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)

Ігровий програмний застосунок в жанрі Roquelike. Веб застосунок для підтримки гри.

(тема)

Виконав:

здобувач (ка) 3 курсу, групи ПЗПІ-22-4

Балицький Ігор

(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 доц. каф. ПІ Наталя ВАЛЕНДА

(посада, Власне ім’я, ПРІЗВИЩЕ)

|  |
| --- |
| Члени комісії (Власне ім’я, ПРІЗВИЩЕ, підпис) |
| Доц. Побіженко І.О. |
| Ст. викл. Широкопетлєва М.С. |
| Ст. викл. Новіков Ю.С. |

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-22-4 | Семестр | 6 |

**ЗАВДАННЯ**

***на курсовий проєкт(роботу) студента***

здобувачеві Балицькому Ігорю Олександровичу

(прізвище, ім’я, по батькові)

1. Тема роботи Ігровий програмний застосунок в жанрі Roquelike. Веб застосунок для підтримки гри.

2. Термін здачі студентом закінченої роботи „20” 06 2025 р.\_

3. Вихідні дані до проєкту: Розробка веб застосунку з використанням HTML, CSS, JavaScript та Python (FastAPI);з базою даних SQLite; організація адміністративної панелі та особистого кабінету користувача; реалізація системи відгуків, блогу та інформаційних сторінок про гру.

4. Перелік питань, що потрібно опрацювати в роботі

Проєктування структури веб застосунку та бази даних; реалізація API-ендпоінтів для основної функціональності; створення UI з урахуванням принципів доступності та зручності; реалізація багатомовного інтерфейсу; забезпечення безпеки авторизації та управління даними; підготовка демонстраційних матеріалів і тестування системи.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
| 1 | Аналіз предметної галузі | 23.03 – 28.03 | виконано |
| 2 | Розробка постановки задачі | 28.03 – 01.04 | виконано |
| 3 | Проєктування ПЗ | 01.04 – 09.03 | виконано |
| 4 | Програмна реалізація | 10.03 – 05.06 | виконано |
| 5 | Аналіз результатів | 05.06 – 08.06 | виконано |
| 6 | Підготовка пояснювальної записки. | 08.06 – 18.06 | виконано |
| 7 | Перевірка на наявність ознак академічного плагіату | 19.06 – 20.06 | виконано |
| 8 | Захист роботи | 22.06 | виконано |

Дата видачі завдання “21” 03 2025р.

Здобувач \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Керівник роботи доц. каф. ПІ Наталя ВАЛЕНДА

(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

**РЕФЕРАТ / ABSTRACT**

Пояснювальна записка містить: 74 с., 33 рис., 10 джерел., 2 додатки.

ВЕБЗАСТОСУНОК, ГРА, ІНТЕРФЕЙС, КОРИСТУВАЧ, АДМІНІСТРАТОР, ВІДГУКИ, НОВИНИ, EMAIL, SQLITE, PYTHON, JAVASCRIPT

Об’єктом розробки є веб застосунок до гри Brick Prince, який виконує функції інформаційного сайту для взаємодії користувачів з ігровим продуктом, з можливістю отримання копії гри, залишення відгуків, ознайомлення з новинами, а також адмініструванням контенту.

Метою проєкту є створення повноцінної клієнт-серверної системи з чітким поділом ролей (користувач / адміністратор), підтримкою відгуків, локалізацією, надсиланням email, переглядом стрічки новин та збереженням усіх відповідних даних у базі SQLite.

Під час реалізації було використано Python (FastAPI) для серверної логіки, JavaScript/HTML/CSS для фронтенду, SQLite як вбудовану базу даних, а також бібліотеки SMTP для надсилання електронних листів. Для зручності розробки застосовувались Live Server та Visual Studio Code. Реалізовано компоненти з перевіркою прав доступу, реакціями до відгуків (лайки/дизлайки), формами з валідацією та адаптивним дизайном.

У результаті проєкту створено ефективний та інтуїтивний веб застосунокіз сучасним інтерфейсом, повним функціоналом взаємодії з користувачами та інструментами адміністрування, який може масштабуватись і адаптуватися до подальших ітерацій гри Brick Prince.

WEB APPLICATION, GAME, INTERFACE, USER, ADMINISTRATOR, FEEDBACK SYSTEM, EMAIL, SQLITE, PYTHON, JAVASCRIPT

The object of the development is a web application for the Brick Prince game, functioning as an information platform for user interaction with the game. It includes features for downloading the game, leaving feedback, reading news, and performing administrative operations.

The goal of the project is to implement a full-featured client-server system with user and administrator roles, support for feedback and localization, email delivery, news management, and persistent data storage in an SQLite database.  
 The backend was built with Python (FastAPI), while the frontend used JavaScript, HTML, and CSS. The application uses SQLite as the database engine and SMTP libraries for sending emails. Development was done in Visual Studio Code with Live Server for rapid prototyping. The application includes access control logic, feedback reactions (likes/dislikes), form validation, and a responsive UI.

As a result, a modern and functional web platform was developed with intuitive user experience, full interaction capabilities, and administrative tools that are suitable for future extensions of the Brick Prince game.

