МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ЗАСОБАМИ PYTHON ТА ANGULAR»**

Виконав: студент 4 курсу, групи 6.1210-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
М.С. Кошелев
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н, Кудін О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, к.т.н. Добровольський Г.А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Кошелеву Микиті Сергійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка інформаційної системи засобами Python та Angular

керівник роботи Кудін Олексій Володимирович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка інформаційної системи на основі теоретичних відомостей.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	12.01.2024	
2.	Збір вихідних даних.	26.01.2024	
3.	Обробка методичних та теоретичних	14.02.2024	
	джерел.		
4.	Розробка першого та другого розділу.	15.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2024	

Студент _____ М.С. Кошелев
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.В. Кудін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка інформаційної системи засобами Python та Angular»: 49 с., 28 рис., 1 табл., 8 джерел, 1 додаток.

АУТЕНТИФІКАЦІЯ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ANGULAR, API, DJANGO, JWT, SQLITE.

Об’єкт дослідження – процеси проєктування, розробки та інтеграції вебдодатків, які забезпечують функціональність інформаційних систем для ігрових спільнот. Зокрема, дослідження зосереджується на використанні фреймворків Django для бекенд-розробки та Angular для фронтенд-розробки, а також на методах забезпечення безпеки, масштабованості та інтерактивності таких систем.

Мета роботи – розробка інформаційної системи для ігрової спільноти “Star Citizen” із використанням сучасних вебтехнологій Django та Angular. Ця система повинна забезпечити зручну платформу для організацію спільних заходів та подій, підвищення активності гравців, а також підтримку інтерактивної взаємодії гравців у режимі реального часу.

Метод дослідження – методи програмної інженерії.

SUMMARY

Bachelor’s qualifying paper “Development of an Information System Using Python and Angular”: 49 pages, 28 figures, 1 table, 8 references, 1 supplement.

AUTHENTICATION, CLIENT-SERVER ARCHITECTURE, ANGULAR, API, DJANGO, JWT, SQLITE.

The object of study is the processes of designing, developing, and integrating web applications that provide the functionality of information systems for gaming communities. In particular, the study focuses on the use of Django frameworks for backend development and Angular for frontend development, as well as methods for ensuring the security, scalability, and interactivity of such systems

The aim of the study is development of an information system for the Star Citizen gaming community using modern web technologies Django and Angular. This system should provide a convenient platform for organizing joint activities and events, increasing player activity, and supporting real-time interactive interaction between players.

The method of research is software engineering methods.

ЗМІСТ

ВСТУП

Інформаційні системи відіграють ключову роль у сучасному суспільстві, забезпечуючи автоматизацію процесів управління, обробку та збереження даних, а також комунікацію між користувачами. Сучасні ігрові спільноти потребують інструментів для організації ігрової діяльності, управління ресурсами та заохочення активності гравців. Створення ефективної інформаційної системи, яка б задовольняла ці потреби, є актуальною науковою та практичною задачею.

Розробка інформаційної системи для спільнот гравців, зокрема для гри Star Citizen, є актуальною через зростаючу популярність багатокористувацьких онлайн-ігор та необхідність ефективного управління ігровими подіями та ресурсами. Така система не лише спрощує організацію заходів та управління базою даних, але й сприяє підвищенню активності гравців шляхом запровадження системи заохочень.

Аналіз показує, що більшість інформаційних систем для ігрових спільнот базуються на ботах для Discord, через те, що ця платформа одразу ж надає можливості для спілкування, при цьому залишається проблема в зручності так, як взаємодії з ботами відбуваються через команди.

Метою даної роботи є розробка інформаційної системи засобами Python та Angular, яка забезпечить зручний інтерфейс для організації заходів, а також підвищення активності у спільноті

Для досягнення цієї мети необхідно вирішити наступні завдання:

- проаналізувати існуючі підходи до розробки інформаційних систем для ігрових спільнот;
- розробити структуру бази даних для зберігання інформації;
- реалізувати функціонал адміністрування заходів та управління базами даних;
- розробити систему заохочень для підвищення активності гравців.

Об’єктом дослідження є інформаційні системи для ігрової спільноти.

Предметом дослідження є методи та засоби розробки інформаційної.

У процесі дослідження використовувались методи аналізу та розробки інформаційних систем, проєктування баз даних, програмування засобами Python та Angular, а також методи тестування та інтеграції програмного забезпечення.

Робота складається з вступу, основної частини, яка включає розділи теоретичного аналізу, проєктування та реалізації інформаційної системи, а також висновків і рекомендацій щодо подальшого розвитку. У додатках наведені приклади коду та результати тестування системи.

1 ТЕОРЕТИЧНІ ОСНОВИ

1.1 Аналіз предметної області

Ігрові спільноти є невід'ємною частиною сучасної індустрії відеоігор, що об'єднують гравців навколо спільних інтересів та забезпечують платформу для взаємодії, обміну досвідом та організації спільних заходів. У контексті стрімкого розвитку цифрових технологій ігрові спільноти набули значного соціального та економічного значення, впливаючи на різні аспекти життя людей.

Ігрові спільноти формуються навколо конкретних відеоігор або жанрів ігор, де гравці можуть спілкуватися, обговорювати стратегії, ділитися досягненнями та створювати контент, пов'язаний з грою. Такі спільноти можуть існувати як у віртуальному, так і в реальному світі, об'єднуючи людей через форуми, соціальні мережі, платформи для стримінгу, а також на спеціалізованих заходах.

Основні риси ігрових спільнот включають:

- спільний інтерес: гравці об'єднуються навколо конкретної гри або жанру, що створює базу для спілкування та взаємодії;
- колективна діяльність: спільні ігрові сесії, обговорення стратегій та участь у турнірах сприяють активній взаємодії між учасниками спільноти;
- виробництво контенту: гравці створюють та поширюють власний контент, такий як відео, статті, гайди та модифікації до гри, що сприяє розвитку спільнот;
- підтримка та допомога: учасники спільноти надають один одному підтримку, допомагаючи вирішувати проблеми, пов'язані з грою, та навчаючи новачків.

Значення ігрових спільнот:

- формування соціальних зв’язків: ігрові спільноти сприяють формуванню нових соціальних зв’язків та дружніх відносин, що особливо важливо для людей, які можуть відчувати соціальну ізоляцію в реальному житті, через взаємодію у спільнотах гравці з різних країн та культур можуть обмінюватися знаннями та досвідом, сприяючи культурному взаєморозумінню;
- підтримка індустрії відеоігор: активні спільноти сприяють зростанню популярності ігор, що, в свою чергу, збільшує продажі та прибутки розробників;
- участь в ігрових спільнотах допомагає розвивати різні навички, включаючи комунікацію, командну роботу, стратегічне мислення та технічні здібності;
- гравці мають можливість виражати себе через створення контенту та участь у творчих проєктах, пов’язаних з грою.

У Star Citizen гравці можуть брати участь у космічних боях, досліджувати планети, виконувати місії, торгувати та розвивати свою кар’єру в різних галузях, таких як торгівля, дослідження, піратство або найманство, використовуючи різноманітні типи кораблів.

Центральною ідеєю цієї роботи є створення інструментарію, який би дозволяв адміністратору створювати оголошення про заходи в спільноті для збору гравців на активності, самі активності проходять в грі. Це робиться для більш зручного та структурованого збору гравців. Для підвищення активності гравців в нашій системі має бути система заохочень, яка повинна складатися з нагород, рейтингу і посад. А саме, адміністратор повинен мати можливість керувати базою даних нагород і посад, тобто які посади доступні гравцям і які нагороди вони можуть отримувати. Після завершення активності в грі і по її результатам, адміністратор видає нагороди гравцям, які брали участь. Коштовність отриманих нагород додається до рейтингу гравців, що підвищує дух суперництва, також при досягненні потрібної

кількості очков рейтингу, гравець отримує посаду.

Так, як Star Citizen гра про космос то невід’ємну роль в ній грають космічні кораблі, тому в нашій інформаційній системі має бути інструментарій, який би дозволяв оцінювати флот спільноти. Для цього адміністратор повинен мати змогу керувати базою даних кораблів. А користувачі в свою чергу додавати їх до себе в ангар, що буде виступати відображенням, які кораблі в них є в самій грі.

1.2 Аналіз технологій, Django

Django – це високорівневий фреймворк для швидкої розробки вебдодатків на мові програмування Python. Його архітектура ретельно спроєктована для забезпечення ефективної розробки та підтримки великих та складних вебпроєктів. Його основна мета – дозволити розробникам швидко створювати надійні та безпечні вебдодатки [2].

Основні компоненти Django включають моделі, представлення, шаблони, форми, URL маршрутизація, адмін інтерфейс.

Моделі відповідають за структуру даних додатку. Вони визначаються за допомогою класів Python, які успадковуються від django.db.models.Model. Вони відображаються в базу даних у вигляді таблиць [1].

Представлення відповідають за логіку обробки запитів та формування відповідей. Представлення отримує запит від клієнта, взаємодіє з моделями, та передає дані у шаблони для відображення. Є два основних типи представлень: функціональні та класові [1]. Функціональні представлення реалізуються у вигляді звичайних функцій Python. Вони прості в розумінні та використанні, особливо для невеликих додатків або простих задач. Класові представлення дозволяють організовувати логіку обробки запитів за допомогою класів. Це забезпечує кращу структурованість та можливість повторного використання коду. Функціональні представлення добре

підходять для простих задач та невеликих додатків, тоді як класові представлення забезпечують кращу структурованість та можливість повторного використання коду, що є особливо корисним для складних та великих проєктів. Вибір між функціональними та класовими представленнями залежить від конкретних потреб вашого додатку та особистих уподобань.

URL маршрутизація визначає, які функції будуть обробляти певні URL. Це здійснюється за допомогою urls.py файлів, де кожен URL пов’язується з відповідним представленням [1].

