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 для гри у монополію "MonopolyUA". Серверна асинхронна частина.
(тема)

Виконав:
здобувач 3 курсу, групи ПЗПІ-22-6
Олег СКРЯГІН
(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 Скрягіну Олегу Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для гри у монополію "MonopolyUA". Серверна асинхронна частина.

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

3. Вихідні дані до проєкту Технічне завдання з переліком функцій і вимог, вибрані технології та середовище розробки, опис Redis-структур, документація WebSocket-подій

4. Перелік питань, що потрібно опрацювати в роботі

Як гарантувати унікальність ідентифікаторів лобі та ігрових сесій у розподіленому середовищі? Які структури даних Redis оптимальні для зберігання стану лобі та учасників (хеші, множини, списки)? Як реалізувати надійну доставку повідомлень через WebSocket з урахуванням повторного підключення клієнта? Яким чином організувати моніторинг та обробку таймаутів на черговість ходів? Як побудувати документацію та діаграми, які чітко відображають внутрішню архітектуру і процеси системи?

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	07.06.2025	виконано
2	Розробка постановки задачі	07.06.2025	виконано
3	Проектування ПЗ	08.06.2025	виконано
4	Програмна реалізація	08.06.2025	виконано
5	Аналіз результатів	08.06.2025	виконано
6	Підготовка пояснювальної записки.	08.06.2025	виконано
7	Перевірка на наявність ознак академічного плагіату	19.06.2025	виконано
8	Захист роботи		виконано

Дата видачі завдання “ 26 ” лютого 2025р.

Здобувач Скрягін Олег
(підпис)

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

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 74 стор., 21 рис., 10 джерел.

МОНОПОЛІЯ, ВЕБ-ГРА, DJANGO, REDIS, ASGI, DJANGO CHANNELS, WEBSOCKET, BACKEND, PYTHON, POSTGRESQL, ЛОГІКА ГРИ

Об’єктом розробки є серверна система для веб-онлайн гри «Монополія», яка забезпечує одночасну гру до чотирьох гравців через браузер з підтримкою оновлення стану в реальному часі.

Метою розробки є створення програмного рішення, яке гарантує коректне виконання правил гри «Монополія», управління грошовими операціями та активами гравців, а також формування післяігрового рейтингу без затримок.

Реалізація серверної частини базується на Python та фреймворку Django 5 для побудови REST API та вебсокетів, з використанням Django Channels на базі ASGI для двосторонньої комунікації. Redis застосовується для швидкого зберігання ігрових сесій і черг повідомлень, а PostgreSQL — для фіксації даних користувачів та статистики.

У результаті створено масштабований та надійний бекенд для MonopolyUA, що підтримує повну логіку гри, оперативну синхронізацію між учасниками, безпечні транзакції й відображення підсумкових результатів для всіх гравців.

MONOPOLY, WEB GAME, DJANGO, REDIS, ASGI, DJANGO CHANNELS, WEBSOCKET, BACKEND, PYTHON, POSTGRESQL, GAME LOGIC

The development object is the server-side system for the MonopolyUA web-based game, which enables up to four players to play simultaneously through a browser with real-time state updates.

The development goal is to create a software solution that ensures correct enforcement of Monopoly game rules, management of financial transactions and player assets, and generation of post-game rankings without latency.

The server-side implementation is based on Python and the Django 5 framework for building REST APIs and WebSockets, utilizing Django Channels on ASGI for bi-directional communication. Redis is used for fast storage of game sessions and message queues, while PostgreSQL records user data and game statistics.

As a result, a scalable and reliable backend for MonopolyUA has been created, supporting full game logic, real-time synchronization among participants, secure transactions, and presentation of final results for all players.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

REST – Representational State Transfer

API – Application Programming Interface

ASGI – Asynchronous Server Gateway Interface

HTTP – Hypertext Transfer Protocol

CRUD – Create, Read, Update, Delete

HTTPS – Hypertext Transfer Protocol Secure

WSS – WebSocket Secure

TLS – Transport Layer Security

UTC – Coordinated Universal Time

ISO – International Organization for Standardization

DRF – Django REST Framework

JWT – JSON Web Token

JSON – JavaScript Object Notation

AOF – Append Only File

RDB – Redis Database

ВСТУП

У сучасному цифровому середовищі попит на інтерактивні онлайн-ігри, що забезпечують можливість одночасної взаємодії багатьох користувачів у реальному часі, невпинно зростає. Класичні настільні ігри, такі як Monopoly, завдяки своїм стратегічним правилам і соціальному компоненту, залишаються популярними протягом десятиліть. Водночас розвиток веб-технологій і серверних рішень, зокрема асинхронних WebSocket-зв'язків [1] та швидкодіючих баз даних на основі Redis, відкриває нові можливості для створення стабільних і масштабованих мультиплеєрних платформ.

Актуальність роботи полягає у необхідності поєднати класичні правила Monopoly з можливостями сучасних веб-рішень, забезпечивши користувачам зручний інтерфейс, мінімальну затримку відгуку та прозору синхронізацію ігрового стану між усіма учасниками. Реалізація такої системи сприятиме популяризації настільних ігор у цифровому середовищі та відкриє нові комерційні й освітні перспективи.

Метою роботи є розробка та впровадження WebSocket-орієнтованого серверного ядра для онлайн-версії Monopoly з використанням Django Channels [9] та Redis, яке гарантує високу швидкодію, узгодженість ігрового процесу та можливість одночасної гри до чотирьох учасників із підтримкою логування ходів, чату та повного ігрового циклу (ходи, купівля, оренда, будівництво, банкрутство).

Для досягнення цієї мети поставлено такі завдання:

– побудувати механізм черговості ходів із таймаутами та автоматичним пропуском ходу;

– реалізувати модулі обробки дій гравців (кидання кубиків, переміщення фішок, купівля й оренда власності, управління будівлями);

– організувати систему клієнт-серверної взаємодії через WebSocket із використанням Redis Channel Layer для розсилки подій;

– забезпечити ведення журналу гри та інтегрований чат між гравцями;

– розробити логіку банкрутства та завершення гри з виведенням результатів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Предметна галузь даного проєкту охоплює сферу розробки багатокористувацьких онлайн-версій класичних настільних ігор із режимом реального часу. У центрі уваги — гра «Монополія», що передбачає поєднання економічних механік купівлі-продажу нерухомості, стратегії управління активами та черговості ходів між кількома гравцями. Основна мета системи полягає в тому, щоб відтворити знайомий ігровий процес на цифровій платформі, забезпечивши при цьому коректність правил, узгодженість ігрового стану для всіх учасників і комфортну взаємодію в реальному часі.

Динаміка гри передбачає постійне оновлення інформації про розташування фішок на ігровому полі, фінансовий баланс кожного гравця, власність на ті чи інші ділянки. Ділова логіка гри формується як послідовність станів: початок ходу, кидок кубиків, пропозиція купівлі, стягнення орендної плати, будівництво та продаж активів, операції із заставою та викупом, перебування в «тюрмі» й випадкові події на спеціальних клітинках. Перехід від однієї фази до іншої повинен відбуватися в суворій відповідності до правил гри [2], не допускаючи суперечностей та розбіжностей між гравцями.

Особлива увага приділяється черговості ходів та механізму таймаутів. Кожен гравець має обмежений проміжок часу на виконання своєї дії — від кидка кубиків до вирішення фінансових питань. Це запобігає «зависанню» партії та забезпечує безперервність гри для всіх учасників. Якщо гравець не встигає ухвалити рішення, система автоматично реалізує стандартне правило пропуску ходу або початку процедури примусової ліквідації активів.

У межах організації ігрового процесу передбачається також створення умов для формування тимчасових ігрових кімнат (лобі). У лобі гравці збираються перед початком партії, очікують необхідну кількість учасників і лише після цього формують нову гру. Ігрові кімнати повинні підтримувати механізми приєднання, виходу та виключення учасників за рішенням організатора.

Ключовим аспектом предметної галузі є синхронізація стану гри між усіма клієнтами. Система має гарантувати, що незалежно від порядку й швидкості виконання дій на стороні клієнта, фінальний візуалізований стан завжди відповідає єдиній «істині» на сервері. Усе рішення зберігається централізовано, а клієнтська частина лише відображає зміни — таким чином виключається можливість шахрайства чи розбіжностей у правилах.

Узагальнення вказаного аналізу підкреслює, що побудова онлайн-версії «Монополії» потребує ретельного моделювання ігрових станів, чіткої організації черговості та таймаутів ходів, а також механізмів створення ігрових кімнат. Усе це має реалізуватися з урахуванням принципу єдиної централізованої обробки логіки гри і гарантії узгодженості стану для багатьох одночасних користувачів.

1.2 Виявлення та вирішення проблем

При побудові бекенд-частини онлайн-версії «Монополії» виникає потреба забезпечити безперервний та узгоджений ігровий процес для одночасної присутності десятків і сотень гравців. Однією з головних проблем є гарантування, що будь-яка дія одного учасника — від створення чи приєднання до ігрової кімнати до виконання ходу чи оплати оренди — миттєво відображається в усіх інших клієнтських інтерфейсах. Для її розв’язання застосовується архітектура «Pub/Sub», коли кожна зміна стану публікується в єдиний канал обміну повідомленнями, а всі активні серверні процеси та WebSocket-consumers одночасно отримують це повідомлення і оновлюють внутрішню модель гри та відправляють клієнтам відповідний пакет даних [3].

Ще одна критична задача полягає в оптимізації зберігання ігрового стану. Операції читання і запису балансу, позиції фішок і властивостей клітинок мають відбуватися з мінімальною затримкою. Для цього вибирається in-memory сховище, в якому кожен гравець та кожна ігрова сесія представляються окремими хеш-структурами, а списки учасників лобі — множинами та впорядкованими списками. Така модель дає змогу оновити єдине поле (наприклад, баланс одного гравця), не перезаписуючи весь об’єкт, і виконати це за постійну амортизовану складність.

Організація черги ходів реалізується через зберігання поточної фази гри та дедлайну дії в окремому записі стану ходу. Після отримання результатів кидка кубиків чи відповіді на запит купівлі нерухомості сервер оновлює цю фазу, формує нове завдання для клієнта (очікування купівлі, оплати оренди, продажу активів тощо) і встановлює таймер. Якщо гравець не встигає підготувати рішення до вказаного часу, відбувається автоматичне пропускання ходу або перехід до процедури ліквідації активів, що не дозволяє грі «зависнути».

Масштабованість досягається шляхом того, що всі серверні інстанси працюють з одним сховищем стану та підписані на ті самі канали подій. Додавання нових процесів не вимагає складної синхронізації: вони просто починають читати події з каналу та оновлювати локальні кеші, гарантуючи єдину «істину» у всіх компонентах системи.

Таким чином, поєднання високошвидкісного in-memory зберігання, моделі «Pub/Sub» та чіткої організації черги ходів забезпечує надійний, узгоджений ігровий досвід без затримок та блокувань, що є критично важливим для підтримки динаміки класичної гри «Монополія» в онлайн-форматі.

1.3 Аналіз аналогів програмного забезпечення

Аналіз аналогів програмного забезпечення полягає у вивченні існуючих продуктів, що забезпечують близький до нашого функціонал, з метою виявлення їхніх сильних і слабких сторін та формулювання оптимальної стратегії розвитку власного рішення. У контексті веб-орієнтованої реалізації гри «Монополія» доречно розглянути платформи, що підтримують багатокористувацький режим, механізми створення ігрових кімнат і синхронне оновлення стану гри.

Онлайн-платформа Board Game Arena пропонує універсальне середовище для широкого спектра настільних ігор, у тому числі карткових і стратегічних. Її архітектура розрахована на масштабованість і одночасну гру десятків тисяч користувачів, реалізовано як реальний час через вебсокети, так і покроковий режим із збереженням історії ходів. Проте універсальність BGA обумовлює складність масштабного налаштування логіки конкретної гри: серверна частина реалізує

загальні механізми відправки повідомлень і збереження загального стану, але менше уваги приділяє тонкощам правил однієї гри, а їхня точна імплементація лягає на клієнтські скрипти або плагіни.

Мобільний та веб-додаток Rento спеціалізується саме на онлайн-версії «Монополії». У ньому передбачені як гра з реальними суперниками, так і протистояння з комп'ютерними опонентами, а також локальні партії по Wi-Fi. Така орієнтація на мультиплатформність і роботу без постійного інтернет-з'єднання розширює аудиторію, але водночас вимагає від серверної частини реалізації механізмів синхронізації офлайн-даних та розв'язання конфліктів, коли гравці повторно підключаються.

Десктопні рішення на кшталт Tabletop Simulator і Tabletopia забезпечують середовище для симуляції будь-якої настільної гри в тривимірній графіці. Вони націлені на універсальність і фізичну достовірність взаємодії з ігровими елементами, але не оптимізовані під чітке дотримання правил однієї конкретної гри та потребують значних обчислювальних ресурсів. Бекенд у таких рішеннях здебільшого виконує роль загальної шини повідомлень між клієнтами, залишаючи логіку правил на клієнтському рівні.

Ігрові адаптації «Монополії» під брендом Hasbro (Monopoly Plus, Monopoly Go та ін.) мають власні серверні рішення для підтримки великої кількості користувачів і часто поєднують соціальні функції (запрошення, рейтинги). Водночас внутрішній API та формат зберігання стану гри закриті, а розвиток нових механік обмежений комерційними інтересами.

