НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Веб-застосунок для ігрової соціальної платформи

Виконав студент IV курсу, групи ІТ-02
(шифр групи)

Данилевич Антон Павлович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Крамар Ю. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ІСТ, к.т.н. АМОНС О. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ –2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Данилевичу Антону Павловичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Веб-застосунок для ігрової соціальної платформи
керівник проєкту Крамар Юлія Михайлівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27» травня 2024 р. №2112-с

2. Термін подання студентом проєкту « 17 » червня 2024 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі.
2) Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних.
3) Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів.
4) Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна класів програмного забезпечення

3) Схема бази даних

4) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «11» березня 2024 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	18.04.2024	
3	Постановка та формалізація задачі	20.04.2024	
4	Розробка інформаційного забезпечення	22.04.2024	
5	Алгоритмізація задачі	25.04.2024	
6	Обґрунтування вибору використаних технічних засобів	28.04.2024	
7	Розробка програмного забезпечення	17.05.2024	
8	Налагодження програми	24.05.2024	
9	Виконання графічних документів	26.05.2024	
10	Оформлення пояснювальної записки	30.05.2024	
11	Подання ДП на попередній захист	02.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студент _____ Антон ДАНИЛЕВИЧ

(підпис) (ініціали, прізвище)

Керівник _____ Юлія КРАМАР

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’ятьох розділів, містить 33 таблиць, 21 рисунків та 9 джерел – загалом 56 сторінки.

Дипломний проєкт присвячений розробці вебзастосуку для ігрової соціальної платформи.

Мета — створення вебзастосунку, який забезпечить користувачам можливість планування та обговорення майбутніх ігрових зустрічей, надаючи якісне, швидке та надійне спілкування

Об'єкт дослідження: вебзастосунок для ігрової соціальної платформи

Предмет дослідження: Розробка та розгортання вебзастосунку для ігрової соціальної платформи з метою обговорення обговорення та планування майбутніх ігрових зустрічей

У розділі «ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ» розглянуто та проаналізовано предметну область проєкту. Визначено програмні рішення, методології та допоміжні засоби, а також складено бізнес-процеси за допомогою BPMN діаграм. Сформульовано мету та завдання розробки дипломного проєкту.

У розділі «РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ» викладені основні вимоги до програмного забезпечення. Створено діаграму варіантів використання, описані функціональні та нефункціональні вимоги. Висунуті вимоги до серверної та клієнтської частини.

У розділі «КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ» розроблено архітектуру програмного забезпечення, діаграму класів та структуру бази даних. Обґрунтовано вибір засобів розробки.

У розділі «АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ» розглянуто методи аналізу коду. Проведено мануальна тестування та розглянуто контрольний приклад, що підтвердив правильне функціонування програмного забезпечення.

У розділі «РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ» було описано процес розгортання програмного забезпечення.

Проєкт був розгорнутий з використанням Docker. Визначено вимоги, виконання яких необхідне для успішної підтримки проєкту.

КЛЮЧОВІ СЛОВА: ВБЗАСТОСУНОК, VISUAL STUDIO CODE, PYTHON, REACT, POTGRES, NEO4J, КЛІЄНТ-СЕРВЕР, DOCKER

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 33 tables, 21 figures and 9 sources – in total 56 pages.

The purpose of the diploma project is he development of a web application for a social gaming platform.

The goal is to create a web application that will provide users with the ability to plan and discuss upcoming game meetings, providing high-quality, fast and reliable communication

Research object: a web application for a social gaming platform

Subject of research: Development and deployment of a web application for a gaming social platform for the purpose of discussing and planning future gaming meetings

In the section "PRE-PROJECT SUBJECT AREA SURVEY", the project subject area is reviewed and analyzed. Software solutions, methodologies, and auxiliary tools are identified, and business processes are drawn up using BPMN diagrams. The goal and objectives of the diploma project development are formulated.

In the section "DEVELOPMENT OF SOFTWARE REQUIREMENTS", the basic requirements for the software are outlined. A diagram of use cases is created, functional and non-functional requirements are described. The requirements for the server and client parts are put forward.

In the section "SOFTWARE DESIGN AND DEVELOPMENT" the software architecture, class diagram and database structure are developed. The choice of development tools is substantiated.

In the section "QUALITY ANALYSIS AND SOFTWARE TESTING" the methods of code analysis are considered. Manual testing is performed and a control example is considered to confirm the correct functioning of the software.

In the section "DEPLOYMENT AND SUPPORT OF SOFTWARE" the process of software deployment was described. The project was deployed using Docker. The requirements for successful project support have been identified.

KEYWORDS: APPLICATION, VISUAL STUDIO CODE, PYTHON, REACT, POTGRES, NEO4J, CLIENT-SERVER, DOCKER

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-9402.045440.01.91	Технічне завдання	16	
3	А4	КПІ.ІТ-9402.045440.02.81	Пояснювальна записка	56	
4	А4	КПІ.ІТ-9402.045440.03.12	Текст програми	29	
5	А4	КПІ.ІТ-9402.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІТ-9402.045440.05.34	Керівництво користувача	8	
7	А3	КПІ.ІТ-9402.045440.06.99	Графічний матеріал	6	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-9402.045440.00.90		
Розроб.		Данилевич А. П			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Крамар Ю. М.					
Н.контр.		Ліщук К.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-02	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2024 р.

ВЕБ-ЗАСТОСУНОК ДЛЯ ІГРОВОЇ СОЦІАЛЬНОЇ ПЛАТФОРМИ

Технічне завдання

КПІ.ІТ-9402.045440.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Юлія КРАМАР

Нормоконтроль:

_____ Катерина ЛІЩУК

Виконавець:

_____ Антон ДАНИЛЕВИЧ

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ 3
2 ПІДСТАВА ДЛЯ РОЗРОБКИ 4
3 ПРИЗНАЧЕННЯ РОЗРОБКИ 5
4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 6
5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ 15
6 СТАДІЇ І ЕТАПИ РОЗРОБКИ 16
7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ 17

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-застосунок для ігрової соціальної платформи

Галузь застосування:

Наведене технічне завдання поширюється на розробку програмного забезпечення «Веб-застосунок для ігрової соціальної платформи», котра використовується для швидкого та зручного доступу користувачів до функцій пошуку користувачів, планування ігрових подій та обговорення ігор. Це ПЗ орієнтоване на геймерів, які бажають ефективно взаємодіяти, обговорювати ігрові стратегії та координувати свої дії для командної гри.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки «Веб-застосунок для ігрової соціальної платформи» є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для спрощення взаємодії геймерів в онлайн-середовищі.

Метою розробки є створення веб-застосунку, що забезпечує такі можливості: пошук користувачів для спільної гри, планування ігрових подій, організація та участь в ігрових спільнотах.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

— сторінка реєстрації (рис. 4.1);
— сторінка авторизації (рис. 4.2);
— сторінка профілю користувачів (рис. 4.3);
— сторінка спільноти (рис. 4.4) ;
— сторінка пошуку користувачів (рис. 4.5);
— сторінка перегляду запланованих подій (рис. 4.6);
— панель перегляду нотифікацій (рис. 4.7).

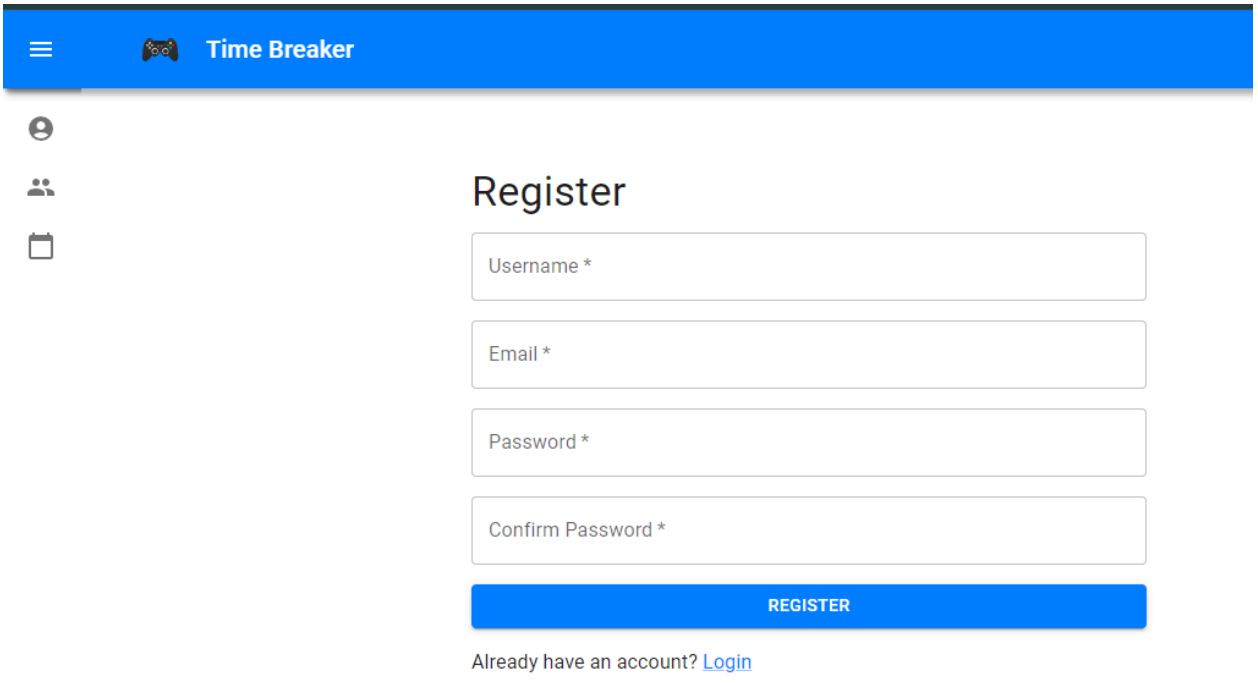


Рисунок 4.1 — Сторінка реєстрації облікового запису користувачів

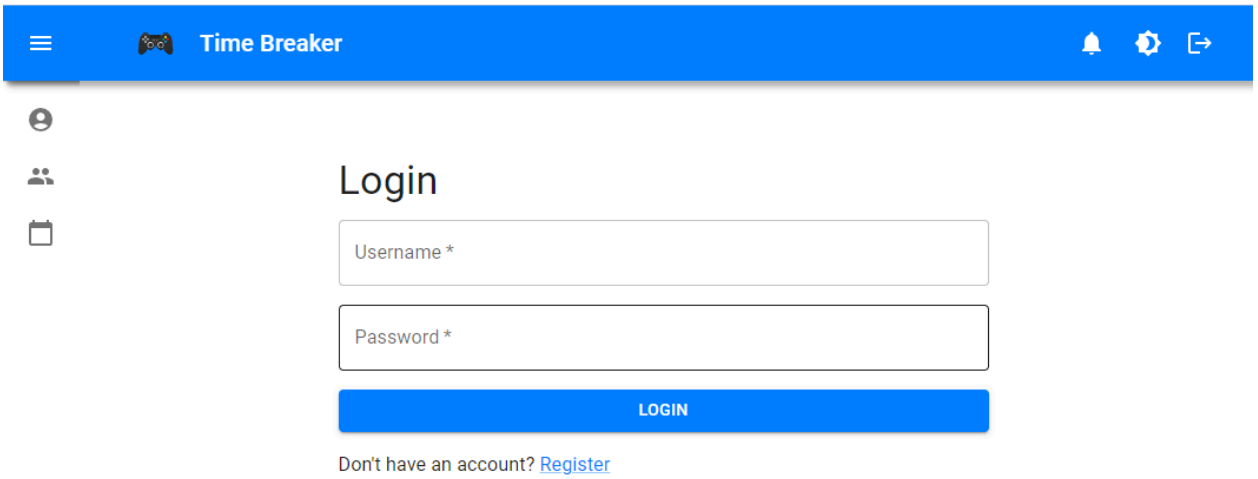


Рисунок 4.2 — Сторінка авторизації користувача

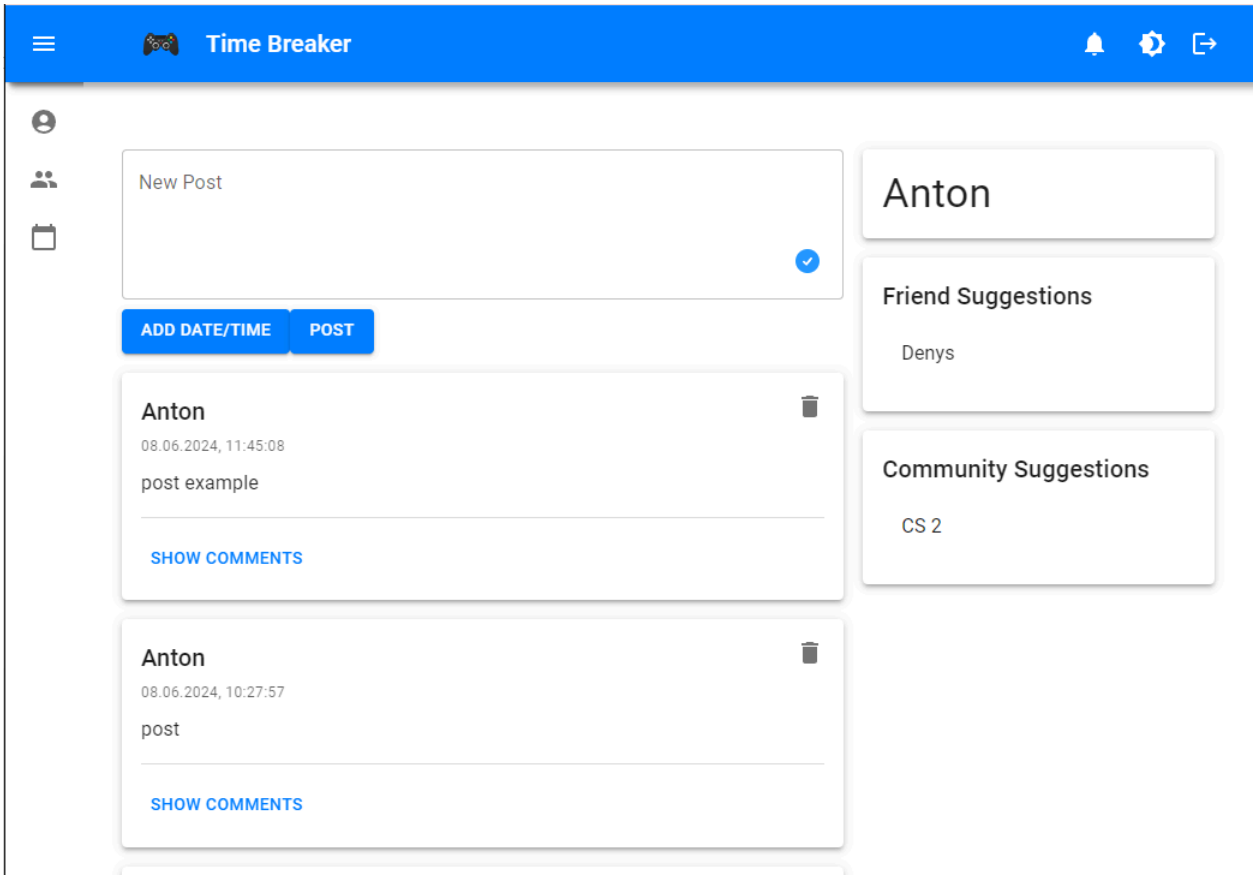


Рисунок 4.3 — Сторінка профілю користувача

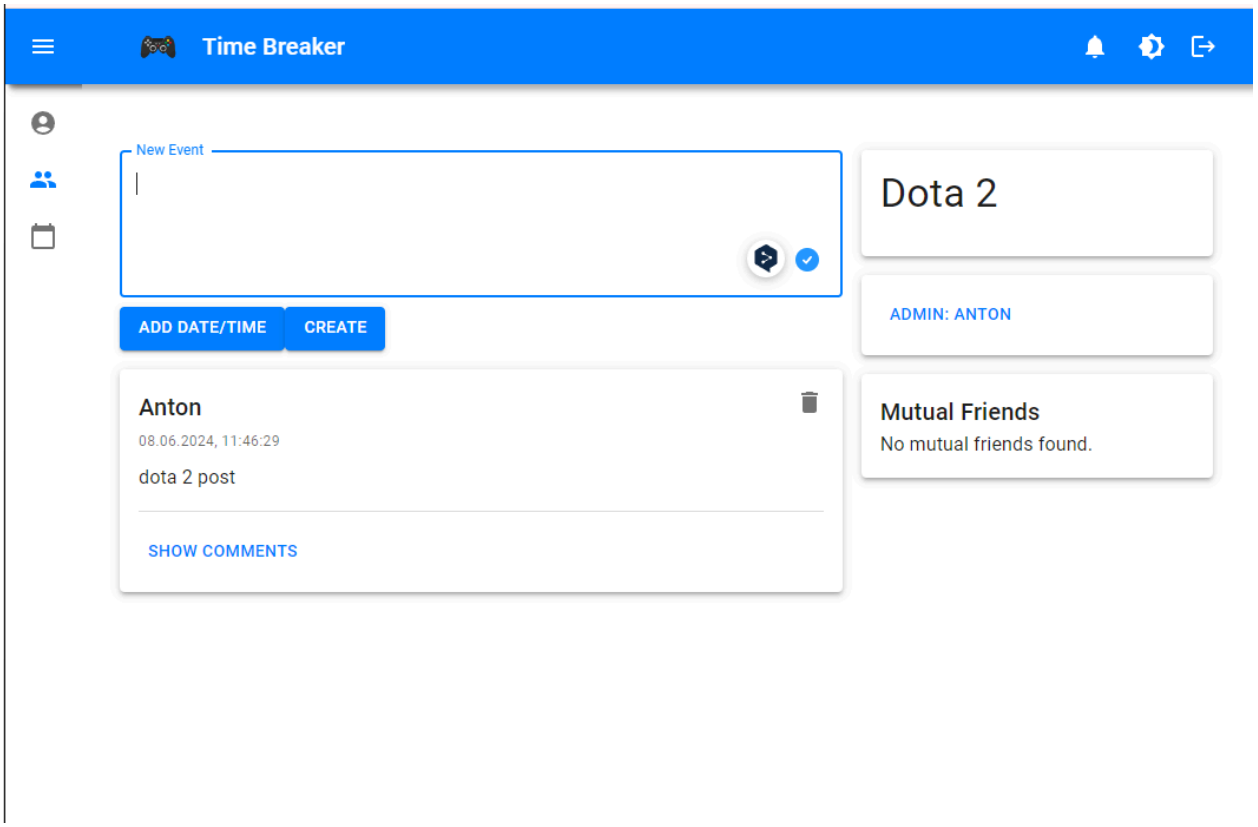


Рисунок 4.4 — Сторінка спільноти

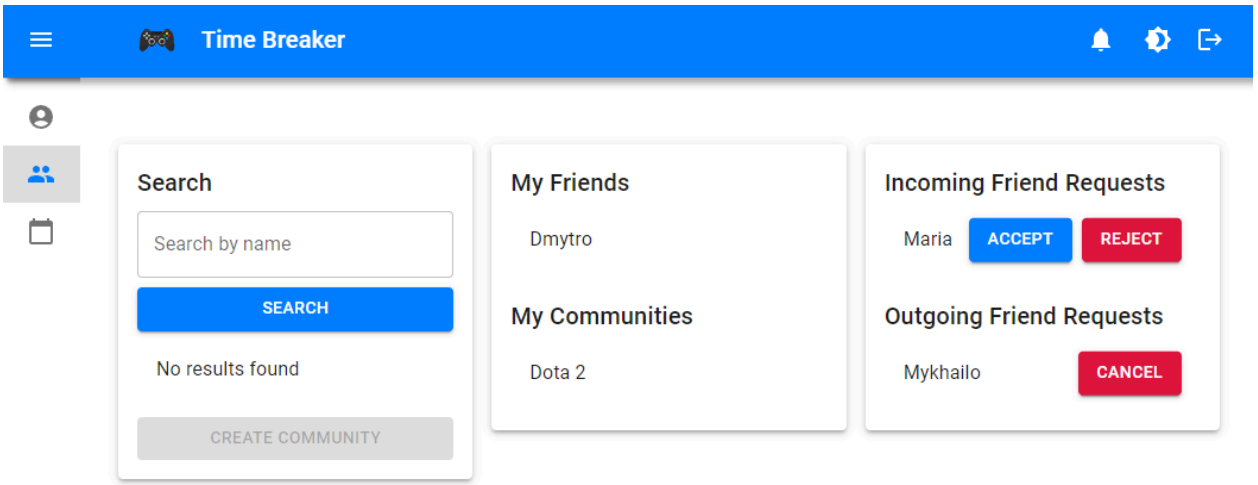


Рисунок 4.5 — Сторінка пошуку користувача

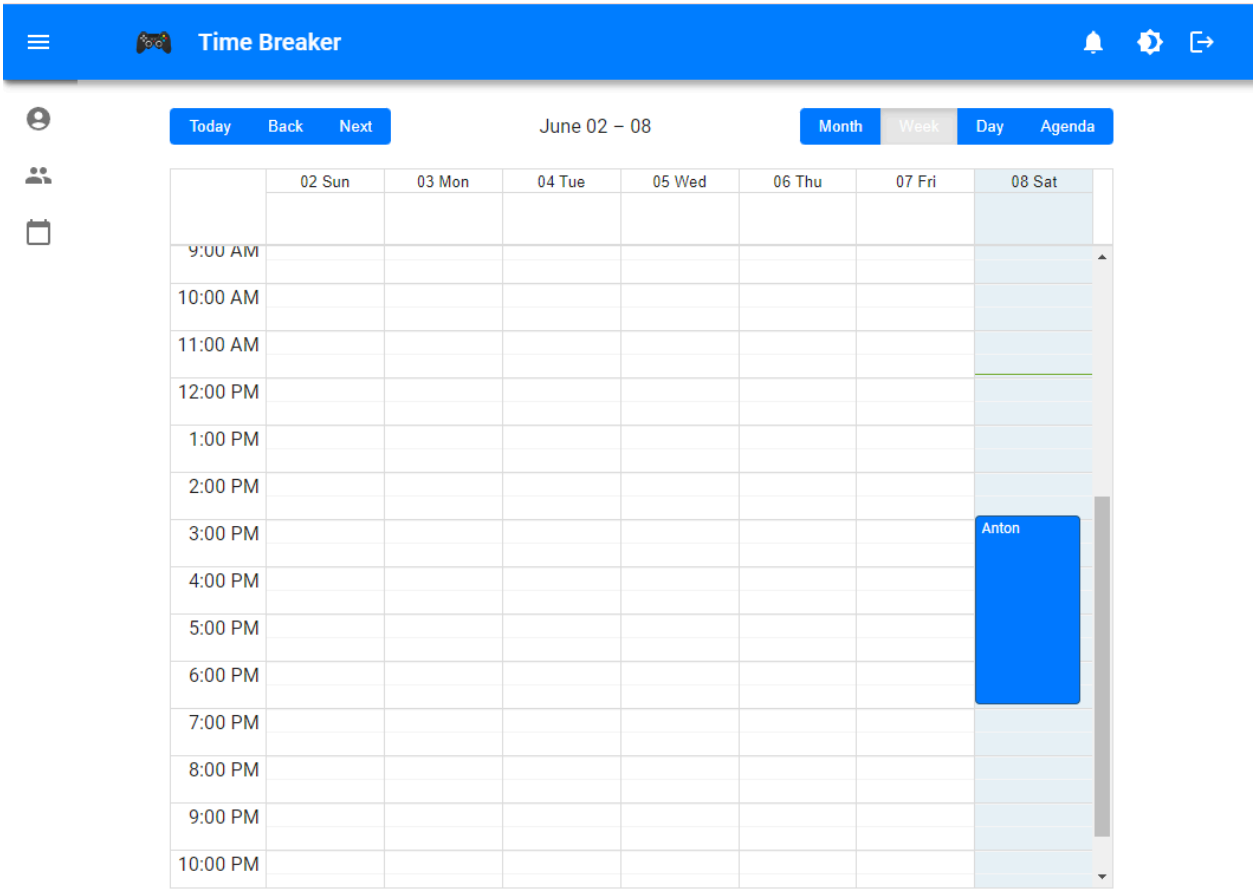


Рисунок 4.6 — Сторінка перегляду запланованих подій

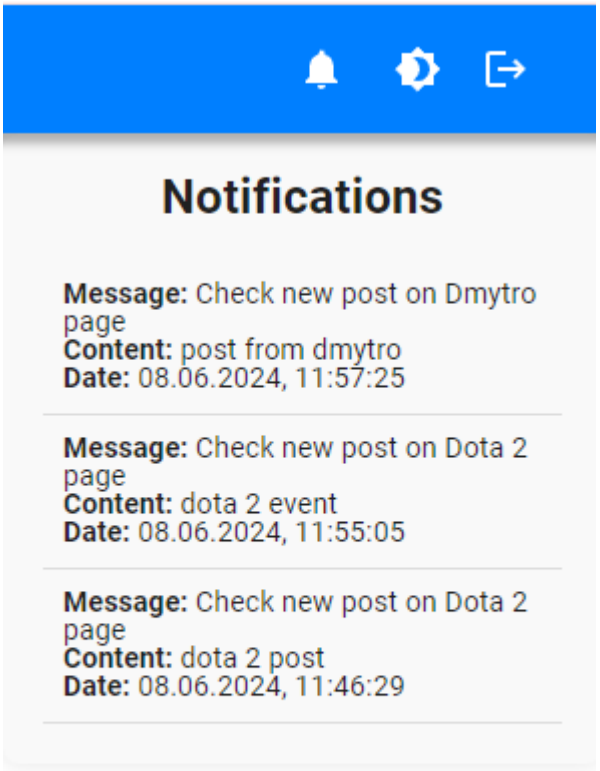


Рисунок 4.7 — Сторінка перегляду нотифікацій

4.1.2 Для користувача:

— реєстрація профілю;
— вхід в обліковий запис користувача;
— публікування постів на особистій сторінці;
— публікація постів на сторінці спільноти;
— створення особистих подій;
— створення подій спільноти;
— видалення особистих постів;
— коментування постів;
— створення спільноти;
— адміністрування спільноти;
— отримування нотифікацій;

— перегляд запланованих подій;
— отримування рекомендацій друзів;
— отримування рекомендацій спільнот;
— надсилання запитів дружби;
— отримування запитів дружби;
— прийняття запитів дружби;
— відхилення запитів дружби;
— скасовувати надсилання запиту дружби;
— видалення з друзів;
— отримування списку спільних друзів;
— підписання на спільноту;
— відписка від спільноти;
— перегляд адміністратора спільноти.

4.2 Вимоги до надійності

— передбачити контроль введення інформації та захист від некоректних дій користувачів.

— передбачити захист та цілісність даних у БД, особливо для чутливої інформації. Паролі користувачів повинні хешуватись.

