МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КОМПЛЕКСНИЙ КУРСОВИЙ ПРОЄКТ
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система «MonopolyUA». Синхронна серверна частина.

(тема)

Виконав:
Здобувач 3 курсу, групи ПЗПІ-22-6
Богдан ШАРАЄВ
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник ст. викл. кафедри ПІ Костянтин ОНИЩЕНКО
(посада, Власне ім'я, ПРІЗВИЩЕ)

Члени комісії (Власне ім'я, ПРІЗВИЩЕ, підпис)

Віра ГОЛЯН

Наталія ГОЛЯН

Ольга КАЛИНИЧЕНКО

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

Курс 3 Група ПЗПІ-22-6 Семестр 6

ЗАВДАННЯ
на курсовий проект(роботу) студента

здобувачеві Шараєву Богдану Олеговичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система «MonopolyUA». Синхронна серверна частина.

2. Термін здачі студентом закінченої роботи „20” червня 2025 р.

3. Вихідні дані до проєкту Технічне завдання та вимоги, вибрані технології, середовище розробки.

4. Перелік питань, що потрібно опрацювати в роботі
Аналіз аналогів, визначення основних функції, проєктування архітектури та бази даних серверної частини, реалізація REST API, аналіз отриманих результатів.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	04.06.2025	виконано
2	Розробка постановки задачі	04.06.2025	виконано
3	Проектування ПЗ	05.06.2025	виконано
4	Програмна реалізація	05.06.2025	виконано
5	Аналіз результатів	08.06.2025	виконано
6	Підготовка пояснювальної записки.	14.06.2025	виконано
7	Перевірка на наявність ознак академічного плагіату	19.06.2025	виконано
8	Захист роботи		виконано

Дата видачі завдання “ 26 ” лютого 2025р.

Здобувач Шараєв Богдан .
(підпис)

Керівник роботи _____ ст.викл. кафедри ПІ Костянтин ОНИЩЕНКО
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 72 стор., 23 рис., 10 джерел.

ВЕБ-ГРА, ІГРОВИЙ ЗАСТОСУНОК, СЕРВЕРНА ЧАСТИНА, DJANGO, DRF, PYCHARM, PYTHON

Об’єкт розробки – синхронна серверна частина для веб-додатку.

Мета розробки – розробити функціональну серверну частину, яка забезпечує основні можливості взаємодії користувачів поза межами активного ігрового процесу, включаюси авторизацію, управління профілем, взаємодію з друзями, транзакції ігрових предметів та перегляд ігрової статистики.

Метод рішення – середовище розробки PyCharm, мови програмування Python та фреймворки Django і Django REST Framework.

У результаті розробки створено серверну частину, яка реалізує повноцінний функціонал для користувача.

BACKEND, WEB APP, POSTGRESQL, PYCHARM, PYTHON

The object of development is a synchronous server part for a web application.

The goal of the development was to develop a functional server side that provides basic user interaction capabilities outside the active game process, including authorization, profile management, interaction with friends, transactions of game items, and viewing game statistics.

The solution was based on the PyCharm development environment, the Python programming language, as well as Django and the Django REST Framework.

As a result of the development, a server side was created that implements the full functionality for the user: registration and authorization (including Google OAuth support), profile editing, viewing game statistics, a system of friends and messages, a history of games played, management of game items through the inventory, as well as a marketplace for buying and selling game items.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

СУБД – Система Управління Базами Даних

HTTP – Hypertext Transfer Protocol

REST – Representational State Transfer

API – Application Programming Interface

DRF – Django REST Framework

MVT – Model View Template

SQL – Structured Query Language

CSRF – Cross-Site Request Forgery

JSON – JavaScript Object Notation

JWT – JSON Web Token

UML – Unified Modeling Language

ORM – Object-Relational Mapping

ВСТУП

Сьогодні онлайн-ігри користуються великою популярністю, оскільки дозволяють людям грати разом незалежно від місця перебування. Це особливо актуально для настільних ігор, які дедалі частіше адаптуються до онлайн формату. Перехід у веб-формат відкриває чимало нових можливостей — це не лише розширює аудиторію та спрощує технічні процеси, а й дозволяє створити зручні інструменти для взаємодії між користувачами. У такому форматі гравці можуть легко створювати й переглядати профіль, знаходити друзів, стежити за своєю статистикою та торгувати предметами всередині системи. Це робить використання платформи простішим, швидшим і приємнішим, що в результаті значно підвищує зацікавленість у грі та залученість користувачів.

Програмна система «MonopolyUA» — це веб-застосунок, який відтворює основні механіки гри «Монополія» та одночасно надає розширений користувацький функціонал. Основна увага приділяється створенню серверної інфраструктури застосунку – обробці запитів від клієнта та забезпечення взаємодії між користувачами в системі. У межах проєкту реалізовано такий функціонал, як реєстрація та авторизація користувачів, створення та перегляд профілів, ведення ігрової статистики, внутрішній маркетплейс для торгівлі ігровими предметами, а також система сповіщень та система друзів.

Серверна частина спроєктована з урахуванням основних вимог до сучасних веб-систем — зручності, безпеки, стабільності та можливості подальшого розширення. Застосовано сучасні технології й інструменти, що дозволяють ефективно працювати з базою даних, обробляти одночасні підключення великої кількості гравців і підтримувати постійний зв'язок між клієнтами в режимі реального часу. Таким чином, «MonopolyUA» поєднує в собі класичний ігровий досвід із сучасними технічними рішеннями, створюючи основу для зручної та безпечної платформи, готової до подальшого розвитку і вдосконалення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Настільні веб-ігри стрімко набирають популярності [1], завдяки своїй доступності, для того щоб почати грати, достатньо мати браузер і доступ до інтернету. Однією з відомих ігор, що адаптована під онлайн-формат, є «Монополія» — класична настільна гра, яка в онлайн форматі отримує нові можливості: інтерактивність, зручність взаємодії між гравцями тощо.

Сучасні веб-ігри вже давно вийшли за межі звичайного ігрового процесу. Сьогодні гравцю пропонують цілу різноманітність функціоналу: особистий профіль, списки друзів, внутрішні магазини, досягнення, статистику, рейтинги тощо. Завдяки цьому взаємодія з грою стає глибшою, а користувачі мають змогу не лише розважатися, а й відчувати прогрес, розвиватися і спілкуватися з іншими.

Такі елементи персоналізації та взаємодії користувачів сприяють створенню ігрової спільноти. Гравці можуть підтримувати зв’язок, обмінюватися досягненнями, змагатися за місце в таблиці лідерів або торгувати предметами. Це не просто додає інтересу — це мотивує повертатися до гри знову і знову.

Усе це робить веб-ігри не просто способом розваги, а справжніми багатофункціональними платформами. Саме через це такі проєкти залишаються актуальними та затребуваними, бо вони відповідають очікуванням користувачів, які шукають не лише цікаву гру, а й можливість взаємодії, самовираження та розвитку в межах спільноти.

У межах цього проєкту буде реалізовано серверну синхронну частину для веб-застосунку «MonopolyUA». Основна мета — забезпечити зручний, стабільний і сучасний багатокористувацький функціонал.

Актуальність такого рішення сьогодні дуже висока. В умовах швидкого зростання індустрії онлайн-ігор користувачі очікують не лише цікавого ігрового процесу, а й зручної та функціональної платформи. Такі рішення роблять гру повноцінним середовищем для взаємодії, самовираження та розвитку, значно підвищуючи залученість користувачів.

Реалізація надійної серверної частини є критично важливою, адже саме вона забезпечує збереження даних, синхронізацію дій між користувачами та стабільну роботу всіх функцій, пов’язаних з профілями, сповіщеннями, торгівлею та статистикою. Саме тому розробка серверної частини є дуже важливою.

1.2 Виявлення та вирішення проблем

У розробці серверної синхронної бекенд-частини для веб-застосунків, виникає багато проблем, які потребують комплексного підходу до їх вирішення. Особливо це стосується побудови системи для взаємодії між користувачами, яка має гарантувати стабільну й передбачувану роботу всієї розроблюваної системи.

Однією з ключових проблем є забезпечення надійної та безперебійної взаємодії між клієнтами через сервер, який має обробляти одночасні запити від багатьох користувачів. Це вимагає правильної організації синхронної обробки запитів, щоб уникнути конфліктів та забезпечити коректне оновлення даних профілів, списку друзів, маркетплейсу та сповіщень.

Ще однією суттєвою проблемою є захист персональних даних користувачів і забезпечення безпеки аутентифікації та авторизації. Необхідно впровадити надійні механізми аутентифікації, управління сесіями та захист від типових загроз веб-додатків, таких як SQL-ін’єкції, CSRF тощо. Особливу увагу слід приділити збереженню паролів: вони не повинні зберігатися у відкритому вигляді. Для цього використовуються криптографічні хеш-функції, які дозволяють зберігати лише хешовану версію паролю.

Також важливою задачею є забезпечення продуктивності системи — здатності ефективно обробляти зростаючу кількість користувачів без зниження швидкодії, для цього необхідна оптимізація бази даних.

Вирішення цих проблем здійснюється через застосування сучасних підходів до розробки серверної частини, використання транзакцій для збереження цілісності даних при одночасній взаємодії кількох користувачів, реалізацію безпечної аутентифікації через JWT, що дозволяє зберігати токени доступу лише на стороні клієнта. А також було реалізовано оптимізацію запитів до бази даних з

урахуванням потенційного навантаження. Для цього використовувалися індески для полів, які часто використовуються в пошуку. Завдяки впровадженню цих рішень, система працюватиме стабільно та безпечно, та буде готова до подальшого розширення функціональності й збільшення кількості користувачів.

1.2.1 Цільова аудиторія

Цільовою аудиторією будуть:

- гравці новачки;
- досвідчені гравці;
- користувачі з інтересами до настільних ігор;
- користувачі, що шукаються спілкування онлайн;
- гравці, які віддають перевагу зручному онлайн-доступу;
- користувачі, які цікавляться онлайн-торговлею та колекціонуванням.

Цільова аудиторія веб-застосунку охоплює широкий спектр користувачів із різними інтересами, досвідом та мотиваціями. Основну групу становлять як новачки, які лише починають знайомство зі світом настільних ігор, так і досвідчені гравці, які вже мають глибоке розуміння ігрового процесу та шукають нові можливості для участі в онлайн-іграх. Застосунок також орієнтований на користувачів, які захоплюються настільними іграми як хобі, виявляють зацікавленість у знаходженні нових зв’язків через веб-застосунок, а також прагнуть зручного доступу до ігрового середовища. Додаткову групу користувачів становлять ті, хто цікавиться колекціонуванням ігрових елементів або бере участь в онлайн-торгівлі ігровими предметами. Таким чином, система орієнтована на створення комфортного, інтерактивного середовища для взаємодії різних категорій гравців, що забезпечує не лише гнучкість у користуванні, а й розширені можливостей для розвитку у веб-застосунку.

1.2.2 Монетизація

Впровадження монетизація у веб-застосунку «MonopolyUA» не планується. Усі внутрішні транзакції будуть здійснюватися виключно за допомогою ігрової

валюти, яку можна отримати під час участі в іграх. Такий підхід забезпечує рівні умови для всіх гравців, незалежно від їхнього фінансового стану, та сприяє створенню чесного ігрового середовища, де успіх залежить від активності, навичок та взаємодії, а не від можливості вкладати кошти.

Такий дизайн системи орієнтований на максимальну відкритість, чесність і зручність, що позитивно впливає на залучення та утримання користувачів. Крім того, відсутність потреби у фінансових операціях спрощує юридичну та технічну частину реалізації проєкту, зменшуючи ризики, пов’язані з безпекою платежів, поверненням коштів чи дотриманням законодавства про оплату в інтернеті.

1.3 Аналіз аналогів програмного забезпечення

У процесі розробки серверної частини для програмної системи «MonopolyUA» було проведено аналіз кількох програмних рішень, які вже реалізували подібні функціональні модулі та здобули популярність серед користувачів. Метою аналізу є виявлення ефективних практик для побудови ключових модулів для розроблюваної серверної частини — авторизації, системи профілів, взаємодії між користувачами, маркетплейсу, інвентаря, ігрової статистики та сповіщень. Порівняння існуючих аналогів дозволяє визначити сильні та слабкі сторони існуючих рішень, аби уникнути їхніх недоліків у процесі реалізації власної серверної частини.

Одним із найвідоміших веб-застосунків у галузі цифрових настільних ігор є Tabletopia [2]. Ця платформа надає користувачам можливість запускати настільні ігри онлайн з високим рівнем візуального відтворення ігрових компонентів. Особливу роль у Tabletopia [2] відіграє модуль користувацького профілю, який дозволяє зберігати базову інформацію про користувача — ім’я, аватар, список останніх ігор, улюблені ігри, а також персональну ігрову історію. Дані профілю інтегруються з іншими модулями: списками друзів, кімнатами для гри, інтерфейсом запрошень.

Основним недоліком є відсутність перегляду детальної ігрової статистики щодо зіграних партій, що перешкоджає повноцінному відстеженню особистого прогресу.

Ще одним прикладом ефективного веб-застосунку є Board Game Arena [3]. На відміну від Tabletopia [2], ця платформа зосереджена більше на самій ігровій взаємодії між користувачами. Особливу увагу тут приділено простоті додавання друзів, створенню груп для гри та загальній зручності спілкування. Система сповіщень інформує користувача про важливі події, запрошення та статуси друзів. Авторизація підтримує кілька способів входу, включаючи зовнішні сервіси.

Попри це, застосунок має певні недоліки: сповіщення реалізовано фрагментовано, оскільки частина повідомлень надходить у сам застосунок, а інша через електронну пошту, що знижує загальну зручність. Крім того, у системі відсутня повноцінна деталізована статистика щодо ігрових досягнень користувача, що не дозволяє гравцям повноцінно відстежувати прогрес або порівнювати результати.

У якості ще одного прикладу розглянуто онлайн-сервіс Steam, який хоч і не є безпосередньо орієнтованим на настільні ігри, однак демонструє зразкову реалізацію маркетплейсу, інвентаря та користувацьких сервісів. Інвентар у Steam тісно пов’язаний із кожним окремим обліковим записом, де всі предмети мають власну унікальну ідентифікацію, що забезпечує можливість створення надійної системи обміну та торгівлі. Маркетплейс підтримує відкриту торгівлю між користувачами за внутрішню валюту Steam [4]. Система обробки транзакцій автоматизована, з повною історією купівель, збереженням лотів, фільтрацією за ціною, ринковою вартістю, кількістю доступних екземплярів. Надійність забезпечується багатофакторною аутентифікацією, верифікацією угод і захистом через мобільний додаток. Усі зміни в інвентарі оновлюються практично в реальному часі, що дозволяє уникати конфліктів і гарантує консистентність даних.

Серед потенційних недоліків системи варто відзначити комісію, що стягується з продажів і купівель на маркетплейсі, що може впливати на мотивацію користувачів до активної торгівлі.

2 ПОСТАНОВКА ЗАДАЧІ

Метою даного проєкту є розробка серверної синхронної частини для програмної системи «MonopolyUA», яка забезпечить стабільну та безпечну взаємодію між користувачами. Серверна частина повинна містити всі необхідні функціональні модулі для повного забезпечення роботи програмної системи.

Одним із першочергових завдань є реалізація механізмів авторизації та реєстрації користувачів, що дозволяє створювати нові облікові записи, а також надійно ідентифікувати користувачів під час входу у систему. Важливо, щоб цей модуль забезпечував безпечне зберігання конфіденційної інформації користувачів. Наступним важливим завданням є управління профілем користувача, що передбачає можливість редагування особистої інформації. Також особливу увагу приділено організації взаємодії між друзями – система повинна дозволяти додавати користувачів у список друзів, надсилати запрошення, приймати або відхиляти їх. Для покращення комунікації передбачена система сповіщень, яка буде інформувати користувачів про нові події та запрошення. Система повинна підтримувати функціонал маркетплейсу, що дасть змогу користувачам купувати та продавати ігрові предмети за віртуальну валюту між собою, цей модуль включає механізми формування пропозицій на продаж, обробки транзакцій, а також оновлення стану предметів у режимі реального часу. Також буде реалізовано функціонал ігрової статистики, який зберігатиме та оброблятиме дані про активність користувачів, їх досягнення, виграші, програші та інші показники зіграних ігор.