Порівняння цих продуктів показує, що для стабільної роботи й точного дотримання класичних правил «Монополії» найкращим підходом є виділена серверна компонента, яка виконує всю логіку гри у відокремленому середовищі, гарантує узгодженість стану та запобігає шахрайству. Запропоноване рішення фокусується саме на серверному управлінні ігровими сесіями, створенні ігрових кімнат та синхронному розповсюдженні подій у реальному часі, водночас реалізуючи всі правила класичної «Монополії» на сервері, що забезпечує абсолютну цілісність і відповідність стандартам гри.

2 ПОСТАНОВКА ЗАДАЧІ

У межах даного курсового проєкту необхідно створити ефективну, масштабовану та надійну серверну платформу для організації та проведення онлайн-сесій гри «Монополія», яка забезпечує весь життєвий цикл партії — від формування кімнати для гри (лобі) до показу підсумкового рейтингу та статистики гравців. Для досягнення цих цілей передбачено низку ключових завдань:

- створення та управління лобі. Потрібно забезпечити унікальність ідентифікаторів лобі за допомогою UUID (uuid.uuid4()), зберігати метадані (назва лобі, регіон, максимальна кількість гравців, налаштування приватності, адмін, дата створення) та контролювати доступ тільки для авторизованих користувачів через DRF permission_classes = [IsAuthenticated]. Для швидкого додавання й видалення учасників, визначення кількості гравців і формування переліку використовуються Redis-хеші для метаданих лобі, множини для списків гравців і глобальне множення активних лобі;
- миттєве інформування учасників. Кожна подія — від приєднання нового гравця до старту партії чи виходу учасника — повинна надсилатися всім через WebSocket [1]. Для цього застосовуються Django Channels [9] для встановлення довготривалих з’єднань і Redis Pub/Sub як шина повідомлень для low latency доставки [4]. При підключенні клієнт отримує поточний список лобі, а далі синхронізується через події create, update, remove;
- використання Redis як основного сховища стану. Redis-хеші утримують атрибути сесії та стан гравців (баланс, позиція, статус застави, колір фішки). Redis-множини зберігають учасників лобі, а черги (RPUSH/BLPOP) забезпечують впорядковану послідовність ходів. Такі структури підтримують O(1) операції читання/запису для високопродуктивного обслуговування великої кількості одночасних гравців;

– реалізація бізнес-логіки «Монополії». Сервер обробляє кидок кубиків, переміщення фішки, перевірку спеціальних клітин (Start, Jail, GoToJail, Chance, Community Chest). Пропонує купівлю нерухомості, рахує оренду залежно від типу власності та кількості будинків або готелів. Здійснює будівництво та продаж будинків/готелів із дотриманням правил «рівномірності», оформлення застави й викуп із 10 % комісії. Забезпечує тюремну логіку (до трьох ходів у в’язниці або використання картки «Вихід із в’язниці») та сценарії банкрутства з автоматичним підрахунком підсумкового рейтингу;
– асинхронний моніторинг дедлайнів. Для ключових дій (купівля, сплата оренди чи утиліти) вводяться таймаути. Якщо гравець не відповідає вчасно, його хід пропускається автоматично й передається наступному учаснику. Такі події фіксуються в журналі гри;
– ведення журналу подій і чату. Сервер записує всі події (купівля, сплата оренди, будівництво, банкрутство, виключення гравця) у Redis у окрему гілку логів. Повідомлення надсилаються учасникам у режимі реального часу в форматах game_log і game_chat через WebSocket-з’єднання [1].

Реалізація цих завдань забезпечить побудову повноцінної серверної частини веб-гри «Монополія», котра наддасть можливість великій кількості гравців одночасно брати участь у партіях, дотримуватися класичних правил, отримувати актуальні оновлення стану гри та брати участь у загальному чаті без затримок.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

У процесі проєктування серверної частини «MonopolyUA» були виокремлені ключові сценарії взаємодії гравця із системою, від створення та управління ігровим лобі до безпосереднього ігрового процесу (див. рис. 3.1).

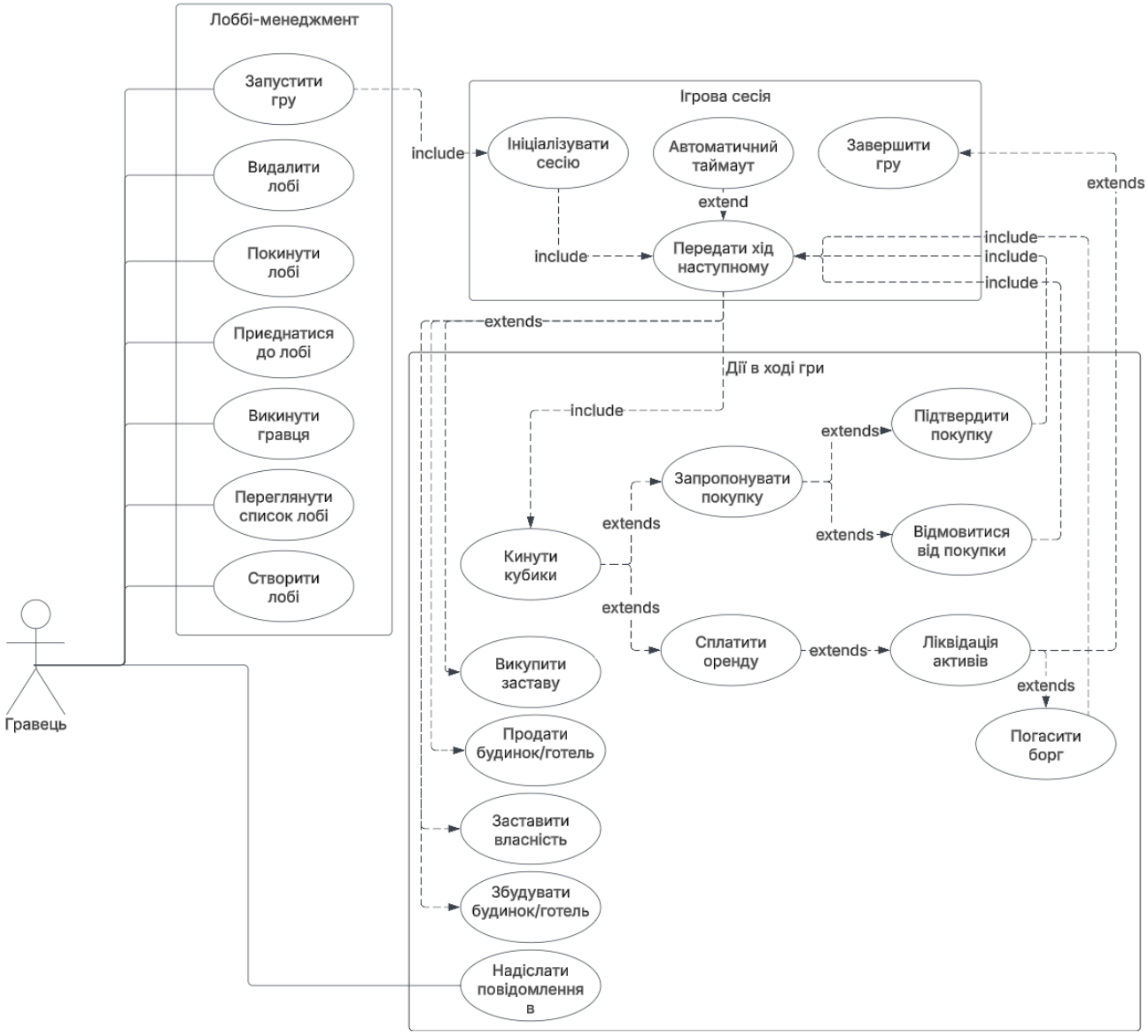


Рисунок 3.1 – UML-діаграма прецедентів взаємодії гравця із серверною частиною «MonopolyUA»

Спочатку користувач створює нове лобі або приєднується до вже існуючого, має змогу переглянути всі доступні кімнати, вільно залишити або покинути лобі, а

також, за наявності відповідних прав, видалити лобі чи виключити з нього іншого гравця. Після ініціалізації гри в лобі власник викликає старт сесії, що призводить до створення ігрової сесії з унікальним ідентифікатором, налаштуванням черги ходів і початкового стану гравців. Далі в рамках цієї сесії учасник може кинути кубики, отримати пропозицію купівлі нерухомості з можливістю підтвердити чи відмовитися, сплатити оренду та будувати чи продавати будинки й готелі, а також закладати й викупати власність. У будь-який момент під час свого ходу користувач може залишити повідомлення в загальному чаті. Усі ці дії підтримують автоматичний таймаут, передачу ходу наступному гравцю та завершення гри у разі банкрутства.

Внутрішня логіка обробки ходу реалізована у вигляді послідовності дій, зображених на діаграмі активності (див. рис. 3.2).

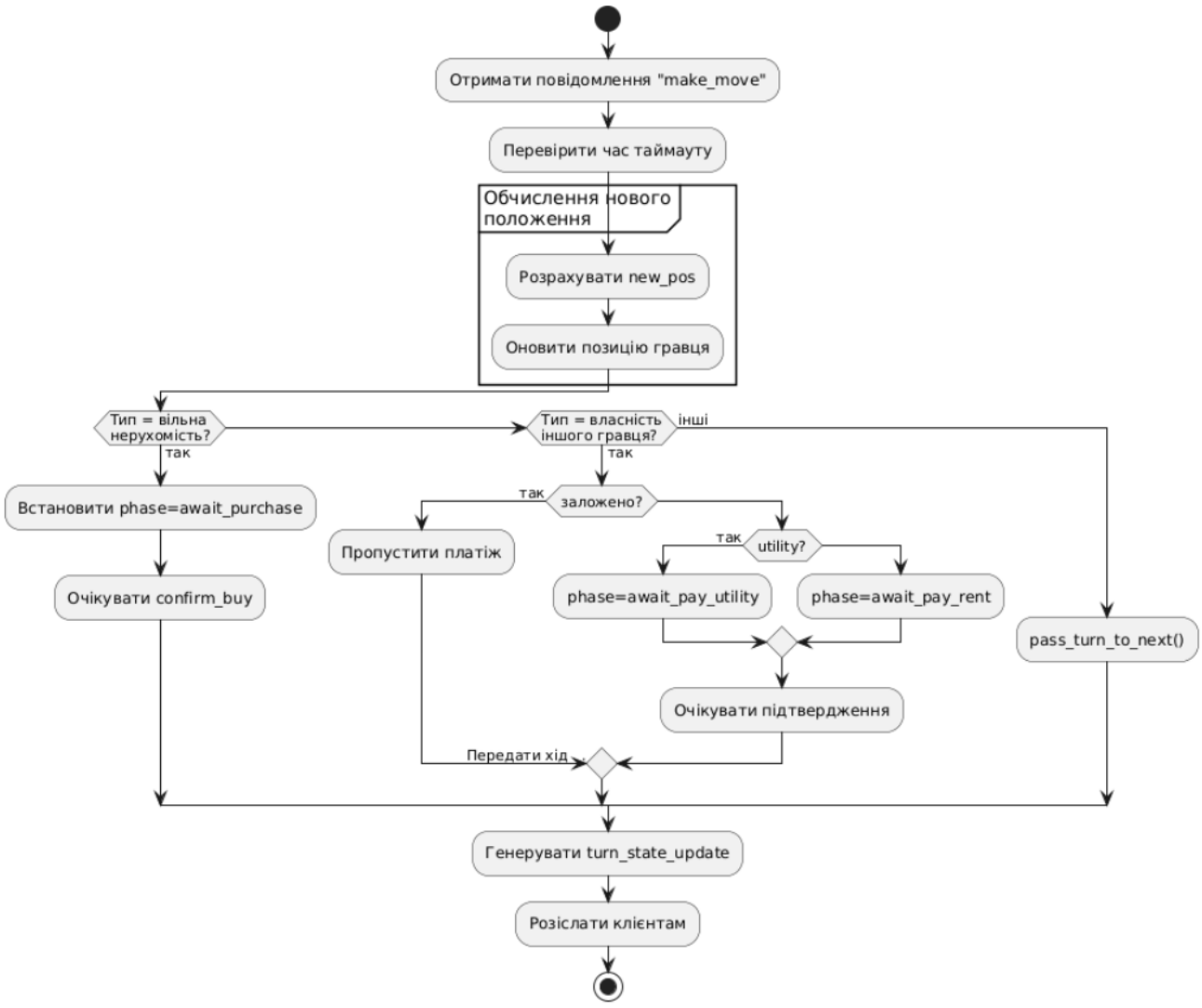


Рисунок 3.2 – UML-діаграма активності обробки ходу гравця

Після отримання сервером запиту на виконання ходу перевіряється, чи не минув час, відведений на виконання дії, потім обчислюється нова позиція фішки – і, залежно від типу клітинки, встановлюється наступна фаза: очікування рішення про купівлю вільної нерухомості, очікування сплати оренди (з урахуванням застави), або безумовна передача ходу далі. Наприкінці кожної гілки формуються та розсилаються всім підписаним клієнтам оновлення стану гри, що гарантує синхронну відображуваність гри у реальному часі.

3.2 Проєктування архітектури ПЗ

Архітектура програмного забезпечення «MonopolyUA» спирається на класичну клієнт-серверну модель, де браузер гравця виступає клієнтом, а всю бізнес-логіку та збереження стану гри обслуговує єдиний серверний застосунок. Серверна частина реалізована на основі фреймворку Django у монолітному вигляді, але з підтримкою асинхронних WebSocket-з’єднань через Django Channels [5]. Унаслідок цього всі запити на створення чи перегляд ігрового лобі обробляються звичайним HTTP-інтерфейсом, а обмін подіями в режимі реального часу — через постійні WebSocket-канали, що дозволяє одразу інформувати всіх учасників про нові ходи, повідомлення в чаті чи зміну складу гравців.

