МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри_____ Є. О. Давиденко

підпис

«___» _____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Вебзастосунок багатокористувацького ведення нотаток

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 409.22010917

Здобувач _____В. С. Олексієнко

підпис

«__» _____ 2024 р.

Керівник канд. техн. наук, доцент _____ Г. В. Горбань

підпис

«__» _____ 2024 р.

Консультант канд. техн. наук, доцент _____ А. О. Алексєєва

підпис

«__» _____ 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. кафедри _____ Є. О. Давиденко

«_____» _____ 2024 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи бакалавра

Видано здобувачу групи 409 факультету комп'ютерних наук

_____Олексієнко Володимир Сергійович_____

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи

Вебзастосунок багатокористувацького ведення нотаток

_____

_____

Затверджена наказом по ЧНУ від «22» _____грудня_____ 2023 р. № ___269_____

2. Строк представлення кваліфікаційної роботи «____» _____ 20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Очікуваним результатом є вебзастосунок багатокористувацького ведення нотаток_____

_____

4. Перелік питань, що підлягають розробці:

- дослідження предметної області та аналіз існуючих аналогів;
- формування специфікації вимог до програмного забезпечення;
- визначення архітектури для проєктування програмного забезпечення;
- моделювання та проєктування програмного забезпечення;
- розробка програмного забезпечення;
- здійснення тестування роботи програмного забезпечення;

5. Перелік графічних матеріалів

__Презентація_____

6. Завдання до спеціальної частини

__Аналіз охорони праці на робочих місцях фахівців з інформаційних технологій__

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Алексєєва А. О.	Кафедра екології	Охорона праці

Керівник роботи _____канд. техн. наук, доцент Горбань Г. В._____
(посада, прізвище, ім’я, по батькові)

_____
(підпис)

Завдання прийнято до виконання

_____Олексієнко Володимир Сергійович_____
(прізвище, ім’я, по батькові)

_____
(підпис)

Дата видачі завдання «____» _____ 2024р.

КАЛЕНДАРНИЙ ПЛАН
виконання бакалаврської кваліфікаційної роботи

Тема: «Вебзастосунок багатокористувацького ведення нотаток»

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КРБ	15.01.2024	21.01.2024	Виконано
2	Огляд літератури за темою роботи	22.01.2024	28.01.2024	Виконано
3	Складання календарного плану КРБ	29.01.2024	04.02.2024	Виконано
4	Аналіз предметної області	05.02.2024	18.02.2024	Виконано
5	Розробка проєктних рішень	19.03.2024	21.03.2024	Виконано
6	Моделювання та конструювання ПЗ	22.03.2024	31.03.2024	Виконано
7	Кодування ПЗ, тестування та апробація розробленого ПЗ, аналіз результатів тестування	01.04.2024	22.05.2024	Виконано
8	Розробка спеціальної частини з охорони праці	23.06.2024	30.05.2024	Виконано
9	Оформлення КРБ та презентації	31.05.2024	03.05.2024	Виконано
10	Відгук керівника КРБ	04.06.2024	04.06.2024	Виконано
11	Попередній захист	05.06.2024	05.06.2024	Виконано
12	Рецензування	13.06.2024	13.06.2024	Виконано
13	Завершення оформлення КРБ та презентації	15.06.2024	15.06.2024	Виконано
14	Захист кваліфікаційної роботи	26.06.2024	26.06.2024	Виконано

Розробив здобувач Олексієнко В. С. _____
(прізвище, ім'я, по батькові) *(підпис)*
«24» січня 2024 р.

Керівник роботи канд. техн. наук, доцент Горбань Г. В. _____
(посада, прізвище, ім'я, по батькові) *(підпис)*
«24» січня 2024 р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«Вебзастосунок багатокористувацького ведення нотаток»

Здобувач 409 гр.: Олексієнко Володимир Сергійович

Керівник: канд. техн. наук, доцент Горбань Г. В.

Дана робота присвячена розробці програмного забезпечення для ведення нотаток з можливістю додавання інших користувачів.

Актуальність обраної теми відокремлюється у необхідності підвищення продуктивності та ефективності користувачів у роботі з нотатками через доступність й зручність вебзастосунків. Розвиток інформаційних технологій за останні роки створює можливості для створення інноваційних рішень у цій області, що має значення як для користувачів індивідуального рівня, так і для команд робочих груп та організаційного рівня.

Об’єктом кваліфікаційної роботи є процес спільного ведення нотаток через вебзастосунок.

Предметом кваліфікаційної роботи є функціональні можливості та інтерфейс вебзастосунку, спрямованого на оптимізацію управління нотатками та спільною роботою над ними.

Метою даної роботи є розробка вебзастосунку багатокористувацького ведення нотаток для поліпшення організації робочих процесів та підвищення продуктивності користувачів. Це досягається за допомогою розробки програмного забезпечення, яке забезпечує зручний доступ до нотаток та спільну роботу над ними для користувачів.

Перший розділ містить аналіз предметної області, аналіз готових рішень та специфікацію вимог.

Другий розділ містить спроєктовані діаграми варіантів використання, класів та послідовностей.

У третьому розділі представлено вибір технологій розробки, опис структури бази даних та розроблений мокап системи.

У четвертому розділі представлено реалізацію розробки вебзастосунку.

КРБ викладена на 81 сторінок, вона містить 4 розділи, 31 ілюстрацію, 23 джерел в переліку посилань

Ключові слова: вебзастосунок, багатокористувацьке ведення нотаток, спільна робота, організація інформації, інтерактивний інтерфейс.

ABSTRACT

to the qualification work of the bachelor

"Multi-user note-taking web application"

Student 409 gr.: Oleksiienko Volodymyr Serhiyovych

Supervisor: candidate technical of Sciences, associate professor G. V. Horban

This work is devoted to the development of software for keeping notes with the possibility of adding other users.

The relevance of the chosen topic is separated in the need to increase the productivity and efficiency of users in working with notes through the accessibility and convenience of web applications. The development of information technologies in recent years creates opportunities for creating innovative solutions in this area, which is important both for users at the individual level and for teams of work groups and organizational level.

The object of the qualification work is the process of joint note-taking through a web application.

The subject of the qualification work is the functionality and interface of a web application aimed at optimizing the management of notes and joint work on them.

The purpose of this work is to develop a multi-user note-taking web application to improve the organization of work processes and increase user productivity. This is achieved by developing software that makes it easy for users to access and collaborate on notes.

The first section contains the analysis of the subject area, the analysis of ready-made solutions and the specification of requirements.

The second section contains designed diagrams of use cases, classes and sequences.

The third section presents a selection of development technologies, a description of the database structure, and a developed mockup of the system.

The fourth chapter presents the implementation of web application development.

BQW is laid out on 81 pages, it contains 4 sections, 31 illustrations, 23 sources in the list of references

Keywords: web application, multi-user note-taking, collaboration, information organization, interactive interface.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 4
ВСТУП 5
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ 7
1.1 Аналіз предметної області 7
1.2 Аналіз готових рішень 8
1.2.1 Google Keep 8
1.2.2 Microsoft OneNote 9
1.2.3 Notion 11
1.3 Специфікація вимог 13
Висновки до розділу 1 15
2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ВЕДЕННЯ НОТАТОК 16
2.1 Розробка діаграми варіантів використання 16
2.2 Розробка діаграми класів 17
2.3 Розробка діаграми послідовностей дій системи 19
2.4 Розробка моделі пакетів даних системи 21
2.5 Розробка діаграм діяльності системи 22
2.6 Розробка діаграми станів та переходів 25
2.7 Розробка діаграм компонентів та розгортання системи 26
Висновки до розділу 2 29
3 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ ТА ОПИС БАЗИ ДАНИХ 30
3.1 Опис технологій розробки 30
3.2 Опис структури бази даних 37
3.3 Розробка мокапів системи 40
Висновки до розділу 3 43
4 РЕЗУЛЬТАТИ РОЗРОБКИ 44
4.1 Кодування застосунка 44
4.1.1 Створення сервісу для роботи з нотатками 44
4.1.2 Створення сервісів для роботи з користувачами 45
4.1.3 Створення сервісів для роботи з діями користувачами 46

4.1.4 Створення компоненту для роботи з списками нотаток 47
4.1.5 Створення компонентів для роботи з нотатками 49
4.2 Опис створеного інтерфейсу 50
Висновки до розділу 4 56
ВИСНОВКИ 58
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 59
ДОДАТОК А 61
ДОДАТОК Б 65
ДОДАТОК В 67
ДОДАТОК Г 69
ДОДАТОК Д 71
ДОДАТОК Е 72
ДОДАТОК Ж 74
ДОДАТОК И 75
ДОДАТОК К 78
ДОДАТОК Л 80

ПЕРЕЛІК СКОРОЧЕНЬ

CRUD	–	create read update delete
JVM	–	java virtual machine
MVC	–	model view controller
TS	–	typescript
JS	–	javascript
ECMAScript	–	european computer manufacturers association script
HTML	–	hypertext markup language
XML	–	extensible markup language
JSX	–	javascript XML
IDE	–	integrated development environment
DOM	–	document object model
ACID	–	atomicity, consistency, isolation, durability
CLI	–	command line interface
DTO	–	data transfer object
JSON	–	javascript object notation
JWT	–	JSON web token
HTTP	–	hypertext transfer protocol
HTTPS	–	hypertext transfer protocol secure
TCP	–	transmission control protocol
IP	–	internet protocol

ВСТУП

У сучасному цифровому світі, де інформація знаходиться в центрі уваги, ефективне управління нотатками стає ключовим аспектом продуктивності та організації робочого процесу. Розвиток технологій вебпрограмування та зростання потреб користувачів у зручних та ефективних інструментах для зберігання та обробки інформації створює підґрунтя для розробки вебзастосунків, спрямованих на краще управління нотатками.

Актуальність обраної теми відокремлюється у необхідності підвищення продуктивності та ефективності користувачів у роботі з нотатками через доступність й зручність вебзастосунків. Розвиток інформаційних технологій за останні роки створює можливості для створення інноваційних рішень у цій області, що має значення як для користувачів індивідуального рівня, так і для команд робочих груп та організаційного рівня.

Об’єктом кваліфікаційної роботи є процес спільного ведення нотаток через вебзастосунок.

Предметом кваліфікаційної роботи є функціональні можливості та інтерфейс вебзастосунку, спрямованого на оптимізацію управління нотатками та спільною роботою над ними.

Метою даної роботи є розробка вебзастосунку багатокористувацького ведення нотаток для поліпшення організації робочих процесів та підвищення продуктивності користувачів. Це досягається за допомогою розробки програмного забезпечення, яке забезпечує зручний доступ до нотаток та спільну роботу над ними для користувачів.

Для досягнення визначеної мети визначено такі завдання:

1) аналіз сучасного стану вебзастосунків для управління нотатками;
2) розробка архітектури та функціоналу вебзастосунку;
3) реалізація інтерфейсу користувача та основних функцій програми;
4) тестування та виправлення помилок;
5) оцінка ефективності та зручності використання вебзастосунку.

Аналіз сучасного стану вебзастосунків для управління нотатками підтверджує наявність прогалин у функціоналі та зручності використання таких програм. Існуючі рішення не завжди відповідають потребам користувачів у багатокористувацькій співпраці та зручному доступі до нотаток. Розробка нового вебзастосунку спрямована на заповнення цих прогалин та підвищення ефективності управління нотатками.

Основні проєктні рішення включають в себе розробку інтерфейсу, що враховує потреби користувачів у зручному та інтуїтивному доступі до функціоналу програми, а також реалізацію механізмів багатокористувацької співпраці та синхронізації нотаток.

Результати даної роботи можуть бути застосовані в будь-якій сфері, де важливе ефективне управління нотатками та спільна робота над ними, зокрема в освітній, науковій, бізнесовій та особистій сферах.

Таким чином, розробка вебзастосунку для багатокористувацького ведення нотаток відповідає сучасним вимогам продуктивності та організації робочого процесу, а також може мати широке застосування в різних галузях діяльності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Аналіз предметної області багатокористувацького ведення нотаток розкриває багато різних аспектів, які вимагають ретельного аналізу. Перед початком будь-якого проєктування вебзастосунку, слід ретельно дослідити цю область, враховуючи потреби користувачів, конкурентне середовище, технічні можливості та інші ключові аспекти.

Однією з ключових складових вебзастосунку багатокористувацького ведення нотаток є розуміння потреб та вимог користувачів. Нотатки використовуються для збереження різноманітної інформації, починаючи від планів та задач і закінчуючи важливими думками та ідеями. Тому важливо провести аналіз цих потреб та визначити функціональність, яка найбільш важлива для користувачів. Це може включати можливість створювати, редагувати та видаляти нотатки, організацію їх у категорії та мітки, а також спільний доступ до них для користувачів.

Крім того, необхідно дослідити конкурентне середовище. На сьогоднішній день існує багато вебзастосунків та мобільних додатків для ведення нотаток, і важливо вивчити їх функціональність та особливості. Це дозволить зрозуміти переваги та недоліки існуючих рішень та визначити те, що може зробити вебзастосунок унікальним та конкурентоспроможним.

Технічні аспекти також відіграють важливу роль у процесі проєктування вебзастосунку. Вибір технологій розробки, архітектурних рішень та інфраструктури має вирішальне значення для успішної реалізації проєкту. Наприклад, для забезпечення масштабованості та ефективності може бути важливим написання серверної частини програмного забезпечення та обрання відповідного хмарного середовища для хостингу.

Також важливо врахувати аспекти безпеки та конфіденційності даних. Оскільки нотатки можуть містити особисту та конфіденційну інформацію,

система повинна забезпечувати надійний захист від несанкціонованого доступу та втрати даних.

1.2 Аналіз готових рішень

1.2.1 Google Keep

Розробник: Google LLC [1]

Архітектура: Web application, мобільні застосунки для Android та iOS

Мова реалізації: JavaScript (Frontend), Java (Backend)

Інтерфейс застосунку Google Keep представлено на рис. 1.1.

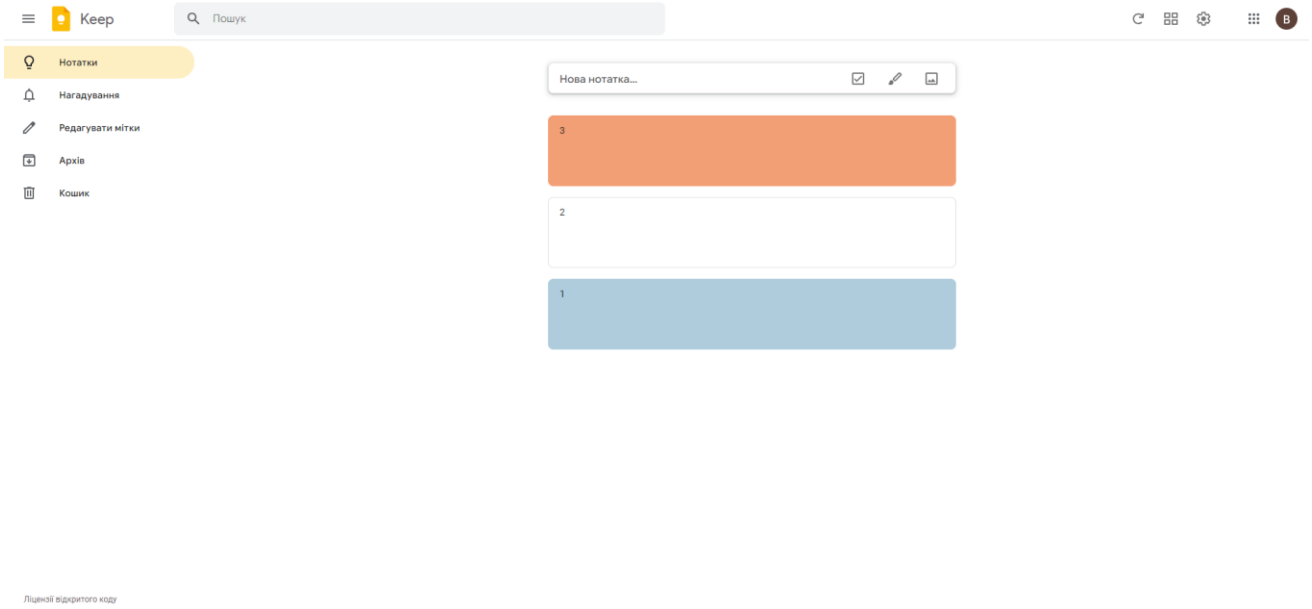

Рисунок 1.1 – Інтерфейс застосунку Google Keep

Функції та характеристики:

– збереження коротких нотаток, списків, фотографій, голосових нагадувань;

– можливість організації нотаток за мітками;

– синхронізація між різними пристроями через обліковий запис Google;

– пошук та фільтрація нотаток;

– доступ до нотаток в офлайн-режимі.

Переваги:

– простий та легкий спосіб для збереження нотаток, списків, та інших корисних матеріалів;

– інтеграція з екосистемою Google, що дозволяє користувачам миттєво синхронізувати свої нотатки з Gmail, Google Drive, та іншими сервісами;

– простий інтерфейс дозволяє швидко створювати нотатки, використовуючи різні формати, такі як тексти, зображення, або голосові нагадування;

– безкоштовний та доступний на різних платформах, включаючи вебверсію та мобільні додатки для Android і iOS.

Недоліки:

– обмежена функціональність порівняно з іншими подібними продуктами: відсутні розширені можливості організації нотаток, таких як створення папок або категорій, що може ускладнювати управління великим обсягом даних та особливо складно буде спільно використовувати цей застосунок групою людей.

1.2.2 Microsoft OneNote

Розробник: Microsoft Corporation [2]

Архітектура: Desktop application, Web application, мобільні застосунки для Android, iOS

Мова реалізації: JavaScript (Frontend), C# та .NET Framework (Backend)

Інтерфейс застосунку Microsoft OneNote представлено на рис. 1.2.

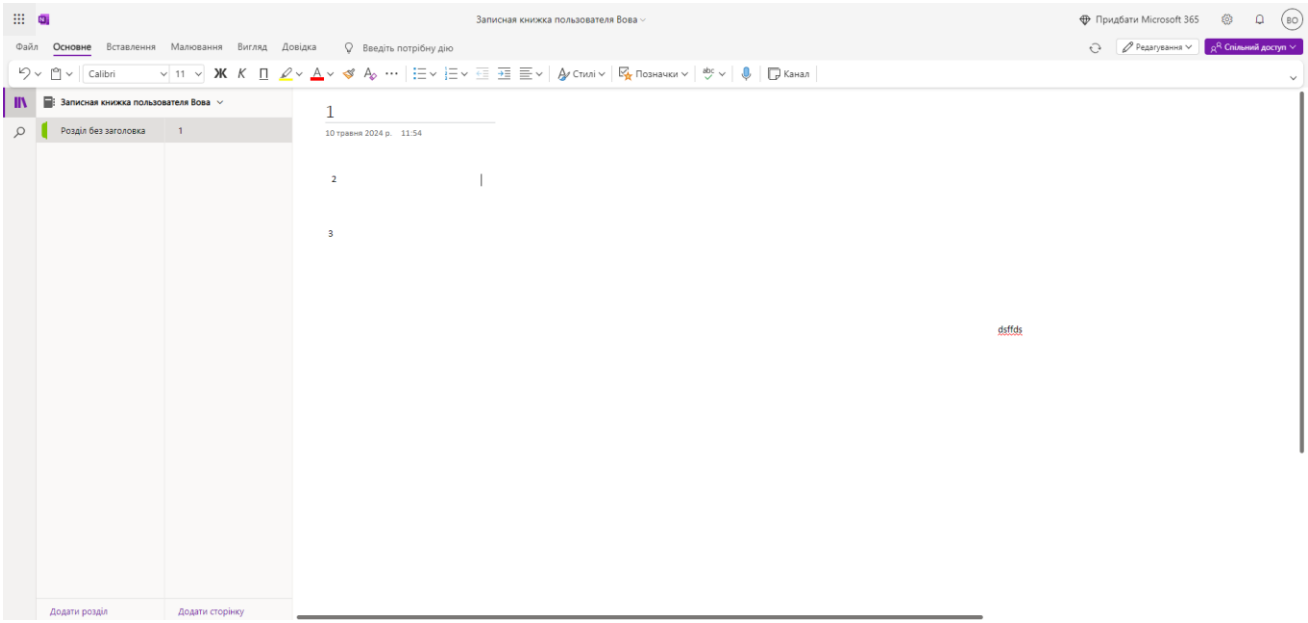

Рисунок 1.2 – Інтерфейс застосунку Microsoft OneNote

Функції та характеристики:

– створення структурованих нотаток у вигляді блокнотів, секцій, сторінок;
– підтримка текстових, графічних, аудіо та відео нотаток;
– синхронізація через обліковий запис Microsoft;
– колективна робота над нотатками у реальному часі;
– інтеграція з іншими продуктами Microsoft Office.

