Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ВЕБПЛАТФОРМИ ДЛЯ ОРГАНІЗАЦІЇ ІГРОВИХ СЕСІЙ З ВИКОРИСТАННЯМ DJANGO, REACT ТА MYSQL

DEVELOPMENT OF A WEB PLATFORM FOR ORGANIZING GAMING SESSIONS USING DJANGO REACT AND MYSQL

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-42
Борбич Олег Валентинович

Керівник:
Суринович Олена Миколаївна

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«*10*» *12* 20*24* р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Борбичу Олегу Валентиновичу
(прізвищ. ім'я. по батькові)

1. Тема кваліфікаційної роботи *«Розробка вебплатформи для організації ігрових сесій з використанням Django, React та MySQL»*
Керівник роботи: *к.т.н., доцент Суринович О. М.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *Django REST Framework, Django Channels, React, Tailwind, MySQL, Visual Studio Code, методичні вказівки до виконання кваліфікаційної роботи бакалавра*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *Аналіз предметної сфери та наявних реалізацій, окреслення вимог до вебзастосунку для керування ігровими командами, обрання сучасних технологій, розробка серверної частини, бази даних, адаптивного інтерфейсу з авторизацією через Steam, тестування та вибір оптимальної платформи для розміщення.*
5. Перелік графічного матеріалу: *7 рисунків, 1 додаток*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Суринович О. М.*		
Специфікація вимог до розробленої системи	*Суринович О. М*		
Розробка об'єкта проектування	*Суринович О. М*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	1,77 %		
Академічна доброчесність	*Суринович О. М*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	вик.
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	вик.
3	Обгрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	вик.
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	вик.
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	вик.
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	вик.
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	вик.

Здобувач вищої освіти _____ Олег БОРБИЧ

Керівник кваліфікаційної роботи _____ Олена СУРИНОВИЧ

АНОТАЦІЯ

Борбич О. В. Розробка вебплатформи для організації ігрових сесій з використанням Django, React та MySQL. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі здійснено дослідження актуального стану сервісів з організацією геймерських команд, проведено огляд існуючих підходів і сформульовано завдання. В другому розділі окреслено вимоги до системи, розроблено її архітектуру, базу даних та обґрунтовано вибір технологій, таких як Django REST Framework, React та Django Channels для реалізації командних чатів. У третьому розділі детально описано процес розробки: реалізацію механізму авторизації через Steam, функціонал роботи з командами та запрошеннями, розробку системи чатів, інтеграцію фронтенду з бекендом та процедуру розгортання проекту. У висновках підсумовано результати виконаної роботи, досягнуті цілі та перспективи подальшого розвитку даної платформи.

Ключові слова: вебзастосунок, команди, Steam авторизація, Django, React, REST API, Django Channels, VS Code, онлайн-комунікація, командний чат.

ABSTRACT

Borbych O. Development of a Web Platform for Organizing Gaming Sessions Using Django, React and MySQL. Manuscript. Qualification work of the bachelor's program "Software engineering" in the specialty "Software engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The first chapter presents the current state of services for organizing gaming teams, reviews existing approaches and formulates the objectives. The second chapter outlines the requirements for the system, develops its architecture, database, and justifies the choice of technologies such as Django REST Framework, React, and Django Channels for the implementation of team chats. The third chapter describes the development process in detail: the implementation of the Steam authorization mechanism, the functionality of working with teams and invitations, the development of the chat system, the integration of the frontend with the backend, and the project deployment procedure. The conclusions summarize the results of the work performed, the goals achieved, and the prospects for further development of this platform.

Keywords: web application, commands, Steam authorization, Django, React, REST API, Django Channels, VS Code, online communication, team chat.

ЗМІСТ

ВСТУП

Актуальність теми. Розробка вебплатформи для створення команд та управління ігровими сесіями є актуальною необхідністю. Це пояснюється стрімким розвитком кіберспорту та дедалі більшою популярністю онлайн-ігор, які об’єднують мільйони геймерів з усього світу. Зростання числа як аматорських, так і професійних колективів, а також необхідність у злагодженій взаємодії між учасниками, спонукають до потреби у сучасних засобах, що полегшують створення команд, планування ігрових сесій та спілкування.

Таким чином, метою даної кваліфікаційної роботи є розробка вебплатформи, яка забезпечить зручне створення команд, організацію ігрових сесій та ефективну комунікацію між гравцями, сприяючи покращенню координації та ігрового досвіду в онлайн-іграх.

Об’єктом кваліфікаційної роботи бакалавра є процеси формування команд та управління ігровими сесіями в онлайн-іграх, а також технічні способи їх автоматизації шляхом створення власної вебплатформи. Це передбачає аналіз механізмів збору груп гравців, розробку графіку ігрових подій, координування взаємодії між учасниками та забезпечення ефективної співпраці в умовах швидкого розвитку кіберспорту та багатокористувацьких ігрових просторів. Дослідження охоплює як організаційні питання, так і технічні рішення, спрямовані на полегшення цих процесів за допомогою цифрових інструментів.

Предметом роботи є методи та інструменти розробки вебплатформи, що забезпечують створення команд, організацію ігрових сесій та підтримку комунікації між гравцями в онлайн-іграх.

Для досягнення поставленої мети були поставлені наступні завдання:

– розробити структуру бази даних для зберігання інформації про користувачів, їхні Steam-акаунти, команди, запрошення та взаємозв’язки між ними, забезпечуючи нормалізацію та оптимальний вибір типів полів для ефективного зберігання даних;

– реалізувати серверну логіку проєкту, включаючи створення моделей, представлень, маршрутизацію, механізми авторизації через Steam OpenID, обробку запрошень до команд, їх прийняття або відхилення, а також управління складом команд;

– створити зручний і адаптивний інтерфейс користувача з використанням JavaScript, React та CSS-фреймворку Tailwind, забезпечуючи інтеграцію з бекендом через REST API;

– реалізувати механізм реєстрації та авторизації користувачів виключно через Steam-акаунт, використовуючи бібліотеки для роботи з OpenID та збереження профільної інформації гравців;

– проаналізувати хостинг-платформи для розгортання клієнт-серверного додатку, оцінити їх за критеріями продуктивності, масштабованості, вартості та зручності, та обрати оптимальне рішення для розміщення бекенду та фронтенду.

Апробація результатів дослідження (додаток А): Борбич О. В. Суринович О. М. Авторизація через сторонні сервіси у соціальних мережах. Тези доповідей X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025) (23-24 травня 2025 року). Луцьк: ЛНТУ, 2025. С. 437-440.

РОЗДІЛ 1
АНАЛІЗ РОЗРОБКИ ВЕБПЛАТФОРМИ ДЛЯ ОРГАНІЗАЦІЇ ІГРОВИХ СЕСІЙ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Протягом останніх десятиліть цифрові новації докорінно змінили те, як люди взаємодіють, навчаються, працюють та проводять дозвілля. Найбільш суттєвими ці зміни виявились у галузі відеоігор, яка з нішевого хобі поступово перетворилася на масову культурну та економічну індустрію. З розвитком інтернету, швидкого доступу до мережі та ростом популярності мультиплеєрних режимів, відеоігри перестали бути суто індивідуальним заняттям – натомість стали соціальним феноменом. Ігри вже не обмежуються лише графікою та геймплеєм: вирішальну роль відіграє комунікація між гравцями, можливість об’єднуватись у команди, планувати спільні дії, брати участь у змаганнях та вибудовувати цілі ігрові спільноти.

На тлі цих змін виникла об’єктивна необхідність у спеціалізованих онлайн-платформах, які б забезпечували повний спектр функціональних можливостей для гравців: від створення профілю та зберігання статистики до організації командної взаємодії та участі у змаганнях різних рівнів. У кіберспортивній індустрії, що продовжує стрімко розвиватися, такі платформи стали незамінним інструментом як для звичайних користувачів, що грають задля задоволення, так і для професійних команд, які прагнуть здобути визнання, перемагати в турнірах та будувати кар’єру у сфері e-sports.

В цьому контексті сервіси для організації командних ігор, підбору суперників, управління ігровими профілями та участі у кіберспортивних подіях набули шаленої популярності. Вони дозволяють не тільки координувати дії між гравцями, а й створюють екосистему, де кожен учасник має можливість розвиватися, покращувати свої навички, аналізувати статистику та будувати власну репутацію. Одним із ключових елементів таких платформ є система

формування команд, адже саме командна взаємодія є основою багатьох популярних ігор – від шутерів до стратегій.

Зважаючи на це, не дивно, що наразі існує ціла низка відомих платформ, які надають гравцям інструменти для об’єднання у команди, спілкування, участі у турнірах, планування матчів, а також керування ігровими профілями. Ці сервіси стали важливою складовою цифрового середовища для мільйонів користувачів у всьому світі. Вони задовольняють як базові потреби гравців – наприклад, пошук напарників – так і більш складні задачі, пов’язані з організацією змагань, рейтингами, трансляціями та аналітикою.

Отже, у сучасному цифровому світі платформи для організації командних ігор, підбору суперників та участі у кіберспортивних турнірах стали не просто бажаним, а фактично необхідним інструментом для всіх категорій гравців – від новачків до професіоналів. Їхній функціонал включає широкий спектр можливостей, серед яких – створення та управління командами, внутрішня комунікація, система запрошень, перегляд статистики та участь у змагальних подіях.

Одними з найвідоміших представників цієї ніші є платформи Faceit та Challengermode. Їхні функції, технічні характеристики, переваги та недоліки є важливими об’єктами аналізу при розробці власного програмного забезпечення.

Faceit – це одна з найбільш авторитетних платформ у сфері онлайн-підбору суперників та організації кіберспортивних змагань, особливо в іграх на кшталт Counter-Strike: Global Offensive. Платформа дозволяє гравцям створювати профілі, приєднуватися до команд, брати участь у турнірах і здобувати рейтинги за свої досягнення [4]. На рисунку 1.1 зображена сторінка пошуку команди для гри. Однією з ключових особливостей Faceit є інтеграція зі Steam, що забезпечує швидкий доступ до платформи та дозволяє аутентифікувати користувача безпосередньо через його ігровий акаунт. Система підбору суперників є досить точною та адаптованою під рівень гравця, а також є система античиту для боротьби з нечесною грою.

Рисунок 1.1 – Сторінка пошуку команди для гри на Faceit [4]

Проте, незважаючи на переваги, Faceit має і недоліки. По-перше, інтерфейс користувача є доволі перевантаженим та може видатися складним для новачків. По-друге, функціонал командної взаємодії є доволі обмеженим, якщо порівнювати з більш гнучкими платформами. Наприклад, можливості організації внутрішніх командних чатів, запрошень до команд або перегляду історії спільної гри реалізовані не надто інтуїтивно. Окрім того, частина функцій, таких як участь у преміум-турнірах, доступна лише за підпискою, що створює бар’єр для нових користувачів.

Іншою великою платформою, яка активно розвивається у сфері кіберспорту, є Challengermode. Вона також надає можливість створення команд, участі у турнірах, автоматичного підбору суперників та підтримки різних ігор. Візуально Challengermode виглядає більш сучасно та має спрощений доступ до основного функціоналу [2]. Інтеграція з обліковими записами Steam, Discord та Twitch робить платформу зручною для стрімерів та геймерів, котрі активно взаємодіють з аудиторією. Великим плюсом є можливість створення власних турнірів та налаштування їх під конкретні правила. На рисунку 1.2 зображена сторінка команди на сайті Challengermode.