Основними будівельними блоками системи є модулі HTTP-API, підписка на події, обробка бізнес-логіки і зовнішні сервіси зберігання. Запити користувача спочатку проходять через шар авторизації, що гарантує доступ лише зареєстрованим гравцям. Далі контролери REST-інтерфейсу валідують дані й звертаються до підсистеми збереження, яка тримає поточний стан лобі та гри в оперативному сховищі. Як тільки стан змінюється — наприклад, хтось приєднується до лобі або кидає кубики — відповідна подія миттєво публікується у внутрішній канал розсилки. Споживачі цих подій, що працюють у рамках WebSocket-сесій, отримують оновлення та транслюють їх клієнтам, забезпечуючи мінімальну затримку між дією одного гравця та її відображенням у браузерах усіх інших [6]. На рівні компонентної діаграми (див. рис. 3.3) виділено кілька взаємозалежних модулів:

– REST API [7] – обробляє всі HTTP-запити для роботи з лобі (створення, список, приєднання, вихід, видалення), виконує валідацію та авторизацію;
– LobbyService – бізнес-логіка управління ігровими кімнатами, звертається до Redis для збереження метаданих та списку гравців і публікує події в Pub/Sub;
– GameEngine – реалізує правила «Монополії» для кожної сесії: ініціалізацію нового раунду, черговість ходів, кидок кубиків, купівлю-продаж нерухомості, розрахунок оренди, будівництво/продаж будинків, заставу, ліквідацію активів і банкрутство;
– EventDispatcher – підписується на внутрішні канали Redis, транслює події (lobby_update, lobby_start, game_move, game_log тощо) через WebSocket (Django Channels) до клієнтів у режимі реального часу;
– TurnManager – фоновий асинхронний компонент, що стежить за дедлайнами ходів, автоматично пропускає таймаути та передає хід далі.

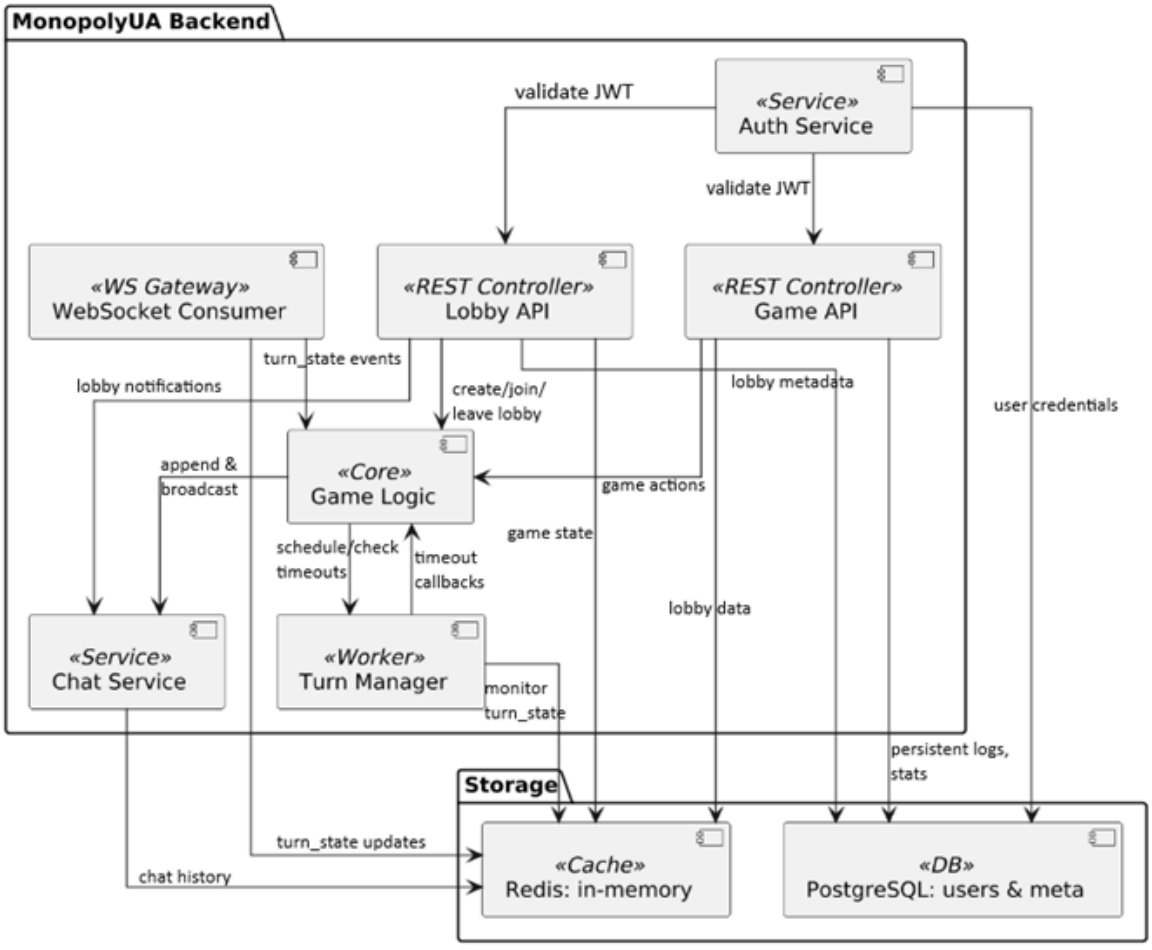


Рисунок 3.3 – UML-діаграма компонентів

У вузлі застосунку окремими підсистемами виділено обробку запитів REST-API, механізм публікації-підписки на події, набір функцій ігрової логіки та фоновий менеджер черги ходів зі стеженням за таймаутами. HTTP-шар відповідає за усі CRUD-операції з лобі — створення кімнат, зміну їхніх властивостей та перевірку прав користувачів. Асинхронний шар, навпаки, займається саме грою: від ініціалізації нового раунду до передачі ходу та обробки унікальних правил «Монополії» — купівлі нерухомості, розрахунку орендної плати, будівництва та продажу будинків, застави й банкрутства. Завдяки чіткому розділенню відповідальності між синхронною і асинхронною частинами, а також єдиному джерелу правди в оперативному сховищі, система гарантує, що всі гравці працюють із однаковим станом гри та неможливо здійснити несанкціоновану зміну даних.

На діаграмі розгортання (див. рис. 3.4) показано, що браузер гравця підключається до сервера застосунку по HTTPS (REST-запити) та WSS (WebSocket-канали). Монолітний сервер розгортається на окремому віртуальному вузлі, має доступ до кластеру Redis [8] (in-memory сховище) та, за потреби, до PostgreSQL для довготривалих даних (наприклад, облікові записи користувачів). Мережеві з’єднання захищені TLS, що гарантує конфіденційність і цілісність переданих повідомлень.

Нефункціональні вимоги накладають такі обмеження та стратегії їх задоволення:

- продуктивність: мінімальна затримка при оновленні стану завдяки in-memory сховищу Redis та неблокуючим WebSocket-каналам;
- масштабованість: горизонтальне додавання інстансів серверу, що спільно читають/пишуть у Redis та підписані на ті ж канали;
- надійність: автоматичне повторне підключення клієнта за втратою з’єднання, моніторинг стану turn-state і захист від “зависання” на очікуванні дії;
- безпека: тільки автентифіковані користувачі можуть змінювати стан лобі чи гри, всі канали захищені TLS.

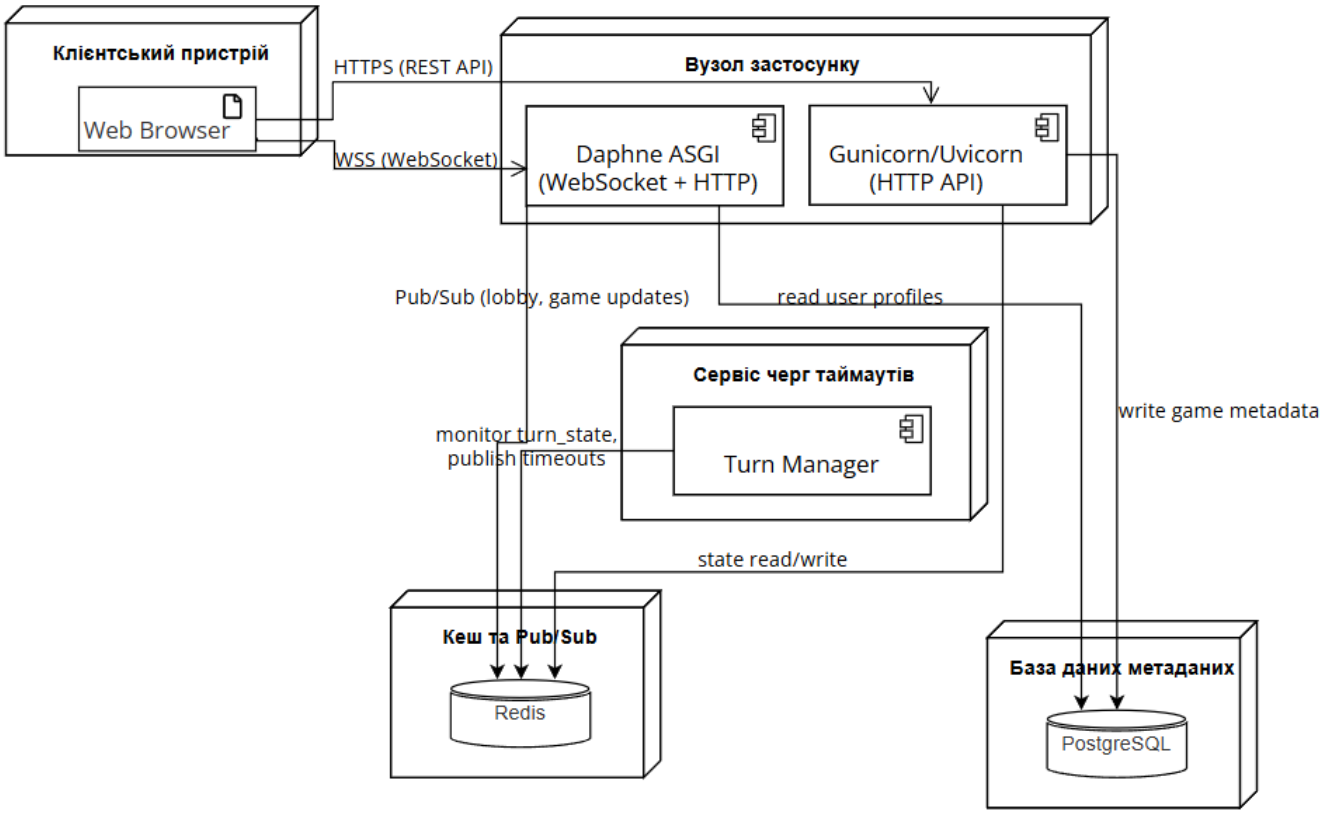


Рисунок 3.4 – UML-діаграма розгортання

Серед потенційних ризиків слід відзначити: втрата повідомлень Pub/Sub [6] може призвести до розбіжностей стану гри; для зменшення цього ризику запроваджено механізм перевірки і повторної доставки невідправлених подій. Збій одного з вузлів Redis чи застосунку може призупинити гру, тому планується впровадження реплікації Redis та балансування WebSocket-з'єднань у наступному етапі розгортання.

Таким чином запропонована архітектура поєднує переваги монолітного підходу — єдину кодову базу та простоту розгортання — з гнучкістю асинхронних каналів і високою продуктивністю оперативного сховища. Вона забезпечує чітку логіку управління лобі, бездоганну синхронізацію стану під час гри та можливість масштабувати сервіс під навантаження великих одночасних сесій.

3.3 Проєктування структури зберігання даних

Redis обрано як основне сховище даних для реалізації ігрових лобі та сесій у «MonopolyUA» насамперед через його надзвичайно високу швидкодію, простоту

горизонтального масштабування та вбудовані механізми pub/sub-оновлень, які ідеально підходять для мультиплеєрних ігор із реальним часом. Використання Redis дозволяє утримувати всі дані про тимчасові лобі та активні ігрові сесії «in-memory», гарантує миттєвість операцій читання/запису і підтримує атомарні структури (хеші, множини, списки), що знижує складність бізнес-логіки.

У логічній ER-діаграмі Redis-сховища «MonopolyUA» (див. рис. 5) відображено основні сутності та їхні взаємозв’язки.