— передбачити відновлення системи. Система повинна відновити свою роботу впродовж 15 хвилин після критичного збою.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Розв'язання проблем у випадку звернень користувачів. Моніторинг та аналіз роботи серверної частини.

4.3.2 Обслуговуючий персонал

Для обслуговування системи потрібен один full-stack спеціаліст зі знанням технологій python fast api та react або два спеціалісти — по одному на кожен компонент системи.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на IBM-сумісних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

— процесор: двоядерний процесор з тактовою частотою 2.0 ГГц або вище з підтримкою віртуалізації;

— жорсткий диск: 50 ГБ вільного місця;

— підключення до мережі Інтернет зі швидкістю від 100 мегабітів.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN32 (Windows XP, Windows 7, Windows 10 і новіші) або Unix (включаючи Linux та macOS).

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в наступному форматі:

Користувач — для збереження даних про користувача:

— ім’я користувача;

— хешований пароль;

— роль.

Пост — для збереження даних про пост

— автор посту;

— зміст посту;

— дата створення посту;

— час події пов’язаної з постом.

— профіль, до якого прив’язаний пост.

Коментар — для збереження даних про комента:

— автор коментаря;

— зміст коментаря;

— час коментаря;

— пост, який коментується.

4.5.2 Вимоги до вихідних даних

Вихідні дані програмного забезпечення відсутні.

4.5.3 Вимоги до мови розробки

Розробку виконати мовою програмування Python для серверної частини, а клієнтську частину виконати мовою програмування JavaScript за допомогою фреймворку React.

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі Visual Studio Code.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді добре структурованих файлів з розширенням .py та .jsx.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні вимоги не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

— пояснювальна записка;

— технічне завдання;

— текст програми;

— програма та методика тестування;

— керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

— схема структурна варіантів використання;

— схема структурна класів програмного забезпечення;

— схема бази даних;

— креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	11.03	
2.	Розробка технічного завдання	14.04	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	20.04	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	05.05	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	17.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	24.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	28.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	30.05	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	02.06	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка

до дипломного проекту

на тему: Веб-застосунок для ігрової соціальної платформи

КПІ.ІТ-9402.045440.02.81

Київ – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 4
ВСТУП 5
1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ 6
1.1 Аналіз предметної області 6
1.2 Аналіз існуючих рішень 8
1.2.1 Аналіз відомих програмних продуктів 9
1.2.1.1 Discord 9
1.2.1.2 Steam 10
1.2.1.3 Xbox 11
1.2.2 Аналіз відомих алгоритмічних та технічних рішень 13
1.3 Опис бізнес-процесів 14
1.4 Постановка задачі 17
Висновки до розділу 18
2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 19
2.1 Варіанти використання програмного забезпечення 19
2.2 Аналіз системних вимог 24
2.3 Розроблення функціональних вимог 25
2.4 Розроблення нефункціональних вимог 28
Висновки до розділу 29
3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 31
3.1 Архітектура програмного забезпечення 31
3.2 Обґрунтування вибору засобів розробки 33
3.3 Конструювання програмного забезпечення 35
3.4 Аналіз безпеки даних 39
Висновки до розділу 39
4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 41
4.1 Аналіз якості ПЗ 41
4.2 Опис процесів тестування 41
4.3 Опис контрольного прикладу 46
Висновки до розділу 48
5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 49

5.1 Розгортання програмного забезпечення 49
5.2 Супровід програмного забезпечення 50
Висновки до розділу 51
ВИСНОВОКИ 53
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 55
ДОДАТКИ 56
ДОДАТОК А Звіт подібності 56

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE – Integrated Development Environment – інтегроване середовище розробки

API – Application programming interface

IT – Інформаційні технології

ER – Entity-Relation diagram

ОС – Операційна система

СУБД – Система управління базами даних

ORM – Object-Relational Mapping – об'єктно-реляційне відображення

HTTPS – HyperText Transfer Protocol Secure – захищений протокол передачі гіпертексту

REST – Representational State Transfer – передача репрезентативного стану

JSON – JavaScript Object Notation – формат обміну даними

ВСТУП

В наш час ігрова індустрія займає все більш значиме місце у нашому житті. Вибір ігор у цій сфері надзвичайно великий і сучасній людині доволі складно його зробити. Тому існують сервіси, які допомагають зробити вибір у цій сфері, але вони не враховують те, що переважна більшість сучасних ігор розроблена з розрахунком на гру в команді. Виникає необхідність сервісу, що дозволить знайти гру, яка тобі до вподоби, знайти гравців собі у команду та запланувати спільну ігрову сесію. Така ігрова соціальна платформа буде у нагоді багатьом користувачам, тому що дозволить планувати ігрові сесії з однодумцями заздалегідь, що є невідмінною перевагою у сьогоднішньому постійно-змінному житті.

Постійний ріст популярності онлайн-ігор створює нові можливості для розробників та гравців. Збільшення аудиторії грає на користь обох сторін, оскільки дозволяє задіяти більшу кількість розробників, а користувачам дає більш зручні ігрові та комунікаційні інструменти. Платформи, які забезпечують зручні інструменти для організації командної гри, можуть значно покращити досвід користувача. Впровадження зручних рішень для спілкування сприятиме не лише підвищенню задоволення від гри, але й розвитку міцніших соціальних зв'язків між гравцями.

Наразі інструменти спілкування гравців представлені безліччю сервісів і кожен з них має свої переваги та недоліки. Ця робота буде сфокусована на розробці саме такого сервісу, який б надавав користувачам лише те, що їм дійсно необхідне. Врахування індивідуальних уподобань гравців допоможе створити більш персоналізований досвід, що сприятиме довготривалій зацікавленості користувачів у платформі. Гравці зможуть знаходити нових друзів та обмінюватись досвідом.

Створення такої платформи є важливим кроком для забезпечення комфортного та приємного ігрового досвіду. З її допомогою користувачі зможуть ефективніше використовувати свій ігровий час, насолоджуючись командною грою у колі однодумців.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

В нас час ігрова індустрія займає все більше значиме місце у нашому житті. Вона включає розробку, просування та дистрибуцію відеоігор. Відеоігри стали не лише популярним видом розваг, але й потужним інструментом для навчання, соціалізації та розвитку когнітивних навичок. Вони сприяють розвитку стратегічного мислення, підвищують рівень креативності та поліпшують реакцію і координацію. Ігрова індустрія також має значний вплив на економіку, створюючи робочі місця та генеруючи значні доходи. Зростання ігрової індустрії також призводить до розвитку суміжних IT-секторів, таких як виробництво обладнання, кіберспортивні турніри, а також стрімінгові сервіси.

За даними німецької компанії, що спеціалізується на ринкових і споживацьких даних, Statista, кількість гравців та ринок відеоігор в Україні демонструє стабільне зростання [1]. Кількість гравців та їх витрати на ігри постійно збільшуються, що створює сприятливі умови для розвитку ігрових платформ та сервісів. Розширення ринку також стимулює інновації та розвиток нових технологій, що, у свою чергу, сприяє зростанню популярності відеоігор серед різних вікових груп.

Основні напрямки розробок в ігровій індустрії включають:

— розробку ігор — є основним напрямком в індустрії. Націлений на створення нових ігор для різних ігрових платформ(ПК, консолі, мобільні пристрої). Розробка нових ігор включає процеси дизайну, програмування, тестування та випуску продукту на ринок;

— гейміфікація — використання ігрових механік у неігрових контекстах. Часто цей напрямок можна побачити в освіті, де гейміфікація спрямована на підвищення мотивації та залученості учнів. Ці платформи дозволяють гравцям взаємодіяти, ділитися досягненнями та брати участь у спільних ігрових сесіях;

— ігрові платформи та сервіси — напрямок, на який націлена ця робота. Розробка платформ для дистрибуції, соціалізації та спільної гри;

— віртуальна та доповнена реальність — інтеграція VR та AR технологій у відеоігри для створення більш зануреного ігрового досвіду. Віртуальна реальність дозволяє гравцям повністю зануритися у віртуальний світ, а доповнена реальність додає віртуальні елементи до реального світу.

Ігрова індустрія спирається на теорію ігор, психологію, когнітивні науки та інформатику. Важливими концепціями є ігровий дизайн, механіки гри, гейміфікація та інтерактивність. Теорія ігор вивчає моделі прийняття рішень у конкурентних середовищах, що є фундаментальним для розробки стратегічних та кооперативних ігор. Сучасні ігрові платформи використовують складні алгоритми для рекомендацій, персоналізації контенту, аналітики та відстеження прогресу гравців. Це дозволяє створювати більш персоналізований ігровий досвід, який відповідає індивідуальним потребам кожного гравця.

Попри значний прогрес, існують певні недоліки у поточному стані речей:

— фрагментація сервісів: за даними Statista, найпопулярніші соціальні медіа-платформи для геймерів у США станом на серпень 2022 року включають YouTube, Facebook, Instagram, Discord, TikTok та Twitch [2]. Кожна з цих платформ забезпечує окремі аспекти комунікації, але жодна не забезпечує комплексного рішення для геймерів, що ускладнює процес комунікації та координації. Більшість з цих сервісів надає зручні послуги для своїх користувачів, але не всі цими послугами користуються, оскільки використовують сервіс лише для малої частини доступних функцій;

— відсутність комплексного підходу: більшість платформ зосереджені на окремих аспектах ігрового досвіду, не пропонуючи повного спектру необхідних функцій. Гравці часто використовують різні платформи для комунікації, пошуку команд та організації спільні зустрічі, що створює незручні розриви в інформаційному потоці, що може призвести до втрати

даних або не вчасному їх отриманні. Вимушена потреба гравців перемикатися між різними сервісами знижує зручність користування;

— складність планування: недостатньо зручні інструменти для узгодження графіків між гравцями та планування спільних ігрових зустрічей. Відсутність інтегрованих календарів та нагадувань ускладнює координацію командних дій;

Для покращення ситуації у сфері розробок IT у контексті ігрової індустрії можна запропонувати:

— інтеграція функцій: створення платформ, що поєднують необхідні функції для гравців: пошук команд, планування спільних зустрічей та комунікацію. Це дозволить уникнути фрагментацію сервісів та забезпечить більш зручний користувацький досвід;

— покращення UX/UI: розробка інтуїтивно зрозумілих інтерфейсів для полегшення користування платформою. Інтерфейс повинен бути зручним та доступним, що сприятиму залученню нових користувачів.

У рамках дипломного проєкту було обрано шлях розробки вебзастосунку, який інтегрує необхідні функції для сучасних геймерів. Це включає зручний пошук користувачі, можливість планування спільних подій, а також ефективні засоби для комунікації та координації. Реалізація такого підходу дозволить створити платформу, що максимально відповідає потребам сучасних геймерів та сприяє розвитку ігрової спільноти.

Крім того, важливо враховувати аспекти кібербезпеки під час розробки таких платформ. Захист персональних даних та запобігання кіберзагрозам є ключовими для забезпечення безпеки користувачів. Важливо також забезпечити стабільність та надійність платформи, щоб уникнути збоїв під час інтенсивного використання.

У цілому, розробка комплексної платформи для геймерів є актуальною та перспективною задачею, яка може значно покращити якість ігрового досвіду та сприяти розвитку ігрової індустрії.

1.2 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні програмне забезпечення у даній області та технічні рішення, що допоможуть у реалізації веб застосунку для ігрової соціально платформи. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.2.1 Аналіз відомих програмних продуктів

Сучасні платформи, такі як Discord, Steam та Xbox, пропонують різні функції для гравців, включаючи комунікацію, пошук команд та організацію спільних зустрічей для проведення онлайн ігор. Однак кожна з цих платформ має свої обмеження. Наприклад, Discord добре підходить для голосового та текстового спілкування, але не має зручних інструментів для планування спільного ігрового дозвілля. Steam дозволяє купувати та завантажувати ігри, але його соціальні функції не настільки розвинені. Xbox Live пропонує чудовий сервіс для онлайн-ігор на консолях Xbox, але його функціональність обмежена для гравців на інших платформах.

Основою порівняння платформ буде можливість користувача використовувати їх для планування майбутніх ігрових зустрічей. Ми розглянемо, які інструменти пропонують різні платформи для цієї мети і наскільки вони ефективні та зручні.

1.2.1.1 Discord

Discord є однією з найпопулярніших платформ для голосового та текстового спілкування серед геймерів. Однак його можливості для планування майбутніх ігрових зустрічей обмежені. Гравці можуть створювати текстові канали для обговорення майбутніх зустрічей та використовувати боти для автоматизації деяких аспектів планування. Наприклад, існують боти, які можуть створювати події та нагадування, але ці функції не інтегровані безпосередньо у платформу, що ускладнює процес планування.

Проте Discord має і деякі недоліки. Зокрема, платформа може бути важкою для новачків через велику кількість функцій та налаштувань. Також, хоча Discord забезпечує базовий захист даних, завжди існує ризик зламу або

втрати конфіденційності, особливо якщо сервери не належним чином налаштовані. Для великих спільнот можуть виникати проблеми з управлінням користувачами та забезпеченням безпеки, що вимагає додаткових зусиль з боку адміністраторів серверів. Приклад спільноти наведений на рисунку 1.1

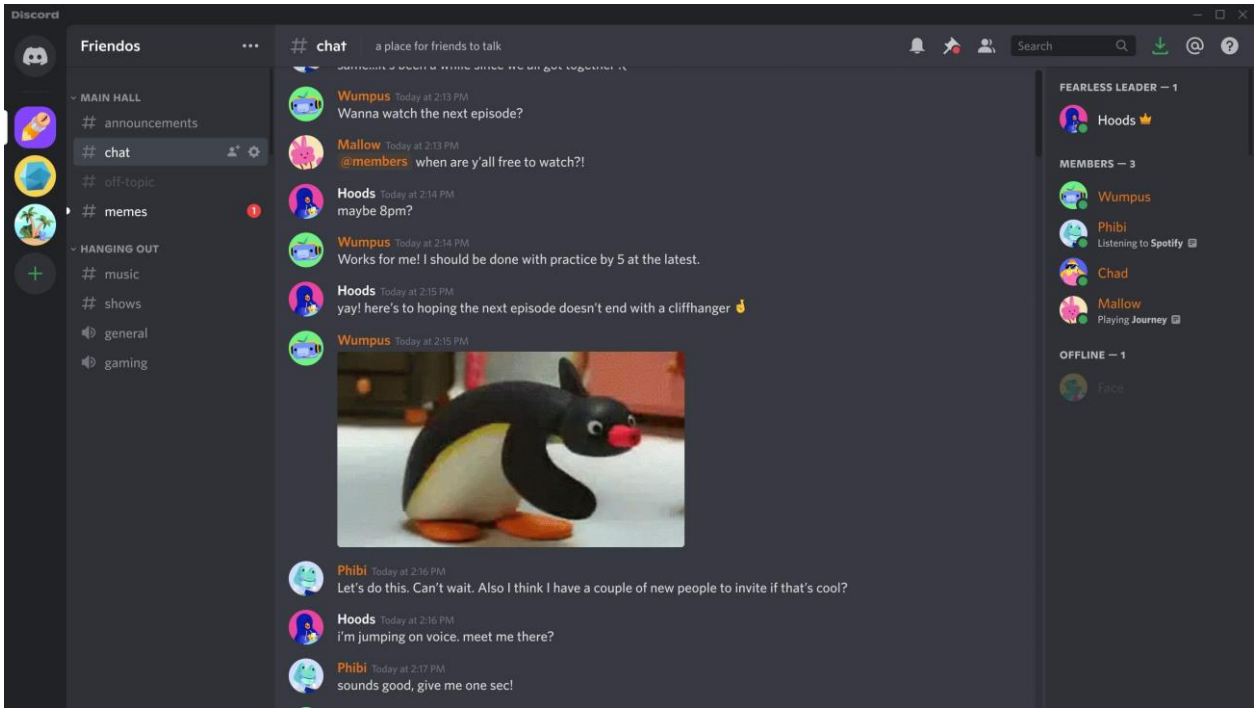


Рисунок 1.1 — Приклад спільноти Discord

1.2.1.2 Steam

Steam пропонує багато соціальних функцій, таких як списки друзів, групи та форуми, що дозволяє гравцям обговорювати ігри та організовувати спільні сесії. Вбудовані чати та можливість надсилання повідомлень роблять комунікацію між гравцями зручною та ефективною. Крім того, Steam надає можливість обміну цифровими товарами, таким як ігрові предмети та картки, що додає інтерактивності та залученості.

Однак можливості для планування майбутніх зустрічей також обмежені. Гравці можуть домовлятися про час і дату зустрічей через чат або форуми, але немає спеціалізованих інструментів для створення подій чи нагадувань. Це створює певні незручності для гравців, які бажають мати чіткий графік ігрових зустрічей. Також, хоча Steam має систему досягнень та лідербордів, її соціальні функції не настільки інтерактивні, як у деяких інших платформах.

Відсутність вбудованих календарів та систем нагадувань вимагає від гравців використовувати сторонні інструменти для організації ігрового часу, що може бути менш зручним. Приклад спільноти Steam наведений на рисунку 1.2.

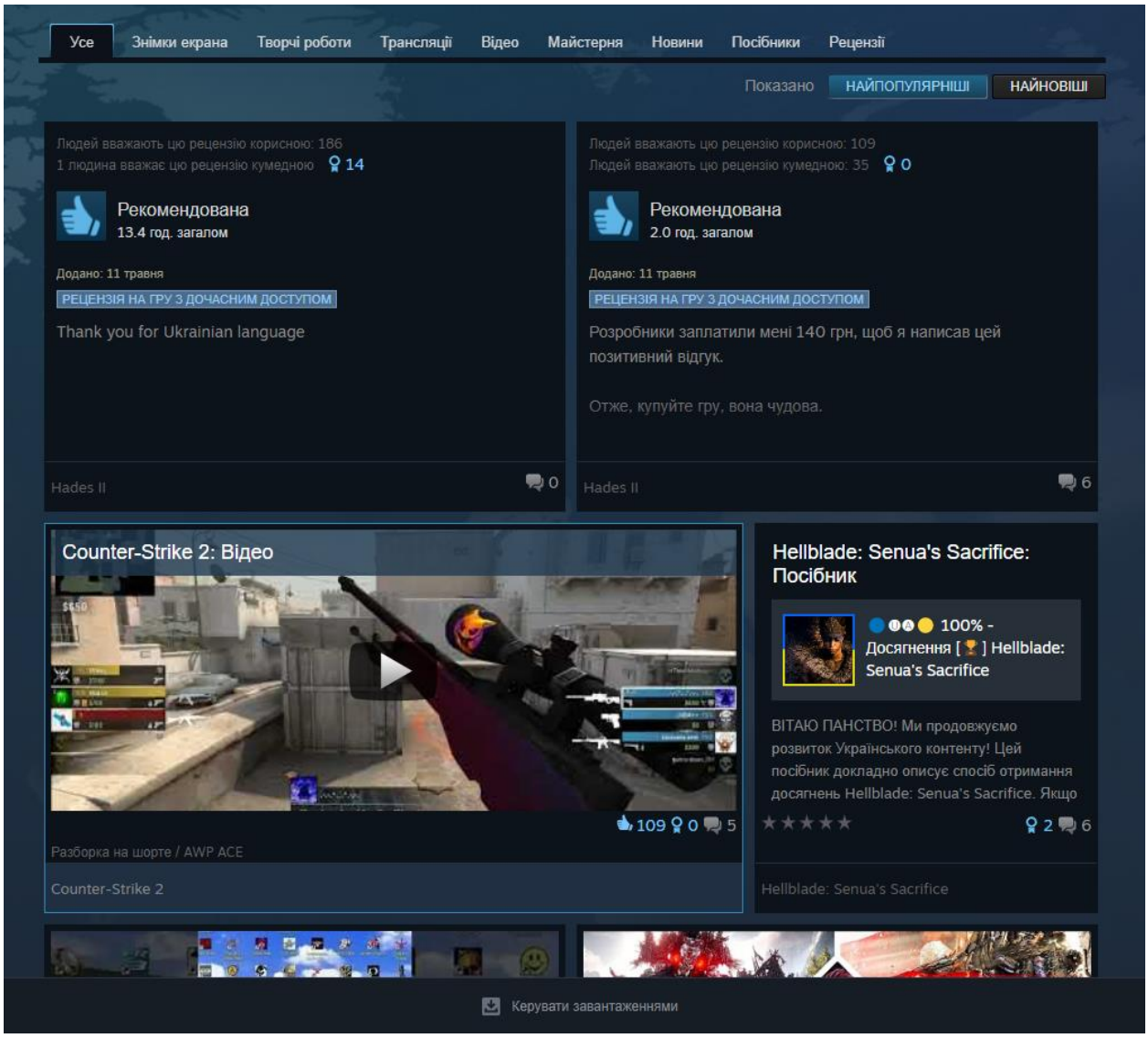


Рисунок 1.2 — Приклад спільноти Steam

1.2.1.3 Xbox

Xbox має розвинені інструменти для організації спільних ігрових зустрічей, особливо на консолях Xbox. Але ця платформа найбільше

Однак для користувачів інших платформ функціональність Xbox обмежена, що може бути недоліком для тих, хто грає на ПК або мобільних пристроях. Це означає, що гравці, які використовують різні платформи, не завжди можуть скористатися всіма перевагами Xbox, що може знизити їхню зручність та інтеграцію в ігрову спільноту.

Також функції спілкування на Xbox не є основними, і платформа не має окремої вкладки для цього, тому їх використання є обмеженим. Спілкування відбувається переважно через голосовий чат під час гри або через текстові повідомлення, але ці функції не такі розвинені, як на спеціалізованих платформах для спілкування, таких як Discord. Приклад чату наведений на рисунку 1.3

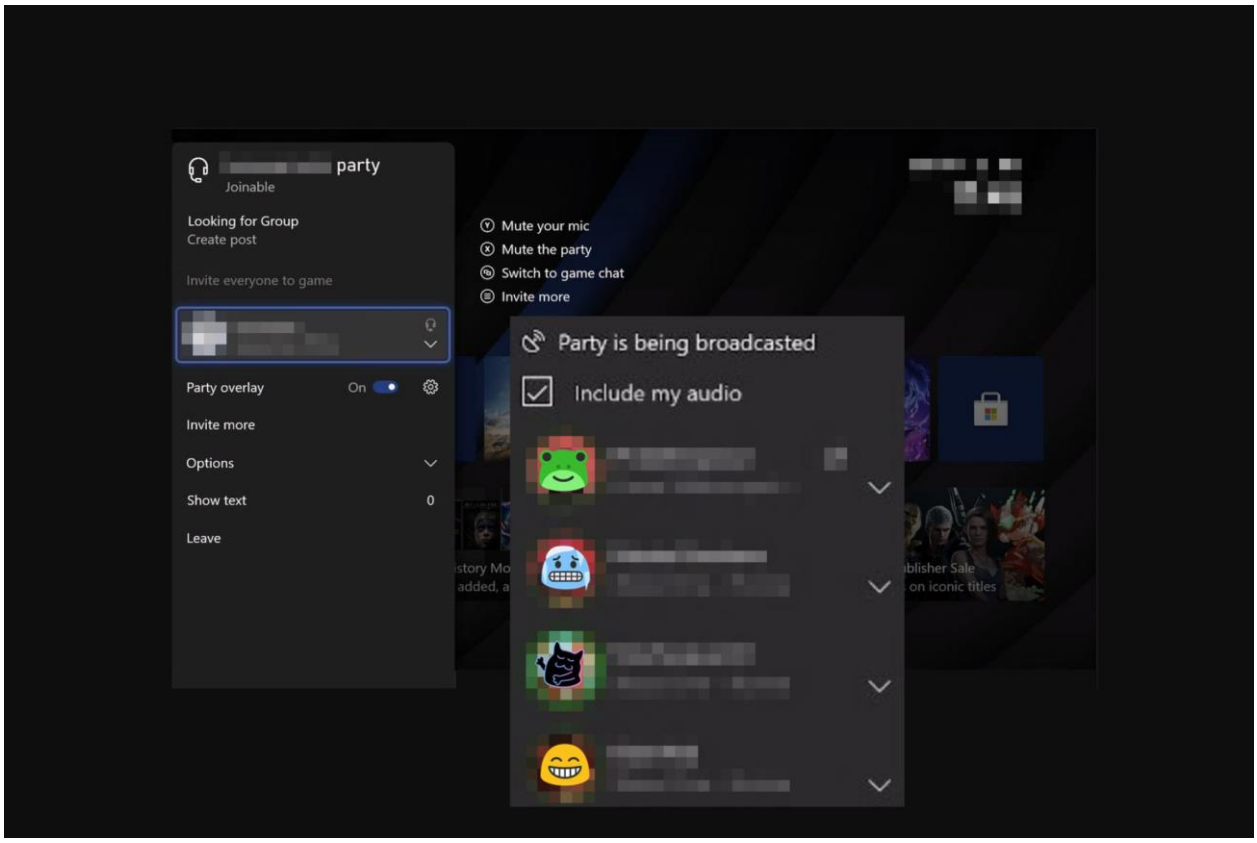


Рисунок 1.3 — Приклад час Xbox

Для порівняння дипломної роботи з аналогами скористаємось порівняльною таблицею 1.1.

Таблиця 1.1 — Порівняння з аналогами

Функціонал	Time-breaker	Steam	Discord	Xbox	Пояснення
Ведення панелей дискусій	+	+	+	-	Можливість вести дискусії серед однодумців

Продовження таблиці 1.1

Пошук гравців	+	+	-	+	Можливість знайти користувачів зі схожими інтересами
Список друзів	+	+	+	+	Функція для додавання та управління друзями
Сповіщення	+	+	+	+	Можливість отримувати повідомлення про оновлення
Планувати спільні зустрічі	+	-	-	-	Інструменти для створення та організації ігрових подій
Експорт календарю	+	-	-	-	Можливість експортувати сесії в зовнішні календарі
Рекомендації гравців	+	-	-	-	Можливість знайти гравців з спільними інтересами

1.2.2 Аналіз відомих алгоритмічних та технічних рішень

Для реалізації рекомендаційних алгоритмів у даній роботі доцільно використати графову базу даних. Графові бази даних є ефективними для

обробки запитів, що пов’язані із зв’язками між об’єктами. Такі запити роблять їх ефективними для обробки зв’язків між об’єктами, такі як: відсутність зв’язку, запит на дружбу та дружба. За допомогою таких зв’язків, можна визначати потенційних друзів, оскільки це за звичай друзі друзів.

Основні технології , які будуть використані у проєкті:

— Python: основна мова програмування для реалізації backend частини проєкту;

— FastAPI: вебфреймворк для створення API;

— PostgreSQL: реляційна база даних для зберігання основних даних платформи;

— Neo4j: графова база даних для зберігання інформації про зв’язки між користувачами;

— React: фронтенд-бібліотека для створення інтерактивного користувацького інтерфейсу.

