головний скрипт застосунку за допомогою однієї команди.

Розподіл маршрутів API було здійснено відповідно до їх функціонального призначення та логічної організації. Це дозволило структурувати код та забезпечити зручність подальшої підтримки та розширення системи. В результаті аналізу існуючих сутностей та функціональних вимог до системи, кінцеві точки було

розділено на три основні компоненти: такі, що відповідають за авторизацію, реєстрацію та керування обліковим записом; такі, що відповідають за отримання списку та керування Kanban-дошками та переліком статусів на них; такі, що відповідають за картки на дошках, їх редагування та завантаження файлів до них.

Такий розподіл маршрутів забезпечує логічну структуризацію API та дозволяє ефективно організувати роботу з різними сутностями системи. Кожен компонент відповідає за свою специфічну область функціональності, що спрощує розуміння та підтримку коду.

3.6.1 Кінцеві точки для роботи з користувачем

У файл blueprints/user.py винесено імплементацію кінцевих точок, що виконують операції з авторизаційними токенами та сутностями користувачів.

Таблиця 3.6 – Кінцеві точки для роботи з користувачем

HTTP метод	URL	Опис	Вхідні параметри	Можливі коди статусів
POST	/login	Авторизація користувача	email password	200 – Успіх 400 – Неправильна пошта або пароль
POST	/register	Реєстрація	name email password	201 – Створено нового користувача 400 – Некоректні дані
POST	/logout	Вихід користувача з системи		200 – Успіх 401 – Користувач не авторизований
GET	/profile	Отримання даних про поточного користувача		200 – Успіх 401 – Користувач не авторизований

Продовження таблиці 3.6

PATCH	/profile	Редагування імені, пошти або паролю	name email currentPassword password	200 – Успіх 400 – Некоректні дані 401 – Користувач не авторизований
PATCH	/settings	Редагування налаштувань користувача	defaultStatuses	200 – Успіх 400 – Некректні дані 401 – Користувач не авторизований

Імплементація цих кінцевих точок дозволяє виконувати операції реєстрації, авторизації (рисунок 3.4) та аутентифікації, переглядати та редагувати налаштування та інформацію профілю.

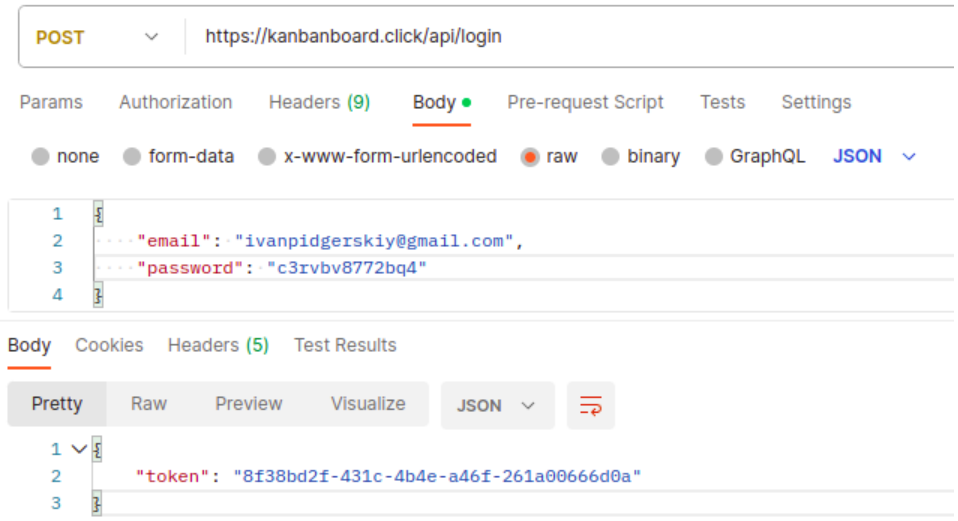

Рисунок 3.4 – Приклад виконання запиту на авторизацію з утилітою Postman

Таким чином відбувається вхід в систему, і сервер надає клієнту спеціальний токен, який потрібно використовувати в заголовках наступних API-запитів.

3.6.2 Кінцеві точки для управління дошками

У файл blueprints/boards.py винесено кінцеві точки, що відповідають за виконання загальних операцій над Kanban-дошками.

Таблиця 3.7 – Кінцеві точки для управління дошками

HTTP метод	URL	Опис	Вхідні параметри	Можливі коди статусів
GET	/boards	Отримання списку доступних користувачу дошок		200 – Успіх 400 – Некоректні дані 401 – Користувач не авторизований
POST	/boards	Створення нової дошки	name description statuses	200 – Успіх 401 – Користувач не авторизований 404 – Дошку не знайдено
GET	/boards/{id}	Отримання детальної інформації про дошку	<boardID>	200 – Успіх 400 – Некоректні дані 401 – Користувач не авторизований 404 – Дошку не знайдено
PATCH	/boards/{id}	Редагування назви, опису дошки	<boardID> name description	200 – Успіх 400 – Некоректні дані 401 – Користувач не авторизований 404 – Дошку не знайдено

Продовження таблиці 3.7

PATCH	/boards/{id}/publicAccess	Редагування відкритого доступу до дошки	<boardID> publicAccess	200 – Успіх 400 – Некоректні дані 401 – Користувач не авторизований 404 – Дошку не знайдено
POST	/boards/{id}/invite	Запрошення користувача до роботи з дошкою	<boardID> email	200 – Успіх 400 – Некоректні дані 401 – Користувач не авторизований 404 – Дошку не знайдено
DELETE	/boards/{id}/invite	Видалення запрошеного користувача із дошки	<boardID> userID	200 – Успіх 400 – Некоректні дані 401 – Користувач не авторизований 404 – Дошку не знайдено
POST	/boards/{id}/status	Додавання нового статусу на дошці	<boardID> name color	200 – Успіх 400 – Некоректні дані 401 – Користувач не авторизований 404 – Дошку не знайдено

Продовження таблиці 3.7

PATCH	/boards/{id}/ status	Редагування назви, кольору статусу	<boardID> statusName newStatusName newStatusColor	200 – Успіх 401 – Користувач не авторизований
DELETE	/boards/{id}/ status	Видалення статусу на дошці	<boardID> statusName	200 – Успіх 400 – Некоректні дані 401 – Користувач не авторизований

Таким чином, із реалізацією даного переліку кінцевих точок відкривається можливість для перегляду списку існуючих дошок користувача, отримання детальної інформації про конкретну дошку(рисунок 3.5), створення нових дошок, зміна режимів стороннього доступу до них та редагування основної інформації, такої як назва, опис, перелік статусів на дошці.

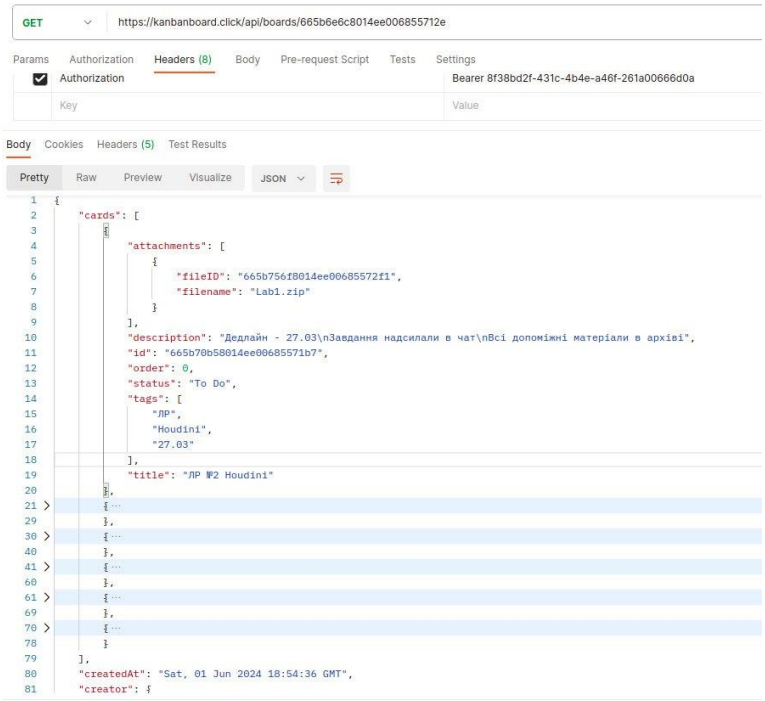

Рисунок 3.5 – Приклад виконання авторизованого запиту на отримання даних про Kanban-дошку

Як можна помітити, у заголовку “Authorization” використовується раніше отриманий токен для того, щоб отримати доступ до конфіденційних даних користувача. Дана кінцева точка дозволяє отримати детальну інформацію про певну Kanban-дошку та перелік статусів і карток на ній.