**ЗМІСТ**

Вступ................................................................................................................................8

1. Аналіз предметної області……………………...........................................................9
   1. Аналіз предметної області……………………………………………….…9
   2. Виявлення та вирішення проблем…………………………………..……..10
   3. Аналіз аналогів програмного забезпечення………………...………...…..11
2. Постановка задачі........................................................................................................13
3. Архітектура та проєктування програмного забезпечення ………………………..15
   1. UML проєктування ПЗ ………………………………………....….………15
   2. Проєктування архітектури ПЗ ……………………………..….……..……19
   3. Проєктування структури зберігання даних…………….………..…..……22
   4. Приклади використаних алгоритмів та методів……………….…..……..27
   5. Проєктування UI/UX або іншого дизайну системи………….………..…30
4. Опис прийнятих програмних рішень…………………………………...……..……33
   1. Середовище розробки та особливості запуску…………...………………33
   2. Описання фізичної моделі бази даних………………………………..…..34
   3. Опис програмної реалізації……………………………………..….......….35
5. Аналіз отриманих результатів………………………………………………....……58

Висновки.....................................................................................................................61

Перелік джерел посилання........................................................................................62

ДОДАТОК А…………………………………………………………………………..63

ДОДАТОК Б……………………………………………………………….…………..64

**ПЕРЕЛІК СКОРОЧЕНЬ**

UI – User Interface – користувацький інтерфейс

UX – User Experience – досвід користувача

API – Application Programming Interface – інтерфейс прикладного програмування

DB – Database – база даних

SQL – Structured Query Language – мова структурованих запитів

CRUD – Create, Read, Update, Delete – створення, читання, оновлення, видалення

ER – Entity-Relationship – сутність-зв’язок (діаграма зв’язків сутностей)

IDE – Integrated Development Environment – інтегроване середовище розробки

SMTP – Simple Mail Transfer Protocol – простий протокол передавання пошти

HTML – HyperText Markup Language – мова розмітки гіпертексту

CSS – Cascading Style Sheets – каскадні таблиці стилів

JS – JavaScript – мова програмування JavaScript

CSV – Comma-Separated Values – формат файлів з розділенням даних комами

JSON – JavaScript Object Notation – формат обміну даними

REST – Representational State Transfer – архітектурний стиль передачі даних

VS Code – Visual Studio Code – текстовий редактор для розробки програмного забезпечення

SRS – Software Requirements Specification – специфікація вимог до програмного забезпечення

**ВСТУП**

У сучасному світі цифрових технологій комп’ютерні ігри займають одну з провідних позицій у сфері розваг, навчання та соціальної взаємодії. Популярність ігор невпинно зростає, і разом із цим зростає потреба у створенні не лише якісного ігрового контенту, але й у відповідних сервісах підтримки, які забезпечують гравцям комфортну взаємодію з грою, а розробникам — ефективне управління контентом і користувачами. У цьому контексті розробка веб застосунку для підтримки гри Brick Prince набуває особливої актуальності, оскільки дозволяє поєднати інтерактивність, зручність та сучасні технології для створення екосистеми навколо ігрового продукту.

Метою цієї роботи є створення повнофункціонального веб застосунку, який реалізує можливості реєстрації та авторизації користувачів, управління інформацією про гру, взаємодію з контентом, зокрема через систему новин і відгуків, а також надасть адміністративні інструменти для підтримки й модерування спільноти. Розробка такого сервісу не лише сприятиме підвищенню задоволеності користувачів і залученню нових гравців, а й створить умови для масштабування проєкту та гнучкого впровадження нових функціональних можливостей.

В процесі виконання поставленої мети необхідно вирішити низку завдань, серед яких глибокий аналіз предметної галузі та існуючих аналогів, розробка архітектури та технічного дизайну системи, а також реалізація фронтенд та бекенд-частин із дотриманням сучасних стандартів безпеки, продуктивності й зручності користування. Особливу увагу приділено питанню забезпечення якісної взаємодії користувачів із сервісом, що досягається завдяки інтуїтивному інтерфейсу, адаптивному дизайну і надійній системі управління даними.

Результати роботи можуть бути використані у сфері розробки ігрових веб платформ, створення спільнот гравців, а також у будь-яких проєктах, де необхідна інтеграція ігрового контенту з веб-сервісами.

**1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**

1.1 Аналіз предметної галузі

Сучасна індустрія комп’ютерних ігор переживає період стрімкого розвитку та трансформації, у центрі якого опинилися не лише безпосередньо ігрові продукти, а й супутні сервіси, що забезпечують підтримку, комунікацію та взаємодію гравців. Веб застосунки для ігор стають невід’ємною частиною ігрової екосистеми, адже саме через них формується спільнота, здійснюється управління контентом, відбуваються оновлення, і реалізується маркетингова підтримка проєкту.

У цій галузі особливого значення набуває забезпечення стабільного та зручного доступу користувачів до актуальної інформації про гру, можливість швидко отримувати оновлення, брати участь у форумах і подіях, а також отримувати технічну підтримку. Веб застосунки, що супроводжують ігри, здебільшого мають багатофункціональність: від авторизації та реєстрації користувачів до персоналізованих кабінетів, стрічок новин, систем донатів, модерації відгуків, адміністрування контенту і ролей користувачів.

Особливість цієї сфери полягає в тому, що такі застосунки повинні не лише бути технічно надійними та безпечними, але й мати приємний і інтуїтивний інтерфейс, що відповідає стилістиці самої гри та створює позитивний користувацький досвід. Забезпечення швидкого доступу до різноманітного функціоналу зручно організованими сторінками та меню є ключовим фактором, що визначає лояльність та залучення гравців.

Нині існує широкий спектр технологій для реалізації веб застосунків у цій галузі — від традиційних монолітних рішень до сучасних клієнт-серверних архітектур з розділенням фронтенду та бекенду. Популярність набирають SPA (single-page applications) та PWA (progressive web applications), які дозволяють забезпечити швидку реакцію інтерфейсу та комфортну роботу на різних пристроях. Вибір технологій напряму впливає на масштабованість, безпеку, продуктивність та подальший розвиток проєкту.