Переваги:

– багато функцій та можливостей;
– повна інтеграція з іншими продуктами Microsoft Office, що дозволяє користувачам легко обмінюватися нотатками та документами між різними програмами;
– можливість спільної роботи над нотатками у реальному часі, що робить його ідеальним інструментом для командної роботи та спільного навчання;
– підтримка різних платформах, включаючи Windows, macOS, Android та iOS, що забезпечує доступність для широкого кола користувачів.

Недоліки:

– складність використання для новачків через велику кількість функцій та можливостей, а також не завжди інтуїтивно зрозумілий інтерфейс;

– перенасичений інтерфейс, що також викликає труднощі в користуванні;

– для використання деяких функцій потребується наявності преміум аккаунта;

– дані всіх нотаток зберігаються в Microsoft OneDrive, що через його стандартну обмеженість в розмірі може викликати проблеми в збереженні нових нотаток та синхронізації з іншими пристроями користувача.

1.2.3 Notion

Розробник: Notion Labs Inc. [3]

Архітектура: Desktop application, Web application, мобільні застосунки для Android, iOS

Мова реалізації: JavaScript (Frontend), Node.js (Backend)

Інтерфейс застосунку Notion представлено на рис. 1.3.

Рисунок 1.3 – Інтерфейс застосунку Notion

Функції та характеристики:

– створення структурованих нотаток у вигляді сторінок та блоків;

– підтримка текстових, графічних, аудіо та відео нотаток;

– синхронізація через обліковий запис Notion;

– колективна робота над нотатками у реальному часі;

– інтеграція з іншими сервісами та застосунками, такими як Google Drive, Trello, Slack, GitHub та інші;

– можливість створення та управління завданнями, таблицями, канбан-дошками;

– підтримка шаблонів для різних типів нотаток і проєктів.

Переваги:

– універсальний інструмент для створення та організації нотаток, проєктів та робочих процесів;

– велика гнучкість та можливість налаштування під конкретні потреби користувача або команди;

– підтримка спільної роботи в реальному часі, що дозволяє ефективно працювати над спільними проєктами;

– інтеграція з іншими популярними сервісами;

– наявність шаблонів;

– підтримка різних платформ, включаючи Windows, macOS, Android та iOS, що забезпечує зручний доступ до нотаток з будь-якого пристрою.

Недоліки:

– складність використання для новачків через велику кількість функцій та можливостей;

– не завжди інтуїтивно зрозумілий інтерфейс;

– прив’язка нотаток, збережених в списку до автора, що обмежує організацію списків нотаток окремо для кожного користувача;

– досить суттєві обмеження безкоштовної версії на кількість створених блоків, що може бути проблемою для користувачів з великими обсягами даних;

– потреба в преміум-акаунті для повноцінного використання більшості можливостей застосунку.

1.3 Специфікація вимог

Призначення та межі проєкту

Призначення системи:

Призначенням системи, що розробляється, є поліпшення організації робочих процесів та підвищення продуктивності користувачів при веденні спільних нотаток.

Погодження, що ухвалені в програмній документації:

Погоджень не ухвалено.

Межі проєкту ПЗ:

Крайня дата завершення роботи над ПЗ – хх.06.2024р.

Загальний опис

Сфера застосування:

Вебзастосунок призначений для користувачів, які потребують спільного доступу до своїх нотаток, у будь-якій сфері діяльності.

Характеристики користувачів:

Користувачі будуть мати різний рівень досвіду з використання вебзастосунків. Вебзастосунок матиме аудиторію з різноманітним віковим спектром. Користувачі будуть використовувати застосунок з різних пристроїв.

Загальна структура і склад системи:

Вебзастосунок буде складатися з фронтенду (інтерфейсу користувача) та бекенду (логіки програми та бази даних).

Загальні обмеження:

Обмеження для роботи з ПЗ: доступ тільки для авторизованих користувачів, наявність доступу до мережі Інтернет та функціональні і технічні можливості вебзастосунку.

Функції системи

Функції системи багатокористувацького ведення нотаток:

- реєстрація та авторизація користувачів;
- створення, редагування та видалення текстових нотаток;

– групування нотаток в списки;

– зберігання нотаток в списку в певному порядку з можливістю його зміни відповідно до бажання користувача;

– відзначення важливих нотаток;

– багатокористувацький доступ до списків нотаток.

Вимоги до інформаційного забезпечення

Джерела і зміст вхідної інформації (даних):

Інформація про користувачів, списки нотаток та їх налаштування. Джерелом даних є користувачі.

Нормативно-довідкова інформація:

Вимоги відсутні.

Вимоги до способів організації, збереження та ведення інформації:

Для збереження інформації обрано базу даних з документно-орієнтованою моделю – MongoDB.

Вимоги до технічного забезпечення:

Основною вимогою до технічного забезпечення користувача є наявність пристрою з встановленим сучасним браузером.

Архітектура програмної системи:

Застосунок має триярусну архітектуру, що складається з клієнтської частини, серверної частини та бази даних.

Системне програмне забезпечення:

Будь-яка операційна система з встановленим Docker.

Мова і технологія розробки ПЗ:

Для розроблення бекенд частини застосунку має бути використано мову програмування Java та фреймворк Spring, а для фронтенд частини – Javascript та React відповідно.

Інтерфейс користувача:

Інтерфейс користувача повинен бути розроблений відповідно до дизайну зазначеного на мокапах для забезпечення зручності та ефективності роботи з вебзастосунком.

Апаратний інтерфейс:

Пристрій який використовується користувачем.

Висновки до розділу 1

В першому розділі кваліфікаційної роботи бакалавра проведено дослідження предметної області, що стосується вебзастосунків для багатокористувацького ведення нотаток. Виявлено, що такі системи є важливим інструментом для збереження інформації. Під час аналізу предметної області було визначено ключові аспекти, такі як потреби користувачів, конкурентне середовище та технічні можливості. В аналізі існуючих систем також виявлено вдалі рішення, які можна врахувати під час розробки власної платформи.

На підставі аналізу була сформульована постановка задачі для розробки власного вебзастосунку, що спрямований на розширення способів ведення нотатків. Серед функціональних вимог визначено створення, редагування та видалення нотаток, їх групування у списки з певним порядком зберігання, можливість відзначення важливих нотаток та багатокористувацький доступ до них. Такий підхід дозволить розробити вебзастосунок, який буде ефективним і зручним для користувачів, забезпечуючи їм широкий функціонал та можливості спільної роботи над нотатками.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ВЕДЕННЯ НОТАТОК

2.1 Розробка діаграми варіантів використання

Діаграма варіантів використання – це ілюстрація функціональних можливостей системи з погляду її користувачів, де демонструється, які дії можуть бути виконані користувачами та системою в реакції на конкретні події або запити.

Діаграма варіантів використання вебзастосунку, що проєктується, представлена на рис. 2.1.

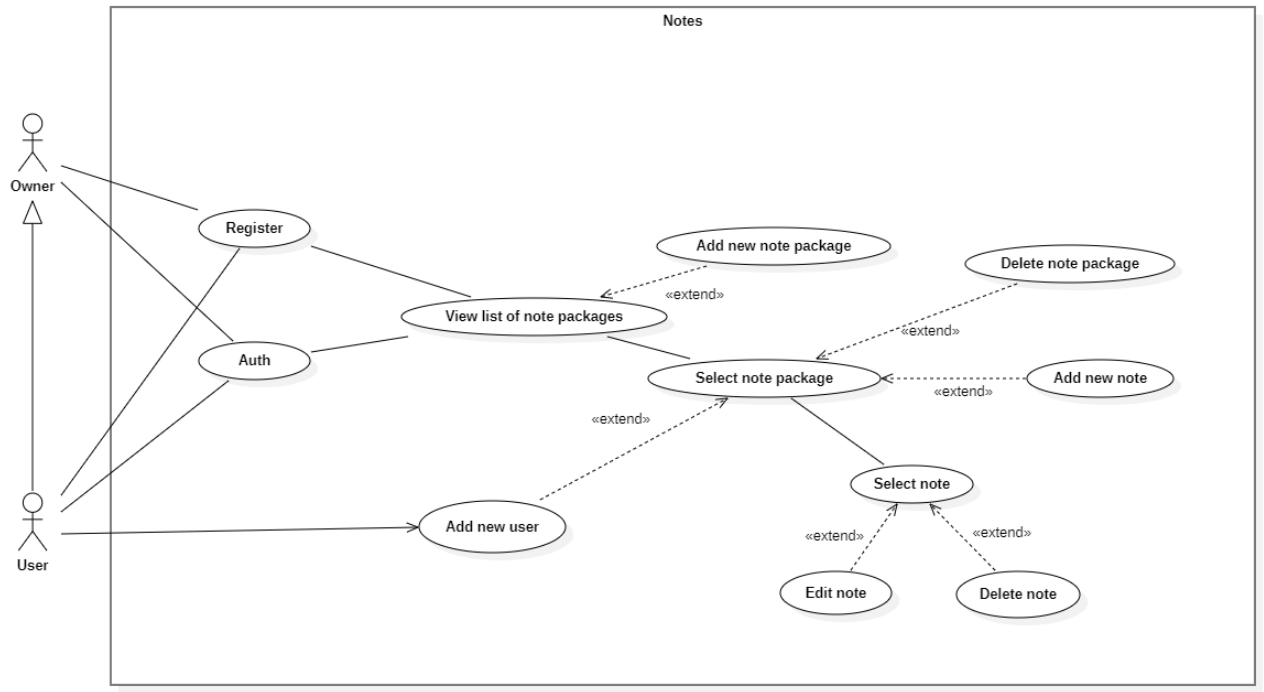

Рисунок 2.1 – Діаграма варіантів використання вебзастосунку багатокористувацького ведення нотаток

Основні варіанти використання, що зображені на діаграмі для вебзастосунку багатокористувацького ведення нотаток, включають реєстрацію та авторизацію користувача. Після авторизації користувачі можуть переглядати списки нотаток, а також створювати нові. Після вибору певного списку власник може видалити його, додавати нові нотатки до нього та додавати нових користувачів для цього списку нотаток. Після вибору конкретної нотатки, користувачі можуть її редагувати та видаляти.

2.2 Розробка діаграми класів

Діаграма класів – це вид діаграми в моделюванні програмного забезпечення, який показує структуру системи, її класи та взаємозв'язки між ними. Вона використовується для візуалізації структури програми, включаючи класи об'єктів, їх атрибути та методи, а також зв'язки між цими класами.

Початок діаграми класів з атрибутами та методами представлено на рис. 2.2.

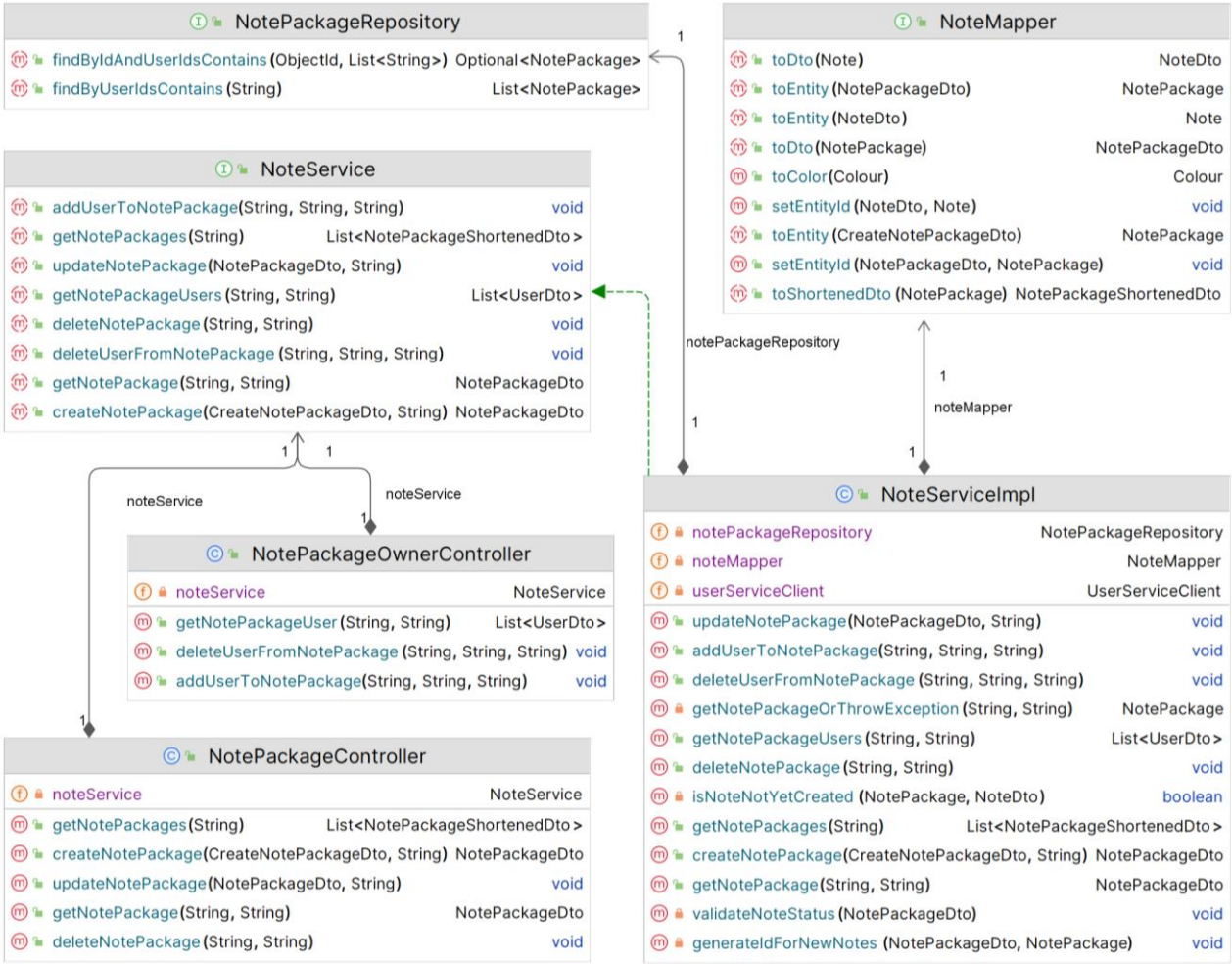

Рисунок 2.2 – Діаграма класів

Класи шару для роботи з мережею Інтернет:

– NotePackageOwnerController клас для оброблення запитів управління учасниками списку нотаток (додавання, видалення та отримання вже доданих);

– NotePackageController клас для оброблення запитів управління нотатками та списками нотаток (створення, видалення, оновлення, отримання всіх

списків користувача з короткими відомостями про них та отримання одного списку з усіма його нотатками).

Класи шару для роботи з базою даних:

– NotePackageRepository клас для виконання запитів до бази даних з методами для CRUD операцій, які реалізовані фреймворком, та власними методами для пошуку одного та декількох списків нотаток за певними критеріями.

Класи з основною логікою застосунка:

– інтерфейс NoteService та його реалізація, клас NoteServiceImpl, містять всю основну логіку застосунку для роботи з нотатками та користувачами, що володіють ними.

NoteMapper клас з методами для конвертацій об’єктів.

Продовження діаграми класів з атрибутами та методами представлено на рис. 2.3.

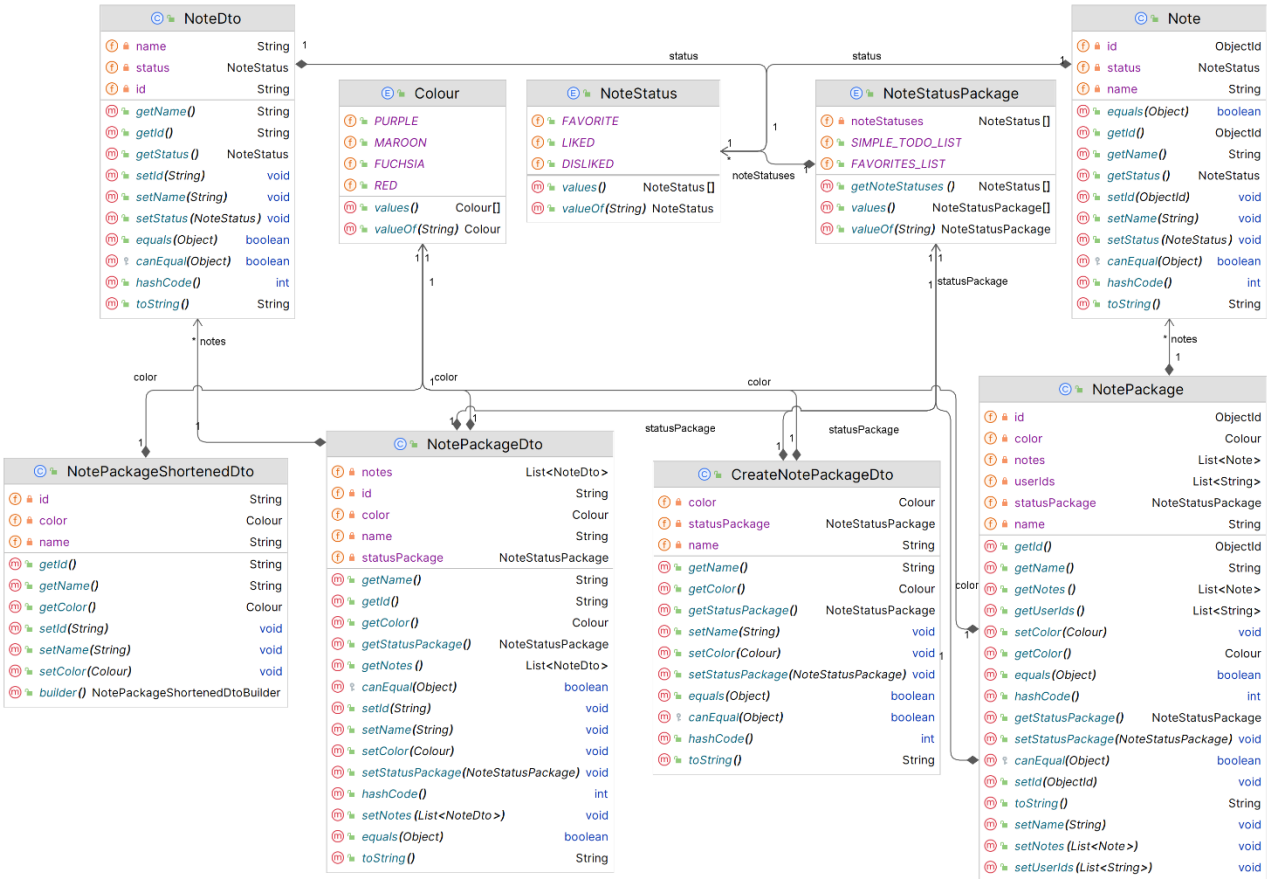

Рисунок 2.3 – Діаграма класів

Класи з константами:

– Colour – це enum, що містить основні кольори доступні користувачу;

– NoteStatus – це enum, що містить можливі статуси нотаток;

– NoteStatusPackage – це enum, що містить можливі пакети статусів для списків нотаток;

Класи entity:

– Note – клас, що відповідає сутності нотатки.