Алгоритми для розв’язання задач у даній розробці включають:

— алгоритм рекомендацій: використання графових алгоритмів для рекомендацій потенційних друзів на основі спільних зв'язків;

— алгоритм планування: створення інструментів для планування ігрових зустрічей та узгодження графіків між гравцями.

Ці алгоритми та технології забезпечать ефективну розробку для досягнення основних функціональних вимог.

1.3 Опис бізнес-процесів

Для опису бізнес-процесів вашої розробки використовується BPMN модель

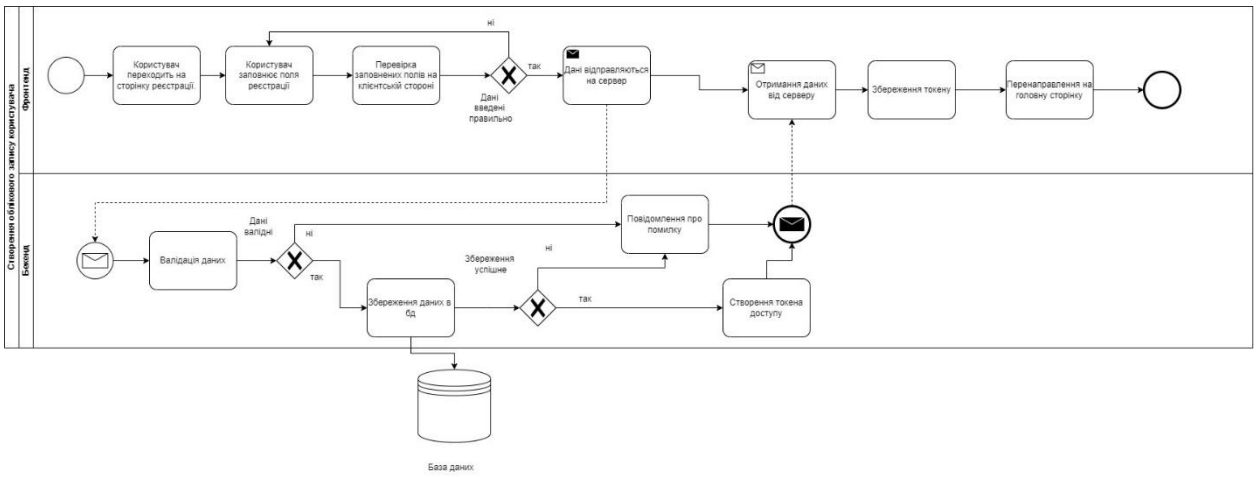


Рисунок 1.4 — Схема створення облікового запису користувача

Опис послідовності створення облікового запису користувача:

— користувач переходить на сторінку реєстрації;

— користувач заповнює поля, які необхідні для реєстрації;

— валідація даних користувача;

— якщо дані валідні, то створюється новий запис в бд та повертається токен доступу, а на користувач перенаправляється на головну сторінку;

— якщо дані помилкові або користувач уже існує, то прийде повідомлення про помилку;

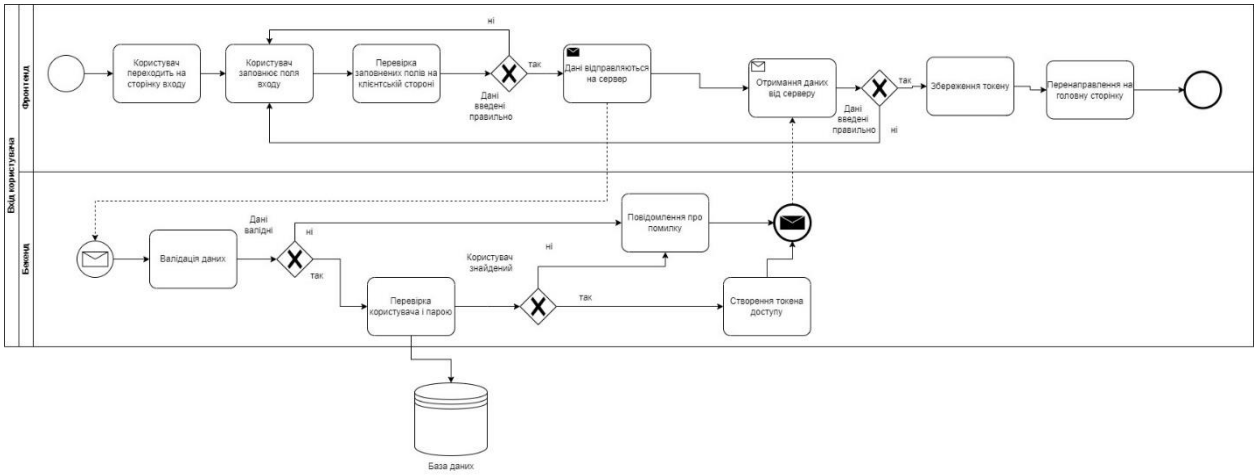


Рисунок 1.5 — Схема входу користувача

Опис послідовності входу користувача:

— користувач переходить на сторінку входу;

— користувач заповнює поля, які необхідні для входу;

— валідація даних користувача;

— якщо дані валідні, то перевіряється існування користувача та паролю з базою даних та повертається токен доступу, а користувач перенаправляється на головну сторінку;

— якщо дані помилкові або користувача не існує, то прийде повідомлення про помилку.

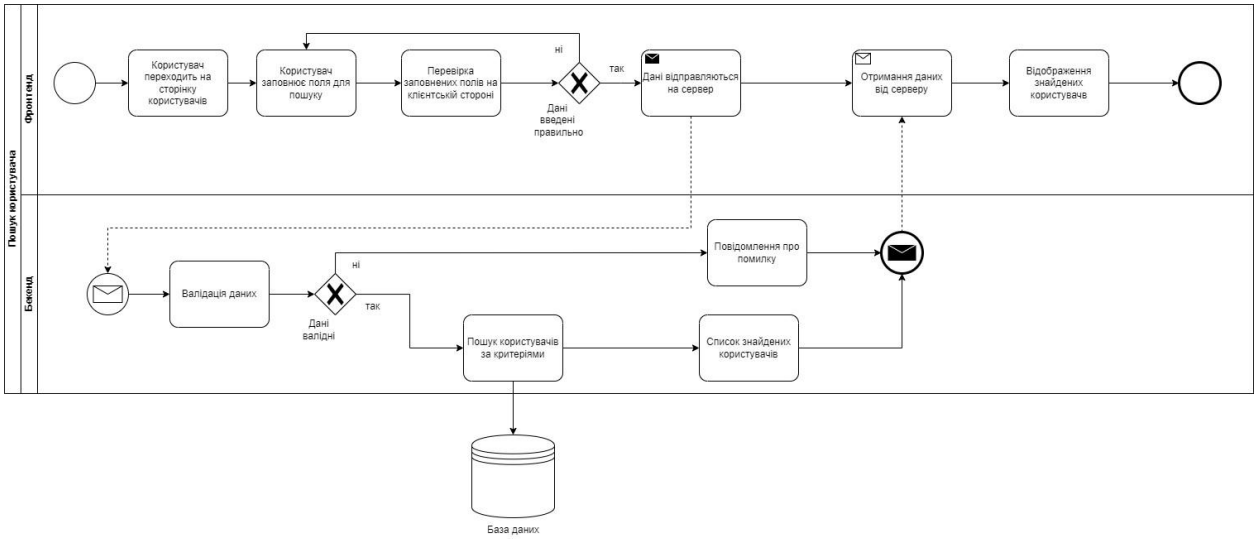


Рисунок 1.6 — Схема пошуку користувача

Опис послідовності пошуку користувача:

— введння даних користувача, якого шукають;

— валідація даних;

— пошук у базі даних;

— відображення знайдених користувачів.

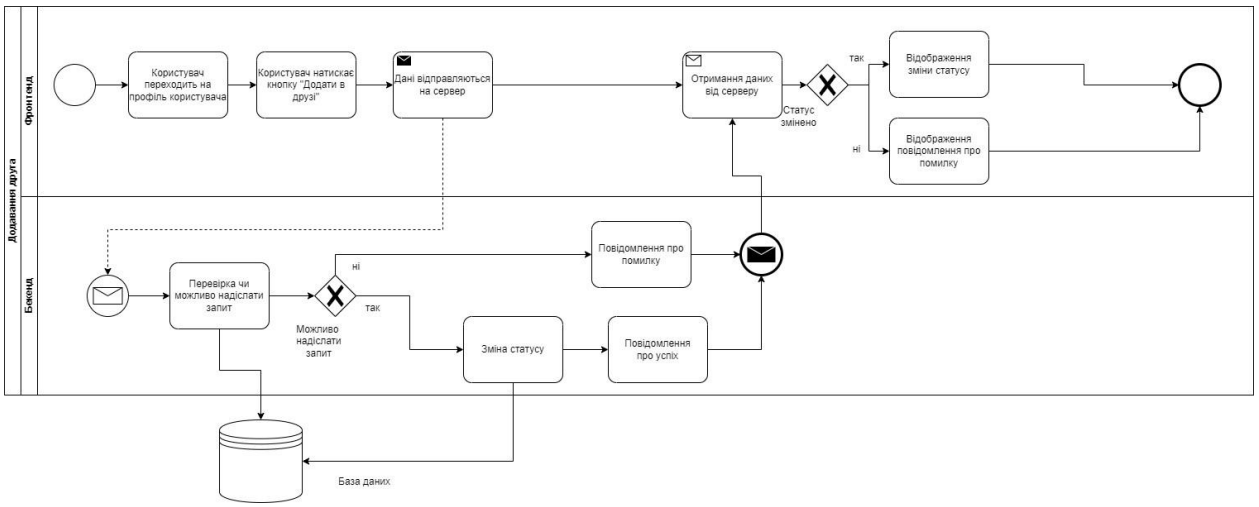


Рисунок 1.7 — Схема додавання в друзі

Опис послідовності додавання в друзі:

— перехід до профілю користувача;

— натискання на кнопку «Додати у друзі»;

— перевірка можливості додавання;

— якщо додати неможливо, то повертається помилка та відображається вона на стороні користувача;

— якщо створити можливо, то оновлюється база даних і відображається оновлений статус на стороні користувача.

1.4 Постановка задачі

Метою розробки є створення веб-застосунку, який забезпечить користувачам можливість планування та обговорення майбутніх ігрових зустрічей, надаючи якісне, швидке та надійне спілкування. Сервіс дозволить користувачам:

— знайти спільноту для обговорення гри;

— знайти користувачів зі спільними вподобаннями;

— запланувати спільні зустрічі для проведення онлайн ігор.

Платформа дозволить планувати спільні зустрічі з однодумцями заздалегідь, що є великою перевагою в умовах сучасного динамічного життя.

Цілі розробки:

— розробити інтуїтивно зрозумілий інтерфейс користувача, який дозволить легко орієнтуватись та здійснювати основні дії:

— авторизація та введення профілю користувача;

— створення спільнот користувачів;+

— планування зустрічей для різних спільнот;

— творення панелей дискусій відповідно до спільнот або запланованих зустрічей.

— забезпечити надійне спілкування користувачів.

Задачі розробки:

— розробити бекенд-частину застосунку:

— реалізація API за допомогою FastAPI;

— створення реляційної бази даних PostgreSQL для зберігання

основних даних платформи;

— інтеграція графової бази даних Neo4j для зберігання та обробки інформації про зв'язки між користувачами.

— розробка фронтенд-частини застосунку:

— використання React для створення інтерфейсу користувача.

Висновки до розділу

У розділі було проведений всебічний аналіз предметної області, існуючих рішень та способів реалізації, а також було описано основні бізнес процеси вебзастосунку для ігрової соціальної платформи.

Аналіз показав, динамічність та змінність ігрової індустрії як сучасної галузі економіки, яка має вплив не лише на розваги, а й на інші сфери, такі як навчання та освіта. Виявлені основні напрями розвитку індустрії, такі як розробка ігор, гейміфікація, ігрові платформи та сервіси, а також віртуальна та доповнена реальність. Було акцентовано увагу на недоліках існуючих сервісів, таких як фрагментація, відсутність комплексного підходу та складність планування подій, що створює передумови для розробки нових рішень.

Аналіз існуючих рішень виявив обмеження сучасних платформ, , у контексті планування ігрових спільні зустрічей та організації часу проведення ігор. Кожна з платформ має свої сильні та слабкі сторони, але жодна з них не вирішує задачу планування.

Опис бізнес-процесів дозволив окреслити основні сценарії взаємодії користувачів з платформою, такі як створення облікового запису, вхід користувача, планування ігрової сесії та інші. Було виділено ролі фронтенд та бекенд частин у кожному процесі, що забезпечує зрозумілість і послідовність виконання кожного сценарію.

Постановка задачі чітко визначила мету розробки, яка полягає у створенні веб-застосунку для планування та обговорення майбутніх ігрових спільні зустрічіей, а також цілі та задачі, які необхідно виконати для досягнення цієї мети. Зокрема, було наголошено на важливості забезпечення надійного спілкування користувачів та реалізації функціоналу для цього.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Головною функцією програмного забезпечення є забезпечення користувачам можливості планування та обговорення майбутніх ігрових зустрічей, надаючи якісне, швидке та надійне спілкування. Додаткові функції включають пошук ігор та гравців, створення команд, додавання друзів.

Рисунок 2.1 — Діаграма варіантів використання

В таблицях 2.1 — 2.10 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 — Варіант використання UC-01

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Реєстрація нового користувача в системі
Actors	Неавторизований користувач
Trigger	Користувач бажає зареєструватися
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку реєстрації. Користувач вводить дані: пошта, пароль, підтвердження пароля. Натискає кнопку реєстрації. Після успішної реєстрації з’являється повідомлення та перенаправлення на сторінку входу.
Extension	Введення некоректних даних робить кнопку реєстрації неактивною. Некоректне поле підсвічується червоним.
Post-Condition	Створення сторінки користувача, перехід на сторінку входу.

Таблиця 2.2 — Варіант використання UC-02

Use case name	Авторизація користувача
Use case ID	UC-02
Goals	Вхід користувача в систему
Actors	Неавторизований користувач
Trigger	Користувач бажає увійти в систему
Pre-conditions	Користувач має бути зареєстрованим
Flow of Events	Користувач переходить на сторінку входу. Вводить логін та пароль. Натискає кнопку входу. Після успішної авторизації з’являється головна сторінка.
Extension	Введення некоректних даних робить кнопку входу неактивною. Некоректне поле підсвічується червоним
Post-Condition	Користувач увійшов в систему та знаходиться на головній сторінці

Таблиця 2.3 — Варіант використання UC-03

Use case name	Пошук користувача
Use case ID	UC-03
Goals	Знайти користувача
Actors	Авторизований користувач
Trigger	Користувач бажає знайти іншого користувача
Pre-conditions	Користувач має бути авторизованим
Flow of Events	Користувач переходить на сторінку пошуку користувачів. Вводяться критерії пошуку. Система шукає та відображає результати.
Extension	Якщо гравець не знайдений, користувачу відображається відповідне повідомлення.
Post-Condition	Відображення знайдених гравців.

Таблиця 2.4 — Варіант використання UC-04

Use case name	Створення спільноти
Use case ID	UC-04
Goals	Створення нової спільноти
Actors	Авторизований користувач
Trigger	Користувач бажає створити спільноту
Pre-conditions	Користувач має бути авторизованим
Flow of Events	Користувач переходить на сторінку пошуку спільноти. Вводяться дані чату. Якщо такої спільноти ще не існує, то користувач натискає кнопку створення. Система створює спільноту та відображає її.
Extension	Якщо дані введені некоректно, кнопка створення неактивна.
Post-Condition	Створення нової спільноти, відображення її сторінки користувачу.

Таблиця 2.5 — Варіант використання UC-05

Use case name	Додавання в друзі
Use case ID	UC-05
Goals	Додавання нового друга
Actors	Авторизований користувач
Trigger	Користувач бажає додати друга
Pre-conditions	Користувач має бути авторизованим
Flow of Events	Користувач переходить на профіль іншого користувача. Натискання кнопки "Додати у друзі". Система надсилає запит іншому користувачу.
Extension	Якщо запит не може бути надісланий, користувачу відображається повідомлення про помилку.
Post-Condition	Запит надісланий, інший користувач отримує повідомлення про новий запит.

Таблиця 2.6 — Варіант використання UC-06

Use case name	Планування спільної ігрової події
Use case ID	UC-06
Goals	Створення нової спільної зустрічі
Actors	Авторизований користувач
Trigger	Користувач бажає запланувати ігрову сесію
Pre-conditions	Користувач має бути авторизованим
Flow of Events	Користувач вводить опис події. Вводяться дані (дата, час). Користувач натискає кнопку створення.
Extension	Якщо дані введені некоректно, кнопка створення неактивна. Некоректне поле підсвічується червоним.
Post-Condition	Створення нової ігрової зустрічі.

Таблиця 2.7 — Варіант використання UC-07

Use case name	Створення посту
Use case ID	UC-07
Goals	Створити нову текстову дискусію
Actors	Авторизований користувач
Trigger	Користувач бажає створити нову текстову дискусію
Pre-conditions	Користувач має бути авторизованим
Flow of Events	Користувач переходить на сторінку спільноти. Вводить текст повідомлення у відповідне поле. Натискає кнопку "Створити". Система створює нову текстову дискусію.
Extension	Якщо дискусія не може бути створена, користувачу відображається повідомлення про помилку.
Post-Condition	Дискусія створена, інший користувачі отримують повідомлення про оновлення в спільноті.

Таблиця 2.8 — Варіант використання UC-08

Use case name	Прийняти запрошення
Use case ID	UC-08
Goals	Прийняти запрошення в друзі
Actors	Авторизований користувач
Trigger	Користувач отримує запрошення в друзі
Pre-conditions	Користувач має бути авторизованим
Flow of Events	Користувач отримує сповіщення про запрошення. Натискає кнопку "Прийняти". Система підтверджує прийняття.
Extension	Якщо запрошення не може бути прийняте, користувачу відображається повідомлення про помилку.
Post-Condition	Запрошення прийняте, стан збережено в системі.

Таблиця 2.9 — Варіант використання UC-09

Use case name	Відхилити запрошення
Use case ID	UC-09
Goals	Відхилити запрошення у друзі
Actors	Авторизований користувач
Trigger	Користувач отримує запрошення
Pre-conditions	Користувач має бути авторизованим
Flow of Events	Користувач отримує сповіщення про запрошення. Натискає кнопку "Відхилити". Система відхиляє запрошення.
Extension	Якщо запрошення не може бути відхилене, користувачу відображається повідомлення про помилку.
Post-Condition	Запрошення відхилене, стан збережено в системі.

Таблиця 2.10 — Варіант використання UC-10

Use case name	Редагувати профіль
Use case ID	UC-10
Goals	Редагування інформації профілю
Actors	Авторизований користувач
Trigger	Користувач бажає редагувати свій профіль
Pre-conditions	Користувач має бути авторизованим
Flow of Events	Користувач переходить на сторінку редагування профілю. Вносить зміни у поля. Натискає кнопку "Зберегти". Система зберігає зміни та відображає оновлену інформацію.
Extension	Якщо дані введені некоректно, кнопка збереження неактивна. Некоректне поле підсвічується червоним.
Post-Condition	Профіль відредагований, оновлена інформація збережена.

2.2 Аналіз системних вимог

Системні вимоги апаратного і програмного забезпечення, які необхідні

для успішного функціонування системи:

— процесор: двоядерний процесор з тактовою частотою 2.0 ГГц або вище з підтримкою віртуалізації;

— оперативна пам'ять: 8 ГБ або більше;

— жорсткий диск: 50 ГБ вільного місця;

— пропускна здатність: Мінімум 100 Мбіт/с;

— операційна система: Windows, macOS, Linux.

2.3 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.11 наведено загальну модель вимог, а в таблиці 2.12 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.3.

Таблиця 2.11 – Загальна модель вимог

№	Назва	ID вимоги	Пріоритети	Ризики
1	Перегляд сторінки привітання	FR-1	Низький	Низький
2	Система авторизації користувача	FR-2	Високий	Високий
2.1	Реєстрація користувача	FR-3	Високий	Високий
2.2	Авторизація користувача	FR-4	Високий	Високий
2.3	Вихід із акаунту	FR-5	Середній	Низький
3	Система друзів	FR-6	Високий	Середній
3.1	Додавання в друзі	FR-7	Високий	Середній
3.2	Прийняти запрошення	FR-8	Високий	Середній
3.3	Відхилити запрошення	FR-9	Високий	Середній
3.4	Видалити з друзів	FR-10	Високий	Середній
3.5	Знайти друга	FR-11	Високий	Середній
4	Система спільнот	FR-12	Високий	Середній

Продовження таблиці 2.11

4.1	Створення дискусії	FR-13	Високий	Середній
4.2	Створення коментарю	FR-14	Високий	Середній
4.3	Видалення дискусії	FR-15	Середній	Середній
4.4	Підписка до спільноти	FR-16	Високий	Середній
4.5	Вихід із спільноти	FR-17	Високий	Середній
5	Система управління профілем	FR-18	Високий	Високий
5.1	Створення профілю	FR-19	Високий	Високий
5.2	Редагування профілю	FR-20	Високий	Високий
6	Система планування зустрічей	FR-21	Середній	Середній
6.1	Створити нову зустріч	FR-22	Середній	Середній
6.2	Редагування зустрічі	FR-23	Середній	Середній
6.3	Видалення зустрічі	FR-24	Середній	Середній

Таблиця 2.12 – Перелік функціональних вимог

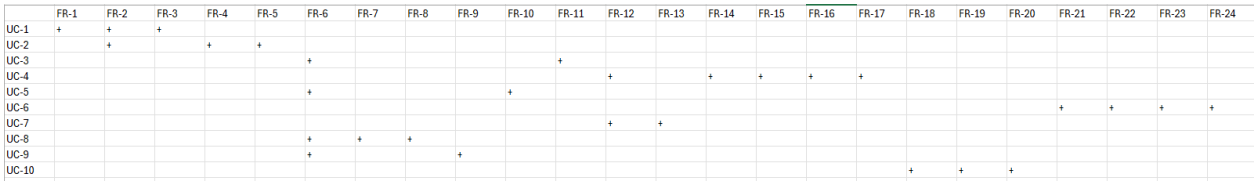
ID вимоги	Назва та опис
FR-1	Перегляд сторінки привітання. Неавторизований користувач бачить веб-сторінку привітання. На ній можна обрати можливість зареєструватися або авторизуватися.
FR-2	Система авторизації користувача. Користувач повинен мати можливість увійти в систему, використовуючи свої облікові дані.
FR-3	Реєстрація користувача. Новий користувач повинен мати можливість зареєструватися в системі, ввівши необхідні дані.
FR-4	Авторизація користувача. Користувач, що вже зареєстрований, повинен мати можливість увійти в систему.
FR-5	Вихід із акаунту. Авторизований користувач повинен мати можливість вийти зі свого акаунту.
FR-6	Система друзів. Користувач повинен мати можливість керувати своїми друзями на платформі.

Продовження таблиці 2.12

FR-7	Додавання в друзі. Авторизований користувач повинен мати можливість додавати інших користувачів у друзі.
FR-8	Прийняти запрошення. Користувач повинен мати можливість приймати запрошення у друзі від інших користувачів.
FR-9	Відхилити запрошення. Користувач повинен мати можливість відхиляти запрошення у друзі.
FR-10	Видалити з друзів. Користувач повинен мати можливість видаляти користувачів зі списку друзів.
FR-11	Користувач повинен мати можливість здійснювати пошук інших користувачів.
FR-12	Система спільнот. Користувач повинен мати можливість створювати і керувати спільнотами.
FR-13	Створення дискусій. Користувач повинен мати можливість створювати текстові дискусії з іншими користувачами.
FR-14	Створення коментарю. Авторизований користувач повинен мати можливість додавати власні коментарі до існуючої дискусії.
FR-15	Видалення дискусії. Користувач повинен мати можливість видаляти існуючі дискусії.
FR-16	Підписка користувача на спільноту. Користувач повинен мати можливість входити до спільнот за інтересами.
FR-17	Вихід користувача із спільноти. Користувач повинен мати можливість виходити із спільнот.
FR-18	Система управління профілем. Користувач повинен мати можливість керувати своїм профілем.
FR-19	Створення профілю. Новий користувач повинен мати можливість створити профіль, ввівши необхідну інформацію.
FR-20	Редагування профілю. Користувач повинен мати можливість редагувати свій профіль, змінюючи інформацію про себе.

Продовження таблиці 2.12

FR-21	Система планування зустрічей. Користувач повинен мати можливість запланувати ігрову зустріч, вказавши дату і час.
FR-22	Створити нову зустріч. Користувач повинен мати можливість створити нову ігрову зустріч.
FR-23	Редагування зустрічі. Користувач повинен мати можливість редагувати існуючі ігрові зустрічі.
FR-24	Видалення зустрічі. Користувач повинен мати можливість видаляти існуючі ігрові зустрічі.



	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17	FR-18	FR-19	FR-20	FR-21	FR-22	FR-23	FR-24
UC-1	+	+	+																					
UC-2		+		+	+																			
UC-3						+					+													
UC-4												+		+	+	+	+							
UC-5						+				+														
UC-6																					+	+	+	+
UC-7												+	+											
UC-8						+	+	+																
UC-9						+			+															
UC-10																		+	+	+				

Рисунок 2.2 — Матриця трасування вимог

2.4 Розроблення нефункціональних вимог

Нефункціональні вимоги визначають критерії, за якими можна оцінити роботу системи, але які не впливають на її функціональність. Ось деякі основні нефункціональні вимоги для вебзастосунку:

— продуктивність: система повинна забезпечувати швидкий і плавний відгук на запити користувачів, незалежно від навантаження;

— надійність: система повинна бути високодоступною та стійкою до збоїв, забезпечуючи безперервну роботу та швидке відновлення в разі неполадок;

— безпека: система повинна гарантувати надійний захист даних користувачів, включаючи механізми аутентифікації та авторизації, а також шифрування даних;

— масштабованість: система повинна бути легко масштабованою, щоб ефективно обробляти збільшене навантаження та ріст кількості

користувачів;

— зручність використання: Інтерфейс системи повинен бути інтуїтивно зрозумілим і зручним для користувачів, забезпечуючи легкий доступ до всіх функцій;

— сумісність: система повинна бути сумісна з основними операційними системами та веб-браузерами, забезпечуючи безперебійну роботу на різних платформах;

— мобільність: система повинна мати адаптивний дизайн або мобільний додаток, який забезпечує доступ до функцій з мобільних пристроїв;

— обслуговуваність: Система повинна мати можливість легкої підтримки та обслуговування, включаючи зручні механізми логування та оновлення.

Висновки до розділу

У розділі були розглянуті вимоги до програмного забезпечення, яке забезпечує користувачам можливість планування та обговорення майбутніх ігрових зустрічей, а також надає інструменти для пошуку спільнот і користувачів, створення спільнот і додавання друзів.