Для ігрових платформ важливо враховувати і соціальний аспект — інтеграцію з популярними соціальними мережами, системами аутентифікації, можливість залишати відгуки, оцінки, здійснювати пожертви (донати), що стимулює активність користувачів та фінансову підтримку проєкту. Веб застосунок стає точкою дотику між розробниками та аудиторією, підтримуючи довготривалу взаємодію.

Отже, предметна галузь веб застосунків для ігор — це поєднання сучасних IT-технологій, UX-дизайну, соціальних інтеграцій і високих стандартів безпеки, що забезпечують ефективне та зручне середовище для геймерів та адміністраторів ігрових проєктів.

1.2 Виявлення та вирішення проблем

Незважаючи на стрімкий розвиток ігрової індустрії та супутніх веб технологій, існує ряд суттєвих викликів і проблем, які постають перед розробниками веб застосунків для ігор, що потребують ефективних та інноваційних рішень.

Перш за все, проблемою є забезпечення високої продуктивності та масштабованості системи. Зростання аудиторії гравців тягне за собою збільшення навантаження на сервери та бази даних, що без належної архітектури призводить до уповільнень, перебоїв у роботі та погіршення користувацького досвіду. Багато існуючих рішень базуються на застарілих монолітних архітектурах, які ускладнюють масштабування і технічне обслуговування.

Другим викликом є складність інтеграції різних функцій, від аутентифікації, управління контентом, відгуків, до адміністративних інструментів. Часто ці компоненти реалізуються окремо і не мають єдиного узгодженого інтерфейсу, що створює дискомфорт для користувачів і адміністрації. Відсутність централізованого контролю ускладнює підтримку та розвиток системи.

Ще одним важливим аспектом є безпека. Ігрові платформи приваблюють численних користувачів, у тому числі й потенційних зловмисників, які можуть здійснювати атаки на сервери, викрадати дані або порушувати роботу сервісів. Багато проєктів недостатньо уваги приділяють впровадженню сучасних механізмів захисту, таких як багатофакторна аутентифікація, шифрування даних, захист від DDoS-атак.

Також помітним недоліком багатьох систем є слабка підтримка мобільних пристроїв та адаптивність інтерфейсу. Враховуючи популярність мобільного геймінгу, цей фактор значно впливає на рівень залученості та задоволення користувачів.

Для вирішення цих проблем наш веб застосунок побудований на сучасній клієнт-серверній архітектурі, що дозволяє гнучко масштабувати систему, розподіляти навантаження і забезпечувати стабільну роботу. Впроваджено централізовану систему управління користувачами, відгуками, новинами, що підвищує зручність адміністрування та покращує користувацький досвід.

Безпека забезпечується використанням шифрування паролів, сесійної аутентифікації, а також планується інтеграція додаткових механізмів захисту. Інтерфейс розроблено з урахуванням адаптивності та UX-стандартів, що гарантує комфортну роботу як на десктопах, так і на мобільних пристроях.

Таким чином, описані проблеми враховані при розробці, що робить наше рішення сучасним, зручним та безпечним, готовим до подальшого розвитку та інтеграції нових функцій.

1.3 Аналіз аналогів програмного забезпечення

Аналіз існуючих аналогів є важливим етапом для визначення сильних і слабких сторін сучасних рішень, а також для пошуку натхнення і уникнення повторення помилок. У контексті розробки веб застосунку для ігрової платформи «Brick Prince» ми розглянули декілька популярних та успішних проєктів, що виконують схожі функції з управлінням контентом, користувачами, та інтерактивною взаємодією.

Першим і найближчим за функціоналом є офіційний сайт гри «Minecraft», який пропонує інтеграцію новин, форуму, облікових записів користувачів, а також можливості для донатів і адміністрування. Перевагами цього рішення є зручний інтерфейс, добре продумана система авторизації та підтримка великої кількості користувачів. Однак, складність коду і велика кількість функціоналу створюють виклики в адаптації та підтримці, особливо для менших команд розробників.

Другий приклад – «Steam Community», яка виконує роль платформи для взаємодії гравців, включно з відгуками, рейтингами, новинами та іншими соціальними функціями. Ця платформа демонструє високий рівень масштабованості і інтеграції з різними ігровими продуктами. З недоліків можна відзначити складність системи та вимогу потужних серверних ресурсів, що не завжди підходить для невеликих проєктів.

Ще одним популярним рішенням є офіційні веб портали інших інді-ігор, таких як «Hollow Knight» або «Celeste», які зазвичай мають спрощений дизайн, але реалізують основний набір функцій: новини, підтримка, відгуки та інтеграція з платіжними системами для донатів. Такі платформи є зразком ефективної простоти та зручності, проте часто не мають розширених адміністративних інструментів.

Наш веб застосунок, орієнтований на специфіку гри «Brick Prince», поєднує переваги зазначених рішень, одночасно зберігаючи простоту і доступність. Ми акцентуємо увагу на чіткій структурі, легкості масштабування та безпеці, зосереджуючись на тих функціях, які є критично важливими для цільової аудиторії. Унікальною особливістю є зручна адмін-панель, що дозволяє оперативно керувати користувачами, новинами та відгуками, а також гнучка система донатів з підтримкою різних сценаріїв взаємодії.