– NotePackage – клас, що відповідає сутності списку нотаток.

Класи dto:

– NoteDto – клас, що відповідає сутності нотаток.

– NotePackageDto – клас, що відповідає сутності списку нотаток.

– NotePackageShortenedDto – клас, що відповідає сутності списку нотаток з короткою інформацією про нього.

– CreateNotePackageDto – клас, що відповідає сутності запиту на створення списку нотаток.

2.3 Розробка діаграми послідовностей дій системи

Діаграма послідовності – це діаграма, яка показує послідовність обміну повідомленнями між об’єктами або класами в системі. Вона використовується для візуалізації взаємодії між об’єктами в певному процесі або сценарії.

Діаграма послідовності системи з відображенням процесу авторизації, створення списку нотаток та його заповненням представлена на рис. 2.4.

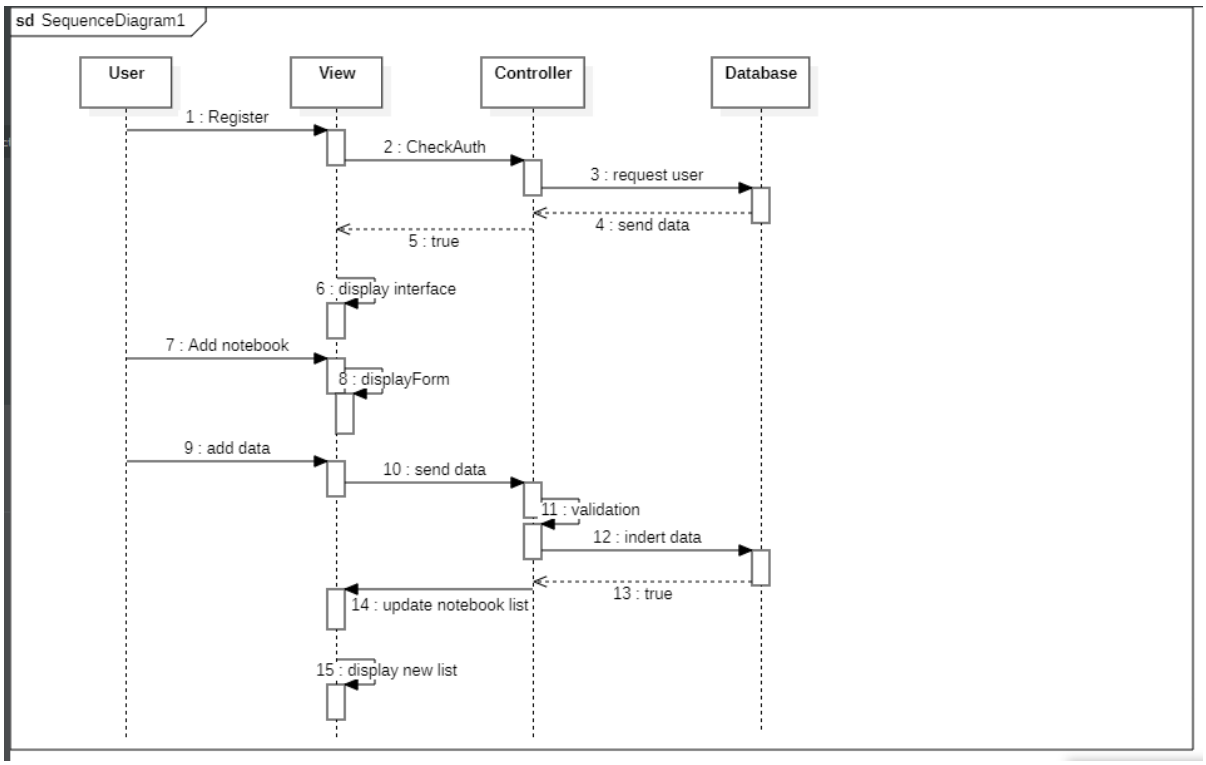

Рисунок 2.4 – Діаграма послідовності системи

Діаграма послідовності ілюструє процес реєстрації користувача та додавання нотаток у вебзастосунку для багатокористувацького ведення нотаток. Спочатку користувач реєструється, після чого надсилає запит до модуля представлення для перевірки аутентифікації. Модуль представлення передає запит контролеру, який звертається до бази даних для запиту інформації про користувача. База даних надсилає відповідні дані контролеру, який у свою чергу повідомляє модуль представлення про успішну аутентифікацію. Потім модуль представлення відображає інтерфейс користувача.

Коли користувач додає нову нотатку, модуль представлення відображає форму для введення даних. Після введення даних користувачем, ці дані надсилаються до модуля представлення, який передає їх контролеру для валідації. Контролер здійснює перевірку даних і передає їх до бази даних для збереження. База даних підтверджує успішне збереження даних, про що контролер інформує модуль представлення. Нарешті, модуль представлення оновлює список нотаток та відображає новий список користувачу.

2.4 Розробка моделі пакетів даних системи

Діаграма пакетів – це структурна діаграма в об’єктно-орієнтованому програмуванні, яка відображає взаємозв’язки між різними пакетами програмного забезпечення. Пакети представляють собою групу класів, інтерфейсів та інших об’єктів, які логічно пов’язані між собою.

У діаграмі пакетів показується, як пакети взаємодіють один з одним через залежності. Це може включати в себе залежності від інших пакетів, використання класів з інших пакетів та інші форми взаємодії. Діаграми пакетів допомагають розібратися в структурі програмного забезпечення, розуміти його організацію та сприяють підтримці його модульності.

Діаграму пакетів системи представлено на рис. 2.5.

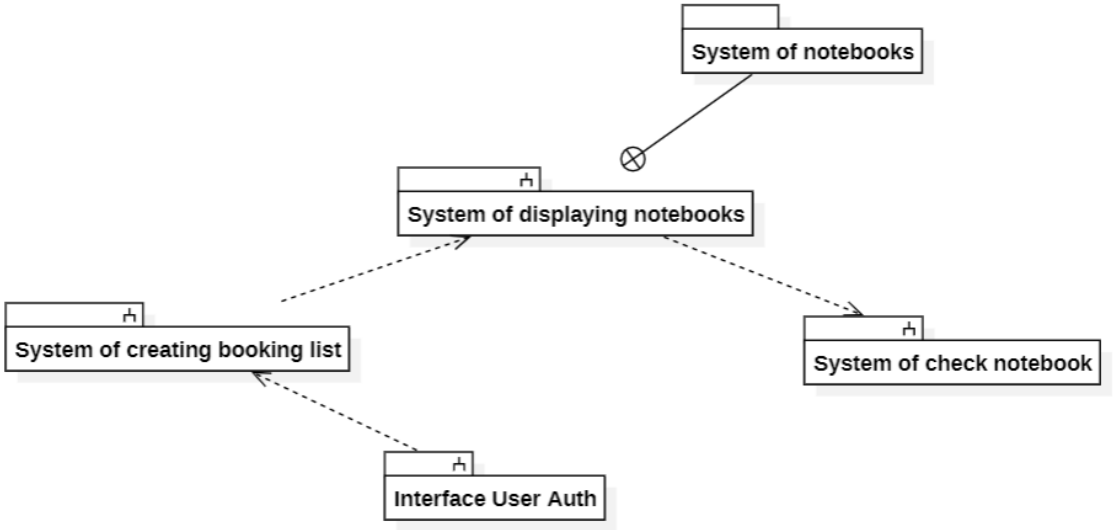

Рисунок 2.5 – Діаграма пакетів системи

Діаграма пакетів зображує структуру вебзастосунку для багатокористувацького ведення нотаток, розділеного на кілька взаємопов'язаних систем. «System of notebooks» служить основним сховищем і управляє нотатками. «System of displaying notebooks» є головним пакетом, який взаємодіє з іншими системами, використовуючи їх функціональні можливості. «System of check notebook» відповідає за перевірку нотаток. «System of creating booking list»

забезпечує створення списку нотаток. «Interface User Auth» надає функції аутентифікації користувачів.

2.5 Розробка діаграм діяльності системи

Діаграма діяльності – це графічний засіб для моделювання послідовності операцій або дій, які відбуваються в системі. Вона дозволяє візуалізувати роботу системи, виділяючи основні кроки або етапи виконання процесу. Діаграма діяльності системи використовується для аналізу, проєктування та розробки програмного забезпечення та бізнес-процесів.

Діаграма діяльності системи з відображенням процесу вибору списку нотаток або створення нового представлена на рис. 2.6.

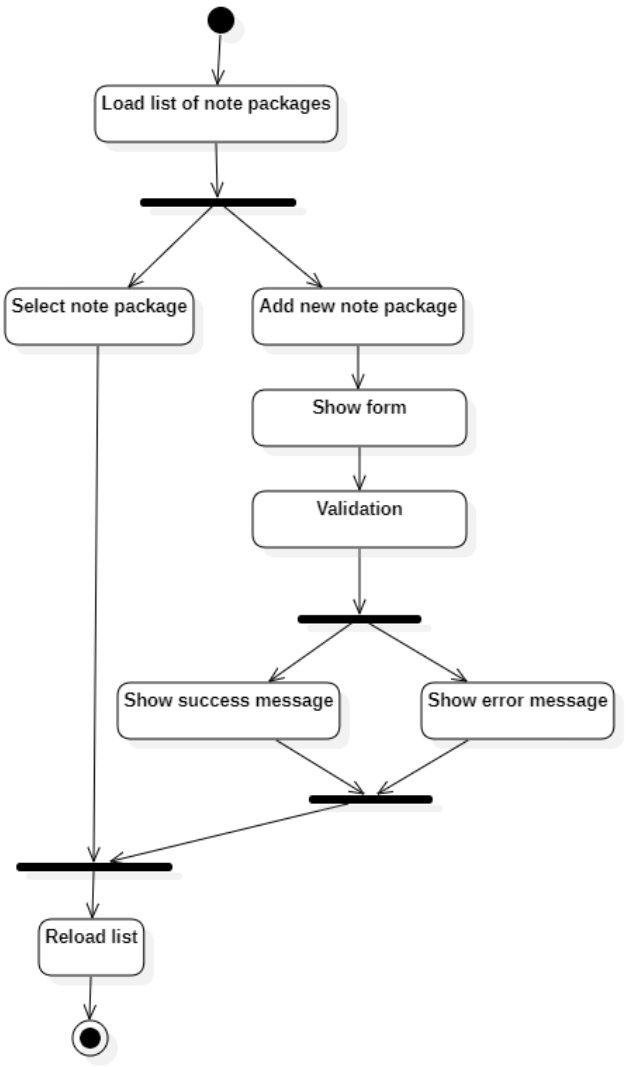

Рисунок 2.6 – Діаграма діяльності для процесу вибору списку нотаток або створення нового

Процес починається із завантаження списку пакетів нотаток. Далі користувач може вибрати один з двох варіантів: вибрати пакет нотаток або додати новий пакет. Якщо вибирається існуючий пакет нотаток, процес завершується.

Якщо користувач вирішує додати новий пакет, показується форма для введення даних. Введені дані проходять валідацію. У разі успішної валідації користувач бачить повідомлення про успіх, і список пакетів перезавантажується. Якщо валідація неуспішна, користувач бачить повідомлення про помилку, і процес завершується.

Діаграма діяльності системи з відображенням процесу редагування або видалення нотатки в списку представлена на рис. 2.7.

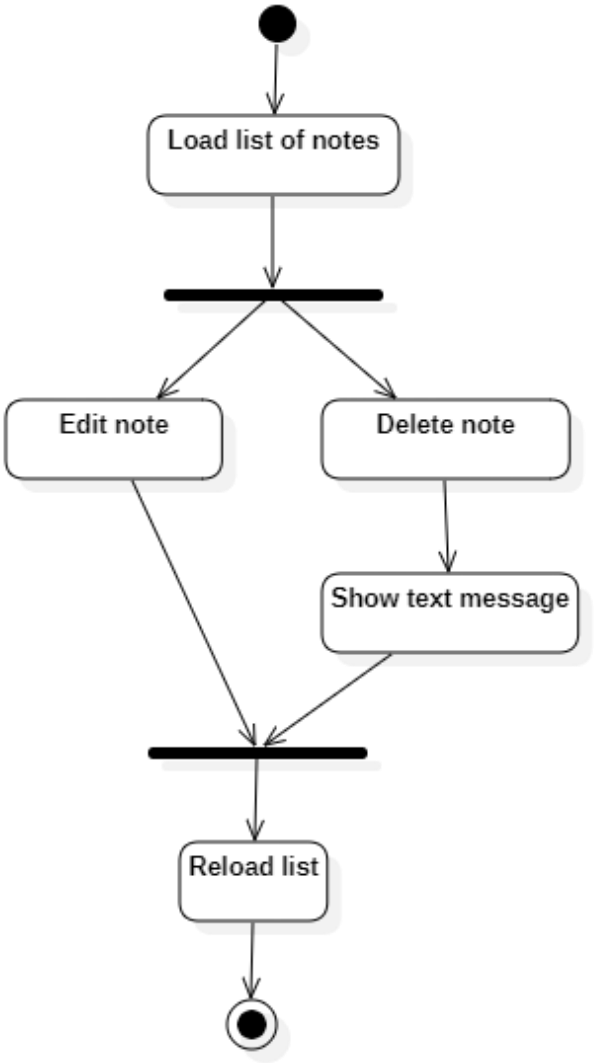

Рисунок 2.7 – Діаграма діяльності для процесу редагування або видалення нотатки в списку

Процес починається із завантаження списку нотаток. Потім користувач вибирає дію: редагування або видалення нотатки. Якщо вибрано редагування, виконується відповідна дія з редагування нотатки. Якщо вибрано видалення, нотатка видаляється, і користувачу показується повідомлення про успішне видалення. Після цього список нотаток перезавантажується для відображення актуального стану і процес завершується.

Діаграма діяльності системи з відображенням процесу додавання нового користувача до власників списку нотатків представлена на рис. 2.8.

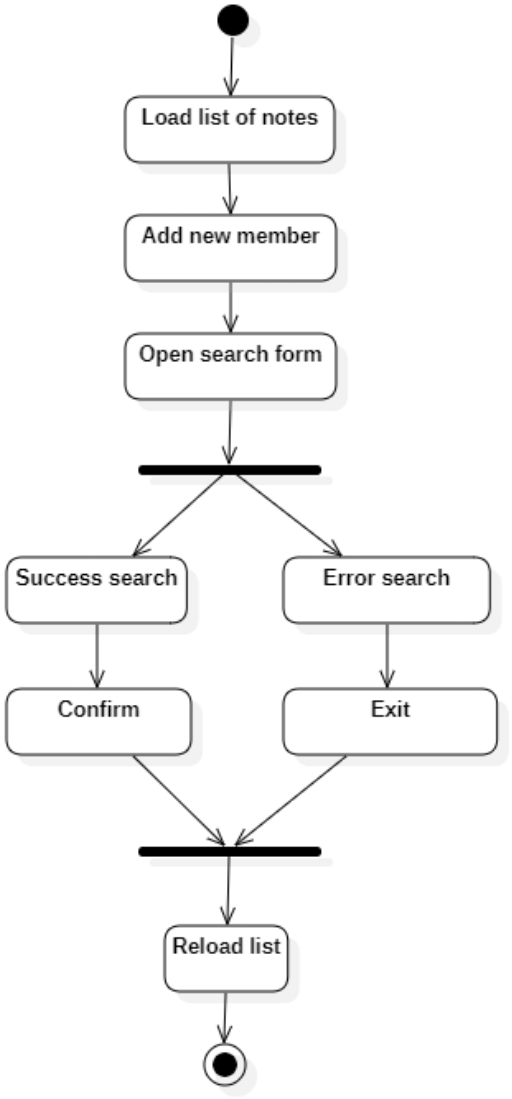

Рисунок 2.8 – Діаграма діяльності для процесу додавання нового користувача до власників списку нотатків

Процес починається із завантаження списку нотаток. Далі після початку додавання нового учасника відкриється форма пошуку користувачів. Користувач

вводить електронну пошту і здійснює пошук потрібної людини. Якщо пошук успішний, користувач може підтвердити своє рішення. Якщо сталася помилка, то система виведе відповідне повідомлення та вийде з форми пошуку. Після цього список нотаток перезавантажується для відображення актуального стану і процес завершується.

2.6 Розробка діаграми станів та переходів

Діаграма станів та переходів (State Transition Diagram) є графічним представленням всіх можливих станів об’єкта або системи та переходів між ними. Вона широко використовується в розробці програмного забезпечення, системному аналізі, а також у проєктуванні апаратного забезпечення. Діаграма станів дозволяє візуалізувати поведінку системи, що допомагає у виявленні та виправленні можливих помилок і в оптимізації процесів.

Діаграма станів та переходів системи з відображенням загального використання вебзастосунку представлена на рис. 2.7.

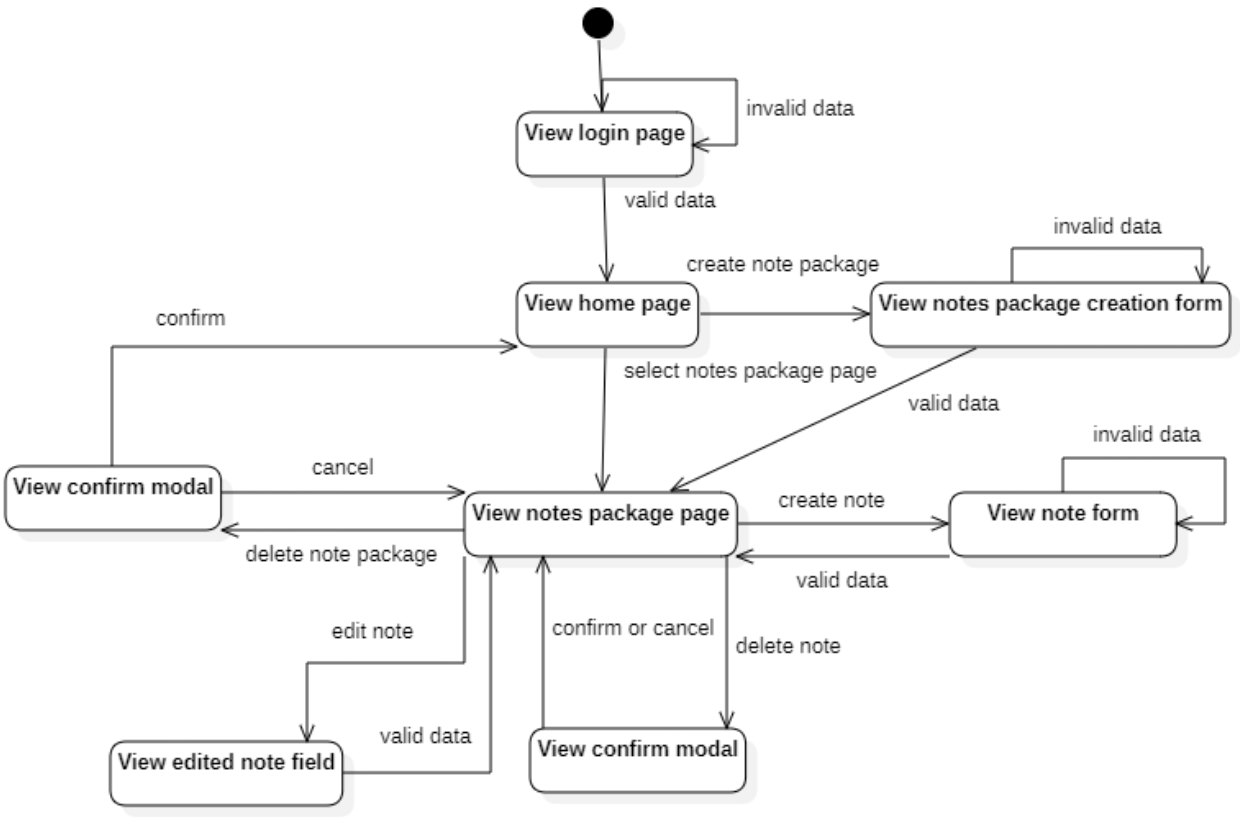

Рисунок 2.9 – Діаграма станів та переходів системи

Діаграма показує різні екрани, які користувач бачить, і можливі дії, які призводять до переходів між цими екранами.