3.6.3 Кінцеві точки для управління картками на дошках

У програмному файлі blueprints/cards.py реалізовано кінцеві точки для редагування та пересування карток із завданнями на Kanban-дошці.

Таблиця 3.8 – Кінцеві точки для управління картками на дошках

HTTP метод	URL	Опис	Вхідні параметри	Можливі коди статусів
POST	/boards/{id}/cards	Додавання нової картки на дошку	<boardID> statusName statusColor	200 – Успіх 400 – Некоректні дані 401 – Користувач не авторизований 404 – Дошку не знайдено
PATCH	/boards/{id}/cards	Редагування назви, опису, тегів картки	<boardID> <cardID> name description tags	200 – Успіх 400 – Некоректні дані 401 – Користувач не авторизований 404 – Дошку або картку не знайдено
DELETE	/boards/{id}/cards	Видалення картки на дошці	<boardID> <cardID>	200 – Успіх 401 – Користувач не авторизований 404 – Дошку не знайдено

Продовження таблиці 3.8

POST	/uploads/{cardID}	Прикріплення файлу до картки	<cardID>	200 – Успіх 400 – Некоректний файл 401 – Користувач не авторизований 404 – Картку не знайдено
GET	/uploads/{fileID}	Скачування прикріпленого файлу	<fileID>	200 – Успіх 401 – Користувач не авторизований 404 – Файл не знайдено

З реалізацією зазначених кінцевих точок з’являється можливість інтерактивно взаємодіяти з дошкою та картками на ній, створюючи нові та видаляючи існуючі завдання, редагуючи їх вміст та пересуваючи їх між статусами.

3.7 Публікація застосунку в мережі Інтернет

Для того, щоб забезпечити доступ користувачів до розробленого веб-застосунку, необхідно розмістити його на віддаленому сервері та налаштувати доменне ім'я. Це дозволить користувачам звертатись до застосунку за зручним та зрозумілим посиланням.

У якості хостингу для розміщення застосунку було обрано платформу Contabo, яка надає послуги оренди приватних віртуальних серверів (VPS). Contabo є надійним постачальником та пропонує одні з найдешевших тарифів з потужними характеристиками серверів. Для розгортання застосунку було обрано базовий план з сервером, розташованим в Європі, що включає 8ГБ оперативної пам'яті, 200ГБ SSD-

накопичувач та швидке інтернет-з'єднання. Такі параметри є достатніми для стабільної роботи розробленої системи та забезпечення комфортної роботи користувачів.

Окрім розміщення застосунку на сервері, важливим етапом є придбання доменного імені. Доменне ім'я – це унікальна адреса веб-сайту в мережі Інтернет, яка дозволяє користувачам легко знаходити та звертатись до нього. Використання власного доменного імені підвищує довіру до застосунку, дозволяє легко поширювати посилання на нього та запам'ятовувати його адресу. Крім того, доменне ім'я відіграє важливу роль у формуванні бренду та іміджу продукту.

Серед найпопулярніших реєстраторів доменних імен було розглянуто Namecheap та GoDaddy. В результаті пошуку вдалося знайти недорогий, проте цікавий та підходящий для застосунку варіант на Namecheap – “kanbanboard.click”. Враховуючи специфіку розроблюваної системи, було вирішено підлаштувати брендинг та розробити відповідний логотип під обране доменне ім'я (рисунок 3.6).

Рисунок 3.6 – Логотип веб-застосунку

Для того, щоб прив'язати придбане доменне ім'я до сервера на Contabo, необхідно виконати наступні кроки:

1) в панелі керування Contabo створити DNS-зону для домену та вказати IP-адресу сервера;

2) скопіювати nameservers, надані Contabo (зазвичай вони мають вигляд ns1.contabo.net, ns2.contabo.net, ns3.contabo.net);

3) в обліковому записі Namecheap перейти до налаштувань домену та вказати скопійовані nameservers у розділі “Custom DNS”;

4) зачекати деякий час (до 48 годин) для повного поширення DNS-записів.

Оскільки на даному етапі на орендованому сервері функціонуватиме лише один розроблений застосунок, достатньо відкрити його на 80-й порт. Таким чином, він автоматично буде доступний за обраним доменним іменем [20].

Крім цього, для забезпечення захищеності та конфіденційності даних користувачів необхідно використовувати протокол HTTPS (HTTP Secure). Це гарантує, що інформація, яку вводять користувачі (наприклад, паролі чи особисті дані), буде надійно захищена від перехоплення зловмисниками.

Щоб налаштувати підтримку HTTPS для доменного імені, було використано інструмент Certbot. Він спрощує процес отримання та налаштування SSL-сертифікату від центру сертифікації Let's Encrypt. Слідуючи чітким інструкціям з офіційного гайду, вдалося налаштувати HTTPS за лічені хвилини.

У результаті виконаних дій, розроблений веб-застосунок успішно опубліковано в мережі Інтернет та він доступний для використання за захищеним посиланням https://kanbanboard.click. Користувачі можуть звертатись до застосунку, реєструватись, створювати власні канбан-дошки та ефективно організовувати свої щоденні завдання.

Публікація застосунку є важливим етапом, який дозволяє перетворити локальну розробку на продукт, доступний широкому колу користувачів. Обраний хостинг Contabo забезпечує надійну та швидку роботу сервера, а придбане доменне ім'я “kanbanboard.click” робить застосунок впізнаваним та зручним у використанні. Налаштування HTTPS гарантує конфіденційність та цілісність даних користувачів, що є критично важливим для веб-застосунку такого типу.

4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для належного функціонування системи організації щоденних справ з дошкою Kanban та забезпечення коректної роботи всіх її компонентів важливим є дотримання визначених вимог та інструкцій користувачем. Від правильності виконання наведених нижче вказівок безпосередньо залежить стабільність, продуктивність та зручність використання продукту.

4.1 Системні вимоги

Оскільки розроблена система є веб-застосунком, розгорнутим на сервері, для її використання користувачам достатньо мати сучасний веб-браузер та підключення до Інтернету. На відміну від традиційних стільникових або мобільних застосунків, веб-застосунок не вимагає встановлення на комп'ютер або мобільний пристрій користувача, що робить його доступним з будь-якого пристрою, здатного відкривати веб-сторінки.

Система організації щоденних справ з дошкою Kanban підтримується більшістю сучасних веб-браузерів, таких як Google Chrome, Mozilla Firefox, Microsoft Edge, Safari та Opera. Для забезпечення оптимальної роботи застосунку рекомендується використовувати останні стабільні версії цих браузерів.

Крім того, для забезпечення належної швидкості та продуктивності системи, користувачам рекомендується мати стабільне підключення до Інтернету з достатньою швидкістю завантаження та відвантаження даних. Хоча застосунок не вимагає надзвичайно високих швидкостей Інтернету, низька швидкість підключення може негативно вплинути на продуктивність застосунку та досвід користувача.

4.2 Робота користувача з програмним продуктом

Першою сторінкою, на яку потрапляє користувач, є сторінка авторизації. На ній користувач бачить форму для входу в систему, де йому потрібно ввести адресу своєї електронної пошти та пароль у відповідні поля (рисунок 4.1). Якщо введені дані правильні, то після натискання на кнопку “Вхід” користувач потрапляє на головну сторінку застосунку. Якщо ж дані некоректні, то з'явиться повідомлення про помилку.