**2 ПОСТАНОВКА ЗАДАЧІ**

У межах розробки веб застосунку для гри "Brick Prince" виникла необхідність чітко окреслити задачі, які має реалізовувати програмне забезпечення. Враховуючи специфіку сучасної ігрової індустрії, де важливим фактором є не лише якість самого продукту, але й його супровід, комунікація з гравцями та прозорість взаємодії, постало завдання створити веб інтерфейс, який би об’єднував ці складові. Розробка мала враховувати потреби як потенційних користувачів гри, так і самої команди розробників, що повинні мати змогу керувати вмістом сайту, реагувати на відгуки та оновлювати контент.

Основною метою цієї роботи є створення сучасного, динамічного, локалізованого та масштабованого веб застосунку, який виступатиме як інформаційний та функціональний майданчик гри. Цей застосунок повинен забезпечити як інформування потенційних гравців про гру, так і забезпечити зворотний зв’язок, дозволити адмініструвати вміст та користувачів, а також організовувати взаємодію між різними ролями в системі.

Насамперед, потрібно було розробити логічну архітектуру, яка дозволяє розділити застосунок на незалежні модулі. Це включало проєктування взаємодії між клієнтською та серверною частинами, реалізацію API для обробки запитів та відповідей, а також побудову структури бази даних, яка б задовольняла потреби всіх функціональних компонентів[1].

Особлива увага приділялася реалізації системи реєстрації та автентифікації користувачів. Оскільки платформа має підтримувати різні типи користувачів (звичайний користувач, адміністратор, автор збору), необхідно було впровадити рольову модель, яка б дозволяла гнучко керувати правами доступу. Кожна роль отримує доступ лише до відповідного функціоналу, що зменшує ризики несанкціонованого втручання в роботу системи.

Крім того, необхідно було реалізувати функціонал публікації та перегляду відгуків користувачів. Це включає обробку реакцій на відгуки (схвалення або несхвалення), можливість редагування власного допису, а також видалення. Така взаємодія формує довіру користувачів до платформи та стимулює активність у межах спільноти.

Складовою системи також стала панель адміністратора, яка відкриває доступ до керування користувачами, відгуками, перевіркою авторів зборів, модерацією новин. Адміністратор має змогу шукати конкретного користувача, змінювати його статус верифікації, видаляти чи змінювати записи.

Ще однією суттєвою вимогою стала підтримка багатомовності. З огляду на потенційне розширення аудиторії гри на міжнародний рівень, веб застосунок одразу проєктувався з урахуванням локалізації. В інтерфейс були інтегровані механізми перемикання мови, а текстові елементи були винесені у файли перекладу, що значно спрощує подальше розширення мовної підтримки[2].

Не менш важливим етапом стало впровадження розсилки на електронні пошти. З метою інформування користувачів про важливі події, підтвердження дій або доставки матеріалів, було обрано інтеграцію з SendGrid[3]. Цей сервіс дозволяє реалізувати функціонал надсилання листів з підтвердженням або інформуванням, і вже на цьому етапі веб застосунок включає базову реалізацію відправки листа з посиланням на завантаження гри.

Останньою, проте важливою вимогою стало створення інтерфейсу, який відповідає сучасним стандартам UI/UX[4]. Це включало адаптивність дизайну, просту навігацію, логічне розташування елементів, а також приємний візуальний стиль.

Загалом, набір поставлених задач у проєкті був орієнтований як на функціональність, так і на зручність користувача, безпеку, розширюваність та подальше масштабування системи. Саме така постановка задач дозволила сформувати комплексне бачення архітектури та функціоналу, необхідного для успішного запуску веб платформи гри "Brick Prince".

**3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

3.1 UML проєктування ПЗ

На цьому етапі здійснюється логічне структурування функціональності веб застосунку для гри "Brick Prince". Основною метою є формалізація всіх випадків взаємодії користувача з системою, що дозволяє забезпечити чітке розуміння вимог до функціональності.

Однією з ключових UML-діаграм, яка буде використана, є діаграма прецедентів (див.рис.1.1). На ній буде відображено основні ролі (актори) у системі:

* Гість (неавторизований користувач);
* Зареєстрований користувач;
* Адміністратор.

Кожен з акторів має свій набір взаємодій з системою. Наприклад, гість може переглядати новини, відгуки, кампанії збору коштів та проходити реєстрацію. Зареєстрований користувач має змогу залишити відгук, редагувати профіль. Адміністратор, у свою чергу, має доступ до панелі управління користувачами та кампаніями, імпорту/експорту даних та управління контентом.



Рисунок 3.1 – Use Case Diagram

Додатково, доцільно побудувати діаграми активностей, що відображають логіку сценаріїв: процес реєстрації користувача, створення відгуку, адмін панель. Ці діаграми дають чітке уявлення про етапи обробки кожної з дій у системі.