Початковий стан – це «View login page». Це екран входу в систему, де користувач вводить свої облікові дані. Після успішного входу користувач потрапляє на «View home page». Це головна сторінка, де користувач бачить огляд і має доступ до функцій застосунку. На цій сторінці користувач переглядає створені ним раніше списки нотаток та має можливість створити новий список нотаток, що переводить користувача на екран «View notes package creation form», або переглянути існуючі пакети нотаток, що переводить на «View notes package page».

На сторінці створення списку нотаток («View notes package creation form») користувач може заповнити форму для створення нового списку нотаток. Після заповнення і збереження форми користувач переходить на сторінку перегляду тільки що створеного списку нотаток («View notes package page»).

На сторінці перегляду списку нотаток («View notes package page») користувач може створювати нові нотатки, натиснувши відповідну кнопку, що переводить на екран «View note form». Тут користувач може заповнити форму для створення нової нотатки і після збереження повертається на сторінку перегляду списку нотаток. Також користувач може видалити список нотаток, що викликає модальне вікно підтвердження («View confirm modal»), де користувач підтверджує або скасовує дію. Якщо користувач бажає редагувати існуючу нотатку, він переходить на екран «View edited note field», де може внести зміни і зберегти їх, після чого повертається на сторінку перегляду списку нотаток. Видалення нотатки викликає модальне вікно підтвердження («View confirm modal»), де користувач підтверджує або скасовує дію.

2.7 Розробка діаграм компонентів та розгортання системи

Діаграма компонентів системи (component diagram) – це діаграма, на якій відображаються компоненти, залежності та зв’язки між ними. Компоненти програмного забезпечення включають компоненти вихідних кодів, бінарні

компоненти, та компоненти, що можуть виконуватись. Діаграма також показує як компоненти взаємодіють один з одним і які інтерфейси використовуються для їхньої комунікації. Ця діаграма допомагає розробникам зрозуміти структуру системи, планувати її розвиток та інтеграцію, а також забезпечує зручний спосіб спілкування між членами команди, сприяючи ефективнішому керуванню проєктом.

Діаграму компонентів системи представлено на рис. 2.10.

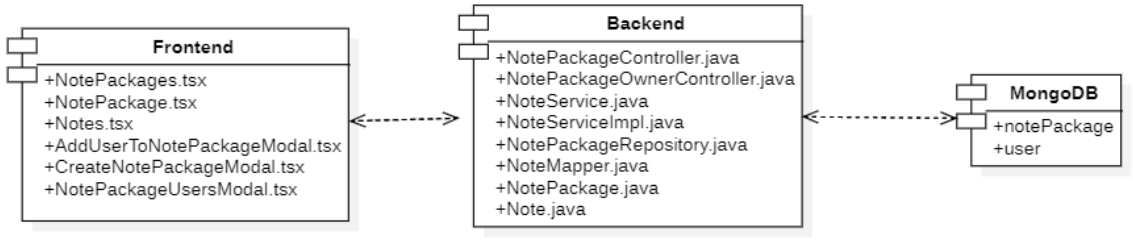

Рисунок 2.10 – Діаграма компонентів системи

Діаграма зображає три ключові компоненти вебзастосунку багатокористувацького ведення нотаток: клієнтська частина (Frontend), серверна частина (Backend) та база даних (MongoDB).

Клієнтська частина займається відображенням даних та взаємодією з користувачами. Вона включає компоненти для відображення списків нотаток, конкретного списку нотаток, конкретних нотаток, модальних вікон для додавання користувачів до списку нотаток, створення нових списків та управління користувачами, які мають доступ до списку нотаток. Інтерфейс користувача забезпечує зручну взаємодію із застосунком та відправляє відповідні запити на сервер для виконання певної логіки.

Серверна частина обробляє ці запити, виконуючи необхідну логіку та взаємодіючи з базою даних. Вона складається з контролерів, які приймають і обробляють HTTP-запити від клієнта, сервісів, що реалізують бізнес-логіку, та репозиторіїв, які відповідають за доступ до даних у базі. Контролери орієнтовані на роботу з списками нотаток і користувачами, забезпечуючи відповідні CRUD

(створення, читання, оновлення, видалення) операції. Сервіси виконують більш складну логіку обробки даних, наприклад, фільтрацію та валідацію, а репозиторії здійснюють прямий доступ до бази даних, перетворюючи дані у відповідні формати для зберігання та обробки.

База даних MongoDB зберігає основні дані застосунку: колекції списків нотаток та користувачів. Вона забезпечує надійне і ефективне зберігання даних, дозволяючи швидко здійснювати операції читання та запису. Сервер взаємодіє з базою даних для отримання, збереження та оновлення даних, необхідних для функціонування застосунку.

Діаграма розгортання системи (deployment diagram) дозволяє показати фізичну архітектуру системи. Вона використовується для відображення того, як програмні компоненти встановлюються та виконуються на апаратних пристроях. На цій діаграмі представлені вузли, що можуть бути як фізичними, так і віртуальними.

Діаграму розгортання системи представлено на рис. 2.11.

Рисунок 2.11 – Діаграма розгортання системи

На діаграмі показано три основні компоненти вебзастосунку для багатокористувацького ведення нотаток. Клієнт користувача взаємодіє з вебсервером через захищене з'єднання HTTPS. Вебсервер обробляє запити від клієнта і взаємодіє з сервером баз даних через протокол TCP/IP для збереження та отримання даних.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи бакалавра спроєктовано вебзастосунок багатокористувацького ведення нотаток. Проєктування цієї системи виявилося комплексним завданням, що вимагає ретельного аналізу функціональних та структурних аспектів системи. Діаграма варіантів використання, діаграми класів та послідовності дій, а також діаграма пакетів були розроблені для забезпечення якості та ефективності розробки.

На основі діаграм варіантів використання і класів вдалося чітко визначити функціональні можливості системи та її структуру. Класи були розподілені за відповідальністю, що сприяє збереженню чіткості та легкості управління кодом. Діаграма послідовності системи вказує на послідовність взаємодії об’єктів та модулів, що спрощує розуміння логіки програми. Діаграма пакетів системи вказує на логічну організацію програми та розподіл відповідальності між різними частинами системи.

Отже, аналіз та моделювання функціональних та структурних аспектів системи дозволяють створити ефективний та добре структурований вебзастосунок для багатокористувацього ведення нотаток, який відповідатиме вимогам та потребам користувачів.

3 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ ТА ОПИС БАЗИ ДАНИХ

3.1 Опис технологій розробки

Під час розробки програмного забезпечення використано наступні технології:

– Java;
– Spring Framework;
– TypeScript;
– React;
– MongoDB;
– Redis;
– Docker.

Java є однією з найпоширеніших мов програмування, яка була створена компанією Sun Microsystems у 1995 році і пізніше придбана Oracle Corporation [4]. Основною особливістю Java є її платформо-незалежність, що досягається завдяки використанню віртуальної машини Java (JVM). Програми, написані на Java, компілюються у байт-код, який може виконуватися на будь-якій платформі, що має відповідну JVM. Це робить Java особливо популярною для розробки міжплатформних застосунків.

Java використовується у різних сферах програмування, включаючи веброзробку, розробку корпоративних застосунків, мобільних застосунків, а також у вбудованих системах. Однією з найбільших переваг Java є її об’єктно-орієнтована природа, що дозволяє розробникам створювати модульні програми з повторно використовуваним кодом [5]. Крім того, Java має багату стандартну бібліотеку, яка охоплює все від структур даних і алгоритмів до мережевих протоколів і графічних інтерфейсів користувача.

JVM забезпечує незалежність виконання програм від конкретної апаратної платформи. Також JVM включає в себе збирач сміття, який автоматично управляє пам’яттю, та має механізми для забезпечення безпеки виконання програм [6]. Вона

також підтримує різні мовні інтерфейси, що дозволяє використовувати JVM не лише для Java, але й для інших мов програмування, таких як Kotlin і Scala.

Крім того, Java підтримує багатопоточність, що дозволяє створювати високопродуктивні програми, які можуть ефективно використовувати багатоядерні процесори [7]. Це важливо для розробки серверних застосунків і великих обчислювальних систем. Java має велику спільноту розробників, яка постійно підтримує та розширює можливості мови та її екосистеми.

Використання Java в сучасному світі також включає хмарні обчислення, де вона часто використовується для створення масштабованих і надійних серверних рішень. Різні фреймворки, такі як Spring, Hibernate та Apache Struts, роблять розробку на Java більш ефективною та продуктивною, дозволяючи розробникам зосередитися на бізнес-логіці, а не на інфраструктурних питаннях.

Таким чином, Java залишається ключовою технологією у світі програмування завдяки своїй стабільності, безпеці, платформній незалежності та потужній екосистемі інструментів і бібліотек. Вона продовжує еволюціонувати, адаптуючись до нових викликів і потреб ринку, що робить її актуальною і в майбутньому.

Spring Framework є одним з найпопулярніших фреймворків для розробки на Java, спеціально розробленим для спрощення створення корпоративних застосунків. Він був створений Родом Джонсоном і вперше випущений у 2002 році. Spring пропонує комплексне рішення для розробки серверних додатків, надаючи розробникам набір інструментів і шаблонів, які полегшують створення та тестування застосунків [8].

Основною особливістю Spring є його архітектура інверсії управління (IoC) та впровадження залежностей (DI). Це дозволяє створювати гнучкі і легко тестовані застосунки, де компоненти можуть бути вільно пов’язані між собою без жорсткого зв’язування. Цей підхід значно покращує модульність і повторюваність коду.

Spring також включає модулі для різних аспектів розробки застосунків, таких як Spring MVC для створення вебзастосунків, Spring Data для роботи з

базами даних, Spring Security для забезпечення безпеки додатків, і Spring Boot для швидкої та простої конфігурації застосунків. Spring Boot особливо популярний серед розробників завдяки своїй здатності швидко створювати готові до розгортання застосунки з мінімальними зусиллями [9].

Однією з основних переваг Spring є його інтеграція з іншими технологіями та фреймворками. Він підтримує різні формати даних і технології зберігання, такі як SQL, NoSQL, і навіть хмарні сервіси [10]. Це дозволяє розробникам використовувати Spring як основний фреймворк для створення складних та масштабованих застосунків, які можуть працювати в різних середовищах.

Spring також відомий своєю гнучкістю та розширюваністю. Розробники можуть легко додавати нові функціональні можливості до своїх застосунків, використовуючи широкий набір бібліотек та модулів, доступних у Spring екосистемі. Це робить його ідеальним вибором для великих корпоративних застосунків, де необхідна висока гнучкість і масштабованість.

Завдяки своїй потужній спільноті та постійним оновленням, Spring Framework залишається актуальним і популярним серед розробників. Він продовжує адаптуватися до нових технологій і підходів, таких як мікросервіси та хмарні обчислення, що робить його важливим інструментом для сучасних розробників програмного забезпечення.

TypeScript – це мова програмування, розроблена компанією Microsoft, яка є надбудовою над JavaScript [11]. Вона була вперше представлена у 2012 році і швидко завоювала популярність серед розробників завдяки своїм розширеним можливостям та додатковим інструментам для розробки великих застосунків. TS додає статичну типізацію до JS, що дозволяє виявляти помилки на етапі компіляції, а не під час виконання програми.

Однією з головних переваг TypeScript є її сумісність з JavaScript. Будь-який дійсний код JavaScript є також дійсним кодом TypeScript. Це дозволяє поступово переходити від JavaScript до TypeScript без необхідності переписувати весь існуючий код. TypeScript компілюється у чистий JavaScript, який може

виконуватися у будь-якому середовищі, що підтримує JavaScript, включаючи веббраузери та серверні платформи, такі як Node.js.

Статична типізація в TypeScript дозволяє розробникам визначати типи для змінних, функцій та об’єктів, що допомагає запобігати багатьом поширеним помилкам [12]. Типи можуть бути простими (наприклад, число, рядок) або складними (наприклад, об’єкти, функції з певними параметрами та типами повернення). Це значно полегшує розробку та підтримку великих кодових баз, де важливо забезпечити правильність та узгодженість даних.

TypeScript також підтримує сучасні можливості JavaScript, включаючи нові синтаксичні конструкції та функції, визначені в стандарті ECMAScript [13]. Це дозволяє розробникам використовувати найновіші функції JavaScript, навіть якщо вони ще не підтримуються у всіх браузерах. Крім того, TypeScript надає розширені можливості, такі як інтерфейси, перетини та об’єднання типів, які роблять код більш виразним та гнучким.

Однією з ключових особливостей TypeScript є його інтеграція з редакторами коду та IDE. Інструменти, такі як Visual Studio Code та IntelliJ IDEA, надають потужні можливості для автозаповнення, рефакторингу та налагодження коду, що значно підвищує продуктивність розробників. TypeScript також підтримує модулі та простори імен, що дозволяє організовувати код у масштабних проєктах.

TypeScript став стандартом де-факто для розробки великих вебзастосунків і широко використовується у багатьох відомих проєктах та компаніях [14]. Його використання допомагає створювати надійний, зрозумілий та легко підтримуваний код, що особливо важливо для командної розробки та довгострокових проєктів.

React – це JavaScript-бібліотека для створення користувацьких інтерфейсів, яка була розроблена Facebook і вперше випущена у 2013 році. React дозволяє розробникам створювати великі вебзастосунки, які можуть змінювати дані без перезавантаження сторінки. Головною ідеєю React є компонентний підхід до розробки інтерфейсів, що дозволяє розробникам створювати повторно використовувані компоненти [15].

Однією з ключових особливостей React є використання віртуального DOM (Document Object Model) [16]. Замість того, щоб маніпулювати реальним DOM безпосередньо, React працює з його віртуальним представленням, що дозволяє значно зменшити кількість операцій з реальним DOM і підвищити продуктивність застосунків. Коли стан компонента змінюється, React створює нове віртуальне дерево і порівнює його з попереднім станом, застосовуючи тільки ті зміни, які необхідні для оновлення реального DOM.

React також підтримує односпрямовану прив’язку даних, що робить код більш передбачуваним і легким для налагодження [17]. Розробники можуть використовувати JSX, розширення синтаксису JavaScript, яке дозволяє писати HTML-подібний код всередині JavaScript, що робить компоненти більш читабельними та зрозумілими.

Ще однією важливою особливістю React є його екосистема. React не є повноцінним фреймворком, а лише бібліотекою для створення інтерфейсів користувача. Проте, існує велика кількість бібліотек та інструментів, які доповнюють React, таких як React Router для маршрутизації, Redux для управління станом, і багато інших. Це дозволяє розробникам створювати потужні та гнучкі застосунки, які легко масштабувати та підтримувати.

Завдяки своїй простоті, гнучкості та потужності, React став одним з найпопулярніших інструментів для розробки сучасних вебзастосунків. Його використання значно прискорює процес розробки, покращує продуктивність застосунків і знижує витрати на їх підтримку.

MongoDB – це нереляційна база даних, яка належить до категорії NoSQL баз даних. Вона була створена у 2007 році компанією 10gen (зараз відомою як MongoDB Inc.) і стала популярною завдяки своїй гнучкості, масштабованості та високій продуктивності [18]. MongoDB зберігає дані у форматі BSON (Binary JSON), що дозволяє зберігати документи з різними структурами у одній колекції.

Однією з основних переваг MongoDB є її документно-орієнтована модель. На відміну від реляційних баз даних, де дані зберігаються у таблицях з фіксованою схемою, MongoDB дозволяє зберігати дані у вигляді документів, які можуть мати

різні структури. Це робить MongoDB ідеальною для роботи з динамічними даними, де структура може змінюватися з часом. Документи в MongoDB можуть містити вкладені документи і масиви, що дозволяє зберігати складні об’єкти у їх природному вигляді.

MongoDB також підтримує горизонтальну масштабованість через шардинг, що дозволяє розподіляти дані по багатьом серверам [19]. Це робить MongoDB придатною для роботи з великими обсягами даних та забезпечення високої доступності і відмовостійкості. Реплікація є ще однією важливою функцією MongoDB, яка дозволяє автоматично створювати резервні копії даних та забезпечувати безперервність роботи у випадку відмови основного сервера.

Запити в MongoDB виконуються за допомогою мови запитів, яка дозволяє виконувати складні операції над даними, включаючи фільтрацію, сортування, агрегацію та об’єднання. MongoDB також підтримує індексацію, що дозволяє значно прискорити виконання запитів. Індекси можуть бути створені на будь-яких полях документа, включаючи вкладені поля та масиви.

Ще однією важливою особливістю MongoDB є її підтримка транзакцій, що дозволяє забезпечувати атомарність, узгодженість, ізольованість та надійність (ACID) операцій. Це особливо важливо для застосунків, де необхідно забезпечити цілісність даних у складних операціях.

MongoDB також має велику екосистему інструментів та бібліотек, які роблять її використання ще більш зручним та ефективним. MongoDB Atlas – це хмарне рішення, яке надає повністю керовану версію MongoDB, що дозволяє розробникам зосередитися на розробці застосунків, а не на управлінні базою даних.

Завдяки своїй гнучкості, масштабованості та продуктивності, MongoDB стала однією з найпопулярніших NoSQL баз даних і використовується у багатьох великих проєктах та компаніях по всьому світу. Вона підходить для різних сценаріїв використання, включаючи вебзастосунки, мобільні застосунки, аналітичні системи та багато інших.

Redis (Remote Dictionary Server) – це високо продуктивна система управління базами даних, що зберігає дані в оперативній пам’яті, забезпечуючи надзвичайно швидкий доступ до них [20]. Це ключ-значення сховище з відкритим вихідним кодом, яке підтримує різні структури даних, такі як рядки, хеші, списки, множини, відсортовані множини, бітові поля, геопросторові індекси та потоки. Redis часто використовується як кеш, брокер повідомлень і база даних.

Основна характеристика Redis – це його швидкодія. Завдяки зберіганню даних в оперативній пам’яті, запити до бази даних виконуються значно швидше порівняно з традиційними дисковими базами даних. Ця швидкодія дозволяє Redis використовуватися в сценаріях, де необхідно обробляти великі обсяги даних у реальному часі, таких як аналітика, обробка потоків даних та відстеження сесій користувачів.

Redis підтримує асинхронну реплікацію, що забезпечує високу доступність та надійність даних [21]. Він може автоматично зберігати знімки даних на диск через певні інтервали часу або з використанням журналу команд, що дозволяє відновлювати дані у випадку збою. Це робить Redis підходящим для критично важливих застосунків, які вимагають мінімальних втрат даних і швидкого відновлення.

Ще однією важливою особливістю Redis є його здатність до горизонтального масштабування. Завдяки механізму розподілу даних між кількома вузлами, Redis може ефективно обробляти великі обсяги даних і підтримувати високу продуктивність навіть у випадку збільшення навантаження.

Redis також підтримує транзакції, забезпечуючи атомарність операцій, що дозволяє виконувати кілька команд як єдине ціле без ризику часткового виконання. Це особливо корисно в застосунках, які потребують консистентності даних.