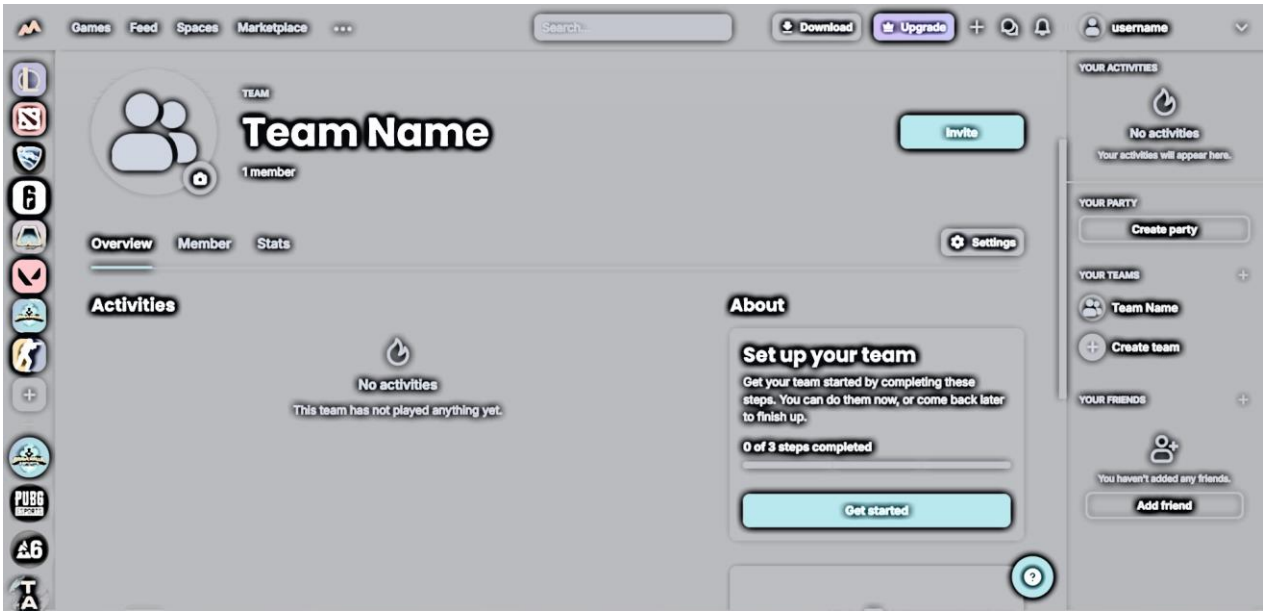

Рисунок 1.2 – Сторінка команди на Challengermode [2]

Проте Challengermode також має свої обмеження. Наприклад, платформа досі не підтримує деякі популярні ігри, через що її функціональність може бути недостатньою для окремих користувачів. Окрім того, незважаючи на зручний інтерфейс, часом виникають складнощі з навігацією між вкладками, особливо на мобільних пристроях. Система сповіщень реалізована не надто ефективно: запрошення до команд або турнірів не завжди вчасно надходять користувачеві. Також, взаємодія між гравцями поза межами турнірів є досить обмеженою, і платформа зосереджена переважно на турнірній складовій.

Зважаючи на чималу кількість онлайн-майданчиків, що сприяють проведенню кіберспортивних змагань, слід зауважити, що переважна більшість з них сфокусована на професійних або напівпрофесійних турнірах, де на першому місці стоїть результат, рейтинг та структура турніру. Ці сервіси приділяють основну увагу організації офіційних матчів, регламентам, призовим фондам, трансляціям та статистичній аналітиці. Вони ідеально підходять для команд, які прагнуть професійного рівня, брати участь у кваліфікаціях, лігах та відкритих турнірах. Прикладами таких платформ є Faceit, Challengermode, Toornament та інші аналогічні сервіси з потужними механізмами турнірного менеджменту.

Разом з тим, в екосистемі онлайн-ігор стає очевидним суттєвий недолік платформ, які були б спрямовані не тільки на змагальний аспект, але й на щоденне мультиплеєрне дозвілля гравців. Адже більшість користувачів, особливо тих, хто не планує ставати кіберспортсменами, бажають просто комфортно провести час у своїх улюблених іграх – таких як CS:GO, Dota 2, Valorant, PUBG, Apex Legends тощо – у компанії однодумців. Їм не завжди потрібні турніри з чітким розкладом та жорсткими умовами, натомість вони цінують можливість легко знайти команду, створити лобі для гри, домовитись про час зустрічі, розподілити ролі й спілкуватися до та після гри.

Цей сегмент залишається відносно мало освоєним. Існує небагато сервісів, які пропонують зручний інтерфейс і продуману систему саме для звичайних, неофіційних матчів. Більшість платформ або надто заточені під кіберспорт, або надають надто загальні інструменти, які не враховують специфіку командної взаємодії у повсякденному геймінгу. Бракує сервісів, де було б зручно створити власну команду без зобов’язань участі в турнірах, запросити друзів через Steam, домовитися про гру на вечір та підтримувати соціальні зв’язки у невеликій ігровій спільноті.

Відтак, попри широкий вибір платформ для організації турнірів, залишається суттєва потреба у сервісах, орієнтованих на звичайного гравця – того, хто хоче грати не заради кубків чи призів, а заради задоволення, командної взаємодії та спілкування. Саме заповнення цієї ніші відкриває перспективи для розвитку нових платформ, які зможуть поєднати зручність, простоту та соціальну складову, не ставлячи користувача в жорсткі рамки змагального середовища.

У цьому контексті розробка власної платформи для організації геймерських команд та взаємодії між користувачами виглядає обґрунтованою. Головною відмінністю платформи, що розробляється, є простота та гнучкість у роботі з командами: користувачі можуть створювати команди, надсилати запрошення іншим гравцям, приймати або відхиляти запрошення без додаткової складності або підписок. Завдяки реалізації бекенду на базі Django та Django

REST Framework, можливе ефективне створення API, що дозволяє фронтенд-частині на React швидко та безпечно взаємодіяти із сервером. Авторизація через Steam забезпечує швидкий вхід на платформу та підтвердження справжності користувача без введення логіну та пароля, що знижує ризики безпеки та спрощує вхід для користувачів, які вже мають акаунт у Steam.

У порівнянні з наявними аналогами, нова платформа пропонує краще опрацьовану систему запрошень до команд, простіший та більш інтуїтивний інтерфейс, створений за допомогою Tailwind CSS, а також більшу гнучкість у розширенні функціоналу. Система побудована таким чином, що навіть у початковій версії вона відкрита для масштабування – у майбутньому можна буде додати можливості чату, підтримку різних ігор, систему рейтингу гравців, персоналізовану аналітику ігор, а також внутрішні міні-турніри.

В той час, як платформи Faceit та Challengermode мають усталені моделі роботи, вони не завжди враховують потреби малих геймерських спільнот, які прагнуть більшої гнучкості в управлінні командами та плануванні матчів без участі у великих турнірах. Натомість розроблюване програмне забезпечення пропонує користувачу зручний інтерфейс і зосереджується на ключових аспектах – створенні команд, системі запрошень, управлінні профілем, авторизації через Steam та можливості швидкого долучення до командної взаємодії.

Окрім того, важливим аспектом нового рішення є його відкритість до подальшої інтеграції з іншими сервісами та можливість кастомізації. Це надає користувачам більше контролю над функціональністю системи, дозволяє врахувати особливості локальних спільнот та адаптуватися під конкретні потреби.

Загалом, запропонований підхід до створення нової платформи базується на досвіді та аналізі конкурентів, а також на сучасних технологіях та принципах UI/UX дизайну, що дозволяє створити зручне, ефективне та адаптивне середовище для геймерів, які хочуть організовувати власні команди та спільно брати участь у грі.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

У процесі виконання кваліфікаційної роботи потрібно досягти низки ключових цілей, зосереджених на створенні вебплатформи для організації командних багатокористувацьких ігор із застосуванням сучасних технологій веброзробки. Ці задачі включають в себе:

– вивчення сучасних інструментів, фреймворків та бібліотек, що допоможуть у реалізації проєкту, з обґрунтуванням вибору мови Python, фреймворку Django як основи серверної частини та Django REST Framework для побудови API;

– розробку структури бази даних для зберігання даних про користувачів, їхні Steam-акаунти, команди, запрошення та взаємозв’язки між ними, із забезпеченням нормалізації та вибором відповідних типів полів для ефективного зберігання інформації;

– реалізацію серверної логіки проєкту, зокрема створення моделей, представлень, маршрутизації та механізмів авторизації через Steam OpenID, обробку запрошень до команд, прийняття та відхилення запрошень, а також управління складом команд;

– створення зручного та зрозумілого інтерфейсу користувача за допомогою JavaScript, React та CSS-фреймворку Tailwind, забезпечення адаптивності інтерфейсу для різних пристроїв та реалізацію основного функціоналу взаємодії з бекендом через REST API;

– реалізацію механізму реєстрації та авторизації користувачів виключно через обліковий запис Steam, використовуючи відповідні бібліотеки для взаємодії з OpenID та збереження інформації про профіль гравця;

– аналіз наявних хостинг-платформ для розгортання повноцінного клієнт-серверного додатку, оцінювання їх за критеріями продуктивності, масштабованості, вартості та зручності використання, вибір оптимального рішення для розміщення бекенду та фронтенду.

Висновки до розділу 1

Отже, створення нової платформи стає своєчасним рішенням, що відповідає потребам пересічних геймерів, які відчувають потребу у простих, комфортних і адаптивних сервісах для організації команд та соціальної взаємодії. Наявні платформи, як-от Faceit і Challengermode, зосереджуються в основному на професіоналах, тоді як запропонований підхід забезпечує доступність функціоналу навіть для новачків, спрощуючи реєстрацію, запрошення в команди та користування платформою загалом.

Запропоноване рішення інтегрує передові технології, масштабовану архітектуру та соціальну орієнтованість, що відкриває перспективи для формування активних ігрових спільнот. Ця платформа здатна не тільки заповнити нішу щоденного геймінгу, а й стати ефективним інструментом для підтримки цифрових соціальних контактів між гравцями.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

У цьому розділі описується процес моделювання, архітектурного проектування та формалізації вимог до інформаційної системи, яка зараз у розробці. Головна її мета – створення онлайн-платформи для збору геймерських команд, з можливістю авторизації через Steam, забезпечення внутрішньокомандної комунікації, планування ігрових сесій та ефективної взаємодії між учасниками. Система має відповідати актуальним стандартам зручності, безпеки, масштабованості та інтуїтивно зрозумілого інтерфейсу.

Об’єктно-орієнтований підхід до моделювання системи був обраний через складність взаємозв’язків між основними об’єктами (користувач, команда, повідомлення, запрошення тощо). Для візуалізації структури та взаємодії елементів використано UML, зокрема діаграму класів. Це дало змогу формалізовано показати компоненти програмного забезпечення, їх властивості та взаємодії. Розглянемо моделі розробки на UML-діаграмі:

– User – базова модель користувача в Django. Цей клас є ключовим для авторизації і з ним взаємодіють всі інші об’єкти. Має зв’язки з командами, повідомленнями, запрошеннями та базується на SteamUser;

– SteamUser – розширення User, реалізоване зв’язком один до одного. Містить інформацію з профілю Steam: ім’я, URL профілю, аватар та жанрові теги;

– GenreTag – відповідає за жанрові теги. Має зв’язок багато до багатьох з класом SteamUser;

– Team – команда, створена користувачем. Має такі поля: назва, опис, власник, список учасників і максимальна кількість учасників. Користувач може бути як власником, так і учасником кількох команд;

– TeamInvite – запрошення до команди. Зберігає посилання на команду, відправника, отримувача та дату створення. Кожне запрошення унікальне для певного поєднання команди та отримувача;

– ChatMessage – повідомлення в чаті команди. Має зв’язок з конкретною командою та користувачем-відправником. Включає текст повідомлення та дату його відправлення.

Діаграму класів наведено на рисунку 2.1.

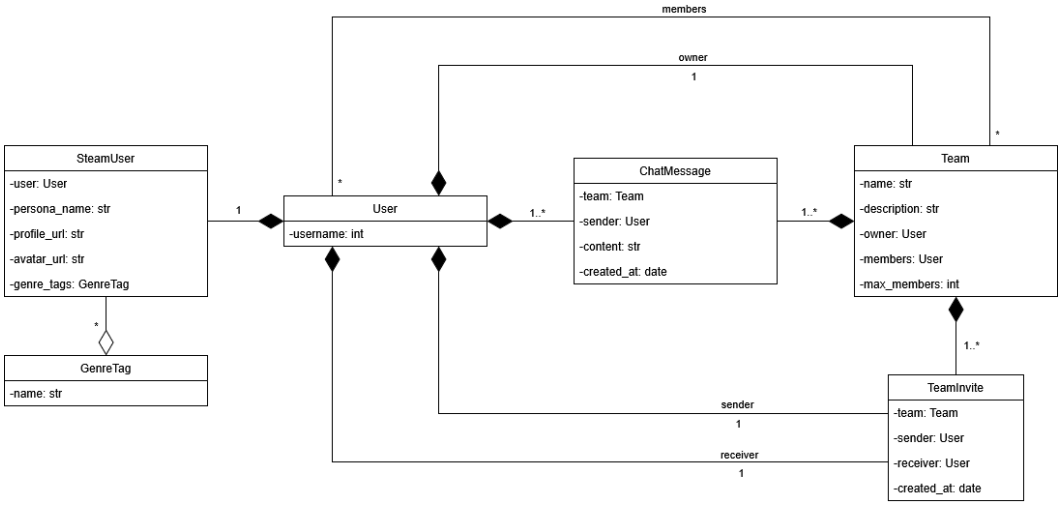

Рисунок 2.1 – Діаграма класів яка зображає структуру моделей Django

Результатом моделювання є цілісна схема, що дає змогу втілити логіку платформи та передбачити її масштабування. Прийняті рішення обґрунтовані як з погляду структурованості даних, так і з погляду зручності роботи користувачів із системою.