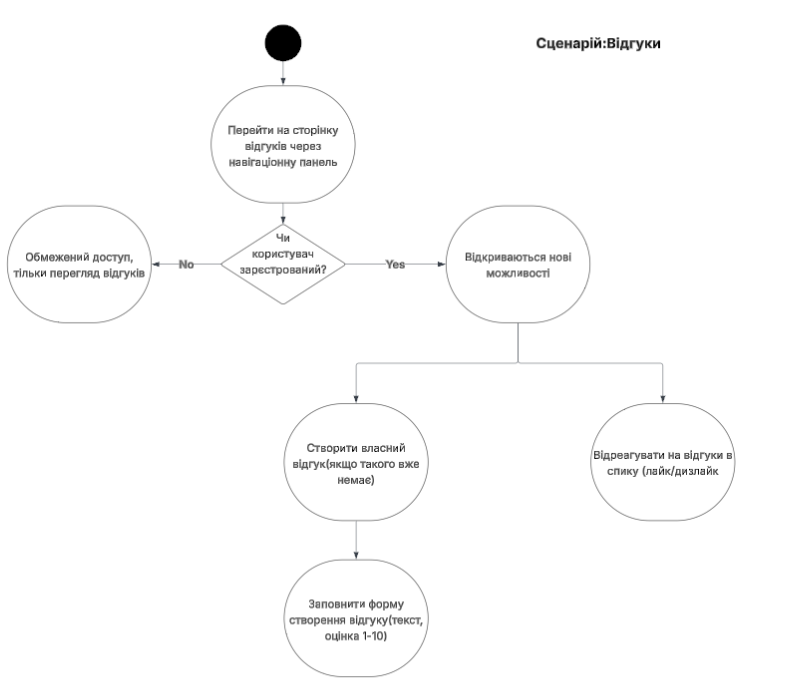


Рисунок 3.2 – Діаграма активності, сценарій відгук

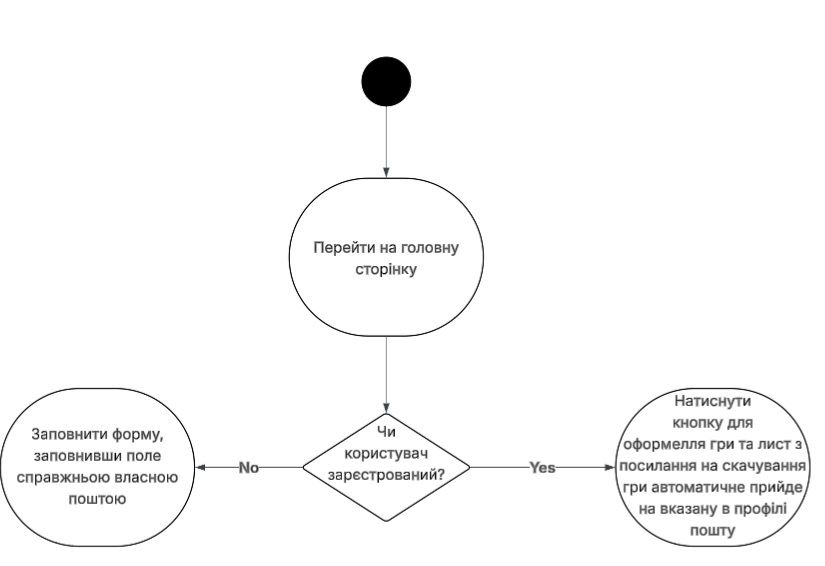


Рисунок 3.3 – Діаграма активності, сценарій отримання гри

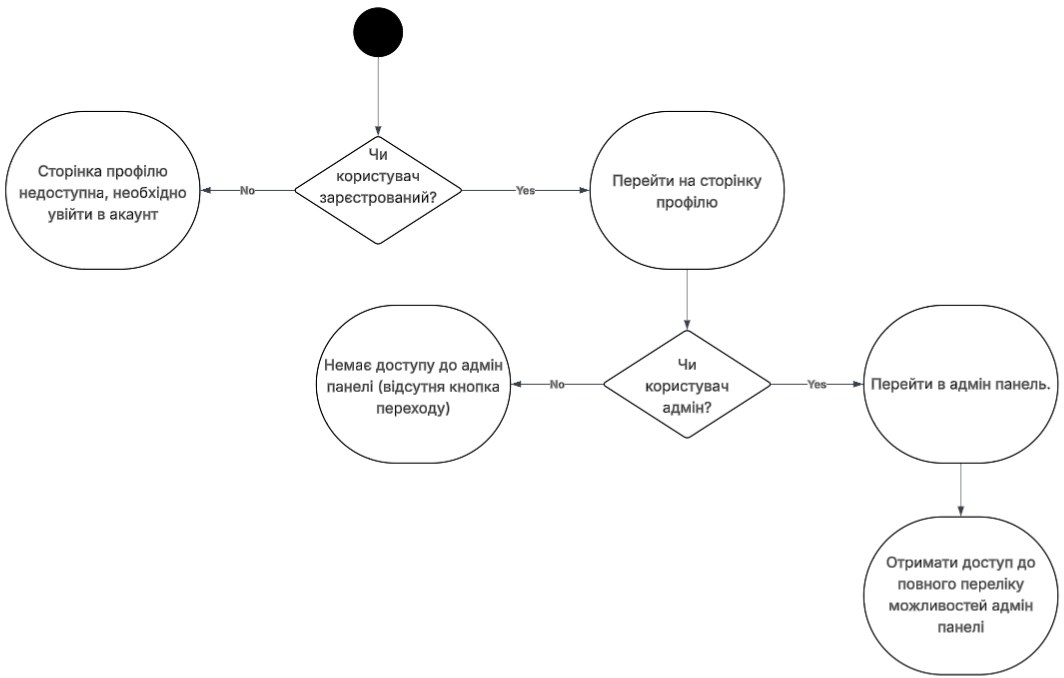


Рисунок 3.4 – Діаграма активності, сценарій – адмін панель

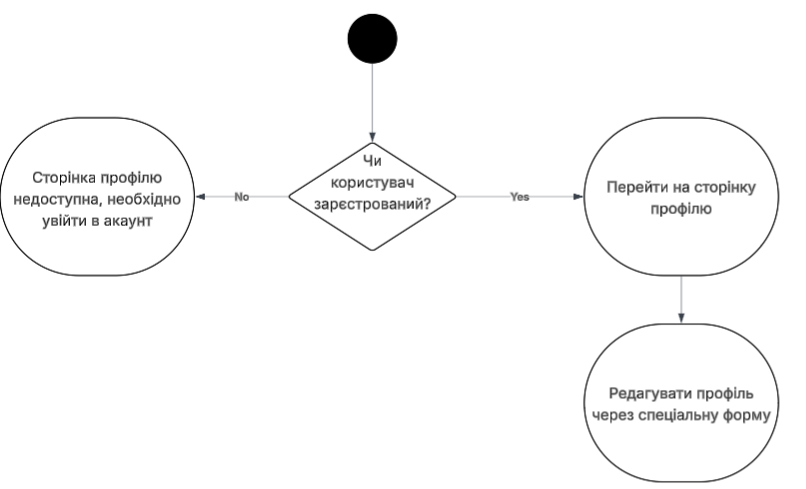


Рисунок 3.5 – Діаграма активності, сценарій – редагування профілю

3.2 Проєктування архітектури програмного забезпечення

Архітектура програмного забезпечення нашого веб застосунку відіграє ключову роль у забезпеченні його надійності, масштабованості та зручності в підтримці. У цьому підрозділі детально розглядаються всі основні складові архітектурного рішення, які були прийняті під час розробки системи, з урахуванням вимог до функціоналу, безпеки, продуктивності та зручності розгортання.

Передусім, слід наголосити на тому, що обраною архітектурною моделлю для даного веб застосунку стала класична клієнт-серверна модель. Така модель дозволяє чітко відокремити інтерфейсну частину від серверної логіки, що істотно спрощує підтримку, модернізацію та масштабування системи. Зокрема, клієнтська частина, реалізована з використанням сучасних веб технологій, взаємодіє з сервером через стандартизовані REST API-запити. Сервер, у свою чергу, обробляє запити, звертається до бази даних, обробляє бізнес-логіку та надсилає відповідь назад клієнту у форматі JSON.

Функціонально система поділена на кілька ключових компонентів:

* Інтерфейс користувача (frontend), що дозволяє авторизованим користувачам переглядати інформацію, залишати відгуки, керувати профілем, а адміністраторам — виконувати модерацію та управління контентом;