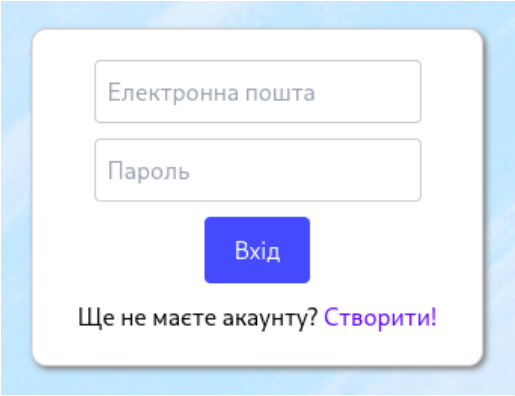

Рисунок 4.1 – Сторінка авторизації (Vue-компонент LoginPage)

Якщо користувач ще не зареєстрований, під формою авторизації є посилання “Створити!”. Якщо користувач ще не має акаунту в системі, він може натиснути на це посилання та перейти на сторінку реєстрації (рисунок 4.2).

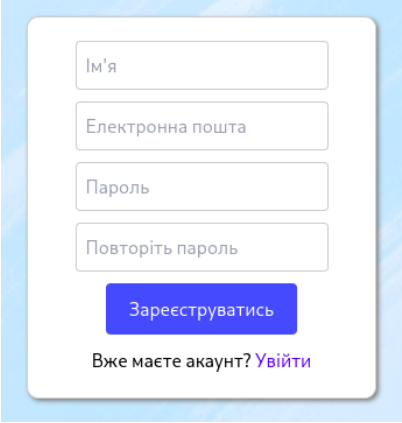

Рисунок 4.2 – Сторінка реєстрації (Vue-компонент RegisterPage)

На сторінці реєстрації показана форма для створення нового акаунту користувача. Потрібно ввести своє ім'я, адресу електронної пошти, пароль та повторити пароль. Всі поля обов'язкові для заповнення. При введенні email та паролю відбувається їх перевірка на коректність. Також перевіряється збіг введених паролів (рисунок 4.3).

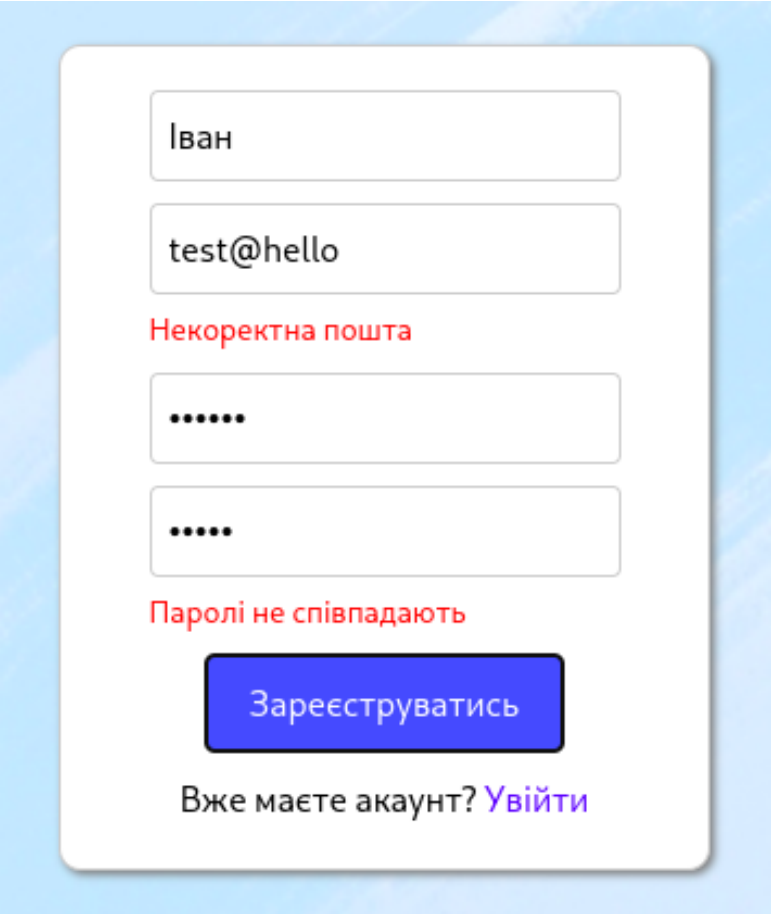

Рисунок 4.3 – Перевірка введених даних на сторінці реєстрації

Якщо користувач вже має акаунт, то під формою реєстрації є посилання “Увійти”. Натиснувши на нього, користувач перейде на сторінку авторизації, де зможе увійти в систему зі своїми даними.

Головна сторінка містить список дошок користувача (рисунок 4.4). Кожна дошка показана окремим блоком, де вказана її назва, ім'я користувача, який її створив, дата останнього оновлення. Для створення нової дошки є спеціальний блок з кнопкою “+”.

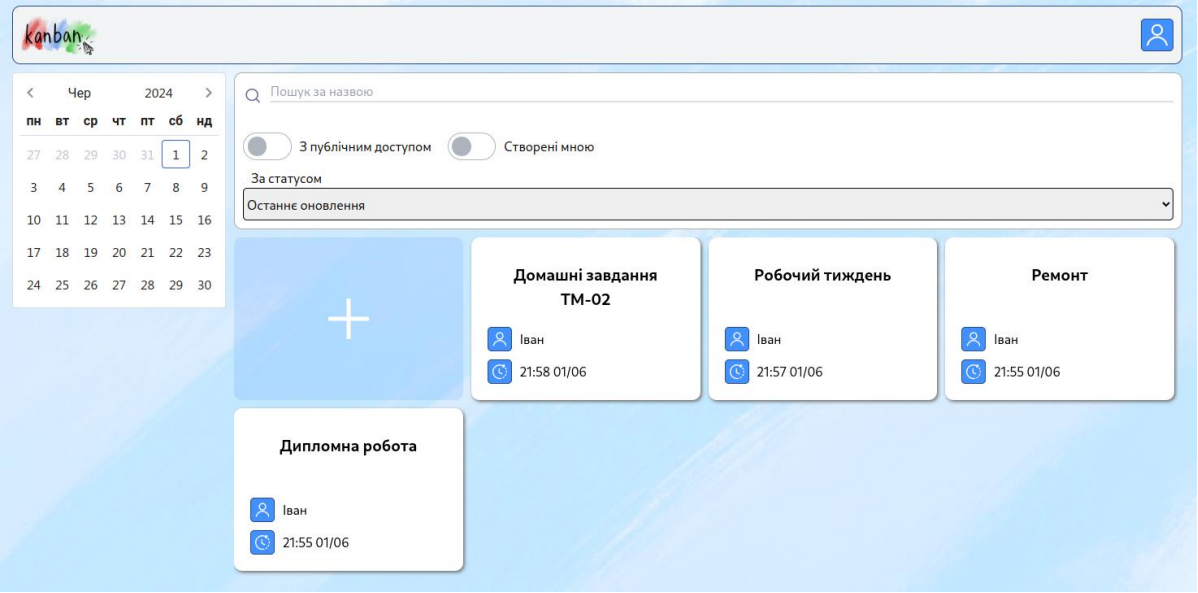

Рисунок 4.4 – Головна сторінка (Vue-компонент BoardsList)

Ліворуч на сторінці показано календар. Дати, на які заплановані дошки, підсвічуються (рисунок 4.5). Таким чином користувач може бачити, на які дні вже є дошки, а на які – ні, а також переглядати історію дошок за минулі дні. Клік на підсвічену дату відкриває відповідну дошку. Це зручно для планування та перегляду дошок за датами.