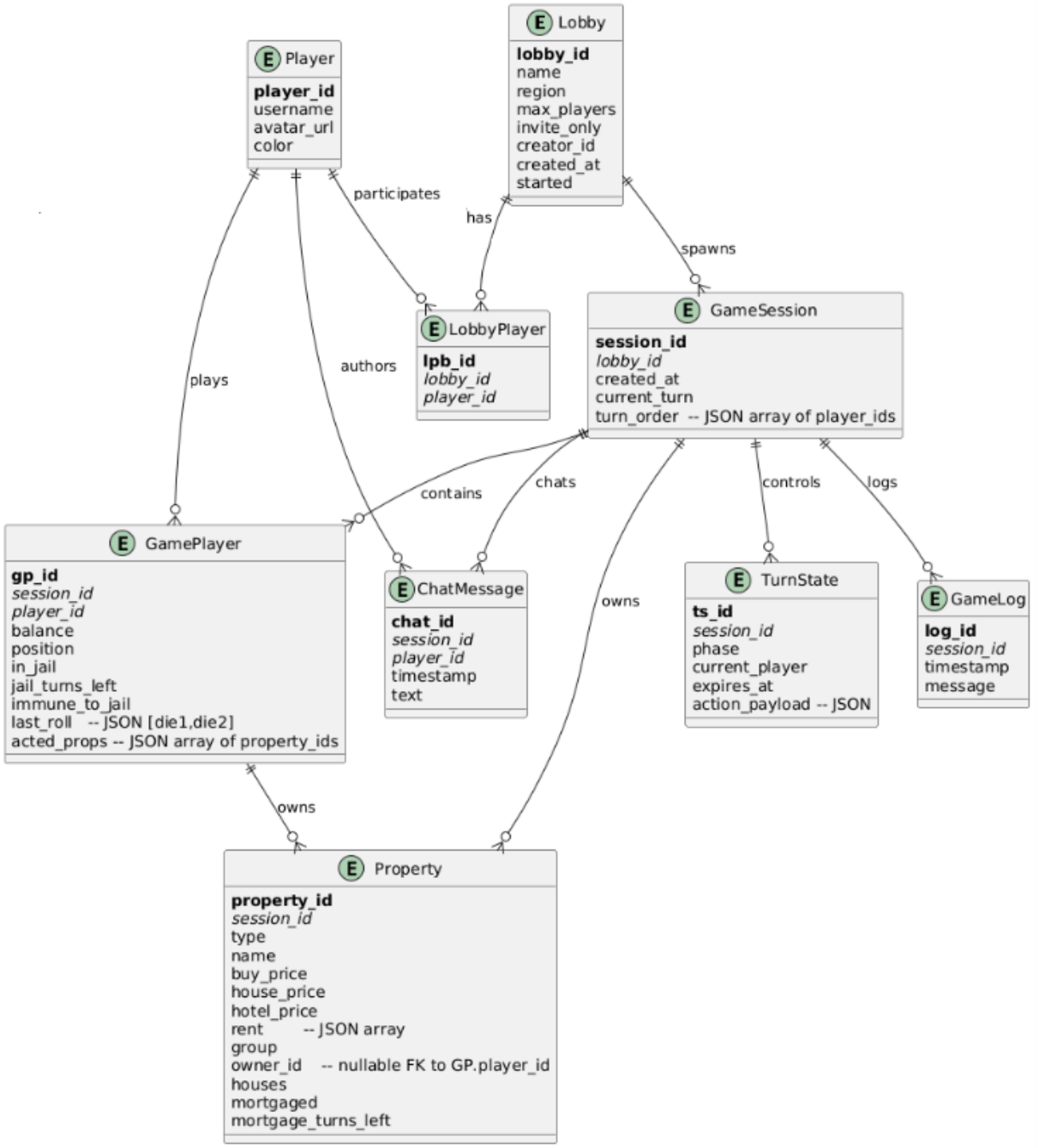


Рисунок 3.5 – Схема бази даних

У цій моделі реалізовано такі ключові компоненти:

- при створенні лобі через API генерується унікальний lobby_id і зберігається хеш lobby:{lobby_id}:meta із полями name, region, max_players, invite_only, creator, started=false, created_at;
- одночасно у множину lobby:{lobby_id}:players додається ID творця, що гарантує унікальність учасників і дає змогу швидко отримувати їхню кількість (SCARD) і перелік (SMEMBERS;
- в глобальну множину lobbies:available додається кожен новостворений lobby_id, що дозволяє оперативно формувати список доступних лобі;
- після будь-якої зміни (створення, приєднання, вихід, видалення) публікується повідомлення в канал lobbies:updates, яке WebSocket-консьюмери транслюють усім клієнтам у режимі реального часу;
- при старті гри функція start_lobby генерує session_id, додає його в множину game:active і створює хеш game:{session_id}:meta з полями turn_order (JSON-масив порядку ходів), current_turn (ID гравця, що ходить) і created_at;
- всіх учасників сесії записують у впорядкований список game:{session_id}:players, а для кожного гравця створюється хеш game:{session_id}:player:{user_id} із початковими атрибутами: balance, position, in_jail, jail_turns_left, immune_to_jail, last_roll, color, username, avatar, acted_props;
- поле гри ініціюється створенням хешів game:{session_id}:property:{pid} за константою PROPERTY_DATA, де вказано type, ціну покупки, ціну будинку, масив ренти, групу монополії, а також owner="", houses=0, mortgaged=0;
- стан поточної фази гри зберігається в хеші game:{session_id}:turn_state, де поля phase (наприклад, new_game, roll_dice, await_purchase, await_pay_rent, await_pay_utility, await_pay_debt, game_over), current_player, expires_at (UNIX-час таймауту) і action_payload (JSON-

контекст рішення) дають змогу turn_manager автоматично обробляти прострочені ходи;

– журнали подій гри та повідомлення чату хронологічно зберігаються у списках game:{session_id}:logs і game:{session_id}:chat відповідно; кожна операція create_game_log або append_chat_message додає новий запис і відразу штовхає його через group_send відповідній групі клієнтів.

Завдяки такій моделі всі дії клієнтів (створення лобі, приєднання, старт гри, хід гравця, купівля/аренда/будівництво/залог/банкрутство, чат-повідомлення) одразу відображаються в Redis. WebSocket-консьюмери підписані на відповідні канали (як lobbies:updates, так і game:{session_id}:updates), читають pub/sub-повідомлення через client.pubsub().get_message() і шлють їх клієнтам. Це забезпечує синхронний стан гри для всіх учасників без необхідності періодичного опитування API та мінімізує затримки.

3.4 Приклади використання алгоритмів та методів

У цьому підрозділі наведені два ключові алгоритми, які дозволяють забезпечити надійність ігрового процесу та централізовану обробку подій без дублювання логіки у різних компонентах.

3.4.1 Менеджер ходів із таймаутом

Алгоритм реалізовано як фонову корутину monitor_all_games(), яка запускається при старті сервера і виконується в циклі: після кожної секунди паузи виконується читання з Redis множини активних ігрових сесій game:active (див. рис. 3.6). Для кожної сесії дістається хеш turn_state із полями phase (поточна фаза ходу), current_player (ID гравця) та expires_at (мітка часу завершення фази). Якщо поточний час більший або рівний expires_at і фаза не змінилася з моменту останньої перевірки, викликається функція handle_timeout_skip(), яка обробляє конкретну фазу:

– для await_purchase автоматично відхиляє купівлю, створює лог і передає хід далі;

– для await_pay_rent і await_pay_utility намагається списати кошти. Якщо грошей недостатньо, ініціює ліквідацію активів або банкрутство;
– для roll_dice фіксує пропуск ходу;
– для інших фаз просто створює узагальнений лог про пропуск.

Після обробки таймауту алгоритм викликає pass_turn_to_next(), що оновлює стан гри та повідомляє клієнтів через WebSocket [1] про новий хід.

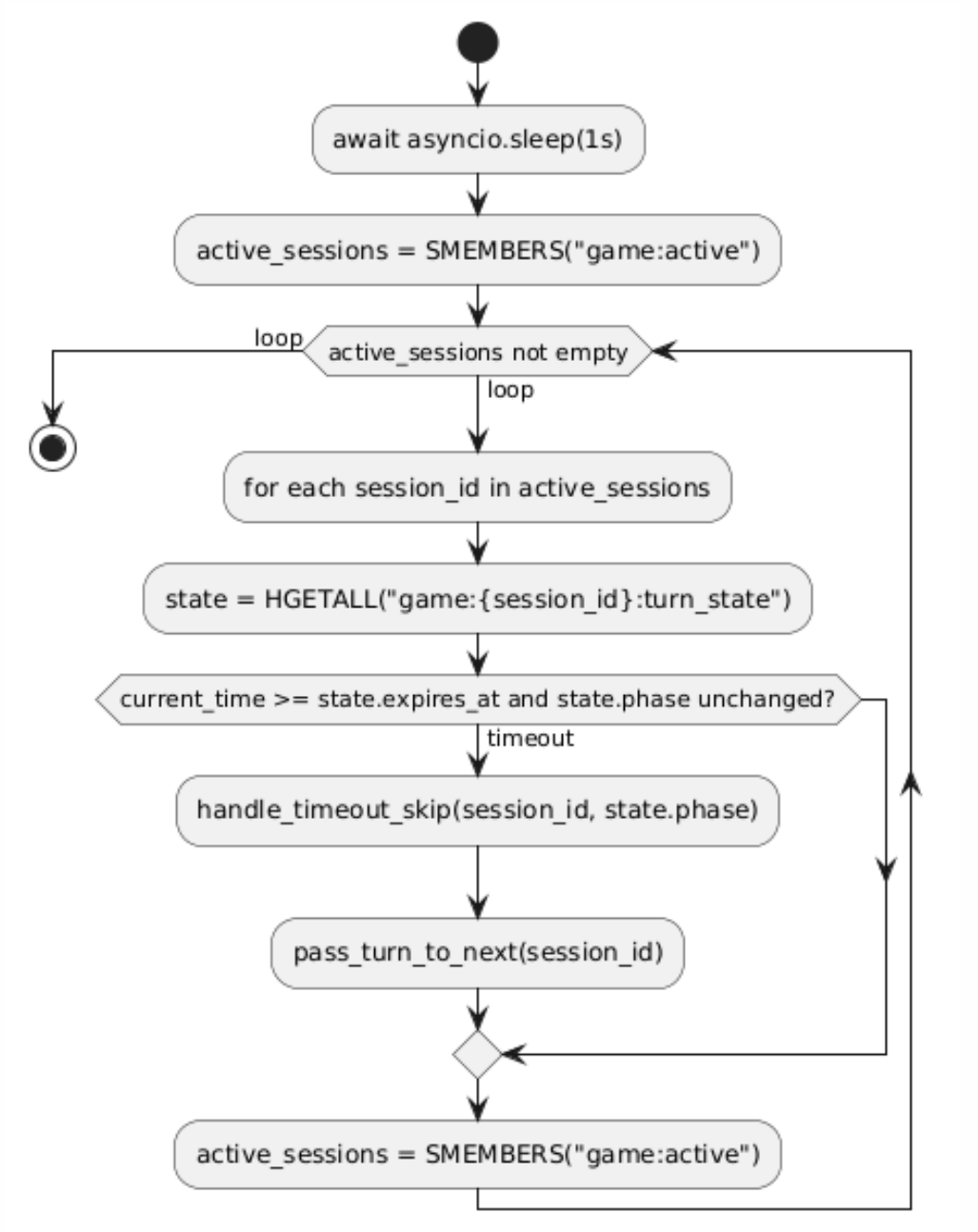


Рисунок 3.6 – UML-діаграма активності перевірки актуальності фази гри

Цей підхід дозволяє централізовано контролювати дедлайни всіх фаз ходу без надлишкових перевірок у кожному консьюмері і гарантує відновлення гри навіть у разі втрати клієнтської відповіді.

3.4.2 Передача ходу та обробка заставлених властивостей

Функція pass_turn_to_next(session_id) відповідає за визначення наступного гравця та одночасну обробку заставлених активів (див. рис. 3.7).

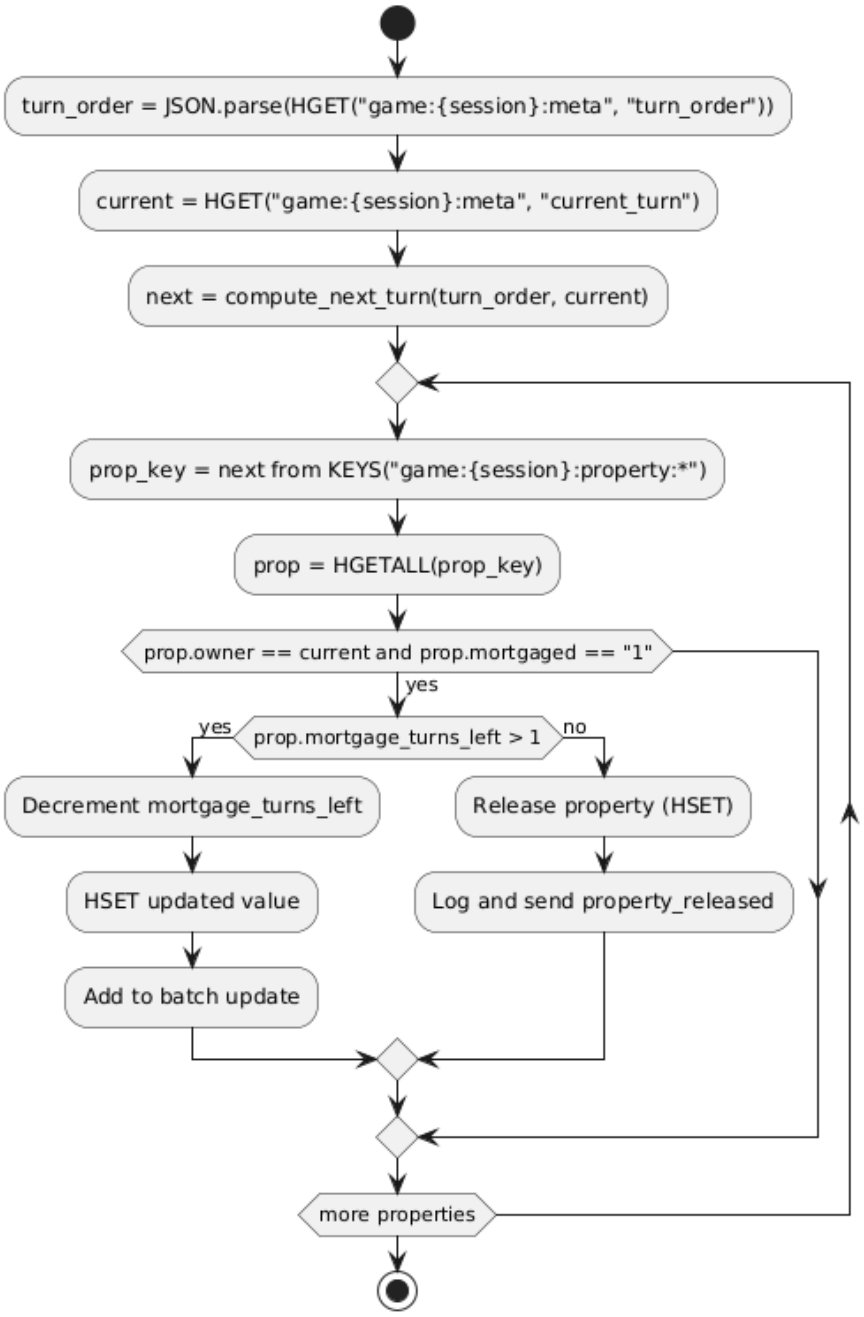


Рисунок 3.7 – UML-діаграма активності обробки та передачі ходу

Спочатку з Redis-хеша `game:{session_id}:meta` читається список `turn_order`. Якщо нинішнього гравця вже немає в черзі (вихід, банкрутство) або його статус було змінено раніше в поточному ході, виконується пошук першого доступного гравця в оновленому списку. Цей підхід гарантує, що черговість ніколи не порушується, навіть у випадку непередбаченої втрати з'єднання.