В результаті, розроблювана серверна частина має виконувати поставлені функціональні вимоги, забезпечувати стабільне виконання всіх передбачених операцій. Розроблювана система має бути стійкою до помилок і готовою до подальшого розвитку та вдосконалення.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

Після детального аналізу функцій користувача була створена Use-case діаграма (див. рис. 3.1).

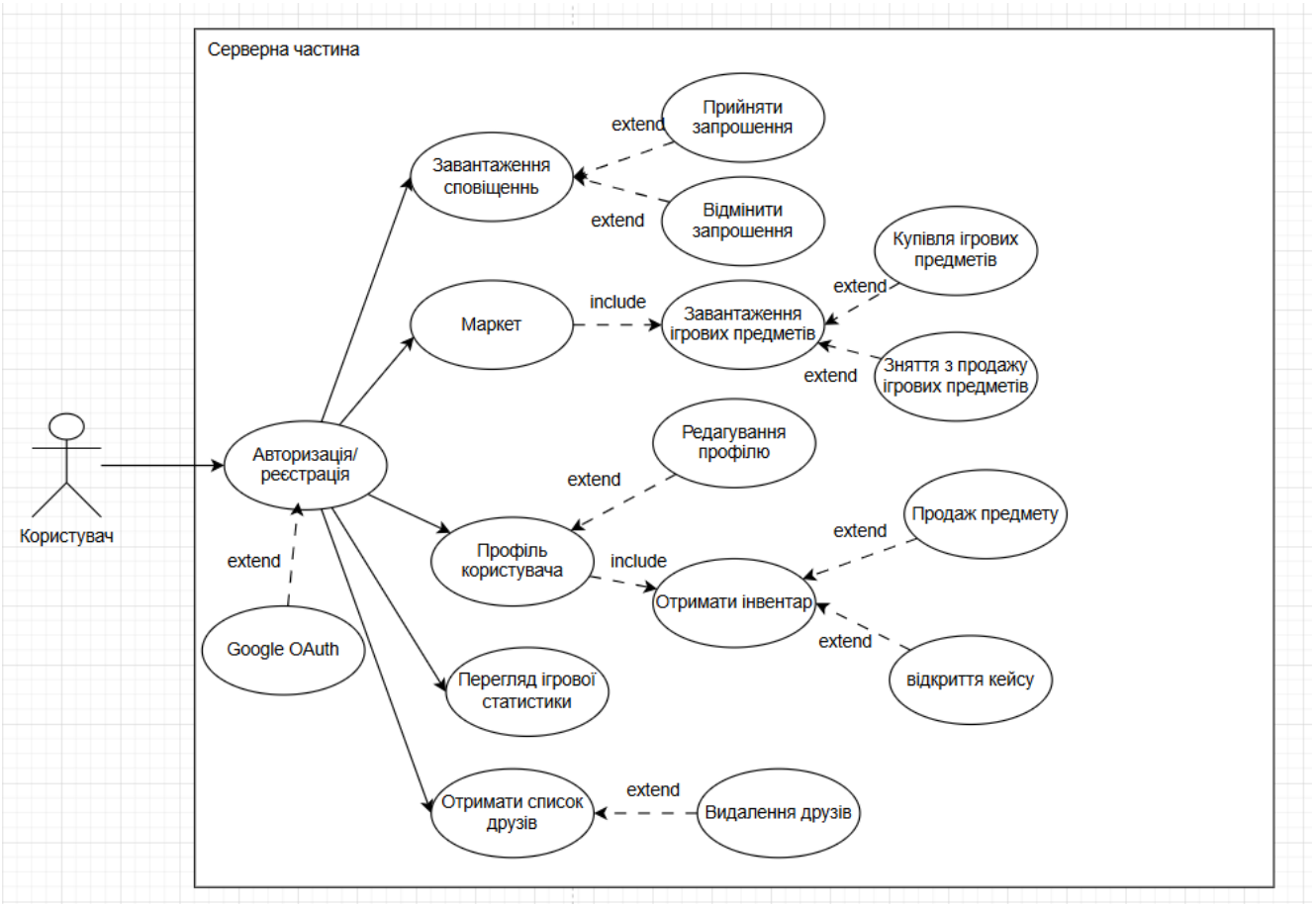


Рисунок 3.1 – Use-case діаграма користувача з серверною частиною (рисунок створено самостійно)

Користувач починає возаємодію із серверною частиною, проходячи авторизацію або реєстрацію, включаючи варіант авторизації через Google OAuth. Після успішного входу він отримує доступ до персонального профілю, де може переглядати або редагувати інформацію про себе, переглядати ігрову статистику, а також переглядати інвентар, який дозволяє переглядати або продавати ігрові предмети користувача. Також користувач може отримати доступ до списку друзів, приймати або скасовувати запрошення, а також видаляти друзів зі списку. Користувач може перейти на маркет, де він може взаємодіяти з товарами, купувати,

продавати або знімати з продажу.

Уся ця діаграма демонструє основні функції, що підтримуються серверною частиною, а також взаємозв’язки між окремими модулями.

3.2 Проєктування архітектури ПЗ

Архітектура програмного забезпечення веб-застосунку «MonopolyUA» побудована на основі клієнт-серверної архітектури [5]. Це означає, що вся система розділена на клієнтську та серверну частини, кожна з яких виконує чітко визначені функції. Клієнтська частина відповідає за взаємодію з користувачем, відображення інтерфейсу та надсилання запитів до сервера. У свою чергу, серверна частина приймає ці запити, обробляє їх, виконує відповідні дії згідно з бізнес-логікою, взаємодіє з базою даних і повертає клієнту відповіді у форматі JSON через HTTP запити. Комунікація між клієнтом і сервером відбувається через REST API [6], що забезпечує зручний та універсальний спосіб обміну даними.

Серверна частина реалізована як набір окремих Django-додатків [7], які логічно розділяють функціонал системи. Це забезпечує розділення відповідальностей, спрощує масштабування системи та дозволяє легше вносити зміни без ризику вплинути на інші частини застосунку.

У побудові внутрішньої структури серверної частини застосовано архітектурний патерн MVT, який є типовим для Django [8]. У цій моделі компоненти (Django-додатки) [7] системи поділяються на модель, представлення та шаблон. Модель відповідає за структуру та збереження даних у базі, представлення реалізує бізнес-логіку та обробку HTTP-запитів, а шаблон — за формування відображення даних для користувача. Такий підхід дозволяє легко розширювати функціональність, повторно використовувати код та ізолювати зміни в межах одного модуля без впливу на інші частини системи.

До складу серверної частини «MonopolyUA» входять наступні основні модулі. Модуль Auth реалізує повний цикл аутентифікації користувача. У його межах зосереджено логіку реєстрації, авторизації через класичну форму входу, а також підтримку сторонньої автентифікації через Google OAuth.

Модуль GameData відповідає за зберігання статистики, пов’язаної з перебігом ігрових сесій. У ньому зосереджено логіку обліку зіграних ігор, збереження результатів та рейтингу.

Модуль Items реалізує основну логіку, пов’язану з управлінням ігровими предметами. У межах цього модуля об’єднано дві функціональні підсистеми — інвентар та маркет. Функціональність інвентаря охоплює зберігання, перегляд, застосування та оновлення предметів, що належать користувачу. Маркетплейс, у свою чергу, відповідає за механізми торгівлі між користувачами — зокрема за створення торгових пропозицій, перегляд доступних лотів, купівлю, продаж та оновлення цін.

Модуль Notifications реалізує логіку повідомлень і сповіщень, інформуючи користувачів про важливі події, такі як запрошення або системні повідомлення.

Модуль Friends реалізує систему дружби між користувачами системи.

Для детальнішого відображення основних модулів серверної частини, було розроблено UML-діаграму компонентів серверної частини (див. рис. 3.2).

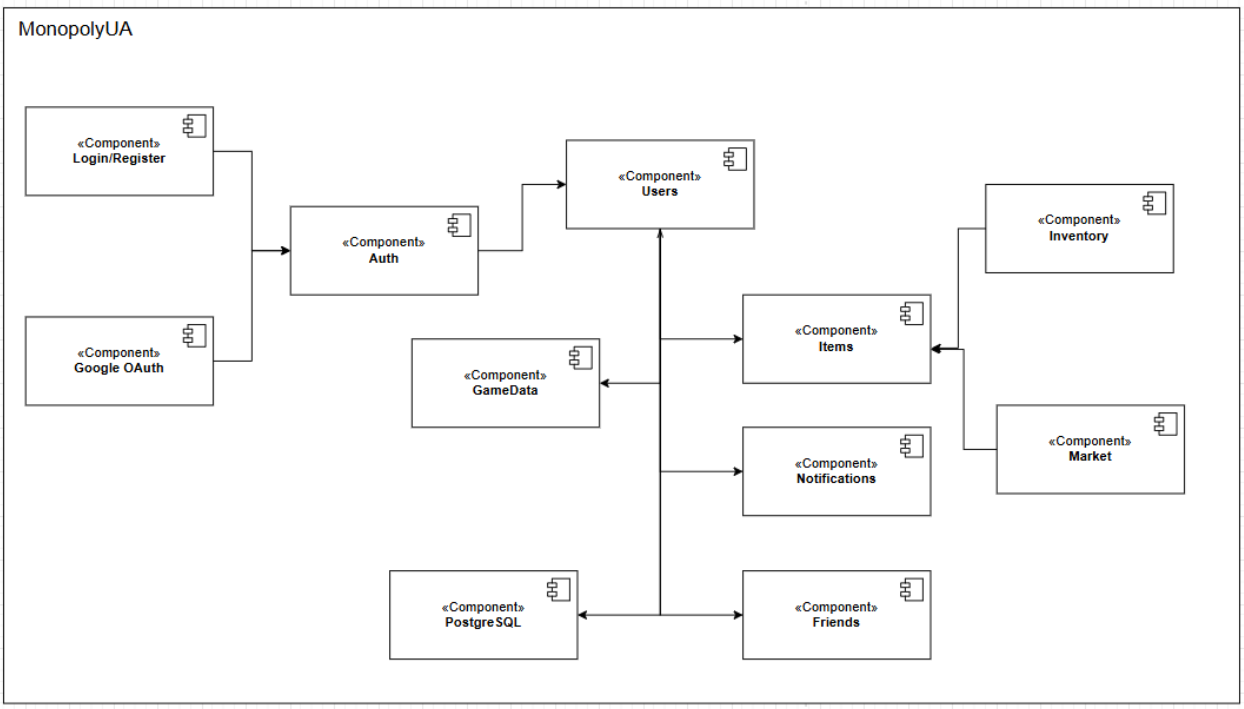


Рисунок 3.2 – UML-діаграма компонентів (рисунок створено самостійно)

Діаграма компонентів демонструє розподіл системи на окремі логічні модулі, такі як модуль авторизації, модуль користувачів, модуль друзів, сповіщень та інші.

Ця діаграма відображає, як ці компоненти взаємодіють між собою та з базою даних, а також як організовано обмін даними всередині системи.

Після проєктування архітектури ПЗ, була розроблена UML-діаграма розгортання (див. рис. 3.3)

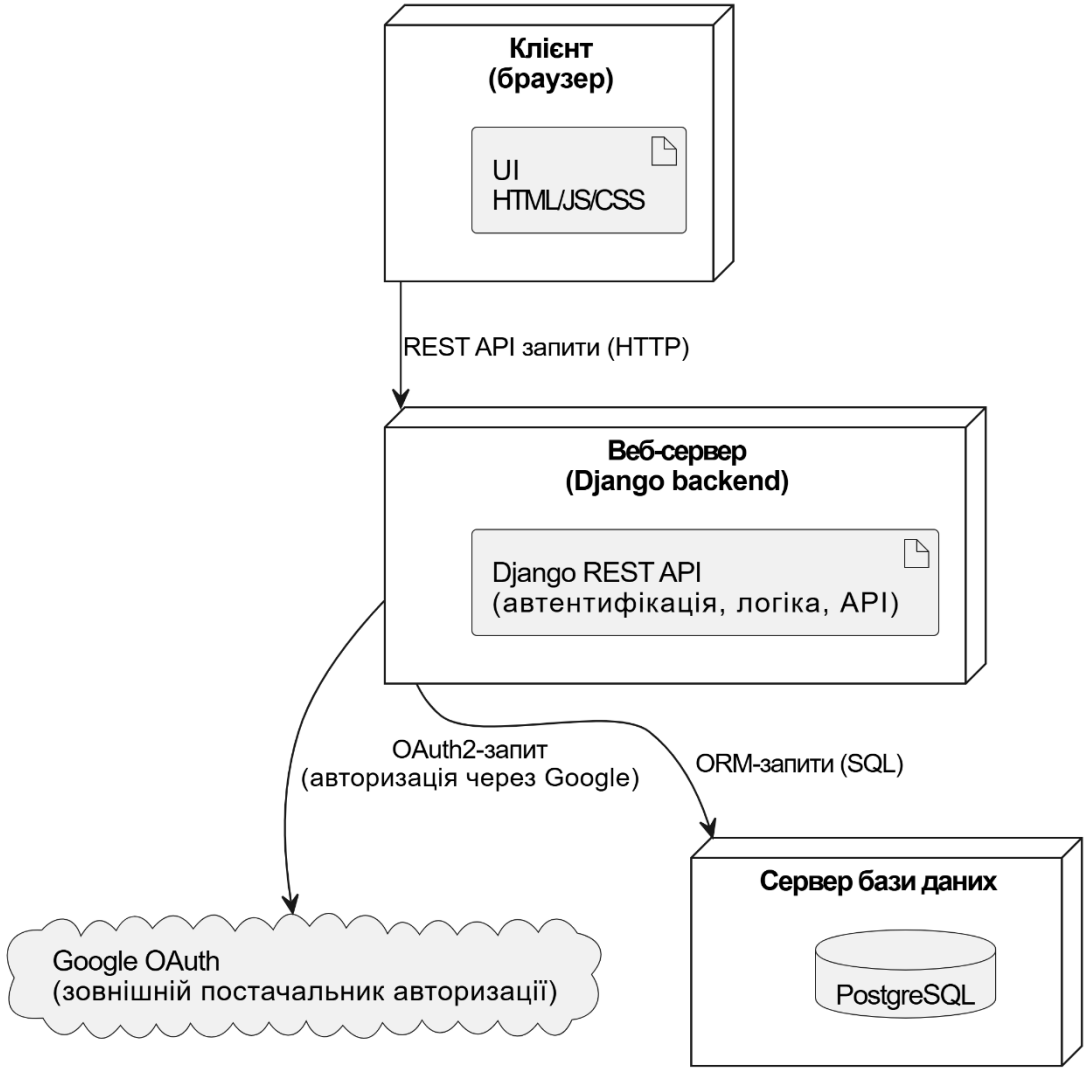


Рисунок 3.3 – UML-діаграма розгортання розроблюваної системи (рисунок створено самостійно)

Діаграма розгортання ілюструє фізичну структуру системи «MonopolyUA» та взаємодію ключових компонентів під час роботи застосунку. Клієнт, який здійснює запити через REST API [6] до серверної частини. Сервер обробляє ці запити, виконує бізнес-логіку та взаємодіє з базою даних PostgreSQL для збереження й отримання даних. Окрім цього, сервер підтримує аутентифікацію через сторонній

сервіс Google OAuth, надсилаючи відповідні запити до сервісів Google для підтвердження особи користувача. Таким чином, діаграма демонструє інтегровану систему, де клієнт, сервер, база даних і зовнішні сервіси взаємодіють для забезпечення стабільної та зручної роботи застосунку.