Такі компоненти як шаблони, форми і адміністративний інтерфейс не будуть використовуватись в цій роботі. Так, як Angular працює на стороні клієнта і всі шаблони Angular також обробляються на стороні клієнта. Використання шаблонів Django для рендерингу HTML на сервері не відповідає клієнтсько-серверній архітектурі, яку застосовує Angular. А адміністративний інтерфейс ми маємо реалізувати самотужки.

Django використовує архітектурний патерн MTV (Model-Template-View), який є варіацією MVC (Model-View-Controller) [1].

Модель відповідає за бізнес-логіку та доступ до даних. У Django це базовий клас, який представляє таблиці бази даних і дозволяє легко виконувати операції CRUD (Create, Read, Update, Delete) [2].

Шаблон відповідає за відображення даних користувачу. Шаблони визначають, як повинні виглядати HTML сторінки, які користувач бачить у браузері. Django надає багатий набір тегів та фільтрів для роботи з шаблонами [2].

Представлення відповідає за обробку запитів, взаємодію з моделями та передавання даних у шаблони. У представленнях описується логіка додатку, яка вирішує, які дані слід отримати з моделі та як їх слід відобразити [2].

Розглянемо Django у порівнянні з фреймворками, такими як Flask, Express.js та Ruby on Rails.

Django і Flask, виділимо головні особливості та різницю цих

фреймворків. Django підходить для складних та великих проєктів, оскільки має вбудований набір інструментів і готовий стек технологій. Flask легкий та мінімалістичний, ідеальний для швидкої розробки невеликих та середніх проєктів, де потрібна більша гнучкість.

Django і Express.js, виділимо головні особливості та різницю цих фреймворків. Django забезпечує структуровану архітектуру та повний стек технологій для розробки великих та складних вебдодатків. Express.js легкий та простий у використанні, підходить для створення мікросервісів та API, особливо для додатків, що потребують швидкого розгортання.

Django і Ruby on Rails виділимо головні особливості та різницю цих фреймворків. Django має широкий спектр можливостей для розробки складних вебдодатків, особливо в області науки та освіти, де Python є популярним. Ruby on Rails відмінний вибір для швидкої розробки прототипів та стартапів, з низкою функцій, які спрощують розробку.

Тож можемо вивести основні мінуси Django в порівнянні з іншими фреймворками. Повний стек функціональності може бути зайвим для невеликих проєктів, що може призвести до зайвого коду та складнощів у розумінні. Django може бути складним для освоєння через велику кількість вбудованих функцій та особливостей. Порівняно з більш мінімалістичними фреймворками, Django може бути менш гнучким у деяких сценаріях, де потрібен високий рівень настроювання.

1.3 Архітектура Angular та основні компоненти

Angular – це популярний фреймворк для створення вебдодатків, розроблений командою Google. Він базується на TypeScript і пропонує розширений набір інструментів для створення розширюваних і динамічних вебдодатків.

Angular додаток розділяється на модулі. Кожен модуль має свою

функціональність і може містити компоненти, сервіси, директиви тощо. Головний модуль додатку – AppModule, в якому визначається головний компонент, необхідні модулі та провайдери сервісів.

Компоненти – це будівельні блоки Angular додатків. Кожен компонент має свій шаблон (HTML), клас (TypeScript) та CSS-стилі.
Компоненти можуть взаємодіяти один з одним через властивості та події.

Сервіси використовуються для розділення логіки, яка не повинна бути прив’язана до конкретного компонента. Сервіси можуть бути використані для обробки даних, взаємодії з сервером, роботи з локальним сховищем тощо.

Angular має два типи директив: структурні та атрибутні. Структурні директиви, такі як *ngIf та *ngFor, дозволяють приймати рішення про те, які елементи DOM має створювати або видаляти компонент. Атрибутні директиви, такі як ngClass або ngStyle, змінюють вигляд або поведінку існуючих елементів DOM.

Роутінг в Angular дозволяє перехоплювати URL та відображати відповідні компоненти. Angular має вбудований модуль RouterModule, який дозволяє налаштовувати маршрути для додатку.

Angular надає можливості для роботи з реактивними та шаблонними формами. Реактивні форми базуються на реактивному програмуванні та використовують класи для представлення форм. Шаблонні форми використовують директиви в шаблоні для управління формами [3].

1.4 Висновки до розділу 1

У цьому розділі ми розглянули теоретичні основи та провели аналіз предметної області, що стосується розробки інформаційної системи для ігрової спільноти Star Citizen. Виокремили основні характеристики ігрових спільнот, їхнє значення та вплив на сучасне суспільство. Далі ми розглянули

архітектуру та основні компоненти фреймворків Django та Angular.

Django, як високорівневий фреймворк для розробки вебдодатків, пропонує багато вбудованих функцій та ефективні засоби для роботи з базою даних та обробки запиті.

Angular, у свою чергу, забезпечує потужні можливості для створення динамічних та інтерактивних вебдодатків.

Враховуючи особливості задачі та можливості фреймворків Django та Angular, планується розробити інформаційну систему, яка буде забезпечувати зручний та функціональний інтерфейс для гравців Star Citizen, сприяючи їхній взаємодії та розвитку спільноти.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Постановка задачі

Ігрові спільноти дають можливість, користувачам спілкуватися, обмінюватись досвідом та знаходити нових друзів. Наша задача покращити комунікацію між гравцями зробивши зручну на ефективну інформаційну систему, яка дозволить зробити ігрові активності більш централізованими та структурованими.

Для реалізації такої системи було обрано сучасний стек технологій, що включає Django, Angular та JWT (JSON Web Token) аутентифікацію. Django забезпечить надійний і швидкий бекенд, Angular – динамічний і зручний фронтенд, а JWT – безпечну аутентифікацію користувачів.

Для зберігання даних буде використано СУБД Django за замовчуванням, тобто SQLite.

SQLite – це легка, вбудована реляційна база даних, яка зберігає всю базу даних в одному файлі на диску. Вона не вимагає налаштування серверу бази даних і є чудовим вибором для невеликих проєктів, прототипів і додатків, які не потребують обробки великого обсягу даних або високої продуктивності під навантаженням [5].

SQLite вбудована безпосередньо у додаток, що означає відсутність необхідності налаштовувати або адмініструвати окремий сервер бази даних. Вся база даних зберігається в одному файлі, що робить її простою для розгортання і перенесення. SQLite має невеликий розмір і мінімальний вплив на ресурсну інтенсивність додатку. SQLite забезпечує підтримку транзакцій з дотриманням властивостей атомарності, консистентності, ізольованості та довговічності (ACID). SQLite підтримує основні функції SQL, включаючи запити, транзакції, індекси та тригери [6].

Основні етапи розробки включають:

- розробка серверної частини: використовуючи Django, ми створимо API для взаємодії з клієнтською частиною, забезпечимо управління користувачами, аутентифікацію та авторизацію за допомогою JWT, а також роботу з базою даних для зберігання інформації про користувачів та інші дані;
- розробка клієнтської частини: на базі Angular буде створено інтерфейс користувача, який забезпечить зручну взаємодію з системою, а також реалізуємо функціональність для відправки і отримання даних в режимі реального часу;
- інтеграція JWT для аутентифікації: ми забезпечимо безпечний обмін даними між клієнтом і сервером, використовуючи JWT для аутентифікації та авторизації користувачів, що дозволить захистити конфіденційну інформацію та забезпечити безпеку взаємодії.

Користуючись аналізом предметної області з розділу 1 можемо визначити функціонал який повинна надавати інформаційна система.

Першочергово потрібно реалізувати реєстрацію і авторизацію користувачів, а також розділення прав на адміністратора і звичайного користувача.

Система повинна надавати адміністратору можливості управління базою даних, а користувачу доступ до даних і взаємодію з ними. А саме, адміністратор контролює такі дані: кораблі, нагороди, ролі в подіях, оголощення про події, посади.

Система має рахувати рейтинг користувачів, їх посаду та загальну кількість кожного типу кораблів у користувачів.

Користувачу доступні такі дані: події, кораблі в гравців, рейтинг гравців, власні посада та нагороди.

Користувач може взаємодіяти з кораблями додаючи чи прибираючи з ангару, подавати заявки на події.

2.2 Створення ER-діаграми та діаграм прецедентів

ER-діаграма (Entity-Relationship Diagram, діаграма сутність-зв’язок) – це графічне відображення структури бази даних, яке використовується для моделювання і проєктування її логічної структури [8].

Основна мета ER-діаграми – візуалізувати сутності, атрибути цих сутностей та зв’язки між ними.

Основні компоненти ER-діаграми. Сутності представляють об’єкти або концепції, що мають значення для системи. В ER-діаграмі сутності зображаються у вигляді прямокутників. Атрибути – це характеристики або властивості сутностей. В ER-діаграмі атрибути зазвичай зображуються у вигляді овалів, з’єднаних з сутностями. Зв’язки визначають, як сутності взаємодіють одна з одною. В ER-діаграмі зв’язки зображуються у вигляді ромбів або ліній, що з’єднують сутності [8].

Типи зв’язків. Один до одного – кожній сутності А відповідає одна сутність В і навпаки. Один до багатьох – одній сутності А може відповідати багато сутностей В, але кожній сутності В відповідає лише одна сутність А. Багато до багатьох – кожній сутності А може відповідати багато сутностей В і навпаки [8].

ER-діаграми використовуються на етапах проєктування баз даних для виявлення і структурування всіх необхідних елементів та зв’язків між ними. Діаграми допомагають інженерам, аналітикам та іншим зацікавленим сторонам легко зрозуміти структуру бази даних та її логіку. ER-діаграми служать як документ, що описує структуру бази даних, що може бути корисним для підтримки, оновлення та розширення системи.

ER-діаграма для цього проєкту зображена на рисунку 2.1.