Варіанти використання програмного забезпечення були детально описані в таблицях 2.1 — 2.10. Для кожного з варіантів використання визначено ключові цілі, актори, умови початку, основний потік подій, розширення та кінцевий стан. Це дозволило чітко визначити, як користувачі будуть взаємодіяти з системою, що спрощує процес розробки та тестування.

Аналіз системних вимог допоміг визначити апаратні та програмні характеристики, необхідні для успішного функціонування системи. Це включає вимоги до процесора, оперативної пам'яті, жорсткого диска, пропускної здатності та операційних систем.

Розроблення функціональних вимог включало визначення основних функцій системи, таких як реєстрація, авторизація, управління друзями, створення спільнот і планування ігрових зустрічей. У таблицях 2.11 і 2.12 наведено перелік функціональних вимог, що забезпечує чітке розуміння

необхідних функцій та їхніх пріоритетів.

Нефункціональні вимоги були розглянуті для забезпечення критеріїв, за якими можна оцінити роботу системи. Це включає продуктивність, надійність, безпеку, масштабованість, зручність використання, сумісність, мобільність та обслуговуваність. Ці вимоги є критично важливими для забезпечення високої якості програмного забезпечення та задоволення потреб користувачів.

Загалом, розділ забезпечив детальне визначення як функціональних, так і нефункціональних вимог, що є основою для успішної розробки, впровадження та експлуатації програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Вебзастосунки розробляються за допомогою клієнт сервісної архітектури. Але деталі внутрішньої організації клієнта і сервера також мають свої архітектурні підходи. Для розробки даного вебзастосунку доцільно використати архітектурний патерн «Моноліт», де бекенд-сервіси будуть організовані за триярусною архітектурою. У даній архітектурі виділяють наступні рівні:

— рівень представлення (Frontend): реалізований за допомогою бібліотеки React. Відповідає за взаємодію з користувачем, відображення даних та обробку подій на стороні клієнта;

— рівень бізнес-логіки (Backend): реалізований за допомогою веб-фреймворка FastAPI. Відповідає за обробку запитів від клієнта, виконання бізнес-логіки та взаємодію з іншими сервісами;

— рівень даних (Databases): складається з реляційної бази даних PostgreSQL та графової бази даних Neo4j. PostgreSQL використовується для зберігання основних даних платформи, таких як профілі користувачів, повідомлення, чати тощо. Neo4j використовується для зберігання та обробки інформації про зв'язки між користувачами, що дозволяє ефективно реалізувати функціонал рекомендацій та пошуку друзів.

Бекендова сторона складється з основних копонентів: це контролери, бізнес логіка та репозиторій(рівень даних). Кожен компонет відповідає за певну частину роботи. Деталізація компонентів системи:

— Authorization Service: відповідає за управління користувачами, реєстрацію, авторизацію та аутентифікацію користувачів. Цей сервіс взаємодіє з PostgreSQL для зберігання даних користувачів та з Neo4j для ініціалізації користувача в графовій базі даних;

— Relationship Service: відповідає за управління списком друзів, відправлення запитів на додавання у друзі та прийняття або відхилення

запитів. Цей сервіс взаємодіє з Neo4j для зберігання інформації про зв'язки між користувачами;

— Calendar Service: забезпечує функціонал для планування ігрових зустрічей, узгодження графіків між гравцями та надсилання нагадувань про заплановані сесії. Цей сервіс взаємодіє з PostgreSQL та Notification Service;

— Profile Service: відповідає за створення та ведення сторінки користувача;

— Comments Service: Відповідає за управління коментарями до постів у платформі;

— Notification Service: Відповідає за надсилання різних сповіщень користувачам, таких як коментарі, пости, нагадування про заплановані сесії тощо. Взаємодіє з PostgreSQL для зберігання нагадувать та статусів їх отримання;

— Post Service: Відповідає за управління постами користувачів, включаючи створення, редагування та видалення постів. Взаємодіє з PostgreSQL для зберігання постів;

— Community Service: Відповідає за створення та управління спільнотами.

Розглянемо деталі взаємодії системи та її компонентів. На рисунку 3.1 представлена ця логіка, що відображає взаємодію між компонентами системи для кращого розуміння її функціонування.

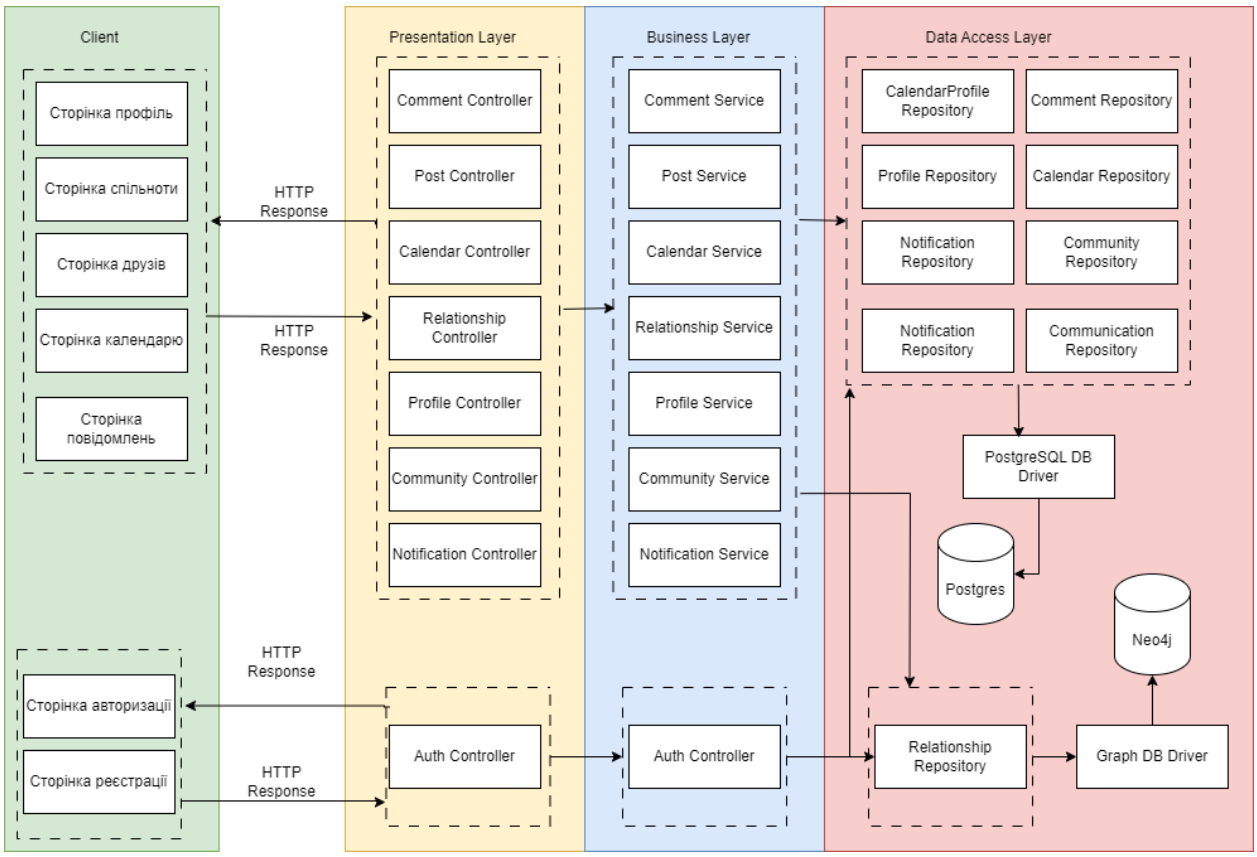


Рисунок 3.1 — Взаємодія компонентів системи

3.2 Обґрунтування вибору засобів розробки

Визначимось із програмними засобами та засобами розробки, що будуть використані для створення вебзастосунку. Розглянемо переваги вибору Python, React, PostgreSQL, Neo4j, FastAPI, Docker та інших інструментів розробки.

Python [3] є однією з найпопулярніших мов програмування для веб-розробки завдяки своїй простоті, читабельному синтаксису та широкому набору бібліотек. Використання Python у веб-розробці забезпечується за допомогою потужних фреймворків, таких як Django і FastAPI. Python дозволяє швидко розробляти та підтримувати код, що є важливим для створення прототипів і випуску продуктів у короткі терміни. Крім того, Python має велику спільноту розробників та багато навчальних матеріалів, що спрощує навчання і підтримку.

FastAPI [4] — це сучасний, високопродуктивний веб-фреймворк для Python, призначений для створення API. Він забезпечує автоматичну

генерацію документації за допомогою Swagger і Redoc, підтримує асинхронні запити, що дозволяє створювати швидкі та ефективні веб-додатки. FastAPI також легко інтегрується з іншими бібліотеками Python, що робить його відмінним вибором для розробки серверної частини.

React [5] — це популярна бібліотека JavaScript для створення інтерфейсів користувача, розроблена компанією Facebook. Вона дозволяє створювати динамічні та інтерактивні веб-додатки за допомогою компонентно-орієнтованого підходу. React має високу продуктивність завдяки віртуальному DOM та підтримує односторонній потік даних, що спрощує відстеження стану додатка. React також має великий набір сторонніх бібліотек і компонентів, що значно спрощує розробку складних інтерфейсів.

PostgreSQL [6] — це потужна реляційна база даних з відкритим вихідним кодом, яка відома своєю надійністю, продуктивністю та відповідністю стандартам SQL. Вона підтримує складні запити, транзакції та індекси, що дозволяє ефективно працювати з великими обсягами даних. PostgreSQL має багатий набір функцій для роботи з даними, включаючи підтримку JSON, що робить її гнучкою для різних типів додатків.

Neo4j [7] — це провідна графова база даних, яка забезпечує збереження та обробку даних з високою продуктивністю завдяки використанню графових структур. Вона ідеально підходить для роботи з даними, які мають складні взаємозв'язки, такі як соціальні мережі, рекомендаційні системи та інші. Neo4j дозволяє швидко виконувати складні запити на основі зв'язків між об'єктами, що є важливим для реалізації функціоналу рекомендацій та пошуку друзів.

Docker [8] — це платформа для контейнеризації, яка дозволяє розробникам упаковувати додатки та всі їхні залежності в контейнери, що забезпечує їх ізоляцію та переносимість. Використання Docker спрощує розгортання додатків на різних середовищах, забезпечуючи однакову поведінку як на локальних машинах розробників, так і в продуктивному середовищі. Docker також сприяє швидкому масштабуванню додатків і спрощує управління залежностями.

Alembic [9] — це інструмент для керування міграціями баз даних у Python. Він дозволяє розробникам легко створювати, відслідковувати та керувати змінами в схемі бази даних під час розробки. Alembic інтегрується з SQLAlchemy, що спрощує процес визначення та застосування міграцій, забезпечуючи узгодженість та цілісність даних.

Отже, вибір Python для бекенд-розробки, React для фронтенду, PostgreSQL та Neo4j для зберігання даних, а також використання FastAPI, Docker, що забезпечує ефективність, гнучкість та зручність у розробці вебзастосунку.

3.3 Конструювання програмного забезпечення

Розглянемо ER діаграму, що показує сутності та зв'язки між ними в системі (рис. 3.2). Ця діаграма дозволяє наочно уявити структуру бази даних та взаємодію між різними сутностями, що покращує розуміння вимог до системи та спрощує процес розробки.

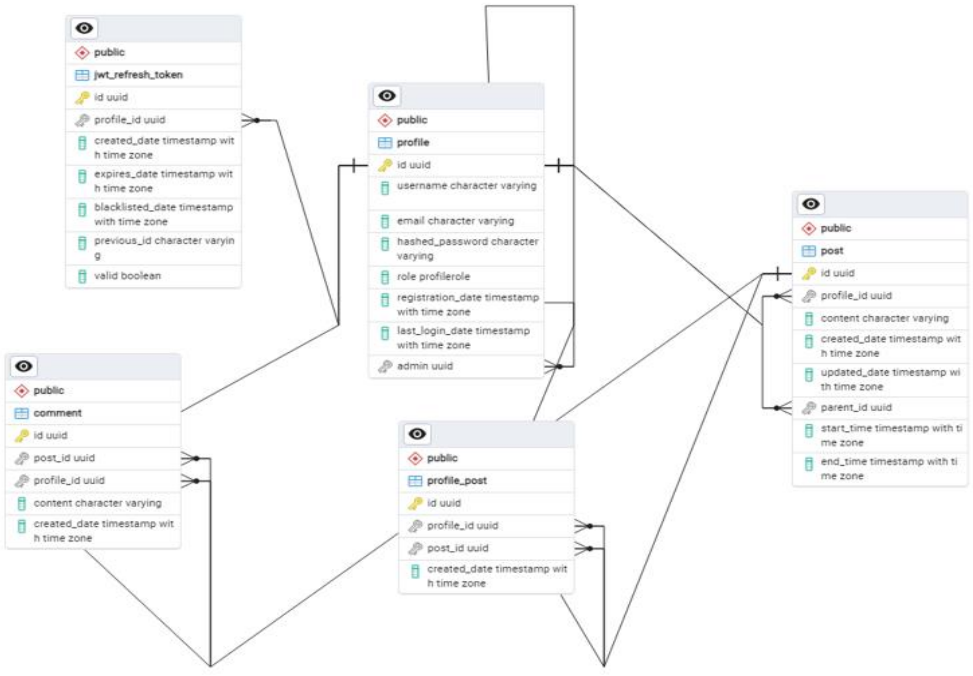


Рисунок 3.2 — ER діаграма сутностей системи

Системою управління бази даних є PostgreSQL. База даних складається

з наступних сутностей: profile, post, jwt_refresh_token, profile_post, comment, які описані в таблицях нижче.

Таблиця 3.1 — Опис таблиці profile

Назва поля	Тип даних	Опис
Id	UUID	Ідентифікаційний номер профілю, первинний ключ
Username	String	Ім'я користувача, унікальне, індексоване, обов'язкове
email	String	Електронна пошта користувача, унікальна, обов'язкова
hashed_password	String	Хешований пароль користувача, обов'язковий
role	Role	Роль користувача, за замовчуванням user
registration_date	DateTime	Дата реєстрації, встановлюється автоматично
last_login_date	DateTime	Дата останнього входу, встановлюється автоматично

Таблиця 3.2 — Опис таблиці jwt_refresh_token

Назва поля	Тип даних	Опис
Id	UUID	Ідентифікаційний номер токена, первинний ключ
profile_id	UUID	Ідентифікатор профілю, зовнішній ключ
created_date	DateTime	Дата створення токена,
expires_date	DateTime	Дата закінчення дії токена, обов'язкова
blacklisted_date	DateTime	Дата додавання в чорний список
previous_id	UUID	Попередній ідентифікатор токена
valid	Boolean	Дійсність токена, за замовчуванням True, обов'язкова

Таблиця 3.3 — Опис таблиці profile_post

Назва поля	Тип даних	Опис
Id	UUID	Ідентифікаційний номер слоту, первинний ключ
Profile_id	UUID	Ідентифікаційний номер профілю
Post_id	UUID	Ідентифікаційний номер події
created_date	DateTime	Дата створення слоту, встановлюється автоматично

Таблиця 3.4 — Опис таблиці comment

Назва поля	Тип даних	Опис
id	UUID	Ідентифікаційний номер коментаря, первинний ключ
post_id	UUID	Ідентифікатор посту, зовнішній ключ, каскадне видалення
profile_id	UUID	Ідентифікатор профілю, зовнішній ключ, каскадне видалення
content	String	Зміст коментаря, обов'язковий
created_date	DateTime	Дата створення коментаря, встановлюється автоматично

Таблиця 3.5 — Опис таблиці post

Назва поля	Тип даних	Опис
id	UUID	Ідентифікаційний номер посту, первинний ключ
profile_id	UUID	Ідентифікатор профілю, зовнішній ключ, каскадне видалення
content	String	Зміст посту, обов'язковий
start_time	DateTime	Час початку, обов'язкова

Продовження таблиці 3.5

created_date	DateTime	Дата створення посту, встановлюється автоматично
updated_date	DateTime	Дата оновлення посту, встановлюється автоматично
parent_id	UUID	Ідентифікатор користувачу, де створений пост, зовнішній ключ, каскадне видалення
end_time	DateTime	Час закінчення, обов'язкова

Також розглянемо діаграму класів, щоб показати, які сутності присутні у системі та як вони взаємодіють один з одним. На діаграмі класів ми бачимо основні сутності, такі як Користувач, Пост, Коментар, Спільнота, Відносини та Подія.

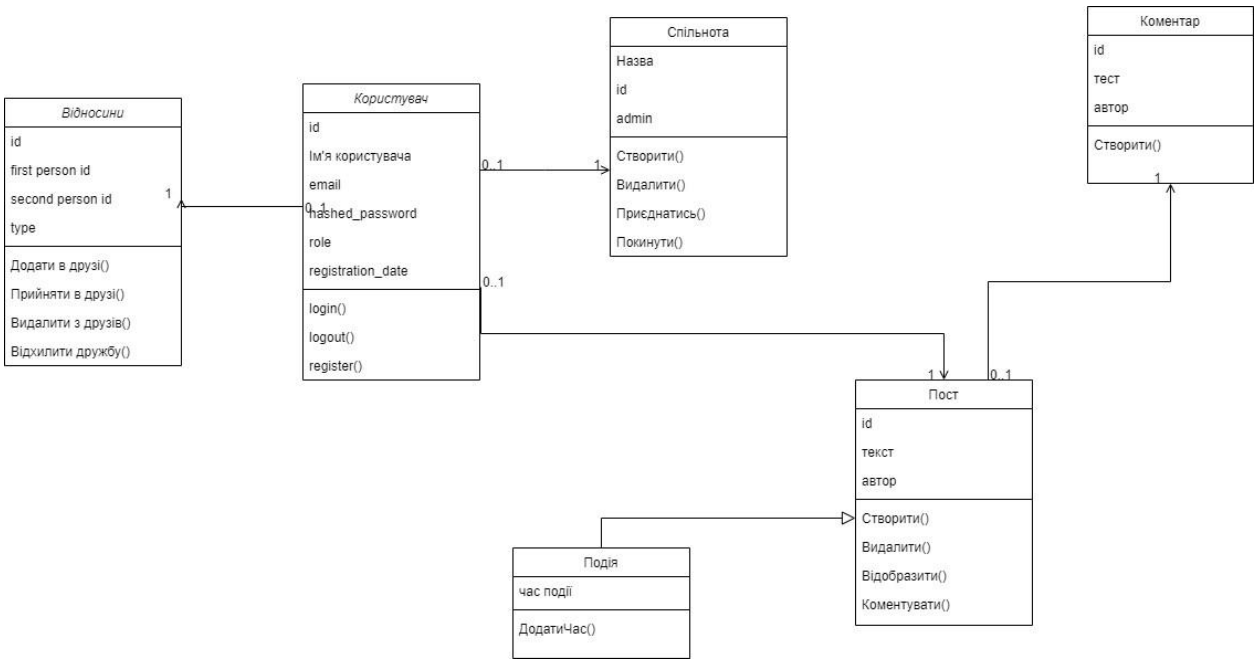


Рисунок 3.3 — Схема структури класів

Опис утилі, що використовуються для розробки програмного забезпечення наведений в таблиці 3.12

Таблиця 3.6 — Опис утиліт для розробки

№ п/п	Назва утиліти	Опис застосування
	Neo4j Desktop	Інструмент для розробки, адміністрування та управління базами даних Neo4j
	Visual Studio Code	Середовище розробки з відкритим вихідним кодом, яке підтримує широкий спектр мов програмування та фреймворків.
	Postman	Інструмент для тестування API, який дозволяє надсилати HTTP-запити та отримувати відповіді

3.4 Аналіз безпеки даних

Безпека застосунку соціальної платформи є критичною, оскільки застосунок працює з конфіденційною інформацією. Захист від несанкціонованого доступу — є критично важливою, щоб зберегти довіру користувачі.

Доступ до застосунки можливий лише авторизованим користувачам. Застосунок повинен підтримувати шифрування інформації та захист від SQL-ін’єкції, потрібне постійне введення логів безпеки та моніторинг. Застосування цих заходів допоможе забезпечити високий рівень безпеки для користувачів соціальної платформи, зберігаючи довіру і захист конфіденційної інформації.

Висновки до розділу

У розділі розглянуто підхід до архітектури та вибору засобів для розробки програмного забезпечення для вебзастосунку із використанням клієнт-серверної архітектури.

Для даного вебзастосунку обрано архітектурний патерн «Моноліт» з триярусною архітектурою:

— рівень представлення (Frontend): використовується бібліотека React для взаємодії з користувачем;

— рівень бізнес-логіки (Backend): реалізований за допомогою FastAPI, відповідає за обробку запитів і виконання бізнес-логіки;

— рівень даних (Databases): включає реляційну базу даних PostgreSQL для основних даних і графову базу даних Neo4j для інформації про зв'язки між користувачами.

Бекендова сторона організована за принципом модульності, де кожен сервіс відповідає за певну функціональність (авторизація, управління друзями, планування зустрічей, управління спільнотами, коментарями, постами та сповіщеннями).

Проаналізовано структуру бази даних за допомогою ER діаграми, що відображає сутності системи та зв'язки між ними, забезпечуючи цілісне уявлення про структуру даних і полегшуючи розробку.

Розробка забезпечує високий рівень безпеки, включаючи авторизацію, шифрування даних та захист від SQL-ін'єкцій, що критично важливо для збереження конфіденційної інформації користувачів.

Загалом, даний підхід до розробки забезпечує ефективність, гнучкість і зручність створення вебзастосунку, забезпечуючи високий рівень безпеки та продуктивності.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Аналіз якості є важливою частиною розробки програмного забезпечення. В якості аналізатору якості коду було обрано SonarCloud. SonarCloud, який підходить для цього проєкту через технології, які були у ньому використані. SonarCloud — це хмарний сервіс для аналізу коду, який постійно спрацьовує при змінах у репозиторії та підтримує різноманітні мови програмування, забезпечує глибокий аналіз якості коду. Його перевагами є зручність використання, легка інтеграція з хмарними CI/CD сервісами, автоматичні оновлення та мінімальні вимоги до налаштування і адміністрування. В процесі аналізу були отримані оцінки з наступних метрик:

— reliability — рейтинг A, проблем не виявлено;

— security — рейтинг A, проблем не виявлено;

— maintainability — рейтинг A, technical debt ratio 0.3%;

— duplication — 2% дублювань коду.

4.2 Опис процесів тестування

Було виконано мануальне тестування програмного забезпеення, опис відповідних тестів наведено у таблицях

Таблиця 4.1 — Тест 1.1 Реєстрація користувача

Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні дані	Логін, пошта, пароль облікового запису
Опис проведення тесту	У відповідні поля форми реєстрації вводяться коректні дані. Пошта та логін, які ще не використовувались для реєстрації. Та пароль довший мінімальної кількості символів — 8. Натискається кнопка реєстрації.

Продовження таблиці 4.1

Очікуваний результат	Користувач успішно реєструється у платформі та перенаправляється на сторінку входу.
Фактичний результат	Користувач успішно реєструється у платформі та перенаправляється на сторінку входу.

Таблиця 4.2 — Тест 1.2 Вхід користувача

Початковий стан системи	Користувач знаходиться на сторінці входу
Вхідні дані	Логін та пароль облікового запису
Опис проведення тесту	У відповідні поля форми входу вводяться коректні дані. Натискається кнопка входу.
Очікуваний результат	Користувач успішно входить у платформу та перенаправляється на головну сторінку.
Фактичний результат	Користувач успішно входить у платформу та перенаправляється на головну сторінку.

Таблиця 4.3 — Тест 1.3 Публікація нового посту

Початковий стан системи	Користувач знаходиться на власній сторінці
Вхідні дані	Текст посту
Опис проведення тесту	У поле посту вводяться коректні дані. Натискається кнопка публікації.
Очікуваний результат	На сторінці з’являється новий пост
Фактичний результат	На сторінці з’являється новий пост

Таблиця 4.4 — Тест 1.4 Видалення посту

Початковий стан системи	Користувач знаходиться на власній сторінці
Вхідні дані	Пост для видалення
Опис проведення тесту	На відповідному посту натискається кнопка видалення

Продовження таблиці 4.4

Очікуваний результат	На сторінці зникає вибраний пост
Фактичний результат	На сторінці зникає вибраний пост

Таблиця 4.5 — Тест 1.5 Пошук спільноти

Початковий стан системи	Користувач знаходиться на сторінці пошуку
Вхідні дані	Назва спільноти
Опис проведення тесту	На відповідному полі пошуку вводиться назва спільноти. Натискається кнопка пошуку.
Очікуваний результат	На сторінці пошуку відображаються усі знайдені спільноти
Фактичний результат	На сторінці пошуку відображаються усі знайдені спільноти

Таблиця 4.6 — Тест 1.6 Пошук користувача

Початковий стан системи	Користувач знаходиться на сторінці пошуку
Вхідні дані	Ім’я користувача
Опис проведення тесту	На відповідному полі пошуку вводиться ім’я користувача. Натискається кнопка пошуку.
Очікуваний результат	На сторінці пошуку відображаються усі знайдені користувачі
Фактичний результат	На сторінці пошуку відображаються усі знайдені користувачі

Таблиця 4.7 — Тест 1.7 Створення спільноти

Початковий стан системи	Користувач знаходиться на сторінці пошуку
Вхідні дані	Назва спільноти
Опис проведення тесту	На відповідному полі пошуку назва спільноти. Натискається кнопка пошуку. Спільнота не знайдена. Натискається кнопка створення спільноти з цією назвою. Натискається кнопка підтвердження.

Продовження таблиці 4.7

Очікуваний результат	Створюється спільнота з відповідним ім’ям, користувач, який її створив стає її адміністратором та перенаправляється на екран спільноти.
Фактичний результат	Створюється спільнота з відповідним ім’ям, користувач, який її створив стає її адміністратором та перенаправляється на екран спільноти.

Таблиця 4.8 — Тест 1.8 Перегляд нотифікацій

Початковий стан системи	Користувач авторизований на будь-якій сторінці
Вхідні дані	Дані користувача
Опис проведення тесту	На верхній панелі натискається кнопка перегляду нотифікацій
Очікуваний результат	Відображається панель нотифікацій з усіма знайденими подіями для користувача.
Фактичний результат	Відображається панель нотифікацій з усіма знайденими подіями для користувача.

Таблиця 4.9 — Тест 1.9 Створення запланованої події

Початковий стан системи	Користувач знаходиться на сторінці спільноти
Вхідні дані	Дані події, дата та час.
Опис проведення тесту	У полі посту вводяться дані події, натискається кнопка додавання час та вводяться часові рамки події.
Очікуваний результат	Відображається пост з можливістю підписатись на подію та списком людей, які уже підписались.

Продовження таблиці 4.9

Фактичний результат	Відображається пост з можливістю підписатись на подію та списком людей, які уже підписались.