Загалом, архітектура демонструє добре структурований підхід до розробки на Django, де кожен модуль має чітко визначену сферу відповідальності, що сприяє розділенню логіки, полегшує тестування та підтримку проєкту.

3.3 Проєктування структури зберігання даних.

Розроблено структуру зберігання даних для системи «MonopolyUA» на основі реляційної моделі з використанням системи управління базами даних PostgreSQL із застосуванням Django ORM [9]. PostgreSQL обрана через її високу надійність та підтримку складних операцій з даними [10]. Ця СУБД дозволяє забезпечити цілісність і узгодженість інформації завдяки механізмам транзакцій та обмежень цілісності, що особливо важливо для розроблюваної системи.

Далі наведена схема розробленої бази даних (див. рис. 3.4).

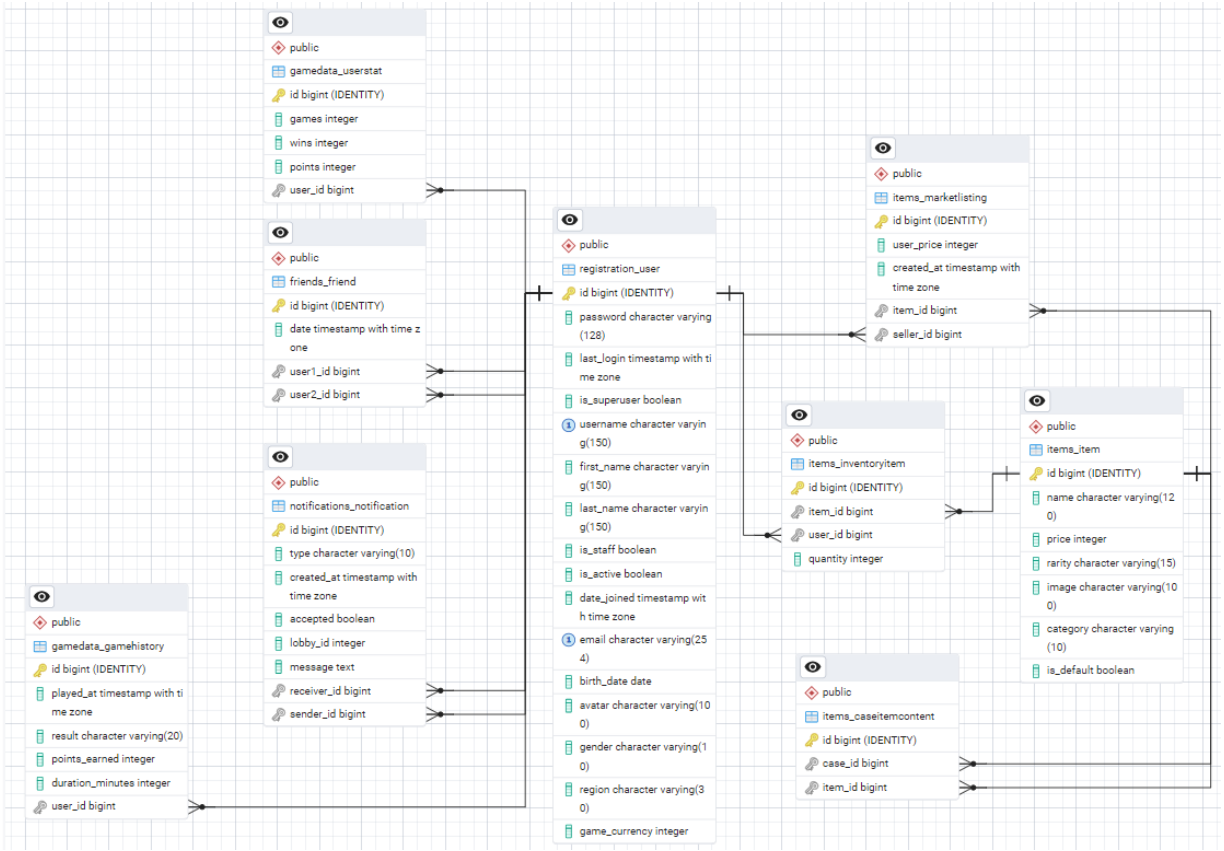


Рисунок 3.4 – Схема розробленої бази даних.

Структура бази даних орієнтована на забезпечення цілісності, узгодженості та масштабованості даних. Центральним елементом є таблиця користувачів, яка зберігає основну інформацію про кожного зареєстрованого користувача системи. Ця таблиця є базовою для встановлення зв’язків із більшістю інших таблиць в БД.

Інвентар користувача реалізований через окрему таблицю, яка фіксує зв’язок між користувачем та предметами, що йому належать. Всі ігрові предмети зберігаються у спеціальній таблиці, яка описує кожен предмет у системі, включаючи його характеристики. Таким чином, таблиця інвентаря виступає як проміжна таблиця між користувачем і таблицею предметів.

Для реалізації можливості торгівлі між користувачами створено таблицю виставлених на продаж предметів, яка фіксує предмети, виставлені на продаж із зазначенням продавця, ціни та часу публікації оголошення. Ця таблиця пов’язана як з користувачем, так і з предметом, що виставлено на продаж.

Окремо реалізовано таблицю, яка відповідає за вміст кейсів – контейнерів, що містять ігрові предмети. Таблиця встановлює відповідність між кейсом та набором предметів в ньому.

Для сповіщень, передбачено окрему таблицю, яка зберігатиме інформацію про нові події та різні запрошення. Вона містить інформацію про зміст, відправника, одержувача, дату та статус.

Також було реалізовано таблиці, що дозволяють відстежувати прогрес користувачів у грі, таблиця ігрової статистики, зберігає всю статистику користувача, кількість ігр, перемог та набраних очок. Друга таблиця зберігає історію зіграних ігор, включаючи дату, результат, тривалість та кількість отриманих очок за конкретний матч.

Усі таблиці поєднані між собою через зовнішні ключі, що забезпечує логічну цілісність даних і дозволяє підтримувати узгодженість та актуальність даних у межаї усієї системи.

Отже, використання PostgreSQL та ORM дає змогу легко працювати зі складними запитами і великими обсягами інформації, що необхідно для реалізації основного функціоналу програмної системи [10]. Крім того, розширюваність

PostgreSQL дозволяє інтегрувати додаткові механізми обробки даних і аналітики, що підвищує гнучкість і довгострокову перспективу розвитку «MonopolyUA». Таким чином, обрана СУБД повністю відповідає вимогам надійності та продуктивності проєкту.

3.4 Приклади використання алгоритмів та методів.

В програмній системі «MonopolyUA» одним із важливих елементів реалізації є алгоритм відкриття кейсів, який поєднує кілька технічних рішень, що забезпечують надійність обробки даних та коректність бізнес-логіки.

Перш за все, варто відзначити використання транзакції, яка охоплює увесь процес відкриття кейсу — від зменшення кількості кейсів у користувача до додавання нового предмета до інвентарю. Використання transaction.atomic в Django ORM дозволяє забезпечити атомарність операцій, що сприяє підтриманню цілісності даних [9]. Це означає, що всі дії виконуються як єдине ціле: якщо на будь-якому етапі виникає помилка, жодна з попередніх змін не буде збережена. Такий підхід захищає базу даних від непослідовних станів (наприклад, коли кейс вже зник, а предмет ще не додано), що особливо важливо для транзакцій, пов’язаних з ігровими предметами.

Ще одним елементом є система розподілу ймовірностей на основі рідкостей предметів. Замість встановлення фіксованих ймовірностей для кожного окремого предмета, система використовує базові ваги для кожної категорії рідкості (звичайна, рідкісна, епічна, легендарна), а потім автоматично розподіляє ці ваги між усіма предметами відповідної категорії, що є в кейсі. Завдяки цьому досягається баланс: користувачі мають однаковий шанс отримати предмети в межах рідкісності, незалежно від їх кількості в кейсі.

Також реалізовано алгоритм масової купівлі предметів з маркету. Логіка передбачає можливість придбання кількох предметів за вказаною ціною. Алгоритм перебирає відповідні активні лоти на маркеті, сортує їх від найнижчої ціни і по черзі купує стільки, скільки було вказано користувачем. Кожна купівля виконується у межах транзакції, що гарантує цілісність даних. Якщо користувач,

наприклад, вказав купити 5 предметів, але вдалося лише 3 через відсутність активних лотів або недостатність коштів, алгоритм коректно обробляє часткову покупку та повертає повідомлення з поясненням.

Метод відновлення пароля передбачає, що користувач, який забув свій пароль, надсилає запит із електронною адресою. Система знаходить відповідний обліковий запис і генерує новий складний пароль, що складається з 8 символів і включає літери різного регістру, цифри та спеціальні символи. Цей пароль автоматично встановлюється у системі замість старого, а користувачу надсилається лист з новим паролем. Такий підхід забезпечує автоматичне та безпечне відновлення доступу до акаунту, мінімізуючи ризики та підвищуючи зручність для користувача.

Ще однією важливою частиною системи є алгоритм додавання ігрової статистики, який відповідає за обробку результатів завершеної гри та оновлення відповідних даних користувачів.

Процес виконується в рамках транзакції, що забезпечує цілісність і узгодженість інформації. Алгоритм приймає список гравців, тривалість гри та дату проведення. Для кожного користувача визначається його місце у грі, на основі якого присвоюються певні рейтингові бали та ігрова валюта за встановленими таблицями відповідностей. Користувачам додається відповідна кількість ігрової валюти, а також оновлюється їх статистика: загальна кількість зіграних ігор, кількість перемог та загальна сума очок.

Після оновлення даних користувача, створюється запис у історії ігор, що містить інформацію про користувача, результат гри (перемога чи поразка), набрані очки, тривалість партії та дату проведення. Таким чином забезпечується прозорість і можливість подальшого аналізу ігрової активності.

Завдяки використанню транзакції забезпечується надійність та цілісність збереження ігрових даних — у разі будь-яких помилок усі зміни скасовуються, що виключає можливість неконсистентності бази даних.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Авторизація та реєстрація користувачів.

Функціонал реєстрації та авторизації користувачів у даному веб-застосунку реалізовано з використанням технології JWT, що дозволяє організувати доступ до API. Такий підхід було обрано з огляду на його масштабованість, зручність інтеграції з фронтендом, а також загальну ефективність у контексті розподілених архітектур, де фронтенд і бекенд є окремими компонентами системи. JWT забезпечує зберігання мінімуму стану на сервері, оскільки інформація про автентифікацію користувача міститься безпосередньо в токені, який клієнт передає у заголовку кожного запиту. Це спрощує реалізацію автентифікації та робить систему більш придатною для майбутнього масштабування.

Для реалізації було використано фреймворк Django у поєднанні з Django REST Framework, що дозволяє гнучко створювати RESTful API. Для підтримки механізму авторизації за допомогою токенів застосовано бібліотеку rest_framework_simplejwt, яка є рекомендованим стандартом для організації JWT-автентифікації в середовищі Django. Реєстрація користувача реалізована за допомогою окремого API-ендпоінту, що приймає дані у форматі POST-запиту, зокрема ім’я користувача, електронну пошту та пароль. Після проходження валідації на сервері створюється новий обліковий запис, а у відповідь клієнту одразу надсилаються access та refresh токени. Такий підхід дає змогу користувачеві одразу після реєстрації бути авторизованим у системі, що робить використання швидшим і зручнішим.

```
refresh = RefreshToken.for_user(user)
            return Response({
                "refresh": str(refresh),
                "access": str(refresh.access_token),
            }, status=status.HTTP_200_OK)

return Response({"error": "Неправильні дані"},
status=status.HTTP_400_BAD_REQUEST)
```

4.2 Система відновлення паролю.

Для підвищення зручності користувачів та забезпечення безпеки у випадку втрати паролю реалізовано функціонал відновлення паролю. Цей механізм дає змогу користувачу швидко отримати доступ до свого акаунту без необхідності звертатися до адміністратора або використовувати складні процедури.

В основі рішення лежить API-ендпоінт, який приймає POST-запит із електронною поштою користувача. Сервер виконує пошук користувача у базі за вказаною електронною адресою. У разі успішного пошуку генерується новий тимчасовий пароль за допомогою функції generate_password, яка створює надійний пароль із комбінації великих і малих літер, цифр та спеціальних символів. Це забезпечує високий рівень складності нового паролю, що мінімізує ризик несанкціонованого доступу.

```
def generate_password(length=8):
    required = [
        random.choice(string.ascii_uppercase),
        random.choice(string.ascii_lowercase),
        random.choice(string.digits),
        random.choice("!@#$%^&*")
    ]
    all_chars = string.ascii_letters + string.digits + "!@#$%^&*"
    remaining = random.choices(all_chars, k=length - 4)
    password = ''.join(random.sample(required + remaining, length))
    return password
```

Новий пароль автоматично зберігається у хешованому вигляді в базі даних, замінюючи старий, а також відправляється користувачу на електронну пошту через вбудовану функцію надсилання листів send_mail. Таким чином, користувач отримує швидкий і безпечний спосіб відновлення доступу до свого акаунту.

```
user.set_password(new_password)
        user.save()
        send_mail(
            subject="Відновлення пароля",
            message=f"Ваш новий пароль: {new_password}",
            from_email=settings.DEFAULT_FROM_EMAIL,
            recipient_list=[email],
            fail_silently=False
)
```

```
return Response({"message": "Новий пароль надіслано на вашу пошту"}, status=status.HTTP_200_OK)
```

Було обрано саме такий підхід у реалізації, оскільки:

– забезпечує безпеку за рахунок випадкової генерації паролю з урахуванням складних символів;
– мінімізує навантаження на службу підтримки, адже користувач може самостійно відновити доступ;
– використовує вбудовані механізми Django для роботи з паролями і електронною поштою, що підвищує надійність і сумісність системи.

Завдяки цьому функціоналу користувачі отримують зручний і безпечний інструмент відновлення доступу до свого облікового запису, що підвищує загальний рівень задоволеності від використання веб-застосунку.

4.3 Алгоритм відкриття кейсів.

Функціонал відкриття кейсів реалізовано як окремий API-ендпоінт, доступний лише авторизованим користувачам. Основна мета цього алгоритму — забезпечити користувачеві можливість отримати випадковий предмет із віртуального кейсу згідно з його вмістом і заданими шансами випадіння, залежно від рідкості предметів.

Для побудови логіки відкриття кейсу було реалізовано клас OpenCaseAPIView, який обробляє POST-запити. Він перевіряє наявність кейсу у користувача, списує одиницю з інвентарю (або видаляє, якщо це останній екземпляр), після чого виконує вибір предмета із заданого набору предметів кейсу.

В основі механізму лежить імовірнісний розподіл, який реалізовано на основі словника RARITY_WEIGHTS. У цьому словнику для кожного рівня рідкості (COMMON, RARE, EPIC, LEGENDARY) задається загальна вага, яка потім пропорційно розподіляється між усіма предметами відповідної рідкості у кейсі. Причина вибору такого підходу полягає в необхідності забезпечення керованого й справедливого випадіння ігрових предметів з кейсів. Це дозволяє підтримувати баланс між цінністю предметів та ймовірністю їх випадіння.

```
RARITY_WEIGHTS = {
    Item.RARITY_COMMON: 50,
    Item.RARITY_RARE: 30,
    Item.RARITY_EPIC: 20,
    Item.RARITY_LEGENDARY: 10,
}
for content in contents:
    rarity = content.item.rarity
    if rarity in RARITY_WEIGHTS and rarity_counts[rarity] > 0:
       items.append(content.item)
       weight_per_item = RARITY_WEIGHTS[rarity]
/rarity_counts[rarity]
       weights.append(weight_per_item)

if not items:
    return Response({'detail': 'Предмети відсутні.'},
status=status.HTTP_400_BAD_REQUEST)

selected_item = random.choices(items, weights=weights, k=1)[0]
```

4.4 Функція масової купівлі предметів.