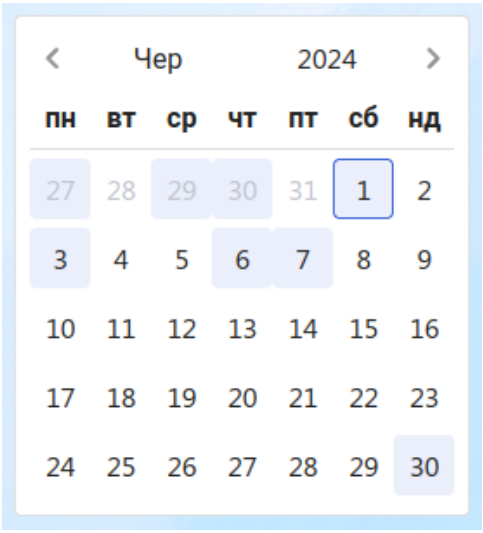

Рисунок 4.5 – Календар на головній сторінці з позначенням дат, де вже створені дошки

Над списком дошок є панель пошуку, де користувач може ввести текст для фільтрації дошок за їх назвою (рисунок 4.6). Також є фільтр для відображення тільки публічних дошок або створених самим користувачем. Можна застосувати сортування дошок за датою оновлення, створення або за назвою.

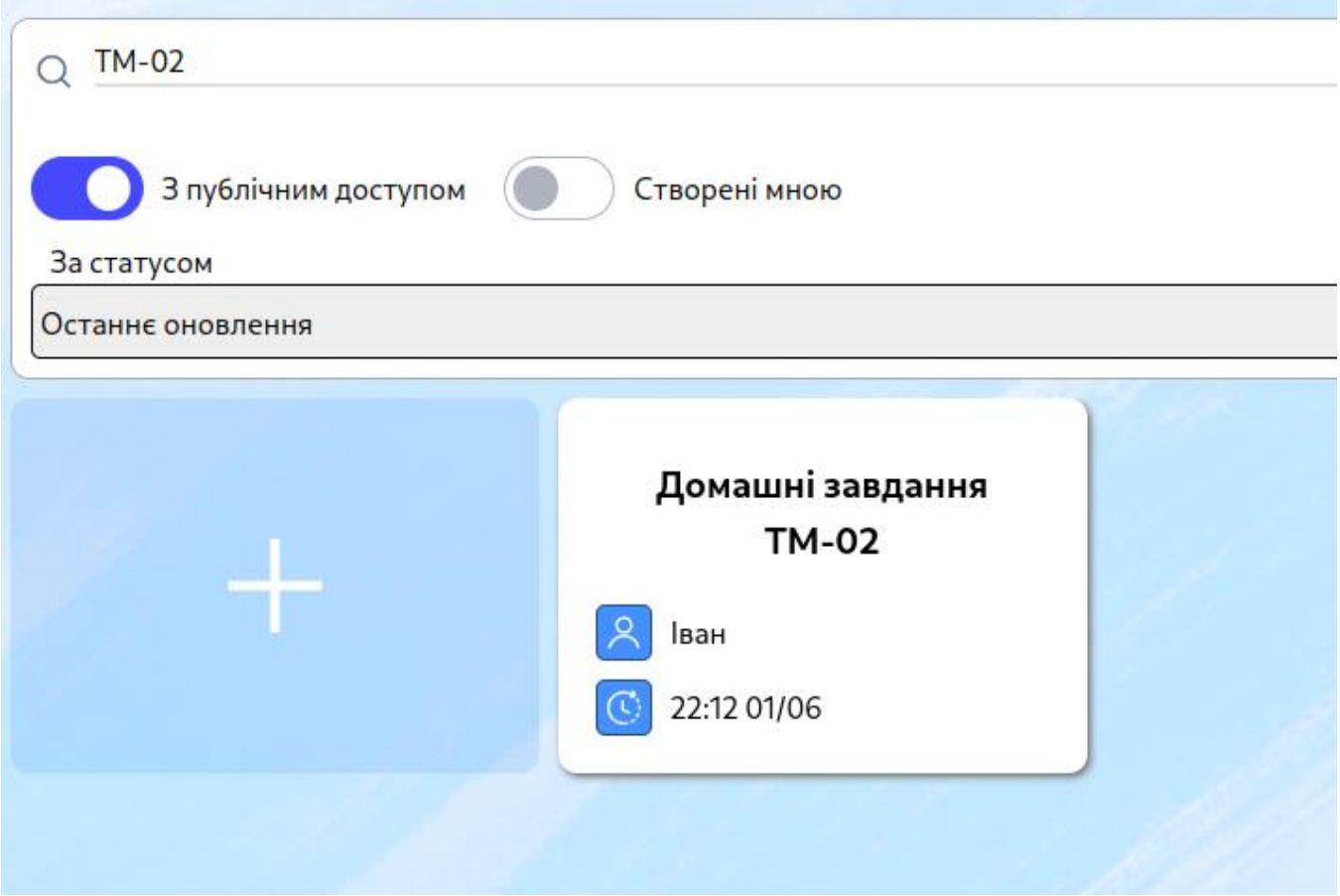

Рисунок 4.6 – Пошук дошки та відкритим публічним доступом

Сторінка конкретної дошки складається з її назви, опису та безпосередньо канбан-дошки (рисунок 4.7). Назву та опис можна редагувати, просто ввівши новий текст у відповідні поля.

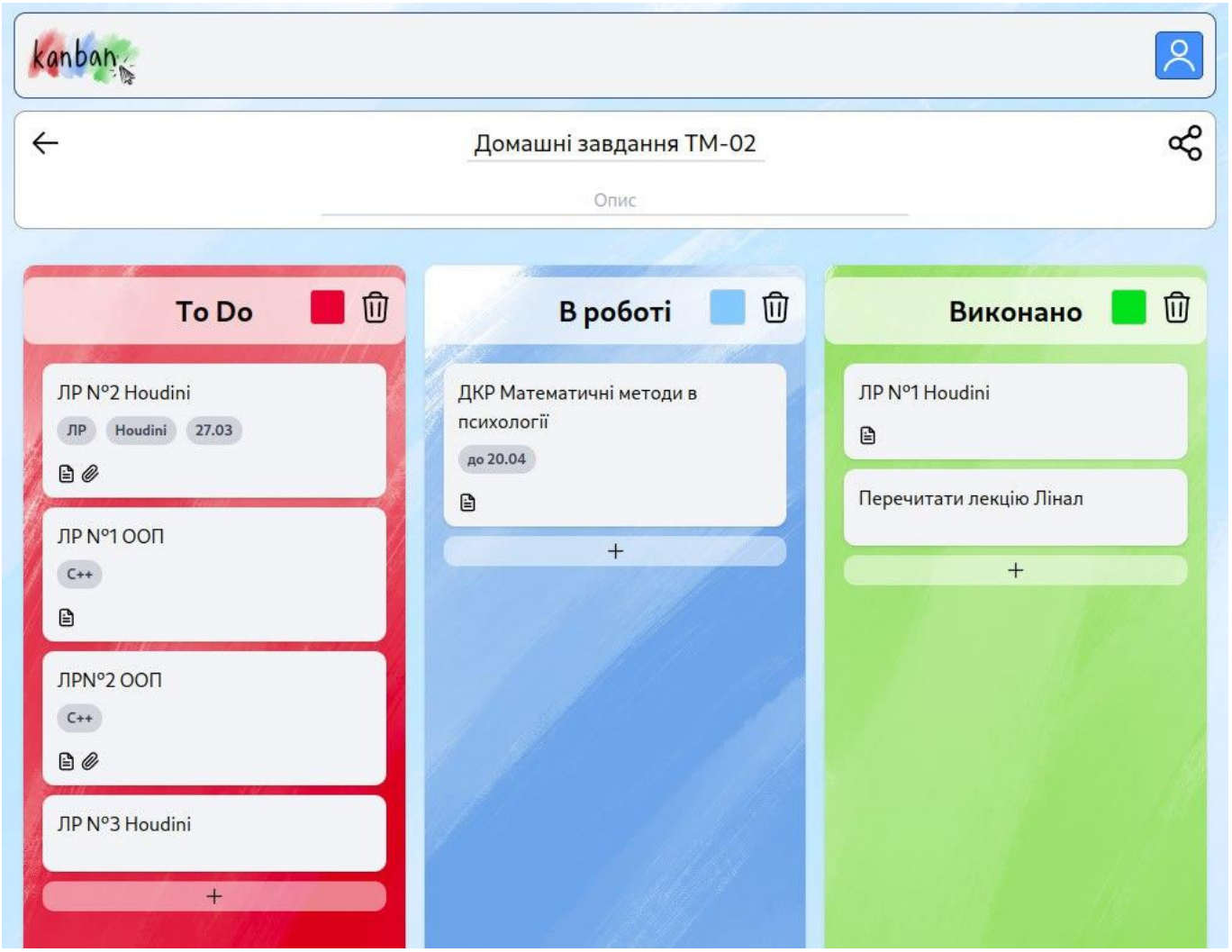

Рисунок 4.7 – Сторінка дошки (Vue-компонент KanbanBoardPage)