Таблиця 4.10 — Тест 1.10 Додавання коментарю до посту

Початковий стан системи	Користувач знаходиться на сторінці спільноти
Вхідні дані	Текст коментаря
Опис проведення тесту	Користувач натискає кнопку відображення коментарів до відповідного посту. У полі нового коментаря вводить текст коментарю. Натискає кнопку коментувати
Очікуваний результат	З’являється новий коментар для посту
Фактичний результат	З’являється новий коментар для посту

Таблиця 4.11 — Тест 1.11 Підписка на подію

Початковий стан системи	Користувач знаходиться на сторінці спільноти
Вхідні дані	Дані про подію
Опис проведення тесту	Користувач натискає кнопку підписки для вибраної події
Очікуваний результат	Користувач з’являється у списку підписників події. Подія відображається у календарі користувача.
Фактичний результат	Користувач з’являється у списку підписників події. Подія відображається у календарі користувача.

Таблиця 4.12 — Тест 1.12 Відписка від події

Початковий стан системи	Користувач знаходиться на сторінці календарю
Вхідні дані	Подія

Продовження таблиці 4.12

Опис проведення тесту	Користувач натискає на час у календарі, коли відбувається подія. З'являється вікно підтвердження. Користувач підтверджує дію
Очікуваний результат	Підписка на подію видаляється
Фактичний результат	Підписка на подію видаляється

4.3 Опис контрольного прикладу

Розглянемо контрольний приклад для мануального тестування процесу створення нової події у таблицях 4.13 – 4.14.

Таблиця 4.13 — Тест 4.13 Створення порожньої події

Початковий стан системи	Користувач знаходиться на сторінці спільноти
Вхідні дані	Подія
Опис проведення тесту	Користувач натискає кнопку створити без заповнення поля події
Очікуваний результат	Відображається повідомлення про помилку
Фактичний результат	Відображається повідомлення про помилку

Ілюстрації до тесту наведені на рисунках 4.1-4.2

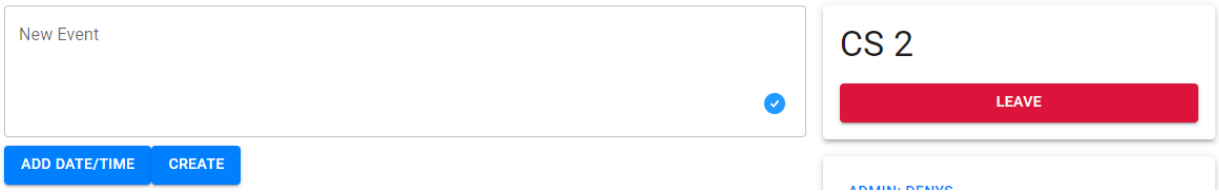


Рисунок 4.1 — Початковий стан системи

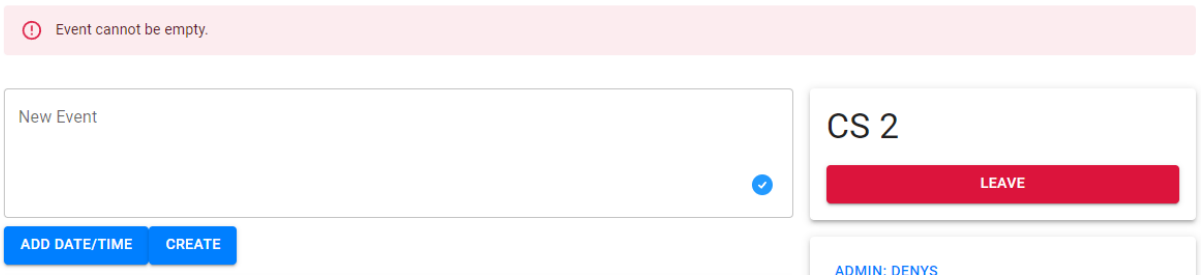


Рисунок 4.2 — Стан системи після натискання кнопки «Create»

Таблиця 4.14 — Тест 4.14 Створення події без кінцевої дати

Початковий стан системи	Користувач знаходиться на сторінці спільноти
Вхідні дані	Подія, початкова дата
Опис проведення тесту	Користувач заповнює поле події і дату початку події, але залишає поле кінцевої дати порожнім
Очікуваний результат	Відображається повідомлення про помилку, що пропущена обидва поля повинні бути заповненими
Фактичний результат	Відображається повідомлення про помилку, що пропущена обидва поля повинні бути заповненими

Ілюстрації до тесту наведені на рисунках 4.3-4.5

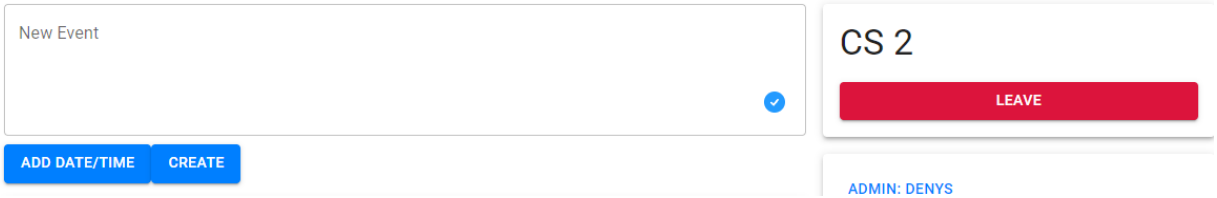


Рисунок 4.3 — Початковий стан системи

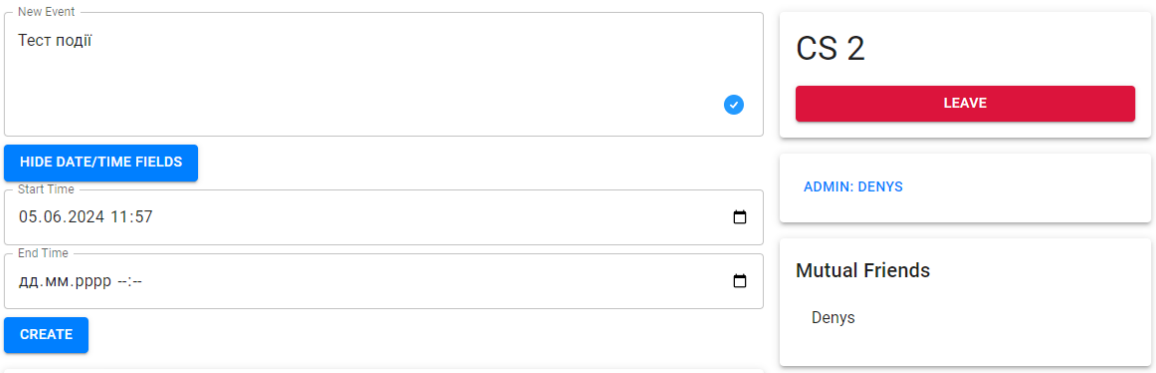


Рисунок 4.4 — Стан системи після введеня тексту події та натискання кнопки «Add Date/Time»

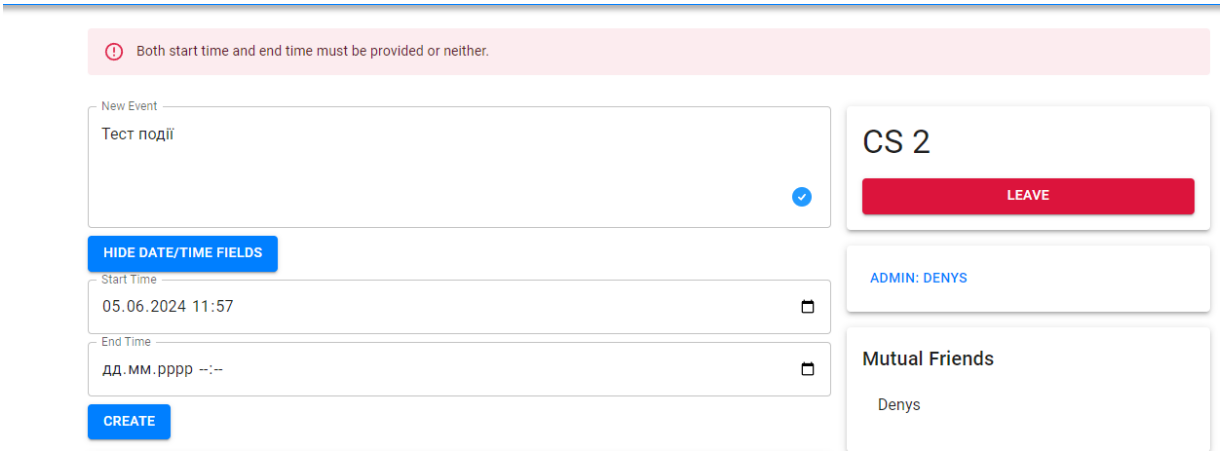


Рисунок 4.5 — Стан системи після натискання кнопки «Create»

Висновки до розділу

Проведений аналіз якості та тестування програмного забезпечення показав, що обрані інструменти та підходи забезпечують високий рівень надійності та безпеки розробленого продукту. Використання сервісу SonarCloud дозволило виявити та усунути потенційні проблеми в коді, зокрема забезпечити його надійність, безпеку та підтримуваність. Результати аналізу метрик показали відсутність критичних помилок, низький рівень технічного боргу та мінімальний рівень дублювання коду, що свідчить про високу якість програмного забезпечення.

Процеси тестування включали мануальне тестування основних функцій системи, таких як реєстрація користувачів, авторизація, створення спільнот та планування ігрових подій. Було розроблено та проведено тестові сценарії для перевірки коректності роботи всіх ключових функцій. Результати тестування підтвердили відповідність функціональних вимог, що забезпечує зручність використання та стабільність роботи системи.

Загалом розділ показав, що розроблене програмне забезпечення відповідає всім встановленим вимогам щодо якості, надійності та безпеки. Проведений аналіз і тестування підтвердили готовність системи до використання кінцевими користувачами, забезпечуючи високу продуктивність та зручність у роботі. Це створює передумови для успішного впровадження та подальшого розвитку ігрової соціальної платформи.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Програмне забезпечення було розгорнуте за допомогою Docker Compose, що дозволяє автоматизувати процес налаштування та розгортання сервісів, що складаються з декількох контейнерів. Цей підхід дозволяє забезпечити ізоляцію сервісів, легкість масштабування та спрощує управління залежностями.

Для початку роботи необхідно встановити Docker відповідно до вашого Linux дистрибутиву. Перевірити правильність встановлення можна наступною командою — рисунок 5.1.

```
(.venv) cerpany@lv-server:~/git/time-breaker$ docker --version
Docker version 24.0.5, build ced0996
(.venv) cerpany@lv-server:~/git/time-breaker$
```

Рисунок 5.1 — Встановлення Docker.

Для зручності розгортання був створений наступний Docker Compose файл — рисунок 5.2. Docker Compose описує, як повинні бути налаштовані та запущені Docker контейнери для застосунку.

```
services:
  api:
    build:
      context: ./api
      dockerfile: Dockerfile
    depends_on:
      - db
    restart: unless-stopped
    environment:
      - DATABASE_URL=postgresql://postgres:postgres@db/time-breaker
      - GRAPH_URL=neo4j://graph:7687
      - MAX_WORKERS=4
    ports:
      - "8000:8000"
    command: bash -c "uvicorn run:app --host 0.0.0.0 --port 8000 --workers ${MAX_WORKERS} --reload --forwarded-allow-ips='*'"
    networks:
      platform:
        ipv4_address: 10.1.0.2
    volumes:
      - ./api:/app
  db:
    image: "postgres:latest"
    restart: unless-stopped
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_DB=time-breaker
    volumes:
      - postgres_data_dev:/var/lib/postgresql/data
    networks:
      platform:
        ipv4_address: 10.1.0.3
  graph:
    image: "neo4j:latest"
    restart: unless-stopped
    ports:
      - "7474:7474"
      - "7687:7687"
    environment:
```

Рисунок 5.2 — Docker Compose файл.

Файл складається з наступних сервісів: api, db, graph і web. Api — сервіс, що відповідає за бекендову частину застосунку. Web — сервіс, що відповідає за фронтенд частину застосунку. DB — сервіс для PostgreSQL бази даних і graph — сервіс для Neo4j бази даних.

Для збірки та запуску проєкту використаємо команду docker compose up -d –build. Результат виконання команди зображений на рисунку 5.3.

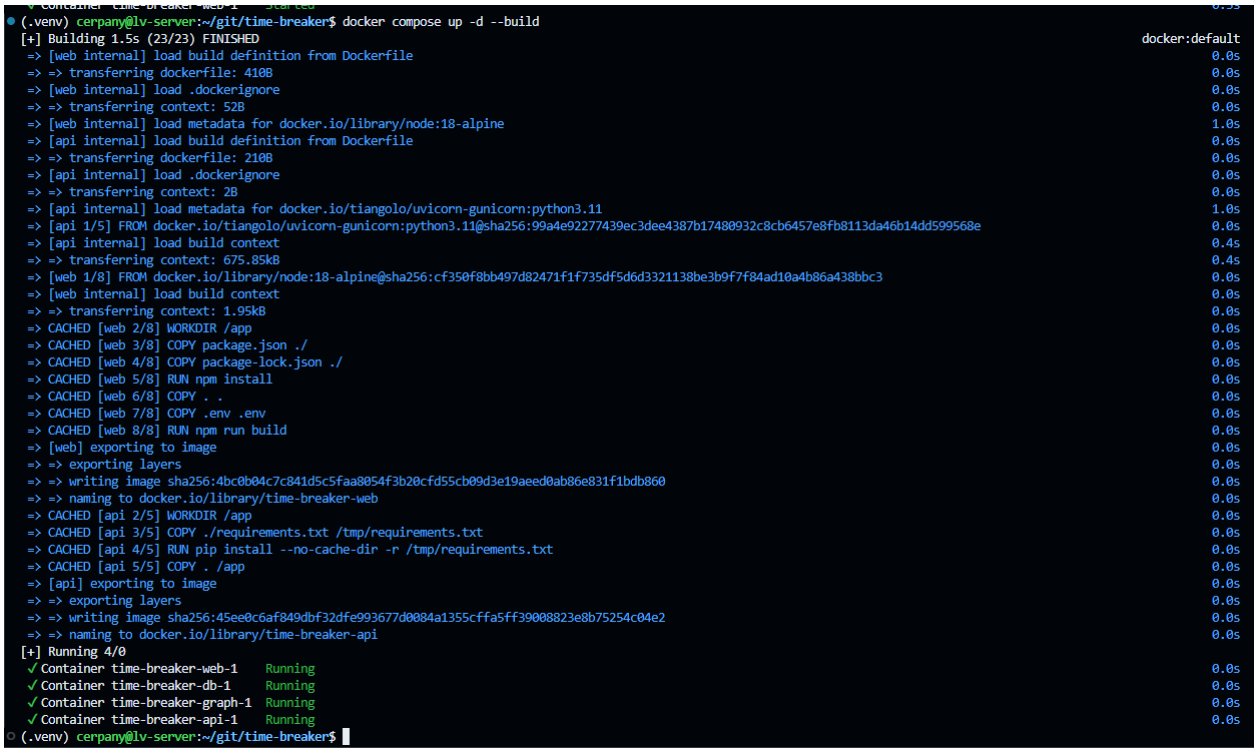


Рисунок 5.3 — Запуск проекту за допомогою docker compose.

З рисунку видно, що проект успішно зібрався. Перевірити стан запущених сервісів можна командою: docker compose ps — рисунок 5.4.

Рисунок 5.4 — Успішний запуск застосунку

5.2 Супровід програмного забезпечення

Супровід програмного забезпечення – це процес, що включає в себе постійну підтримку, вдосконалення та оновлення програмних систем після їх початкового розгортання. Цей процес важливий для забезпечення надійної роботи програмного забезпечення, виправлення помилок, вдосконалення

функціоналу та адаптації до нових вимог і середовищ. Основні завдання супроводу включають виявлення та виправлення помилок, оптимізацію коду, а також оновлення системи з урахуванням нових технологій і стандартів. Ефективний супровід допомагає продовжити життєвий цикл програмного забезпечення та забезпечити його відповідність потребам користувачів.

Для внесення змін у розроблене програмне забезпечення необхідно внести зміни у GitHub репозиторій проєкту. Розробник, вносить зміни на гілку пов’язану з функціональною вимогою, яку він виконує. Після проходження перевірки розробленого функціоналу на вимоги безпеки та надійності, зміни вносяться до головної гілки репозиторію. Після оновлення коду на сервері, необхідно здійснити перезбірку контейнерів для того, щоб кінцеві користувачі змогли користуватись найновішою версією продукту.

Висновки до розділу

У п'ятому розділі було розглянуто процес розгортання та супроводу програмного забезпечення для ігрової соціальної платформи. Основні аспекти включали розгортання за допомогою Docker Compose та принципи супроводу ПЗ.

Розгортання програмного забезпечення здійснювалося з використанням Docker Compose, що дозволило автоматизувати налаштування та запуск сервісів, необхідних для роботи платформи. Був створений Docker Compose файл, що описує конфігурацію сервісів, включаючи API, фронтенд, реляційну базу даних PostgreSQL та графову базу даних Neo4j. Цей підхід забезпечив ізоляцію компонентів, легкість масштабування та зручність управління залежностями. Було розглянуто процес встановлення Docker, збірки та запуску проєкту за допомогою командної стрічки, що підтвердило успішне розгортання платформи.

Супровід програмного забезпечення охоплює постійну підтримку, вдосконалення та оновлення системи після її розгортання. Цей процес включає виявлення та виправлення помилок, оптимізацію коду, впровадження нових функцій та адаптацію до змін у вимогах та технологіях. Важливими аспектами

супроводу є управління змінами через GitHub репозиторій, проведення перевірок на відповідність вимогам безпеки та надійності, а також оновлення контейнерів для забезпечення користувачів найновішою версією продукту.

Таким чином, у розділі було висвітлено ключові етапи та інструменти для розгортання та супроводу програмного забезпечення, що забезпечують надійність, масштабованість та зручність у підтримці ігрової соціальної платформи. Ці заходи сприяють стабільній роботі системи та задоволенню потреб користувачів у динамічному ігровому середовищі.

ВИСНОВОКИ

В результаті роботи над дипломним проєктом було спроєктовано та реалізована ігрова соціальна платформа. Ця платформа дозволяє формувати спільноти користувачів за їх вподобаннями та планувати спільні зустрічі для проведення онлайн ігор.

Ця платформа дає змогу покращити ігровий досвід користувачів: знайти однодумців, планувати спільні зустрічі, вести дискусії на теми улюблених ігор, отримувати рекомендації друзів та спільнот зі спільними інтересами. Високий рівень задоволення гравців стимулює розвиток ігрової індустрії. Соціальна платформа, як місце дискусій, стимулює популяризацію та ріст нішевих ігор, оскільки система рекомендацій і вільне місце для їх обговорення дають передумови для їх розвитку. Ріст та розвиток маловідомих ігор, який дає ця платформа, підкреслює соціально-економічну значусщість проєкту.

В якості середовища розробки був використаний редактор коду Visual Studio Code, функціонал якого дозволяє працювати з різноманітними мовами програмування, фреймворками та бібліотеками.

В якості БД використано Postgres, як реляційна база даних для зберігання загальної інформації про користувачів та спільноти, і Neo4j, як графова база даних для зберігання та оброблення зв’язків між спільнотами та користувачами.

У ході роботи над дипломним проєктом було вирішено наступні поставлені задачі:

— авторизація та введення профілю користувача;

— створення спільнот користувачів;

— планування зустрічей для різних спільнот;

— створення панелей дискусій відповідно до спільнот або запланованих зустрічей.

Таким чином, результати дипломного проєкту відповідають сучасним вимогам розробки програмного забезпечення та безпеки даних і забезпечують

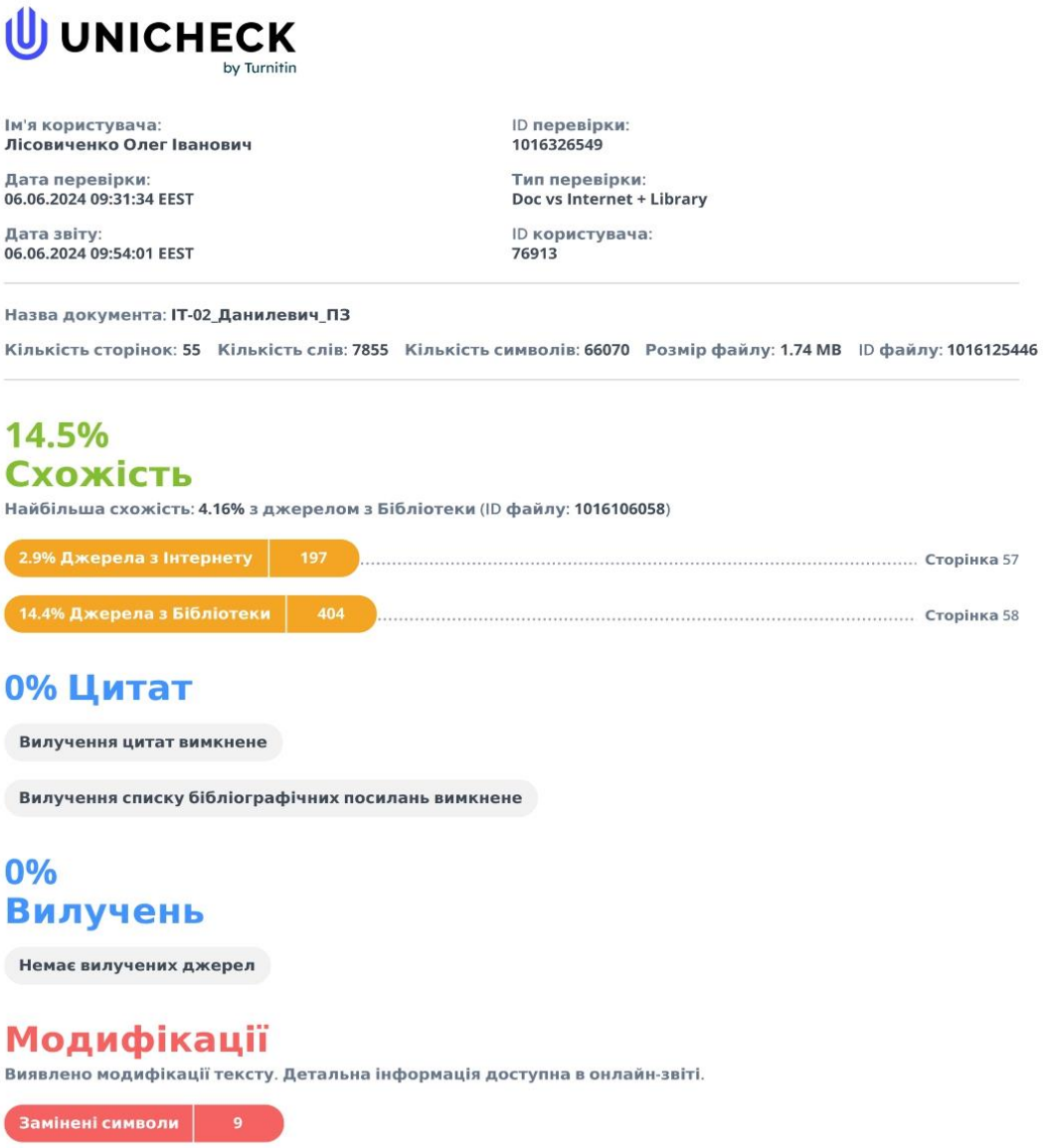
користувачів зручним інструментом для проведення ігрового дозвілля.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Відеоігри – Україна [Електронний ресурс] // Statista. — Режим доступу до ресурсу: https://www.statista.com/outlook/dmo/digital-media/video-games/ukraine. — Назва з екрана.
2) Найпопулярніші соціальні медіа-платформи для геймерів у США щодо ігор станом на серпень 2022 року [Електронний ресурс] // Statista. — Режим доступу до ресурсу: https://www.statista.com/statistics/1351103/us-most-used-social-related-to-gaming/. — Назва з екрана.
3) Python [Електронний ресурс] — Режим доступу до ресурсу https://www.python.org/.
4) FastAPI [Електронний ресурс] — Режим доступу до ресурсу https://fastapi.tiangolo.com/.
5) React [Електронний ресурс] – Режим доступу до ресурсу https://react.dev/
6) PostgreSQL [Електронний ресурс] — Режим доступу до ресурсу https://www.postgresql.org/.
7) Neo4j [Електронний ресурс] — Режим доступу до ресурсу https://neo4j.com/.
8) Docker [Електронний ресурс] — Режим доступу до ресурсу https://www.docker.com/.
9) Alembic [Електронний ресурс] — Режим доступу до ресурсу https://alembic.sqlalchemy.org/en/latest/.