Виявлення функціональних і нефункціональних вимог здійснювалось за допомогою методів аналізу предметної області та сценарного підходу. Зібрані дані використано для формування детальної специфікації вимог до системи. До основних функціональних вимог належать: реєстрація та аутентифікація через Steam, створення команд, управління учасниками, надсилання та обробка запрошень, а також ведення командного чату. Нефункціональні вимоги містять безпеку даних, адаптивний інтерфейс, швидкодію та масштабованість системи.

Система передбачає існування різних типів користувачів, а саме звичайний користувач та власник команди. Звичайний користувач може переглядати профілі, надсилати заявки або приймати запрошення до команд, використовувати чати. Власник команди, окрім перелічених вище функцій, має можливість створювати команди, редагувати їх описи, керувати учасниками та надсилати запрошення іншим користувачам.

Архітектура проєкту передбачає використання Django на бекенді з Django REST Framework для розробки API. Авторизація виконується за допомогою OpenID-протоколу, який надає Steam. Користувач авторизується на стороні Steam, після чого сайт отримує унікальний SteamID – основний ідентифікатор користувача в системі. У відповідь видається JWT-токен, що зберігається на клієнтській стороні та використовується для подальшої аутентифікації у запитах API. Такий підхід гарантує як безпеку, так і зручність входу.

Серед ключових функціональних можливостей платформи – збереження профілю користувача, прив’язаного до облікового запису Steam, разом з нікнеймом, аватаром та тегами улюблених жанрів. Користувачі мають можливість створювати команди або приєднуватись до існуючих, використовуючи запрошення. Власник команди керує учасниками та може спілкуватись у вбудованому чаті. Для зручності реалізовано зберігання історії повідомлень із зазначенням автора та часу.

Для забезпечення безпеки система передбачає автентифікацію через Steam без паролів, використання JWT для захищеного доступу до API, серверну валідацію всіх змінних запитів, обмеження доступу на основі ролей користувачів та контроль дій власника команди щодо зміни інформації.

UX/UI-дизайн платформи розробляється з акцентом на простоту, логічність та адаптивність. Інтерфейс користувача реалізовано за допомогою React та TailwindCSS, що забезпечує швидку розробку та гнучке налаштування стилів. Інтерфейс адаптується під різні пристрої – мобільні телефони, планшети та десктопи. Для підтвердження дій використовуються модальні вікна, передбачено візуальні підказки та анімації для покращення взаємодії.

Після успішної авторизації користувач переходить на головну сторінку, де йому доступні перегляд власного профілю, перелік доступних команд, запрошень, а також можливість створення або пошуку команд. Особливу увагу приділено інтерактивності елементів: сповіщення про нові запрошення, відображення статусу користувача в команді, динамічне оновлення списку чатів.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Розробка вебдодатку, який забезпечуватиме автентифікацію через Steam, створення команд, систему запрошень, персональні профілі та чат у реальному часі, вимагатиме ретельного підходу до вибору технологій, методів і алгоритмів. Цей процес буде ключовим для створення системи, що відповідатиме вимогам масштабованості, безпеки, швидкості обробки даних і зручності для користувачів. На початковому етапі буде проведено детальний аналіз вимог, що дозволить визначити основні задачі: створення інтерактивного інтерфейсу, забезпечення швидкої обробки запитів, надійної автентифікації та можливості подальшого розширення функціоналу. Для досягнення цих цілей буде обрано архітектуру клієнт-сервер, яка чітко розділятиме бекенд і фронтенд. Такий підхід спростить розробку, тестування, підтримку системи та відкриє перспективи для інтеграції додаткових клієнтів, наприклад, мобільних додатків, у майбутньому.

Для розробки цього проєкту було використано інтегроване середовище розробки Visual Studio Code (VS Code). Воно стало ключовим інструментом через свою універсальність та можливість розширення функціональності. Вибір VS Code обґрунтовано легким стартом, відносно високою швидкістю роботи навіть на не дуже потужних комп’ютерах, а також великою кількістю доступних розширень, що дозволяють гнучко налаштовувати його під конкретні завдання проекту. Зручна інтеграція з системами контролю версій, зокрема Git, та

вбудований термінал для запуску команд забезпечили ефективну роботу з комплексними компонентами, полегшуючи розробку як для одного розробника, так і для командної роботи.

VS Code, розроблений Microsoft, є легким редактором коду, який завдяки розширюваності ідеально підходить для розробки на React та Django. Він швидко запускається та не перевантажує систему, що особливо важливо при роботі з великими проектами. Для Django VS Code надає зручний інтегрований термінал, де можна виконувати команди безпосередньо з редактора. Розширення для Python додають підтримку автодоповнення, налагодження та лінтингу, наприклад, через pylint_django, а для React розширення, такі як ES7 React, Redux, GraphQL, React-Native snippets і ESLint, забезпечують підсвітку синтаксису JSX, автодоповнення та форматування коду. Налаштування VS Code легко здійснюється за допомогою JSON-файлів, що дозволяє адаптувати його до специфічних потреб проєкту, як-от, додавши Djaneiro для сніпетів Django або React Native Tools для React. VS Code безкоштовний, працює на всіх популярних платформах (Windows, macOS, Linux) і має вбудовану інтеграцію з Git, спрощуючи командну роботу.

PyCharm від JetBrains – повнофункціональне IDE, що спеціалізується на Python та Django. У професійній версії (платній) реалізовано глибоку інтеграцію з Django, зокрема автодоповнення для шаблонів, підтримку «manage.py», налагодження та прямий доступ до баз даних. Для React PyCharm підтримує JavaScript і JSX, але потребує додаткових плагінів та поступається спеціалізованим IDE для фронтенду за зручністю використання. PyCharm має більші ресурсовитрати, ніж VS Code, що може впливати на швидкість запуску та роботи на слабких машинах. Але його функція «перейти до визначення» та вбудована підтримка віртуальних середовищ (зокрема в WSL або віддалених системах) перевершують можливості VS Code. Безкоштовна Community Edition має обмежену підтримку Django, що може бути суттєвим обмеженням.

WebStorm, також від JetBrains, є IDE, сфокусованим на JavaScript та фронтенді, що робить його ідеальним для React. Він пропонує потужне автодоповнення, рефакторинг та налагодження для JSX, TypeScript та CSS, а також інтегровану підтримку npm та Vite, які часто використовуються в React проектах. Для Django WebStorm менш зручний, оскільки не має спеціалізованих інструментів для Python і потребує додаткового налаштування, такого як PyCharm або інший інструмент для роботи з бекендом. WebStorm платний, проте він швидший за PyCharm та має кращу продуктивність для фронтенду, ніж VS Code, хоча програє останньому в легкості та універсальності.

IntelliJ IDEA (Ultimate Edition, платна) від JetBrains поєднує можливості PyCharm та WebStorm, що робить її універсальним вибором для повного стеку React та Django. Вона підтримує Python, Django, JavaScript та React в одному середовищі, дозволяючи працювати з усіма частинами проєкту, не перемикаючись між різними IDE. IntelliJ IDEA надає розширені функції налагодження, рефакторингу та інтеграції з Git, проте через свою складність вона важча та повільніша за VS Code і потребує більше часу для освоєння. Безкоштовна Community Edition не підтримує веброзробку та Django, що обмежує її застосування.

Для роботи з React і Django стеком, VS Code виграє завдяки легкості, універсальності та безкоштовності. Він ефективно працює з обома частинами стека через розширення, що дозволяє швидко перемикатися між бекендом та фронтендом. PyCharm найкращий для складних Django-проєктів, але менш ефективний для React, а WebStorm ідеальний для React, але вимагає додаткового інструменту для Django. IntelliJ IDEA підходить для великих команд, яким потрібне все в одному, але її вага і ціна можуть бути надмірними для невеликих проєктів. Зважаючи на баланс між продуктивністю та функціональністю, VS Code – оптимальний вибір, особливо враховуючи активну спільноту та регулярні оновлення.

Для серверної частини проєкту буде використано мову програмування Python. Її вибір обґрунтовано простотою синтаксису, високою читабельністю коду та розвиненою екосистемою бібліотек, які значно прискорять розробку.

Python вирізняється елементарним синтаксисом, високим ступенем абстракції та наявністю потужних фреймворків, зокрема Django і Flask. Django гарантує оперативну розробку повнофункціональних вебдодатків завдяки вбудованому ORM, системі маршрутизації, автентифікації та іншим сервісам. Flask, зі свого боку, забезпечує більшу гнучкість та підходить для мікросервісної архітектури. Python також володіє великою кількістю бібліотек для взаємодії з API, безпекою, машинним навчанням, обробкою даних, що дає змогу розробляти вебдодатки з розширеним функціоналом [9].

У порівнянні з JavaScript, Python традиційно застосовується для бекенду. JavaScript залишається єдиною мовою, яка працює нативно у браузері, і в поєднанні з фреймворками, як-от React, Vue або Angular, він формує основу сучасного фронтенду. Для повного стеку веброзробки передбачений варіант використання лише JavaScript (наприклад, через Node.js), що може забезпечити кращу узгодженість технологій на клієнтській і серверній стороні. Проте Python пропонує вищу читабельність коду та швидший старт для початківців.

Java історично використовується для розробки масштабованих корпоративних вебдодатків. Завдяки фреймворкам Spring та Jakarta EE Java забезпечує високу продуктивність, безпеку і сувору типізацію. Водночас Java потребує більшого обсягу шаблонного коду та довшого циклу розробки, порівнюючи з Python.

PHP – одна з найдавніших мов веброзробки, яка досі актуальна завдяки системам, як-от WordPress та фреймворку Laravel. Незважаючи на спрощену модель розробки, PHP часто критикують за неузгодженість синтаксису та архітектурну хаотичність у старих проєктах. Python, натомість, пропонує більш сучасний підхід і чіткіший поділ логіки.

Ruby разом із фреймворком Ruby on Rails користувався популярністю завдяки принципу «конвенція понад конфігурацію». Але останніми роками поступився позиціями Python через меншу гнучкість, продуктивність і меншу спільноту.

Отже, Python – це відмінний вибір для створення вебдодатків, особливо коли важлива швидкість розробки, зрозумілість коду та інтеграція з системами обробки даних чи машинного навчання. Його обмеження у фронтенді компенсуються ідеальним поєднанням із JavaScript, тоді як для великих, типізованих систем доцільніше використовувати Java або C# [5].

Python ефективно справлятиметься з реалізацією складної бізнес-логіки, обробкою HTTP-запитів, взаємодією з базами даних і інтеграцією з зовнішніми сервісами, зокрема Steam Web API, який буде необхідним для автентифікації користувачів і отримання їхніх даних. Широка популярність Python забезпечить доступ до великої кількості документації та підтримки спільноти, що полегшить вирішення технічних питань.

Основним фреймворком для бекенду стане Django, високорівневий інструмент, який оптимізуватиме процес розробки завдяки своїм вбудованим можливостям. Django пропонуватиме ORM (Object-Relational Mapping), що дозволить працювати з базою даних через Python-об’єкти, уникаючи написання складних SQL-запитів. Це забезпечить зручність при роботі з моделями, такими як користувачі, команди, запрошення чи повідомлення чату. Фреймворк також включатиме механізми захисту від поширених хакерських атак, таких як CSRF, XSS і SQL-ін’єкції, що буде критично важливим для безпеки. Django спростить маршрутизацію запитів, управління автентифікацією та обробку даних, дозволяючи зосередитися на реалізації унікальних функцій додатку.