У кожній колонці є кнопка “+” для додавання нової картки до цього статусу. При натисканні на неї відкривається модальне вікно для введення даних картки (рисунок 4.8). Тут можна змінити назву картки, її опис, додати теги, прикріпити файли або видалити картку. Теги відображаються списком з хрестиком для видалення кожного з них. Для додавання нового тегу потрібно ввести його текст і натиснути Enter або кнопку “+”. Файли показуються списком, де вказана їх назва. Клік на назві файлу починає його завантаження. Для видалення файлу є хрестик поруч з ним. Щоб прикріпити новий файл, його можна перетягнути у спеціальну область або вибрати через діалогове вікно.

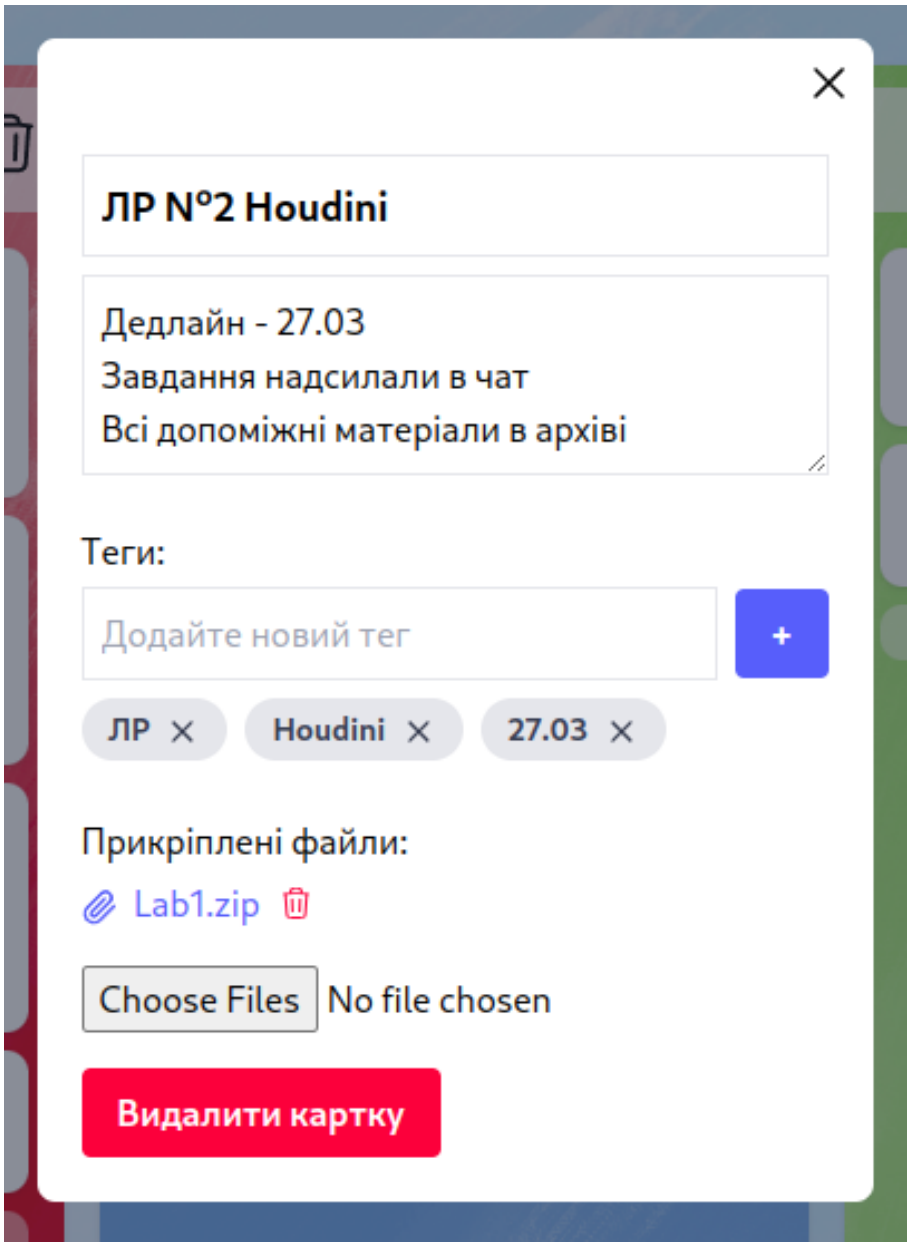

Рисунок 4.8 – Модальне вікно для створення нової картки

При кліку на вже створену картку відкривається модальне вікно з детальною інформацією про неї та можливістю редагування.

Канбан-дошка містить колонки, які відповідають статусам. Горизонтально розташовані колонки, а всередині них – картки із завданнями, розміщені вертикально. Картки можна перетягувати між колонками за допомогою технології drag-and-drop, змінюючи їх статус (рисунок 4.9). Порядок карток всередині колонки також можна змінювати перетягуванням.