Функція масової купівлі предметів реалізована у вигляді API-ендпоінта, що дозволяє користувачам одночасно придбати декілька одиниць одного й того ж товару на маркетплейсі. Такий підхід значно підвищує зручність для користувачів їм не потрібно робити багато окремих замовлень, а можна оформити купівлю одразу потрібної кількості.

```
class BuyMultipleItemAPIView(APIView):
    permission_classes = [IsAuthenticated]
    @transaction.atomic
    def post(self, request, item_id):
```

У роботі функції користувач відправляє запит з параметрами: скільки одиниць товару він хоче купити (user_quantity) і максимальну ціну, яку готовий заплатити за одиницю (user_price). Спершу сервер перевіряє, чи коректно задані ці

параметри — вони мають бути позитивними числами. Потім система шукає всі доступні на ринку пропозиції цього товару, які не належать самому покупцю, і відбирає ті, у яких ціна не перевищує вказану максимальну.

Якщо на ринку немає достатньої кількості товарів або ж немає лотів за потрібною ціною — користувач отримує повідомлення про помилку.

```
market_listings = MarketListing.objects.filter(
    item=item,
    user_price__lte=user_price
).exclude(
    seller=buyer
).order_by('user_price')

total_available = market_listings.count()
full_listing =
MarketListing.objects.filter(item=item).exclude(seller=buyer).count()

if full_listing == 0:
    return Response({'error': 'Недостатньо товарів в продажу.'},
status=status.HTTP_400_BAD_REQUEST)
elif total_available == 0:
    return Response({'error': 'Недостатньо товарів з такою ціною.'},
status=status.HTTP_400_BAD_REQUEST)
```

Далі відбувається ітерація по відібраних лотах: для кожного лоту перевіряється, чи вистачає грошей у покупця. Якщо грошей недостатньо — операція зупиняється. Якщо вистачає — гроші списуються з балансу покупця, зараховуються продавцю, додається відповідна кількість товару до інвентарю покупця, а сам лот видаляється з ринку.

Завдяки використанню атомарної транзакції, у випадку будь-якої помилки (наприклад, нестачі грошей або відсутності товару) всі зміни скасовуються, що запобігає частковому оновленню даних і забезпечує коректність стану системи.

Підсумовуючи, ця функція створена для зручності та безпеки одночасної купівлі декількох одиниць товару з чіткою перевіркою умов і гарантією цілісності операції.

4.5 Додавання інформації про зіграну сесію.

Функція додавання інформації про зіграну ігрову сесію реалізована у вигляді

API-ендпоінту. Основне її призначення — зберігати результати матчів, оновлювати ігрову статистику користувачів і формувати історію зіграних партій. Коли користувач надсилає запит, сервер приймає масив із даними про всіх учасників гри, тривалість сесії та дату її проведення. Усі ці дії обгорнуті в одну транзакцію за допомогою декоратора @transaction.atomic, що забезпечує надійність і узгодженість змін у базі даних навіть у разі помилки.

Дані про кожного гравця обробляються по черзі. Згідно з місцем, яке посів користувач у грі, визначається кількість очок рейтингу та ігрової валюти, які він отримує. Ці значення беруться з наперед заданих словників RATINGS і MONEY. Користувачеві нараховуються ці ресурси, і його дані оновлюються в базі. Якщо запис зі статистикою для цього гравця ще не існує, він створюється автоматично.

```
user = get_object_or_404(User, id=id)
user.game_currency += money
user.save()
user_stat, created = UserStat.objects.get_or_create(user=user)
user_stat.games += 1
if place == 1:
      user_stat.wins += 1
       result = "Перемога"

user_stat.points += rating
user_stat.save()
```

Окрім статистики, також створюється окремий запис у таблиці GameHistory, де фіксується результат зіграної гри, кількість отриманих очок рейтингу, тривалість гри та дата. Це дозволяє користувачу переглядати свою історію ігор, аналізувати прогрес і бачити, як змінюється його успішність із часом.

```
game = GameHistory(
user=user,
      result=result,
      points_earned=rating,
      duration_minutes=duration,
      played_at=played_at
)
game.save()
```

Такий підхід дозволяє надійно зберігати результати гри, формувати

змагальне середовище, а також забезпечує основу для подальшої реалізації системи рейтингів, досягнень або сезонів.

4.6 Авторизація за допомогою Google OAuth.

Для зручності користувачів, а також з метою пришвидшення процесу реєстрації та авторизації, у системі реалізовано механізм авторизації через Google. Цей спосіб дозволяє новим користувачам отримати доступ до застосунку без потреби вручну створювати обліковий запис або запам’ятовувати пароль. Технологічно реалізація базується на протоколі OAuth 2.0, що є сучасним стандартом для делегованої автентифікації.

Процес починається з того, що користувач переходить на спеціальний ендпоінт, який перенаправляє його на сторінку авторизації Google. У цьому запиті передаються такі параметри, як client_id, redirect_uri, область доступу та тип відповіді. Після підтвердження доступу користувачем, Google перенаправляє його назад із кодом авторизації, який далі використовується для отримання токенів доступу та ідентифікації.

```
auth_url = (
            f"https://accounts.google.com/o/oauth2/v2/auth"
            f"?client_id={client_id}"
            f"&redirect_uri={redirect_uri}"
            f"&response_type={response_type}"
            f"&scope={scope}"
)
return redirect(auth_url)
```

Отримавши код, сервер надсилає запит до Google з метою обміну цього коду на access та id токени. Після успішної відповіді відбувається витяг основної інформації про користувача — передусім електронної пошти, яка слугує унікальним ідентифікатором. Якщо в базі вже існує користувач із такою поштою, він авторизується; якщо ні — створюється новий обліковий запис із відповідним ім’ям та поштою. Далі формуються стандартні JWT доступу, які передаються на фронтенд через URL.

5 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

5.1 Аналіз отриманих результатів

У процесі розробки серверної частини системи було реалізовано основний функціонал, який забезпечує отримання, обробку та збереження даних відповідно до вимог проєкту. Реалізовані модулі авторизації, керування профілями, система друзів, маркетплейс, ігрова статистика та сповіщення, всі вони функціонують у відповідності до заданих технічних специфікацій. Серверна частина забезпечує стабільну обробку запитів користувачів. Модуль маркетплейсу демонструє коректну роботу транзакцій із оновленням стану товарів і віртуальної валюти, що сприяє створенню живої економіки в межах системи.

5.2 Порівняння результатів із початковими вимогами

Порівнюючи отримані результати з початковими вимогами, можна сказати, що більшість функціональних завдань була виконана відповідно до поставлених цілей. Модулі авторизації та управління профілями реалізовані повністю, забезпечуючи надійний та безпечний доступ до системи. Взаємодія між користувачами через систему друзів і сповіщень відповідає очікуванням за функціональністю та зручністю використання. Маркетплейс відповідає вимогам щодо операцій з віртуальними товарами та транзакціями, з належним контролем цілісності даних.

Однак, серверна частина має суттєві обмеження при роботі з великою кількістю користувачів та великими обсягами даних.

5.3 Оцінка якості роботи системи

Якість роботи серверної частини оцінюється як висока. Система стабільно виконує основні функції, не допускаючи втрат даних чи збоїв при типовому і помірному навантаженні. Реалізація синхронної обробки запитів дозволяє забезпечити коректність стану гри і взаємодії користувачів у реальному часі. Впроваджені механізми безпеки, зокрема використання JWT для аутентифікації та

захист від типових веб-загроз, підвищують довіру до системи і захищають користувацькі дані.

5.4 Аналіз ефективності прийнятих рішень

Вибрані підходи до розробки серверної частини виявилися ефективними для реалізації поставлених завдань. Використання транзакцій у роботі з базою даних гарантувало відсутність неконсистентності під час одночасних операцій з користувацькими предметами і віртуальною валютою.

Використання JWT у процесі аутентифікації забезпечило гнучкість і безпеку, одночасно зменшуючи навантаження на сервер за рахунок відсутності необхідності зберігати сесійні дані. Це позитивно вплинуло на продуктивність системи та знизило ризики витоку особистих даних.

Застосування Django ORM спростило роботу з базою даних, підвищило ефективність розробки та підтримки коду, а також дозволило легко реалізувати складні операції з даними.

Таким чином, впроваджені технічні рішення створили надійну та платформу, яка відповідає сучасним вимогам і забезпечує можливості для подальшого розвитку.

5.4 Обмеження та недоліки

Незважаючи на позитивні результати, розроблена серверна частина має певні обмеження та недоліки. Відсутність повноцінної системи моніторингу в реальному часі та інструментів аналітики ускладнює оперативне виявлення та локалізацію проблем під час експлуатації, особливо при зростанні навантаження.

На даний момент система не здатна ефективно обробляти великі обсяги даних і значну кількість одночасних користувачів. Відсутність механізмів кешування призводить до затримок та блокувань у роботі, що негативно впливає на продуктивність серверної частини.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1 Впровадження серверної частини

Для впровадження серверної частини програмної системи «MonopolyUA», розробленої на Django, спочатку було здійснено клонування репозиторію з GitHub, що дозволило отримати актуальний код проєкту на комп’ютер. Після цього відбувся перехід до директорії проєкту для подальшої роботи з кодом.

Для забезпечення ізольованого середовища розробки і розгортання застосовано технологію контейнеризації Docker. Було створено файл Dockerfile, в якому послідовно описані дії для налаштування запуску серверної частини Django.

Для спрощення одночасного керування кількома сервісами, сервером застосунку та базою даних PostgreSQL, використано Docker Compose. Конфігурація Docker Compose містить параметри, які забезпечують взаємодію контейнерів між собою через налаштовану мережу, визначення змінних оточення для коректного підключення до бази даних і встановлення секретних ключів, прокидання портів для доступу до сервера ззовні та монтування томів для збереження даних і коду, що дозволяє оновлювати застосунок без необхідності повторної збірки образу.

Перед першим запуском було виконано міграції бази даних за допомогою стандартних команд Django, що дозволило створити необхідні таблиці в PostgreSQL відповідно до розроблених моделей програмної системи.

Після налаштування контейнерів і застосування міграцій серверна частина була розгорнута у Docker-середовищі. Такий підхід забезпечив швидке і надійне розгортання застосунку у ізольованому середовищі, що спростило процес підтримки та оновлення системи, знизило ризики виникнення помилок під час впровадження і спростило масштабування застосуноку у майбутньому.

ВИСНОВКИ

У процесі виконання курсового проєкту була успішно розроблена серверна частина програмної системи «MonopolyUA», яка реалізує основний функціонал. Створена система забезпечує стабільну та безпечну взаємодію користувачів із платформою, включаючи механізми авторизації та реєстрації, управління профілями, організацію соціальних зв’язків через систему друзів та сповіщень. Особлива увага приділена збереженню ігрової статистики, що дозволяє відстежувати активність користувачів, їхні досягнення та результати ігор, а також формувати рейтинги. Важливою частиною проєкту є реалізація маркетплейсу, який підтримує операції купівлі-продажу ігрових товарів.

Архітектура серверної частини побудована на основі фреймворку Django з використанням модульного підходу, що сприяє чіткому розподілу відповідальності між різними частинами системи, полегшує тестування, підтримку та подальший розвиток застосунку.

Для збереження даних використовується реляційна база даних PostgreSQL, що гарантує надійність та цілісність інформації [10]. Уся взаємодія клієнта із сервером відбувається через REST API [6], що забезпечує зручний та ефективний обмін даними.

В ході розробки були впроваджені важливі алгоритмічні рішення, які забезпечують коректність і надійність роботи системи, зокрема реалізовано транзакції при обробці критичних операцій, що виключає ризики неконсистентності даних. Алгоритми обробки відкриття кейсів, купівлі предметів на маркетплейсі, відновлення пароля та оновлення ігрової статистики працюють стабільно та ефективно.

Розроблена серверна частина забезпечує безпечну та ефетивну роботу програмної системи і служить надійною основою для подальшого розвитку та впровадження нового функціоналу.

GitHub репозиторій: https://github.com/NureDukhotaIvan/MonopolyUA

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Online Boards Game Overview [Електронний ресурс] – URL: https://www.marketresearchfuture.com/reports/online-board-games-market-43034 (дата звернення: 10.06.2025)

2. Tabletopia. [Електронний ресурс] – URL: https://tabletopia.com/ (дата звернення: 10.06.2025)

3. Board Game Arena. [Електронний ресурс] – URL: https://en.boardgamearena.com/faq (дата звернення: 10.06.2025)

4. Steam Community Market. What It Is And How To Use It [Електронний ресурс] – URL: https://www.lifewire.com/steam-community-market-what-it-is-and-how-to-use-it-4586933 (дата звернення: 10.06.2025)

5. Client Server Model. [Електронний ресурс] – URL: https://en.wikipedia.org/wiki/Client%E2%80%93server_model (дата звернення: 10.06.2025)

6. What Is a REST API? Examples, Uses and Challenges. [Електронний ресурс] – URL: https://blog.postman.com/rest-api-examples/ (дата звернення: 10.06.2025)

7. Django Applications. [Електронний ресурс] – URL: https://docs.djangoproject.com/en/5.2/ref/applications/ (дата звернення: 10.06.2025)

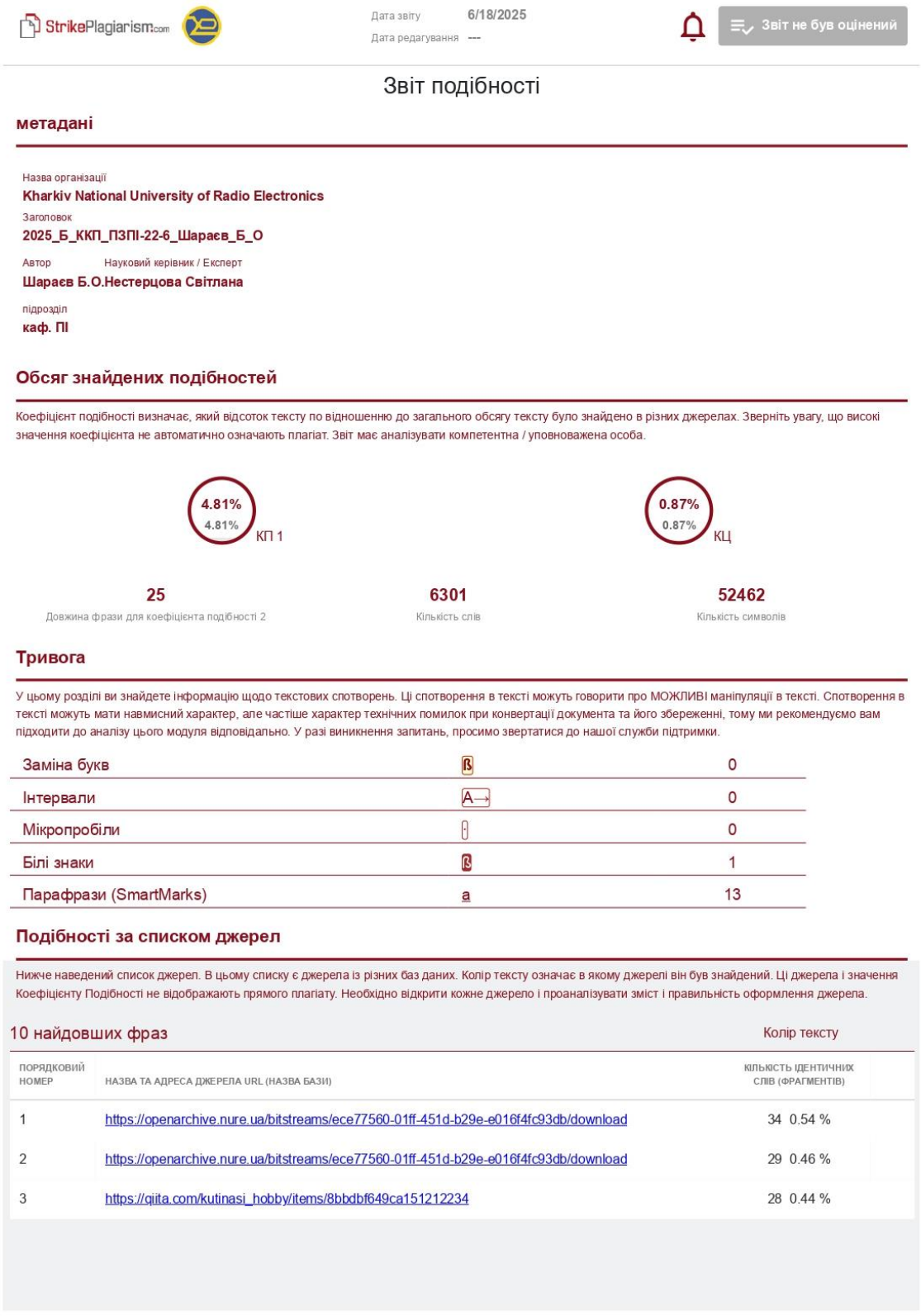
8. Django Project MVT Structure [Електронний ресурс] – URL: https://www.geeksforgeeks.org/python/django-project-mvt-structure/ (дата звернення: 10.06.2025)