Ще одна важлива можливість Redis – це підтримка Lua-скриптів, які дозволяють виконувати складні операції на стороні сервера, зменшуючи кількість запитів до бази даних і покращуючи продуктивність застосунків.

Завдяки своїм особливостям Redis широко використовується у веброзробці, інтернет-магазинах, соціальних мережах та багатьох інших областях, де потрібна

швидка обробка та зберігання даних. Його гнучкість і висока продуктивність роблять його одним з найпопулярніших рішень для розробників та інженерів.

Docker – це платформа для автоматизації розгортання та управління контейнеризованими додатками [22]. Контейнери надають ізольоване середовище, що включає все необхідне для роботи програмного забезпечення. Це забезпечує стабільність і повторюваність в роботі застосунків у різних середовищах, від машин розробників до виробничих серверів.

Docker складається з кількох основних компонентів. Docker Engine включає Docker Daemon, який управляє контейнерами, і Docker CLI, який дозволяє взаємодіяти з Daemon через командний рядок. Docker Images – це шаблони для створення контейнерів, які містять усі необхідні компоненти для роботи програми. Docker Containers є запущеними екземплярами цих образів, забезпечуючи ізольоване середовище для кожного застосунка.

Docker Compose – це інструмент, який спрощує роботу з багатоконтейнерними застосунками [23]. Використовуючи файл конфігурації, можна визначити всі служби, з яких складається застосунок, і керувати ними як єдиним цілим. Це дозволяє запускати кілька контейнерів одночасно і налаштовувати їх взаємодію, що значно спрощує процес розгортання та управління складними системами.

Основні переваги Docker та Docker Compose полягають в ізоляції контейнерів, що запобігає конфліктам між різними версіями залежностей, а також у портативності, що дозволяє легко переносити застосунки між різними середовищами. Масштабованість і ефективне використання ресурсів роблять ці інструменти надзвичайно корисними для розробників, тестувальників і DevOps-інженерів.

3.2 Опис структури бази даних

Для зберігання даних застосунку створено дві колекції в базі даних MongoDB:

– User;

– NotePackage.

Структуру колекції User представлено на рис. 3.1.

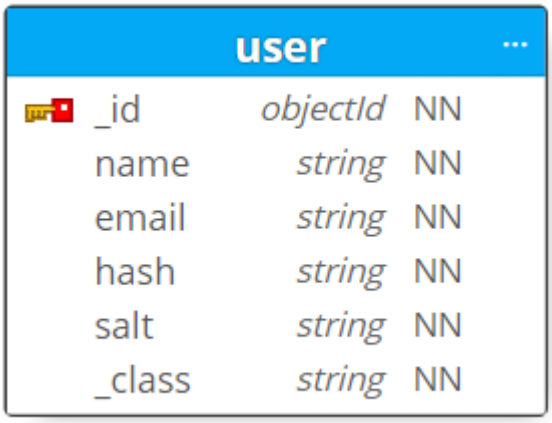

Рисунок 3.1 – Структура документу User

Вона містить такі поля:

1) _id (objectId not null) – унікальний ідентифікатор документу в колекції, який автоматично генерується MongoDB;

2) name (string not null) – ім’я користувача;

3) email (string not null) – електронна пошта користувача;

4) hash (string not null) – хеш паролю користувача;

5) salt (string not null) – сіль, використана для хешування паролю;

6) _class (string not null) – тип класу, що використовується у випадку збереження об’єктів за допомогою відображення на документну модель фреймворком Spring.

Структуру колекції NotePackage представлено на рис. 3.2.

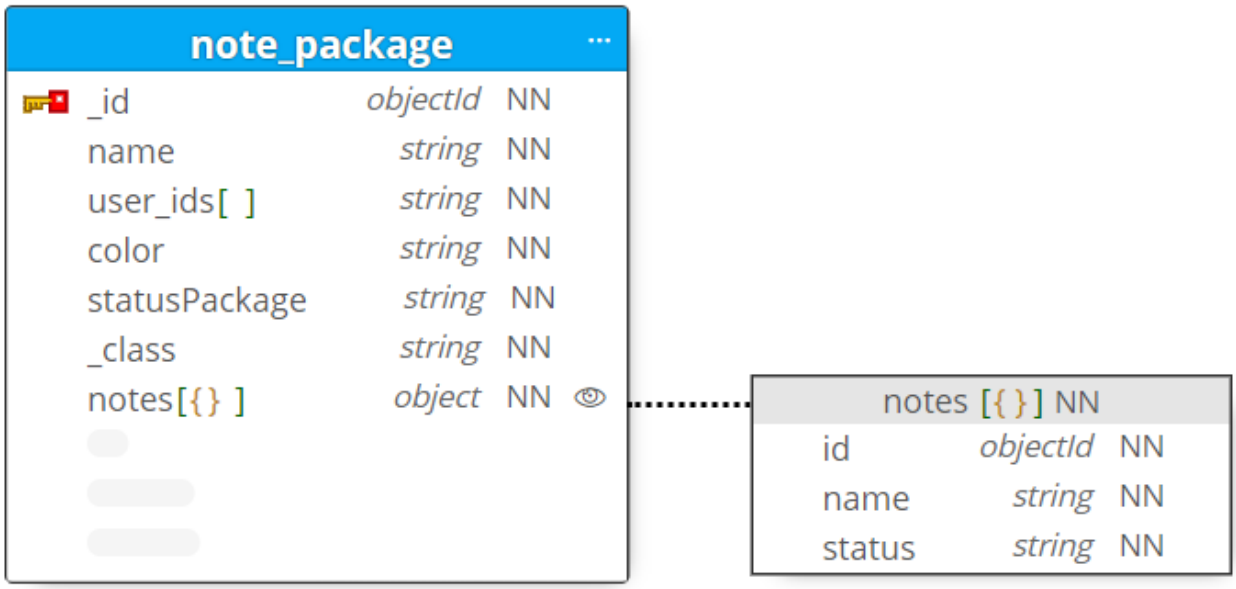

Рисунок 3.2 – Структура документу NotePackage

Вона містить такі поля:

а) _id (objectId not null) – унікальний ідентифікатор документу в колекції, що автоматично генерується MongoDB;

б) name (string not null) – назва списку нотаток;

в) user_ids (array of string not null) – масив ідентифікаторів користувачів, які є власниками цього списку нотаток;

г) color (string not null) – колір списку нотаток;

д) statusPackage (string not null) – варіація статусів нотаток;

е) _class: (string not null) – тип класу, що використовується у випадку збереження об’єктів за допомогою відображення на документну модель фреймворком Spring;

ж) notes (array of object not null) – масив об’єктів, які представляють нотатки. Структура об’єкта в масиві notes:

– id (objectId not null) – унікальний ідентифікатор нотатки;
– name (string not null) – назва нотатки;
– status (string not null) – статус нотатки.

Для зберігання тимчасових даних (кешування) в застосунку використовується Redis. Структура пар ключ-значення, які зберігаються в ньому представлено на рис. 3.3.

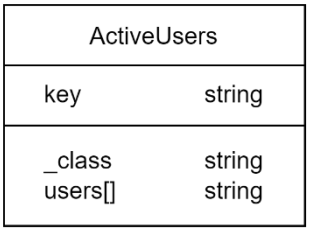

Рисунок 3.3 – Структура пар ключ-значення

Вона містить такі поля:

1) key (string) – унікальний ідентифікатор пари, що складається з константи «activeUsers» та унікального ідентифікатора списка нотаток;

2) _class (string) – тип класу, що використовується у випадку збереження об’єктів за допомогою відображення на модель ключ-значення фреймворком Spring;

3) users (array of string) – масив ідентифікаторів користувачів, які зараз використовують цей список нотаток.

3.3 Розробка мокапів системи

Мокапи системи – це моделі або прототипи програмного забезпечення, які відображають вигляд та функціональність системи перед тим, як вона буде реалізована. Зазвичай мокапи використовуються на ранніх етапах розробки для визначення вимог користувача, тестування концепцій та взаємодії з користувачем.

Застосунок включатиме такі вікна як:

- «Головна сторінка»;
- «Форма створення списку нотаток»;
- «Сторінка звичайного списку нотаток»;
- «Сторінка списку нотаток зі статусами»;
- «Форма пошуку користувачів для додавання в список нотатків».

Мокап головної сторінки представлено на рис. 3.4. Після запуску застосунку буде відкриватись головна сторінка.

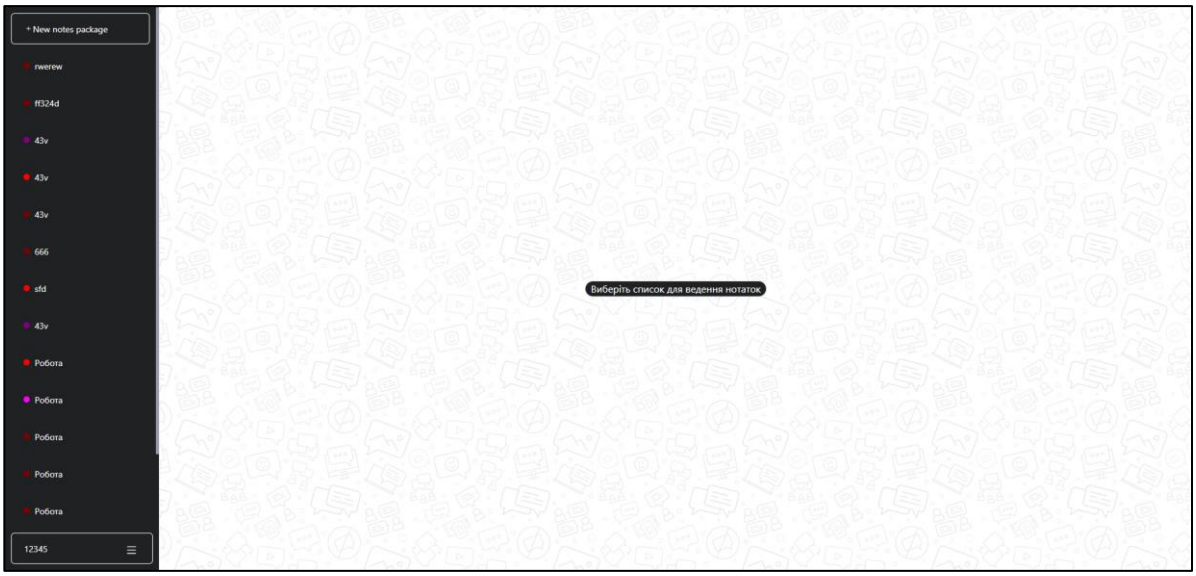

Рисунок 3.4 – Головна сторінка

Після перегляду головної сторінки, користувач може створити список нотатків. Мокапи форми додавання представлено на рис. 3.5.

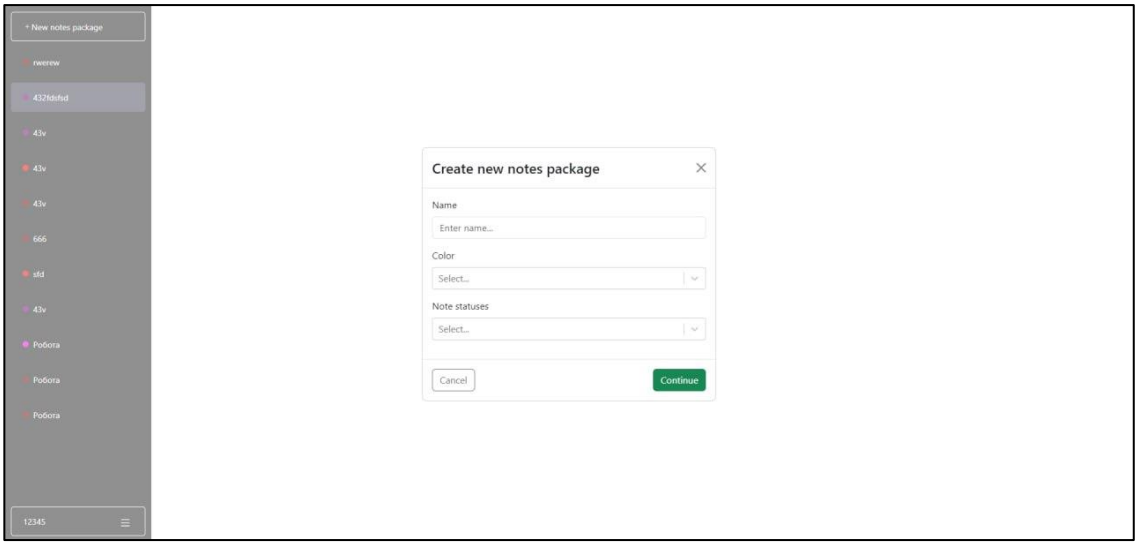

Рисунок 3.5 – Форма створення списку нотаток

Користувач може створити декілька видів списків нотаток. Звичайний список нотаток, тобто той що не містить різних статусів для нотаток, представлено на рис. 3.6.

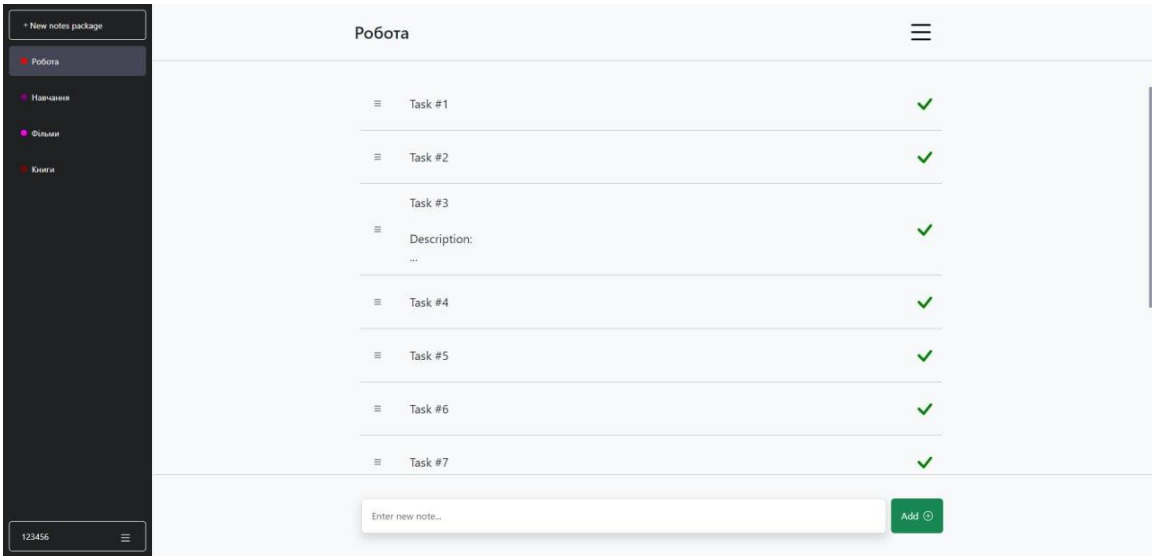

Рисунок 3.6 – Сторінка звичайного списку нотаток

Приклад списку нотаток зі статусами представлено на рис. 3.7.

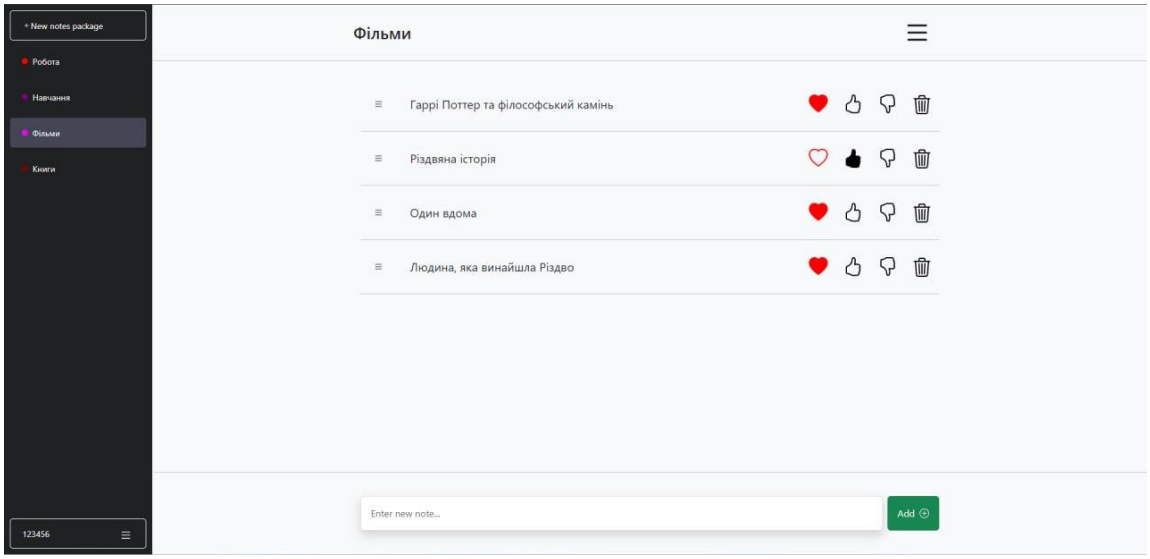

Рисунок 3.7 – Сторінка списку нотаток зі статусами

Після переходу на сторінку списку нотатків користувач може додати інших користувачів до цього списку. Мокапи форми пошуку інших користувачів на рис. 3.8.

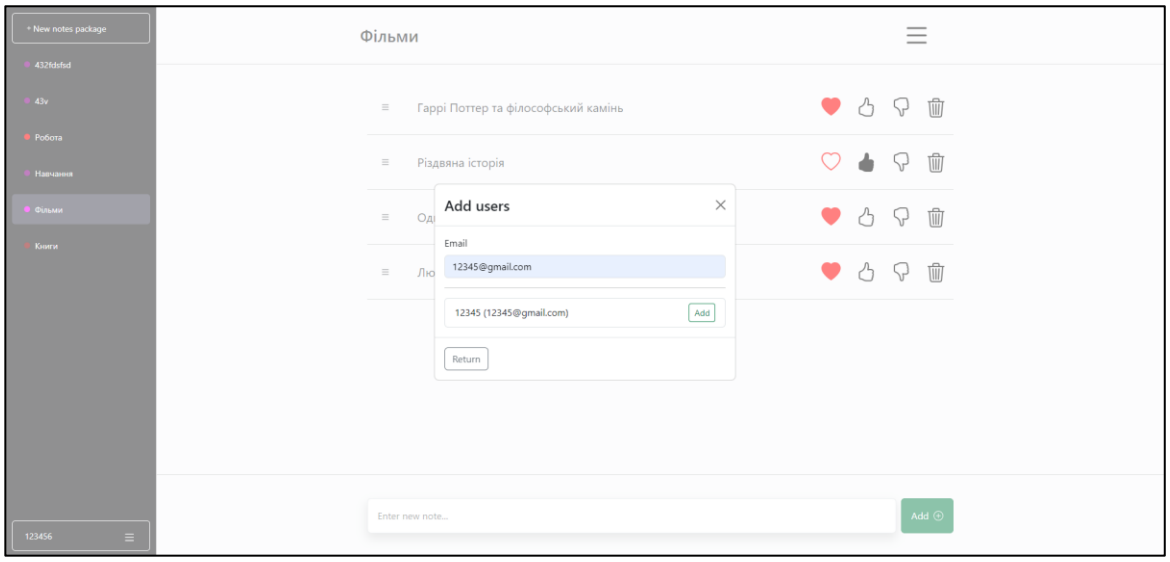

Рисунок 3.8 – Форма пошуку користувачів для додавання в список нотатків

Мокапи системи надають візуальне уявлення про інтерфейс користувача та його функціональність, що є важливим етапом у процесі розробки. Вони дозволяють зрозуміти, як взаємодіє користувач з системою та як вона реагує на його дії.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи представлено опис технологій розробки вебзастосунку: Java, Spring Framework, React, TypeScript та MongoDB.