Розберемо зв’язки між сутностями:

– CustomerUser (абстрактне наслідування) від AbstractBaseUser, PermissionsMixin (CustomerUser успадковує властивості та методи від цих абстрактних класів);

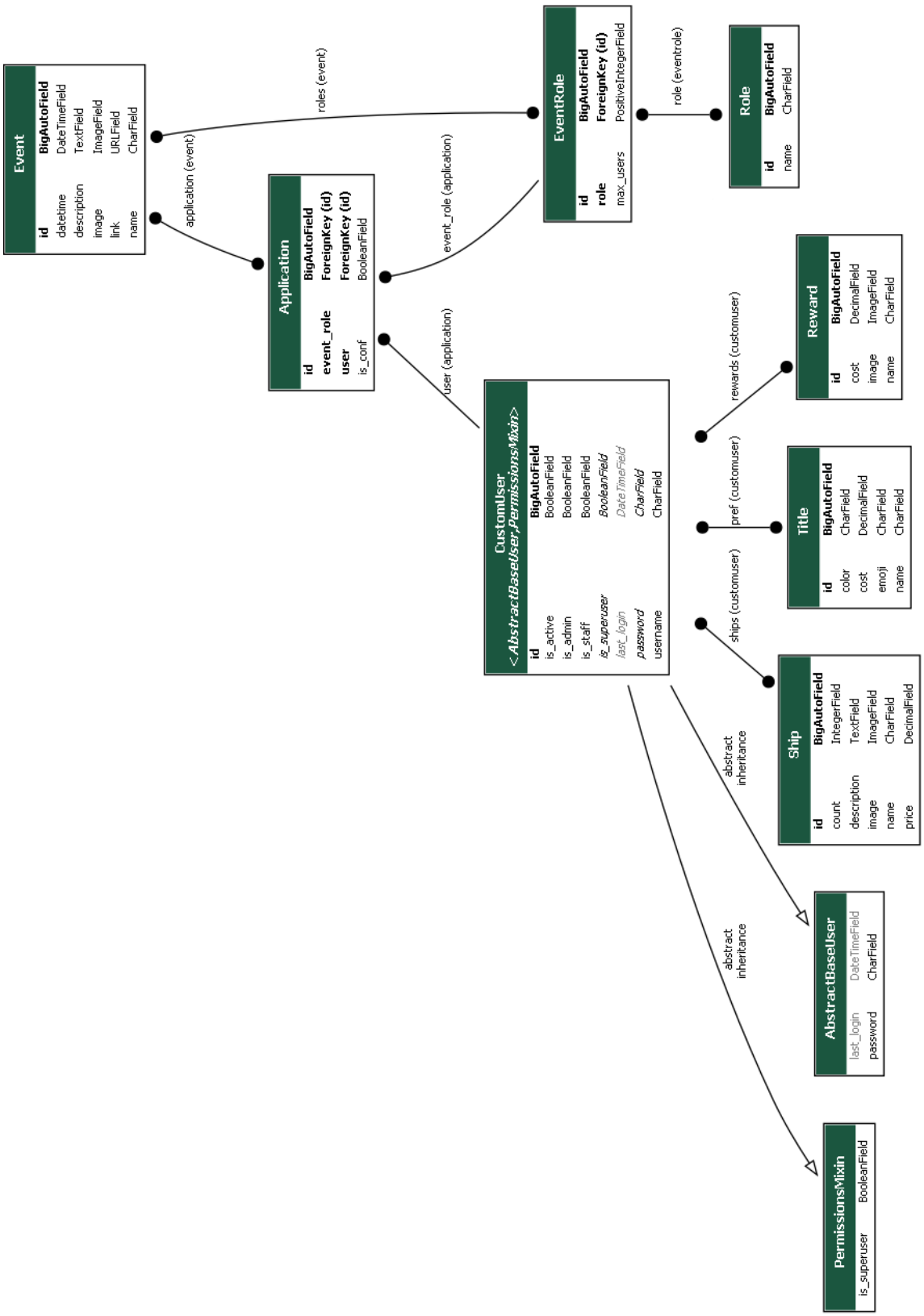

Рисунок 2.1 – ER-діаграма

- CustomerUser – Ship один користувач може мати багато кораблів і один корабель може належати багатьом користувачам (ManyToManyField);
- CustomerUser – Title один користувач може мати лише одну посаду, але одна посада може бути у багатьох користувачів (ManyToOne);
- CustomerUser – Reward один користувач може мати багато нагород і одна нагорода може належати багатьом користувачам (ManyToManyField);
- Event – Application одна подія може мати багато заявок (OneToMany);
- Event – EventRole одна подія може мати багато ролей (OneToMany);
- Application – CustomerUser заявка пов’язана з користувачем через ForeignKey;
- Application – EventRole заявка пов’язана з роллю у події через ForeignKey;
- EventRole – Role роль у події пов’язана з основною роллю через ForeignKey.

Розберемо сутності та їх атрибути в таблиці 1.1.

Таблиця 2.1 – Сутності та їх атрибути

Сутність	**Атрибут**	**Тип**	**Опис**
CustomerUser	id	Унікальний ідентифікатор	Унікальний ідентифікатор користувача
	is_active	Boolean	Чи активний користувач
	is_admin	Boolean	Чи є користувач адміністратором
	is_staff	Boolean	Чи є користувач персоналом

Продовження таблиці 2.1

Сутність	Атрибут	Тип	Опис
	is_superuser	Boolean	Чи є користувач суперкористувачем
	last_login	DateTime	Час останнього входу
	password	String	Пароль користувача
	username	String	Ім’я користувача
PermissionsMixin	is_superuser	Boolean	Чи є користувач суперкористувачем
AbstractBaseUser	last_login	DateTime	Час останнього входу
	password	String	Пароль користувача
Ship	id	Унікальний ідентифікатор	Унікальний ідентифікатор корабля
	count	Integer	Кількість кораблів у користувачів
	description	String	Опис корабля
	image	Image	Зображення корабля
	name	String	Назва корабля
	price	Decimal	Ціна корабля
Title	id	Унікальний ідентифікатор	Унікальний ідентифікатор посади
	color	String	Колір посади
	cost	Decimal	Ціна посади
	emoji	String	Емоджі посади
	name	String	Назва посади
Reward	id	Унікальний ідентифікатор	Унікальний ідентифікатор нагороди
	cost	Decimal	Вартість нагороди
	description	String	Опис нагороди

Продовження таблиці 2.1

Сутність	Атрибут	Тип	Опис
	image	Image	Зображення нагороди
	name	String	Назва нагороди
Event	id	Унікальний ідентифікатор	Унікальний ідентифікатор події
	datetime	DateTime	Дата і час початку події
	description	String	Опис події
	image	Image	Зображення оголошення події
	link	URL	Посилання на брифінг до події
	name	String	Назва події
Application	id	Унікальний ідентифікатор	Унікальний ідентифікатор заявки на участь
	event_role	ForeignKey	Роль у події (зовнішній ключ на EventRole)
	user	ForeignKey	Користувач, який подав заявку
	is_conf	Boolean	Статус підтвердження заявки
EventRole	id	Унікальний ідентифікатор	Унікальний ідентифікатор ролі у події
	role	ForeignKey	Роль (зовнішній ключ на Role)
	max_users	Integer	Максимальна кількість користувачів для цієї ролі у події

Продовження таблиці 2.1

Сутність	Атрибут	Тип	Опис
Role	id	Унікальний ідентифікатор	Унікальний ідентифікатор ролі
	name	String	Назва ролі

Діаграма прецедентів є частиною моделювання вимог до програмного забезпечення і використовується для візуалізації функціональних вимог системи. Вона показує взаємодію між користувачами і системою для досягнення конкретних цілей [7].

Основні компоненти діаграми прецедентів. Актори – це зовнішні по відношенню до системи об’єкти (користувачі або інші системи), що взаємодіють з нею. В діаграмі прецедентів актори зображуються у вигляді чоловічків або паличкових фігурок. Прецеденти описують дії або функціональність, яку система надає актору. Це конкретні випадки використання системи. В діаграмі прецедентів прецеденти зображуються у вигляді овалів з назвою всередині. Зв’язки показують взаємодію між акторами і прецедентами, а також між самими прецедентами. В діаграмі прецедентів зв’язки зображуються у вигляді ліній.

Призначення діаграми прецедентів. Допомагає ідентифікувати і документувати функціональні вимоги до системи, описуючи, що система має робити з точки зору користувача. Сприяє ефективному спілкуванню між розробниками, аналітиками та кінцевими користувачами системи, забезпечуючи спільне розуміння функціональних можливостей системи. Використовується для планування етапів розробки системи, допомагаючи визначити пріоритети в реалізації функціональних можливостей. Служить як частина документації системи, що описує, як і хто буде використовувати різні функціональні можливості системи [7].

Діаграма прецедентів для цього проєкту зображена на рисунку 2.2.