9. Working with Django Model in Python: Best Practices [Електронний ресурс] – URL: https://djangostars.com/blog/django-models-best-practices/ (дата звернення: 10.06.2025)

10. Advantages of PostgreSQL. [Електронний ресурс] – URL: https://www.danytechcloud.com/advantages-of-postgresql/ (дата звернення: 10.06.2025)

Додаток А

Звіт результатів перевірки на унікальність тексту в базу ХНУРЕ



StrikePlagiarism.com

Дата звіту 6/18/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_Б_ККП_ПЗПІ-22-6_Шараєв_Б_О
Автор Науковий керівник / Експерт
Шараєв Б.О.Нестерцова Світлана
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

4.81% 4.81% КП 1

0.87% 0.87% КЦ

25	6301	52462
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		0
Інтервали		0
Мікропробіли		0
Білі знаки		1
Парафрази (SmartMarks)		13

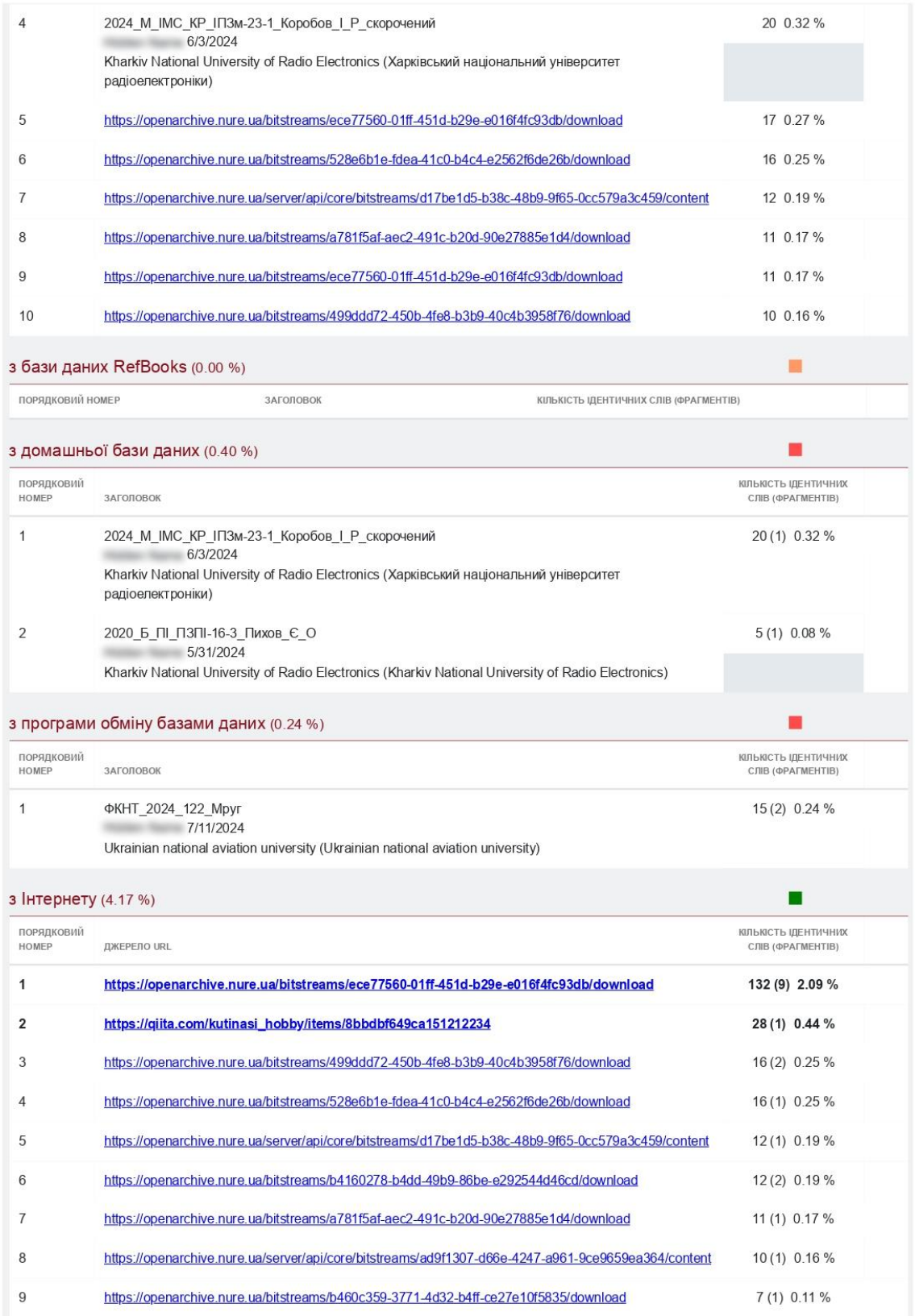
Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://openarchive.nure.ua/bitstreams/ece77560-01ff-451d-b29e-e016f4fc93db/download	34 0.54 %
2	https://openarchive.nure.ua/bitstreams/ece77560-01ff-451d-b29e-e016f4fc93db/download	29 0.46 %
3	https://qiita.com/kutinasi_hobby/items/8bbdbf649ca151212234	28 0.44 %

Рисунок А.1 – Перша сторінка результатів перевірки на унікальність тексту.



4	2024_M_IMC_KP_ІПЗм-23-1_Коробов_І_Р_скорочений [redacted] 6/3/2024 Kharkiv National University of Radio Electronics (Харківський національний університет радіоелектроніки)	20 0.32 %
5	https://openarchive.nure.ua/bitstreams/ece77560-01ff-451d-b29e-e016f4fc93db/download	17 0.27 %
6	https://openarchive.nure.ua/bitstreams/528e6b1e-fdea-41c0-b4c4-e2562f6de26b/download	16 0.25 %
7	https://openarchive.nure.ua/server/api/core/bitstreams/d17be1d5-b38c-48b9-9f65-0cc579a3c459/content	12 0.19 %
8	https://openarchive.nure.ua/bitstreams/a781f5af-aec2-491c-b20d-90e27885e1d4/download	11 0.17 %
9	https://openarchive.nure.ua/bitstreams/ece77560-01ff-451d-b29e-e016f4fc93db/download	11 0.17 %
10	https://openarchive.nure.ua/bitstreams/499ddd72-450b-4fe8-b3b9-40c4b3958f76/download	10 0.16 %

з бази даних RefBooks (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з домашньої бази даних (0.40 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	2024_M_IMC_KP_ІПЗм-23-1_Коробов_І_Р_скорочений [redacted] 6/3/2024 Kharkiv National University of Radio Electronics (Харківський національний університет радіоелектроніки)	20 (1) 0.32 %
2	2020_Б_ПІ_ПЗПІ-16-3_Пихов_Є_О [redacted] 5/31/2024 Kharkiv National University of Radio Electronics (Kharkiv National University of Radio Electronics)	5 (1) 0.08 %

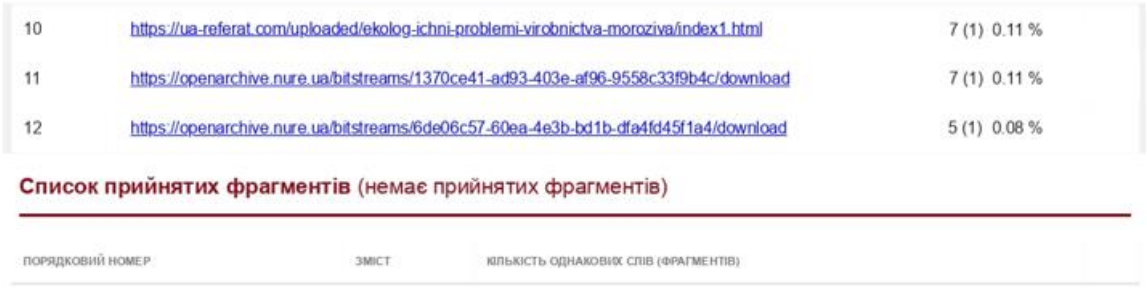
з програми обміну базами даних (0.24 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	ФКНТ_2024_122_Мруг [redacted] 7/11/2024 Ukrainian national aviation university (Ukrainian national aviation university)	15 (2) 0.24 %

з Інтернету (4.17 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**https://openarchive.nure.ua/bitstreams/ece77560-01ff-451d-b29e-e016f4fc93db/download**	**132 (9) 2.09 %**
2	**https://qiita.com/kutinasi_hobby/items/8bbdbf649ca151212234**	**28 (1) 0.44 %**
3	https://openarchive.nure.ua/bitstreams/499ddd72-450b-4fe8-b3b9-40c4b3958f76/download	16 (2) 0.25 %
4	https://openarchive.nure.ua/bitstreams/528e6b1e-fdea-41c0-b4c4-e2562f6de26b/download	16 (1) 0.25 %
5	https://openarchive.nure.ua/server/api/core/bitstreams/d17be1d5-b38c-48b9-9f65-0cc579a3c459/content	12 (1) 0.19 %
6	https://openarchive.nure.ua/bitstreams/b4160278-b4dd-49b9-86be-e292544d46cd/download	12 (2) 0.19 %
7	https://openarchive.nure.ua/bitstreams/a781f5af-aec2-491c-b20d-90e27885e1d4/download	11 (1) 0.17 %
8	https://openarchive.nure.ua/server/api/core/bitstreams/ad9f1307-d66e-4247-a961-9ce9659ea364/content	10 (1) 0.16 %
9	https://openarchive.nure.ua/bitstreams/b460c359-3771-4d32-b4ff-ce27e10f5835/download	7 (1) 0.11 %

Рисунок А.2 – Друга сторінка результатів перевірки на унікальність тексту.



10	https://ua-referat.com/uploaded/ekolog-ichni-problemi-virobnictva-moroziva/index1.html	7 (1) 0.11 %
11	https://openarchive.nure.ua/bitstreams/1370ce41-ad93-403e-af96-9558c33f9b4c/download	7 (1) 0.11 %
12	https://openarchive.nure.ua/bitstreams/6de06c57-60ea-4e3b-bd1b-dfa4fd45f1a4/download	5 (1) 0.08 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

Рисунок А.3 – Третя сторінка результатів перевірки на унікальність тексту.

Додаток Б

Слайди презентації

Веб-застосунок MonopolyUA

Виконав:
ст. гр. ПЗПІ-22-6
Шараєв Б. О.

Керівник:
ст. вик. каф. ПІ
Онищенко К. Г.

09 червня 2025 року

Рисунок Б.1 – Титульний слайд.

Мета роботи

Чітке визначення мети роботи:

Метою проєкту є розробка серверної синхронної частини для веб-застосунку «MonopolyUA», яка забезпечує стабільну, безпечну та ефективну взаємодію між користувачами. Система повинна підтримувати авторизацію, управління профілем, надсилання запрошень, механізми торгівлі ігровими предметами, а також зберігання й обробку ігрової статистики. Серверна частина має гарантувати цілісність даних, надійність виконання всіх операцій і бути готовою до подальшого розвитку функціоналу.

2

Рисунок Б.2 – Мета роботи.

Мета роботи

Актуальність:

Актуальність такого рішення сьогодні дуже висока. В умовах швидкого зростання індустрії онлайн-ігор користувачі очікують не лише цікавого ігрового процесу, а й зручної та функціональної платформи. Такі рішення роблять гру повноцінним середовищем для взаємодії, самовираження та розвитку, значно підвищуючи залученість користувачів.

Рисунок Б.3 – Актуальність програмого рішення.

Аналіз проблеми (аналізіснуючих рішень)

Перелік досліджених конкурентів:

- Tabletopia

Зручний профіль користувача з історією ігор, улюбленими іграми та списком друзів.

- Board Game Arena

Якісна взаємодія між користувачами: додавання друзів, групи, сповіщення.

- Steam

Просунутий інвентар і маркетплейс з унікальною ідентифікацією предметів, автоматизованими транзакціями та захистом.

Рисунок Б.4 – Аналіз існуючих рішень.

Аналіз проблеми (аналізіснуючихрішень)

Зазначення прогалин у наявних аналогах:

- Tabletopia

Відсутність детальної ігрової статистики, що ускладнює відстеження прогресу.

- Board Game Arena

Частина сповіщень надходить у застосунок, а інші на пошту. Відсутність деталізованої статистики про досягнення в зіграних іграх.

- Steam

Наявність комісії, що стягується за кожну операцію на маркеті.

Рисунок Б.5 – Зазначення прогалин у наявних аналогах.

Постановка задачі та опис системи

Чітке формулювання проблеми:

У процесі розробки серверної частини веб-застосунку MonopolyUA виникає низка критичних проблем. Однією з основних є необхідність забезпечення стабільної обробки великої кількості одночасних запитів, що надходять від користувачів, без втрати цілісності або конфліктів у даних. Другою важливою проблемою є захист персональних даних, зокрема реалізація безпечної аутентифікації та протидія поширеним загрозам, таким як SQL-ін’єкції, XSS і CSRF. Окрім цього, система повинна зберігати високу продуктивність і надійність при масштабуванні, тобто зі зростанням кількості користувачів не повинно відбуватися зниження швидкодії або збої в роботі застосунку.

Рисунок Б.6 – Проблема розроблюваної системи.

Постановка задачі та опис системи

Опис очікуваних результатів:

У результаті реалізації серверної частини веб-застосунку «MonopolyUA» очікується створення стабільної та безпечної платформи. Система повинна забезпечити надійні механізми реєстрації, авторизації та керування обліковими записами з дотриманням стандартів захисту персональних даних. Буде реалізовано модуль взаємодії між друзями з можливістю надсилання запрошень та підтвердження дружби. Маркетплейс забезпечуватиме безпечну торгівлю ігровими предметами між користувачами з оновленням інвентаря у режимі реального часу. Очікується також повноцінна система статистики, що зберігатиме результати ігрової активності кожного користувача.

Рисунок Б.7 – Опис очікуваних результатів.

Вибір технологій розробки

Інструментарій та технології, використані в роботі:

Для розробки бекенд-частини програмної веб-застосунку «MonopolyUA» було обрано фреймворк Django, який у поєднанні з Django REST Framework дозволяє ефективно створювати та підтримувати RESTful API. Для безпечної авторизації користувачів застосовано Simple JWT, що забезпечує роботу з токенами доступу, а також інтеграцію з Google OAuth2 для зручного входу через облікові записи Google. Як систему управління базою даних використано PostgreSQL, яка відома своєю стабільністю та масштабованістю.

Рисунок Б.8 – Інструментарій та технологіх, використані в роботі.