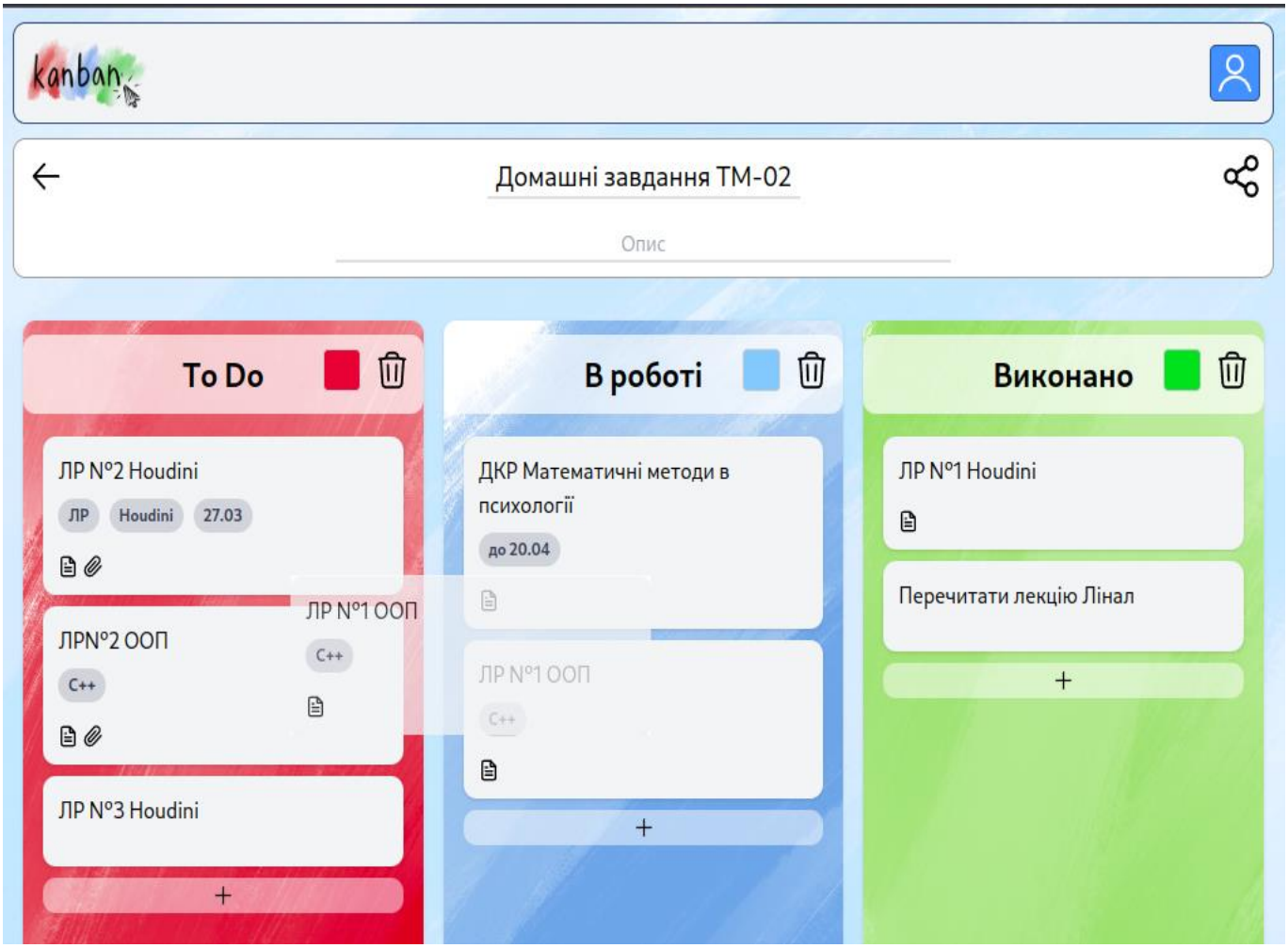

Рисунок 4.9 – Перетягування картки між статусами

У шапці сайту є іконка користувача. При кліку на неї відкривається модальне вікно з налаштуваннями профілю (рисунок 4.10), в якому можна виконати наступні дії:

– змінити своє ім'я, email, пароль (потрібно ввести поточний та новий паролі);

– налаштувати список статусів, які будуть доступні для нових дошок за замовчуванням (для кожного статусу вказується його назва та колір);

– видаляти та додавати нові статуси.

Після внесення всіх змін потрібно натиснути кнопку “Зберегти” для їх збереження.

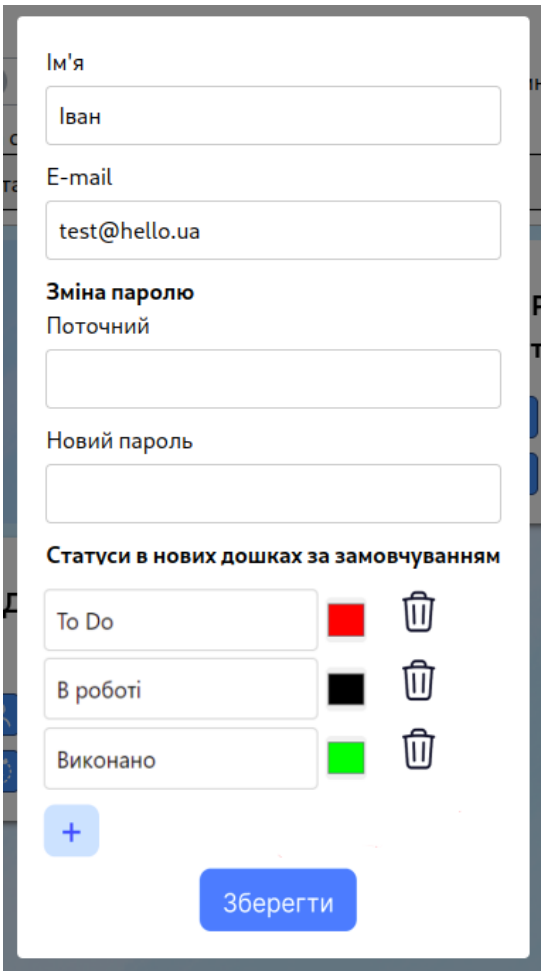

Рисунок 4.10 – Модальне вікно з налаштуваннями профілю користувача та зміною статусів

Для кожного статусу можна змінити його назву та колір фону. Для цього потрібно просто ввести нову назву або вибрати колір (рисунок 4.11). Зміни автоматично збережуться і застосуються до поточної дошки.

Рисунок 4.11 – Зміна кольору статусу

Також є можливість видалити статус з дошки за допомогою кнопки з іконкою корзини (рисунок 4.12). Ці налаштування статусів стосуються тільки даної конкретної дошки і не впливають на інші.

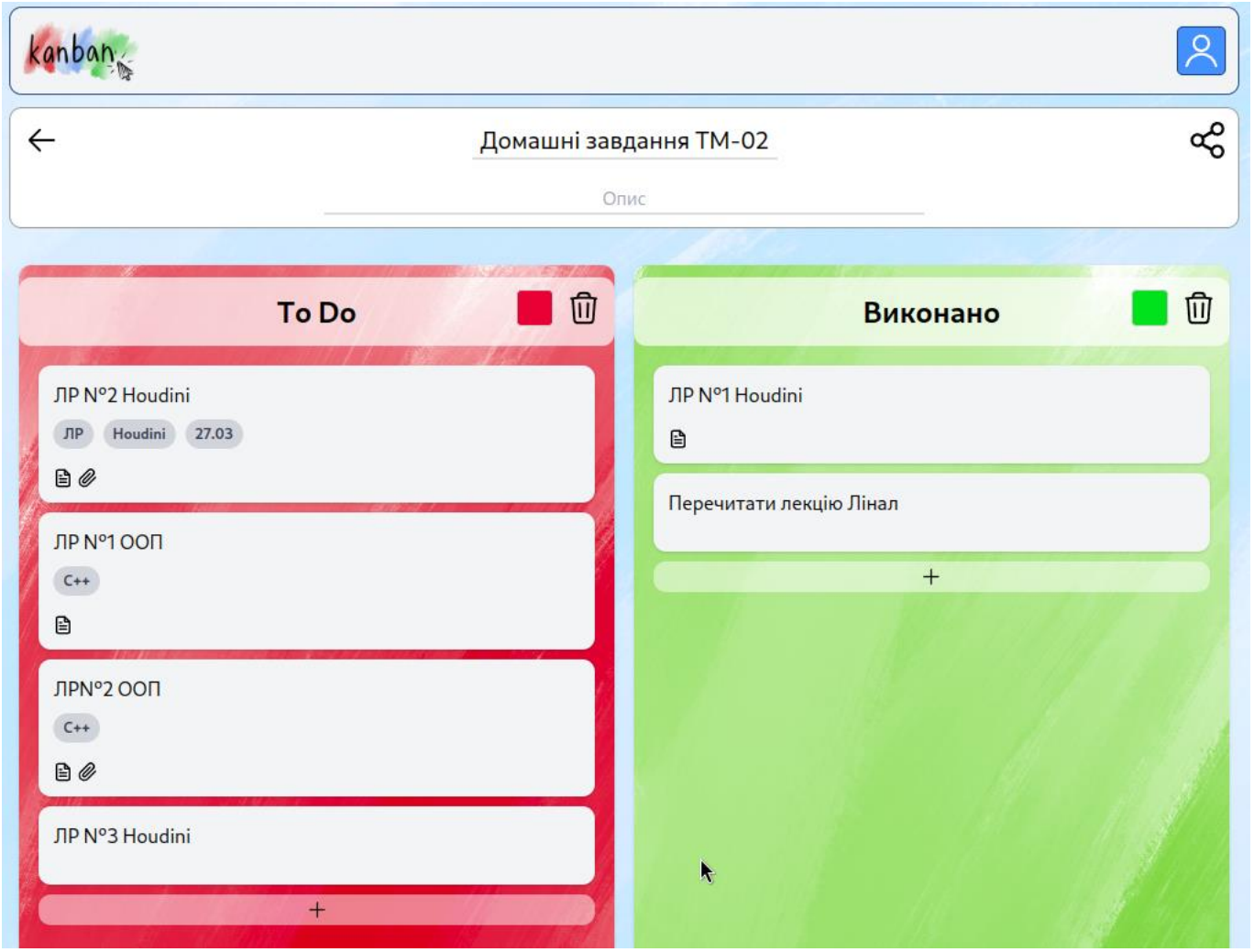

Рисунок 4.12 – Видалення статусу

Якщо користувач є власником дошки, то на сторінці цієї дошки показується іконка “поділитися”. Клік на неї відкриває модальне вікно для керування доступом до дошки. Тут можна увімкнути або вимкнути публічний доступ до дошки через посилання (рисунок 4.13).