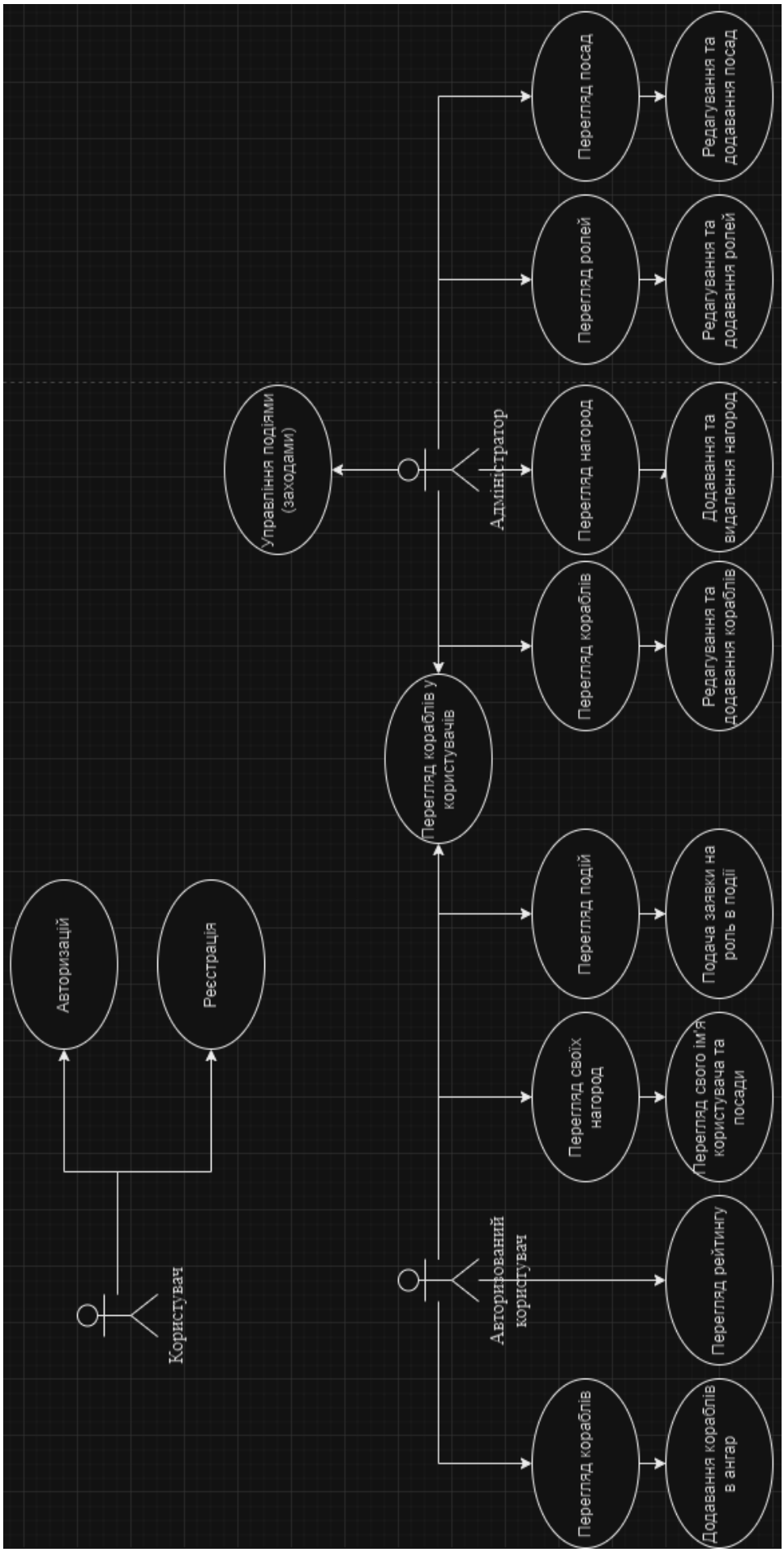

Рисунок 2.2 – Діаграма прецедентів

2.3 Висновки до розділу 2

В даному розділі було представлено проєктування програмного забезпечення для інформаційної системи ігрової спільноти, включаючи постановку задачі та основні етапи розробки. Зокрема, було зроблено ER-діаграму та діаграму прецедентів, що допомагають візуалізувати структуру системи та взаємодію між її компонентами.

Вибір стеку технологій, що включає Django, Angular та JWT (JSON Web Token), забезпечує надійність, безпеку та зручність розробки. Django обрано як бекенд-фреймворк завдяки його можливостям швидкої розробки, вбудованим компонентам для роботи з базами даних та високим стандартам безпеки. Angular використовується для створення динамічного інтерфейсу користувача, який дозволяє зручно обробляти та візуалізувати дані, отримані з бекенду. JWT забезпечує безпечну аутентифікацію та авторизацію користувачів, що є критично важливим для захисту конфіденційних даних.

Таким чином, проведене проєктування та розробка інформаційної системи для ігрової спільноти на основі Django, Angular та JWT дозволяють створити надійну, масштабовану та безпечну систему. Вибраний стек технологій забезпечує високу продуктивність та зручність використання як для розробників, так і для кінцевих користувачів, що сприяє ефективному спілкуванню та обміну досвідом в ігровій спільноті.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Розробка бекенд частини за допомогою Django

У цьому підрозділі ми детально розглянемо процес розробки бекенд частини інформаційної системи для ігрової спільноти, використовуючи фреймворк Django. Django є потужним та універсальним інструментом для створення вебдодатків, що дозволяє швидко розробляти надійні та масштабовані системи. Завдяки вбудованим засобам для роботи з базами даних, а також зручному механізму управління користувачами, Django забезпечує всі необхідні компоненти для реалізації бекенд функціоналу.

У цьому підрозділі ми опишемо структуру проєкту на Django, розглянемо процес налаштування середовища розробки, створення моделей даних, реалізації REST API з використанням Django REST Framework, а також впровадження аутентифікації на основі JSON Web Token (JWT).

Особливу увагу буде приділено реалізації аутентифікації за допомогою JWT, оскільки цей метод забезпечує високий рівень безпеки та зручності для користувачів. JWT є популярним стандартом для забезпечення безпечного обміну інформацією між клієнтом та сервером, і його інтеграція в проєкт допоможе створити надійну систему управління сесіями та правами доступу.

Цей підрозділ служитиме основою для подальшої інтеграції з фронтенд частиною, яка буде розглянута в наступних розділах. Знання, отримані на цьому етапі, дозволять краще зрозуміти загальну архітектуру системи та забезпечити її ефективну роботу.

3.1.1 Реалізація моделей даних

Моделі даних представляють структуру та зв’язки інформації в нашій системі, дозволяючи ефективно зберігати та керувати даними в базі даних.

Використання моделей забезпечує об’єктно-орієнтований підхід до роботи з базою даних, що спрощує розробку та підтримку системи.

База даних у нашій системі використовується для зберігання інформації про користувачів, події, заявки, ролі, кораблі та інші сутності. Django ORM дозволяє нам визначати структуру цих даних за допомогою моделей. Моделі є центральною частиною додатку на Django, оскільки вони визначають структуру таблиць в базі даних, типи даних, зв’язки між таблицями та інші аспекти, що стосуються зберігання інформації.

Центральними моделями в цьому проєкті є моделі CustomUser та Event.

CustomUser використовується для зберігання інформації про користувачів системи. А саме чи є користувач адміністратором, його ім’я користувача та пароль, а також зв’язки з іншими моделями, які описують посаду користувача, його нагороди, кораблі, заявки на події, за це відповідають моделі Reward, Ship, Title і Application.

Event містить дані про події такі, як назва, опис, дата і час початку, зображення, посилання на брифінг.

Кожна подія має ролі для користувачів, за це відповідають моделі Role та EventRole, модель Role містить лише назву ролі і зберігає всі ролі доступні при створенні події.

Модель EventRole, посилається на Role та містить максимальну кількість користувачів, які можуть бути затверджені на роль.

При створенні події адміністратор вказує, які ролі мають бути та максимальну кількість користувачів для них, при створенні обьекта Event створюються і EventRole для нього.

За заявки на роль в події відповідає Application, яка містить дані користувача, роль в події та чи затверджена вона.

Приклади реалізації моделей CustomUser і Event на ведено на рисунку 3.1 та рисунку 3.2.

```python
class CustomUser(AbstractBaseUser, PermissionsMixin):
    username = models.CharField(max_length=15, unique=True)
    ships = models.ManyToManyField(Ship, blank=True)
    rewards = models.ManyToManyField(Reward, blank=True)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    is_admin = models.BooleanField(default=False)
    pref = models.ManyToManyField(Title, blank=True)
    objects = ShipUserManager()
    avatar = models.URLField(max_length=200, blank=True)
    USERNAME_FIELD = "username"
    REQUIRED_FIELDS = []

    def __str__(self):
        return self.username
```

Рисунок 3.1 – Реалізація моделі CustomUser

```python
class Event(models.Model):
    name = models.CharField(max_length=100)
    datetime = models.DateTimeField()
    description = models.TextField()
    image = models.ImageField(upload_to="events")
    link = models.URLField(max_length=200)
    roles = models.ManyToManyField(EventRole, blank=True)
    application = models.ManyToManyField(Application, blank=True)

    def __str__(self):
        return self.name
```

Рисунок 3.2 – Реалізація моделі Event

3.1.2 Обробка подій, що відбуваються в моделях

У нашому проєкті також використовується механізм сигналів (signals) у Django для обробки подій, які відбуваються в моделях. Сигнали дозволяють виконувати певні дії автоматично при виникненні певних подій, таких як створення, оновлення або видалення об’єктів.

Механізм signals з використанням функції receiver дозволяє обробляти події, що виникають у моделях. Це може бути корисним для виконання додаткових дій, таких як надсилання повідомлень, оновлення пов’язаних даних або логування.

В данному проєкті функція receiver використовується для автоматичної видачі або позбавлення посади у користувача, при додаванні або видаленні

нагороди у нього (рис. 3.3), а також підрахування кількості кораблів при додаванні або видаленні корабля в ангар користувача.

```
@receiver(m2m_changed, sender=ShipUser.rewards.through)
def update_user_pref(sender, instance, **kwargs):
    if kwargs["action"] == "post_add" or kwargs["action"] == "post_remove":
        total_rewards_cost = instance.rewards.aggregate(total_cost=models.Sum("cost"))["total_cost"]
        if total_rewards_cost is None:
            total_rewards_cost = 0
        titles = Title.objects.order_by("-cost")
        for title in titles:
            if total_rewards_cost >= title.cost:
                instance.pref.clear()
                instance.pref.add(title)
                break
        else:
            instance.pref.set("")
            instance.save()
```

Рисунок 3.3 – Реалізація сигналу видачі посад

3.1.3 Реалізація аутентифікації за допомогою JWT

У цьому підрозділі ми розглянемо процес реалізації аутентифікації у нашій системі за допомогою JWT (JSON Web Token). JWT є популярним методом аутентифікації, який забезпечує безпечний спосіб передачі інформації між клієнтом і сервером.

JWT – це компактний, самодостатній формат токену, який використовується для передачі інформації між сторонами як JSON-об’єкт. Він складається з трьох частин: заголовок (header), корисне навантаження (payload), та підпис (signature) [4].

Для реалізації JWT аутентифікації ми будемо використовувати бібліотеку “djangorestframework-simplejwt”. У файлі settings.py ми додаємо налаштування для “djangorestframework-simplejwt” та налаштовуємо REST Framework для використання JWT-аутентифікації.