Також було ретельно описано структуру бази даних, а саме колекції User та NotePackage, де для кожного поля наведено опис його призначення та типу даних. Це дозволило отримати чітке уявлення про те, як дані організовані та взаємодіють між собою у системі.

Представлено 5 мокапів для системи та описано їх розробку. Мокапи системи надають візуальне уявлення про інтерфейс користувача та його функціональність, що є важливим етапом у процесі розробки. Вони дозволяють зрозуміти, як взаємодіє користувач з системою та як вона реагує на його дії.

Отже, розділ надав чітке уявлення про технічну сторону проєкту: використані технології, структуру бази даних та візуалізацію функціоналу системи за допомогою мокапів.

4 РЕЗУЛЬТАТИ РОЗРОБКИ

4.1 Кодування застосунка

4.1.1 Створення сервісу для роботи з нотатками

Клас NoteServiceImpl, який наведений у додатку А, є сервісним компонентом, що забезпечує управління нотатками. Він описує CRUD-операції (створення, читання, оновлення, видалення) для роботи з нотатками та керування користувачами, які мають доступ до цих нотаток. Клас надає такі основні методи:

– отримання списку нотаток користувача (getNotePackages): метод отримує список всіх пакунків нотаток, до яких має доступ користувач з заданим userId, з бази даних, конвертує їх в скорочене представлення та повертає;

– отримання конкретного пакета нотаток користувача (getNotePackage): метод отримує конкретний пакет нотаток з бази даних на основі notePackageId та userId, перевіряє, чи існує цей пакет та чи він доступний для даного користувача, конвертує та повертає повне представлення цього пакета нотаток;

– створення нового пакета нотаток (createNotePackage): метод створює новий пакет нотаток на основі наданого DTO, ініціалізує його порожнім списком нотаток, додає до нього користувача з заданим userId, зберігає його в базі даних та повертає відображення створеного пакета нотаток;

– оновлення існуючого пакета нотаток (updateNotePackage): метод на основі наданого DTO перевіряє валідність статусів нотаток у пакеті, генерує ідентифікатори для нових нотаток, оновлює інформацію про пакет та зберігає його в базі даних;

– видалення пакета нотаток (deleteNotePackageString): метод на основі notePackageId та userId перевіряє чи існує відповідний пакет нотаток і чи має користувач доступ до нього, після чого видаляє цей пакет нотаток з бази даних;

– додавання нового користувача до пакета нотаток (addUserToNotePackage): метод на основі notePackageId та ownerUserId перевіряє чи існує відповідний пакет нотаток і чи має користувач доступ до нього, а також

що користувач з guestUserId ще не доданий до цього списку нотаток, після чого додає нового користувача та зберігає зміни в базі даних;

– отримання списку користувачів пакета нотаток (getNotePackageUsers): метод на основі notePackageId та ownerUserId перевіряє чи існує відповідний пакет нотаток і чи має користувач доступ до нього, після чого знаходить в базі даних список всіх користувачів, які мають доступ до пакета нотаток, отримує від іншого мікросервісу детальну інформацію про цих користувачів та повертає її;

– видалення користувача з пакета нотаток (deleteUserFromNotePackage): метод на основі notePackageId та userId перевіряє чи існує відповідний пакет нотаток і чи має користувач доступ до нього, після чого видаляє користувача з пакета нотаток на основі guestUserId.

4.1.2 Створення сервісів для роботи з користувачами

Клас UserServiceImpl, який наведений у додатку Б, відповідає за управління користувачами в системі, включаючи реєстрацію, автентифікацію та отримання інформації про користувачів. Клас надає такі основні методи:

– реєстрація нового користувача (register): метод конвертує надані дані, генерує сіль для пароля, хешує пароль з цією сіллю, зберігає користувача в базі даних і повертає JWT-токен для автентифікації;

– вхід користувача в систему (login): метод знаходить користувача за email, перевіряє хеш пароля з врахуванням солі, і якщо дані правильні, генерує і повертає JWT-токен для автентифікації, або ж у випадку невідповідності даних викидає виключення;

– отримання даних про конкретного користувача (getUser): метод знаходить користувача за його userId і повертає його дані у вигляді DTO, або викидає виключення, якщо користувача не знайдено;

– отримання користувачів за списком email (getUsersByEmails): метод знаходить користувачів за списком email, конвертує кожного з них в DTO і повертає цей список;

– отримання користувачів за списком ідентифікаторів (getUsersByIds): метод знаходить користувачів за списком userId, конвертує кожного з них в DTO і повертає цей список, але якщо хоча б один з користувачів не знайдений, викидається виключення.

Клас AuthorizationUtil, який наведений у додатку В, є компонентом, що забезпечує утиліти для роботи з авторизацією користувачів. Клас надає такі основні методи:

– генерація JWT-токену (generateJwtToken): метод використовує секретний ключ (jwtSecretKey) для підпису токену, встановлює час видачі та термін дії токену (jwtTokenValidity) та повертає сформований токен;

– створення авторизаційного кукі (createAuthorizationCookie): метод створює HTTP-кукі з JWT-токеном, встановлює значення її максимального віку життя та повертає створене кукі;

– видалення авторизаційного кукі (deleteAuthorizationCookie): метод створює HTTP-кукі з null значенням та максимальним віком 0, що призводить до видалення кукі в браузері користувача.

Клас JwtAuthenticationFilter, який наведений у додатку Г, є компонентом, який реалізує фільтр для аутентифікації користувачів за допомогою JWT-токенів у шлюзі (gateway). Клас надає такі основні методи:

– фільтрація HTTP-запитів для аутентифікації користувачів (filter): метод отримує JWT-токен з кукі та перевіряє його дійсність і у випадку успішної валідації, додає ідентифікатор користувача userId до заголовків запиту, інакше встановлює статус відповіді як 401 UNAUTHORIZED.

4.1.3 Створення сервісів для роботи з діями користувачами

Клас ActionServiceImpl, який наведений у додатку Д, відповідає за управління діями користувачів у реальному часі, таких як приєднання або вихід з онлайн кімнати. Клас надає такі основні методи:

– оновлення та отримання списку активних користувачів (updateAndGetActiveUsers): метод використовує headerAccessor, щоб отримати

userId з атрибутів сесії, отримує список активних користувачів для цієї онлайн кімнати з кешу, оновлює його в залежності від типу дії (додає або видаляє користувача), зберігає оновлений список назад у кеш та повертає об’єкт, що містить інформацію про дію, користувача та оновлений список активних користувачів.

Клас ActionController, який наведений у додатку Е, є контролером, що обробляє WebSocket повідомлення, пов’язані з діями користувачів. В кожному публічному методі зберігається в параметрах сесії ідентифікатор активної кімнати для користувача, що викликав сервіс. Викликається метод updateAndGetActiveUsers класу ActionServiceImpl для оновлення та отримання актуального списку активних користувачів. А також відправляє повідомлення з оновленою інформацією всім онлайн користувачам цього списку нотаток. Клас надає такі основні методи:

– обробка приєднання користувача до хабу (join);
– обробка змін списків нотаток користувачем (chat);
– обробка процесу редагування нотатки користувачем (edit);
– обробка переміщення нотатки користувачем (move).

Клас WebSocketEventListener, який наведений у додатку Ж, є компонентом, що обробляє події WebSocket, пов’язані з підключеннями та відключеннями користувачів. Клас надає такі основні методи:

– обробка відключення користувача від WebSocket (handleWebSocketDisconnectListener): метод отримує атрибути сесії, якщо вони присутні, метод отримує з них hubId, викликає метод сервісу для оновлення списку активних користувачів і відправляє оновлені дані всім онлайн користувачам цього списку нотаток.

4.1.4 Створення компоненту для роботи з списками нотаток

React компонент NotePackages, який наведений у додатку И, забезпечує користувачів можливістю керувати пакунками нотаток. Він реалізує

функціональність бокового меню, яке відкривається і закривається, відображає наявні пакунки нотаток, дозволяє створювати нові пакунки та взаємодіяти з ними.

Компонент використовує React хуки для управління станом. Стан toggled визначає, чи відкрите бокове меню, а стан broken використовується для визначення, чи змінилося меню через зміну розміру екрану або інші причини. Дані про пакунки нотаток отримуються через API-запит, реалізований за допомогою notePackageApi.useGetNotePackagesQuery().

Для управління модальним вікном створення нового пакунка нотаток використовується хук useAppModal, який надає функцію openModal. Це модальне вікно відкривається при натисканні відповідної кнопки в меню.

Стилі елементів меню задаються через об’єкт menuItemStyles, який визначає зовнішній вигляд кореневого елемента та кнопок, включаючи їх стан при наведенні та активному стані. Ці стилі забезпечують консистентний і привабливий інтерфейс для користувачів.

Компонент NotePackages складається з двох частин. Перша використовується для відображення списку пакунків нотаток, включаючи кнопку для створення нового пакунка та самі пакунки, які отримуються через API-запит. Друга відображає інформацію про користувача за допомогою компонента UserItem.

Коли користувач взаємодіє з меню, наприклад, натискає кнопку створення нового пакунка або закриває меню, відповідні методи оновлюють стан компонента. Метод handleMenuClose закриває бокове меню, встановлюючи стан toggled у false, тоді як метод handleMenuOpen відкриває меню, встановлюючи стан toggled у true. Метод handleNewPackageCreate відкриває модальне вікно для створення нового пакунка нотаток.

Крім того, компонент використовує CreateNotePackageModal для відображення модального вікна, яке дозволяє створювати нові пакунки нотаток. Компонент Outlet використовується для рендерингу вкладених маршрутів, що дозволяє динамічно відображати вміст залежно від поточного маршруту застосунку.

4.1.5 Створення компонентів для роботи з нотатками

React компонент Notes, який наведений у додатку К, дозволяє користувачам керувати нотатками в пакунках, забезпечуючи функціональність для редагування, видалення нотаток, а також переміщення через інтерфейс перетягування. Компонент приймає декілька параметрів: onSave, onEditing, onMoving, isLoading і processedNotes. Ці параметри визначають функції для обробки подій збереження, редагування і переміщення нотаток, а також передають стан завантаження і нотаток, що в цей момент оброблюються іншими користувачами.

Компонент використовує кілька хуків для управління станом. Хук useAppModal використовується для керування модальним вікном видалення нотаток. useFormContext і useFieldArray слугують для управління станом форми і списку нотаток. Хук watch слідкує за змінами у масиві нотаток, дозволяючи компоненту реагувати на будь-які зміни в реальному часі.

У компоненті реалізовано функції для обробки подій. onMovingThrottled обробляє події переміщення нотаток, обмежуючи частоту виклику функції onMoving за допомогою механізму throttling. Функція updateNoteStatus призначена для оновлення статусу нотатки. Вона знаходить потрібну нотатку за її ідентифікатором, оновлює її статус і зберігає зміни через виклик onSave. Функція moveNote переміщує нотатку у масиві з однієї позиції в іншу і також зберігає зміни.

Компонент використовує бібліотеку react-beautiful-dnd для реалізації функціоналу перетягування нотаток. DragDropContext обгортає весь компонент, забезпечуючи контекст для перетягування. Droppable визначає зону, куди можна перетягувати нотатки, а DraggableNotes відображає самі нотатки, дозволяючи їх перетягувати. При переміщенні нотаток, DragDropContext та Droppable взаємодіють з moveNote для забезпечення коректного оновлення стану.

Модальне вікно для видалення нотаток (DeleteNoteModal) викликається при необхідності, забезпечуючи користувачам можливість видаляти нотатки зі своїх пакунків.

Компонент CreateNote, який наведений у додатку Л, забезпечує інтерфейс для створення нових нотаток. Він приймає два параметри: onSave і isSaving. onSave є функцією, яка викликається для збереження нової нотатки, а isSaving вказує на те, чи відбувається процес збереження в цей момент.

Компонент використовує хук useFieldArray для управління масивом нотаток. Цей хук забезпечує функцію append, яка дозволяє додавати нові нотатки до масиву. Також використовується хук useAppForm, який налаштовує форму з початковими значеннями та схемою валідації. Функція onFormSubmit визначається як обробник подій для форми. Вона виконує додавання нової нотатки до масиву, скидає стан форми та викликає onSave для збереження змін.

Інтерфейс компоненту включає форму, створену за допомогою компонентів FormProvider і Form. Вона містить текстову область для введення нової нотатки, яка реалізована через компонент TextareaAutosize. Цей компонент дозволяє користувачам вводити текст, автоматично підлаштовуючи висоту текстової області залежно від кількості рядків. Поле введення має кілька стилів, що забезпечують зручність користування, і воно вимикається під час процесу збереження, щоб уникнути введення нових даних.

Кнопка «Add» викликає функцію submitForm, яка передає введені дані у форму. Кнопка також вимикається під час процесу збереження, що попереджує користувачів про те, що нова нотатка зберігається. Кнопка стилізована за допомогою класів Bootstrap і має значок PlusCircle для візуальної індикації додавання.

4.2 Опис створеного інтерфейсу

Головна сторінка застосунку представлена на рис. 4.1.

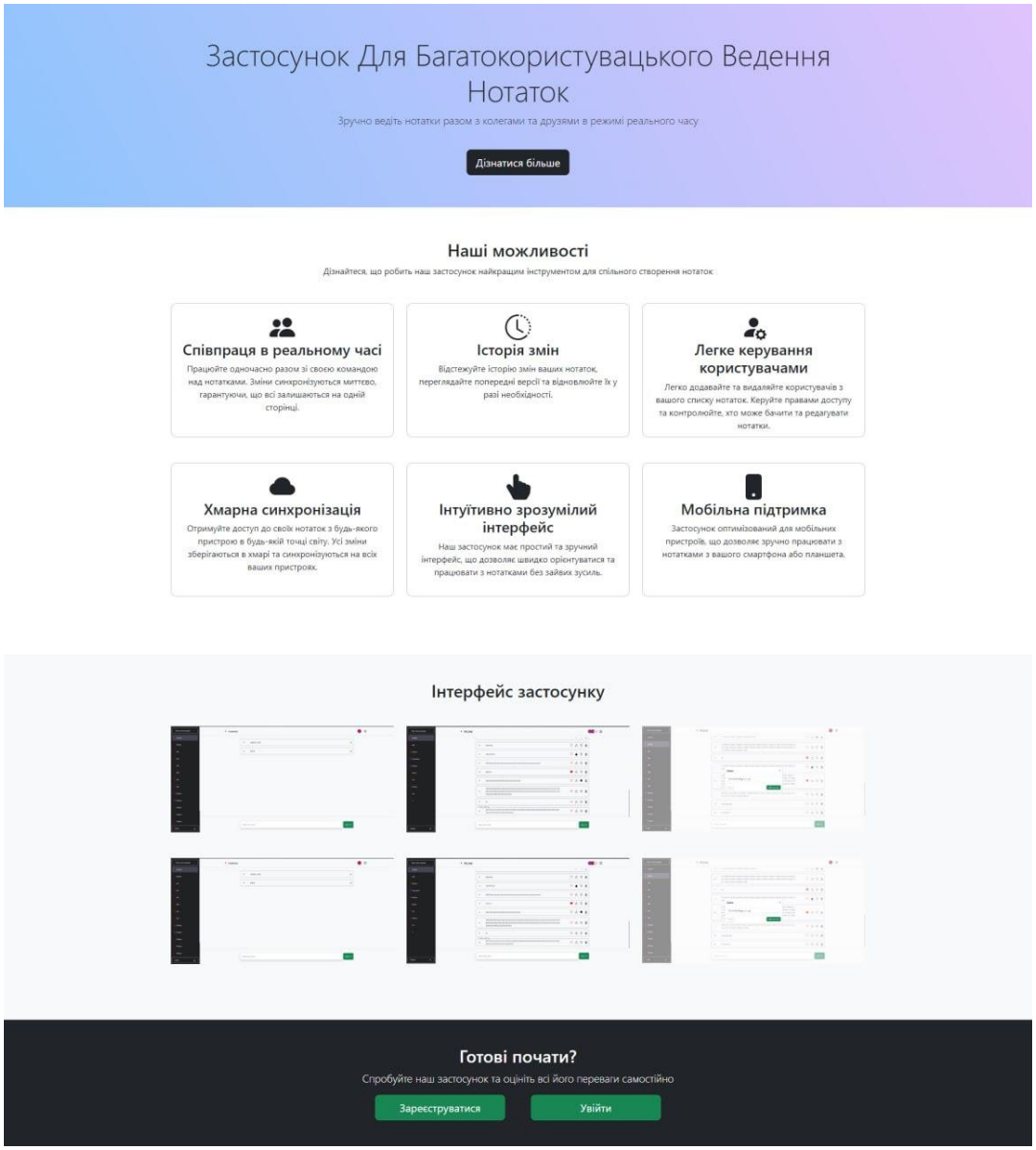

Рисунок 4.1 – Головна сторінка вебзастосунку

Вона представляє його основні функції та переваги, акцентуючи на можливості спільного ведення нотаток в режимі реального часу, а також підкреслює зручність керування користувачами, відстеження історії змін, хмарну синхронізацію та підтримку мобільних пристроїв, а також простий та інтуїтивно зрозумілий інтерфейс. Сторінка також закликає користувачів дізнатися більше про застосунок і пропонує можливість зареєструватися або увійти для початку роботи.

Сторінку входу представлено на рис. 4.2.

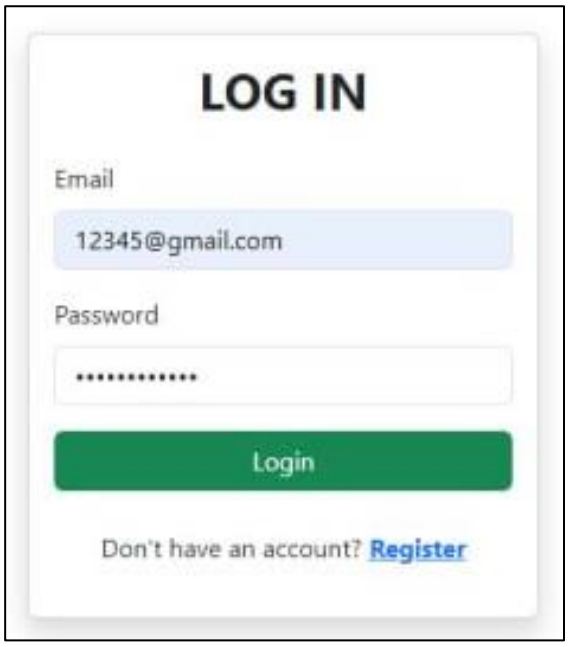

Рисунок 4.2 – Сторінка входу

Вона забезпечує користувачам можливість авторизації, запрошуючи ввести адресу електронної пошти та пароль для доступу до своїх облікових записів. Для нових користувачів передбачено посилання на реєстрацію, яке дозволяє створити новий обліковий запис, якщо його ще немає.

Реалізацію сторінки реєстрації представлено на рис. 4.3.

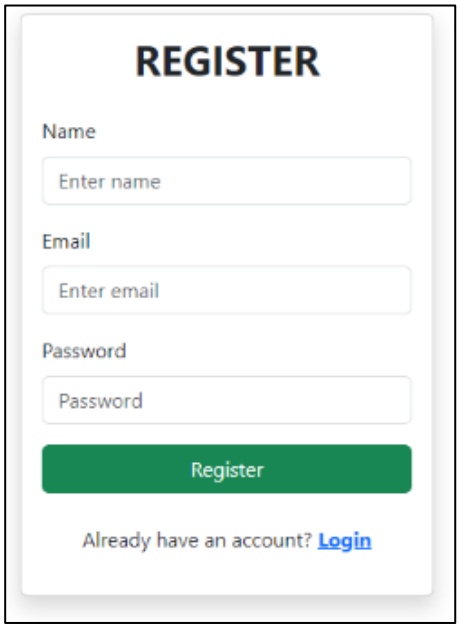

Рисунок 4.3 – Сторінка реєстрації

Сторінка призначена для нових користувачів, які бажають створити обліковий запис. Вона забезпечує можливість введення необхідних даних, таких як ім’я, електронна пошта та пароль, для створення облікового запису. Після введення даних користувач може завершити процес реєстрації. Крім того, сторінка містить посилання для тих, хто вже має обліковий запис, що дозволяє їм перейти до сторінки входу в систему.