* Серверна логіка (backend), яка відповідає за обробку запитів, реалізацію контролерів, аутентифікацію, авторизацію, валідацію даних та логіку зберігання;
* База даних, яка забезпечує надійне зберігання структурованої інформації про користувачів, відгуки, збори, новини тощо.

Між компонентами встановлено чітку комунікацію через протокол HTTPS. Основною формою взаємодії між клієнтом і сервером є REST-запити, що передаються у форматі JSON. Завдяки такому підходу система залишається відкритою до інтеграції з іншими додатками та сервісами в майбутньому.

На діаграмі компонентів(див.рис.3.6) представлені основні модулі системи. Цей рівень абстракції дає змогу легко розширювати проєкт у майбутньому, наприклад, додавати нові типи зборів або інші види користувачів без масштабної переробки основної логіки.

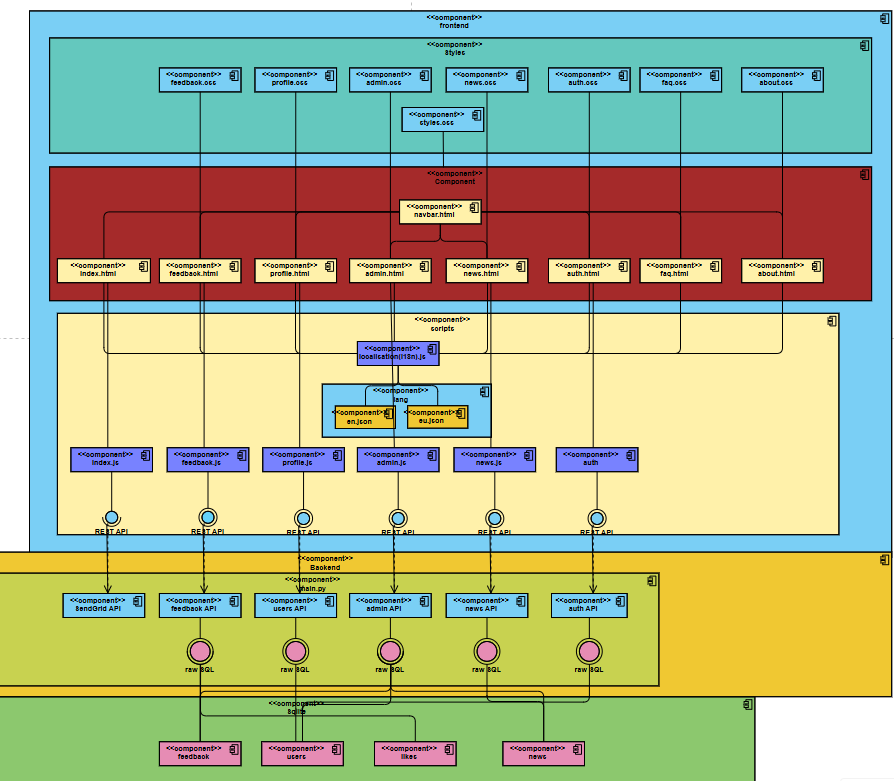


Рисунок 3.6 - UML-діаграма компонентів

Компоненти розгортаються на окремих вузлах у залежності від сценарію використання. У локальній розробці всі компоненти розгортаються на одному пристрої, однак у майбутньому передбачено розгортання на віртуальному сервері, де frontend буде обслуговуватися окремим веб сервером (наприклад, Nginx), а backend — додатком на FastAPI, який запускається через Gunicorn або Uvicorn. База даних, у свою чергу, може бути підключена окремо як локально, так і через хмарні сервіси типу SQLite Remote або PostgreSQL-as-a-Service

На діаграмі розгортання(див.рис.3.7) візуалізовані такі вузли: клієнтський пристрій користувача (браузер), вебсервер (frontend), сервер додатку (FastAPI), база даних SQLite[5] та зовнішні сервіси, зокрема сторонні API (наприклад, поштові служби для верифікації або надсилання повідомлень).