Коли клієнт надсилає запит на отримання токену, він надає свої облікові дані (ім’я користувача та пароль). Сервер перевіряє ці дані та повертає JWT токен у разі успішної аутентифікації. Цей токен

використовується для подальших запитів до захищених ендпоінтів API, надаючи серверу можливість ідентифікувати та аутентифікувати користувача.

3.1.4 Створення REST API з використанням Django REST Framework

У цьому підрозділі ми розглянемо процес створення REST API для нашої інформаційної системи за допомогою Django REST Framework (DRF). REST API (Representational State Transfer Application Programming Interface) дозволяє взаємодіяти з нашою системою через HTTP-запити, що робить її доступною для клієнтських додатків, таких як веббраузери або мобільні додатки.

Django REST Framework (DRF) забезпечує потужні засоби для створення API, включаючи серіалізацію даних, дозволи та обробку запитів.

Розглянемо основні компоненти DRF. Серіалізатори перетворюють складні типи даних, такі як запити та моделі Django, у формати, які легко рендеряться у JSON, XML тощо. Представленння обробляють HTTP-запити та повертають відповідні HTTP-відповіді. Маршрути автоматично генерують URL-шляхи для різних ендпоінтів API.

Серіалізатори відповідають за перетворення даних моделі у JSON-формат та навпаки. Вони визначають, які поля з моделі будуть включені в API-відповідь. Приклад серіалізатора в проєкті наведено на рисунку 3.4.

```
class EventSerializer(serializers.ModelSerializer):
    application = ApplicationSerializer(many=True, read_only=True)
    roles = EventRoleSerializer(many=True, read_only=True)

    class Meta:
        model = Event
        fields = "__all__"
```

Рисунок 3.4 – Серіалізатор для моделі Event

Permissions (дозволи) у Django визначають, хто має доступ до певних ресурсів або може виконувати певні дії в системі. Вони є важливою частиною забезпечення безпеки та контролю доступу в додатках [2].

У Django дозволи можуть бути реалізовані на декількох рівнях: модельному, представлень (views), та на рівні API з використанням Django REST Framework (DRF) [2].

На рисунку 3.5 наведено дозвіл який перевіряє чи є користувач авторизованим та чи є в нього права адміністратора.

```
class IsAdmin(BasePermission):

    def has_permission(self, request, view):
        return request.user and request.user.is_authenticated and request.user.is_admin
```

Рисунок 3.5 – Реалізація дозволу адміністратора

Представлення обробляють HTTP-запити та використовують серіалізатори для перетворення даних. DRF надає різні класи представлень, такі як APIView, GenericAPIView, та ViewSets, які дозволяють автоматично генерувати CRUD (Create, Read, Update, Delete) операції [1].

На рисунку 3.6 маємо реалізацію представлення для адміністрування бази даних кораблів.

Маршрутизатори генерують URL-шляхи для API автоматично на основі визначених ViewSet. Це спрощує процес визначення ендпоінтів API (див. рис. 3.7).

Створення REST API з використанням Django REST Framework дозволяє забезпечити стандартизований та зручний спосіб взаємодії з нашою системою.

Використання серіалізаторів, представлень та маршрутизаторів спрощує процес реалізації API та робить його більш структурованим та зрозумілим.

```python
class ShipControlView(APIView):
    parser_classes = (MultiPartParser, FormParser, JSONParser)
    permission_classes = [IsAdmin]
    serializer_class = ShipSerializer

    def get(self, request, format=None):
        ships = Ship.objects.all()
        serializer = ShipSerializer(ships, many=True)
        return Response(serializer.data)

    def post(self, request, format=None):
        serializer = ShipSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(status=status.HTTP_200_OK)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    def delete(self, request, format=None):
        ship = Ship.objects.get(id=request.data.get("id"))
        ship.delete()
        return Response(status=status.HTTP_200_OK)

    def put(self, request, format=None):

        ship = Ship.objects.get(id=request.data.get("id"))
        ship_data = request.data
        ship_data["image"] = ship.image
        serializer = ShipSerializer(ship, data=ship_data)
        if serializer.is_valid():
            serializer.save()
            return Response(status=status.HTTP_200_OK)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Рисунок 3.6 – Реалізація представлення

```python
urlpatterns = [
    path("admin/ships/", views.ShipControlView.as_view(), name="ship_control"),
    path("admin/events/", views.AdminEventsView.as_view(), name="event_control"),
    path("admin/users/", views.ShowUsers.as_view(), name="show_users"),
    path("admin/rewards/", views.RewardsView.as_view(), name="rewards"),
    path("admin/add_user_reward/", views.UsersAndRewars.as_view(), name="add_user_reward"),
    path("admin/title/", views.AdminTitleView.as_view(), name="title_control"),
    path("admin/roles/", views.AdminRoleView.as_view(), name="roles"),
    path("admin/application_confirm", views.ConfirmApplicationView.as_view(), name="confirm_application"),
    path("admin/users_and_rewards/", views.RewardIdView.as_view(), name="users_and_rewards"),
    path("user_rewards/", views.UserRewardsView.as_view(), name="user_rewards"),
    path("events/", views.EventsView.as_view(), name="events"),
    path("ships/", views.ShipsView.as_view(), name="ships"),
    path("rating/", views.RatingView.as_view(), name="rating"),
    path("application/", views.UserApplicationView.as_view(), name="application"),
    path("token/refresh/", views.CustomTokenRefreshView.as_view(), name="token_refresh"),
    path("login/", views.CustomTokenObtainPairView.as_view(), name="login"),
    path("register/", views.RegisterUserView.as_view(), name="register"),
    path("is_admin/", views.IsAdminView.as_view(), name="is_admin"),
    path("user/", views.CurrentUserView.as_view(), name="user"),
]
```

Рисунок 3.7 – Демонстрація маршрутів

3.2 Розробка фронтенд частини за допомогою Angular

У цьому підрозділі ми детально розглянемо процес розробки фронтенд частини інформаційної системи для ігрової спільноти, використовуючи фреймворк Angular. Angular є потужним та універсальним інструментом для створення динамічних односторінкових додатків (SPA), що дозволяє легко розробляти інтерактивні та високоефективні користувацькі інтерфейси. Завдяки вбудованим засобам для роботи з компонентами, сервісами та модулями, Angular забезпечує всі необхідні компоненти для реалізації фронтенд функціоналу.

У цьому підрозділі ми опишемо структуру проєкту на Angular, розглянемо процес налаштування середовища розробки, створення компонентів та сервісів, а також інтеграцію з бекенд частиною системи через REST API.

3.2.1 Створення та структура проєкту

Angular є потужним фреймворком для створення клієнтських додатків, що базуються на TypeScript. Для успішної розробки та підтримки Angular-проєкту важливо розуміти його структуру та компоненти. У цьому підрозділі ми розглянемо основні теки та файли, які створюються при ініціалізації нового Angular-проєкту, і їх призначення.

Після створення нового Angular-проєкту за допомогою Angular CLI команда “ng new project-name”, ми отримуємо наступну структуру каталогів:

- node_modules/: тека, що містить усі залежності проєкту, встановлені за допомогою npm;
- src/: основна тека з вихідним кодом проєкту;
- app/: містить компоненти, модулі, сервіси та інші файли, що складають додаток;

- assets/: містить статичні файли, такі як зображення, шрифти та інші ресурси;
- environments/: містить файли конфігурації для різних середовищ (наприклад, development, production);
- browserslist: визначає, які браузери підтримуються додатком;
- favicon.ico: значок додатка, що відображається у вкладці браузера;
- index.html: основний HTML-файл, у який вбудовується додаток Angular;
- main.ts: головний файл, що запускає Angular-додаток;
- polyfills.ts: включає полифіли для підтримки старих браузерів;
- styles.css: глобальні стилі для додатка;
- test.ts: налаштування для запуску тестів;
- tsconfig.app.json: конфігурація TypeScript для додатка;
- .editorconfig: конфігураційний файл для налаштування редактора коду;
- .gitignore: файл, що визначає, які файли та теки слід ігнорувати при використанні системи контролю версій Git;
- angular.json: основний конфігураційний файл Angular CLI для налаштування проєкту та його збірки;
- package.json: файл, що містить інформацію про проєкт та залежності;
- README.md: документація проєкту;
- tsconfig.json: головний конфігураційний файл TypeScript.

В теці “app” створюємо необхідні нам теки для більш зручної розробки і масштабованості проєкту, а саме:

- services, що відповідає за сервіси в проєкті;
- interfaces, що відповідає за інтерфейси в проєкті;
- buttons, зберігає компоненти універсальних кнопок;
- використовуються багато разів в проєкті;

– admin, тека з сторінками доступними лише адміністраторам;
– client, тека з сторінками доступними всім.

3.2.2 Створення сервісів та універсальних компонентів

Сервіс ApiService відіграє ключову роль у забезпеченні взаємодії між клієнтською частиною та сервером. Цей сервіс відповідає за виконання HTTP-запитів до бекенду, що дозволяє фронтенд додатку отримувати дані, надсилати форми та взаємодіяти з API, створеним за допомогою Django.

ApiService забезпечує єдиний інтерфейс для виконання HTTP-запитів (GET, POST, PUT, DELETE) до бекенд API. Це спрощує управління запитами та покращує читабельність коду.

Сервіс додає токен аутентифікації (JWT) до заголовків запитів, що дозволяє здійснювати захищену взаємодію з сервером. Токен отримується з локального сховища, де він зберігається після успішної авторизації користувача.

ApiService використовує оператор catchError з бібліотеки RxJS для обробки помилок HTTP-запитів (див. рис. 3.8). Це дозволяє централізовано обробляти помилки, що виникають при взаємодії з API.

Універсальні кнопки відповідають за POST, PUT та DELETE запити при написканні та використовують ApiService для цього.

Розглянемо їх на прикладі “AddButtonComponent”, що відповідає за POST запити. Компонент надає кнопку для взаємодії з користувачем. При натисканні на кнопку виконується HTTP-запит до бекенд-сервера.

Використовуючи ApiService, компонент виконує POST-запит для додавання даних до сервера. Компонент генерує події (add та fail), що дозволяє батьківському компоненту реагувати на успішне або невдале додавання даних. А за допомогою властивості disabled батьківський компонент може керувати коли кнопка повинна бути активна. Код для цієї кнопки зображений на рисунку 3.9.