ДОДАТКИ

ДОДАТОК А Звіт подібності



UNICHECK
by Turnitin

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016326549

Дата перевірки:
06.06.2024 09:31:34 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
06.06.2024 09:54:01 EEST

ID користувача:
76913

Назва документа: IT-02_Данилевич_ПЗ

Кількість сторінок: 55 Кількість слів: 7855 Кількість символів: 66070 Розмір файлу: 1.74 MB ID файлу: 1016125446

14.5%
Схожість

Найбільша схожість: 4.16% з джерелом з Бібліотеки (ID файлу: 1016106058)

2.9% Джерела з Інтернету 197 ..... Сторінка 57

14.4% Джерела з Бібліотеки 404 ..... Сторінка 58

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 9

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

ВЕБ-ЗАСТОСУНОК ДЛЯ ІГРОВОЇ СОЦІАЛЬНОЇ ПЛАТФОРМИ

Текст програми

КПІ.ІТ-9402.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Антон ДАНИЛЕВИЧ

Київ – 2024

Посилання на репозиторій з повним текстом програмного коду

https://github.com/A-Danylevych/time-breaker

Файл CalendarPage.jsx

Реалізація функціональних вимог: Система планування зустрічей, Видалення зустрічі.

```
import React, { useState, useEffect } from 'react';
import { Calendar, momentLocalizer } from 'react-big-calendar';
import moment from 'moment';
import axios from 'axios';
import 'react-big-calendar/lib/css/react-big-calendar.css';
import './CalendarPage.css';
const localizer = momentLocalizer(moment);

const CalendarPage = () => {
  const [events, setEvents] = useState([]);

  useEffect(() => {
    fetchEvents();
  }, []);

  const fetchEvents = async () => {
    try {
      const token = localStorage.getItem('token');
      if (!token) {
        throw new Error('No token found');
      }
      const response = await
axios.get(`${import.meta.env.VITE_API_URL}/api/v1/calendar/events`, {
        headers: {
          Authorization: `Bearer ${token}`,
        },
      });
      const formattedEvents = response.data.map(event => ({
        ...event,
        title: event.username,
        description: event.content,
        start: new Date(event.start_time),
        end: new Date(event.end_time),
      }));
      setEvents(formattedEvents);
    } catch (error) {
      console.error('Error fetching events:', error);
    }
  };

  const unsubscribeEvent = async (eventId) => {
    try {
      const token = localStorage.getItem('token');
      if (!token) {
        throw new Error('No token found');
      }
      await
axios.post(`${import.meta.env.VITE_API_URL}/api/v1/calendar/posts/${eventId}/
unsubscribe`, {}, {
        headers: {
          Authorization: `Bearer ${token}`,
        },
```

```
      });
      fetchEvents();
    } catch (error) {
      console.error('Error unsubscribing from event:', error);
    }
  };

  const handleSelectEvent = (event) => {
    if (window.confirm(`Are you sure you want to unsubscribe from the event
'${event.title}'?`)) {
      unsubscribeEvent(event.id);
    }
  };

  return (
    <div className="calendar-wrapper">
      <div className="calendar-container">
        <Calendar
          localizer={localizer}
          events={events}
          startAccessor="start"
          endAccessor="end"
          defaultView="week"
          selectable
          onSelectEvent={handleSelectEvent}
          tooltipAccessor={(event) => `${event.title}: ${event.description}`}
        />
      </div>
    </div>
  );
};

export default CalendarPage;
```

Файл CommunityPage.jsx

Реалізація функціональних вимог: Система спільнот, Створення дискусії, Створення коментарю, Видалення дискусії, Підписка до спільноти, Вихід із спільноти.

```
import React, { useState, useEffect } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import axios from 'axios';
import { Button, Container, Typography, TextField, Grid, Box, Alert, Paper,
Divider, IconButton, List, ListItem, ListItemText } from '@mui/material';
import DeleteIcon from '@mui/icons-material/Delete';

const CommunityPage = () => {
  const { communityId } = useParams();
  const navigate = useNavigate();
  const [community, setCommunity] = useState(null);
  const [posts, setPosts] = useState([]);
  const [newPostContent, setNewPostContent] = useState('');
  const [newCommentContent, setNewCommentContent] = useState('');
  const [isAdmin, setIsAdmin] = useState(false);
  const [isMember, setIsMember] = useState(false);
  const [token, setToken] = useState(localStorage.getItem('token'));
  const [error, setError] = useState(null);
  const [loadingPosts, setLoadingPosts] = useState(false);
  const [loadingComments, setLoadingComments] = useState({});
  const [hasMorePosts, setHasMorePosts] = useState(true);
  const [subscriptions, setSubscriptions] = useState({});
```

```
  const [showSubscribers, setShowSubscribers] = useState({});
  const [mutualFriends, setMutualFriends] = useState([]);
  const [showDateTimeFields, setShowDateTimeFields] = useState(false);
  const [startTime, setStartTime] = useState('');
  const [endTime, setEndTime] = useState('');

  const fetchPosts = async (olderThan = null, newPage = false) => {
    if ((loadingPosts || !hasMorePosts) && !newPage) return;
    setLoadingPosts(true);
    try {
      const response = await
axios.get(`${import.meta.env.VITE_API_URL}/api/v1/post/`, {
        params: {
          parent_id: communityId,
          older_than: olderThan,
        },
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      const postsData = response.data.map(post => ({ ...post, showComments:
false, comments: [], commentsLoaded: false }));
      for (const post of postsData) {
        await checkSubscriptionStatus(post.id);
      }
      if (response.data.length < 10) setHasMorePosts(false);
      if (newPage) {
        setPosts(postsData);
      } else {
        setPosts(prevPosts => [...prevPosts, ...postsData]);
      }
    } catch (error) {
      setError('Failed to fetch posts.');
    } finally {
      setLoadingPosts(false);
    }
  };

  const fetchCommunity = async () => {
    try {
      const response = await
axios.get(`${import.meta.env.VITE_API_URL}/api/v1/communities/${communityId}`
, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      setCommunity(response.data);
      setIsAdmin(response.data.admin_id ===
localStorage.getItem('myProfileId'));
      fetchRelationshipStatus(response.data.id);
      fetchPosts(null, true);
    } catch (error) {
      setError('Failed to fetch community.');
    }
  };

  const fetchRelationshipStatus = async (otherProfileId) => {
    try {
      const response = await
axios.get(`${import.meta.env.VITE_API_URL}/api/v1/relationship/${localStorage
.getItem('myProfileId')}/relationships/${otherProfileId}`, {
        headers: {
          Authorization: `Bearer ${token}`
```

```
        }
      });
      setIsMember(response.data === 'friend');
    } catch (error) {
      setError('Failed to fetch relationship status.');
    }
  };

  const fetchMutualFriends = async () => {
    try {
      const response = await
axios.get(`${import.meta.env.VITE_API_URL}/api/v1/relationship/${localStorage
.getItem('myProfileId')}/friends/${communityId}/mutual_friends`, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      setMutualFriends(response.data);
    } catch (error) {
      setError('Failed to fetch mutual friends.');
    }
  };

  const validateDateTimeFields = () => {
    if ((startTime && !endTime) || (!startTime && endTime)) {
      setError('Both start time and end time must be provided or neither.');
      return false;
    }
    return true;
  };

  const handleNewPost = async () => {
    if (!validateDateTimeFields()) return;

    try {
      const response = await
axios.post(`${import.meta.env.VITE_API_URL}/api/v1/post/`, {
        content: newPostContent,
        parentId: community.id,
        username: community.username,
        startTime: startTime || null,
        endTime: endTime || null,
      }, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      setPosts([{ ...response.data, showComments: false, comments: [],
commentsLoaded: false }, ...posts]);
      setNewPostContent('');
      setStartTime('');
      setEndTime('');
      setError(null);
    } catch (error) {
      setError('Event cannot be empty.');
    }
  };

  const handleNewComment = async (postId) => {
    try {
      const response = await
axios.post(`${import.meta.env.VITE_API_URL}/api/v1/comment/posts/${postId}/co
mments`, {
        content: newCommentContent,
```

```
      }, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      setPosts(posts.map(post =>
        post.id === postId
          ? { ...post, comments: [response.data, ...post.comments] }
          : post
      ));
      setNewCommentContent('');
    } catch (error) {
      setError('Failed to create comment.');
    }
  };

  const handleDeletePost = async (postId) => {
    try {
      await
axios.delete(`${import.meta.env.VITE_API_URL}/api/v1/post/${postId}`, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      setPosts(posts.filter(post => post.id !== postId));
    } catch (error) {
      setError('Failed to delete post.');
    }
  };

  const handleToggleComments = async (postId) => {
    const updatedPosts = posts.map(post => {
      if (post.id === postId) {
        post.showComments = !post.showComments;
        if (!post.commentsLoaded && post.showComments) {
          fetchComments(post.id);
        }
      }
      return post;
    });
    setPosts(updatedPosts);
  };

  const fetchComments = async (postId, olderThan = null) => {
    setLoadingComments(prevState => ({ ...prevState, [postId]: true }));
    try {
      const response = await
axios.get(`${import.meta.env.VITE_API_URL}/api/v1/comment/posts/${postId}/com
ments`, {
        params: {
          older_than: olderThan,
        },
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      setPosts(posts.map(post =>
        post.id === postId
          ? { ...post, comments: [...response.data, ...post.comments],
commentsLoaded: true }
          : post
      ));
    } catch (error) {
      setError('Failed to fetch comments.');
```

```
    } finally {
      setLoadingComments(prevState => ({ ...prevState, [postId]: false }));
    }
  };

  const loadMoreComments = (postId) => {
    const post = posts.find(post => post.id === postId);
    const lastComment = post.comments[post.comments.length - 1];
    const lastCommentDate = lastComment ? new
Date(lastComment.created_date).toISOString() : null;
    fetchComments(postId, lastCommentDate);
  };

  const handleSubscribe = async (postId) => {
    try {
      await
axios.post(`${import.meta.env.VITE_API_URL}/api/v1/calendar/posts/${postId}/s
ubscribe`, {}, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      setSubscriptions(prevState => ({ ...prevState, [postId]: true }));
    } catch (error) {
      setError('Failed to subscribe.');
    }
  };

  const handleUnsubscribe = async (postId) => {
    try {
      await
axios.post(`${import.meta.env.VITE_API_URL}/api/v1/calendar/posts/${postId}/u
nsubscribe`, {}, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      setSubscriptions(prevState => ({ ...prevState, [postId]: false }));
    } catch (error) {
      setError('Failed to unsubscribe.');
    }
  };

  const checkSubscriptionStatus = async (postId) => {
    try {
      const response = await
axios.get(`${import.meta.env.VITE_API_URL}/api/v1/calendar/posts/${postId}`,
{
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      setSubscriptions(prevState => ({ ...prevState, [postId]:
response.data.isSubscribed }));
    } catch (error) {
      setError('Failed to fetch subscription status.');
    }
  };

  const handleToggleSubscribers = (postId) => {
    setShowSubscribers(prevState => ({ ...prevState, [postId]:
!prevState[postId] }));
    if (!showSubscribers[postId]) {
      fetchSubscribers(postId);
```

```
    }
  };

  const fetchSubscribers = async (postId) => {
    if (!subscriptions[postId]) return;
    try {
      const response = await
axios.get(`${import.meta.env.VITE_API_URL}/api/v1/calendar/posts/${postId}/su
bscribers`, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      setSubscribers(prevState => ({ ...prevState, [postId]: response.data
}));
    } catch (error) {
      setError('Failed to fetch subscribers.');
    }
  };

  const handleJoinCommunity = async () => {/add/
    try {
      await
axios.post(`${import.meta.env.VITE_API_URL}/api/v1/communities/profiles/${loc
alStorage.getItem('myProfileId')}/add/${communityId}`, {}, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      setIsMember(true);
    } catch (error) {
      setError('Failed to join community.');
    }
  };

  const handleLeaveCommunity = async () => {
    try {
      await
axios.delete(`${import.meta.env.VITE_API_URL}/api/v1/communities/profiles/${l
ocalStorage.getItem('myProfileId')}/remove/${communityId}`, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      setIsMember(false);
    } catch (error) {
      setError('Failed to leave community.');
    }
  };

  useEffect(() => {
    fetchCommunity();
    fetchMutualFriends();
  }, [communityId, token]);

  useEffect(() => {
    const handleScroll = () => {
      if (window.innerHeight + document.documentElement.scrollTop ===
document.documentElement.offsetHeight) {
        const lastPost = posts[posts.length - 1];
        const lastPostDate = lastPost ? new
Date(lastPost.createdDate).toISOString() : null;
        fetchPosts(lastPostDate);
      }
```

```
    };

    window.addEventListener('scroll', handleScroll);
    return () => window.removeEventListener('scroll', handleScroll);
  }, [posts, hasMorePosts, loadingPosts]);

  return (
    <Container>
      {error && <Alert severity="error">{error}</Alert>}
      <Grid container spacing={2} marginTop={2}>
        <Grid item xs={12} md={8}>
          {community && (
            <Box marginBottom={2}>
              <TextField
                label="New Event"
                fullWidth
                multiline
                rows={4}
                value={newPostContent}
                onChange={(e) => setNewPostContent(e.target.value)}
              />
              <Button
                variant="contained"
                color="primary"
                onClick={() => setShowDateTimeFields(!showDateTimeFields)}
                sx={{ mt: 1 }}
              >
                {showDateTimeFields ? 'Hide Date/Time Fields' : 'Add
Date/Time'}
              </Button>
              {showDateTimeFields && (
                <>
                  <TextField
                    label="Start Time"
                    type="datetime-local"
                    fullWidth
                    value={startTime}
                    onChange={(e) => setStartTime(e.target.value)}
                    sx={{ mt: 1 }}
                    InputLabelProps={{
                      shrink: true,
                    }}
                  />
                  <TextField
                    label="End Time"
                    type="datetime-local"
                    fullWidth
                    value={endTime}
                    onChange={(e) => setEndTime(e.target.value)}
                    sx={{ mt: 1 }}
                    InputLabelProps={{
                      shrink: true,
                    }}
                  />
                </>
              )}
              <Button variant="contained" color="primary"
onClick={handleNewPost} sx={{ mt: 1 }}>
                Create
              </Button>
            </Box>
          )}
          {posts.map((post, index) => (
```

```
            <Box key={index} component={Paper} elevation={3} padding={2}
marginBottom={2} position="relative">
              <Typography variant="h6">{post.username}</Typography>
              <Typography variant="caption" color="textSecondary">{new
Date(post.createdDate).toLocaleString()}</Typography>
              <Typography variant="body1" sx={{ marginTop: 1
}}>{post.content}</Typography>
              {post.startTime && post.endTime && (
                <Typography variant="body2" color="textSecondary">
                  Event Time: {new Date(post.startTime).toLocaleString()} -
{new Date(post.endTime).toLocaleString()}
                </Typography>
              )}
              <Divider sx={{ my: 2 }} />
              {isAdmin && (
                <IconButton
                  onClick={() => handleDeletePost(post.id)}
                  sx={{ position: 'absolute', top: 8, right: 8 }}
                >
                  <DeleteIcon />
                </IconButton>
              )}
              <Button variant="text" color="primary" onClick={() =>
handleToggleComments(post.id)}>
                {post.showComments ? 'Hide Comments' : 'Show Comments'}
              </Button>
              {post.showComments && (
                <>
                  {post.comments.map((comment, idx) => (
                    <Box key={idx} sx={{ mb: 1 }}>
                      <Typography
variant="body2"><strong>{comment.username}:</strong>
{comment.content}</Typography>
                      <Typography variant="caption"
color="textSecondary">{new
Date(comment.created_date).toLocaleString()}</Typography>
                    </Box>
                  ))}
                  <Button variant="text" color="primary" onClick={() =>
loadMoreComments(post.id)} disabled={loadingComments[post.id]}>
                    {loadingComments[post.id] ? 'Loading...' : 'Load More
Comments'}
                  </Button>
                  <TextField
                    label="New Comment"
                    fullWidth
                    multiline
                    rows={2}
                    value={newCommentContent}
                    onChange={(e) => setNewCommentContent(e.target.value)}
                    sx={{ mt: 2 }}
                  />
                  <Button variant="contained" color="primary" onClick={() =>
handleNewComment(post.id)} sx={{ mt: 1 }}>
                    Comment
                  </Button>
                </>
              )}
              {post.startTime && post.endTime && (
                <Box display="flex" justifyContent="space-between"
alignItems="center">
                  <Box>
                    {subscriptions[post.id] ? (
```

```
                      <Button variant="contained" color="secondary"
onClick={() => handleUnsubscribe(post.id)}>
                        Unsubscribe
                      </Button>
                    ) : (
                      <Button variant="contained" color="primary" onClick={()
=> handleSubscribe(post.id)}>
                        Subscribe
                      </Button>
                    )}
                  </Box>
                </Box>
              )}
              {post.startTime && post.endTime ? (
                <Button variant="text" color="primary" onClick={() =>
handleToggleSubscribers(post.id)}>
                  {showSubscribers[post.id] ? 'Hide Subscribers' : 'Show
Subscribers'}
                </Button>
              ) : null}
              {showSubscribers[post.id] && subscribers[post.id] && (
                <List>
                  {subscribers[post.id].map((subscriber, idx) => (
                    <ListItem key={idx}>
                      <ListItemText primary={subscriber.username} />
                    </ListItem>
                  ))}
                </List>
              )}
            </Box>
          ))}
        </Grid>
        <Grid item xs={12} md={4}>
          {community && (
            <>
              <Box component={Paper} elevation={3} padding={2}
marginBottom={2}>
                <Typography variant="h4">{community.username}</Typography>
                <Typography
variant="body1">{community.description}</Typography>
                <Box marginTop={2}>
                  {!isAdmin && (
                    isMember ? (
                      <Button variant="contained" color="error" fullWidth
onClick={handleLeaveCommunity}>
                        Leave
                      </Button>
                    ) : (
                      <Button variant="contained" color="primary" fullWidth
onClick={handleJoinCommunity}>
                        Join
                      </Button>
                    )
                  )}
                </Box>
              </Box>
              <Box component={Paper} elevation={3} padding={2}
marginBottom={2}>
                <Typography variant="h6">
                  <Button onClick={() =>
navigate(`/profile/${community.admin_id}`)}>{`Admin:
${community.admin_username}`}</Button>
                </Typography>
              </Box>
```

```
              <Box component={Paper} elevation={3} padding={2}
marginBottom={2}>
                <Typography variant="h6">Mutual Friends</Typography>
                {mutualFriends.length > 0 ? (
                  <List>
                    {mutualFriends.map((friend, index) => (
                      <ListItem key={index} button onClick={() =>
navigate(`/profile/${friend.id}`)}>
                        <ListItemText primary={friend.username} />
                      </ListItem>
                    ))}
                  </List>
                ) : (
                  <Typography>No mutual friends found.</Typography>
                )}
              </Box>
            </>
          )}
        </Grid>
      </Grid>
    </Container>
  );
};

export default CommunityPage;
```

Файл LoginPage.jsx

Реалізація функціональної вимоги: Авторизація користувача

```
import React, { useState } from 'react';
import { TextField, Button, Container, Typography, Grid, Box, Alert, Link }
from '@mui/material';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

const LoginPage = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState(null);
  const navigate = useNavigate();

  const handleLogin = async (e) => {
    e.preventDefault();
    try {
      const apiUrl = import.meta.env.VITE_API_URL;
      console.log(`${apiUrl}/api/v1/auth/login`);
      const response = await axios.post(`${apiUrl}/api/v1/auth/login`, {
        username,
        password,
      }, {
        headers: {
          'Content-Type': 'application/json'
        }
      });
      console.log(response.data)
      const token = response.data.accessToken;
      localStorage.setItem('token', token);
      console.log(response.data)
      const userId = response.data.profileId;
      localStorage.setItem('myProfileId', userId);
      console.log('Login successful:', response.data);
      navigate('/');
    } catch (error) {
```

```
      console.log(error)
      if (error.response) {
        setError(error.response.data.detail);
      } else {
        setError('Login failed');
      }
    }
  };

  return (
    <Container maxWidth="sm">
      <Box sx={{ mt: 5 }}>
        <Typography variant="h4" component="h1" gutterBottom>
          Login
        </Typography>
        {error && <Alert severity="error">{error}</Alert>}
        <form onSubmit={handleLogin}>
          <Grid container spacing={2}>
            <Grid item xs={12}>
              <TextField
                fullWidth
                label="Username"
                value={username}
                onChange={(e) => setUsername(e.target.value)}
                required
              />
            </Grid>
            <Grid item xs={12}>
              <TextField
                fullWidth
                label="Password"
                type="password"
                value={password}
                onChange={(e) => setPassword(e.target.value)}
                required
              />
            </Grid>
            <Grid item xs={12}>
              <Button type="submit" variant="contained" color="primary"
fullWidth>
                Login
              </Button>
            </Grid>
          </Grid>
        </form>
        <Box sx={{ mt: 2 }}>
          <Typography>
            Don't have an account? <Link href="/register">Register</Link>
          </Typography>
        </Box>
      </Box>
    </Container>
  );
};

export default LoginPage;
```

Файл CommunicationsPage.jsx

Реалізація функціональних вимог: Знайти друга, Видалити з друзів, Відхилити запрошення.

```
import React, { useState, useEffect } from 'react';
```

```
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import { Button, Container, Typography, TextField, Grid, Box, Alert, Paper,
List, ListItem, ListItemText, ListItemSecondaryAction, Dialog, DialogTitle,
DialogContent, DialogActions } from '@mui/material';

const CommunicationsPage = () => {
  const [searchQuery, setSearchQuery] = useState('');
  const [searchResults, setSearchResults] = useState([]);
  const [friends, setFriends] = useState([]);
  const [communities, setCommunities] = useState([]);
  const [incomingRequests, setIncomingRequests] = useState([]);
  const [outgoingRequests, setOutgoingRequests] = useState([]);
  const [error, setError] = useState(null);
  const [isCreateCommunityDisabled, setIsCreateCommunityDisabled] =
useState(true);
  const [isDialogOpen, setIsDialogOpen] = useState(false);
  const [token, setToken] = useState(localStorage.getItem('token'));
  const navigate = useNavigate();

  const handleSearch = async () => {
    try {
      const response = await
axios.get(`${import.meta.env.VITE_API_URL}/api/v1/profile/profiles`, {
        params: { username: searchQuery },
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      setSearchResults(response.data);
      const matchFound = response.data.some(result => result.username ===
searchQuery || result.name === searchQuery);
      setIsCreateCommunityDisabled(matchFound);
    } catch (error) {
      setError('Failed to search.');
    }
  };

  const handleSearchResultClick = (result) => {
    if (result.type === 'user') {
      navigate(`/profile/${result.id}`);
    } else if (result.type === 'community') {
      navigate(`/community/${result.id}`);
    }
  };

  const fetchFriendsAndCommunities = async () => {
    try {
      const response = await
axios.get(`${import.meta.env.VITE_API_URL}/api/v1/relationship/${localStorage
.getItem('myProfileId')}/friends`, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      const friends = response.data.filter(profile => profile.type ===
'user');
      const communities = response.data.filter(profile => profile.type ===
'community');
      setFriends(friends);
      setCommunities(communities);
    } catch (error) {
      setError('Failed to fetch friends and communities.');
    }
```

```
  };

  const fetchRequests = async () => {
    try {
      const response_outgoing = await
axios.get(`${import.meta.env.VITE_API_URL}/api/v1/relationship/${localStorage
.getItem('myProfileId')}/friend_requests`, {
        headers: {
          Authorization: `Bearer ${token}`
        },
        params: { direction: 'outgoing' }
      });
      const response_incoming = await
axios.get(`${import.meta.env.VITE_API_URL}/api/v1/relationship/${localStorage
.getItem('myProfileId')}/friend_requests`, {
        headers: {
          Authorization: `Bearer ${token}`
        },
        params: { direction: 'incoming' }
      });
      setIncomingRequests(response_incoming.data || []);
      setOutgoingRequests(response_outgoing.data || []);
    } catch (error) {
      setError('Failed to fetch friend requests.');
    }
  };

  const handleAcceptFriendRequest = async (requesterProfileId) => {
    try {
      await
axios.post(`${import.meta.env.VITE_API_URL}/api/v1/relationship/${localStorag
e.getItem('myProfileId')}/friends/${requesterProfileId}`, {}, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      fetchRequests();
    } catch (error) {
      setError('Failed to accept friend request.');
    }
  };

  const handleRejectIncomingRequest = async (requesterProfileId) => {
    try {
      await
axios.delete(`${import.meta.env.VITE_API_URL}/api/v1/relationship/${localStor
age.getItem('myProfileId')}/incoming_friend_requests/${requesterProfileId}`,
{
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      fetchRequests();
    } catch (error) {
      setError('Failed to reject friend request.');
    }
  };

  const handleCancelOutgoingRequest = async (targetProfileId) => {
    try {
      await
axios.delete(`${import.meta.env.VITE_API_URL}/api/v1/relationship/${localStor
age.getItem('myProfileId')}/outgoing_friend_requests/${targetProfileId}`, {
        headers: {
```

```
          Authorization: `Bearer ${token}`
        }
      });
      fetchRequests();
    } catch (error) {
      setError('Failed to cancel friend request.');
    }
  };

  const handleCreateCommunity = async () => {
    try {
      const response = await
axios.post(`${import.meta.env.VITE_API_URL}/api/v1/communities`, {
        name: searchQuery,
        creatorProfileId: localStorage.getItem('myProfileId')
      }, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      navigate(`/community/${response.data.id}`);
    } catch (error) {
      setError('Failed to create community.');
    }
  };

  useEffect(() => {
    fetchFriendsAndCommunities();
    fetchRequests();
  }, []);

  return (
    <Container>
      {error && <Alert severity="error">{error}</Alert>}
      <Grid container spacing={2} marginTop={2}>
        {/* Search Section */}
        <Grid item xs={12} md={4}>
          <Box component={Paper} elevation={3} padding={2}>
            <Typography variant="h6">Search</Typography>
            <TextField
              label="Search by name"
              fullWidth
              value={searchQuery}
              onChange={(e) => setSearchQuery(e.target.value)}
              sx={{ mt: 1 }}
            />
            <Button variant="contained" color="primary" fullWidth
onClick={handleSearch} sx={{ mt: 1 }}>
              Search
            </Button>
            <List>
              {searchResults.length > 0 ? searchResults.map((result, index)
=> (
                <ListItem key={index} button onClick={() =>
handleSearchResultClick(result)}>
                  <ListItemText primary={result.username || result.name} />
                </ListItem>
              )) : <ListItem><ListItemText primary="No results found"
/></ListItem>}
            </List>
            <Button variant="contained" color="secondary" fullWidth
disabled={isCreateCommunityDisabled} onClick={() => setIsDialogOpen(true)}
sx={{ mt: 1 }}>
              Create Community
```

```
            </Button>
          </Box>
        </Grid>

        {/* Friends and Communities Section */}
        <Grid item xs={12} md={4}>
          <Box component={Paper} elevation={3} padding={2}>
            <Typography variant="h6">My Friends</Typography>
            <List>
              {friends.length > 0 ? friends.map((friend, index) => (
                <ListItem key={index} button onClick={() =>
navigate(`/profile/${friend.id}`)}>
                  <ListItemText primary={friend.username} />
                </ListItem>
              )) : <ListItem><ListItemText primary="No friends found"
/></ListItem>}
            </List>
            <Typography variant="h6" sx={{ mt: 2 }}>My
Communities</Typography>
            <List>
              {communities.length > 0 ? communities.map((community, index) =>
(
                <ListItem key={index} button onClick={() =>
navigate(`/community/${community.id}`)}>
                  <ListItemText primary={community.username} />
                </ListItem>
              )) : <ListItem><ListItemText primary="No communities found"
/></ListItem>}
            </List>
          </Box>
        </Grid>

        {/* Incoming and Outgoing Requests Section */}
        <Grid item xs={12} md={4}>
          <Box component={Paper} elevation={3} padding={2}>
            <Typography variant="h6">Incoming Friend Requests</Typography>
            <List>
              {incomingRequests.length > 0 ? incomingRequests.map((request,
index) => (
                <ListItem key={index} button onClick={() =>
navigate(`/profile/${request.id}`)}>
                  <ListItemText primary={request.username} />
                  <ListItemSecondaryAction>
                    <Button variant="contained" color="primary" onClick={()
=> handleAcceptFriendRequest(request.id)}>
                      Accept
                    </Button>
                    <Button variant="contained" color="error" onClick={() =>
handleRejectIncomingRequest(request.id)} sx={{ ml: 1 }}>
                      Reject
                    </Button>
                  </ListItemSecondaryAction>
                </ListItem>
              )) : <ListItem><ListItemText primary="No incoming requests"
/></ListItem>}
            </List>
            <Typography variant="h6" sx={{ mt: 2 }}>Outgoing Friend
Requests</Typography>
            <List>
              {outgoingRequests.length > 0 ? outgoingRequests.map((request,
index) => (
                <ListItem key={index} button onClick={() =>
navigate(`/profile/${request.id}`)}>
                  <ListItemText primary={request.username} />
```

```
                  <ListItemSecondaryAction>
                    <Button variant="contained" color="error" onClick={() =>
handleCancelOutgoingRequest(request.id)}>
                      Cancel
                    </Button>
                  </ListItemSecondaryAction>
                </ListItem>
              )) : <ListItem><ListItemText primary="No outgoing requests"
/></ListItem>}
            </List>
          </Box>
        </Grid>
      </Grid>
      <Dialog open={isDialogOpen} onClose={() => setIsDialogOpen(false)}>
        <DialogTitle>Confirm Community Creation</DialogTitle>
        <DialogContent>
          <Typography>
            Do you really want to create a community with the name
"{searchQuery}"?
          </Typography>
        </DialogContent>
        <DialogActions>
          <Button onClick={() => setIsDialogOpen(false)}
color="primary">Cancel</Button>
          <Button onClick={() => {
            handleCreateCommunity();
            setIsDialogOpen(false);
          }} color="primary">Create</Button>
        </DialogActions>
      </Dialog>
    </Container>
  );
}

export default CommunicationsPage;
```