Архітектура створенного програмного забезпечення

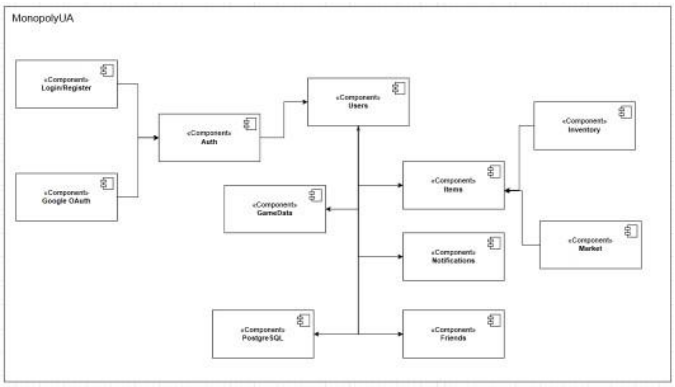


Рисунок Б.9 – Схема компонентів серверної частини.

Архітектура створенного програмного забезпечення

Опис ключових компонентів:
Auth — модуль автентифікації та авторизації користувачів; підтримує реєстрацію, класичний вхід і Google OAuth.
Users — відповідає за зберігання та редагування даних профілю користувача.
GameData — зберігає та обробляє статистику ігор, результати та рейтинги користувачів.
Items — відповідає за інвентар та маркетплейс (торгівля між користувачами, створення лотів, купівля/продаж).
Notifications — модуль сповіщень про запрошення та системні події.
Friends — забезпечує функціонал додавання, прийняття та керування списком друзів.

Рисунок Б.10 – Опис компонентів серверної частини.

Опис програмного забезпечення, що було використано у дослідженні

Опис процесу розробки:

Для розробки серверної частини веб-застосунку «MonopolyUA» було обрано фреймворк Django та архітектурний патерн MVT у межах клієнт-серверної архітектури. Спочатку було створено основні компоненти — моделі, представлення та серіалізатори для реалізації функціоналуавторизації та реєстрації користувачів. Наступним кроком була розробка профілів користувачів із можливістю збереження та редагування персональних даних. Потім було реалізовано систему друзів із логікою взаємодії між користувачами, а також модуль сповіщень для інформування про важливі події. Далі було створено маркетплейс із моделями товарів, пропозицій, інвентар і транзакцій для торгівлі ігровими предметами. Після цього було розроблено модуль збереження інформації про ігрові сесії, результатита статистику.

Рисунок Б.11 – Опис процесу розробки.

Опис програмного забезпечення, що було використано у дослідженні

Вибрані мови програмування та фреймворки:

- Python – основна мова програмування для реалізації логіки серверної частини.
- Django – високорівневий веб -фреймворк для побудови структури застосунку та взаємодії з базою даних.
- Django REST Framework – розширення для Django, що дозволяє створювати RESTful API.

Рисунок Б.12 – Вибрані мови програмування та фреймворки.

Приклад реалізації

Фрагменти коду з відкриттям кейсу

```python
class OpenCaseAPIView(APIView):
    permission_classes = [IsAuthenticated]
    @transaction.atomic
    def post(self, request):
        case_id = request.data.get("case_id")
        case = get_object_or_404(Item, id=case_id, category=Item.CATEGORY_CASE)
        inv_case = InventoryItem.objects.get(user=request.user, item=case)
        if inv_case.quantity <= 1:
            inv_case.delete()
        else:
            inv_case.quantity -= 1
            inv_case.save()
        contents = CaseItemContent.objects.filter(case=case).select_related('item')
        if not contents.exists():
            return Response({'detail': 'Кейс пустий'}, status=status.HTTP_400_BAD_REQUEST)

        rarity_counts = Counter()
        for content in contents:
            rarity_counts[content.item.rarity] += 1

        items = []
        weights = []
```

```python
    for content in contents:
        rarity = content.item.rarity
        if rarity in RARITY_WEIGHTS and rarity_counts[rarity] > 0:
            items.append(content.item)
            weight_per_item = RARITY_WEIGHTS[rarity] / rarity_counts[rarity]
            weights.append(weight_per_item)

    if not items:
        return Response({'detail': 'Предмети відсутні.'}, status=status.HTTP_400_BAD_REQUEST)

    selected_item = random.choices(items, weights=weights, k=1)[0]
    inventory_item, created = InventoryItem.objects.get_or_create(
        user=request.user,
        item=selected_item,
        defaults={'quantity': 1}
    )
    if not created:
        inventory_item.quantity += 1
        inventory_item.save()

    serializer = ItemSerializer(selected_item, context={'request': request})
    return Response(serializer.data, status=status.HTTP_200_OK)
```


Рисунок Б.13 – Фрагменти коду з відкриття кейсу.

Приклад реалізації

Фрагменти коду з купівлею багатьох товарів на маркеті

```python
class BuyMultipleItemAPIView(APIView):
    permission_classes = [IsAuthenticated]
    @transaction.atomic
    def post(self, request, item_id):
        item = get_object_or_404(Item, id=item_id)
        try:
            user_price = request.data.get('user_price')
            user_quantity = int(request.data.get('user_quantity'))
            if user_quantity <= 0 or user_price <= 0:
                raise ValueError
        except (TypeError, ValueError):
            return Response({'error': 'Неправильна ціна або кількість'}, status=status.HTTP_400_BAD_REQUEST)

        buyer = request.user
        total_spent = 0
        total_bought = 0
        market_listings = MarketListing.objects.filter(
            item=item,
            user_price__lte=user_price
        ).exclude(
            seller=buyer
        ).order_by('user_price')

        total_available = market_listings.count()
        full_listing = MarketListing.objects.filter(item=item).exclude(seller=buyer).count()
```

Рисунок Б.14 – Фрагменти коду з купівлею багатьох товарів на маркеті.

Приклад реалізації

Фрагменти коду з відновлення паролів

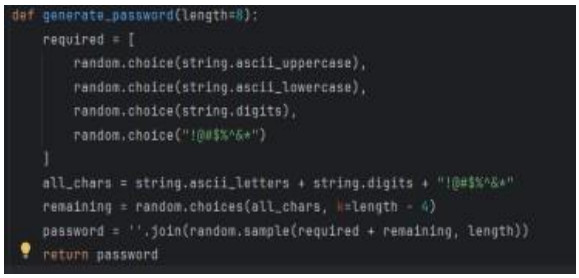


Рисунок Б.15 – Фрагменти коду з відновлення паролів.

Підсумки

У процесі виконання курсового проєкту була успішно створена серверна частина веб-застосунку «MonopolyUA», яка реалізує ключовий функціонал системи. Зокрема, було розроблено модулі аутентифікації та управління профілями користувачів, що забезпечують безпечний та зручний доступ до сервісів платформи. Також реалізовано систему взаємодії між користувачами, яка включає функціонал додавання друзів і сповіщень, що сприяє кращій комунікації та підсилює соціальну складову проєкту.

Окрему увагу приділено збереженню ігрової статистики, що дозволяє фіксувати результати ігор, формувати рейтинг та відкриває можливості для подальшого аналізу даних. Крім того, було реалізовано маркетплейс, який дозволяє користувачам здійснювати операції з віртуальними товарами, що є важливою частиною ігрової економіки.

Архітектура проєкту побудована на базі фреймворку Django з використанням модульного підходу, що дозволило логічно структурувати код і чітко розділити відповідальність між окремими частинами системи. Такий підхід спрощує подальший розвиток застосунку та його підтримку.

У результаті розробки створено надійну й функціональну систему, яка демонструє стабільну роботу, забезпечує необхідний рівень безпеки і є якісною основою для подальшого вдосконалення проєкту.

Рисунок Б.16 – Підсумки.

Додаток В

SRS

1. Introduction

1.1 Purpose

Цей документ є специфікацією вимог до програми (SRS) для веб-застосунку «MonopolyUA». Головна ціль документу це визначення, що повинна виконувати система, щоб розробники могли перейти до проєктування та реалізації.

Аудиторія:

- розробники backend частини;
- розробник frontend частини.

1.2 Scope

Продукт: веб-застосунок «MonopolyUA».

Що робить:

- реєстрація/авторизація користувачів (JWT, Google Auth);
- головна сторінка, профілю, маркету, створення гри, дошки для гри;
- рейтингові таблиці та топ-гравці;
- створення/приєднання до ігрового лобі;
- відображення динамічної дошки гри з ходами, оплатами та купівлею.
- чат у реальному часі;
- обробка онлайн-статусів та AFK.

Що не робить:

- не націлена на мобільні застосунки;
- не підтримує інтеграцію реальних грошей.

Бізнес-мета: створити інтерактивну онлайн-версію «Монополії» з соціальними компонентами та транзакціями між гравцями.

1.3 Definitions, Acronyms and Abbreviations

- HTML / CSS / JS - Технології фронтенду;
- Django - Python фреймворк;
- DRF - Django REST Framework;
- Channels - Django Channels для WebSocket;
- Daphne - ASGI-сервер;
- Redis - in-memory datastore;
- PostgreSQL - Реляційна СУБД;
- Asyncio - Асинхронна бібліотека Python;
- UUID - Унікальний ідентифікатор об’єкта;
- JWT - JSON Web Token;
- CORS - Cross-Origin Resource Sharing;
- Simple JWT - Django-бібліотека для JWT;
- django-redis - Бекенд кешування через Redis.

1.4 References

- RFC 7519 — JSON Web Token;
- Django 5.1 documentation;
- Django REST Framework docs;
- Django Channels docs + ASGI/Redis integration;
- PostgreSQL 15 docs;
- Relevant articles on WebSocket-based real-time games.

1.5 Overview

- Розділ 2: загальний опис системи та її контекст;
- Розділ 3: функціональні та нефункціональні вимоги;
- Розділ 4: сценарії використання;
- Розділ 5-6: зовнішні інтерфейси, інші вимоги.

2 Overall Description

2.1 Product Perspective

Веб-застосунок «MonopolyUA» є незалежною програмною системою, що працює автономно у вигляді повнофункціонального веб-клієнта та серверної частини з підтримкою реального часу. Його основне призначення — забезпечити користувачам інтерактивне середовище для гри у цифрову версію настільної гри «Монополія», орієнтовану на український культурний контекст. Програма не є компонентом більшого корпоративного рішення, однак вона побудована з використанням стандартних веб-протоколів і може масштабуватись або інтегруватись у ширші системи (наприклад, з платформами рейтингів чи сторонніми сервісами для зберігання профілів).

Застосунок включає фронтенд-частину, реалізовану засобами HTML, CSS та JavaScript без використання фреймворків, та бекенд-частину, розроблену на базі Django 5.1 із поділом на синхронні REST API та асинхронну логіку на WebSocket через Django Channels. Основним джерелом збереження даних виступає PostgreSQL, а Redis відіграє роль як кеш-сервера, так і каналу повідомлень. Завдяки цьому архітектура дозволяє реалізувати повноцінну багатокористувацьку гру в реальному часі з інтегрованим чатом та постійною передачею станів гри.

2.1.1 System Interfaces

Система взаємодіє з кількома зовнішніми службами та протоколами, необхідними для її роботи. Однією з ключових інтеграцій є Google OAuth 2.0, що використовується для авторизації користувачів за допомогою облікових записів Google. Це забезпечує безпечний і зручний доступ без потреби вводити пароль. Додатково, для відновлення пароля через email використовується SMTP-сервер, налаштований для надсилання листів із посиланнями на зміну пароля.

Ще одним важливим елементом є Redis, який, окрім кешування, використовується для організації обміну повідомленнями між клієнтом і сервером за допомогою Pub/Sub моделі. Цей механізм критично важливий для роботи чату, ігрових ходів і системи статусів онлайн/офлайн. Сама система також може бути

інтегрована з моніторинговими сервісами або CI/CD пайплайнами для автоматизації розгортання.

2.1.2 Interfaces

Інтерфейс користувача системи представлений у вигляді традиційного багатосторінкового веб-сайту, де всі елементи візуального відображення реалізовані засобами HTML, CSS та JavaScript. Головна мета інтерфейсу — забезпечити простоту навігації, інтуїтивне керування та візуальну ідентичність, притаманну грі «Монополія». Усі основні розділи — профіль, лобі, ігрове поле, магазин — доступні з головного меню, а реакції на дії користувача реалізовані без перевантаження сторінки, з допомогою динамічної взаємодії з API. Інтерфейс також враховує основи доступності: використання контрастних кольорів, альтернативного тексту для зображень.

2.1.3 Hardware Interfaces

Веб-застосунок «MonopolyUA» не має прямої взаємодії з апаратним забезпеченням. Усі процеси виконуються у віртуальному або хмарному середовищі, тому вимоги до апаратної частини мінімальні та стандартні для типових серверів. Система не контролює зовнішні пристрої, як-от сканери, сенсори чи інші периферійні модулі. Усі компоненти взаємодіють через програмні інтерфейси, і будь-яке апаратне забезпечення розглядається лише як середовище виконання.

2.1.4 Software Interfaces

Програмна система «MonopolyUA» побудована на низці зовнішніх бібліотек та платформ. Бекенд реалізовано з використанням фреймворку Django (версія 5.1), що забезпечує базову структуру, маршрутизацію та взаємодію з базою даних PostgreSQL. Для створення REST API використовується Django REST Framework. Авторизація реалізована з допомогою бібліотеки Simple JWT, яка надає функціональність створення, оновлення та перевірки токенів доступу.

Асинхронна частина системи базується на Django Channels, що працює поверх ASGI-сервера Daphne, а Redis служить як кеш і транспортний канал. Частина клієнтської частини системи взаємодіє з сервером через WebSocket-протокол, що забезпечує двосторонню комунікацію в реальному часі, необхідну для гри, чату та статусів гравців.

2.1.5 Communications Interfaces

Застосунок використовує кілька типів комунікаційних інтерфейсів. Основна взаємодія між клієнтом і сервером здійснюється через HTTP, що забезпечує захищене з'єднання з REST API. Для обміну повідомленнями в режимі реального часу система використовує WebSocket-з’єднання, яке працює через ASGI та Redis. Крім цього, система підтримує CORS, що дозволяє безпечний доступ до API з фронтенду, розгорнутого на окремому домені.

2.1.6 Memory Constraints

У системи немає жорстко встановлених обмежень на об’єм оперативної пам’яті, проте з огляду на використання Redis та розподіленого оброблення даних, мінімальна рекомендована конфігурація для сервера передбачає не менше 2 GB RAM. Це дозволяє ефективно підтримувати декілька одночасних ігрових сесій, чат та кешування без значного навантаження. Під час масштабування рекомендується дотримуватись балансу між доступною пам’яттю Redis і кількістю активних WebSocket-підключень.

2.1.7 Operations

Система працює у постійному інтерактивному режимі. Більшість дій користувачів відбуваються у реальному часі, з мінімальним затримками та без необхідності оновлення сторінки. Адміністративна панель передбачає базові функції моніторингу, а також можливість автоматичного перезапуску сесій гри у разі збоїв. Для збереження стабільної роботи системи рекомендовано збереження даних про завершені ігри.

2.1.8 Site Adaptation Requirements

Перед початком роботи з системою необхідно виконати кілька підготовчих кроків, зокрема розгортання інфраструктури з встановленням Redis, PostgreSQL, Django-серверу та необхідних залежностей. У випадку хостингу в хмарному середовищі (наприклад, AWS або DigitalOcean), також слід враховувати налаштування SSL-сертифікатів і проксісерверів. Система може бути адаптована до конкретних умов інсталяції шляхом налаштування конфігураційних файлів середовища, а також вибору відповідного плану масштабування залежно від кількості користувачів.

2.2 Product Functions

Основне функціональне призначення веб-застосунку «MonopolyUA» полягає у забезпеченні гравцям повноцінного онлайн-досвіду, наближеного до класичної настільної гри «Монополія», із розширеннями, адаптованими до цифрового формату. Програма повинна надавати користувачу повний цикл взаємодії — від реєстрації та входу до участі в динамічному ігровому процесі з іншими гравцями, управління власним профілем, обміну предметами та спілкування у реальному часі.