Далі цикл обробки властивостей:

- за допомогою `KEYS("game:{session_id}:property:*")` отримуємо всі ключі активів;
- для кожного ключа через `HGETALL` дістаємо атрибути `owner` та `mortgaged`;
- якщо `mortgaged == "1"` та `mortgage_turns_left > 1`, зменшуємо лічильник на 1 і повертаємо оновлене значення через `HSET`;
- якщо `mortgage_turns_left <= 1`, автоматично звільняємо власність (`HSET owner=""`, `mortgaged="0"`), записуємо подію `property_released` у лог гри та публікуємо відповідне повідомлення клієнтам.

Після завершення обробки застави:

- новий поточний гравець записується в `game:{session_id}:meta` через `HSET current_turn = next`;
- в `game:{session_id}:turn_state` встановлюється фаза `roll_dice` із полем `deadline = now + TIMEOUT`, де `TIMEOUT` — налаштований проміжок для кидка кубиків;
- через Redis Pub/Sub [6] надсилаються події `turn_update` і `property_updates` всім підписаним WebSocket-клієнтам.

Завдяки цьому методу відбувається одночасне оновлення черги ходу та логіки заставлених активів, що позбавляє необхідності розносити ці перевірки по різним місцям коду.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Створення ігрової сесії

При отриманні HTTP-запиту на старт гри від користувача викликається функція start_lobby(lobby_id: str) -> str, яка послідовно переводить Lobby у повноцінну ігрову сесію. Спочатку методом get_lobby_meta(lobby_id) зчитуються з Redis метадані лоббі, зокрема масив players зі списком ідентифікаторів учасників. Потім генерується унікальний session_id за допомогою uuid.uuid4() та фіксується поточний час UTC у форматі ISO.

```
session_id = str(uuid.uuid4())
now = datetime.utcnow().isoformat()
client.sadd("game:active", session_id)
meta_key = f"game:{session_id}:meta"
client.hset(
    meta_key,
    mapping={
        "session_id": session_id,
        "current_turn": players[0],
        "turn_order": json.dumps(players),
        "created_at": now,
    },
)
client.expire(meta_key, 24 * 3600)
```

Після виклику start_lobby новий session_id додається в множину game:active через SADD, що забезпечує O(1) вставку та миттєве отримання списку активних ігор. Під ключем game:{session_id}:meta зберігаються базові параметри сесії: ідентифікатор, поточний гравець, порядок ходів та час створення. Для автоматичної очистки неактивних ігор у Redis встановлено TTL 24 години, що знижує ризик накопичення «застарілих» сесій і полегшує адміністрування.

Далі формується Redis список гравців game:{session_id}:players та ініціалізується стан кожного учасника гри: із Django моделі User вилучаються

username та avatar_url, обчислюється колір фішки за індексом у константі PLAYER_COLORS, а всі початкові параметри (баланс 1500, позиція на старті, прапор перебування в тюрмі, список дій тощо) записуються в Redis хеш game:{session_id}:player:{user_id}.

```
players_key = f"game:{session_id}:players"
for uid in players:
    client.rpush(players_key, uid)
for idx, uid in enumerate(players):
    user = User.objects.filter(pk=uid).first()
    client.hset(
        players_key,
        mapping={
            "balance": 1500,
            "position": 0,
            "in_jail": 0,
            "jail_turns_left": 0,
            "immune_to_jail": 0,
            "username": user.username if user else "",
            "avatar": user.avatar.url if user and user.avatar else "",
            "color": PLAYER_COLORS[idx % len(PLAYER_COLORS)],
        },
    )
client.expire(players_key, 24 * 3600)
```

Після встановлення станів гравців створюється налаштування ігрового поля. Використовуючи константу PROPERTY_DATA, що містить опис усіх клітинок, у Redis під ключами game:{session_id}:property:{property_id} через HSET зберігаються лише змінні атрибути (тип, власник, кількість будинків), а незмінні беруться з коду:

```
for pid, data in PROPERTY_DATA.items():
    key = f"game:{session_id}:property:{pid}"
    mapping = {"type": data["type"], "owner": "", "skin_path":
compute_skin(pid)}
    if data["type"] == "company":
```

```
            mapping.update({
                "name": data["name"],
                "buy_price": data["buy_price"],
                "rent": json.dumps(data["rent"]),
                "houses": 0,
                "mortgaged": 0,
            })
        client.hset(key, mapping=mapping)
```

Щоб запобігти спадщині старих сесій, видаляються логи та чат:

```
client.delete(f"game:{session_id}:logs", f"game:{session_id}:chat")
client.hset(f"game:{session_id}:turn_state", mapping={"phase": "new_
game"})
remove_lobby(lobby_id)
client.publish(
    "lobbies:updates",
    json.dumps({"action": "start", "session_id": session_id,
"players": players}),
)
```

Після виконання очищення Redis повертається до стану, якого очікують клієнти при старті нової сесії, без залишкових даних від попередніх ігор. Запис mapping={"phase": "new_game"} у хеші turn_state необхідний для WS-consumers гри: при першому підключенні клієнта вони перевіряють цю фазу й розуміють, що гра на стадії початку. Далі consumer випадково обирає гравця, що ходить першим, оновлює його статус у turn_state і негайно надсилає йому повідомлення про початок ходу через WebSocket. Код повертає `session_id`, який використовується далі в HTTP-відповідях і WebSocket-консьюмерах для коректного маршрутування подальших подій.

4.2 Реалізація API та WebSocket-консьюмерів

HTTP-API для керування лоббі та старту сесії побудоване з використанням Django REST Framework. Усі кінцеві точки, зокрема LobbyCreateAPI, LobbyListAPI,

LobbyJoinAPI тощо, наслідують від DRF-APIView і захищені класом permission_classes = [IsAuthenticated], що автоматично відкидає неавторизовані запити. Сериалізатори (LobbyCreateSerializer тощо) відповідають за валідацію вхідних даних. Наприклад, при створенні лоббі у POST-тілі приймаються name, region, max_players та прапор invite_only.

```
class LobbyCreateAPI(APIView):
    permission_classes = [IsAuthenticated]
    def post(self, request):
        serializer = LobbyCreateSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        meta = serializer.validated_data
        create_lobby(meta)
        return Response(meta, status=status.HTTP_201_CREATED)
```

Аутентифікація WebSocket-з’єднань реалізована за допомогою кастомного middleware JwtAuthMiddleware, яке перехоплює параметр token із query string, розшифровує його через jwt.decode із налаштованим секретним ключем та алгоритмом, і в разі успіху записує user_id у scope. Це дозволяє в асинхронних конс’юмерах обробляти тільки авторизовані з’єднання без використання сесій чи куків.

```
class JwtAuthMiddleware:
    def __init__(self, app): self.app = app
    async def __call__(self, scope, receive, send):
        qs = scope.get("query_string", b"").decode()
        token = next((p.split("=",1)[1] for p in qs.split("&") if
p.startswith("token=")), None)
        if token:
            payload = jwt.decode(token, settings.SECRET_KEY,
algorithms=[...])
            scope["user_id"] = payload.get("user_id")
        return await self.app(scope, receive, send)
```

Основний WebSocket-консюмер (LobbyConsumer та аналогічні для гри) при підключенні виконує await self.accept(), після чого викликає синхронну функцію list_lobbies() в окремому потоці через database_sync_to_async, щоб надіслати початкове повідомлення {"type":"init","lobbies": [...]} із повним переліком доступних лоббі. Після цього за допомогою redis_client.pubsub() відбувається підписка на канал "lobbies:updates".

```
await self.accept()
lobbies = await database_sync_to_async(list_lobbies)()
await self.send_json({"type": "init", "lobbies": lobbies})
self.pubsub = redis_client.pubsub()
self.pubsub.subscribe("lobbies:updates")
asyncio.create_task(self.listen_pubsub())
```

Створюється фонове завдання, що в нескінченному циклі читає повідомлення з Pub/Sub [6] та надсилає їх клієнту як JSON типу lobby_update або lobby_start.

```
async def listen_pubsub(self):
    for message in self.pubsub.listen():
        data = json.loads(message["data"])
        await self.send_json({"type": message["action"], **data})
```

Така архітектура забезпечує реактивне оновлення списку кімнат і старту гри без необхідності періодичних HTTP-запитів. Для ігрових подій (ходи, баланс, лог тощо) реалізовано аналогічний GameConsumer, який підписується на канали "game_{session_id}" і транслює події типу game_move, game_log, game_balance_update тощо безпосередньо групі гравців сесії.

4.3 Обробка банкрутства гравця

У функції pay_rent(session_id, payer_id, owner_id, amount) перша перевірка достатності коштів у платника відбувається через читання балансу з Redis-хешу:

```
payer_key = f"game:{session_id}:player:{payer_id}"
payer_balance = int(client.hget(payer_key, "balance") or 0)
```

Якщо платіж можна здійснити, через hincrby зменшується баланс платника та збільшується баланс отримувача. Якщо ж готівки недостатньо, викликається calculate_max_balance(session_id, payer_id), яка перебирає всі ключі game:{session_id}:property:* і рахує вартість активів гравця за половину купівельної ціни (buy_price * 0.8) та будинків/готелів (house_price * 0.5, hotel_price * 0.5).

```
def calculate_max_balance(session_id, player_id):
    total_assets = 0
    for key in client.keys(f"game:{session_id}:property:*"):
        prop = client.hgetall(key) or {}
        if prop.get("owner") != str(player_id):
            continue
        if prop["mortgaged"] == "0":
            total_assets += int(int(prop["buy_price"]) * 0.8)
        houses = int(prop.get("houses","0"))
        if houses:
            total_assets += houses * int(prop.get("house_price","0"))
* 0.5
    return total_assets
```

Якщо сума готівки та активів покриває орендну плату, встановлюється фаза await_pay_debt у хеші game:{session_id}:turn_state разом із payload, що містить currentCash, totalOwed, totalAssets, creditorId і строк дії (через time.time() + 180). Це дозволяє клієнту відобразити інтерфейс продажу активів протягом трьох хвилин.

```
payload = {...}
client.hset(
    f"game:{session_id}:turn_state",
    mapping={"phase":"await_pay_debt","action_payload":
json.dumps(payload),"expires_at":str(deadline)}
)
```

Якщо ж навіть повний розрахунок активів не дозволяє сплатити борг, викликається process_bankruptcy(session_id, payer_id, payer_name). У цій функції підбираються всі властивості гравця, очищуються їхні атрибути власника і застави.

```
for prop_key in client.keys(f"game:{session_id}:property:*"):
    prop = client.hgetall(prop_key) or {}
    if prop.get("owner") == player_id:
        client.hset(prop_key,
mapping={"owner":"","mortgaged":"0","houses":"0"})
```

Далі зчитується поточний порядок ходів із game:{session_id}:meta:turn_order, видаляється ID банкрута, а йому присвоюється місце place = total_players. Якщо після його виключення залишається лише один гравець, для нього задається place = 1, формується рейтинг усіх учасників і фаза game_over записується з payload рейтингу та тайм-аутом 300 секунд:

```
if len(new_order) == 1:
    client.hset(turn_state_key, mapping={
        "phase":"game_over",
        "action_payload":json.dumps({"rankings":rankings}),
        "expires_at":str(time.time()+300)
    })
    return{"action":"game_over","turn_state":
client.hgetall(turn_state_key)}
```

Якщо гравців більше одного, функція повертає дію bankrupt із переліком повернутих властивостей, записаним у логі, та оновленим turn_order. Таким чином реалізується повний цикл банкрутства: від спроби сплати боргу до остаточного виключення з гри та підбиття підсумків.

5 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

У результаті реалізації проєкту створено повноцінну систему онлайн-гри “MonopolyUA”, яка в повному обсязі відповідає початковим технічним та функціональним вимогам, закріпленим у ТЗ. Система охоплює весь ігровий цикл — від створення та керування лобі до обробки ходів, фінальних розрахунків активів і моделі банкрутства — і забезпечує високу надійність та масштабованість.