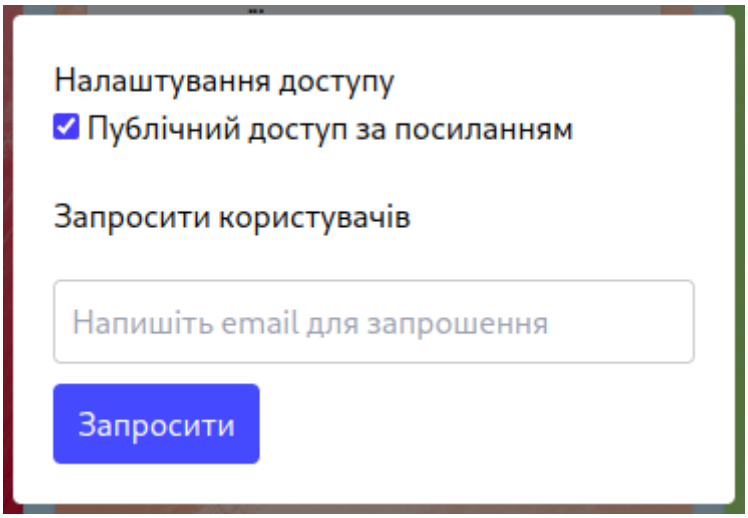

Рисунок 4.13 – Модальне вікно для керування доступом до дошки

Нижче показується список користувачів, які мають доступ до дошки. Для додавання нового користувача потрібно ввести його email та натиснути кнопку “Запросити” (рисунок 4.14).

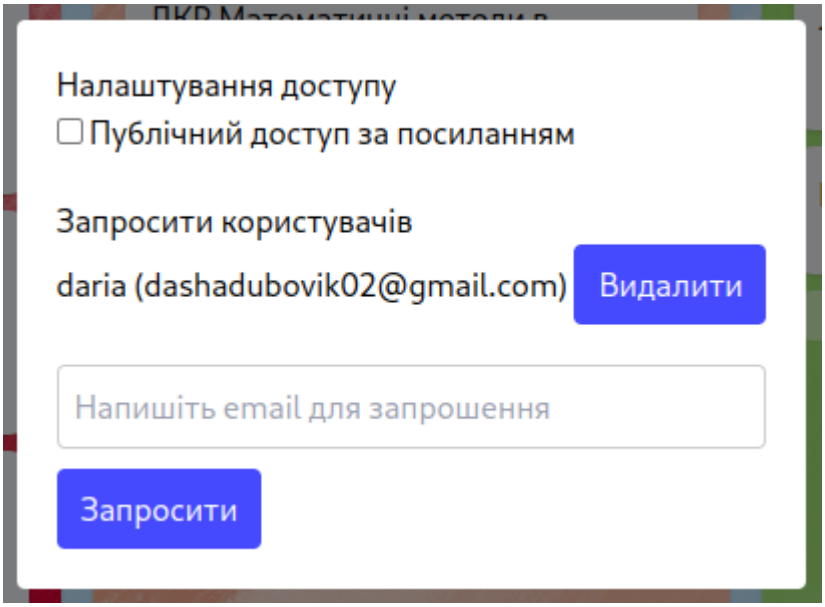

Рисунок 4.14 – Запрошення користувача

Для видалення доступу користувача є кнопка “Видалити” напроти кожного з них у списку.

Вище описані основні функції розроблюваного продукту. Для більш ефективного планування користувачам також краще притримуватись деяких правил.

1. Візуалізуйте процес. Створіть Kanban дошку з колонками “Запланована робота”, “Виконується” та “Готово”, щоб наочно бачити всі поточні та майбутні завдання.

2. Обмежте кількість незавершених завдань. Встановіть ліміт, наприклад, не більше 3 завдань одночасно у колонці “Виконується”. Це допоможе зосередитися й уникнути розпорошення уваги.

3. Визначте чіткі правила. Встановіть для себе зрозумілі правила використання Kanban дошки, щоб діяти послідовно і не відхилятися від обраної системи.

4. Постійно вдосконалюйте процес. Kanban – гнучка система, тож періодично переглядайте та оптимізуйте власний робочий процес відповідно до накопичених знань та даних.

5. Візуалізуйте пріоритети завдань за допомогою кольорів, міток чи порядку карток на дошці. Це допоможе зосередитися на найважливіших справах.

6. Вказуйте очікувану тривалість виконання завдання на картках. Це сприятиме кращому плануванню часу та виявленню затримок.

Дотримуючись цих рекомендацій, ви зможете ефективно застосовувати Kanban дошку для візуального планування, контролю та безперервної оптимізації особистих щоденних справ, максимізуючи продуктивність.

ВИСНОВОК

1. У ході виконання роботи, було проведено аналіз таких платформ, як: Trello, Asana та Notion. Розроблюваний програмний продукт має на меті надати користувачам альтернативне рішення з унікальними функціями та персоналізованими налаштуваннями, при цьому залишаючи всі функції безкоштовними та відкриваючи доступ до вихідного коду проєкту. Це дозволить іншим людям доповнювати проект бажаними доповненнями та виправленнями, потреба в яких виникає у них як у реальних користувачів застосунку, а також даючи іншим розробникам приклад використання та взаємодії обраних технологій.

2. Загалом проект складається з трьох основних компонентів: модуль Flask, модуль Vue.js та база даних MongoDB. Кожен з цих модулів представлено у вигляді окремого Docker-контейнера, що забезпечує гнучкість та незалежність кожного компонента. Для налаштування зв'язку між контейнерами та їх об'єднання в одну мережу було створено файл конфігурації docker-compose.yml, який описує налаштування та взаємодію між різними частинами системи.

3. У рамках даної дипломної роботи була розроблена система організації щоденних справ. Створена система є веб-застосунком, який дозволяє користувачам ефективно планувати, відстежувати та керувати своїми завданнями за допомогою візуальної дошки Kanban.

4. Під час тестування програмного продукту було підтверджено зручність та ефективність у плануванні. Користувачі високо оцінили інтуїтивний інтерфейс, швидкодію та можливість персоналізації застосунку відповідно до власних потреб.

Розроблена система може бути корисною для широкого кола користувачів, включаючи студентів, фрілансерів, малі та середні підприємства, менеджерів, а також людей з активним способом життя.

У майбутньому система може бути розширена та вдосконалена шляхом додавання додаткових функцій, таких як інтеграція з календарями, покращена аналітика та звітність, а також підтримка командної роботи .

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Anderson D.J. Kanban: Successful Evolutionary Change for Your Technology Business. Sequim , Washington : Blue hole press, 2010. 261 p.

2. Productivity Management Software Market Size Report, 2030. Market Research Reports & Consulting | Grand View Research, Inc: website. URL: https://www.grandviewresearch.com/industry-analysis/productivity-management-software-market (date of access: 11.05.2024).

3. Lavelle J. Gartner CFO Survey Reveals 74% Intend to Shift Some Employees to Remote Work Permanently. Gartner : website. URL: https://www.gartner.com/en/newsroom/press-releases/2020-04-03-gartner-cfo-surey-reveals-74-percent-of-organizations-to-shift-some-employees-to-remote-work-permanently2 (date of access: 08.05.2024).