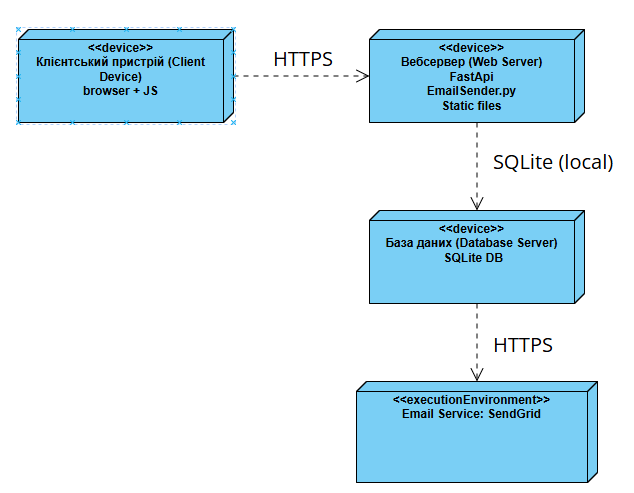


Рисунок 3.7 - UML-діаграма розгортання

Важливою складовою архітектурного рішення є нефункціональні вимоги. Система має забезпечувати стабільну роботу навіть при значній кількості одночасних запитів, тому реалізовано кешування, відкладену обробку деяких запитів, а також структуровану обробку помилок. Усі критичні дії логуються, що дає змогу швидко виявляти проблеми. Надійність також забезпечується через застосування авторизаційних токенів, перевірку сесій, валідацію вхідних даних як на клієнті, так і на сервері.

Щодо ризиків, які можуть виникнути в майбутньому, варто відзначити можливі проблеми з масштабуванням у випадку збільшення навантаження, а також необхідність переходу на більш потужну базу даних або сервер при зростанні кількості користувачів. Щоб мінімізувати ці ризики, архітектура спроєктована таким чином, що заміна окремих компонентів (наприклад, SQLite на PostgreSQL) не потребуватиме суттєвого переписування коду. Безпека даних гарантується використанням HTTPS, захищеного зберігання паролів (через хешування), регулярними оновленнями залежностей та перевіркою запитів на наявність шкідливих ін’єкцій.

У підсумку, розроблена архітектура забезпечує гнучкість, надійність та масштабованість системи, що дозволяє з упевненістю планувати її подальший розвиток. Сформовані діаграми компонентів і розгортання надають візуальне уявлення про структуру системи та взаємозв’язки між її частинами, а їх відповідність описаним вимогам гарантує ефективне впровадження на практиці.

3.3 Проєктування структури зберігання даних

У процесі розробки веб застосунку особливу увагу було приділено організації надійного, масштабованого та логічно обґрунтованого сховища даних, яке забезпечуватиме швидкий доступ до інформації, підтримку взаємозв’язків між сутностями та цілісність даних навіть за умов високих навантажень. Для досягнення цього було обрано реляційну модель бази даних, реалізовану на основі системи керування базами даних SQLite, що добре підходить для локального і середнього рівня навантаження, особливо під час активної фази розробки.

Розробка структури зберігання даних розпочалася з детального аналізу предметної області сайту, який супроводжує ігровий проєкт Brick Prince. Основними об’єктами даних виступають користувачі, профілі, новини про гру, відгуки, голосування за контент, а також адміністративна аналітика. Всі ці об’єкти мають власні унікальні характеристики, але між ними існують чіткі логічні зв’язки, які й було покладено в основу побудови бази даних.

Загалом, під час проєктування було визначено низку сутностей, які описують ключові елементи взаємодії користувача з веб платформою. Наприклад, користувач має профіль, може залишати відгуки, оцінювати їх за допомогою лайків/дизлайків, а також взаємодіяти з новинами, створеними через адмінпанель, користувачем який має роль адміністратора. Самі ж новини можуть бути прив’язані до певних ігрових подій або етапів розробки.

На етапі логічного моделювання для точного уявлення про структуру було розроблено ER-діаграму, яка дозволяє візуально уявити зв’язки між сутностями.

Наприклад, сутність User має первинний ключ id, що пов’язується із зовнішніми ключами у таблицях Feedback, Likes, News, забезпечуючи логічну цілісність даних. Аналогічно, сутність Feedback прив’язана до User, а також пов’язана з таблицею Likes, яка описує реакцію інших користувачів.



Рисунок 3.8 – ER diagram

1. Сутність «Користувач»
   1. Атрибути:
      * #Користувач\_id;
      * Ім’я;
      * Пошта;
      * Роль (користувач або адмін);
      * Дата створення.
   2. Зв'язки:
      * Один користувач може мати лише один відгук. (один до одного;
      * Один користувач може мати багато лайків (один до багатьох);
      * Один користувач(адмін) може створити багато новин (один до багатьох).
2. Сутність «Новина»
   1. Атрибути:
      * #Новина\_id;
      * Заголовок;
      * Контент;
      * Дата створення.
   2. Зв'язки:
      * Одна новина може мати тільки одного автора (багато до одного).
3. Сутність «Відгук»
   1. Атрибути:
      * #Відгук\_id;
      * Текст;
      * Оцінка;
      * Дата створення.
   2. Зв'язки:
      * Один відгук може бути створений лише одним користувачем (один до одного);
      * Один відгук може мати лише один лайк від користувача.
4. Сутність «Лайк»
   1. Атрибути:
      * #Лайк\_id;
      * Лайк чи дізлайк.
   2. Зв'язки:
      * Користувач може поставити лише 1 лайк на будь-який відгук. (багато до одного);
      * Тільки один лайк від користувача може бути на одному конкретному відгуку.