```
switch (method) {
  case 'GET':
    requestObservable = this.http.get(url, { headers });
    break;
  case 'POST':
    requestObservable = this.http.post(url, data, { headers });
    break;
  case 'PUT':
    requestObservable = this.http.put(url, data, { headers });
    break;
  case 'DELETE':
    requestObservable = this.http.delete(url, { headers, body: data});
    break;
  default:
    throw new Error('Unsupported HTTP method');
}

return requestObservable.pipe(
  catchError(error => {
    return throwError(error.error);
  })
);
```

Рисунок 3.8 – Центральний метод ApiService

```
export class AddButtonComponent implements AddButtonProps {
  @Input() text: string = 'Додати';
  @Input() data: any;
  @Input() endpoint: string = '';
  @Input() disabled: boolean = false;
  @Output() add = new EventEmitter<void>();
  @Output() fail = new EventEmitter<void>();

  constructor(private apiService: ApiService) {}

  async handleAdd(): Promise<void> {
    try {
      const response = await this.apiService.request(this.endpoint, 'POST', this.data).toPromise();
      this.add.emit();
    } catch (error) {
      this.fail.emit();
    }
  }

}
```

Рисунок 3.9 – Універсальна кнопка для POST запитів

3.2.3 Інтерфейс адміністратора

Більшість сторінок в адміністративному інтерфейсі складаються з трьох основних компонентів. Таблиця, що відображає данні, форма для додавання даних та меню адміністратора.

Меню адміністратора складається з посилань на сторінки доступні лише адміністратору та деякі сторінки доступні усім, які можуть знадобитися для адміністратора. Меню наведено на рисунку 3.10.

```
<nav>
  <a routerLink="/admin/ships">Кораблі</a>
  <a routerLink="/admin/rewards">Нагороди</a>
  <a routerLink="/admin/roles">Ролі</a>
  <a routerLink="/admin/events">Заходи</a>
  <a routerLink="/admin/positions">Посади</a>
  <a routerLink="/ships">Кораблі в гравців</a>
</nav>
```

Рисунок 3.10 – Меню адміністратора

Форми для додавання даних містяться на всіх сторінках, які дозволяють керувати базою даних, код такої форми на прикладі додавання ролей зображено на рисунку 3.11 для TypeScript коду і рисунку 3.12 для HTML.

```
async handleSubmit(): Promise<void> {
  if (this.roleForm.invalid) {
    this.error = 'Заповніть всі поля';
    return;
  }

  const formData = new FormData();
  formData.append('name', this.roleForm.value.name);

  this.apiService.request('admin/roles/', 'POST', formData).subscribe(
    response => {
      this.onSuccess.emit();
    },
    error => {
      if (error instanceof ProgressEvent && error.type === 'error') {
        this.error = 'Помилка з`єднання з сервером, спробуйте пізніше';
        return;
      }
      this.error = Object.values(error)[0];
      if (this.error === 'Given token not valid for any token type') {
        this.error = 'Помилка з`єднання з сервером, спробуйте пізніше';
        return;
      }
      this.errorField = Object.keys(error)[0];
      this.error = this.error[0];
      if (this.error.includes('no more than')) {
        let max_chars = this.error.match(/\d+/);
        max_chars = parseInt(max_chars[0], 10);
        this.error = this.localizeService.maxCharsError(max_chars);
```

Рисунок 3.11

```
<form [formGroup]="roleForm" (ngSubmit)="handleSubmit()">
  <span>{{error}}</span>
  <input type="text" placeholder="Назва" formControlName="name" [ngClass]="{ 'error': errorField === 'name' }" required />
  <button class="cancel-button" type="button" (click)="handleClose()">Відміна</button>
  <button class="submit-button" type="submit">Додати</button>
</form>
```

Рисунок 3.12

На рисунку 3.13 зображено повний код сторінки адміністратора для управління ролями.

```
<div class="page-content">
<app-admin-menu></app-admin-menu>
<div class="main-content">
    <div class="search-and-add">
        <div></div>
        <button *ngIf="!showAddRoleForm" (click)="showAddRoleForm = true">Додати роль</button>
        <div *ngIf="showAddRoleForm">
        <app-add-role-form
            (onSuccess)="handleAddSuccess()"
            (onCancel)="handleAddCancel()">
        </app-add-role-form>
        </div>
    </div>
    <table>
        <thead>
            <tr>
                <th style="width: 80vh;">Назва</th>
                <th style="width: 20vh;">Дії</th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor="let role of roles">
                <td>
                    <input *ngIf="editedRoleId === role.id.toString()" type="text" [(ngModel)]="tempRoleChanges.name">
                    <ng-container *ngIf="editedRoleId !== role.id.toString()">
                        {{ role.name }}
                    </ng-container>
                </td>
                <td>
                    <ng-container *ngIf="editedRoleId === role.id.toString(); else editButtons">
                        <app-put-button
                            [data]="tempRoleChanges"
                            endpoint="admin/roles/"
                            (put)="confirmEdit()"
                            (fail)="handleAction()">
                        </app-put-button>
                        <button class="edit-button-cancel" (click)="cancelEdit()">Відмінити</button>
                    </ng-container>
                    <ng-template #editButtons>
                        <button class="edit-button" (click)="editRole(role)">Змінити</button>
                        <app-delete-button
                            [id]="role.id.toString()"
                            endpoint="admin/roles/"
                            (delete)="handleAction()"
                            (fail)="handleAction()">
                        </app-delete-button>
                    </ng-template>
                </td>
            </tr>
    </table>
</div>
```

Рисунок 3.13 – Сторінка управління ролями

3.2.4 Користуватський інтерфейс

Сторінки для звичний користувачів також мають своє меню (рис. 3.14).

```
<nav>
    <a routerLink="/events">Заходи</a>
    <a routerLink="/ships">Кораблі в гравців</a>
    <a routerLink="/rating">Рейтинг</a>
    <a routerLink="/hangar">Ангар</a>
    <a routerLink="/rewards">Мої нагороди</a>
</nav>
```

Рисунок 3.14 – Меню користувача

Центральною сторінкою для користувачі виступає сторінка з оголошеннями про події. На рисунку 3.15 зображено TypeScript код для неї, а на рисунку 3.16 HTML код.

```
events: Event[] = [];
appliedEvents: [string] = [""];
constructor(private tokenExpirationCheck: TokenExpirationCheck, private apiService: ApiService, private router: Router) { }
fetchData() {
  this.apiService.request("is_admin/", "GET").subscribe(
    (response) => {
      this.router.navigate(['/admin/events']);
    }
  );
  this.apiService.request("events/", "GET").subscribe(
    (response) => {
      this.events = response.all_events;
      this.appliedEvents = response.applied_events.map((event: any) => event.id);
    },
    (error) => {
      console.log(error);
    }
  );
}
isEventStarted(event: Event) {
  const event_date_utc = new Date(event.datetime);
  const event_date_local = new Date(event_date_utc.getTime() + event_date_utc.getTimezoneOffset() * 60000);
  const now = new Date();
  return event_date_local >= now;
}
localDate(date: string) {
  const date_utc = new Date(date);
  const date_local = new Date(date_utc.getTime() + date_utc.getTimezoneOffset() * 60000);
  return new Date(date_local).toLocaleString();
}
handleAction() {
  this.fetchData();
}

isApplied(event: Event) {
  return this.appliedEvents.includes(event.id);
}
isRoleFull(event: Event, roleId: string) {
  const appliedRoles = event.application.filter(application => application.event_role === roleId && application.is_conf === true);
  console.log(appliedRoles);
  const role = event.roles.find(role => role.id === roleId);
  if (!role) {
    return false;
  }
  return appliedRoles.length >= role.max_users;
}
```

Рисунок 3.15 – TypeScript код сторінки з подіями