Першим кроком реалізації стало побудова HTTP-інтерфейсу на базі Django REST Framework із permission_classes = [IsAuthenticated], що гарантує захист усіх кінцевих точок API на рівні валідації запитів і авторизації. CRUD-операції над лобі і сесіями реалізовані через APIView із DRF-серіалізаторами, які провадять автоматичну перевірку полів (наприклад, max_players — у межах 2–4, region — зі списку підтримуваних значень тощо). OpenAPI-документація генерується автоматично, полегшуючи фронтенд-інтеграцію і тестування сторонніми інструментами.

Для подій реального часу використано Django Channels [9] з WebSocket-consumers. Кожне підключення перевіряє JWT-токен та origin-заголовок згідно з профілем RFC 7523 [10]: у consumers реалізовано middleware, що читає Authorization: Bearer <token> із заголовка, декодує JWT, перевіряє поля iss, exp і aud, а також порівнює origin із дозволеним списком. Поточна конфігурація використовує ws://, проте перехід на wss:// для продакшен-середовища вимагає лише оновлення налаштувань SSL у channels.layers і проксі-сервера.

Redis виступає одночасно сховищем стану та шиною повідомлень. Метою було зменшити затримки і виключити конфлікти при паралельних запитах, тому метадані кожного лобі зберігаються в Redis-хеші lobby:{lobby_id}:meta, а список гравців — у множині lobby:{lobby_id}:players. При будь-якій дії (створення/зміна лобі, приєднання, вихід гравця, старт гри) виконується публікація JSON-повідомлення у канал lobbies:updates, до якого підписані всі активні WebSocket-consumers. Такий підхід забезпечує затримки передачі даних на рівні

одиниць мілісекунд і дозволяє уникнути надлишкових HTTP-опитувань, що значно знижує навантаження на сервер при високій частоті подій.

Для кожної ігрової сесії створюється окремий Redis-хеш game:{session_id}:meta із полями current_turn, turn_order, created_at тощо, а також набір ключів для зберігання стану полів і гравців (game:{session_id}:player:{user_id}). Логіка ходів реалізована як кінцевий автомат: повідомлення make_move проходить через шар turn_manager, який обчислює нову позицію, застосовує бізнес-правила (купівля, оренда, іпотека, штрафи), записує результат у Redis і публікує game_move та game_log події.

Якість реалізації оцінюється як висока. Архітектура розв’язує задачі швидко, коректно й без втрати даних при нормальних умовах. Використання Redis відповідає профільним вимогам low-latency, real-time, ігрових застосунків – саме такі use-case описані у документації Redis. Архітектурний підхід – decoupled, stateless WebSocket сервери з централізованим Redis бенчмарком – оптимальний для розширення системи шляхом додавання нових worker-ів без перенастроювання backend-серверів.

Досягнуті результати також відповідають заданим в проєкті критеріям. Завдання з реалізацією всіх базових правил Monopoly, чатів, логування і WebSocket-оновлень виконано повністю. Важливим аналітичним висновком можна назвати: обрана структура дає максимальну адаптивність, оскільки під час навантаження можна безболісно масштабувати окремі компоненти (WebSocket-мапи, Redis) незалежно один від одного — без переробки логіки або структури коду.

Таким чином, досягнуті результати повністю відповідають поставленим технічним і функціональним вимогам: система швидка, надійна та масштабована, забезпечує повний цикл ігрового процесу з лоббі, логікою гри, реальним часом та чатом. Обрана архітектура дозволяє легко розширювати систему, додаючи нові worker-и та ігрові сесії без суттєвих змін.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1 Визначення плану впровадження

Розгортання та підтримка онлайн-сервісу «Монополія UA» здійснюються за три основні етапи: локальна розробка, тестове середовище (staging) та production. Локально проект піднімається за допомогою Docker -контейнерів: сервіс web (Django + Channels), сервіс db (PostgreSQL) та redis (Redis 6.x). Для цього достатньо виконати git clone репозиторію, створити .env з параметрами підключення до бази та Redis, а потім запустити docker-compose up --build -d. Після старту контейнерів необхідно прогнати міграції (python manage.py migrate) та, за потреби, завантажити початкові дані (loaddata initial_data.json).

Staging-середовище реплікує конфігурацію production, але на окремому сервері або в хмарі (AWS/GCP). Після злиття гілки develop в staging через CI/CD (наприклад, GitHub Actions) автоматично виконується тестування (pytest, flake8), збірка Docker-образів і деплой на тестову інфраструктуру. Це дозволяє протестувати оновлення без відчутного ризику для кінцевих користувачів.

У production-інфраструктурі застосовується горизонтальне масштабування: кілька екземплярів Django Channels запускаються в Docker Swarm або Kubernetes, а за балансування HTTP і WebSocket-запитів відповідає NGINX (або ELB в хмарі). Для даних користувачів та довготривалих записів використовується кластер PostgreSQL з реплікацією, а Redis налаштовано у вигляді кластеру з кількома вузлами для високої доступності. Ідентифікатори лоббі та ігрових сесій генеруються через uuid.uuid4(), що гарантує унікальність і відсутність колізій.

Процес релізу побудовано за моделлю Git Flow: розробники працюють у гілках feature/*, після закінчення функціоналу створюється Pull Request у develop, проходять тести і, в разі успіху, відбувається злиття в staging. Після перевірки в тестовому середовищі — гілка staging зливається в main, і CI/CD автоматично розгортає нову версію на production.

ВИСНОВКИ

У ході реалізації онлайн-гри “MonopolyUA” створено надійну, швидку й масштабовану платформу, яка повністю відповідає поставленим технічним та функціональним вимогам. Виконано весь цикл: від створення та керування ігровими лоббі до детальної бізнес-логіки Monopoly (купівля, будівництво, застави, банкрутство, Jail, Chance/Kazna), а також організовано оновлення стану у реальному часі через WebSocket і Redis Pub/Sub. Архітектура DRF + JWT для HTTP та WebSocket із Redis як єдиним джерелом правди забезпечує безпеку, відмовостійкість і можливість горизонтального масштабування без втручання в кодову базу. Репозиторій проєкту доступний на GitHub за наступною адресою: https://github.com/NureDukhotaIvan/MonopolyUA.git.

Для подальшого вдосконалення проєкту доцільно реалізувати такі напрями:

- персистентність та відновлення стану. Додати AOF або RDB-снапшоти в Redis, щоб після рестарту зберігати ігрові сесії та історію ходів; розглянути використання Redis Streams для гарантованої доставки подій;
- моніторинг і логування. Інтегрувати систему метрик (Prometheus/Grafana) для відстеження затримок, навантаження Redis, кількості активних сесій; вдосконалити логування помилок та винятків для швидшого виявлення та виправлення помилок у продакшені;
- розширення ігрових можливостей. Додати опцію створення турнірів, рейтингову систему, статистику гравців і чат-бот підтримки, щоб підвищити залученість та тривалість сесій.

Впровадження цих покращень зробить платформу більш стійкою, зручною для користувачів та готовою до високих навантажень, а також відкриє можливості для подальшого розвитку в напрямку великомасштабного мультиплеєрного сервісу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. WebSocket API [Електронний ресурс] – URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 07.06.2025)

2. Classic Monopoly Game Rules and History [Електронний ресурс] – URL: https://www.hasbro.com/common/instruct/monins.pdf (дата звернення: 07.06.2025)

3. Building a multiplayer game using WebSockets [Електронний ресурс] – URL: https://dev.to/sauravmh/building-a-multiplayer-game-using-websockets-1n63 (дата звернення: 8.06.2025)

4. Scaling Pub/Sub with WebSockets and Redis [Електронний ресурс] – URL: https://ably.com/blog/scaling-pub-sub-with-websockets-and-redis (дата звернення: 10.06.2025)

5. Implementing real-time functionality in Django with WebSockets and django-channels [Електронний ресурс] – URL: https://medium.com/@simeon.emanuilov/implementing-real-time-functionality-in-django-with-websockets-and-django-channels-7240f75ee9b5 (дата звернення: 10.06.2025)

6. Pub/Sub (Publish/Subscribe) – Redis Glossary [Електронний ресурс] – URL: https://redis.io/glossary/pub-sub/ (дата звернення: 10.06.2025)

7. Django REST Framework [Електронний ресурс] – URL: https://www.django-rest-framework.org/ (дата звернення: 07.06.2025)

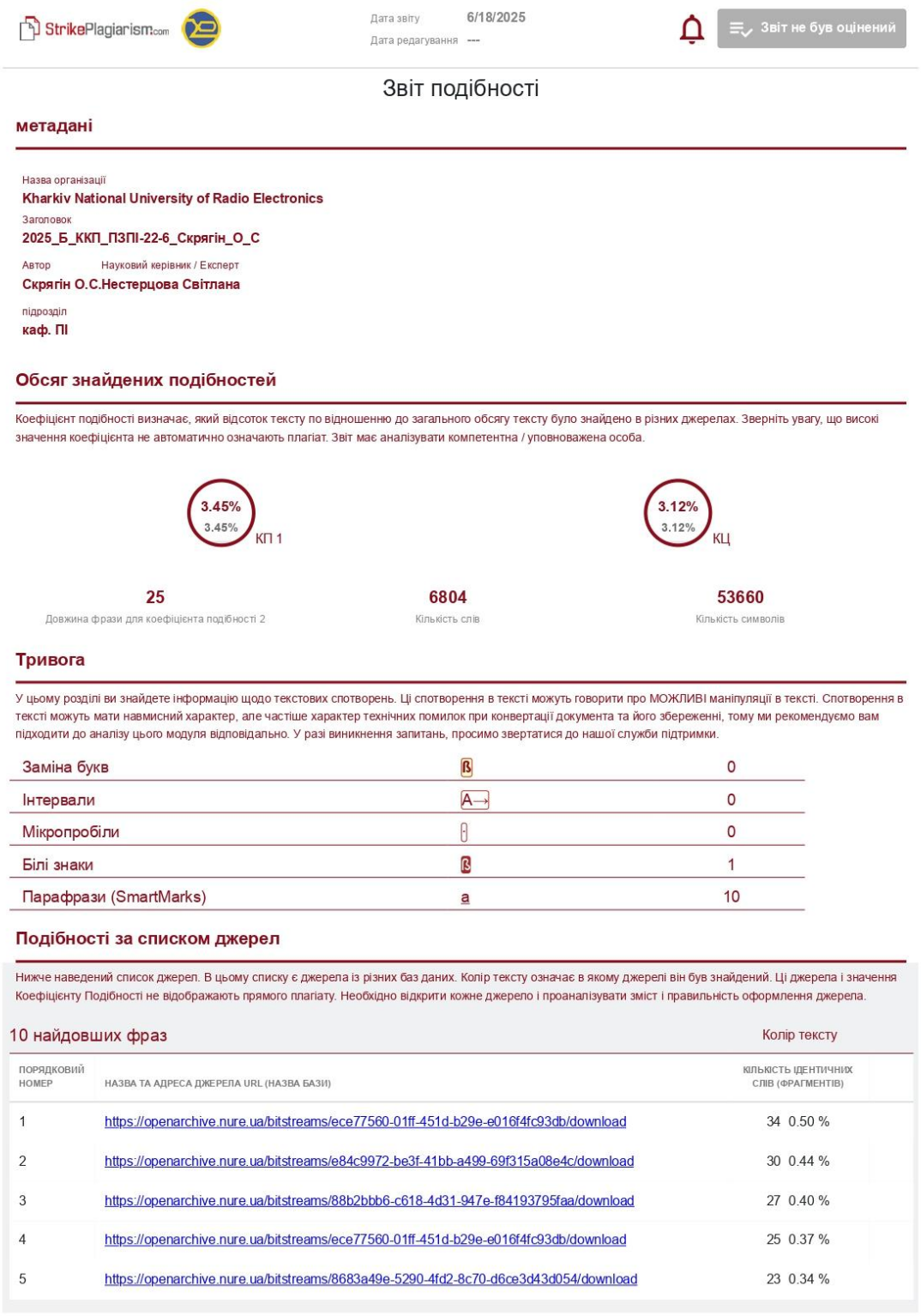
8. Redis Pub/Sub and Data Structures [Електронний ресурс] – URL: https://redis.io/docs/latest/develop/data-types/ (дата звернення: 07.06.2025)

9. Django Channels Documentation [Електронний ресурс] – URL: https://channels.readthedocs.io/en/stable/ (дата звернення: 10.06.2025)

10.RFC 7523: JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants [Електронний ресурс] – URL: https://datatracker.ietf.org/doc/html/rfc7523 (дата звернення: 12.06.2025)

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_Скрягін_О_С
Автор Науковий керівник / Експерт
Скрягін О.С.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.

3.45% 3.45% КП 1

3.12% 3.12% КЦ

25 Довжина фрази для коефіцієнта подібності 2

6804 Кількість слів

53660 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.