4. Конспект лекцій з дисципліни «Архітектура та проектування програмного забезпечення» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Інженерія програмного забезпечення» із спеціальності 121 / Укл. В.В.Завгородній, К.М.Ялова. Кам’янське: ДДТУ, 2019. 144 с.

5. Посібник: знайомство з React – React. React – JavaScript-бібліотека для створення користувацьких інтерфейсів : website. URL: https://uk.legacy.reactjs.org/tutorial/tutorial.html#what-is-react (дата звернення: 08.05.2024).

6. Angular. Home • Angular : website. URL: https://angular.dev/overview (date of access: 10.05.2024).

7. Що таке Node.js? Основи серверної розробки на JavaScript. DevZone : website. URL: https://devzone.org.ua/post/shcho-take-nodejs-osnovy-servernoyi-rozrobky-na-javascript (дата звернення: 09.05.2024).

8. Patni S. API Design and Modeling. Pro RESTful APIs. Berkeley, CA, 2017. P. 11–31.

9. Ramo A. C., Diaz R. G., Tsaregorodtsev A. DIRAC RESTful API. Journal of Physics: Conference Series. 2012. Vol. 396, no. 5. P. 052019.

10. What Is Three-Tier Architecture? | IBM. IBM in Deutschland, Österreich und der Schweiz : website. URL: https://www.ibm.com/topics/three-tier-architecture (date of access: 11.05.2024).

11. Kaluža M., Vukelić B. Comparison of front-end frameworks for web applications development. Zbornik Veleučilišta u Rijeci. 2018. Vol. 6, no. 1. P. 261–282.

12. Roth R. User Interface and User Experience (UI/UX) Design. Geographic Information Science & Technology Body of Knowledge. 2017. Vol. 2017, Q2.

13. 7 fundamental UX design principles all designers should know - UX Design Institute. UX Design Institute: website. URL: https://www.uxdesigninstitute.com/blog/ux-design-principles/ (date of access: 08.05.2024).

14. What is FIGMA: Advantages and Disadvantages. FITA Academy: website. URL: https://www.fita.in/what-is-figma-advantages-and-disadvantages/ (date of access: 07.05.2024).

15. Alkasassbeh M. Handbook of Computer Networks and Cyber Security: Springer International Publishing, 2020. 30 p.

16. Yadav A. K., Garg M. L., Ritika. Docker Containers Versus Virtual Machine-Based Virtualization. Advances in Intelligent Systems and Computing. Singapore, 2018. P. 141–150.

17. Boost your productivity with kanbans. Read how. Firmbee: website. URL: https://firmbee.com/the-kanban-method (date of access: 10.05.2024).

18. The Official Kanban Guide. Armory Way Ste 101, #188, Seattle, Mauvius: Group Inc, 2022. 20p.

19. Top 20 Kanban Software To Level Up Your Project Management. Hive : website. URL: https://hive.com/blog/kanban-software (date of access: 08.05.2024).

20. How to Point a Domain Name from Namecheap to Contabo VPS : website. UgacompURL: https://www.ugacomp.com/how-to-point-a-domain-name-from-namecheap-to-contabo-vps/ (date of access: 12.05.2024).

Дотаток А

Реалізація логіки редагування та перетягування карток на Kanban дошці

Програмні засоби:

– мова програмування – Python;

– API-фреймворк – Flask;

– JavaScript фреймворк – Vue.js;

– MongoDB драйвер – PyMongo.

```
@boards_bp.route('/<string:board_id>/cards', methods=['PATCH'])
@auth_required
def update_board_cards(board_id):
    data = request.json

    if not data:
        return json_response({'message': 'No data provided'}, 400)

    db = get_db()

    if ObjectId.is_valid(board_id):
        board_id = ObjectId(board_id)
    else:
        return json_response({'message': 'Invalid board ID'}, 400)

    board = db.Boards.find_one({'_id': board_id, '$or': [{'creatorID': g.user['_id']}, {'invited': g.user['_id']}, {'publicAccess': True}]})

    if not board:
        return json_response({'message': 'Board not found'}, 404)
```

```
action = data.get('action')

if not action or action not in ['add', 'update', 'delete']:
    return json_response({'message': 'Missing or invalid action field'}, 400)

now = datetime.datetime.now()

card = data.get('card')
if not card:
    return json_response({'message': 'Missing card data'}, 400)

resp = {}
if action == 'add':
    card = {
        'boardID': board_id,
        'status': card.get('status', board['statuses'][0]['name']),
        'title': card.get('title', ''),
        'description': card.get('description', ''),
        'tags': card.get('tags', []),
        'order': card.get('order', 0)
    }
    result = db.Cards.insert_one(card)

    resp['cardId'] = result.inserted_id
elif action == 'update':
    card_id = card.get('id')
    if not card_id:
        return json_response({'message': 'Missing card ID'}, 400)

    update_data = {}
```

```
        if 'status' in card:
            update_data['status'] = card['status']
        if 'title' in card:
            update_data['title'] = card['title']
        if 'description' in card:
            update_data['description'] = card['description']
        if 'tags' in card:
            update_data['tags'] = card['tags']

        if update_data:
            db.Cards.update_one({'_id': ObjectId(card_id)}, {'$set': update_data})
    elif action == 'delete':
        card_id = card.get('id')
        if not card_id or not ObjectId.is_valid(card_id):
            return json_response({'message': 'Invalid card ID'}, 400)

        db.Cards.delete_one({'_id': ObjectId(card_id)})
        db.Uploads.delete_many({'cardID': ObjectId(card_id)})

    db.Boards.update_one({'_id': board_id}, {'$set': {'updatedAt': now}})

    return json_response(resp)
```

```
<template>
    <div class="kanban-board">
        <div class="columns">
            <KanbanColumn
                class="column"
                v-for="status in statuses"
```

```
            :boardId="boardId"
            :key="status.name"
            :status="status"
            :cards="filteredCards(status.name)"
            @add-card="addNewCard(status.name)"
            @move-card="moveCard"
            @reorder-card="reorderCard"
            @open-details="openCardEdit"
            @update-status="updateStatus"
            @delete-status="deleteStatus"
          />
        </div>
        <!-- ... -->
    </div>
</template>

<script>
// ...
async moveCard(cardId, newStatus) {
  const card = this.cards.find((card) => card.id === cardId);
  if (card) {
    const oldStatus = card.status;
    const oldOrder = card.order;

    this.cards = this.cards.filter((card) => card.id !== cardId);

    card.status = newStatus;
    card.order = this.getMaxOrder(newStatus) + 1;
    this.cards.push(card);
```

```
    this.updateCardOrders(oldStatus, oldOrder);

    await this.updateCardOnBoard({ boardId: this.boardId, card: card });
  }
},
reorderCard(cardId, newOrder) {
    const card = this.cards.find((card) => card.id === cardId);
    if (card) {
        const oldOrder = card.order;
        card.order = newOrder;
        this.updateCardOrders(card.status, oldOrder, newOrder);
    }
},
updateCardOrders(status, oldOrder = null, newOrder = null) {
    const cards = this.filteredCards(status);
    cards.forEach((card, index) => {
        if (oldOrder !== null && newOrder !== null) {
            if (card.order > oldOrder && card.order <= newOrder) {
                card.order--;
            } else if (card.order < oldOrder && card.order >= newOrder) {
                card.order++;
            }
        }
        card.order = index;
    });
}
// ...
</script>
```

Дотаток Б

Реалізація 3-рівневої архітектури застосунку у файлі конфігурації Docker Compose

Програмні засоби:

– мова розмітки – YAML;

– утиліта для управління контейнерами – Docker Compose

```
version: '3'

services:
  frontend:
    build: ./frontend
    ports:
      - 80:80
    depends_on:
      - backend
    networks:
      - app-network

  backend:
    build: ./backend
    expose:
      - 5000
    environment:
      - MONGO_URI=mongodb://db:27017
    depends_on:
      - db
    networks:
      - app-network
```

```
  db:
    image: mongo:4.4
    volumes:
      - mongodb_data:/data/db
    networks:
      - app-network

volumes:
  mongodb_data:

networks:
  app-network:
```