Для створення REST API, через який фронтенд взаємодіятиме з бекендом, буде використано Django REST Framework. Цей інструмент інтегруватиметься з Django та забезпечуватиме зручну серіалізацію даних, обробку відповідей і

налаштування прав доступу. DRF дозволить створювати API-ендпоінти, які повертатимуть дані у JSON-форматі, наприклад, для списків команд, запрошень чи історії чату. Він також підтримуватиме інтеграцію з djangorestframework-simplejwt, що використовуватиметься для видачі JWT-токенів після автентифікації через Steam. Такий підхід гарантуватиме безпечну авторизацію користувачів і захист даних під час обміну між клієнтом і сервером [3].

Чат у реальному часі буде реалізовано за допомогою Django Channels, який розширюватиме можливості Django для роботи з WebSocket. Проєкт Django Channels розширює можливості Django, надаючи інструменти для асинхронного спілкування. До них належать WebSockets, HTTP2 push та фонові завдання. Завдяки цьому можна створювати вебзастосунки, що працюють у реальному часі, забезпечуючи двосторонній обмін даними між клієнтом і сервером [14]. Django Channels підтримуватиме концепцію груп, що дасть змогу надсилати повідомлення всім користувачам, підключеним до певної команди. Для тестування використовуватиметься InMemoryChannelLayer, але для продакшену планується перехід на RedisChannelLayer, який забезпечуватиме кращу продуктивність і масштабування при великій кількості одночасних з’єднань.

Як базу даних буде обрано MySQL, яка вирізняється швидкістю, надійністю та широкою підтримкою в екосистемі Django. MySQL ефективно оброблятиме запити до моделей із складними зв’язками, такими як OneToOneField між User і SteamUser чи ManyToManyField для учасників команд. Її вибір буде обґрунтовано простотою налаштування, високою продуктивністю при роботі з великими обсягами даних і сумісністю з Django ORM. MySQL також підтримуватиме транзакції, що забезпечить цілісність даних під час операцій, таких як створення запрошень чи збереження повідомлень чату.

Фронтенд буде розроблено з використанням JavaScript, який є основною мовою для створення динамічних вебінтерфейсів. Для побудови інтерфейсу буде обрано React, бібліотеку, що дозволить створювати модульні компоненти. React

використовуватиметься для реалізації сторінок профілів, списків команд, модальних вікон для запрошень і чату.

React у порівнянні з іншими фронтенд-фреймворками: Vue.js, Angular, Svelte, Solid.js. React – це JavaScript бібліотека, розроблена Facebook для створення інтерфейсів, орієнтована на компонентний підхід та декларативне програмування. Вона має гнучкість та дозволяє інтегруватися з різноманітними інструментами. Екосистема React багата на бібліотеки, серед яких Redux, React Router, Next.js, а також може похвалитися активною спільнотою. JSX, що поєднує HTML і JavaScript, полегшує створення компонентів, хоча й потребує звикання. React швидкий завдяки віртуальному DOM, проте може виявитися складнішим у великих проєктах через потребу у додаткових бібліотеках. Крива навчання помірна, вимагає знання JavaScript та JSX [11].

Vue.js – легкий фреймворк з простим синтаксисом, чудово підходить для малих та середніх проєктів. API інтуїтивно зрозумілий, а крива навчання нижча, ніж у React. Vue постачається з вбудованими інструментами, такими як Vuex та Vue Router, що зменшує залежність від зовнішніх бібліотек. Продуктивність на рівні з React, проте Vue легший в оптимізації для менших проєктів. Популярність зростає, але вона менша, ніж у React.

Angular – це повноцінний фреймворк від Google, найкращий варіант для великих корпоративних застосунків. Він містить усе необхідне: двостороннє зв’язування даних, вбудовану маршрутизацію, інструменти для тестування. Angular складніший через TypeScript та суворі шаблони, що збільшує криву навчання. Продуктивність на хорошому рівні, але він важчий за React та Vue. Популярний у великих компаніях, але менш гнучкий.

Svelte – сучасний фреймворк, що компілює код у «чистий» JavaScript, не використовуючи віртуальний DOM. Це забезпечує високу продуктивність та менший розмір бандлів. Синтаксис простий, нагадує Vue, проте код більш компактний. Екосистема менша, що може ускладнювати пошук рішень. Svelte

оптимальний для проєктів, де пріоритетом є швидкість, але популярність його нижча.

Solid.js – подібний до React за синтаксисом, проте використовує реактивність без віртуального DOM, що збільшує його швидкість. Він мінімалістичний, з невеликою екосистемою, але складніший у вивченні через концепції реактивності. Популярність нижча, ніж у React, але зростає серед розробників, які шукають продуктивність.

React перемагає у гнучкості та розгалуженості екосистеми, Vue – у простоті, Angular – у масштабованості, Svelte та Solid.js – у продуктивності. Вибір залежить від потреб конкретного проєкту: React для універсальності, Vue для легкості, Angular для великих застосувань, Svelte та Solid.js для швидкості.

Завдяки віртуальному DOM, React забезпечуватиме швидке оновлення лише тих елементів інтерфейсу, які змінилися, що підвищить продуктивність і покращить користувацький досвід. Модульність компонентів спростить розробку та подальшу підтримку коду, дозволяючи легко додавати нові функції.

Стилізація інтерфейсу буде реалізована через Tailwind CSS, утилітарний фреймворк, який прискорить створення адаптивного дизайну. Tailwind дозволить визначати стилі безпосередньо в HTML-класах, що зменшить обсяг CSS-коду та полегшить підтримку стилів. Цей інструмент підтримуватиме кастомізацію тем, адаптивну верстку та інтеграцію з React, що зробить його ідеальним для створення візуальних елементів, таких як кнопки, списки, картки профілів і модальні вікна. Використання Tailwind сприятиме створенню сучасного, гнучкого дизайну, який однаково добре виглядатиме на різних пристроях. Tailwind має зручну документацію яка відзначається простотою, логічною структурою та чудово підходить навіть для новачків у веброзробці [12].

Взаємодія між фронтендом і бекендом здійснюватиметься через REST API. Користувацькі дії, такі як створення команди, надсилання запрошення чи відправлення повідомлення в чат, генеруватимуть HTTP-запити, які

оброблятимуться сервером. Бекенд перевірятиме права доступу, взаємодіятиме з базою MySQL через ORM і повертатиме відповідь у JSON-форматі. Серіалізатори DRF формуватимуть структуровані дані, наприклад, для запрошень із деталями про відправника та команду, що дозволить фронтенду відображати інформацію без додаткових запитів.

Автентифікація через Steam буде реалізована за допомогою Steam OpenID. Користувач перенаправлятиметься на сайт Steam, де підтверджуватиме свою особу. Після цього сервер отримуватиме SteamID, перевірятиме його через Steam Web API, створюватиме або оновлюватиме обліковий запис користувача в базі MySQL і видаватиме JWT-токен через simplejwt. Цей токен використовуватиметься для авторизації подальших запитів, включаючи підключення до WebSocket для чату.

Обраний стек технологій, що включатиме Python, Django, Django REST Framework, Django Channels, MySQL, React і Tailwind CSS, відповідатиме вимогам проєкту. Ці інструменти забезпечуватимуть стабільну роботу бекенду, швидкий і зручний фронтенд, а також створюватимуть основу для впровадження нових функцій у майбутньому. Використання MySQL як бази даних підтримуватиме ефективну роботу з даними, тоді як Django Channels забезпечуватиме асинхронну взаємодію для чату. React і Tailwind створюватимуть сучасний інтерфейс, який легко адаптуватиметься до потреб користувачів. Така комбінація технологій дозволить досягти балансу між продуктивністю, безпекою та зручністю, відповідаючи стандартам веброзробки та залишаючи простір для подальшого розвитку системи.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи проведено детальний розбір потреб до системи, що розробляється. Обґрунтовано вибір технологій та

спроектовано архітектуру вебзастосунку, призначеного для організації команд гравців, їх аутентифікації через Steam, забезпечення зв’язку та керування ігровими сесіями. Визначено основні функціональні можливості, включаючи авторизацію за допомогою Steam OpenID з використанням JWT-токенів, створення й адміністрування команд, систему запрошень, чат у реальному часі та адаптивний інтерфейс. Нефункціональні вимоги, зокрема, безпека, продуктивність, масштабованість та юзабіліті, покладено в основу формування структури системи.

Для втілення проєкту обрано технологічний стек: Python, Django, Django REST Framework, Django Channels, MySQL, React та Tailwind CSS. Python і Django забезпечують стабільний бекенд із зручним ORM для роботи з MySQL, що вирізняється швидкістю та надійністю. Django REST Framework і simplejwt будуть відповідати за безпечний REST API, а Django Channels надасть можливість асинхронного спілкування через WebSocket. На стороні фронтенду React і Tailwind CSS дадуть змогу створити модульний, адаптивний та сучасний інтерфейс, який буде задовольняти потреби користувачів на різних пристроях.

РОЗДІЛ 3
РОЗРОБКА ВЕБПЛАТФОРМИ ДЛЯ ОРГАНІЗАЦІЇ ІГРОВИХ СЕСІЙ З ВИКОРИСТАННЯМ DJANGO, REACT ТА MYSQL

3.1 Практична реалізація об’єкта проектування

Розробка вебдодатку для геймерів – це багатогранний процес, який вимагає застосування найсучасніших технологій для забезпечення комфорту використання, надійності та можливості масштабування. Розробка здійснювалася з використанням мови Python, разом із Django, Django REST Framework і Django Channels для бекенду, бази даних MySQL, а також React та Tailwind CSS для фронтенду.

Ієрархія файлів проєкту презентує добре структуровану організацію, що інтегрує серверну та клієнтську частини, і має кілька ключових складових, що визначають його функціонал. На вершині проєкту знаходяться два основні каталоги: backend та frontend, кожен з яких виконує свою специфічну роль в розробці. На рисунку 3.1 можна побачити сформовану ієрархію файлів.

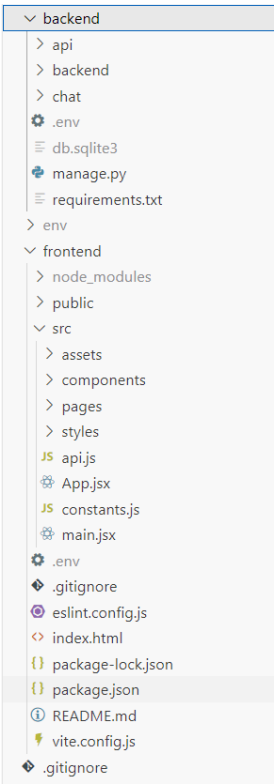

Рисунок 3.1 – Ієрархія файлів проєкту

У каталозі backend зосереджено серверну логіку проєкту. Центральним компонентом тут є файл manage.py, який виступає основним скриптом для управління проєктом. Зазвичай, у таких фреймворках, як Django, цей файл використовується для виконання команд, наприклад, міграція бази даних або запуск сервера. Іншим важливим елементом є db.sqlite3, файл бази даних SQLite, яка використовувалась для тестування бази даних при розробці, де зберігаються всі дані проєкту, включно з користувачами, налаштуваннями чи іншою інформацією, необхідною для роботи бекенду.

Каталог frontend відповідає за клієнтську частину проєкту та містить кілька важливих елементів. Найважливішим підкаталогом є src, який містить ядро коду фронтенду. У ньому файл App.jsx є головним компонентом, що визначає структуру та логіку основного інтерфейсу користувача, а main.jsx є точкою входу для рендерингу застосунку, де ініціалізується весь клієнтський код.

На кореневому рівні frontend також знаходяться ключові конфігураційні файли. package.json відіграє роль маніфесту проєкту, де вказані всі залежності, а також скрипти для запуску, тестування чи збирання коду. Файл vite.config.js забезпечує налаштування Vite – сучасного інструменту для розробки фронтенду, який оптимізує швидкість компіляції та гаряче оновлення під час розробки. Ці елементи разом створюють комфортне середовище для роботи з клієнтською частиною, інтегруючи сучасні технології та інструменти.

Ця структура описує гібридний підхід до розробки, де backend Django обробляє логіку та дані, а frontend на React надає інтерактивний інтерфейс. Кожен з виділених елементів виконує незамінну роль у функціонуванні проєкту, відображаючи його масштабованість та модульність.