Файл calendar_controller.py

Реалізація функціональних вимог: Система планування зустрічей, Видалення зустрічі, Створити нову зустріч.

```
from fastapi import APIRouter, Depends
from uuid import UUID
from pydantic import BaseModel
from common.models import User
from common.token import retrieve_user_from_token
from services.calendar import subscribe_to_post_service,
unsubscribe_from_post_service, check_subscription_status_service,
get_subscribers_service, get_events_service

calendar_router = APIRouter()

class SubscribeRequest(BaseModel):
    post_id: UUID

@calendar_router.post("/posts/{post_id}/subscribe")
async def subscribe_to_post(post_id: UUID, current_user: User =
Depends(retrieve_user_from_token)):
    """
    Subscribe to a post.

    Args:
    post_id: UUID - The post id.
```

```
    current_user: User - The user data.

    Returns:
    Dict: The response message.

    Raises:
    HTTPException: If the user is already subscribed.
    """

    return await subscribe_to_post_service(post_id, current_user)

@calendar_router.post("/posts/{post_id}/unsubscribe")
async def unsubscribe_from_post(post_id: UUID, current_user: User =
Depends(retrieve_user_from_token)):
    """
    Unsubscribe from a post.

    Args:
    post_id: UUID - The post id.
    current_user: User - The user data.

    Returns:
    Dict: The response message.

    Raises:
    HTTPException: If the user is not subscribed.
    """

    return await unsubscribe_from_post_service(post_id, current_user)

@calendar_router.get("/posts/{post_id}")
async def check_subscription_status(post_id: UUID, current_user: User =
Depends(retrieve_user_from_token)):
    """
    Check the subscription status of a post.

    Args:
    post_id: UUID - The post id.
    current_user: User - The user data.

    Returns:
    Dict: The response message.
    """

    return await check_subscription_status_service(post_id, current_user)

@calendar_router.get("/posts/{post_id}/subscribers")
async def get_subscribers(post_id: UUID, current_user: User =
Depends(retrieve_user_from_token)):
    """
    Get subscribers of a post.

    Args:
    post_id: UUID - The post id.
    current_user: User - The user data.

    Returns:
    List[Dict]: The subscribers data.
    """



    return await get_subscribers_service(post_id, current_user)

@calendar_router.get("/events")
```

```
async def get_events(current_user: User = Depends(retrieve_user_from_token)):
    """
    Get the events of a user.

    Args:
    current_user: User - The user data.

    Returns:
    List[Dict]: The events data.
    """

    return await get_events_service(current_user)
```

Файл calendar_service.py

Реалізація функціональних вимог: Система планування зустрічей, Видалення зустрічі, Створити нову зустріч.

```
from uuid import UUID, uuid4
from sqlalchemy import select
from fastapi import HTTPException, status
from common.models import User
from infrastructure.calendar import profile_post
from infrastructure.profile import profile
from infrastructure.post import post
from api.infrastructure.db import db_conn

async def subscribe_to_post_service(post_id: UUID, current_user: User):
    """
    Subscribe to a post.

    Args:
    post_id: UUID - The post id.
    current_user: User - The user data.

    Returns:
    Dict: The response message.

    Raises:
    HTTPException: If the user is already subscribed.
    """

    subscription = await
db_conn.fetch_one(profile_post.select().where(profile_post.c.profile_id ==
current_user.id, profile_post.c.post_id == post_id))
    if subscription:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail="Already subscribed")

    new_subscription = profile_post.insert().values(id=uuid4(),
profile_id=current_user.id, post_id=post_id)
    await db_conn.execute(new_subscription)

    return {"message": "Subscribed successfully"}

async def unsubscribe_from_post_service(post_id: UUID, current_user:
User):
    """
    Unsubscribe from a post.

    Args:
    post_id: UUID - The post id.
    current_user: User - The user data.
```

```
    Returns:
    Dict: The response message.

    Raises:
    HTTPException: If the user is not subscribed.
    """

    subscription = await
db_conn.fetch_one(profile_post.select().where(profile_post.c.profile_id ==
current_user.id, profile_post.c.post_id == post_id))
    if not subscription:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail="Not subscribed")

    delete_subscription =
profile_post.delete().where(profile_post.c.profile_id == current_user.id,
profile_post.c.post_id == post_id)
    await db_conn.execute(delete_subscription)

    return {"message": "Unsubscribed successfully"}

async def check_subscription_status_service(post_id: UUID, current_user:
User):
    """
    Check the subscription status of a post.

    Args:
    post_id: UUID - The post id.
    current_user: User - The user data.

    Returns:
    Dict: The status of the subscription.
    """

    subscription = await
db_conn.fetch_one(profile_post.select().where(profile_post.c.profile_id ==
current_user.id, profile_post.c.post_id == post_id))
    status = bool(subscription)
    return {"isSubscribed": status}

async def get_subscribers_service(post_id: UUID, current_user: User):
    """
    Get the subscribers of a post.

    Args:
    post_id: UUID - The post id.
    current_user: User - The user data.

    Returns:
    List[Dict]: The subscribers data.
    """

    query = select(profile.c.username).select_from(
        profile.join(profile_post, profile.c.id == profile_post.c.profile_id)
    ).where(profile_post.c.post_id == post_id)

    subscribers = await db_conn.fetch_all(query)
    return subscribers

async def get_events_service(current_user: User):
    """
    Get the events of a user.
```

```
    Args:
    current_user: User - The user data.

    Returns:
    List[Dict]: The events data.
    """

    query = select(
        post.c.id,
        profile.c.username,
        post.c.content,
        post.c.start_time,
        post.c.end_time
    ).select_from(
        post.join(profile_post, post.c.id == profile_post.c.post_id)
            .join(profile, profile_post.c.profile_id == profile.c.id)
    ).where(profile_post.c.profile_id == current_user.id)

    events = await db_conn.fetch_all(query)
    return [
        {
            "id": event.id,
            "username": event.username,
            "content": event.content,
            "start_time": event.start_time,
            "end_time": event.end_time
        } for event in events
    ]
```

Файл calendar_infrastructure.py

Реалізація функціональних вимог: Система планування зустрічей, Видалення зустрічі, Створити нову зустріч.

```
from infrastructure.db import db_meta
from sqlalchemy import (UUID, Column, DateTime, ForeignKey, Table, func)


profile_post = Table(
    "profile_post",
    db_meta,
    Column("id", UUID, primary_key=True),
    Column("profile_id", UUID, ForeignKey("profile.id", ondelete="CASCADE"),
nullable=False),
    Column("post_id", UUID, ForeignKey("post.id", ondelete="CASCADE"),
nullable=False),
    Column("created_date", DateTime(timezone=True),
server_default=func.now()),
)
```

Файл community_controller.py

Реалізація функціональних вимог: Система спільнот, Створення дискусії, Створення коментарю, Видалення дискусії, Підписка до спільноти, Вихід із спільноти.

```
from uuid import UUID
from fastapi import APIRouter, Depends, HTTPException, Request, status
from pydantic import BaseModel
from common.models import User
from common.token import retrieve_user_from_token
```

```
from services.community import (
    create_community_service,
    get_community_service,
    add_to_community,
    remove_membership,
)
from profile.exceptions import RelationsError

community_router = APIRouter()

class CommunityCreateRequest(BaseModel):
    name: str
    creatorProfileId: str

@community_router.post("")
async def create_community(request: CommunityCreateRequest):
    """
    Create a new community profile.

    Args:
    request: Dict - The community data.

    Returns:
    Dict: The community data.

    Raises:
    HTTPException: If the user is not authorized.
    """

    return await create_community_service(request)

@community_router.get("/{id}")
async def get_community(id: str):
    """
    Get community data by id.

    Args:
    id: str - The community id.

    Returns:
    Dict: The community data.

    Raises:
    HTTPException: If the user is not authorized.
    """

    return await get_community_service(id)

@community_router.post("/profiles/{profile_id}/add/{community_id}",
                       status_code=status.HTTP_201_CREATED)
async def add_to_community(request: Request, profile_id: UUID, community_id:
UUID, user: User = Depends(retrieve_user_from_token)):
    """
    Add a profile to a community.

    Args:
    profile_id: UUID - The profile id.
    community_id: UUID - The community id.
    user: User - The user data.

    Returns:
    Dict: The community data.

    Raises:
```

```
    HTTPException: If the user is not authorized.
    """

    if user.id != profile_id:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Unauthorized.")

    try:
        await add_to_community(profile_id, community_id)
    except RelationsError:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail="Cannot add to community.")

@community_router.delete(
    "/profiles/{profile_id}/remove/{community_id}",
    status_code=status.HTTP_204_NO_CONTENT)
async def remove_from_community(
        profile_id: UUID,
        community_id: UUID,
        user: User = Depends(retrieve_user_from_token)):
    """
    Remove a profile from a community.

    Args:

    profile_id: UUID - The profile id.
    community_id: UUID - The community id.
    user: User - The user data.

    Returns:
    Dict: The community data.

    Raises:
    HTTPException: If the user is not authorized.
    """

    if user.id != profile_id:
        raise HTTPException(status_code=status.HTTP_403_FORBIDDEN)
    return await remove_membership(user.id, community_id)
```

Файл RegisterPage.jsx

Реалізація функціональної вимоги: Реєстрація користувача.

```
import React, { useState } from 'react';
import { TextField, Button, Container, Typography, Grid, Box, Alert, Link }
from '@mui/material';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

const RegisterPage = () => {
  const [username, setUsername] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [confirmPassword, setConfirmPassword] = useState('');
  const [error, setError] = useState(null);
  const [success, setSuccess] = useState(null);
  const navigate = useNavigate();

  const handleRegister = async (e) => {
    e.preventDefault();
    if (password !== confirmPassword) {
      setError('Passwords do not match');
```

```
      return;
    }
    try {
      const response = await
axios.post(`${import.meta.env.VITE_API_URL}/api/v1/auth/register`, {
        username,
        email,
        password,
      }, {
        headers: {
          'Content-Type': 'application/json'
        }
      });
      setSuccess('Registration successful!');
      setError(null);
      console.log('Register:', response.data);
      // Navigate to login page after successful registration
      navigate('/login');
    } catch (error) {
      if (error.response) {
        setError(error.response.data.detail);
      } else {
        setError('Registration failed');
      }
      setSuccess(null);
    }
  };

  return (
    <Container maxWidth="sm">
      <Box sx={{ mt: 5 }}>
        <Typography variant="h4" component="h1" gutterBottom>
          Register
        </Typography>
        {error && <Alert severity="error">{error}</Alert>}
        {success && <Alert severity="success">{success}</Alert>}
        <form onSubmit={handleRegister}>
          <Grid container spacing={2}>
            <Grid item xs={12}>
              <TextField
                fullWidth
                label="Username"
                value={username}
                onChange={(e) => setUsername(e.target.value)}
                required
              />
            </Grid>
            <Grid item xs={12}>
              <TextField
                fullWidth
                label="Email"
                type="email"
                value={email}
                onChange={(e) => setEmail(e.target.value)}
                required
              />
            </Grid>
            <Grid item xs={12}>
              <TextField
                fullWidth
                label="Password"
                type="password"
                value={password}
                onChange={(e) => setPassword(e.target.value)}
```

```
                required
              />
            </Grid>
            <Grid item xs={12}>
              <TextField
                fullWidth
                label="Confirm Password"
                type="password"
                value={confirmPassword}
                onChange={(e) => setConfirmPassword(e.target.value)}
                required
              />
            </Grid>
            <Grid item xs={12}>
              <Button type="submit" variant="contained" color="primary"
fullWidth>
                Register
              </Button>
            </Grid>
          </Grid>
        </form>
        <Box sx={{ mt: 2 }}>
          <Typography>
            Already have an account? <Link href="/login">Login</Link>
          </Typography>
        </Box>
      </Box>
    </Container>
  );
};

export default RegisterPage;
```

Файл community_service.py

Реалізація функціональної вимоги: Підписка до спільноти, Вихід із спільноти, Створення дискусії.

```
from uuid import UUID, uuid4
from sqlalchemy import insert, select
from infrastructure.profile import profile, Profile, ProfileRole,
create_profile_graph_transaction
from infrastructure.db import db_conn, db_graph
from services.profile import join_community, remove_from_community

async def create_community_service(request):
    """
    Create a new community profile.

    Args:
    request: Dict - The community data.

    Returns:
    Dict: The community data.
    """
    uid = uuid4()
    community_profile = Profile(
        id=uid,
        username=request.name,
        admin=request.creatorProfileId,
        role=ProfileRole.game,
        email=str(uid),
```

```
        hashed_password=str(uid),
    )

    created_community = await db_conn.fetch_one(insert(profile).values(
                community_profile.model_dump(exclude_none=True)).returning(pr
ofile))

    async with db_graph.session() as session:
        result_id = created_community.id
        result_username = created_community.username
        await session.write_transaction(create_profile_graph_transaction,
                                        result_id, result_username)

    await join_community(request.creatorProfileId, created_community.id)

    return {
        "id": created_community.id,
        "username": created_community.username,
        "role": created_community.role,
        "admin": created_community.admin,
    }

async def get_community_data(id: str):
    """
    Get community data by id.

    Args:
    id: str - The community id.

    Returns:
    Dict: The community data.
    """

    result = await db_conn.fetch_one(select(profile)
                                     .where(profile.c.id == id))
    admin = await db_conn.fetch_one(select(profile)
                                    .where(profile.c.id == result.admin))


    return {
        "id": result.id,
        "username": result.username,
        "role": result.role,
        "admin_id": result.admin,
        "admin_username": admin.username,
    }

async def remove_membership(profile_id: UUID, community_id: UUID):
    """
    Remove a profile from a community.

    Args:
    profile_id: UUID - The profile id.
    community_id: UUID - The community id.

    Returns:
    bool: True if successful, False otherwise.
    """

    try:
        await remove_from_community(profile_id, community_id)
        return True
    except Exception as e:
        return False
```

Файл relationship.py

Реалізація функціональної вимоги: Додавання в друзі, Прийняти запрошення.

```
async def make_friends(sender_id: UUID, receiver_id: UUID, type:
FriendshipStatus):
    """
    Make two profiles friends.

    Args:
    sender_id: UUID - The sender profile id.
    receiver_id: UUID - The receiver profile id.
    type: RelationshipStatus - The relationship status.
    """

    match type:
        case FriendshipStatus.OUTGOING:
            query = initiate_friendship
        case FriendshipStatus.INCOMING:
            return await establish_friendship(sender_id, receiver_id)
        case _:
            raise NotImplementedError
    async with db_graph.session() as session:
        await session.write_transaction(query, sender_id, receiver_id)

async def initiate_friendship(tx, sender_id: UUID, friend_id: UUID):
    """
    Initiate a friendship between two profiles.

    Args:
    tx: Transaction - The transaction.
    sender_id: UUID - The sender profile id.
    friend_id: UUID - The friend profile id.
    """
    query = INITIATE_FRIENDSHIP_QUERY.format(sender_id=sender_id,
friend_id=friend_id)
    await tx.run(query)

async def establish_friendship(
        initiator_id: UUID,
        sender_id: UUID) -> None:
    """
    Establish a friendship between two profiles.

    Args:
    initiator_id: UUID - The initiator profile id.
    sender_id: UUID - The sender profile id.
    """

    async with db_graph.session() as session:
        query = ESTABLISH_FRIENDSHIP_QUERY.format(sender_id=sender_id,
initiator_id=initiator_id)
        await session.write_transaction(lambda tx: tx.run(query))
```