Сторінку вибору списку нотаток представлено на рис. 4.4.

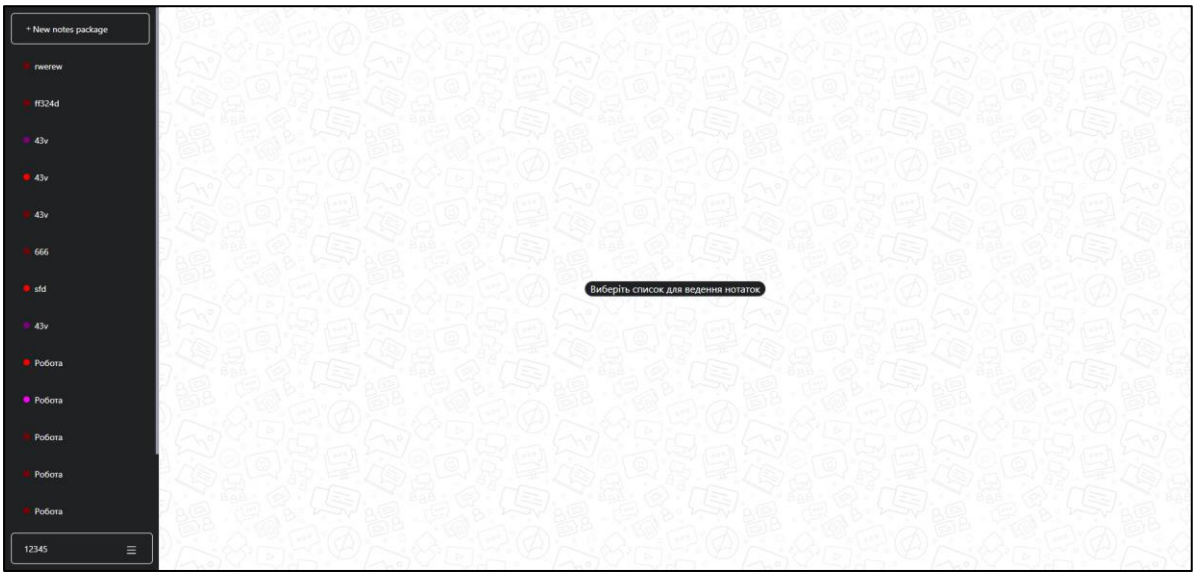

Рисунок 4.4 – Сторінка вибору списку нотаток

Ця сторінка надає користувачам можливість переглядати та вибирати різні списки нотаток для подальшого перегляду або редагування. Ліва частина інтерфейсу містить панель зі списком доступних списків нотаток, кожен з яких позначений кольоровими індикаторами для швидкої ідентифікації. Також присутня кнопка для створення нового списку нотаток. Центральна частина сторінки призначена для відображення вмісту обраного списку, але поки не вибрано жодного списку, вона показує повідомлення з підказкою.

При натисканні на кнопку для створення нового списку нотаток на панелі навігації, що знаходиться зліва, відкриється модальне вікно, яке представлено на рис. 4.5.

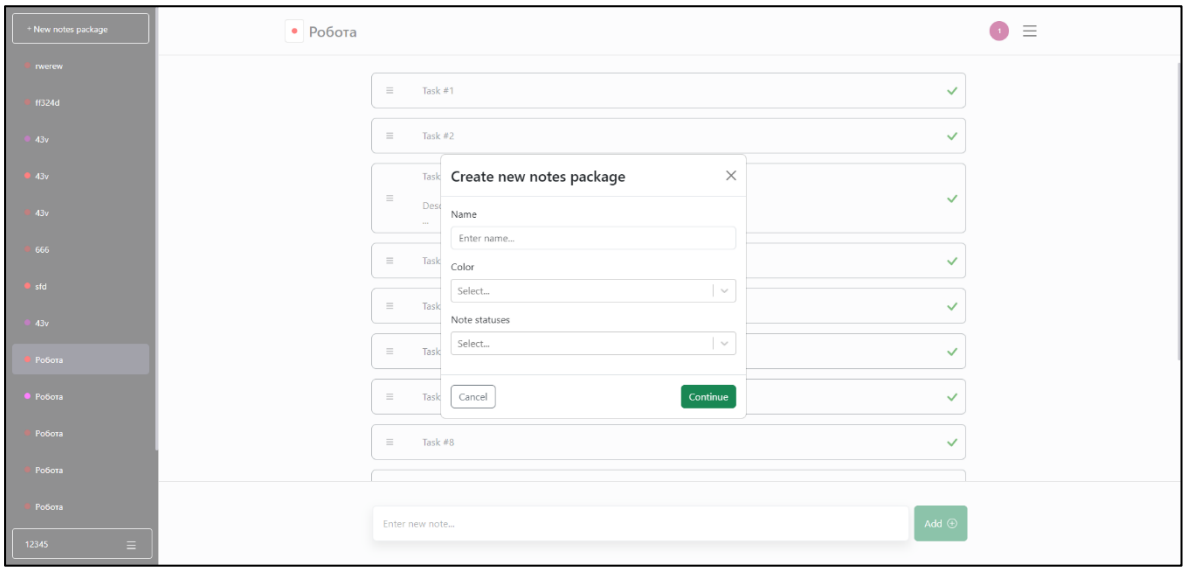

Рисунок 4.5 – Модальне вікно для створення нового списку нотаток

На формі відображаються поле для введення назви нового списку, випадаючі меню для вибору кольору та варіації статусів нотаток. Нижче знаходяться кнопки для підтвердження створення нового списку нотаток та для закриття вікна без збереження змін.

Сторінка обраного звичайного списку нотаток представлена на рис. 4.6.

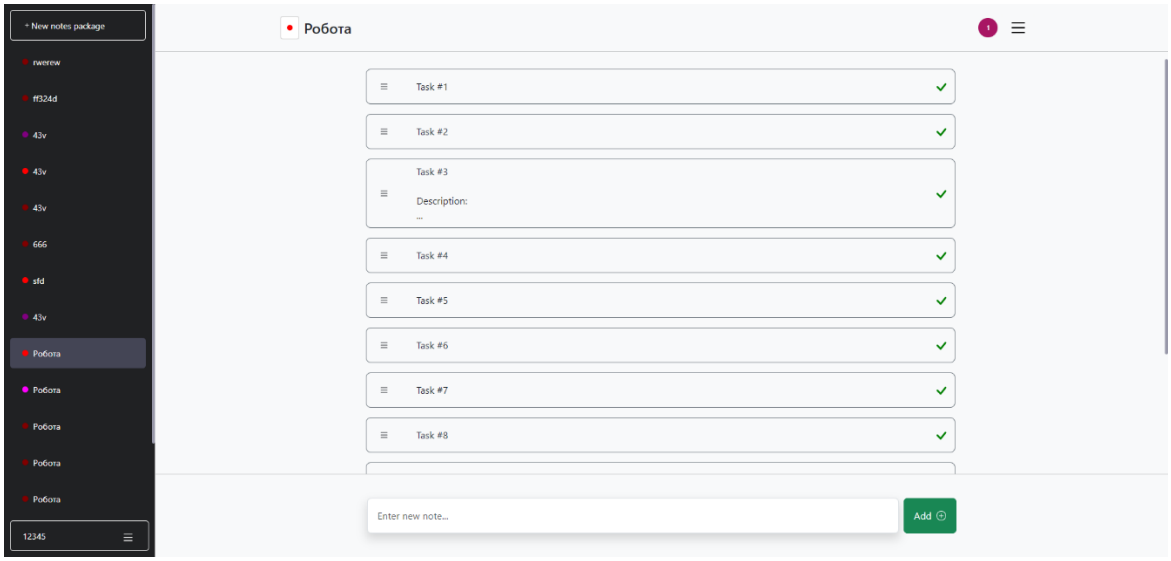

Рисунок 4.6 – Сторінка обраного звичайного списку нотаток

Вона демонструє вміст конкретного списку нотаток, обраного користувачем з панелі ліворуч. Вибраний список містить назву, колір та перелік нотаток, що в ньому зберігається. Кожна нотатка має поле для опису та відзначення виконання

завдання, тобто видалення цієї нотатки. Описані параметри можна редагувати. Також користувач може додавати нові нотатки через текстове поле внизу сторінки.

Сторінка обраного списку нотаток зі статусами представлена на рис. 4.7.

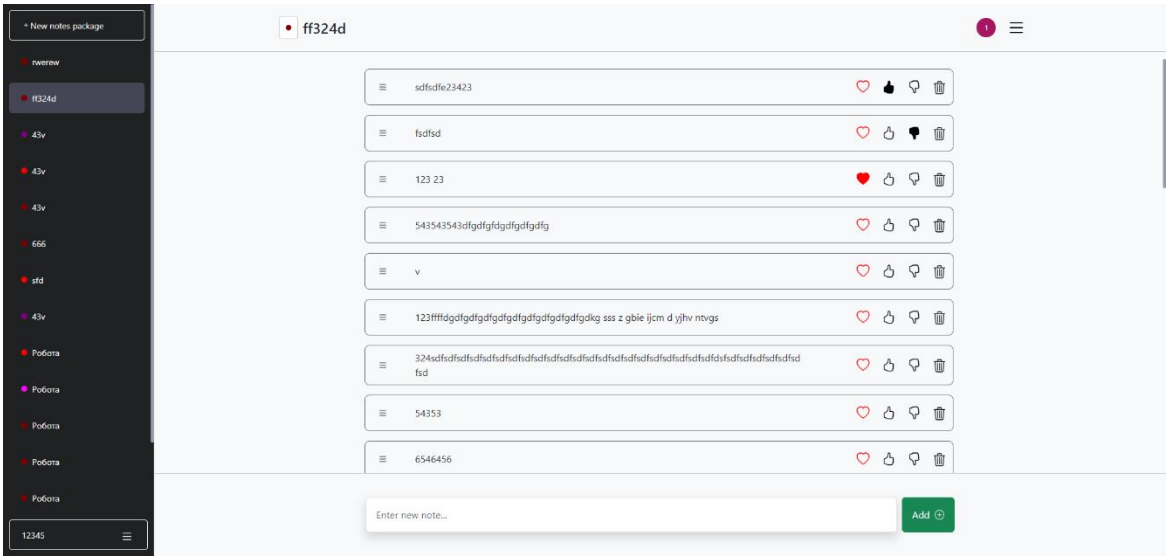

Рисунок 4.7 – Обраний список нотаток зі статусами

Сторінка схожа на попередньо описану на рис. 4.6, але відрізняється можливістю відзначати кожну з нотаток своїм статусом. В наведеному прикладі користувач має можливість позначати нотатку як важливу, ставити «лайки» або «дизлайки».

Реалізоване модальне вікно для перегляду користувачів списку нотаток представлено на рис. 4.8.

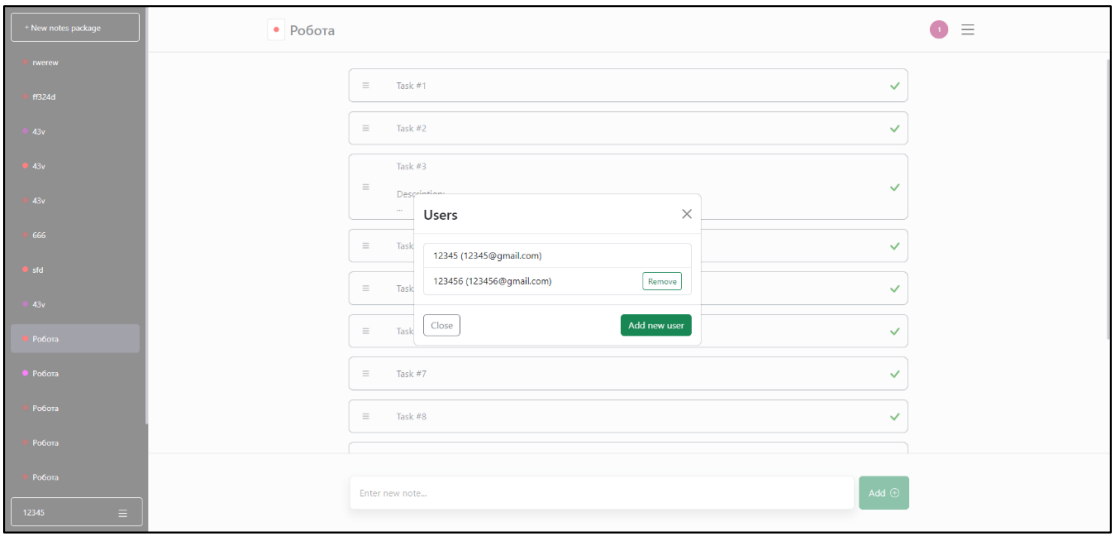

Рисунок 4.8 – Модальне вікно для перегляду користувачів списку нотаток

На ньому відображаються всі користувачі, які мають доступ до певного списку нотаток. А також за допомогою цього ж меню користувачі мають можливість додавати нових учасників або видаляти вже існуючих в обраному списку нотаток.

Модальне вікно для додавання нових користувачів до списку нотаток представлено на рис. 4.9.

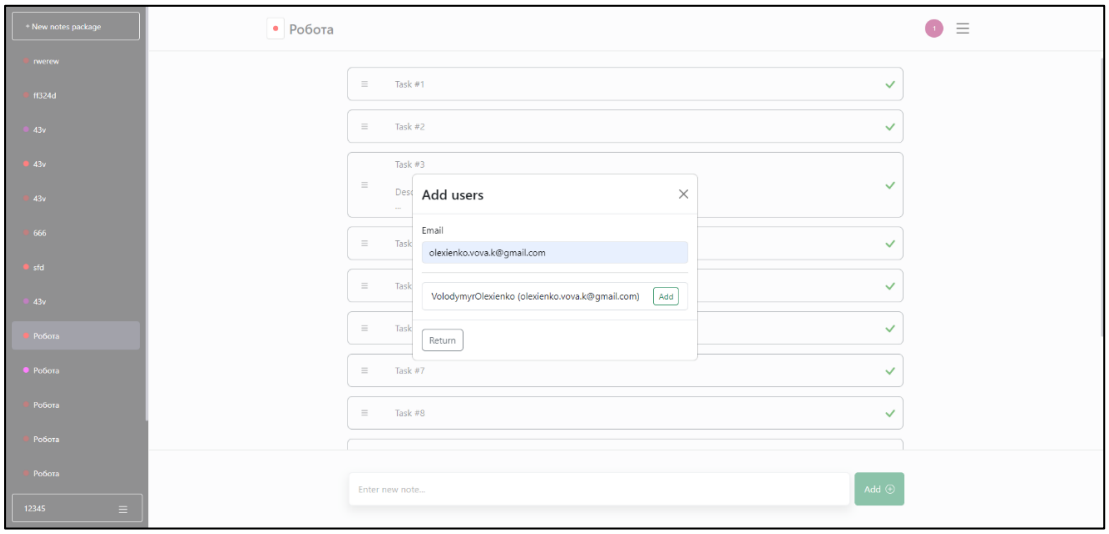

Рисунок 4.9 – Модальне вікно для додавання нового користувача до списку нотатків

Воно містить поле для введення електронної пошти користувача, якого потрібно додати. При введенні даних буде здійснюватися пошук користувачів за цією електронною поштою і відповідно знайдені користувачі виводитися списком з кнопками для підтвердження додавання. Після натискання кнопки «Додати», новий користувач буде доданий до списку. Також є кнопка «Повернутись», яка дозволяє закрити модальне вікно без внесення змін.

Висновки до розділу 4

У четвертому розділі кваліфікаційної роботи представлено відображення розробленого вебзастосунку для багатокористувацького ведення нотаток.

Продемонстровано ключові елементи інтерфейсу та основні функції вебзастосунку. Застосунок забезпечує користувачам інтуїтивний доступ до

створення, редагування та управління нотатками після проходження реєстрації та авторизації.

Структура інтерфейсу сприяє зручності використання, дозволяючи легко організовувати і відстежувати нотатки, а також ефективно співпрацювати з іншими користувачами.

Отже, розроблений вебзастосунок відповідає потребам користувачів у простому та ефективному управлінні нотатками, забезпечуючи широкий спектр функцій для різних сценаріїв використання, а під час розробки виконано всі поставлені задачі.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи розроблено вебзастосунок для багатокористувацького ведення нотаток. Було виконано всі поставлені завдання.

Для аналізу сучасного стану вебзастосунків для управління нотатками проведено дослідження предметної області та відповідно існуючих систем, що дозволило виявити вдалі рішення, які можна використати під час розробки власної платформи. На основі цього аналізу сформульовано задачі для створення вебзастосунку з функціями створення, редагування, видалення нотаток, групування їх у списки, відзначення важливих нотаток та багатокористувацьким доступом.

Розробка архітектури та функціоналу вебзастосунку виявилося комплексним завданням, що вимагало ретельного аналізу функціональних та структурних аспектів системи. Розроблено діаграми варіантів використання, класів, послідовності дій, пакетів. Це забезпечило якість та ефективність розробки, дозволило чітко визначити функціональні можливості та структуру системи, сприяло збереженню чіткості та легкості управління кодом, а також спростило розуміння логіки програми.

Для кодування застосунку вибрано та досліджено наступні технології розробки: Java, Spring Framework, TypeScript, React, MongoDB та Docker. Докладно описано структуру баз даних. Представлено п’ять мокапів, що надають візуальне уявлення про інтерфейс користувача та його функціональність.

Продемонстровано реалізацію інтерфейсу користувача та основних функцій програми. Структура інтерфейсу сприяє зручності використання, дозволяючи легко організовувати і відстежувати нотатки, а також ефективно співпрацювати з іншими користувачами.

Таким чином, розроблений вебзастосунок відповідає потребам користувачів у простому та ефективному управлінні нотатками, забезпечуючи широкий спектр функцій для різних сценаріїв використання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Google Keep : вебсайт. URL : https://keep.google.com (Last accessed: 01.04.2024)

2. Microsoft OneNote : вебсайт. URL : https://www.onenote.com (Last accessed: 03.04.2024)

3. Google Keep : вебсайт. URL : https://www.notion.so (Last accessed: 05.04.2024)

4. Knudsen J., Niemeyer P. Learning Java : підручник. O'Reilly Media, Inc., 2005. 856 p.

5. Evans B., Flanagan D. Java in a Nutshell: A Desktop Quick Reference : підручник. O'Reilly Media, 2018. 456 p.

6. Java docs : вебсайт. URL : https://docs.oracle.com/en/java (Last accessed: 10.05.2024)

7. Bloch J. Effective Java : підручник. Addison-Wesley Professional, 2018. 412 p.

8. Walls C. Spring in Action : підручник. Manning Publications, 2018. 520 p.

9. Walls C. Spring Boot in Action : підручник. Manning Publications, 2016. 264 p.

10. Spring docs : вебсайт. URL : https://docs.spring.io/spring-framework/reference (Last accessed: 15.05.2024)

11. Typescript docs : вебсайт. URL: https://www.typescriptlang.org/docs (Last accessed: 18.05.2024)

12. Vanderkam D. Effective TypeScript: 62 Specific Ways to Improve Your TypeScript : підручник. O'Reilly Media, Incorporated, 2019. 250 p.

13. Rozentals N. Mastering TypeScript: Build enterprise-ready, modular web applications using TypeScript 4 and modern frameworks, 4th Edition : підручник. Packt Publishing, 2021. 538 p.