Бекенд реалізовано на Python з використанням фреймворку Django, що дозволяє швидко розробляти надійні вебдодатки. Django REST Framework використовується для створення REST API для взаємодії з фронтендом, а Django Channels додає підтримку WebSocket для чату. MySQL застосовується як база даних для зберігання інформації про користувачів, команди, запрошення та повідомлення. Фронтенд реалізовано на React, що забезпечує модульність і

швидке оновлення інтерфейсу, а Tailwind CSS відповідає за адаптивне стилізування.

Система містить декілька ключових компонентів. Модуль автентифікації використовує Steam OpenID, де користувача перенаправляють на сторінку Steam, де він підтверджує свою особу, після чого сервер отримує SteamID та створює або оновлює обліковий запис у базі даних. Видається JWT-токен для подальших запитів. Модуль керування командами дозволяє створювати, редагувати команди та приєднуватися до них. Власники команд керують учасниками, а звичайні користувачі надсилають заявки або приймають запрошення. Модуль запрошень дає змогу власникам надсилати запрошення, які одержувачі обробляють через свій профіль. Чат функціонує через WebSocket, надаючи учасникам команд змогу обмінюватися повідомленнями в реальному часі. Модуль профілів відображає інформацію про користувача, отриману через Steam Web API, та список команд.

Архітектура системи розподіляє відповідальність між клієнтом та сервером. Бекенд обробляє запити через REST API та WebSocket, використовуючи Django ORM для роботи з MySQL. Фронтенд відображає дані та надсилає запити, а Steam Web API забезпечує автентифікацію. Взаємодія відбувається наступним чином: користувач автентифікується, отримує токен, надсилає запити до API або підключається до чату, а сервер повертає відповідь.

Python відіграє центральну роль у реалізації бекенду. Завдяки простоті синтаксису та великій кількості доступних бібліотек, Python дозволяє ефективно обробляти запити, взаємодіяти з базою даних та інтегруватися із зовнішніми сервісами, зокрема, Steam Web API. Мова підтримує асинхронне програмування, необхідне для роботи з WebSocket, і забезпечує масштабованість завдяки модульності коду. Використання Python разом із Django полегшує розробку складної логіки, як-от управління командами чи обробка запрошень, та забезпечує високу продуктивність.

Django слугує основою бекенду, надаючи структуру для швидкої розробки. Фреймворк застосовує ORM для взаємодії з MySQL, дозволяючи працювати з

даними через Python-об’єкти. Наприклад, модель Team зв’язана з User через ManyToManyField, що спрощує керування учасниками. Django надає захист від атак, таких як CSRF і XSS, за допомогою вбудованих механізмів. Маршрутизація запитів налаштовується через файли urls.py, де визначаються ендпоінти. Фреймворк підтримує аутентифікацію, інтегруючись із simplejwt для видачі JWT-токенів, і забезпечує масштабованість завдяки модульній структурі.

REST API, реалізований за допомогою Django REST Framework, виступає мостом між бекендом і фронтендом. API надає ендпоінти для взаємодії з командами запрошеннями, профілями та чатом. Кожен ендпойнт повертає дані у форматі JSON, сформовані серіалізаторами DRF. Наприклад, серіалізатор для запрошень містить дані про відправника та команду. DRF обробляє аутентифікацію через JWT, валідацію запитів та пагінацію, забезпечуючи безпеку та ефективність. Запити до API надсилаються за допомогою HTTP-методів (get, post, put), що дозволяє фронтенду динамічно оновлювати дані.

Для аутентифікації користувач перенаправляється на Steam, сервер отримує SteamID, створює запис у MySQL та видає токен. Команди створюються через форму, яка надсилає POST-запит до API, а сервер зберігає дані. Запрошення надсилаються через API, а їхній статус оновлюється після дій одержувача. Чат функціонує через WebSocket: повідомлення надсилаються на сервер, зберігаються та розсилаються учасникам. Інтерфейс оновлюється в реальному часі.

Для розробки клієнтської сторони інформаційної системи було обрано сучасний інструмент для збирання проєктів – Vite. Цей потужний інструмент для фронтенд-розробки забезпечує блискавичний запуск dev-сервера, швидке «гаряче» перезавантаження та оптимізовану фінальну збірку застосунку. За допомогою його простого посібника, вдалось без жодних проблема зібрати проект [6]. Vite спеціально розроблений для роботи з фреймворками нового покоління, такими як React, Vue та Svelte, що робить його оптимальним вибором для реалізації інтерфейсу з використанням React.

Однією з ключових переваг Vite є використання ES-модулів безпосередньо у браузері під час розробки. Це дозволяє уникнути необхідності в повній попередній збірці коду, що значно пришвидшує процес розробки. Зміни одразу відображаються лише у тих частинах проєкту, які було змінено. Під час збирання для продакшену Vite використовує Rollup, забезпечуючи ефективну оптимізацію, мінімізацію та розділення коду для досягнення найкращої продуктивності.

В рамках розробки платформи було ініціалізовано проект з використанням шаблону vite + react. Також було встановлено необхідні бібліотеки, такі як react-router-dom, axios, jwt-decode та інтегровано Tailwind CSS для стилізації. Основний файл main.jsx здійснює рендеринг кореневого компонента App, а реалізація навігації базується на маршрутизації за допомогою react-router.

Враховуючи модульну структуру, яку підтримує Vite, усі компоненти були організовані в окремих каталогах. Кожен каталог має чітко визначену зону відповідальності. Це сприяє підтримці та розширенню клієнтської частини застосунку, а також дозволяє ефективно переносити UI-компоненти між різними розділами системи.

Фронтенд збудований на React, бібліотеці для створення інтерфейсів. React застосовує компонентний підхід, де інтерфейс розділяється на незалежні елементи, такі як Profile, TeamList чи Chat. Компоненти забезпечують повторне використання коду та спрощують підтримку. Наприклад, компонент TeamList відображає список команд, отриманих через API, а Chat обробляє повідомлення WebSocket. React використовує віртуальний DOM для швидкого оновлення тільки змінених елементів, що покращує продуктивність.

Ключові методи React, застосовані у проекті, включають хуки. Локальним станом керує useState, наприклад, списком повідомлень у чаті. useState використовується для встановлення змінних стану нашого компонента. Ми визначаємо ці хуки, як тільки відкриваємо оголошення нашої функції. Жодних класів у цьому демо не використовується, і нам також не потрібно використовувати setState для оновлення стану. useEffect обробляє побічні

ефекти, наприклад, підключення до WebSocket або завантаження даних через API. Він виконує зовсім іншу функцію, ніж useState. Він має справу з побічними ефектами компонента – наприклад, з тим, що обробляється під час процесів монтування та демонтажу компонента. Для прокручування чату до останнього повідомлення використовується useRef [1]. Ці методи дозволяють створювати динамічні інтерфейси з мінімальним обсягом коду. React Router забезпечує навігацію між сторінками, наприклад, між профілем та списком команд, без перезавантаження сторінки.

Tailwind CSS використовується для стилізації фронтенду. Цей утилітарний фреймворк дозволяє визначати стилі за допомогою класів в HTML, наприклад, flex, p-4 чи bg-blue-500. Tailwind прискорює розробку, усуваючи потребу у окремих CSS-файлах, і підтримує адаптивний дизайн за допомогою префіксів, таких як sm:, md:. У проекті Tailwind застосовується для створення кнопок, карток профілів, модальних вікон та списків. Інтеграція з React полегшує стилізацію компонентів, наприклад, додавання тіні до картки за допомогою shadow-lg.

Бекенд налаштовано на сервері з використанням Python. Django підключено до MySQL, а Channels налаштовано для WebSocket. API включає маршрути для всіх модулів. Сервер запускається через Django для тестування, а WebSocket обробляється спеціальним інструментом. Фронтенд розроблено на React з Tailwind CSS, підключеним через npm. Запити до API реалізуються через бібліотеку Axios. Вона пропонує простий і зрозумілий спосіб відправлення HTTP-запитів. Axios підтримує роботу з промісами, що дозволяє зручно обробляти відповіді. Також передбачені можливості перехоплення запитів та відповідей, а ще спрощується робота з помилками і налаштування заголовків., а WebSocket відповідає за обробку чату.

Безпека забезпечується аутентифікацією через Steam, JWT-токенами, валідацією запитів та захистом від атак за допомогою Django. Для масштабування передбачається використання інструментів для чату та кешування. Тестування включає перевірку API, взаємодії фронтенду з бекендом

та роботи інтерфейсу. Оптимізація передбачає скорочення запитів та підтримку пагінації.

Розробка вебдодатку включає в себе ключові функції для комунікації користувачів у ігровому світі. Python та Django гарантують стабільну роботу серверної частини, REST API виступає сполучною ланкою між клієнтом та сервером, а React у поєднанні з Tailwind CSS формують сучасний та зручний інтерфейс. Модульна архітектура забезпечує можливість масштабування та налаштування під потреби цільової аудиторії.

3.2 Тестування та налагодження інформаційно-комп'ютерної системи

В процесі тестування та виправлення помилок інформаційно-обчислювальної системи важливе значення мають інструменти розробника, що доступні у сучасних браузерах, та також спеціалізовані рішення, наприклад, Postman. Вбудовані засоби розробки браузерів (зокрема Chrome, Firefox, Edge) надають можливість відстежувати HTTP-запити, розбирати відповіді сервера, переглядати структуру DOM, досліджувати помилки JavaScript, керувати завантаженням ресурсів та станом кешу. Ці інструменти є незамінними для налагодження клієнтської частини, перевірки правильності звернень до API, а також візуального тестування інтерфейсу.

Для функціональної перевірки REST API та тестування взаємодії з базою даних через бекенд, надзвичайно популярним є Postman – багатогранний інструмент, що пропонує зрозумілий інтерфейс для створення, налаштування та відправлення HTTP-запитів різноманітних типів (get, post, put, delete тощо). За допомогою Postman легко додавати заголовки та параметри, а також формувати тіло запиту у таких форматах, як JSON, дані форм чи інших. Запити в Postman зручно групувати у колекції, що особливо ефективно для організації тестових сценаріїв та їх багаторазового використання. На рисунку 3.2 зображено приклад тестування запиту API за допомогою Postman.

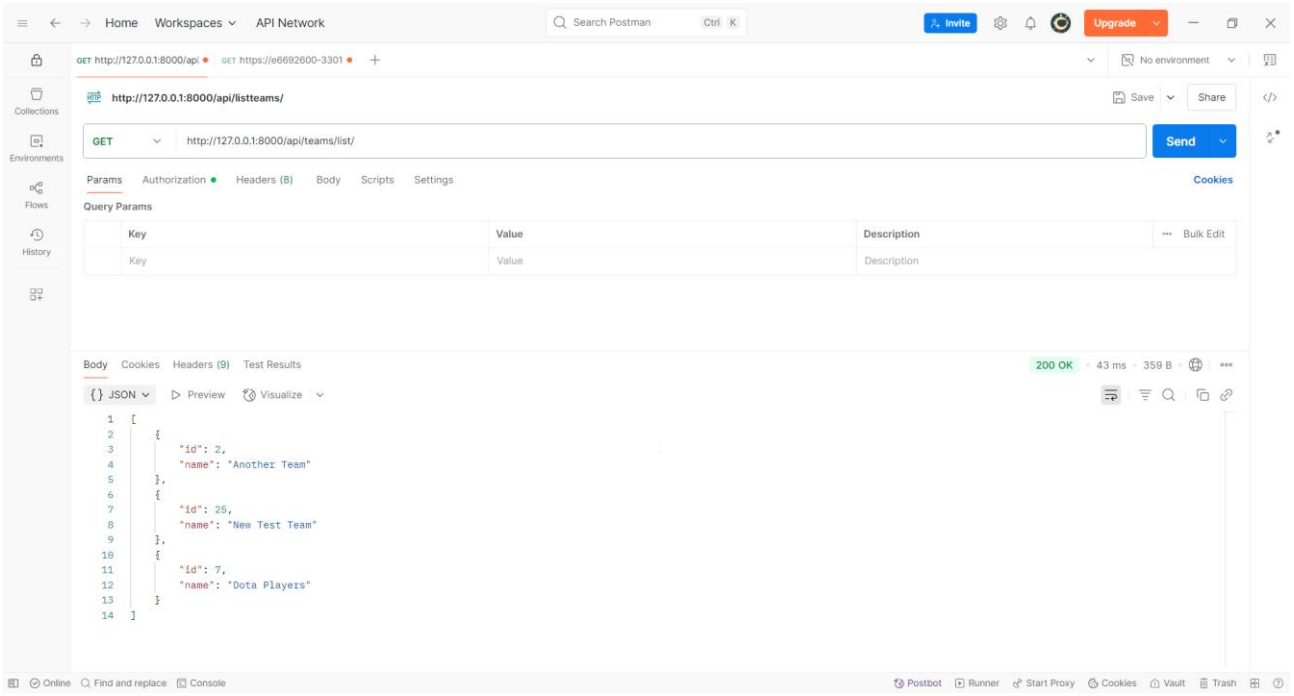

Рисунок 3.2 – Тестування API за допомогою Postman [10]

Postman дозволяє перевірити коректність маршрутизації, обробки запитів, авторизації користувачів, а також відслідкувати, як саме змінюються дані у базі даних в результаті певних дій. Це дає змогу швидко виявляти помилки, перевіряти логіку контролерів та впевнено контролювати якість серверної частини системи.