Першочерговою функцією системи є створення безпечного механізму автентифікації, що реалізується за допомогою власного інтерфейсу логіна з JWT-токенами та авторизації через Google. Сюди ж входить функція відновлення паролю, яка здійснюється через email. Після входу користувач має змогу керувати власним профілем: змінювати особисті дані, нікнейм, аватарку, переглядати ігрову статистику та керувати друзями.

Другою ключовою підсистемою є особистий кабінет гравця. Він включає інвентар з предметами (скінами), які можна застосовувати в грі для персоналізації, продавати та у деяких випадках відкривати, якщо тип предмету – кейс. Також існує система друзів, яка дозволяє додавати, підтверджувати або видаляти. Це посилює соціальний аспект проєкту.

Третій функціональний блок — це ігровий маркет, що дозволяє купівлю предметів інших користувачів. Усі транзакції супроводжуються перевіркою балансу, зняттям коштів та оновленням інвентаря. Додатково реалізована система кейсів із випадковими винагородами, яка стимулює активність у грі.

Найважливішим ядром застосунку є сам ігровий процес. Користувачі можуть створювати або приєднуватись до ігрових лобі, в яких вони налаштовують параметри майбутньої гри. Кожна сесія містить повноцінне ігрове поле з елементами «Монополії»: гральними кубиками, об’єктами, грошима, штрафами та ін. Ігрова логіка реалізована через WebSocket, що дозволяє всім гравцям бачити зміни в реальному часі, включаючи чат, таймери та зміни стану гри.

Крім того, система виконує функції збору статистики, підрахунку перемог і формування рейтингових списків гравців. Ці рейтинги впливають на підбір суперників у майбутніх матчах, а також дозволяють реалізовувати змагання та нагороди для найактивніших учасників.

Усі ці функції пов’язані між собою логічно та реалізовані через взаємодію фронтенд-компонентів і бекенд-служб у режимі реального часу або через API-запити, що забезпечує динамічну і плавну роботу застосунку.

2.3 User Characteristics

Цільова аудиторія застосунку «MonopolyUA» складається з широкого кола користувачів, основну частину яких формують підлітки, молодь та дорослі віком від 14 до 35 років. Більшість користувачів мають базовий або середній рівень комп’ютерної грамотності, тому система повинна бути простою, інтуїтивно зрозумілою та доступною без потреби проходження навчання або читання інструкцій.

Користувачі знайомі з класичними браузерними іграми, соціальними мережами та мобільними додатками, що формує очікування до швидкого відгуку, інтерактивності інтерфейсу та соціальної взаємодії. З огляду на це, інтерфейс застосунку повинен мати мінімалістичний, адаптивний дизайн, бути зрозумілим

навіть без текстових підказок і працювати стабільно на більшості сучасних браузерів.

З огляду на наявність функцій для гри в реальному часі, чатів і рейтингових систем, очікується, що користувачі мають досвід участі в багатокористувацьких онлайн-іграх або соціальних платформах. Це дозволяє припустити достатній рівень комфортності з механіками типу «додати у друзі», «запросити в лобі», «вийти з гри», «перевірити інвентар» тощо.

Водночас розробка враховує й менш досвідчених користувачів, тому реалізована система підказок, автоматичних повідомлень про хід гри, а також механізм обробки AFK-ситуацій, щоб уникнути збоїв у ігровому процесі. Усі ці аспекти враховуються в дизайні інтерфейсів і логіці бекенду з метою створення комфортного та збалансованого досвіду для всіх користувачів.

2.4 Constraints

В процесі розробки веб-гри «MonopolyUA» необхідно врахувати низку обмежень, що визначають технічні, організаційні та регуляторні рамки проєкту:

а) регуляторні політики та захист даних – система обробляє персональні дані користувачів (електронна пошта, аватар, історія ігор), тому розробка та експлуатація повинні відповідати вимогам GDPR (ЄС), Закону «Про захист персональних даних» (Україна) та аналогічних регуляцій. Надсилання паролів і повідомлень через електронну пошту реалізується з використанням TLS-з’єднання та захищеного SMTP-серверу;

б) інтеграція з зовнішніми інтерфейсами:

1) OAuth 2.0 для Google Sign-In;

2) SMTP-сервер для відновлення паролю та системних сповіщень.

Всі API-виклики до сторонніх сервісів повинні відбуватися через захищені HTTP-канали з перевіркою сертифікату.

в) паралельна робота та висока доступність – система підтримує одночасні ігрові сесії та кілька WebSocket-з’єднань на одного користувача. Використання

Redis для розподіленого зберігання станів лобі та ігор забезпечує горизонтальне масштабування й синхронну роботу кількох інстансів сервера.

г) аудит і логування - усі ключові події (створення лобі, приєднання/вихід гравця, хід гравця, фінансові транзакції, банкрутство тощо) фіксуються в хронологічних лістах Redis та можуть бути експортовані для подальшого аналізу. Журнали також відіграють роль розробницького та експлуатаційного аудиту.

д) контрольні та безпекові функції:

1) CSRF-захист для REST-інтерфейсу;

2) валідація та очищення всіх вхідних даних, включно з повідомленнями чату;

3) захист WebSocket-каналів через JWT-мідлвера із обмеженим часом життя токена.

е) критичність додатку – хоча «MonopolyUA» не є системою із життєво критичними операціями, користувачі очікують безперебійної та чуйної роботи гри. Будь-яка втрата стану партії може призвести до розчарування та відтоку гравців;

ж) мовні та платформові вимоги – розробка виконана на Python 3.8+ з використанням Django 5.1 і Django Channels. Для клієнтської частини застосовано сучасні версії JavaScript ES6+. Будь-які сторонні бібліотеки повинні бути з відкритим кодом та підтримуватися спільнотою.

Ці обмеження визначають рамки вибору інструментів, архітектурних рішень та процедур розробки, усуваючи ризики невідповідності регламентам, гарантуючи безпеку й стабільність системи.

2.5 Assumption and Dependencies

Реалізація функціоналу веб-застосунку «MonopolyUA» ґрунтується на низці припущень і зовнішніх залежностей, які безпосередньо впливають на виконання вимог, описаних у цьому документі. У разі зміни цих припущень — вимоги можуть потребувати перегляду.

- вся логіка зберігання стану гри, таймерів, хронології подій та менеджера ходів залежить від стабільної роботи Redis-сервера. У разі зміни

архітектури з Redis на інше in-memory сховище (наприклад, Memcached або PostgreSQL Pub/Sub), вимоги щодо продуктивності та синхронізації повинні бути оновлені;

- передбачається, що серверна частина продовжуватиме функціонувати в середовищі Django + Channels. Заміна цих технологій (наприклад, на FastAPI або Node.js) потребуватиме переосмислення архітектури WebSocket-комунікацій, мідлверів, а також механізмів автентифікації;
- уся ідентифікація користувачів у WebSocket-з’єднаннях базується на JWT-токенах. Якщо зміниться метод автентифікації (наприклад, перехід на session-based або OAuth-only), буде потрібно оновити логіку обробки scope у WS-підключеннях;
- механізм відновлення паролю залежить від можливості відправити лист користувачу з новим згенерованим паролем. У разі недоступності поштового сервісу функціональність скидання пароля буде порушено, що вимагає зміни SRS (наприклад, через перехід до верифікаційних токенів);
- реалізація сторонньої автентифікації залежить від працездатності Google API. У разі зміни політики OAuth або недоступності Google Identity Platform — необхідна адаптація системи (наприклад, через підтримку інших провайдерів або fallback на внутрішню автентифікацію);
- система продажу/купівлі скінів ґрунтується на внутрішньому ігровому балансі користувача. Припускається, що не планується підключення реальних платіжних систем. Якщо таке припущення зміниться, SRS доведеться переглянути з урахуванням фінансових і юридичних вимог (зокрема, підтримки реальних грошей, податків і безпеки транзакцій);
- передбачається, що користувачі взаємодіють із грою через сучасні браузери, які підтримують WebSocket, localStorage та інші сучасні веб-технології. У разі зміни цільової платформи (наприклад, підтримка мобільного додатку або застарілих браузерів), інтерфейс та архітектура клієнта мають бути адаптовані.

Ці припущення є фундаментом для визначення поточних вимог. У разі їх зміни функціональні або нефункціональні вимоги до системи можуть потребувати перегляду, адаптації або переосмислення.

2.6 Apportioning of Requirements

У процесі планування розробки системи «MonopolyUA» було виявлено, що деякі вимоги, хоча й важливі, можуть бути реалізовані на подальших етапах через обмеження часу, ресурсів або складність реалізації. Враховуючи обсяг проекту, розробка розбивається на кілька ітерацій, де пріоритетними є функції, критичні для базової роботи системи.

До першої ітерації включено:

- реалізація основної гри з класичними правилами монополії;
- підтримка WebSocket-з’єднання для гри в реальному часі;
- механізм черги ходів із таймерами;
- базова система логів і чату;
- авторизація (звичайна + Google OAuth);
- профіль користувача з переглядом статистики та редагуванням даних;
- базова система друзів і повідомлень;
- система інвентарю та торгівлі скінів за внутрішню валюту;
- застосування скінів до придбаних власностей під час гри.

До наступних ітерацій (можуть бути реалізовані в майбутньому):

- розширена система анімацій на полі гри;
- візуальна мапа всіх власностей і скінів гравців у поточній сесії;
- повноцінна мобільна версія інтерфейсу (адаптація для touch-інтерфейсів);
- доопрацювання торгової системи з можливістю створення аукціонів або обміну між гравцями;
- введення нагород, досягнень та бойових пропусків;
- створення системи репортів/скарг на гравців;
- статистика по іграх друзів;

- публічні/приватні лобі з паролем і фільтрами;
- можливість створювати кастомні правила гри (варіативність правил).

Пріоритетність реалізації функцій у майбутніх ітераціях визначатиметься відповідно до відгуків користувачів, технічних можливостей та стратегічних цілей проєкту. Рішення про відкладення реалізації певних вимог приймалося у співпраці з замовником і командою розробки.

3 Specific Requirements

3.1 External Interfaces

3.1.1 Інтерфейс автентифікації користувача:

а) опис: Забезпечує реєстрацію, вхід, авторизацію через Google (OAuth) та відновлення паролю;

б) джерело введення: Користувач через форму входу/реєстрації;

в) призначення виводу: Сервер автентифікації;

г) допустимі значення:

1) електронна пошта: у форматі, що відповідає стандарту RFC 5322;

2) пароль: від 6 до 128 символів, щонайменше одна літера та одна цифра.

д) одиниці виміру: Текстові поля (рядки);

е) часова характеристика: Під час взаємодії з формою входу або реєстрації;

ж) зв’язки з іншими інтерфейсами: Профіль користувача, сесії гри.

з) формат вікна: Окремі форми входу та реєстрації;

и) формат даних: JSON;

к) фінальні повідомлення: “Вхід успішний”, “Невірний пароль”, “Користувача не знайдено”.

3.1.2 Інтерфейс профілю користувача:

а) опис: Дозволяє переглядати та редагувати нікнейм, аватар, пароль і переглядати статистику ігор;

б) джерело введення: Користувач на сторінці профілю;

в) призначення виводу: База даних користувачів;

г) допустимі значення:

1) нікнейм: 3–20 символів, тільки латинські літери, цифри, символи "_", "-";

2) Зображення: .png, .jpg, розмір до 5MB.

д) одиниці виміру: Рядки, зображення;

е) часова характеристика: Після входу до профілю;

ж) зв’язки з іншими інтерфейсами: Аутентифікація;

з) формат вікна: Вкладка "Профіль";

и) формат даних: JSON або multipart/form-data;

к) фінальні повідомлення: “Профіль оновлено”, “Файл занадто великий”.

3.1.3 Інтерфейс лобі гри:

а) опис: Дозволяє створювати ігрові кімнати або приєднуватися до них;

б) джерело введення: Користувач із головного меню або за посиланням-запрошенням;

в) призначення виводу: Менеджер сесій гри;

г) допустимі значення:

1) назва кімнати: 3–30 символів;

2) кількість гравців: від 2 до 4.

д) одиниці виміру: Рядки, числа;

е) часова характеристика: До початку гри;

ж) зв’язки з іншими інтерфейсами: Список друзів, повідомлення;

з) формат вікна: Список доступних кімнат або вікно створення;

и) формат даних: JSON;

к) фінальні повідомлення: “Кімната створена”, “Гравець приєднався”, “Гру розпочато”.

3.1.4 Інтерфейс самої гри:

а) опис: Головне ігрове поле, чат, лог ходів, панель гравця, менеджер ходу;

б) джерело введення: Дії користувача під час гри;

в) призначення виводу: Сервер гри через WebSocket;

г) допустимі значення: Команди: "кинути кубики", "купити", "пропустити", "передати", "взяти заставу" тощо.

д) одиниці виміру: Команди, повідомлення;

е) часова характеристика: У режимі реального часу;

ж) зв’язки з іншими інтерфейсами: Менеджер черги, чат, профіль;

з) формат вікна: Ігрове поле з боковою панеллю дій;

и) формат даних: WebSocket (JSON);

к) фінальні повідомлення: “Хід завершено”, “Карточку куплено”, “Гравець банкрут”.

3.1.5 Інтерфейс торгової платформи:

а) опис: Купівля та продаж скінів за внутрішню валюту;

б) джерело введення: Користувач через сторінку ринку;

в) призначення виводу: Сервер торгівлі;

г) допустимі значення:

1) ціна: не менше 1 одиниці внутрішньої валюти;

2) кількість: не менше 1.

д) одиниці виміру: Числові та текстові фільтри;

е) часова характеристика: Доступна будь-коли;

ж) зв’язки з іншими інтерфейсами: Інвентар, база предметів;

з) формат вікна: Таблиця з фільтрами, кнопки дій;

и) формат даних: JSON;

к) фінальні повідомлення: “Предмет куплено”, “Товар знято з продажу”.

3.1.6 Інтерфейс інвентарю:

а) опис: Перегляд наявних скінів, вибір для гри, виставлення на продаж;

б) джерело введення: Користувач із вкладки інвентарю;

в) призначення виводу: Сервер інвентарю / торгівлі;

г) допустимі значення: Тільки ті скіни, що є у власності;

д) одиниці виміру: Назва, тип, кількість;

е) часова характеристика: У будь-який момент;

ж) зв’язки з іншими інтерфейсами: Гра, торгова платформа;

з) формат вікна: Плиткове відображення предметів;

и) формат даних: JSON;

к) фінальні повідомлення: “Предмет виставлено на продажу”.

3.1.7 Інтерфейс повідомлень:

а) опис: Обмін системними повідомленнями, запрошення в гру, запити в друзі;

б) джерело введення: Користувач або система;

в) призначення виводу: Інший користувач або сервер повідомлень;

г) допустимі значення: Повідомлення до 256 символів;

д) одиниці виміру: Текст;

е) часова характеристика: У режимі реального часу;

ж) зв’язки з іншими інтерфейсами: Друзі, лобі гри, профіль;

з) формат вікна: Панель сповіщень або вкладка повідомлень;

и) формат даних: JSON, WebSocket;

к) фінальні повідомлення: “Запрошення надіслано”, “Запит в друзі прийнято”.

3.2 Functions

3.2.1 Реєстрація та авторизація

- система повинна перевіряти коректність введеної електронної пошти за встановленим шаблоном;
- система повинна перевіряти, щоб пароль містив щонайменше одну цифру та одну літеру;

- система повинна надавати користувачу можливість входу через Google OAuth 2.0;
- система повинна надсилати лист із новим згенерованим паролем у разі запиту на відновлення доступу.

3.2.2 Керування профілем користувача

- система повинна дозволяти користувачу змінювати нікнейм, аватар та пароль;
- система повинна перевіряти унікальність нікнейму під час збереження.
- система повинна відображати статистику завершених ігор користувача (кількість ігор, перемоги, поразки тощо);
- система повинна перевіряти розмір та формат файлу аватару.