14. Programming TypeScript: Making Your JavaScript Applications Scale : підручник. O'Reilly Media, 2019. 322 p.

15. Thomas M. T. React in Action : підручник. Manning Publications, 2018. 360 p.

16. Bugl D. Learn React Hooks: Build and Refactor Modern React. js Applications Using Hooks : підручник. Packt Publishing, Limited, 2019. 426 p.

17. React docs : вебсайт. URL: https://react.dev/learn (Last accessed: 20.05.2024)

18. MongoDB docs : вебсайт. URL: https://www.mongodb.com/docs (Last accessed: 25.05.2024)

19. MongoDB: The Definitive Guide, 2nd Edition : підручник. O'Reilly Media, 2013. 432 p.

20. Carlson J. L. Redis in Action : підручник. Manning Publications, 2013. 320 p.

21. Redis docs : вебсайт. URL: https://redis.io/docs/latest (Last accessed: 29.05.2024)

22. Mouat A. Using Docker: Developing and Deploying Software with Containers : підручники. O'Reilly Media, Incorporated, 2015. 358 p.

23. Docker docs : вебсайт. URL: https://docs.docker.com (Last accessed: 31.05.2024)

ДОДАТОК А

Лістинг класу NoteServiceImpl

```
@Service
@RequiredArgsConstructor
public class NoteServiceImpl implements NoteService {
   private final NotePackageRepository notePackageRepository;
   private final NoteMapper noteMapper;
   private final UserServiceClient userServiceClient;
   @Override
   public List<NotePackageShortenedDto> getNotePackages(String userId) {
      return notePackageRepository.findByUserIdsContains(userId).stream()
            .map(noteMapper::toShortenedDto)
            .toList();
   }
   @Override
   public NotePackageDto getNotePackage(String notePackageId, String userId) {
      return noteMapper.toDto(getNotePackageOrThrowException(notePackageId, userId));
   }
   @Override
   public NotePackageDto createNotePackage(CreateNotePackageDto dto, String userId) {
      NotePackage notePackage = noteMapper.toEntity(dto);
      notePackage.setUserIds(List.of(userId));
      notePackage.setNotes(List.of());
      return noteMapper.toDto(notePackageRepository.save(notePackage));
   }
   @Override
   public void updateNotePackage(NotePackageDto dto, String userId) {
```

```
        validateNoteStatus(dto);
        NotePackage oldNotePackage = getNotePackageOrThrowException(dto.getId(), userId);
        generateIdForNewNotes(dto, oldNotePackage);
        NotePackage notePackage = noteMapper.toEntity(dto);
        notePackage.setUserIds(oldNotePackage.getUserIds());
        notePackageRepository.save(notePackage);
    }
    @Override
    public void deleteNotePackage(String notePackageId, String userId) {

notePackageRepository.delete(getNotePackageOrThrowException(notePackageId, userId));
    }
    @Override
    public void addUserToNotePackage(String notePackageId, String ownerUserId, String guestUserId) {
        NotePackage notePackage = getNotePackageOrThrowException(notePackageId, ownerUserId);
        if (notePackage.getUserIds().contains(guestUserId)) {
            throw new ValidationException(String.format("User with id = [%s] already added to notePackage with id = [%s]", guestUserId, notePackageId));
        }
        notePackage.getUserIds().add(guestUserId);
        notePackageRepository.save(notePackage);
    }
    @Override
    public List<UserDto> getNotePackageUsers(String notePackageId, String userId) {
        List<String> userIds = getNotePackageOrThrowException(notePackageId, userId).getUserIds();
        return userServiceClient.getUsersByIds(userIds);
```

```
    }

    @Override
    public void deleteUserFromNotePackage(String notePackageId, String userId, String guestUserId) {
        NotePackage notePackage = getNotePackageOrThrowException(notePackageId, userId);
        notePackage.getUserIds().removeIf(guestUserId::equals);
        if (CollectionUtils.isNotEmpty(notePackage.getUserIds())) {
            notePackageRepository.save(notePackage);
        } else {
            notePackageRepository.delete(notePackage);
        }
    }

    private NotePackage getNotePackageOrThrowException(String notePackageId, String userId) {
        return notePackageRepository.findByIdAndUserIdsContains(new ObjectId(notePackageId), List.of(userId))
                .orElseThrow(() -> new ValidationException(String.format("NotePackage with id = [%s] is not found for user with id = [%s]", notePackageId, userId)));
    }

    private void generateIdForNewNotes(NotePackageDto dto, NotePackage oldNotePackage) {
        dto.getNotes().stream()
                .filter(noteDto -> isNoteNotYetCreated(oldNotePackage, noteDto))
                .forEach(noteDto -> noteDto.setId(ObjectId.get().toString()));
    }

    private boolean isNoteNotYetCreated(NotePackage oldNotePackage, NoteDto noteDto) {
        return CollectionUtils.emptyIfNull(oldNotePackage.getNotes()).stream()
                .noneMatch(note -> note.getId().toString().equals(noteDto.getId()));
    }
```

```
    private void validateNoteStatus(NotePackageDto dto) {
        List<NoteStatus> noteStatuses = Arrays.asList(dto.getStatusPackage().getNoteStatuses());
        CollectionUtils.emptyIfNull(dto.getNotes()).stream()
                .filter(noteDto -> Objects.nonNull(noteDto.getStatus()) && !noteStatuses.contains(noteDto.getStatus()))
                .findFirst()
                .ifPresent(noteDto -> {
                    throw new ValidationException(String.format("Note with id = [%s] in notePackage with statusPackage = [%s] cant have status = [%s]", noteDto.getId(), dto.getStatusPackage(), noteDto.getStatus()));
                });
    }
}
```

ДОДАТОК Б

Лістинг класу UserServiceImpl

```
@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {
    private final UserRepository userRepository;
    private final AuthorizationUtil authorizationUtil;
    private final UserServiceMapper userServiceMapper;
    private final PasswordEncoder passwordEncoder;
    private final Random random = new SecureRandom();
    @Override
    public String register(RegisterDto dto) {
        User user = userServiceMapper.toEntity(dto);
        user.setSalt(generateSalt());
        user.setHash(passwordEncoder.encode(dto.getPassword() + user.getSalt()));
        return
authorizationUtil.generateJwtToken(userRepository.save(user).getId().toString());
    }
    @Override
    public String login(LoginDto dto) {
        User user =
userRepository.findByEmailIn(List.of(dto.getEmail())).stream().findFirst()
                .orElseThrow(() -> new ValidationException("Authentication failed"));
        if (!passwordEncoder.matches(dto.getPassword() + user.getSalt(), user.getHash()))
{
            throw new ValidationException("Authentication failed");
        }
        return authorizationUtil.generateJwtToken(user.getId().toString());
    }
```

```
    @Override
    public UserDto getUser(String userId) {
        User user = userRepository.findById(userId).orElseThrow();
        return userServiceMapper.toDto(user);
    }
    @Override
    public List<UserDto> getUsersByEmails(List<String> emails) {
        List<User> users = userRepository.findByEmailIn(emails);
        return users.stream()
                .map(userServiceMapper::toDto)
                .toList();
    }
    @Override
    public List<UserDto> getUsersByIds(List<String> ids) {
        List<User> users = userRepository.findByIdIn(ids.stream().map(ObjectId::new).toList());
        if (users.size() != ids.size()) {
            throw new ValidationException("Some users was not found by id");
        }
        return users.stream()
                .map(userServiceMapper::toDto).toList();
    }
    public String generateSalt() {
        byte[] salt = new byte[16];
        random.nextBytes(salt);
        return Base64.getEncoder().encodeToString(salt);
    }
}
```

ДОДАТОК В

Лістинг класу AuthorizationUtil

```
@Component
@RefreshScope
public class AuthorizationUtil {
   @Value("${jwt.secret.key}")
   private String jwtSecretKey;
   @Value("${jwt.token.validity}")
   private long jwtTokenValidity;
   @Value("${authorization.cookie.validity}")
   private int cookieValidity;
   public String generateJwtToken(String subject) {
      long currentTimeMillis = System.currentTimeMillis();
      return Jwts.builder()
            .subject(subject)

.signWith(Keys.hmacShaKeyFor(Base64.getEncoder().encode(jwtSecretKey.getBytes(
))))//Jwts.SIG.HS512.key().build()
            .issuedAt(new Date(currentTimeMillis))
            .expiration(new Date(currentTimeMillis + jwtTokenValidity))
            .compact();
   }
   public Cookie createAuthorizationCookie(String jwtToken) {
      Cookie authorizationCookie = new Cookie("Authorization", jwtToken);
      authorizationCookie.setMaxAge(cookieValidity);
      authorizationCookie.setHttpOnly(true);
      authorizationCookie.setPath("/");
      return authorizationCookie;
   }
```

```
    public Cookie deleteAuthorizationCookie() {
        Cookie authorizationCookie = new Cookie("Authorization", null);
        authorizationCookie.setMaxAge(0);
        authorizationCookie.setHttpOnly(true);
        authorizationCookie.setPath("/");
        return authorizationCookie;
    }
}
```

ДОДАТОК Г

Лістинг класу JwtAuthenticationFilter

```
@Service
@RefreshScope
public class JwtAuthenticationFilter implements GatewayFilter {
   private static final List<String> AUTH_API_ENDPOINTS = List.of("/register", "/login");
   @Value("${jwt.secret.key}")
   private String jwtSecretKey;
   @Override
   public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
      ServerHttpRequest request = exchange.getRequest();
      ServerHttpResponse response = exchange.getResponse();
      if (isSecuredEndpoint(request)) {
         String jwtToken = getJwtTokenFromCookies(request);
         try {
            request.mutate().header("userId", String.valueOf(getClaims(jwtToken).getSubject())).build();
         } catch (Exception e) {
            response.setStatusCode(HttpStatus.UNAUTHORIZED);
            return response.setComplete();
         }
      }
      return chain.filter(exchange);
   }
   private boolean isSecuredEndpoint(ServerHttpRequest request) {
      return AUTH_API_ENDPOINTS.stream().noneMatch(uri -> request.getURI().getPath().contains(uri));
```

```
    }

    private String getJwtTokenFromCookies(ServerHttpRequest request) {
        return Optional.ofNullable(request.getCookies().getFirst("Authorization"))
                .map(HttpCookie::getValue)
                .orElse(null);
    }
    public Claims getClaims(String token) {
        return Jwts.parser()

.verifyWith(Keys.hmacShaKeyFor(Base64.getEncoder().encode(jwtSecretKey.getBytes())))
                .build()
                .parseSignedClaims(token)
                .getPayload();
    }
}
```

ДОДАТОК Д

Лістинг класу ActionServiceImpl

```
@Service
@RequiredArgsConstructor
public class ActionServiceImpl implements ActionService {
   private final CacheService cacheService;
   @Override
   public ActionDto updateAndGetActiveUsers(CreateActionDto dto, ActionType actionType, SimpMessageHeaderAccessor headerAccessor) {
      String userId = (String) headerAccessor.getSessionAttributes().get(USER_ID);
      Set<String> activeUsers = cacheService.getActiveUsers(dto.getHubId(), new HashSet<>());
      if (ActionType.LEAVE == actionType) {activeUsers.remove(userId);}
        else {activeUsers.add(userId);}
      cacheService.putActiveUsers(dto.getHubId(), activeUsers);
      ActionDto actionDto = buildActionDto(actionType, userId, activeUsers);
      if (dto instanceof CreateObjectActionDto createObjectActionDto) {
         actionDto.setObjectId(createObjectActionDto.getObjectId());
      }
      return actionDto;
   }
   private ActionDto buildActionDto(ActionType actionType, String userId, Set<String> activeUsers) {
      return ActionDto.builder()
            .actionType(actionType)
            .senderId(userId)
            .activeUserIds(activeUsers)
            .build();
   }
}
```

ДОДАТОК Е

Лістинг класу ActionController

```
@Controller
@Slf4j
@RequiredArgsConstructor
public class ActionController {
    private final SimpMessagingTemplate messagingTemplate;
    private final ActionService actionService;
    @MessageMapping("/join")
    public void join(@Payload CreateActionDto dto, SimpMessageHeaderAccessor
headerAccessor) {
        fillSessionAttributesAndSendMessage(headerAccessor, dto, ActionType.JOIN);
    }
    @MessageMapping("/chat")
    public void chat(@Payload CreateActionDto dto, SimpMessageHeaderAccessor
headerAccessor) {
        fillSessionAttributesAndSendMessage(headerAccessor, dto, ActionType.CHAT);
    }
    @MessageMapping("/edit")
    public void edit(@Payload CreateObjectActionDto dto,
SimpMessageHeaderAccessor headerAccessor) {
        fillSessionAttributesAndSendMessage(headerAccessor, dto, ActionType.EDIT);
    }
    @MessageMapping("/move")
    public void move(@Payload CreateObjectActionDto dto,
SimpMessageHeaderAccessor headerAccessor) {
        fillSessionAttributesAndSendMessage(headerAccessor, dto, ActionType.MOVE);
    }
    private void fillSessionAttributesAndSendMessage(SimpMessageHeaderAccessor
headerAccessor, CreateActionDto request, ActionType actionType) {
```

```
        headerAccessor.getSessionAttributes().put(HUB_ID, request.getHubId());

        ActionDto response = actionService.updateAndGetActiveUsers(request,
actionType, headerAccessor);

        messagingTemplate.convertAndSendToUser(request.getHubId(), ACTION,
response);

    }

}
```

ДОДАТОК Ж

Лістинг класу WebSocketEventListener

```
@Component
@RequiredArgsConstructor
public class WebSocketEventListener {
   private final SimpMessageSendingOperations messagingTemplate;
   private final ActionService actionService;
   @EventListener
   public void handleWebSocketDisconnectListener(SessionDisconnectEvent event) {
      StompHeaderAccessor headerAccessor = StompHeaderAccessor.wrap(event.getMessage());

Optional.ofNullable(headerAccessor.getSessionAttributes()).ifPresent(sessionAttributes -> {
         String hubId = (String) sessionAttributes.get(HUB_ID);
         if (Objects.nonNull(hubId)) {
            ActionDto dto = actionService.updateAndGetActiveUsers(new CreateActionDto(hubId), ActionType.LEAVE, headerAccessor);
            messagingTemplate.convertAndSendToUser(hubId, ACTION, dto);
         }
      });
   }
}
```

ДОДАТОК И

Лістинг компонента NotePackages

```
export const NotePackages = () => {
  const [toggled, setToggled] = useState(false);
  const [broken, setBroken] = useState(false);
  let { data } = notePackageApi.useGetNotePackagesQuery();
  const modalStruct = useAppModal('createNotePackage');
  const handleMenuClose = () => {
    setToggled(false);
  };
  const handleMenuOpen = () => {
    setToggled(true);
  };
  const handleNewPackageCreate = () => {
    modalStruct.openModal();
  };
  const menuItemStyles: MenuItemStyles = {
    root: {
      fontSize: '14px',
      fontWeight: 400,
    },
    button: {
      margin: '10px',
      borderRadius: '5px',
      '&:hover': {
        backgroundColor: '#2d2e39',
      },
      '&.active': {
        backgroundColor: '#444555',
      },
```

```
  },
 };
 return (
  <>
   <div className="d-flex">
    <Sidebar
     toggled={toggled}
     onBreakPoint={setBroken}
     onBackdropClick={handleMenuClose}
     breakPoint="sm"
     backgroundColor="#202123"
     rootStyles={{
      color: 'white',
     }}
     className="fixed-bottom"
     style={{ height: '92vh' }}>
     <Menu menuItemStyles={menuItemStyles}>
      <CreateItem handleNewPackageCreate={handleNewPackageCreate} />
      {data?.map((notePackage) => (
       <Item key={notePackage.id} notePackage={notePackage}
handleMenuClose={handleMenuClose} />
      ))}
     </Menu>
    </Sidebar>
    <Sidebar
     toggled={toggled}
     onBreakPoint={setBroken}
     onBackdropClick={handleMenuClose}
     breakPoint="sm"
     backgroundColor="#202123"
```

```
        rootStyles={{
          color: 'white',
          position: 'fixed',
        }}
        className="fixed-bottom"
        style={{ height: '8vh' }}>
        <Menu menuItemStyles={menuItemStyles} style={{ bottom: 0 }}
className="position-absolute w-100">
          <UserItem />
        </Menu>
      </Sidebar>
    </div>
    {broken && <button onClick={handleMenuOpen}></button>}
    <CreateNotePackageModal />
    <Outlet />
  </>
 );
};
```

ДОДАТОК К

Лістинг компонента Notes

```
export const Notes = ({
  onSave,
  onEditing,
  onMoving,
  isLoading,
  processedNotes,
}: {
  onSave: () => Promise<void>;
  isLoading: boolean;
  onEditing: (objectId: string) => void;
  onMoving: (objectId: string) => void;
  processedNotes: { userName: string; noteId: string; actionType: ActionType }[];
}) => {
  const deleteNoteModalStruct = useAppModal('deleteNote');
  const { watch } = useFormContext<NotePackageDto>();
  const { move, update } = useFieldArray({
    name: 'notes',
    keyName: '_id',
  });
  const notes = watch('notes');
  const onMovingThrottled = useThrottledCallback((update) => onMoving(update.draggableId), 2500);
  const updateNoteStatus = (id: string, noteStatus: NoteStatus) => async () => {
    const noteIndex = notes.findIndex((note) => note.id === id);
    update(noteIndex, { ...notes[noteIndex], status: notes[noteIndex].status !== noteStatus ? noteStatus : null });
    await onSave();
  };
```

```
  const moveNote = async (result: DropResult) => {
    if (result.destination && result.destination.index !== result.source.index) {
      move(result.source.index, result.destination.index);
      await onSave();
    }
  };
  return (
    <>{notes && (
        <DragDropContext                                    onDragEnd={moveNote}
onDragUpdate={onMovingThrottled}>
          <Droppable droppableId="notes">
            {(provided) => (
              <div ref={provided.innerRef} {...provided.droppableProps}>
                <DraggableNotes
                  isLoading={isLoading}
                  fields={notes}
                  onSave={onSave}
                  onEditing={onEditing}
                  onNoteStatusUpdate={updateNoteStatus}
                  deleteNoteModalStruct={deleteNoteModalStruct}
                  processedNotes={processedNotes}
                />
                {provided.placeholder}
              </div>
            )}
          </Droppable>
        </DragDropContext>
      )}
      <DeleteNoteModal onSave={onSave} /></>);
};
```

ДОДАТОК Л

Лістинг компонента CreateNote

```
export const CreateNote = ({ onSave, isSaving }: { onSave: () => Promise<void>;
isSaving: boolean }) => {
  const { append } = useFieldArray({
    name: 'notes',
    keyName: '_id',
  });
  const { formInstance, submitForm } = useAppForm({
    defaultValues: { name: '' },
    validationSchema: noteValidationSchema,
    onFormSubmit: async (note) => {
      append({ ...note });
      formInstance.reset();
      await onSave();
    },
  });
  return (
    <div className="h-100 bg-light d-flex align-items-center">
      <div className="w-75 mx-auto">
        <div className="w-100">
          <FormProvider {...formInstance}>
            <Form>
              <div className="d-flex">
                <FormField id="name" offsetFree errorFree>
                  <TextareaAutosize
                    disabled={isSaving}
                    placeholder="Enter new note..."
                    className="form-control p-3 shadow"
                    maxRows={3}
```

```
            style={{ resize: 'none' }}
           />
          </FormField>
          <Button
           disabled={isSaving}
           variant="success"
           onClick={submitForm}
           className="mt-auto p-3 ms-2 d-flex flex-row align-items-center">
           <span className="me-2">Add</span>
           <PlusCircle color="white" />
          </Button>
         </div>
        </Form>
       </FormProvider>
      </div>
     </div>
    </div>
  );
};
```