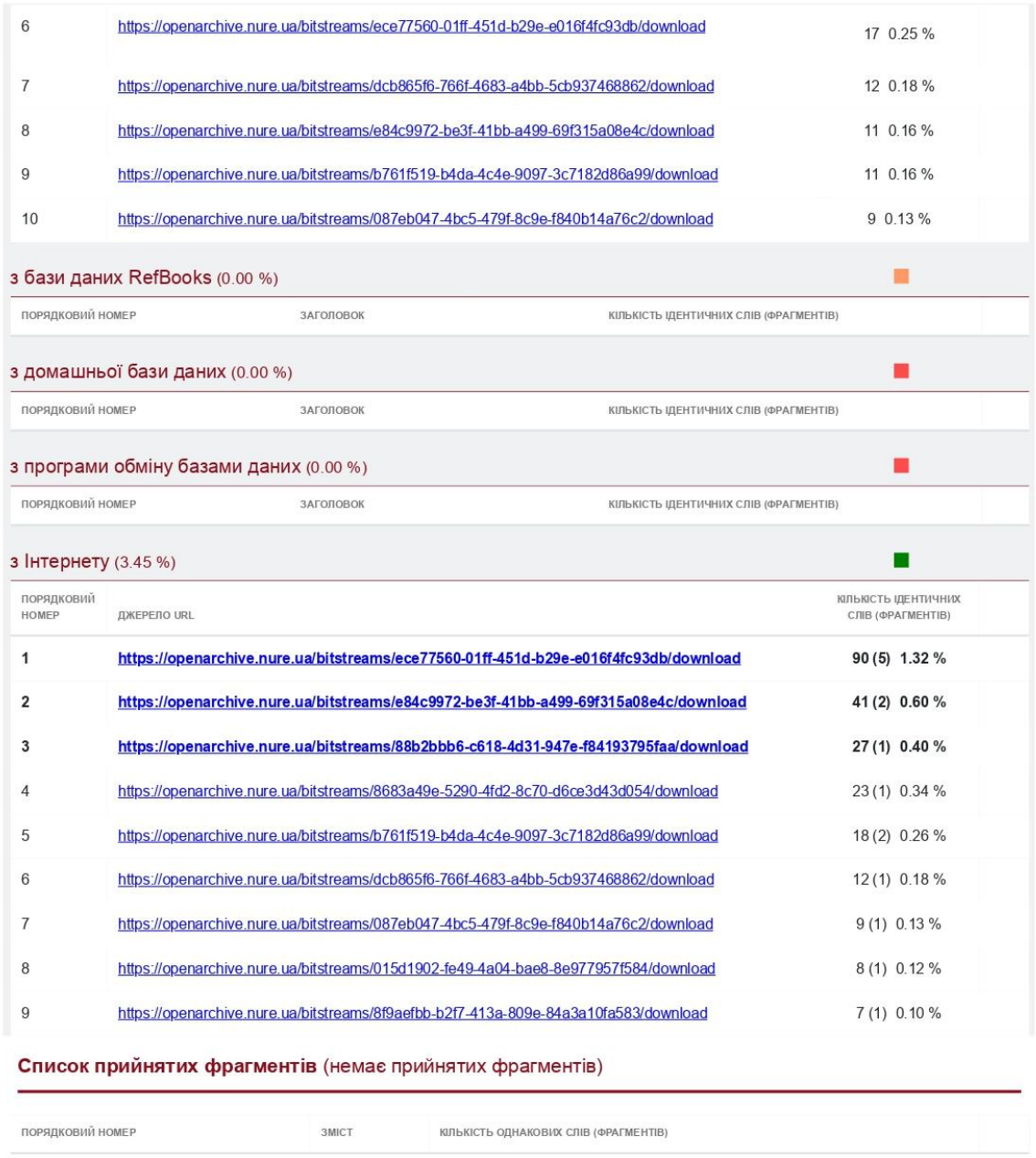
Заміна букв	ß	0
Інтервали	A⟶	0
Мікропробіли	·	0
Білі знаки	ß	1
Парафрази (SmartMarks)	a	10

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 0.50 %
2	https://openarchive.nure.ua/bitstreams/e84c9972-be3f-41bb-a499-69f315a08e4c/download	30 0.44 %
3	https://openarchive.nure.ua/bitstreams/88b2bbb6-c618-4d31-947e-f84193795faa/download	27 0.40 %
4	https://openarchive.nure.ua/bitstreams/ece77560-01ff-451d-b29e-e016f4fc93db/download	25 0.37 %
5	https://openarchive.nure.ua/bitstreams/8683a49e-5290-4fd2-8c70-d6ce3d43d054/download	23 0.34 %

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



6	https://openarchive.nure.ua/bitstreams/ece77560-01ff-451d-b29e-e016f4fc93db/download	17 0.25 %
7	https://openarchive.nure.ua/bitstreams/dcb865f6-766f-4683-a4bb-5cb937468862/download	12 0.18 %
8	https://openarchive.nure.ua/bitstreams/e84c9972-be3f-41bb-a499-69f315a08e4c/download	11 0.16 %
9	https://openarchive.nure.ua/bitstreams/b761f519-b4da-4c4e-9097-3c7182d86a99/download	11 0.16 %
10	https://openarchive.nure.ua/bitstreams/087eb047-4bc5-479f-8c9e-f840b14a76c2/download	9 0.13 %

з бази даних RefBooks (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з домашньої бази даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з програми обміну базами даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з Інтернету (3.45 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**https://openarchive.nure.ua/bitstreams/ece77560-01ff-451d-b29e-e016f4fc93db/download**	**90 (5) 1.32 %**
2	**https://openarchive.nure.ua/bitstreams/e84c9972-be3f-41bb-a499-69f315a08e4c/download**	**41 (2) 0.60 %**
3	**https://openarchive.nure.ua/bitstreams/88b2bbb6-c618-4d31-947e-f84193795faa/download**	**27 (1) 0.40 %**
4	https://openarchive.nure.ua/bitstreams/8683a49e-5290-4fd2-8c70-d6ce3d43d054/download	23 (1) 0.34 %
5	https://openarchive.nure.ua/bitstreams/b761f519-b4da-4c4e-9097-3c7182d86a99/download	18 (2) 0.26 %
6	https://openarchive.nure.ua/bitstreams/dcb865f6-766f-4683-a4bb-5cb937468862/download	12 (1) 0.18 %
7	https://openarchive.nure.ua/bitstreams/087eb047-4bc5-479f-8c9e-f840b14a76c2/download	9 (1) 0.13 %
8	https://openarchive.nure.ua/bitstreams/015d1902-fe49-4a04-bae8-8e977957f584/download	8 (1) 0.12 %
9	https://openarchive.nure.ua/bitstreams/8f9aefbb-b2f7-413a-809e-84a3a10fa583/download	7 (1) 0.10 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