```
<div class="page-content">
<app-user-menu></app-user-menu>
<div class="main-content">
    <div class="event" *ngFor="let event of events">
        <div class="main-info">
            <h2>{{event.name}}</h2>
            <h3>{{ localDate(event.datetime) }}</h3>
        </div>
        <p class="description">{{event.description}}</p>
        <img [src]="MEDIA_URL + event.image" alt="event_image" />
        <div class="test">
            <h3 *ngIf="!isApplied(event)">Ролі:</h3>
            <a href={{event.link}} target="_blank">Брифінг</a>
        </div>
        <div *ngIf="isEventStarted(event)" class="roles-container">
            <div *ngFor="let role of event.roles">
                <div *ngIf="!isApplied(event)">
                    <app-add-button [disabled]="isRoleFull(event, role.id)" text="{{role.role}}" *ngIf="!isApplied(event)" (add)="handleAction()"
                </div>
            </div>
            <h4 *ngIf="isApplied(event)">Ви вже подали заявку на цю подію</h4>
        </div>
    </div>
</div>
</div>
```

Рисунок 3.16 – HTML код сторінки з подіями

3.3 Функціональне тестування

В цьому підрозділі буде перевірено та протестовано чи всі функції відповідають вимогам до інформаційної системи.

Під час реєстрації або авторизації користувачі перевірка відповідність полів виконується як на стороні клієнта так і на стороні сервера (рис. 3.17). А саме відповідність типу значень та чи пусті поля, перевіряється на стороні клієнта, а відповідність самих даних на стороні сервера, наприклад, що ім’я користувача вже зайняте.

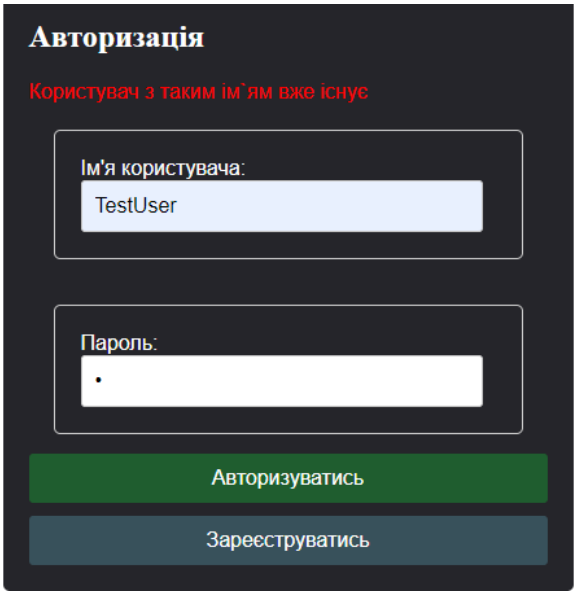

Рисунок 3.17 – Вікно авторизації та реєстрації

Так як центральною ідеєю є створення оголошень про заходи в ігровій спільноті. То домашньою сторінкою, на яку потрапляє користувач, є сторінка з подіями, для звичайного користувача, і сторінка з управлінням подіями для адміністратора. Якщо користувач спробує перейти на сторінку доступну лише адміністратору, чого не передбачає інтерфейс, але можна зробити перейшовши за посиланням, то його буде переадресовано на домашню сторінку так, як сервер відмовить в доступі. Протестуємо можливості створення та управління подіями з боку адміністратора (рис. 3.18), та перегляд і подачу заявок з боку користувача (рис. 3.19).

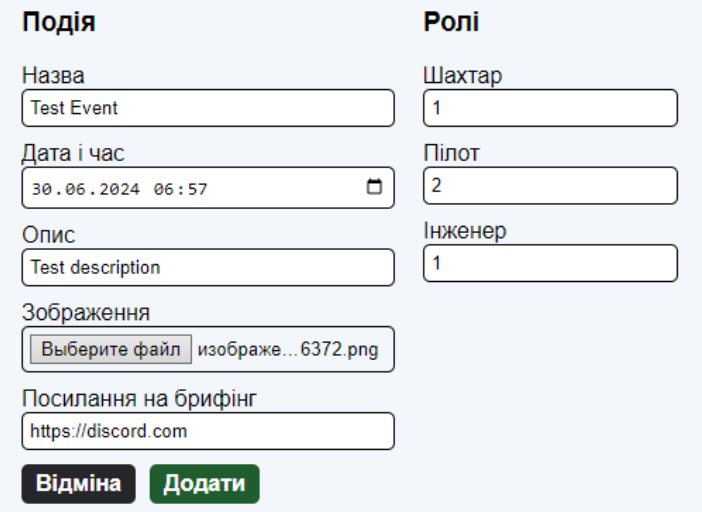

Рисунок 3.18 – Форма для створення оголошень про події

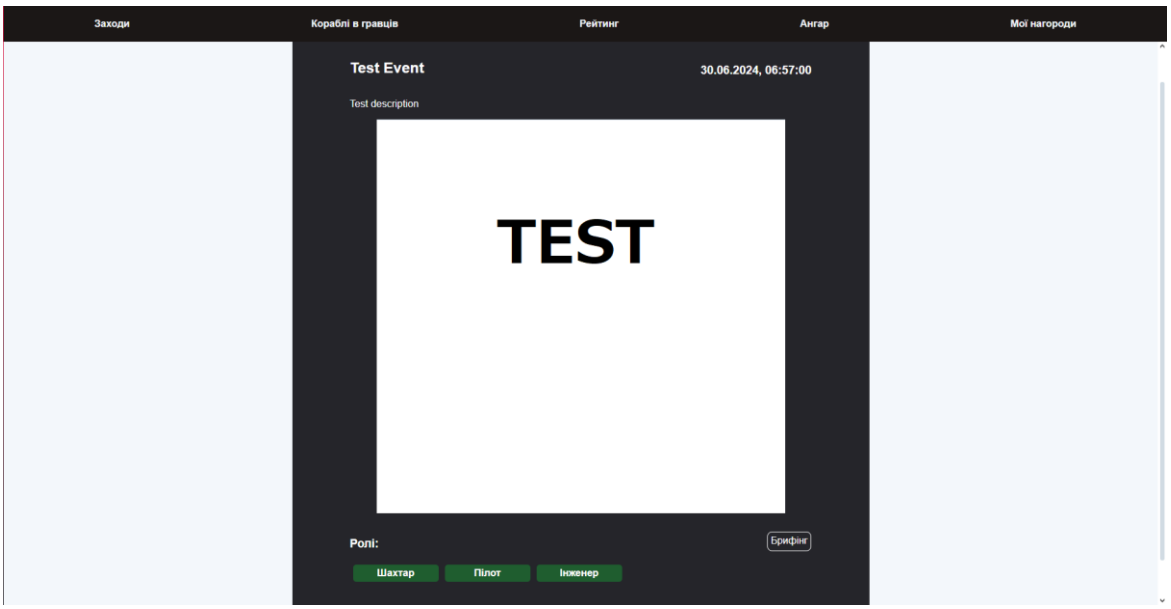

Рисунок 3.19 – Вигляд оголошення з боку користувача

Після подачі користувачем заявки на роль адміністратор може затвердити його на роль (рис. 3.20), а після завершення події видати нагороди (рис. 3.21).

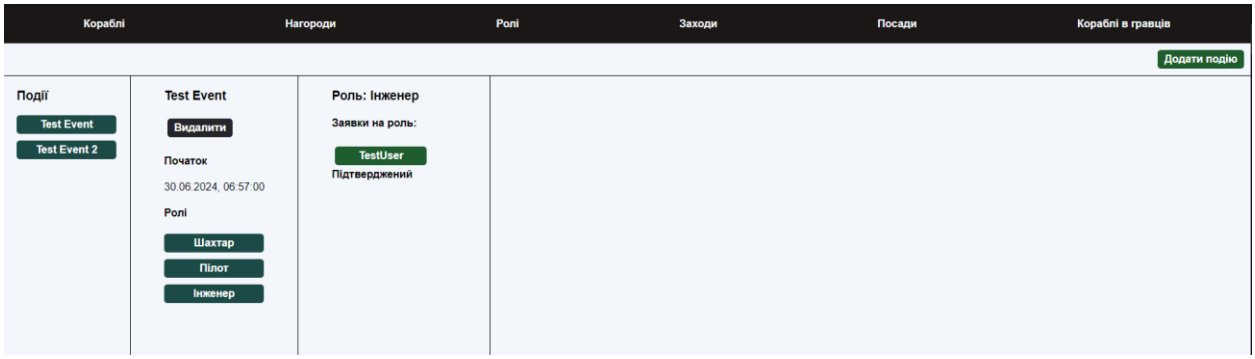

Рисунок 3.20 – Затвердження користувача

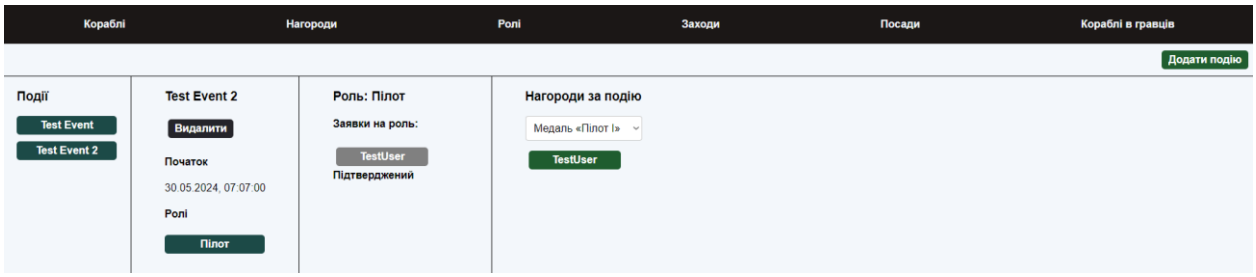

Рисунок 3.21 – Нагородження після завершення події

Після початку події затвердження чи зняття користувача з ролі стає недоступним.

При отриманні нагороди в користувача вона з’являється на сторінці «Мої нагороди» (рис. 3.22).

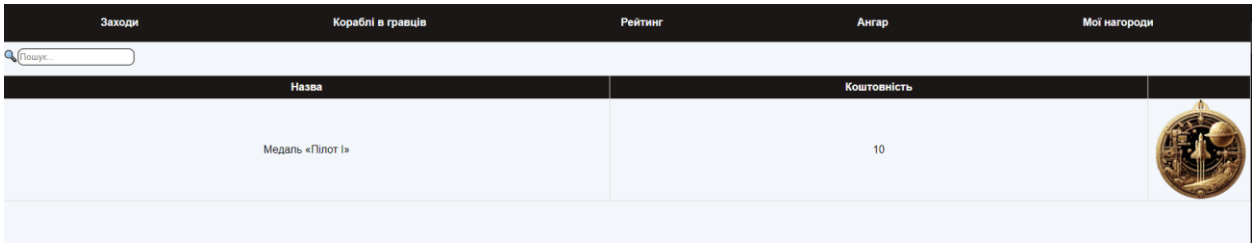

Рисунок 3.22 – Нагороди користувача

Коштовність нагороди додається в рейтинг (див. рис. 3.23).

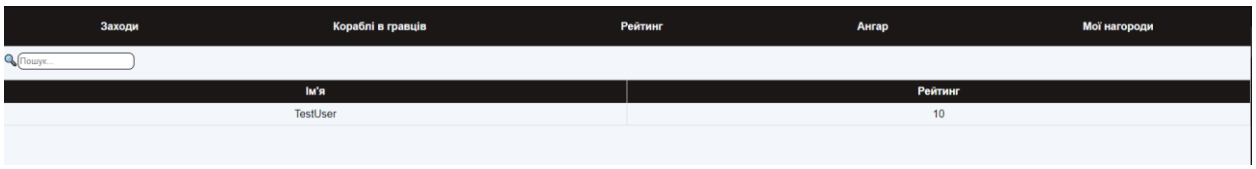

Рисунок 3.23 – Рейтинг

Посади видаються автоматично при досягненні потрібної кількості очок рейтингу (рис. 3.24).

Рисунок 3.24 – Ім’я користувача з виданою посадою

Адміністратор керує базою даних, зокрема базою даних кораблів (рис. 3.25).

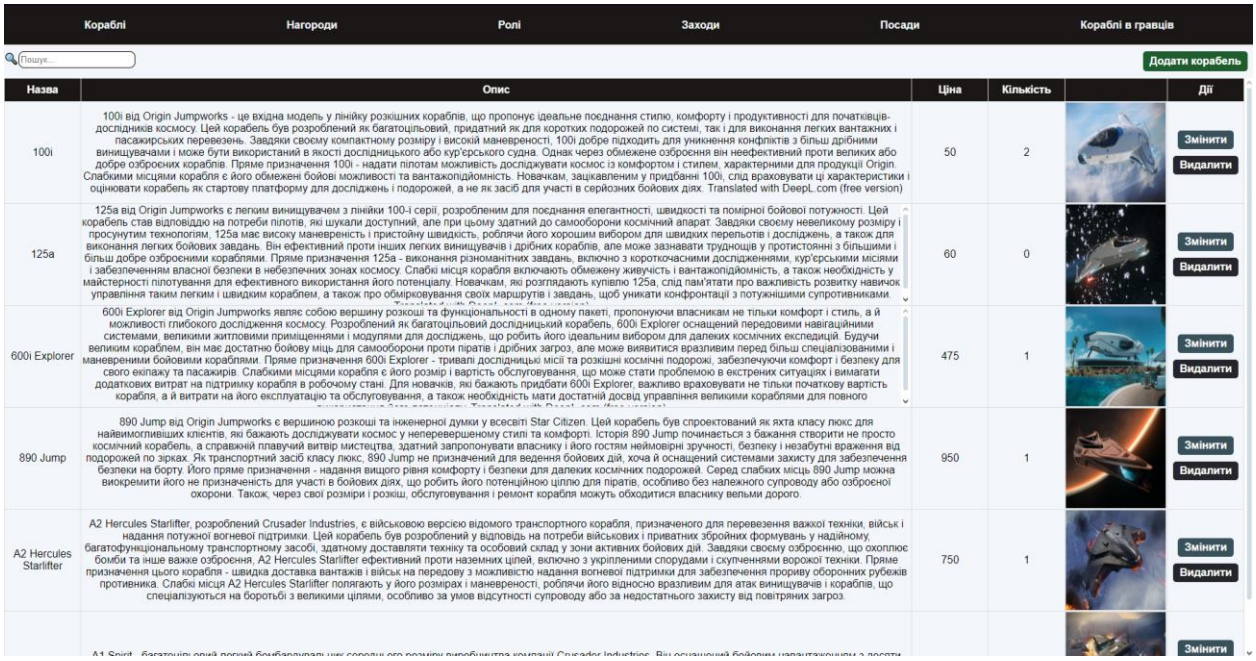

Рисунок 3.25 – Управління базою даних кораблів

Протестуємо можливість додавати кораблі в ангар користувача (рис. 3.26).

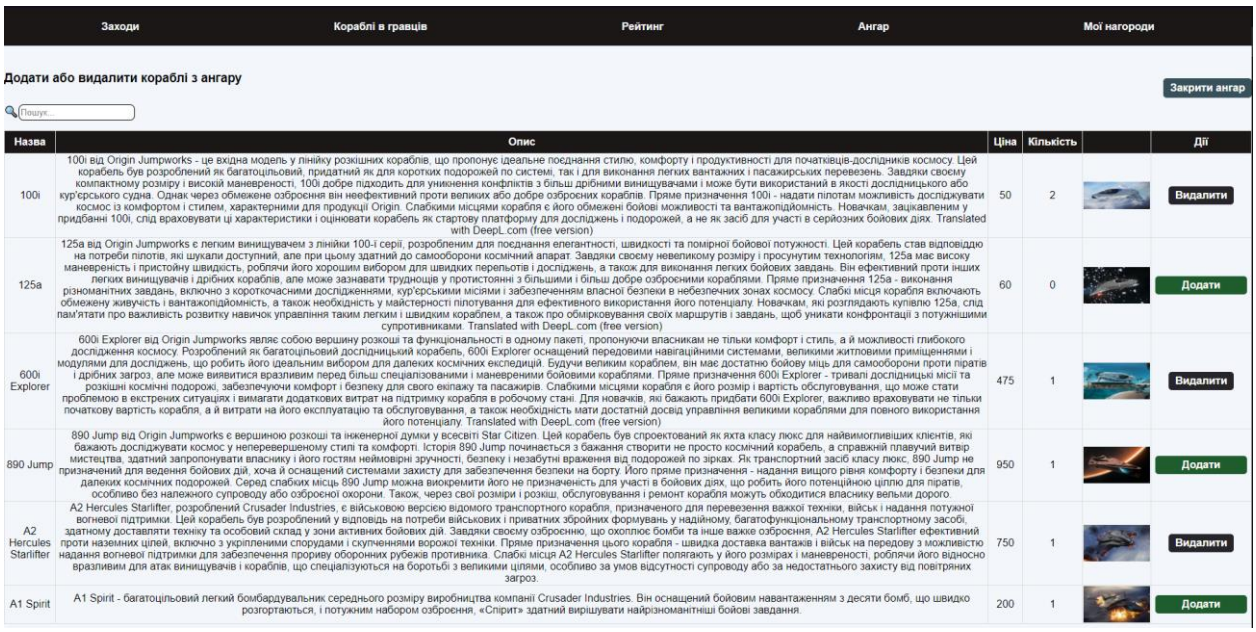

Рисунок 3.26 – Додавання кораблів в ангар

3.4 Висновки до розділу 3

У цьому розділі ми детально розглянули процес програмної реалізації інформаційної системи для ігрової спільноти, використовуючи сучасні вебтехнології, такі як Django та Angular. Робота була поділена на дві основні частини: розробка бекенд та фронтенд складових системи.