3.2.3 Ігрове лобі та менеджер сесій

- система повинна дозволяти створення нової ігрової кімнати з унікальним ідентифікатором;
- система повинна надавати можливість приєднання до існуючої гри за посиланням або зі списку доступних;
- система повинна перевіряти, що кількість гравців не перевищує 6 і не менше 2;
- система повинна автоматично запускати гру після досягнення мінімального числа гравців і підтвердження усіх учасників.

3.2.4 Основна ігрова логіка

- система повинна дозволяти користувачу кидати кубики та рухати фішку відповідно до результату;
- система повинна обробляти події при потраплянні на певну клітинку (купівля, оренда, податки, тюрма тощо);

- система повинна автоматично змінювати вигляд картки на полі, якщо у гравця є відповідний скін;
- система повинна виводити лог подій гри в режимі реального часу;
- система повинна обробляти перемогу, банкрутство та завершення гри згідно з правилами.

3.2.5 Інвентар і скіни

- система повинна дозволяти гравцю переглядати список наявних скінів;
- система повинна дозволяти вибирати активні скіни для гри;
- система повинна перевіряти наявність обраного скіна у гравця до його застосування в грі.

3.2.6 Торгова платформа

- система повинна дозволяти виставити предмет на продаж, вказавши кількість і ціну у внутрішній валюті;
- система повинна дозволяти переглядати доступні товари з фільтрацією за параметрами (тип, рідкість, ціна);
- система повинна перевіряти, чи дійсно користувач володіє відповідною кількістю товару перед виставленням;
- система повинна знімати предмет з торгівлі на запит користувача;
- система повинна зменшувати кількість валюти користувача при покупці та збільшувати її у продавця;

3.2.7 Повідомлення та запрошення

- система повинна дозволяти надсилання запрошення в гру іншим користувачам;
- система повинна дозволяти надсилання запитів у друзі;
- система повинна повідомляти користувача про отримання нових повідомлень у режимі реального часу.

3.2.8 Обробка помилок та відновлення

- система повинна інформувати користувача про неправильні або відсутні поля введення;
- система повинна зберігати стан гри при розриві з’єднання та дозволяти повторне приєднання;
- система повинна автоматично завершувати хід, якщо користувач не діє протягом визначеного часу.

3.2.9 Канали зв’язку

- система повинна підтримувати постійне WebSocket-з'єднання для обміну ігровими подіями в реальному часі;
- система повинна падати у fallback-режим при втраті WebSocket-з’єднання.

3.3 Performance Requirements

3.3.1 Статичні вимоги

- програмний засіб повинен забезпечувати підтримку щонайменше 500 одночасно авторизованих користувачів без деградації продуктивності;
- програмний засіб повинен підтримувати до 50 активних ігрових сесій одночасно на сервері з рекомендованими характеристиками;
- програмний засіб повинен обробляти до 1000 внутрішньоігрових транзакцій на хвилину;
- кількість одночасно відкритих з’єднань WebSocket не повинна перевищувати 1000 для одного інстансу сервера;
- кожен користувач може мати до 500 об’єктів (скінів) у власному інвентарі.

3.3.2 Динамічні вимоги

- 95% усіх ігрових дій (купівля, продаж, хід, обмін) повинні оброблятися протягом менше ніж 1 секунди після підтвердження користувачем;

- повідомлення через WebSocket повинні доставлятися з затримкою не більше 300 мс у межах однієї ігрової сесії;
- час від моменту запуску гри до повного завантаження ігрового поля на клієнтському інтерфейсі не повинен перевищувати 2 секунд;
- відкриття торгової платформи з фільтрами та переліком товарів має відбуватися за менше ніж 1.5 секунд для користувача із середнім інтернет-з'єднанням (10 Мбіт/с);
- пошук по торговій платформі повинен повертати результати за менше ніж 500 мс при вибірці до 1000 елементів.

3.3.3 Продуктивність при пікових навантаженнях

- під час пікових навантажень (до 500 користувачів онлайн), програмний засіб повинен зберігати не нижче 90% від заявленої базової швидкодії для обробки ігрових подій;
- програмний засіб повинен виконувати балансування навантаження при розгортанні на кількох інстансах для підтримки горизонтального масштабування.

3.4 Logical Database Requirements

3.4.1 Типи інформації

До основних типів даних, що використовуються в системі, належать:

- облікові записи користувачів (e-mail, хеш пароля, нікнейм, аватар, ID, Google OAuth-токени);
- друзі та запити в друзі;
- повідомлення (тип, відправник, одержувач, час надсилання, статус);
- ігрові сесії (учасники, стан гри, ігрове поле, лог ходів, чат);
- статистика користувача (кількість перемог, поразок, зіграних ігор, середня тривалість гри);
- інвентар (список скінів, кількість, ID користувача);

- торгівельна платформа (активні лоти, ціна, кількість, продавець);
- скіни (ID, тип, належність до типу властивості на полі);
- ігрова валюта (баланс користувача, історія транзакцій).

3.4.2 Частота використання

- дані профілю користувача: кожен сеанс входу / перегляду профілю;
- ігрові сесії: активне читання/запис під час гри;
- повідомлення: в режимі реального часу (висока частота доступу);
- торгові операції: помірна частота доступу;
- інвентар: висока частота у гравців, що часто змінюють скіни або торгують;
- статистика: обробка після завершення кожної гри, доступ для перегляду в профілі.

3.4.3 Можливості доступу

- дані користувача: доступ тільки авторизованого користувача або адміністратора;
- ігрові сесії: доступ лише учасників гри;
- повідомлення: доступ для відправника/одержувача;
- торгова платформа: публічний перегляд, редагування — лише власник лоту;
- інвентар: доступ користувача до власного інвентаря.

3.4.4 Сутності та зв’язки

Основні сутності:

- user — зберігає інформацію про користувача, має інвентар, скіни, статистику, список друзів, ігрову валюту;
- item — ігровий предмет, який користувачі можуть купити, продати або обмінювати;

- inventoryItem — сутність, що належить користувачу, зберігає item та кількість цього предмета;
- marketListing — оголошення про продаж, створене користувачем, містить item, кількість, ціну та дату створення;
- notifications — сповіщення про події у системі, містить відправника, одержувача, текст повідомлення, дату та статус;
- friends — зв’язок між двома користувачами, ініційований одним користувачем та спрямований іншому;
- gamestat — таблиця, що зберігає статистику користувача: кількість зіграних ігор, перемог, поразок та набраних очок;
- gamehistory — таблиця, що зберігає історію ігор користувача: дата, результат, тривалість, набрані очки;
- lobby — тимчасова кімната, що зберігається як Redis-хеш lobby:{lobby_id}:meta, містить lobby_id, name, region, max_players, invite_only, creator_id, created_at, started;
- player — гравець, представлений як Redis-хеш user:{player_id}, містить player_id, username, avatar_url, color;
- lobbyPlayer — проміжна сутність між lobby та player, зберігається як множина SADD lobby:{lobby_id}:players {player_id}, гарантує унікальність гравців у лобі;
- gameSession — ігрова сесія, створюється при старті лобі, зберігається як Redis-хеш game:{session_id}:meta з полями session_id, lobby_id, created_at, current_turn, turn_order та список гравців RPUSH game:{session_id}:players {player_id};
- gamePlayer — гравець у сесії, представлений хешем game:{session_id}:player:{player_id}, містить gp_id, balance, position, in_jail, jail_turns_left, immune_to_jail, last_roll, acted_props;
- property — клітинка ігрового поля, зберігається як Redis-хеш game:{session_id}:property:{property_id}, містить property_id, type, name,

buy_price, house_price, hotel_price, rent, group, owner_id, houses, mortgaged, mortgage_turns_left;

- turnState — стан ходу гри в сесії, зберігається як Redis-хеш game:{session_id}:turn_state з полями ts_id, phase, current_player, expires_at, action_payload;
- gameLog — лог дій гравців, зберігається як список RPUSH game:{session_id}:logs {log_entry}, кожен запис — JSON з log_id, session_id, timestamp, message;
- chatMessage — повідомлення чату в грі, зберігається у списку RPUSH game:{session_id}:chat {chat_obj}, кожен об'єкт містить chat_id, session_id, player_id, timestamp, text, а при поверненні клієнту додається username, color;

3.4.5 Обмеження цілісності

- унікальність email та нікнейму користувача;
- неможливість продажу більшої кількості скінів, ніж є в інвентарі;
- неможливість участі користувача в декількох активних іграх одночасно;
- валідація цінових значень на торговій платформі (додатні числа);
- захист від SQL-ін’єкцій та некоректного вводу даних.

3.4.6 Вимоги до збереження даних

- дані користувача та інвентарю мають зберігатися постійно, навіть після тривалого періоду неактивності;
- історія ігор та статистика зберігається не менше 1 року;
- повідомлення видаляються після 6 місяців, якщо не прочитані;
- ігрові сесії очищуються через 24 години після завершення гри;
- лоти з торгової платформи видаляються через 30 днів, якщо не реалізовані.

3.5 Design Constraints

3.5.1 Standards Compliance

У процесі розробки веб-застосунку "MonopolyUA" враховуються обмеження, пов'язані з дотриманням сучасних веб-стандартів, особливостями використовуваного середовища розгортання та регламентами обліку даних. Всі звіти та повідомлення, що створюються системою, повинні мати зрозумілий формат, відповідний сучасним вимогам до веб-інтерфейсів. Назви полів і змінних даних повинні відповідати зрозумілим іменам, прийнятим у галузі веб-розробки, із використанням camelCase або snake_case відповідно до призначення. У системі буде реалізовано механізм журналювання критичних змін, зокрема купівлі, продажу або відкриття кейсів, що дозволить створити повну трасу змін стану гравця та інвентарю для перевірки цілісності транзакцій. Дані мають зберігатися відповідно до політик безпечного обміну та захисту персональних даних, що особливо важливо при авторизації через сторонні сервіси, як-от Google API.

3.6 Software System Attributes

3.6.1 Reliability

Система повинна бути достатньо надійною для стабільної роботи під навантаженням до 500 одночасних користувачів. Усі критичні компоненти, як-от обробка авторизації, ігрового процесу та транзакцій у маркеті, повинні бути протестовані на стійкість до збоїв. Надійність перевірятиметься шляхом багатократного відтворення основних сценаріїв взаємодії з користувачем із вимірюванням кількості помилок на 1000 транзакцій. Очікувана середня безвідмовна тривалість роботи має становити не менше 200 годин.

3.6.2 Availability

Програмна система має бути доступною цілодобово, оскільки передбачає багатокористувацький онлайн-доступ до сервісу. У разі збоїв користувач повинен мати можливість поновити сесію або гру із втратою не більше 5 секунд ігрових

даних. Для забезпечення цього буде використовуватись збереження проміжного стану гри та асинхронна обробка запитів.

3.6.3 Security

Система повинна захищати користувацькі дані від несанкціонованого доступу, використовуючи JWT для автентифікації. Критичні дані (наприклад, паролі та токени) мають зберігатися у зашифрованому вигляді. Має бути реалізовано контроль за правильністю передачі повідомлень у WebSocket-каналах.

3.6.4 Maintainability

Проєкт побудований за принципом клієнт-серверної архітектури, де фронтенд і бекенд реалізовані як окремі незалежні додатки. Такий підхід дозволяє розвивати інтерфейс користувача та серверну логіку автономно, що спрощує підтримку й модернізацію системи.

Бекенд реалізовано з використанням Django REST Framework, що забезпечує модульну організацію коду. Основна логіка поділена на функціональні сегменти: автентифікація, управління профілем, маркет, ігрові механіки та чат. Це дає змогу легко змінювати або розширювати окремі частини системи без впливу на інші компоненти.

3.6.5 Portability

Портативність веб-застосунку «MonopolyUA» забезпечується використанням платформонезалежних технологій: Django для бекенду та стандартних технологі HTML/CSS/JS для фронтенду. Усі системозалежні налаштування (наприклад, шляхи до файлів, змінні середовища) винесені у конфігураційні файли, що дозволяє легко адаптувати систему під інші хост-машини або ОС. Використання перевірених інструментів, таких як Docker, забезпечує спільне середовище виконання на будь-якій платформі. Код не містить значної кількості хост-залежних фрагментів і протестований на Windows.

Вимірювання портативності відбуватиметься шляхом запуску застосунку в різних середовищах та оцінки змін, потрібних для успішної роботи. Таким чином, переносимість оцінюється як висока.

3.7 Organizing the Specific Requirements

3.7.1 System Mode

Веб-застосунок «MonopolyUA» має різні режими роботи, зокрема режим реєстрації та авторизації, режим роботи з профілем користувача, ігровий режим, режим роботи магазину та лідерборду. Кожен режим має свої особливості інтерфейсу та логіки, тому вимоги до системи формулюються окремо для кожного режиму, враховуючи специфіку відображення даних і поведінки системи, що дозволяє підвищити зручність та продуктивність.

3.7.2 User Class

Система розподіляє функціонал залежно від класу користувача. Зареєстровані гравці мають доступ до повного набору можливостей — участь в іграх, покупка скінів, додавання друзів. Адміністратори отримують розширені права для керування контентом і користувачами, а незареєстровані користувачі можуть лише ознайомитися з правилами та зареєструватися.

3.7.3 Objects

Основними об’єктами системи є користувач, ігрова сесія, ігрове поле, інвентар скінів, товари магазину, чат, рейтингові таблиці та списки друзів. Кожен об’єкт має свої атрибути (наприклад, профіль користувача містить ім’я, аватар, статистику), а також функції (додавання друга, створення лоббі, відправка повідомлення), що забезпечують реалізацію бізнес-логіки.

3.7.4 Feature

Функції системи визначаються як конкретні послуги, які вона надає користувачам, наприклад, реєстрація, авторизація, створення та пошук ігрових

сесій, купівля скінів, обмін повідомленнями в чаті, оновлення профілю. Для кожної функції описується послідовність дій, необхідних для її виконання, включно з вхідними даними та очікуваною відповіддю системи.

3.8 Additional Comments

Для веб-застосунку «MonopolyUA» доцільно застосовувати одночасно кілька способів організації специфічних вимог, описаних у пункті 3.7, оскільки це допоможе краще структурувати вимоги та врахувати різні аспекти роботи системи.

Зокрема, організація вимог за режимами роботи системи (реєстрація, ігровий режим, магазин тощо) допоможе чітко виділити логіку для кожного сценарію використання. Водночас розподіл за класами користувачів (гравці, адміністратори, гості) дозволить чітко визначити права доступу та функціональні можливості.

Використання об’єктного підходу сприяє кращому опису структури системи та взаємодії між основними компонентами, такими як ігрова сесія, користувачі та інші. Організація вимог за функціями дозволяє деталізувати послідовність дій користувача та реакцію системи, що важливо для реалізації бізнес-логіки.

4. Change Management Process

Зміни в вимогах до веб-застосунку «MonopolyUA» будуть контролюватися через чітко встановлений процес управління змінами. Клієнт не може просто зателефонувати або висловити нову ідею усно — всі пропозиції щодо змін мають подаватися офіційно у письмовій формі електронною поштою. Після отримання запиту команда проекту проводить його оцінку з урахуванням технічних можливостей, впливу на поточний план робіт і бюджету. Всі рішення щодо впровадження змін ухвалюються колективно командою.

5. Document Approvals

TBD.

6. Supporting Information

Цей розділ містить допоміжні матеріали, що полегшують читання, розуміння та використання документа Специфікації вимог до програмного забезпечення (SRS). Хоча вони не є безпосередньою частиною формальних вимог, ці матеріали служать важливим доповненням для розробників, тестувальників, замовників та інших зацікавлених осіб.

Зміст документа представлено на початку файлу й охоплює всі розділи: від вступу до функціональних і нефункціональних вимог, включно з додатками.

Інтерактивний або текстовий покажчик термінів, абревіатур і ключових понять, які використовуються у документі. У разі публікації в цифровому вигляді, передбачена можливість пошуку за ключовими словами (наприклад: "WebSocket", "JWT", "Redis").