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

ДОДАТОК Б

Слайди презентації

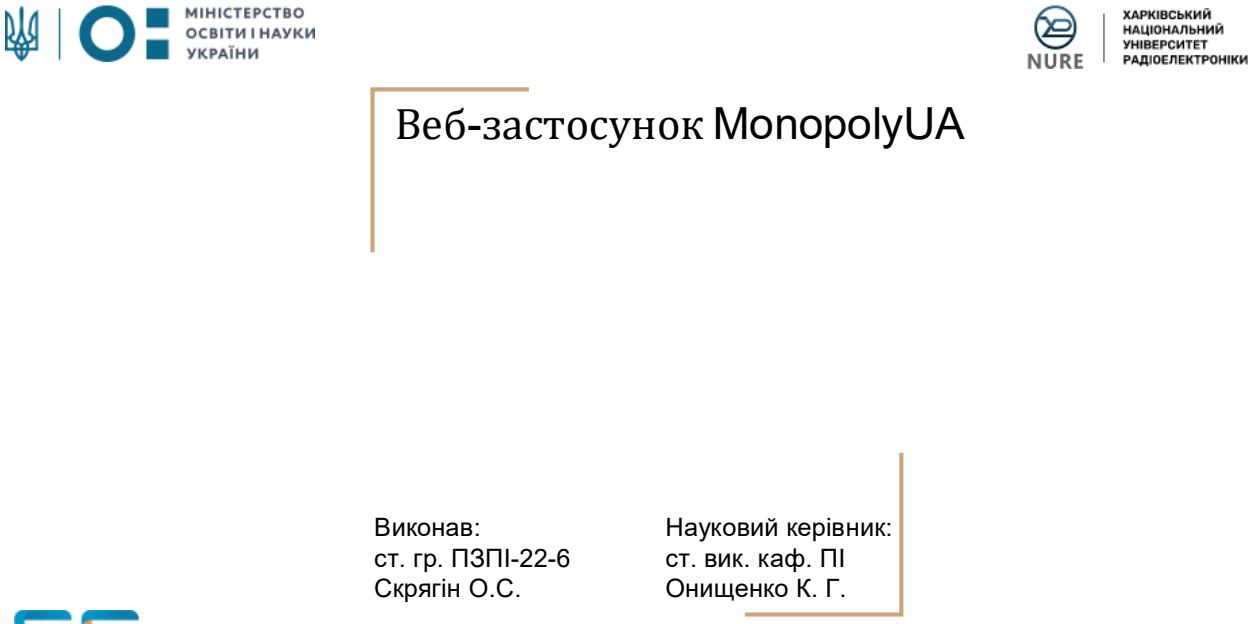


Рисунок Б.1 – Титульний слайд

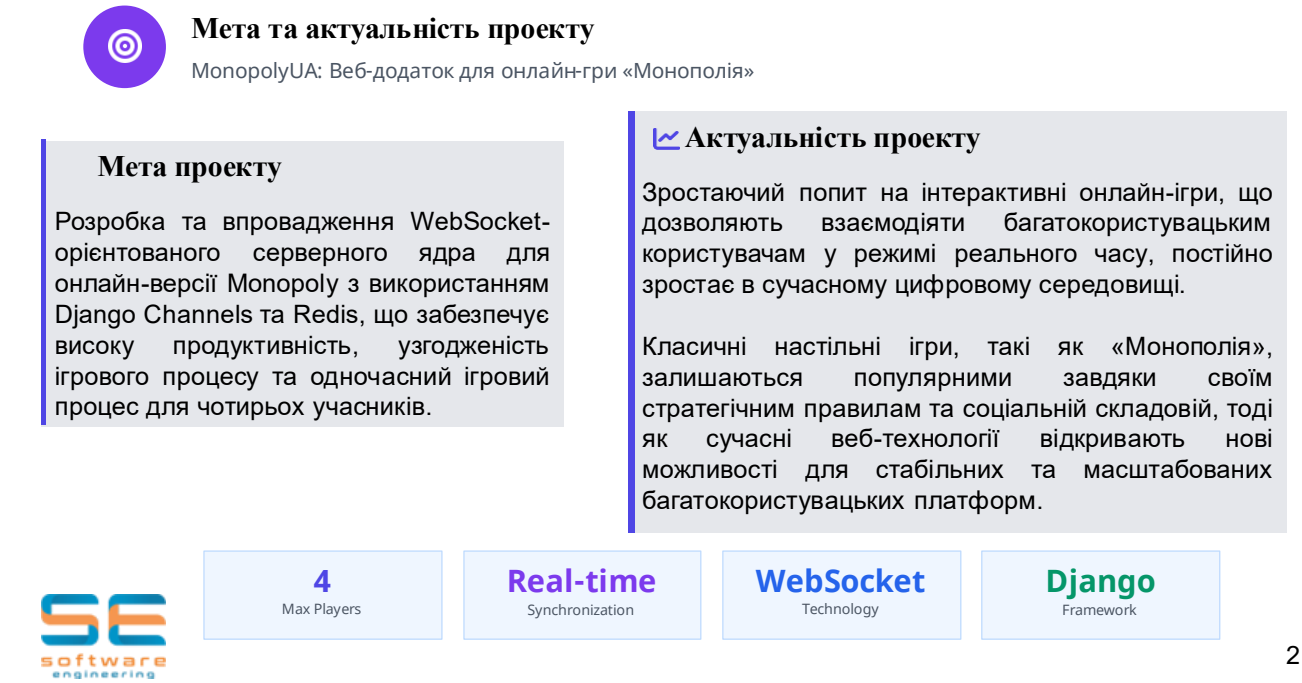


Рисунок Б.2 – Мета та актуальність проекту

Рисунок Б.3 – Аналіз проблеми та існуючі рішення

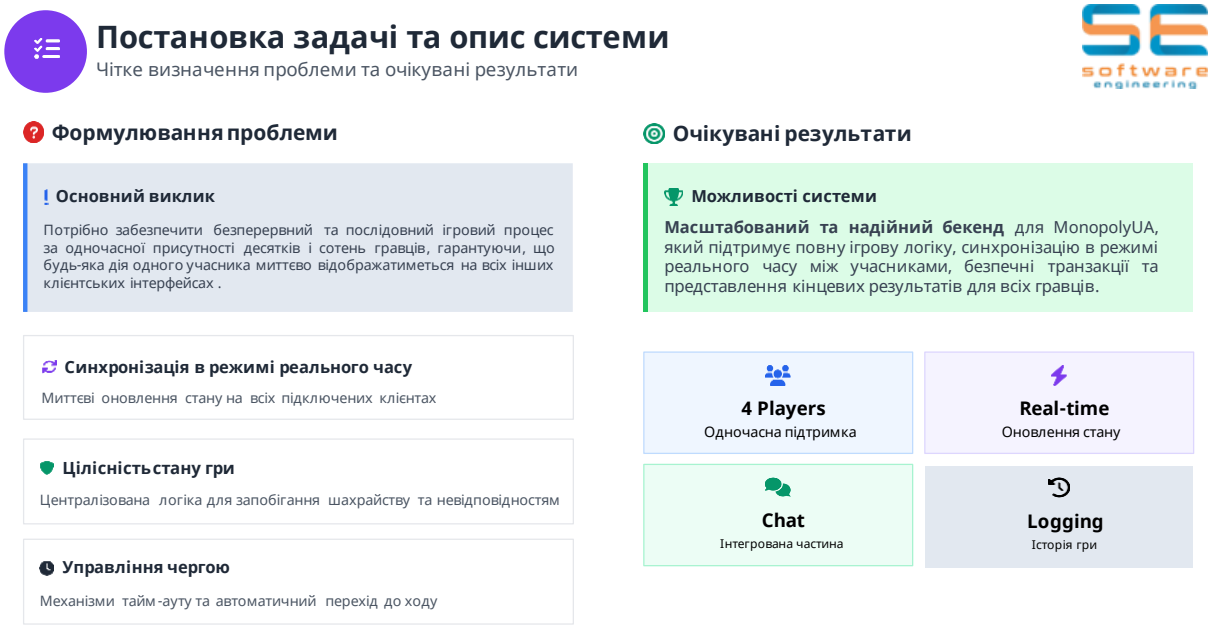


Рисунок Б.4 – Постановка задачі та опис системи

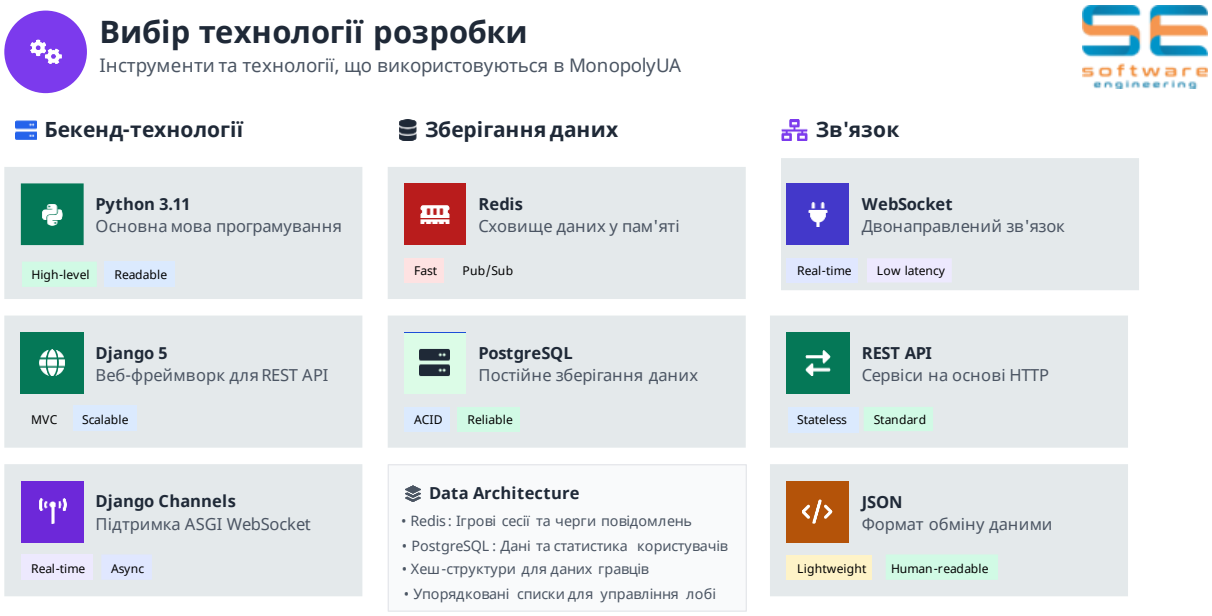


Рисунок Б.5 – Вибір технології розробки

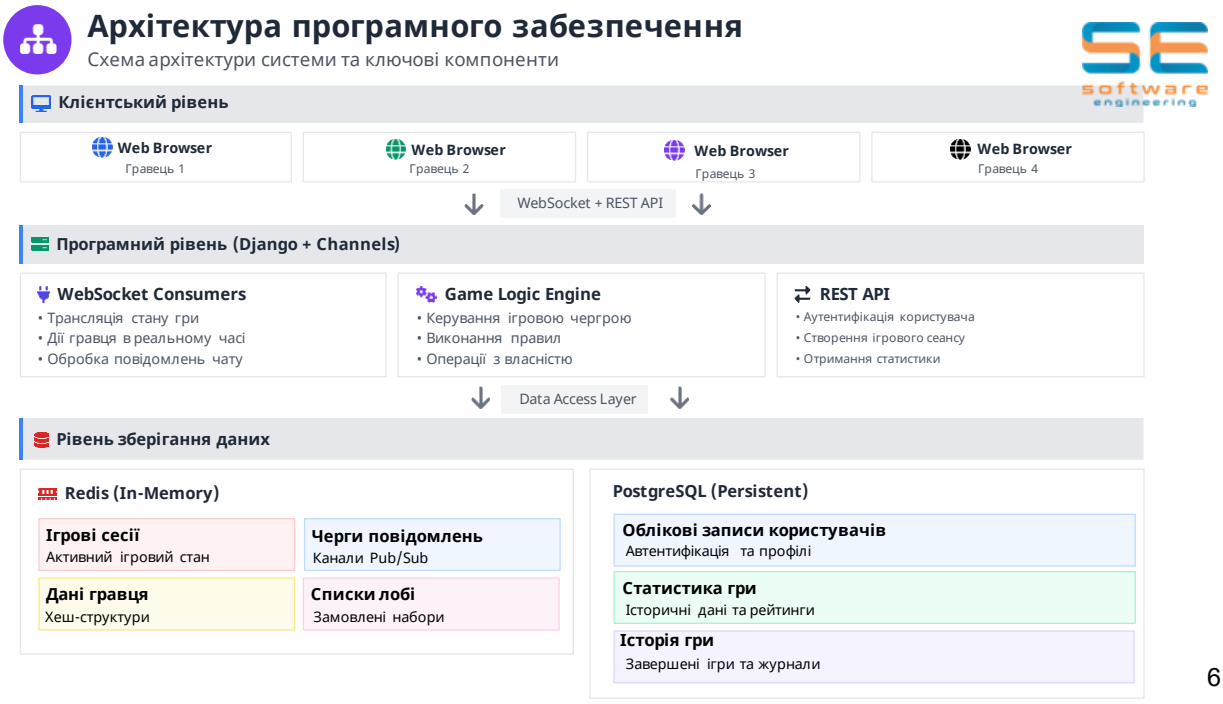
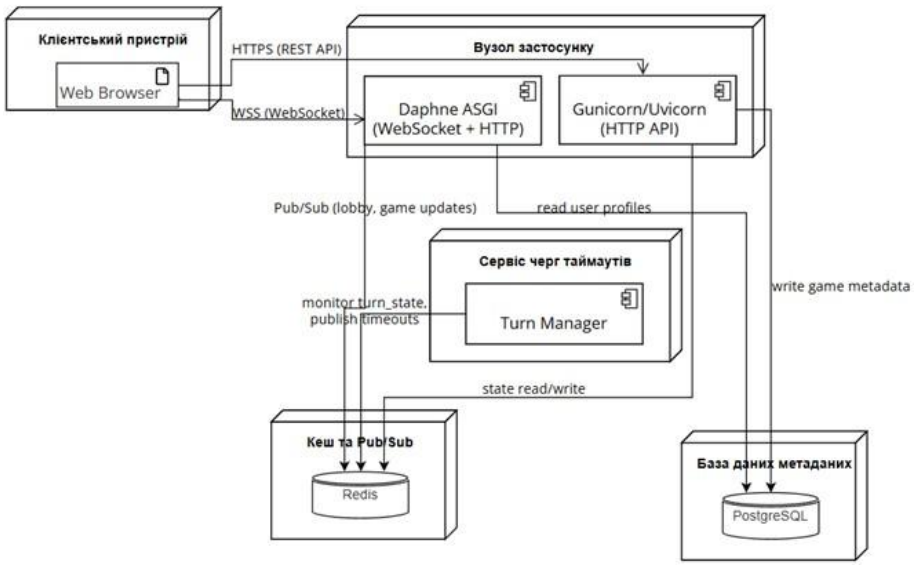


Рисунок Б.6 – Архітектура програмного забезпечення

Архітектура програмного забезпечення

Схема архітектури системи та ключові компоненти



7

Рисунок Б.7 – UML-діаграма розгортання

Процес розробки програмного забезпечення

Мови програмування та фреймворки

Етапи процесу розробки

1 Доменний аналіз
Вивчення правил гри «Монополія» та вимог до онлайн-ігор

2 Визначення завдання
Чітке формулювання проблеми та специфікація системних вимог

3 Дизайн ПЗ
Планування архітектури, проектування баз даних та специфікація API

4 Реалізація
Кодування, тестування та інтеграція всіх компонентів системи

Programming Languages

Python 3.11
Основна серверна мова

Переваги
- Висока читаність
- Багата екосистема
- Швидкий розвиток

Використання
- Реалізація ігрової логіки
- Розробка API
- Обробка даних

JavaScript
Взаємодія на стороні клієнта

Використовується для підключень клієнтів WebSocket, оновлень інтерфейсу користувача в режимі реального часу та взаємодії з ігровим полем

Frameworks and Libraries

Django 5.0 — MVC Pattern, ORM
Високорівневий веб-фреймворк на Python

Django Channels — Real-time, Async
Підтримка ASGI для обробки WebSocket

Django REST Framework — Serialization, Authentication
Потужний набір інструментів для створення веб-API

Redis-py — Caching, Pub/Sub
Клієнт Python для бази даних Redis

8

Рисунок Б.8 – Процес розробки програмного забезпечення

Приклади реалізації
Фрагменти коду та ключові функції розробки

АутентифікаціяWebSocket з’єднань

```python
class JwtAuthMiddleware:

    def __init__(self, app):
        self.app = app

    async def __call__(self, scope, receive, send):
        qs = scope.get("query_string", b"").decode()
        token = next((p.split("=",1)[1] for p in qs.split("&") if p.startswith("token=")), None)

        if token:
            try:
                UntypedToken(token)
                signing_key = settings.SIMPLE_JWT.get("SIGNING_KEY") or settings.SECRET_KEY
                algorithm   = settings.SIMPLE_JWT.get("ALGORITHM", "HS256")
                payload = jwt.decode(token, signing_key, algorithms=[algorithm])

                user = payload.get("user_id")
                if user:
                    scope["user_id"] = user
            except (TokenError, InvalidToken, jwt.DecodeError):
                pass

        return await self.app(scope, receive, send)
```


Рисунок Б.9 – Приклад реалізації аутентифікації WS з’єднань

Приклади реалізації
Фрагменти коду та ключові функції розробки

Менеджер ходів із таймаутом

```python
async def monitor_all_games():
    print("▶ monitor_all_games() запущен в фоновом loop'е")
    while True:
        await asyncio.sleep(1)
        active_sessions = client.smembers("game:active") or []
        now = int(time.time())

        for session_id in active_sessions:
            turn_state_key = f"game:{session_id}:turn_state"
            raw = client.hgetall(turn_state_key) or {}
            if not raw:
                continue

            phase = raw.get("phase")
            player = raw.get("current_player")
            expires_at = int(raw.get("expires_at", "0") or "0")

            if expires_at and now >= expires_at:
                state2 = client.hgetall(turn_state_key) or {}
                if state2.get("phase") == phase and state2.get("current_player") == player:

                    await handle_timeout_skip(session_id, player, phase, state2.get("action_payload"))
```


Рисунок Б.10 – Приклад реалізації менеджера ходів із таймаутом

Приклади реалізації

Фрагменти коду та ключові функції розробки

Асинхронний слухач PubSub

```python
async def listen_pubsub(self):
    try:
        while True:
            message = self.pubsub.get_message()
            if message:
                data = json.loads(message["data"])
                action = data.get("action")

                if action in ("create", "update", "kick", "remove"):
                    print("update")
                    await self.send(
                        json.dumps({"type": "lobby_update", "data": data})
                    )

                elif action == "start":
                    session_id = data.get("session_id")
                    players = data.get("players", [])

                    if self.userId and str(self.userId) in players:
                        await self.send(
                            json.dumps(
                                {"type": "lobby_start", "session_id": session_id}
                            )
                        )
            await asyncio.sleep(0.1)
    except asyncio.CancelledError:
        pass
```


Рисунок Б.11 – Приклад реалізації асинхронного слухача Pub/Sub

Результати та висновки

Результати проекту та майбутній розвиток

Досягнуті результати

Масштабована бекенд-система

Успішно створено надійний бекенд, який підтримує повну ігрову логіку, синхронізацію в режимі реального часу та безпечні транзакції.

Підтримка кількох гравців

Реалізовано одночасний ігровий процес для 4 гравців з оновленнями стану в режимі реального часу та керуванням ходами.

Безпека та цілісність

Забезпечено цілісність гри завдяки дотриманню правил на стороні сервера та централізованому управлінню станом.

Оптимізація продуктивності

Досягнуто швидкодію відгуку завдяки сховищу Redis в оперативній пам'яті та оптимізованим структурам даних.

Можливості використання

Ігрові платформи

Інтеграція з існуючими ігровими платформами та соціальними мережами для ентузіастів настільних ігор.

Освітні програми

Навчання фінансовій грамотності та плануванню бюджету в школах і університетах.

Комерційні рішення

Брендовані онлайн-версії класичних настільних ігор.
Підписка модель, мікротранзакції.

Соціальна інтеграція

Інтеграція з платформами соціальних мереж для покращення залучення користувачів та зростання вірусності.

Майбутній розвиток

Мобільні додатки

Нативні додатки для iOS та Android з офлайн-режимом та кросплатформною синхронізацією.

Супротивники ШІ

Гравці зі штучним інтелектом на основі машинного навчання, з різними рівнями складності та стилями гри.

Особливості налаштування

Вибір зовнішнього вигляду ігрових дошок, налаштування правил гри та тематичні варіації класичної гри «Монополія».

Інформаційна панель

Розширена статистика гравців, аналітика гри та показники продуктивності для змагальної гри.

Рисунок Б.12 – Результати та висновки

ДОДАТОК В

Специфікація програмного забезпечення (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").