Ми розглянули структуру проєкту на Django, описали налаштування середовища розробки та створення моделей даних, які є основою для зберігання та управління інформацією в системі.

Було детально описано процес створення REST API, що дозволяє фронтенд додатку взаємодіяти з бекендом для виконання CRUD операцій (створення, читання, оновлення, видалення даних).

Ми описали структуру проєкту на Angular, процес налаштування середовища розробки та створення компонентів та сервісів, що забезпечують інтерактивність користувацького інтерфейсу.

Особлива увага була приділена інтеграції з бекендом через сервіс ApiService, який забезпечує виконання HTTP-запитів до API та обробку отриманих даних.

Використання Django REST Framework для створення API та JWT для аутентифікації забезпечило надійний та безпечний бекенд, тоді як Angular дозволив створити динамічний та інтуїтивно зрозумілий фронтенд.

ВИСНОВКИ

У процесі виконання дипломної роботи було розроблено інформаційну систему для ігрової спільноти, що базується на сучасних вебтехнологіях Django та Angular. Дана система призначена для забезпечення інтерактивної взаємодії користувачів, управління ігровими подіями, а також безпечного обміну інформацією.

Було проведено детальний аналіз вимог до системи та визначено основні функціональні та нефункціональні вимоги. Розроблено архітектуру системи, що складається з бекенд та фронтенд частин, а також визначено ключові компоненти та їх взаємодію.

Створено серверну частину системи за допомогою фреймворку Django, що забезпечує надійне та ефективне управління даними. Розроблено та реалізовано моделі даних відповідно до створених діаграм.

Створено REST API за допомогою Django REST Framework, що дозволяє здійснювати CRUD операції з даними. Впроваджено аутентифікацію на основі JSON Web Token (JWT), що забезпечує високий рівень безпеки та зручності для користувачів.

Створено клієнтську частину системи за допомогою фреймворку Angular, що забезпечує інтерактивний користувацький інтерфейс. Розроблено основні компоненти та сервіси, що дозволяють здійснювати взаємодію з бекендом та забезпечують зручність використання системи.

Систему було розгорнуто на сервері, що дозволяє продемонструвати її функціональність у реальних умовах.

ПЕРЕЛІК ПОСИЛАНЬ

1. Mele A. Django 3 By Example: Build powerful and reliable Python web applications from scratch : 3rd edition. Packt Publishing, 2020. 568 p.
2. Офіційна документація Django. URL: https://www.djangoproject.com (дата звернення: 20.02.2024).
3. Офіційна документація Angular. URL: https://angular.io (дата звернення: 20.02.2024).
4. JSON Web Tokens Introduction. *JWT Documentation*. URL: https://jwt.io/introduction/ (дата звернення: 16.03.2024).
5. Документація SQLite. URL: https://www.sqlite.org/docs.html (дата звернення: 22.03.2024).
6. Керівництво по використанню SQLite в Django. URL: https://docs.djangoproject.com/en/5.0/ref/databases/#sqlite-notes (дата звернення: 22.03.2024).
7. UML-діаграми. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 18.03.2024).
8. Мартиненко Г. Ю. Модель «сутність-зв’язок» (ER – діаграми). URL: https://web.kpi.kharkov.ua/dpm/wp-content/uploads/sites/123/2020/05/LEKTSIYA-10-11-OBD_ukr.pdf (дата звернення: 18.03.2024).

ДОДАТОК А

Код та демонстрація

А.1 Посилання на репозиторій з кодом проєкту

У цьому репозиторії міститься вихідний код інформаційної системи для ігрової спільноти, розробленої за допомогою Django та Angular. Ви можете знайти всі файли проєкту. URL: https://github.com/honigwein/diplom.

А.2 Посилання на розгорнуту інформаційну систему

Для демонстрації функціоналу розробленої інформаційної системи, вона була розгорнута і доступна за посиланням. URL: http://141.148.245.77:5007.