3.3 Розробка бази даних

У процесі створення інформаційної системи для організації геймерських команд ключовим моментом було розроблення прикладного програмного інтерфейсу (API), що забезпечує взаємодію між клієнтською стороною та серверною частиною. Для цього було обрано Django REST Framework (DRF) – потужний інструментарій, що надає зручні засоби для побудови RESTful API та підтримує роботу з об’єктно-реляційною моделлю Django.

Структура API ґрунтується на взаємодії трьох основних компонентів: моделей, представлень та маршрутів. Моделі визначають структуру даних,

представлення відповідають за логіку обробки HTTP-запитів, а маршрути відповідають за зіставлення URL-адрес із відповідними представленнями.

Першим кроком у створенні API було моделювання ключових сутностей системи, таких як користувач, команда, запрошення, повідомлення в чаті тощо. Для цього було створено відповідні класи, кожен з яких відповідає певній таблиці в базі даних. Django ORM автоматично перетворює ці класи у SQL-запити, забезпечуючи прозору та ефективну роботу з базою MySQL. Це дозволяє уникнути ручного написання SQL-коду, зосереджуючи увагу на логіці додатку. ORM підтримує базові операції: створення, оновлення, видалення, фільтрацію та зв’язки між об’єктами, наприклад, один до багатьох чи багато до багатьох.

Наступним етапом було створення серіалізаторів, що забезпечують перетворення даних між форматами Python-об’єктів та JSON, який використовується для обміну даними з клієнтом. Кожна модель має відповідний серіалізатор, який визначає, які поля будуть доступні для читання та запису.

У views.py реалізовано класові представлення, що успадковуються від APIView або GenericViewSet, які обробляють запити типів get, post, put, delete та інші. Наприклад, для обробки запитів на створення нової команди, перегляд її учасників або відправку запрошення використовуються окремі ендпоінти. При цьому доступ до окремих функцій контролюється за допомогою механізмів авторизації та перевірки прав доступу. Django REST Framework дозволяє реалізувати кастомні permission класи, які перевіряють, чи є користувач власником певного ресурсу, перш ніж дозволити виконання дії.

У urls.py налаштовуються маршрути, що відповідають за перенаправлення запитів до відповідних представлень. Для організації структури API застосовано підхід розбиття маршрутів на логічні блоки (наприклад, /api/teams/, /api/invites/, /api/messages/). DRF дозволяє зручно об’єднувати маршрути за допомогою DefaultRouter, що автоматично генерує стандартні CRUD-шляхи на основі вказаних viewset’ів. Для зберігання та отримання відомостей про користувача Steam були впроваджені два ключові API-ендпоінти:

– SteamUserCreateUpdateView – клас представлення, побудований на базі CreateAPIView. Його задача – створення або оновлення запису про користувача Steam. Специфікою реалізації є перевизначення методу perform_create, де стандартне створення нового об’єкта замінено на метод update_or_create ORM-системи Django. Тобто, якщо об’єкт SteamUser для поточного користувача вже існує, він буде оновлений, а якщо ні – створено новий. Це полегшує роботу користувача, коли дані надсилаються повторно;

– SteamUserView – представлення, реалізоване на основі RetrieveAPIView, яке дозволяє отримати об’єкт профілю Steam користувача. Метод get_object повертає SteamUser, прив’язаний до аутентифікованого користувача. Завдяки такому підходу немає потреби в передачі ідентифікатора користувача у запиті, що підвищує безпеку та спрощує взаємодію з API.

Обидва класи застосовують SteamUserSerializer для серіалізації та десеріалізації об’єктів, а також обмеження доступу за допомогою permission_classes = [IsAuthenticated], забезпечуючи доступ лише авторизованим користувачам. Лістинг 3.1 показує реалізацію зберігання та відображення інформації зі Steam профілю користувача.

Лістинг 3.1 – Код для збереження та відображення даних Steam користувача

```
class SteamUserCreateUpdateView(generics.CreateAPIView):
    serializer_class = SteamUserSerializer
    permission_classes = [IsAuthenticated]

    def perform_create(self, serializer):
        steam_user, created = SteamUser.objects.update_or_create(
            user = self.request.user,
            defaults={
                'persona_name':
serializer.validated_data.get('persona_name'),
                'profile_url':
serializer.validated_data.get('profile_url'),
                'avatar_url':
serializer.validated_data.get('avatar_url'),
            }
        )

class SteamUserView(generics.RetrieveAPIView):
    serializer_class = SteamUserSerializer
```

```
    permission_classes = [IsAuthenticated]

    def get_object(self):
        return SteamUser.objects.get(user=self.request.user)
```

кінець лістингу 3.1

Після виконання запиту до цих API-ендпоїнтів ми отримуємо результат який зображений на рисунку 3.3.

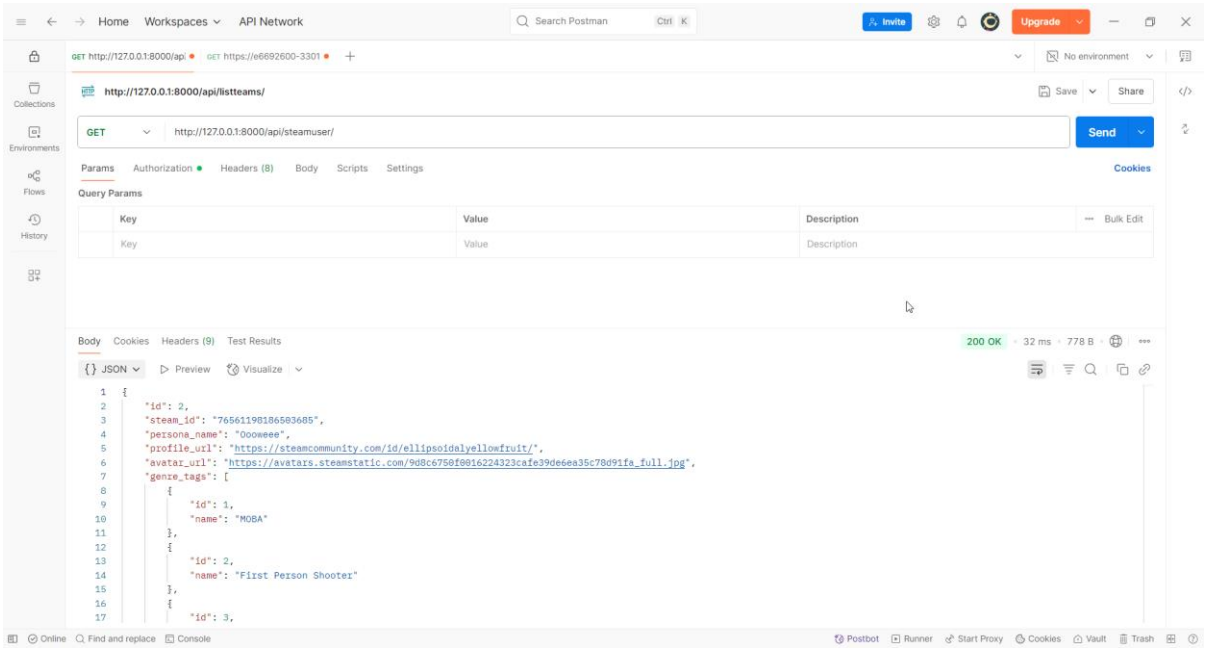

Рисунок 3.3 – Результат запиту до API-ендпоїнта

Серіалізатори в Django, зокрема в рамках Django REST Framework, – це надзвичайно ефективний інструмент для трансформації комплексних даних, скажімо, моделей або запитів до баз даних, у формати, зручні для передачі через API, як-от JSON чи XML, а також для виконання оберненого процесу – десеріалізації даних з метою створення або коригування об’єктів в базі даних. Вони є критично важливим елементом при розробці RESTful API, гарантуючи практичний і захищений спосіб взаємодії між сервером і клієнтом.

Головна концепція серіалізаторів полягає у автоматизації процесів валідації, серіалізації та десеріалізації, дозволяючи розробникам сфокусуватися на бізнес-логіці замість ручного перетворення даних. У DRF серіалізатори

виконують роль аналогічну формам у стандартному Django: вони визначають, які поля моделі будуть відображені в API, як вони будуть представлені, які з них призначені тільки для читання, а також надають можливість додавати власну логіку для обробки даних.

Серіалізатори дають можливість окреслити, які поля моделі будуть експоновані через API, та конфігурувати їхню роботу через атрибут fields у класі Meta. Параметр extra_kwargs у класі Meta дозволяє задавати додаткові налаштування, наприклад, встановити поле лише для читання. Серіалізатори також можуть працювати із вкладеними об’єктами, що дозволяє серіалізувати взаємопов’язані моделі за допомогою таких інструментів як SerializerMethodField або вкладених серіалізаторів.

Специфічна логіка інтегрується через методи, наприклад, SerializerMethodField, які дають розробникам можливість визначити, як обчислюватиметься значення конкретного поля. Це корисний підхід, коли потрібно повернути дані, які не є безпосередніми полями моделі, або коли потрібно зробити додаткові запити до бази даних. Серіалізатори також надають функцію валідації даних: можливо визначати власні методи валідації або перезаписувати метод validate для комплексної перевірки [13].

Особливу увагу приділено взаємодії між компонентами. Наприклад, коли користувач надсилає запит на створення команди, представлення звертається до серіалізатора, який перевіряє коректність отриманих даних та ініціалізує відповідний об’єкт моделі. Модель, у свою чергу, через ORM створює запис у базі даних. Після цього відповідь у форматі JSON повертається на клієнтську сторону.

Уся логіка побудови API поділена на модулі за принципами REST: кожна сутність має власні ендпоінти, і кожен ресурс може бути оброблений за допомогою одного стандартного набору HTTP-методів. Це дозволяє легко масштабувати систему, додаючи нові функції або розширюючи існуючі без порушення архітектури.

Розгляньмо серіалізатор TeamSerializer, що міститься у лістингу 3.2, який ми застосовуємо для роботи з моделлю Team. Його функція – серіалізація даних про команду, зокрема її учасників, власника та дані про поточного користувача.

Лістинг 3.2 – Код серіалізатора для команд

```
class TeamSerializer(serializers.ModelSerializer):
    members = serializers.SerializerMethodField()
    owner = serializers.SerializerMethodField()
    current_user_id = serializers.SerializerMethodField();

    class Meta:
        model = Team
        fields = ["id", "name", "description", "owner", "members",
"max_members", "current_user_id"]
        extra_kwargs = {"owner": {"read_only": True}}

    def get_members(self, obj):
        steam_users =
SteamUser.objects.filter(user__in=obj.members.all())
        return SteamUserSerializer(steam_users, many=True).data

    def get_owner(self, obj):
        steam_user = SteamUser.objects.get(user=obj.owner)
        return SteamUserSerializer(steam_user).data

    def get_current_user_id(self, obj):
        steam_user =
SteamUser.objects.get(user=self.context["request"].user)
        return {'id': steam_user.id, 'steam_id':
self.context['request'].user.username}
```

кінець лістингу 3.2

Клас Meta задає ключові конфігурації серіалізатора. Параметр model = Team визначає, що серіалізатор працює з моделлю Team. Відповідно, він автоматично бере поля з цієї моделі, якщо вони вказані у fields. У fields задається перелік полів, що будуть включені до API. Це прямі поля моделі id, name, description, max_members, а також власні поля owner, members, current_user_id, які обчислюються через методи. Через extra_kwargs поле owner помічено як read_only=True. Отже, його не можна змінити через API, а тільки переглядати.