Файл notification.py

Реалізація функціональної вимоги: Підписка на спільноту, Додавання в друзі.

```
from fastapi import APIRouter, Depends
from sqlalchemy import desc, select

from common.token import retrieve_user_from_token
from common.models import User
from services.profile import service as profile_service
from infrastructure.db import db_conn
from infrastructure.post import post
from infrastructure.profile import profile


notification_router = APIRouter()

@notification_router.get("/")
async def get_notifications(user: User = Depends(retrieve_user_from_token)):
    """
    Get notifications for the user

    Args:
    user: User - The user data.

    Returns:
    List[Dict]: The notifications data.
    """

    friends = await
profile_service.get_friends_by_profile_id(user.profile_id)
    friends_ids = [friend.id for friend in friends]

    query = (
        select(post, profile.c.username)
        .select_from(post.join(profile, post.c.parent_id == profile.c.id))
        .where(post.c.parent_id.in_(friends_ids))
        .order_by(desc(post.c.created_date))
        .limit(20)
    )
    posts = await db_conn.fetch_all(query)

    notifications = [
        {
            "message": f"Check new post on {post.username} page",
            "content" : post.content,
            "datetime": post.created_date.isoformat(),
        }
        for post in posts
    ]

    return notifications
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

ВЕБ-ЗАСТОСУНОК ДЛЯ ІГРОВОЇ СОЦІАЛЬНОЇ ПЛАТФОРМИ

Програма та методика тестування

КПІ.ІТ-9402.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Антон ДАНИЛЕВИЧ

Київ – 2024

ЗМІСТ

1	ОБ’ЄКТ ВИПРОБУВАНЬ	3
2	МЕТА ТЕСТУВАННЯ	4
3	МЕТОДИ ТЕСТУВАННЯ	5
4	ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ	6

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є веб-застосунок, що виконує основні функції соціальної платформи. Веб-застосунок розділений на 2 частини клієнтську, що написана мовою програмування JavaScript з використанням фреймворку React, а серверна частина написана мовою Python з використанням фрейворку FastAPI.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

— перевірка правильності роботи веб-застосунку відповідно до функціональних вимог;

— перевірка збереження даних;

— валідація вхідних даних;

— перевірка сумісності веб-додатку з останніми версіями сучасних браузерів (Chrome, Opera та Firefox);

— знаходження проблем, помилок і недоліків з метою їх усунення;

— перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

— мануальне тестування — тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з використанням наскрізного тестування, з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

— динамічне тестування на відповідність функціональним вимогам;

— тестування на виведення повідомлень про помилку, коли це необхідно;

— тестування інтерфейсу користувача;

— тестування зручності використання;

— тестування створення, видалення та перегляд контенту з бази даних.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

ВЕБ-ЗАСТОСУНОК ДЛЯ ІГРОВОЇ СОЦІАЛЬНОЇ ПЛАТФОРМИ

Керівництво користувача

КПІ.ІТ-9402.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Антон ДАНИЛЕВИЧ

Київ – 2024

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ 3
2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ 4
2.1 Системні вимоги для коректної роботи 4
2.2 Завантаження застосунку 4
2.3 Перевірка коректної роботи 4
3 ВИКОНАННЯ ПРОГРАМИ 5

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Time Breaker — це вебзастосунок для ігрової соціальної платформи, призначений для забезпечення зручного та ефективного способу взаємодії геймерів, планування ігрових подій та обговорення ігрових стратегій.

Деякі з основних функцій та користувацьких можливостей, які реалізовані вебзастосунком Time Breaker, включають:

— пошук користувачів та ігрових спільнот: користувачі можуть легко знаходити інших гравців, об'єднуватись у спільноти за інтересами, а також приєднуватися до вже існуючих груп, зосереджених на певних іграх або жанрах;

— планування ігрових подій: вебзастосунок дозволяє користувачам створювати та планувати майбутні ігрові сесії, організовувати віртуальні зустрічі та координувати дії для командної гри;

— спілкування та обговорення: Time Breaker надає можливості для текстового спілкування між гравцями. Це дозволяє користувачам обговорювати стратегії, ділитися досвідом та координувати свої дії під час гри;

— безпека та надійність: вебзастосунок забезпечує захист особистих даних користувачів та підтримку надійності системи, що включає відновлення після можливих збоїв та захист від несанкціонованого доступу.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для роботи даного застосунку необхідне виконання наступних мінімальних вимог:

— підключення до інтернету;

— сучасний веббраузер;

— операційна система: Windows, MacOS або Linux

Для успішної роботи даного застосунку необхідне виконання наступних рекомендованих вимог:

— процесор з тактовою частотою 1 ГГц;

— оперативна пам’ять 8 Гб;

— операційна система: Windows, MacOS або Linux

— відеокарта з 1 Гб відеопам’яті;

— підключення до інтернету.

2.2 Завантаження застосунку

Програмне забезпечення є вебзастосунок, а отже воно розгорнуте та є доступне цілодобово

2.3 Перевірка коректної роботи

У веббраузері перейти на особисту сторінку користувача та переглянути наявні дані про пости, події та їх коментарі. Також можна перевірити коректність роботи за допомогою створення будь-якого посту, події, коментаря до посту або підписки/відписки до події.

3 ВИКОНАННЯ ПРОГРАМИ

При запуску вебзастосунка користувачу буде відображено два поля для вводу ім’я користувача та паролю для входу в особистий акаунт користувача (рисунок 3.1).

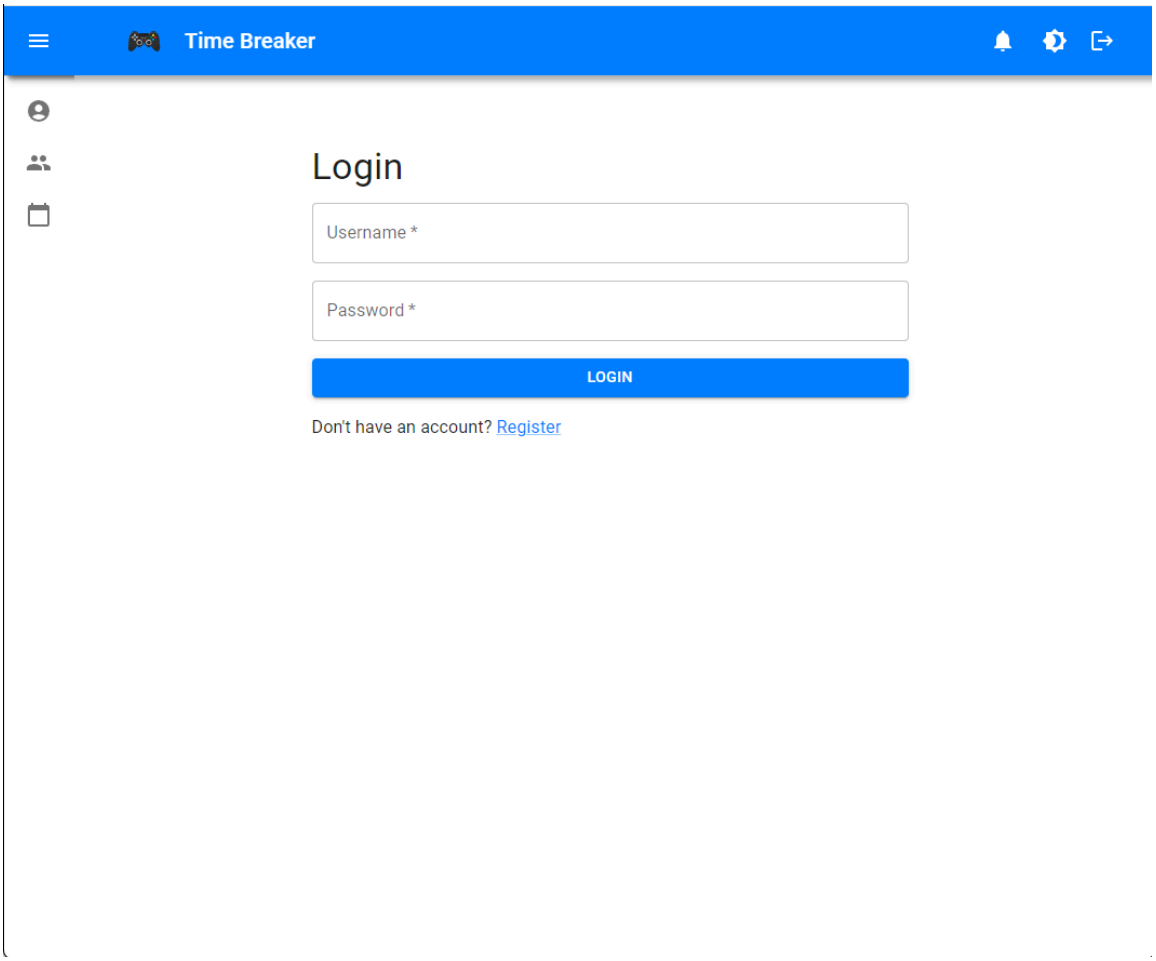


Рисунок 3.1 — Початкова сторінка додатку

Якщо користувач немає облікового запису він має змогу зареєструвати його на сторінці реєстрації (рисунок 3.2).

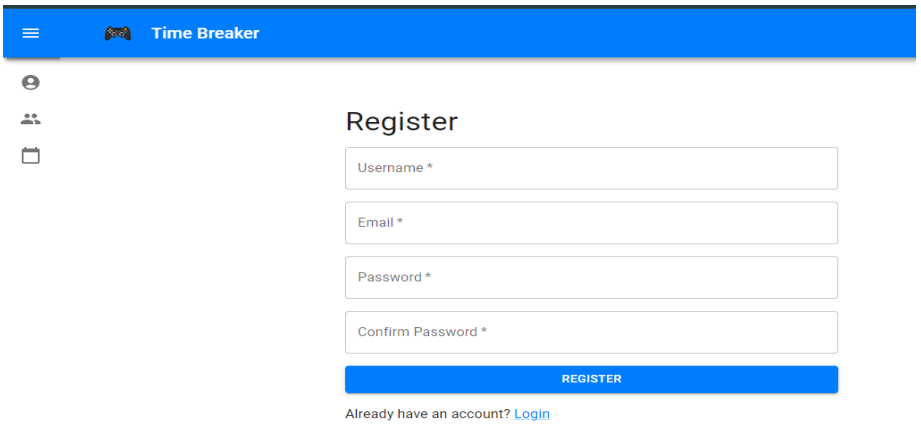


Рисунок 3.2 — Сторінка реєстрації

Користувач на сторінці пошуку має можливість знайти спільноту та користувача, який його цікавить, а також переглянути список друзів, спільнот, запитів на дружбу (рисунок 3.3)

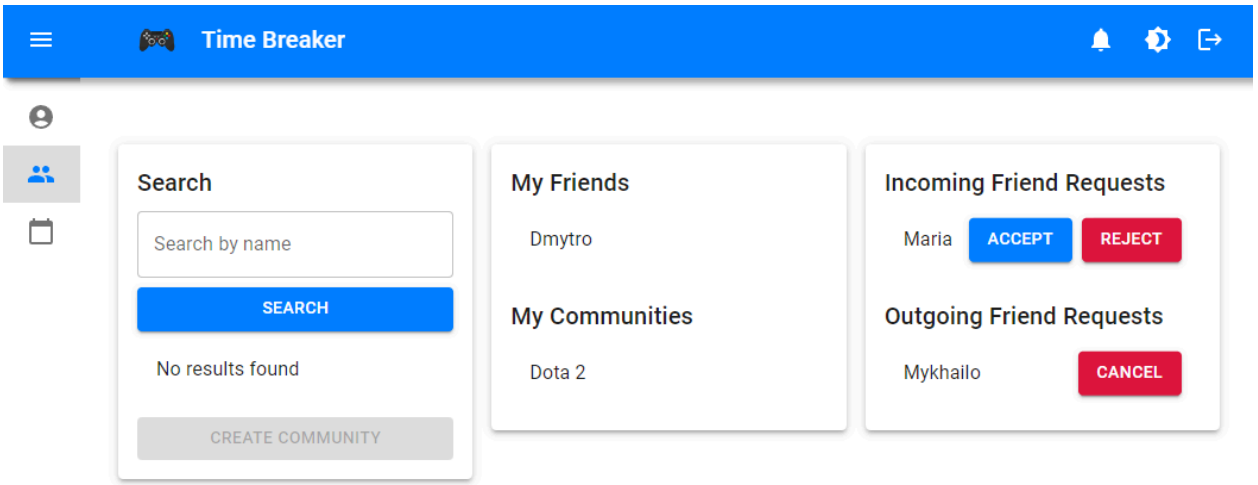


Рисунок 3.3 — Сторінка пошуку

Перейшовши на сторінку спільноти користувач має можливість перегляну події та пости пов’язані зі спільнотою, залишити коментар або створити нову подію (рисунок 3.4).

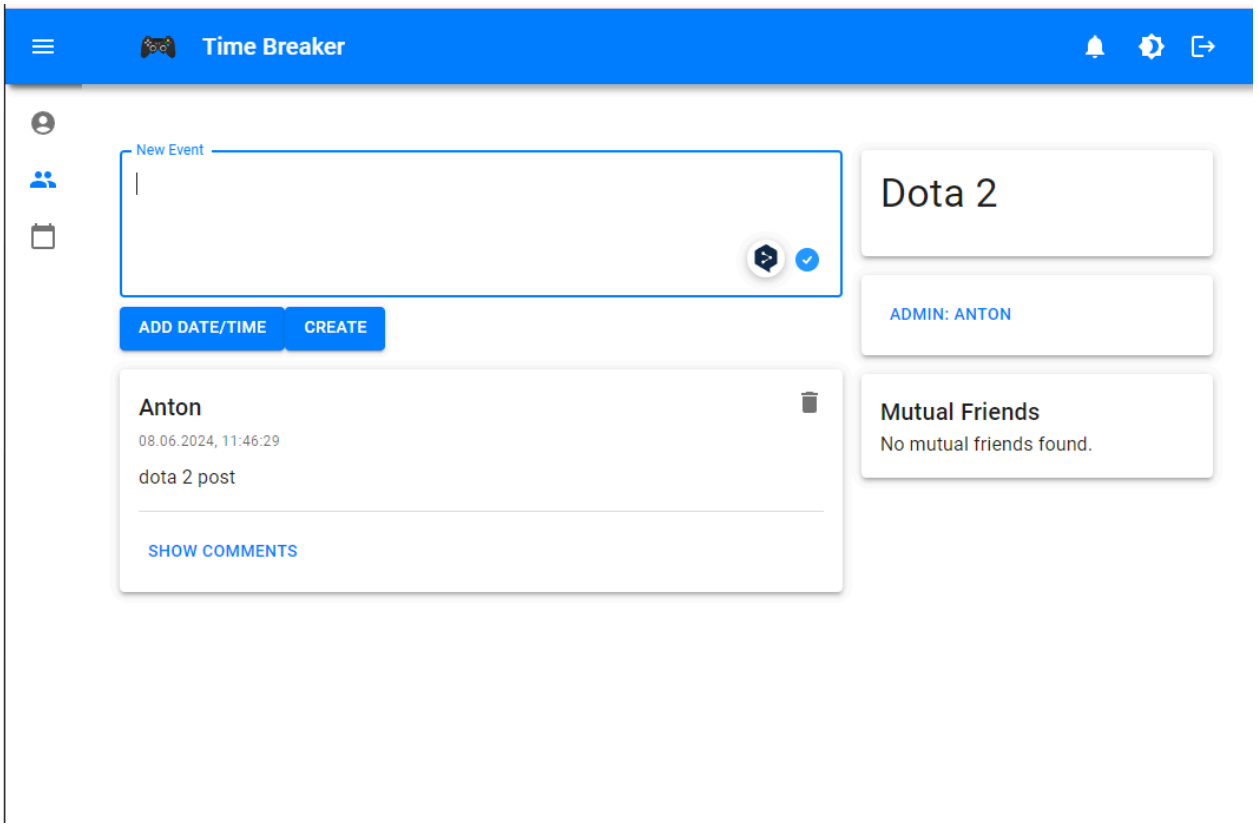


Рисунок 3.4 — Сторінка спільноти

Переглянути свої заплановані події користувач може на сторінці календаря майбутніх подій (рисунок 3.5)

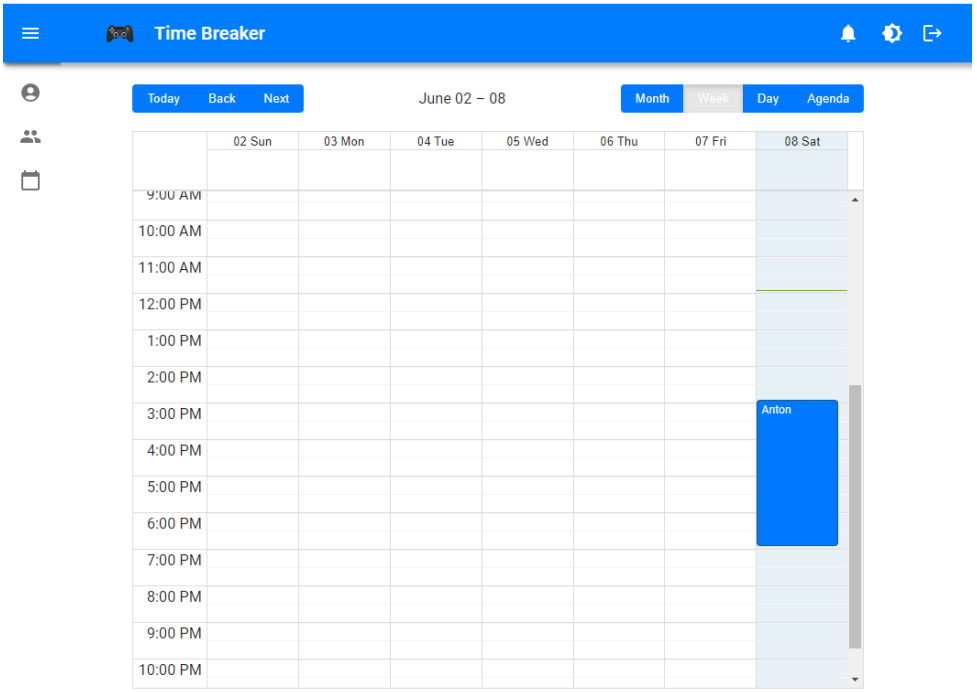


Рисунок 3.5 — Календар майбутніх подій

Користувач також має змогу переглянути список нотифікацій у спеціальному меню (рисунок 3.6)

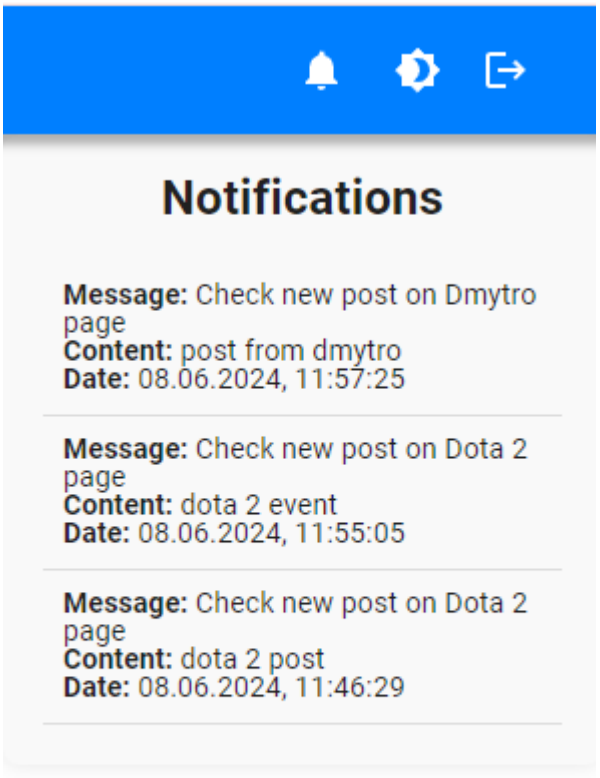


Рисунок 3.6 — Сторінка перегляду нотифікацій

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

ВЕБ-ЗАСТОСУНОК ДЛЯ ІГРОВОЇ СОЦІАЛЬНОЇ ПЛАТФОРМИ

Графічний матеріал

КПІ.ІТ-9402.045440.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

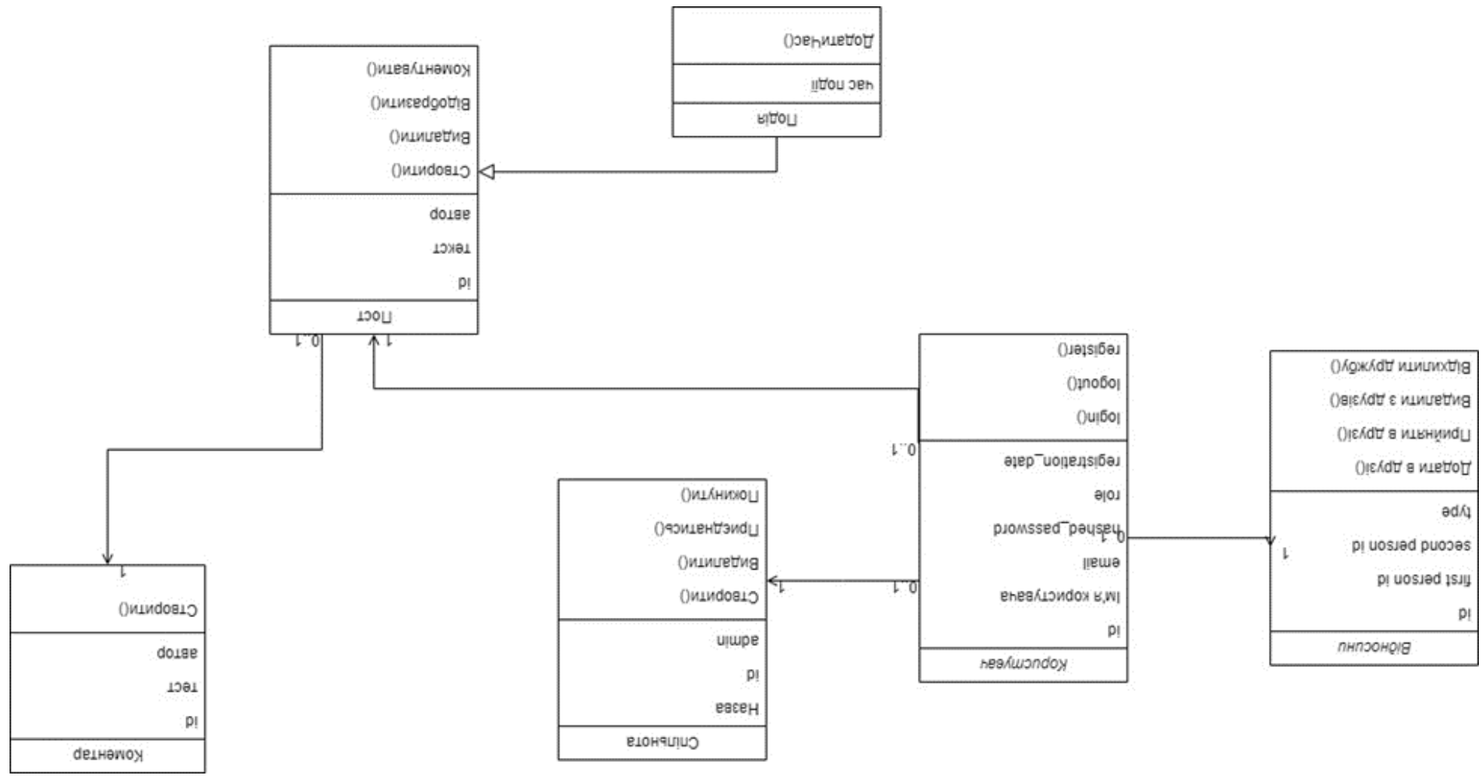
Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Антон ДАНИЛЕВИЧ

Київ – 2024

КПІ.ІТ-9402.045440.06.99.ССВ

					КПІ.ІТ-9402.045440.06.99.ССВ		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використань	Літера	Маса	Масштаб
Розробив		Данилевич А.П.						
Перевірив		Крамар Ю. М.				Аркуш	Аркушів	
Т. контр.					Веб-застосунок для ігрової соціальної платформи	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-02		
Н. контр.		Ліщук К.І.						
Затвердив		Жаріков Е.В.						



КПІ.ІТ-9402.045440.06.99.ССК

Зм.	Арк.	№ документа	Підпис	Дата	КПІ.ІТ-9402.045440.06.99.ССК					
Розробив		Данилевич А.Л.								
Перевірив		Крамар Ю. М.			Схема структура класів			Літера	Маса	Масштаб
Т. контр.										
								Аркуш	Аркушів	
Н. контр.		Лищук К.І.			Веб-застосунок для ігрової соціальної платформи			КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-02		
Затвердив		Жарікова Е.В.								

Зм.	Арк.	№ докум.	Підп.	Дата
Розробив		Данилевич А.П.		
Перевірив		Крамар Ю. М.		
Т. контр.				
Н. контр.		Лищук К.І.		
Затвердив		Жаріков Е.В.		

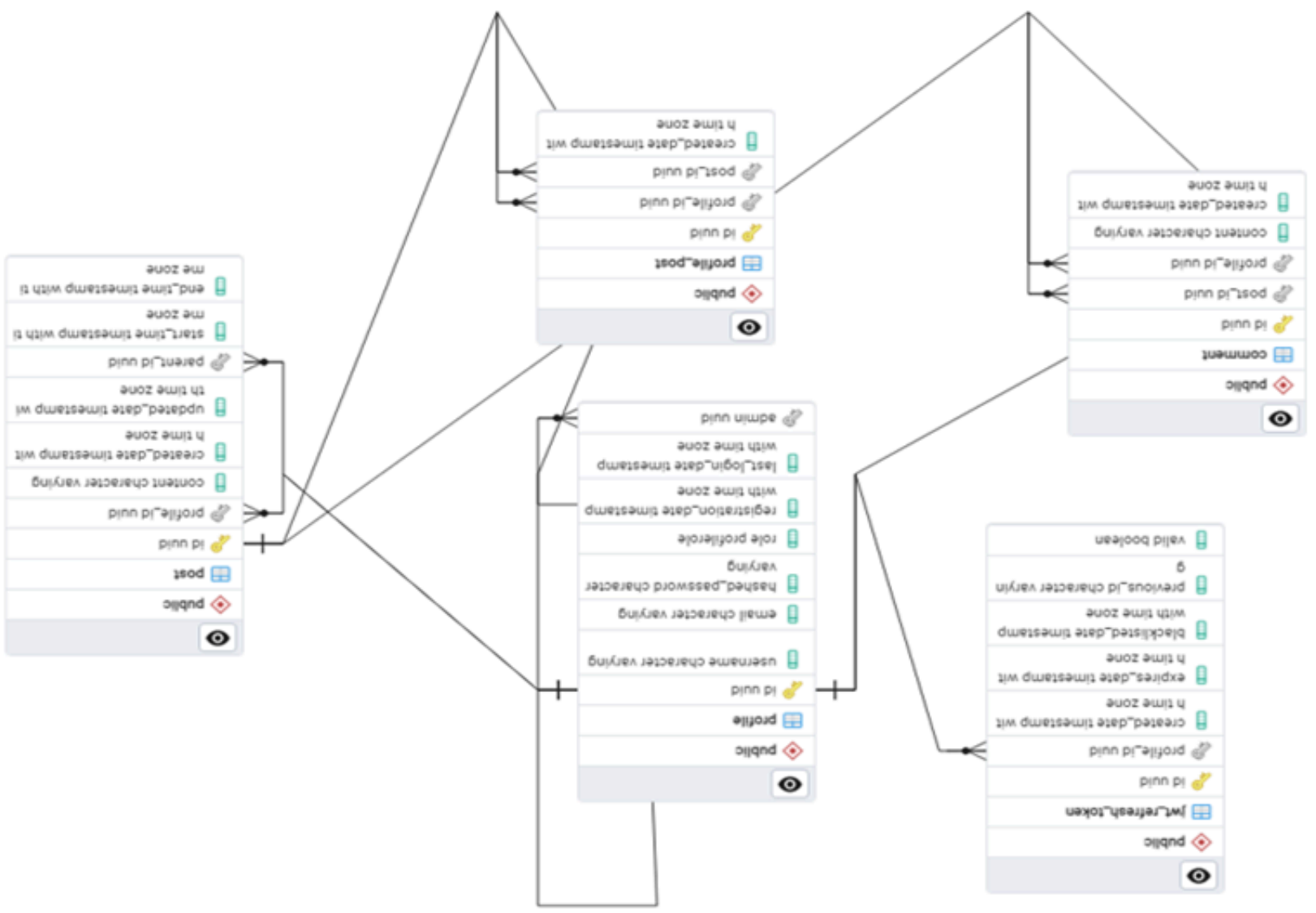
КПІ.ІТ-9402.045440.06.99.СБД

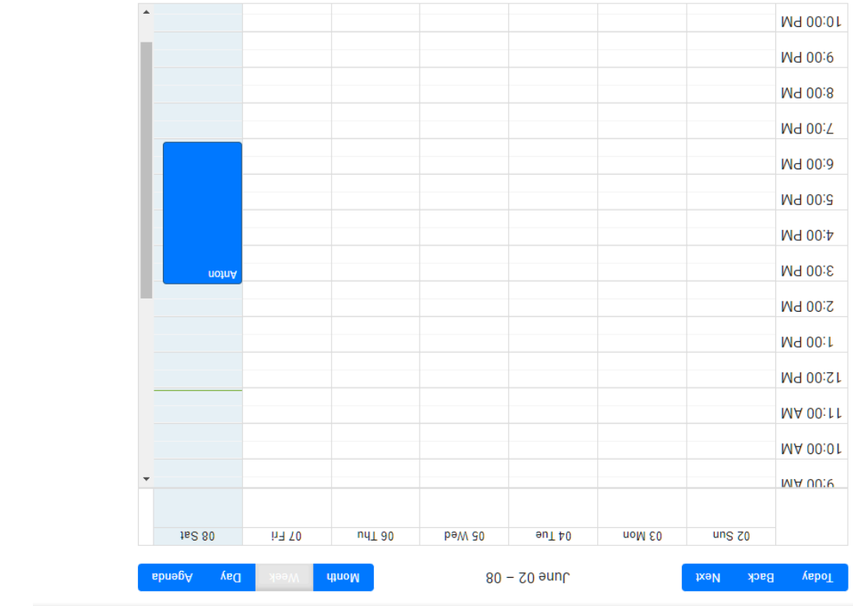
Лит.	Маса	Масштаб
Аркуш	Аркушів	

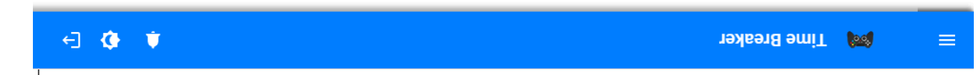
Схема бази даних

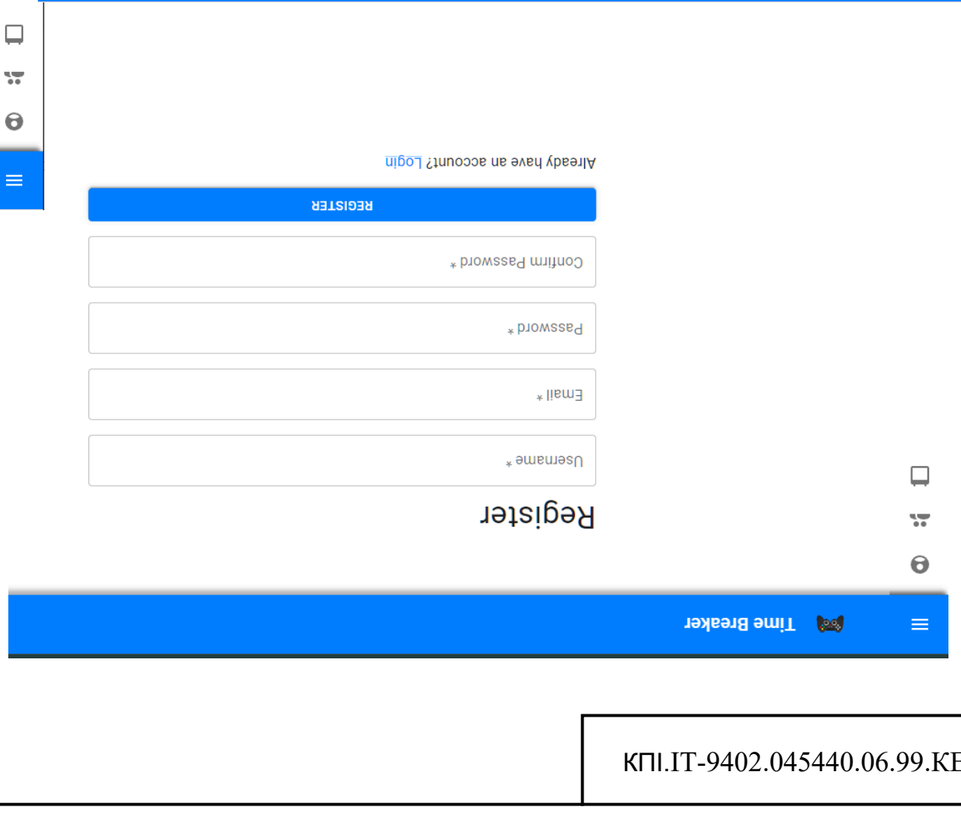
Веб-застосунок для ігрової соціальної платформи

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІТ-02





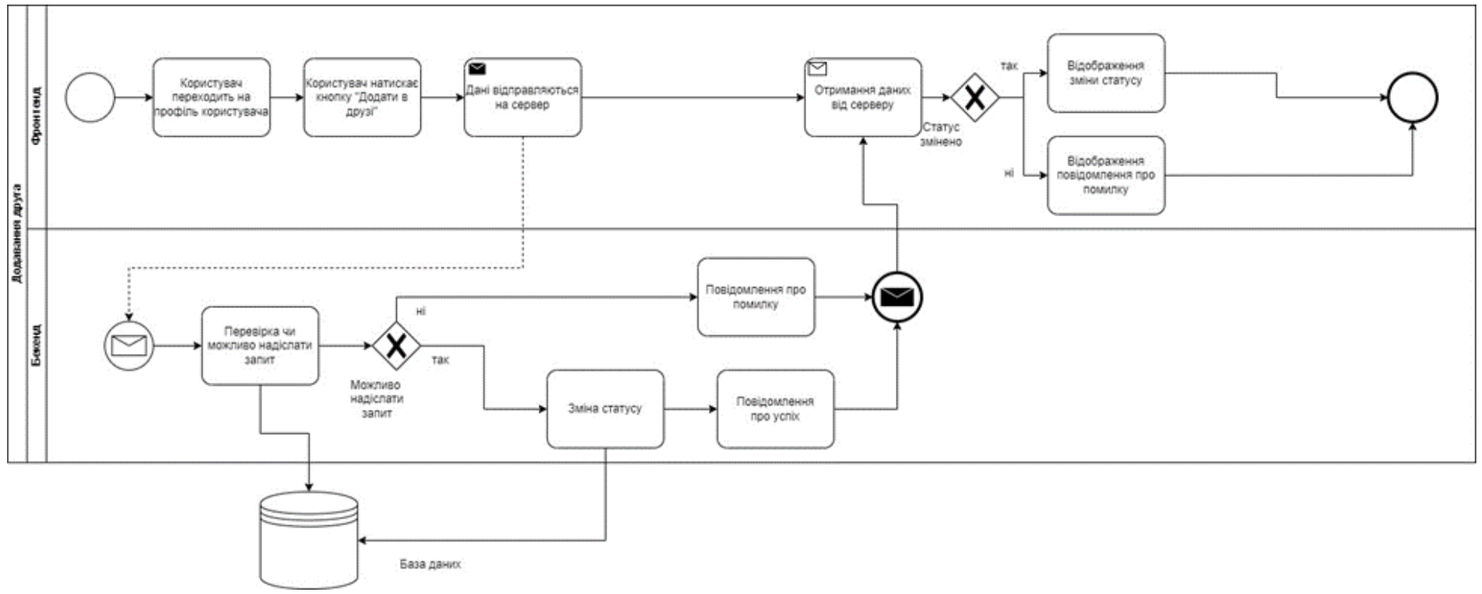




КПІ.IT-9402.045440.06.99.КЕ

Зм.	Арк.	№ документа	Підпис	Дата	КПІ.IT-9402.045440.06.99.КЕ				
Розробив		Данилевич А.Л.			Кресленна вигляду екранних форм			Літера	Маса	Масштаб
Перевірив		Крамар Ю. М.								
Т. контр.								Аркуш	Аркушів	
Н. контр.		Лищук К.І.			Веб-застосунок для ігрової соціальної платформи			КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-02	
Затвердив		Жарікова Е.В.							

Схема бізнес процесу "Додавання в друзі"



Демонстраційний плакат до дипломного проекту

Веб-застосунок для ігрової соціальної платформи

Виконав студент гр. ІТ-02 Данилевич А. П.

Керівник Крамар Ю. М.