Це логічно, оскільки власника команди призначають на етапі створення команди, і його не можна змінити через звичайний запит.

Три поля members, owner та current_user_id визначені як SerializerMethodField. Це означає, що їх значення обчислюються через відповідні методи get_members, get_owner, get_current_user_id. Це дозволяє додати власну логіку для цих полів. Метод get_members повертає список учасників команди. Поле obj.members у моделі Team є ManyToMany-полем, яке пов’язує команду з користувачами. Для кожного користувача з obj.members виконується запит до моделі SteamUser, що розширює стандартну модель User та містить додаткові дані, наприклад, Steam ID. Далі отримані об’єкти SteamUser серіалізуються через SteamUserSerializer з параметром many=True, що вказує на обробку списку об’єктів. Результатом є JSON-список з даними про всіх учасників команди. Метод get_owner повертає дані про власника команди. Поле obj.owner у моделі Team є ForeignKey-полем, яке посилається на одного користувача (модель User). Запит до SteamUser витягує відповідний об’єкт, який потім серіалізується через SteamUserSerializer, що дає JSON-об’єкт з даними власника, наприклад, його ID, ім’я чи Steam ID. Метод get_current_user_id повертає інформацію про поточного користувача, що робить запит до API. Він використовує контекст серіалізатора self.context["request"] для отримання користувача, що виконав запит (request.user). Далі виконується запит до SteamUser, щоб отримати відповідний об’єкт, і повертається словник з двома полями, id та steam_id (Steam ID користувача, що зберігається як username у моделі User). Це корисно для клієнтської частини, щоб визначити, чи є поточний користувач членом команди або її власником.

Поле current_user_id залежить від контексту, зокрема від об’єкта request, який передається в серіалізатор. Це типовий підхід у DRF, коли серіалізатору потрібно знати, хто виконує запит, щоб повернути персоналізовані дані. Наприклад, клієнтська частина може використовувати current_user_id для перевірки прав доступу або відображення відповідних кнопок, таких як «Приєднатися до команди» або «Вийти з команди».

Отже, використання Django REST Framework у поєднанні з ORM дало змогу реалізувати надійний, масштабований та безпечний API, що відповідає вимогам сучасної вебсистеми та забезпечує стабільну взаємодію між клієнтом і сервером.

3.4 Реалізація чатів за допомогою Django Channels

У межах кваліфікаційної роботи для реалізації функціоналу чатів було застосовано Django Channels. Це дало змогу інтегрувати асинхронну обробку WebSocket-з'єднань до Django, зберігаючи при цьому структуру фреймворку. Такий вибір зумовлений його сумісністю з Django, підтримкою асинхронного обміну даними в режимі реального часу та можливістю масштабування для забезпечення роботи чатів в ігровому вебдодатку.

Для збереження активних WebSocket-з'єднань та обміну повідомленнями між клієнтами використано Redis як брокер повідомлень. Redis було встановлено локально або на сервері. Як показано у лістингу 3.3, settings.py було додано налаштування з використанням Redis як бекенду.

Лістинг 3.3 – Налаштування для брокера повідомлень Redis

```
CHANNEL_LAYERS = {
    'default': {
        'BACKEND': 'channels_redis.core.RedisChannelLayer',
        'CONFIG': {
            'hosts': [('127.0.0.1', 6379)],
        },
    },
}
```

кінець лістингу 3.3

Це забезпечило асинхронний обмін даними між різними екземплярами застосунку, що важливо для масштабування чатів.

У роботі чати реалізовано в окремому додатку chat, який містить основну логіку обробки повідомлень. Модель ChatMessage, як можна побачити на лістингу 3.4, створена для зберігання повідомлень у базі даних.

Лістинг 3.4 – Модель повідомлень для чатів

```
class ChatMessage(models.Model):
    team = models.ForeignKey(Team, on_delete=models.CASCADE,
related_name='messages')
    sender = models.ForeignKey(User, on_delete=models.CASCADE)
    content = models.TextField(max_length=500)  # Обмеження для безпеки
    timestamp = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['timestamp']

    def __str__(self):
        return f'{self.sender.username}: {self.content}'
```

кінець лістингу 3.4

Модель ChatMessage містить такі поля як team, для зберігання ідентифікатора команди, sender, для відображення відправника повідомлення, content, для зберігання тексту повідомлення і timestamp, для того щоб розуміти коли було надіслане повідомлення. Для серіалізації повідомлень створено ChatMessageSerializer у serializers.py, який перетворює об’єкти моделі в JSON-формат для передачі через WebSocket.

Маршрутизація WebSocket-з’єднань визначена у файлі routing.py. У ньому створено WebSocket URL-паттерн для підключення до чатів який зображено на лістингу 3.5.

Лістинг 3.5 – URL-паттерн для WebSocket

```
websocket_urlpatterns = [
    re_path(r'ws/chat/(?P<team_id>\d+)/$',
consumers.ChatConsumer.as_asgi()),
]
```

кінець лістингу 3.5

Цей маршрут дозволяє клієнтам підключатися до певної чат-кімнати за її назвою.

Основна логіка чату реалізована у файлі consumers.py каталогу через клас ChatConsumer, який успадковується від AsyncWebsocketConsumer для асинхронної обробки з'єднань. У методі connect клас підключає клієнта до групи каналів Django Channels, що відповідає назві чат-кімнати, отриманій з URL. Це дозволяє надсилати повідомлення всім учасникам кімнати. Метод disconnect видаляє клієнта з групи після завершення з'єднання.

Метод receive обробляє вхідні повідомлення від клієнта. Отримане повідомлення перетворюється з JSON у словник, після чого витягується текст повідомлення та інформація про відправника. Далі повідомлення зберігається в базі даних шляхом створення об'єкта ChatMessage, а потім надсилається всім учасникам кімнати через group_send. Метод chat_message викликається для обробки повідомлень, отриманих з групи, та надсилає їх клієнту у форматі JSON, включаючи текст повідомлення, ім'я відправника та час відправлення.

На стороні фронтенду, створено компонент Chat.jsx для роботи з чатом. У цьому компоненті використано JavaScript WebSocket API для підключення до бекенду через URL. Більш точно про цей інтерфейс описано у його документації [15]. Після підключення клієнт надсилає повідомлення через WebSocket і отримує відповіді від сервера. Отримані повідомлення відображаються в інтерфейсі в режимі реального часу через оновлення стану React-компонента.

Для запуску роботи з підтримкою Django Channels було використано сервер Daphne, який підтримує ASGI. У терміналі з каталогу backend виконано команду daphne -p 8000 backend.asgi:application, що запустило сервер для обробки WebSocket-з'єднань. Redis-сервер запущено окремо на порту 6379. Daphne – це сервер протоколів HTTP, HTTP2 та WebSocket для ASGI та ASGI-HTTP, розроблений для підтримки Django Channels. Вона підтримує автоматичне узгодження протоколів; немає потреби у префіксах URL для визначення кінцевих точок WebSocket порівняно з HTTP кінцевими точками [7]. На фронтенді застосунок запущено через npm run dev з каталогу frontend, що

активувало Vite для розробки. Під час тестування чатів у різних чат-кімнатах було підтверджено коректність роботи: повідомлення надсилалися та відображалися в реальному часі для всіх підключених клієнтів у межах однієї кімнати. На рисунку 3.4 зображена сторінка команди з працюючим чатом.

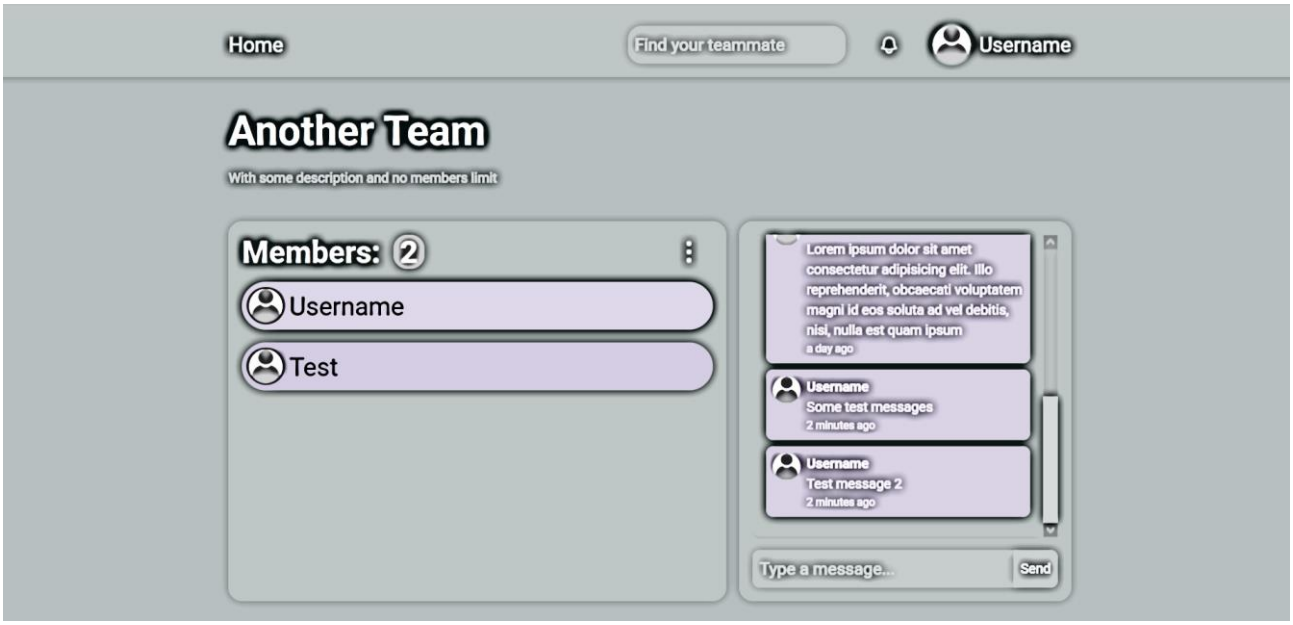

Рисунок 3.4 – Сторінка команди

Отже, Django Channels у поєднанні з React і Redis забезпечило ефективну реалізацію чатів у роботі, що відповідає вимогам ігрового вебдодатку, де швидкий обмін повідомленнями між користувачами є ключовою функціональністю.

3.5 Захист інформаційно-комп’ютерної системи

Одним з найважливіших аспектів у розробці інформаційної системи є забезпечення захисту від несанкціонованого доступу. Для вебплатформи, яка втілена в рамках цього проекту, захист реалізується як на серверному, так і на клієнтському рівнях. У цьому розділі буде розглянуто впровадження механізму авторизації на боці клієнта за допомогою компонента ProtectedRoute в середовищі React.

Компонент ProtectedRoute функціонує як оболонка для захищених сторінок додатку. Його задача – перевірити, чи автентифікований користувач, та надати доступ до внутрішнього вмісту лише за умови підтвердженої авторизації. Усі сторінки, що не повинні бути доступними для неавторизованих користувачів, такі як сторінка профілю, управління командою, чат та інше, обгортаються в ProtectedRoute.

Компонент здійснює перевірку наявності access-токена в localStorage. Якщо токен присутній, він декодується за допомогою бібліотеки jwt-decode, після чого перевіряється його термін дії. У випадку, якщо access-токен прострочений, відбувається спроба його оновлення за допомогою refresh-токена через відповідний endpoint. У разі успішного оновлення, новий access-токен зберігається в localStorage, і користувач отримує доступ до захищеного ресурсу. Якщо авторизація неможлива, токен відсутній або refresh-токен недійсний, користувача перенаправляють на сторінку логіну [8].

Цей механізм дозволяє забезпечити ефективний frontend-контроль доступу, запобігаючи доступу неавторизованих користувачів до внутрішніх частин застосунку. Важливо зазначити, що цей захист є лише додатковим рівнем безпеки: основна перевірка прав доступу виконується на серверній стороні за допомогою JWT-аутентифікації в Django REST Framework.

Отже, компонент ProtectedRoute відіграє важливу роль у побудові сучасної безпечної клієнтської логіки, забезпечуючи контроль сесій та зменшуючи ймовірність доступу до критичних частин системи сторонніми особами.

Висновки до розділу 3

У третьому розділі було створено функціональну інформаційну систему для кооперації геймерів. Ця система включає авторизацію через Steam, можливість створювати та адмініструвати команди, систему запрошень, чат у реальному часі та систему захисту від несанкціонованого доступу. Архітектура проекту розподілена на бекенд і фронтенд, з використанням Django REST

Framework, Django Channels, React та Tailwind CSS, що забезпечило масштабованість, безпеку та сучасний дизайн інтерфейсу. Інтеграція з OpenID Steam та використання JWT гарантують легкість та захищеність процесу аутентифікації користувачів.

Всі ключові елементи системи пройшли ретельне тестування, включаючи REST API, чат, механізм запрошень та функціонал обробки профілів Steam. Втілена архітектура продемонструвала свою продуктивність та здатність до подальшого вдосконалення платформи, підтримуючи високу ефективність та стабільність роботи. Розроблена система показує практичну цінність вибраного стеку технологій для побудови сучасного вебсервісу з функцією роботи в реальному часі та гнучким керуванням користувацькими командами.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було досягнуто поставленої мети шляхом реалізації комплексу завдань, що охоплюють проєктування, розробку та впровадження інформаційної системи для організації геймерських команд.

Було створено структуру бази даних, яка забезпечує зберігання інформації про користувачів, їхні Steam-акаунти, команди, запрошення до них та взаємозв’язки між цими сутностями. Під час розробки схеми було дотримано принципів нормалізації, що дозволило уникнути надлишкових даних і забезпечити логічну цілісність. Типи полів були підібрані відповідно до характеру даних, що оптимізувало продуктивність системи при збереженні та вибірці інформації.

На серверній частині проєкту реалізовано основну логіку взаємодії між компонентами системи. Було створено моделі для опису об’єктів предметної області, визначено маршрутизацію запитів і побудовано представлення для обробки даних. Особливу увагу приділено реалізації механізму авторизації користувачів через Steam OpenID, а також управлінню командною структурою – надсиланню, прийняттю та відхиленню запрошень і керуванню списком учасників команди.

Фронтенд-частина системи реалізована з використанням JavaScript, бібліотеки React та CSS-фреймворку Tailwind. Це дозволило створити адаптивний, зручний та інтуїтивно зрозумілий інтерфейс, який забезпечує якісну інтеграцію з бекендом через REST API. Користувачі мають змогу зручно працювати з функціоналом платформи, включаючи перегляд команд, взаємодію з запрошеннями та керування своїм профілем.

Було реалізовано реєстрацію та авторизацію користувачів виключно через Steam-акаунт, без використання традиційних логінів і паролів. Для цього інтегровано OpenID-автентифікацію за допомогою спеціалізованих бібліотек,

що дозволило отримувати базову профільну інформацію користувача, у тому числі його унікальний Steam ID, ім’я, аватар та інші публічні дані.

У завершальному етапі було здійснено порівняльний аналіз популярних хостинг-платформ з метою розміщення клієнт-серверного застосунку. Оцінювались такі критерії, як продуктивність, масштабованість, вартість використання та зручність інтеграції. За результатами аналізу обрано оптимальне рішення, яке забезпечує надійну роботу бекенду й фронтенду та дозволяє ефективно масштабувати проєкт у майбутньому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bulat R. React Hooks: Managing Web Sockets with useEffect and useState. Medium. URL: https://rossbulat.medium.com/react-hooks-managing-web-sockets-with-useeffect-and-usestate-2dfc30eeceec (date of access: 07.03.2025).

2. Challengermode. URL: https://www.challengermode.com (date of access: 13.01.2025).

3. Emanuilov S. Implementing real-time functionality in Django with WebSockets and Django channels. Medium. URL: https://medium.com/@simeon.emanuilov/implementing-real-time-functionality-in-django-with-websockets-and-django-channels-7240f75ee9b5 (date of access: 22.03.2025).

4. Faceit. URL: https://www.faceit.com/en/matchmaking/ (date of access: 13.01.2025).

5. GeeksforGeeks. 7 best languages for web development. URL: https://www.geeksforgeeks.org/best-web-development-languages/ (date of access: 17.01.2025).

6. Getting started. vitejs. URL: https://vite.dev/guide/ (date of access: 08.02.2025).

7. GitHub – django/daphne: Django Channels HTTP/WebSocket server. URL: https://github.com/django/daphne (date of access: 22.03.2025).

8. JWT authentication with refresh tokens – geeksforgeeks. URL: https://www.geeksforgeeks.org/node-js/jwt-authentication-with-refresh-tokens/ (date of access: 15.03.2025).

9. Matthes E. Python crash course. 3rd ed. No Starch Press, Incorporated, 2022. 552 p.

10. Postman: the world's leading API platform – sign up for free. URL: https://www.postman.com/ (date of access: 10.04.2025).

11. React. URL: https://react.dev/ (date of access: 08.02.2025).

12. Tailwind CSS – Rapidly build modern websites without ever leaving your HTML. URL: https://tailwindcss.com/docs/ (date of access: 10.02.2025).

13. Tomazic N. Effectively using django REST framework serializers. Test-Driven Development, Microservices, Web Development Courses – TestDriven.io. URL: https://testdriven.io/blog/drf-serializers/ (date of access: 05.03.2025).

14. Vincent W. S. Django for APIs: Build web APIs with Python and Django. WelcomeToCode, 2020. 208 p.

15. WebSocket – Web APIs – MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket (date of access: 20.03.2025).

ДОДАТКИ

Додаток А – Підтвердження апробації результатів кваліфікаційної роботи

Міністерство освіти і науки України
Волинська обласна рада
Луцька міська рада
Луцький національний технічний університет
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Національний університет «Львівська політехніка»
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут (м. Київ)
Тернопільський національний педагогічний університет імені В. Гнатюка
Рівненський державний гуманітарний університет
Навчально-науковий інститут «Українська інженерно-педагогічна академія» Харківського національного університету ім. В.Н. Каразіна
Люблінська політехніка (Польща)
Фрайберзька гірнича академія (Німеччина)
Гронінгенський університет (Нідерланди)
Кавказький університет (Грузія)
Політехнічний університет Браганси (Португалія)

ТЕЗИ ДОПОВІДЕЙ
X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)

23-24 травня 2025 р.

Луцьк 2025

СЕКЦІЯ 7. КІБЕРБЕЗПЕКА ТА ЗАХИСТ ІНФОРМАЦІЇ

СЕКЦІЯ 7. КІБЕРБЕЗПЕКА ТА ЗАХИСТ ІНФОРМАЦІЇ

УДК 004.41

АВТОРИЗАЦІЯ ЧЕРЕЗ СТОРОННІ СЕРВІСИ У СОЦІАЛЬНИХ МЕРЕЖАХ

Борбич Олег Валентинович
Луцький національний технічний університет, студент,
бакалавр інженерії програмного забезпечення, olegborbych@gmail.com
Суринович Олена Миколаївна
Луцький національний технічний університет, к.т.н., доцент кафедри
інженерії програмного забезпечення, sivom@ukr.net

AUTHORIZATION THROUGH THIRD-PARTY SERVICES IN SOCIAL NETWORKS

Borbych Oleh Valentynovych
Lutsk National Technical University, Student,
Bachelor of Software Engineering, olegborbych@gmail.com
Surynovych Olena Mykolayivna
Lutsk National Technical University, PhD in Technical Sciences,
Associate Professor of the Software Engineering Department, sivom@ukr.net

Today, social networks are a very important part of people's lives. And data privacy in such places is an integral part. To ensure the minimum probability of information leakage, there are ways to authorize using third-party services from large companies such as Google, Steam, or GitHub. When authorizing through these services, you do not provide any data, and the service you choose to authorize is responsible for data security

***Keywords**: OAuth, authorization, privacy, identification, social networks, usability*

У сучасному цифровому світі, де щодня відбувається величезна кількість онлайн-взаємодій, авторизація користувачів набуває ключового значення. Вона слугує засобом перевірки особистості користувача та дозволяє забезпечити доступ до персоналізованих функцій на вебсайтах та в додатках. Одним із найбільш поширених і зручних способів авторизації є

використання сторонніх сервісів, зокрема соціальних мереж, таких як Facebook, Google, Twitter, GitHub, Steam та інші.

Метою даної роботи було розробити авторизацію через Steam для веб сайту. Суть авторизації через сторонні сервіси полягає в тому, що замість створення окремого облікового запису на кожному сайті, користувач може увійти за допомогою існуючого акаунта в соціальній мережі. Це значно спрощує процес входу, особливо для нових користувачів, які не хочуть проходити реєстрацію та запам'ятовувати ще один логін і пароль. Використання такої авторизації базується на стандартах OAuth або OpenID, які дозволяють безпечно передавати інформацію про користувача від стороннього сервісу до вашого додатку без необхідності зберігати або бачити його пароль [2].

З технічної точки зору, реалізація авторизації через сторонні сервіси не надто складна. Більшість платформ надають докладну документацію та готові SDK (Software Development Kit), які полегшують інтеграцію. Основний принцип полягає в тому, що сайт редіректить користувача на сторінку сервісу (наприклад, Google), де той підтверджує свою згоду на передавання даних. Після цього користувач повертається на сайт із токеном, який підтверджує його ідентичність. Цей токен можна використати для отримання базової інформації про користувача: ім'я, e-mail, аватар тощо [1].

У власному проєкті я реалізував авторизацію через Steam, так як основна аудиторія веб-сайту це фанати ігор. На боці бекенду було налаштовано механізм редіректу користувача на сторінку підтвердження авторизації в Steam, після чого здійснювалась обробка відклику від сервісу та перевірка автентичності користувача за допомогою відкритого ключа. Успішна перевірка дозволяла отримати SteamID – унікальний ідентифікатор користувача, який використовувався як ключ у локальній базі даних. На рисунку 1 зображена сторінка авторизації Steam, яка підтверджує дані користувача та перенаправляє його назад до бекенду.

Рисунок 1 – Сторінка авторизації Steam

На основі SteamID формувався JWT-токен для доступу до захищених ресурсів сайту. Додатково витягувалась базова інформація користувача (нікнейм, аватар), яка зберігалась при першому вході. Це дозволило реалізувати безпарольну авторизацію, де вся відповідальність за захист облікового запису покладалась на Steam. Такий підхід значно спростив користувацький досвід і покращив безпеку, адже сайт не зберігав паролів або конфіденційних даних.

Ще одна перевага – це швидкість та зручність використання. Для входу достатньо одного кліку миші та підтвердження з боку сервісу. Це особливо актуально в мобільних додатках, де користувачі очікують максимально швидкої взаємодії. Сайти, які підтримують соціальну авторизацію, зазвичай отримують кращу конверсію нових користувачів у зареєстрованих, оскільки процес реєстрації мінімізовано.

Проте, окрім переваг, існують і певні ризики. Один із них – залежність від зовнішнього сервісу. Якщо, наприклад, сервіс

змінює умови API або тимчасово недоступний, це може вплинути на роботу вашого сайту. Також є питання конфіденційності: користувачі повинні бути впевнені, що їх дані використовуються лише в межах необхідного функціоналу. Тому важливо дотримуватись принципу мінімального збору інформації та чітко інформувати користувача про те, що саме буде використано.

Щодо використання соціальної авторизації в реальних проектах, варто згадати популярні платформи, які активно її застосовують. Наприклад, сайти електронної комерції (Etsy, Amazon), сервіси потокового відео (Spotify, Netflix), а також численні ігрові платформи (Steam, Epic Games). У кожному з цих випадків соціальна авторизація дозволяє пришвидшити доступ до сервісу, зменшити поріг входу і покращити користувацький досвід.

Підсумовуючи, можна сказати, що авторизація через сторонні сервіси у соціальних мережах є одним із найефективніших рішень для забезпечення безпечного, швидкого та зручного доступу до вебресурсів. Вона поєднує переваги сучасних технологій захисту, зручність для кінцевого користувача та спрощення розробки для розробників. В умовах зростаючих вимог до кібербезпеки та UX, це рішення справедливо залишається одним із найкращих у своїй категорії.

Список використаних джерел

1. User authentication and ownership (steamworks documentation). *Steamworks*. URL: https://partner.steamgames.com/doc/features/auth (date of access: 12.05.2025).

2. OAuth libraries for python. OAuth Community Site. URL: https://oauth.net/code/python/ (date of access: 12.05.2025).