Після погодження логічної моделі було здійснено побудову фізичної схеми реляційної бази даних, яка вже враховує конкретні атрибути, типи даних, обмеження, первинні та зовнішні ключі. Таблиці структурувалися таким чином, щоб зберегти цілісність при високій нормалізації та забезпечити оптимальність запитів.

Наприклад, таблиця Users містить поля id, email, password, username, is\_verified, що дозволяють реалізувати як звичайну автентифікацію, так і рольову модель керування правами.

Таблиця Feedback має поля id, user\_id, rating, text, created\_at, що дозволяють зберігати текстові рецензії з часовими мітками та пов’язувати їх з конкретними користувачами.

У таблиці Likes врахована складна логіка — користувач може поставити лише одну реакцію до кожного відгуку. Для цього було впроваджено композитний унікальний індекс (user\_id, feedback\_id), який не дозволяє дублювання записів, тим самим забезпечуючи логічну цілісність реакцій.

Кардинальність зв’язків між таблицями вказується прямо на схемі, наприклад, один користувач може мати 1 відгук (1:1), але кожен відгук належить лише одному користувачу. Зв’язок між відгуками та реакціями є зв’язком типу один до багатьох, з обмеженням на унікальність пар user\_id + feedback\_id.

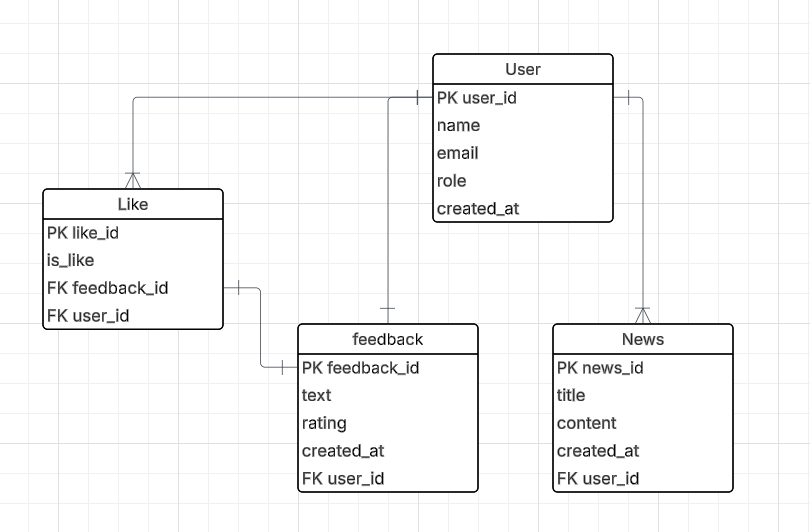


Рисунок 3.8 – Схема Бд

1. Таблиця User

* Кожен користувач має унікальний id, який слугує первинним ключем;
* Кожен атрибут залежить лише від цього ключа, тому транзитивних залежностей немає.

Функціональні залежності:

name, email, password, is\_admin, is\_verified ← #id

2. Таблиця Feedback

* Кожен відгук має унікальний id, що є первинним ключем. Один користувач може залишити один відгук.

Функціональні залежності:

user\_id, rating, content, created\_at ← #id

3. Таблиця News

* Кожна новина має унікальний id. Один користувач (автор) може створити декілька новин.

Функціональні залежності:

title, content, created\_at, author\_id ← #id

4. Таблиця Like

* Кожна реакція має унікальний id. Один користувач може оцінити один відгук тільки один раз.

Функціональні залежності:

user\_id, feedback\_id, value ← #id

У контексті даної системи було прийнято рішення використовувати SQLite це легка, вбудована реляційна СКБД, яка відмінно підходить для проєктів, де акцент робиться на локальну розробку, тестування або невелике навантаження. У разі подальшого розширення платформи структура бази дозволяє з мінімальними змінами перенести дані до PostgreSQL або MySQL, що забезпечить масштабованість у продакшн-середовищі.

3.4 Приклади використаних алгоритмів та методів

У процесі розробки веб застосунку для гри Brick Prince важливо було не лише реалізувати інтерфейсну складову, але й забезпечити чітку внутрішню логіку функціонування окремих модулів, яка впливає на зручність взаємодії користувача із системою. У цьому підрозділі буде розглянуто алгоритмічні та методологічні рішення, що були використані для реалізації ключових сценаріїв роботи системи, з акцентом на ефективність, послідовність дій, перевірку коректності даних та підтримку зручної логіки взаємодії між модулями.

Одним із яскравих прикладів застосування алгоритмічного мислення стала реалізація системи реакцій на відгуки, яка дозволяє користувачам оцінювати відгуки інших учасників через лайки або дизлайки. Завдання полягало не лише у виведенні кількості оцінок, а й у гарантуванні, що кожен користувач може залишити лише одну реакцію на конкретний відгук. У разі повторного натискання на вже активну кнопку — реакція має зникати, а при зміні лайка на дизлайк (або навпаки) — стара реакція повинна замінюватися новою. Така логіка вимагає не просто оновлення інтерфейсу, а й точного обліку у базі даних, який реалізовано через умовне оновлення записів та перевірку існування попередніх реакцій.

Цей процес реалізовано з урахуванням транзакційності запитів до бази даних, що виключає можливість виникнення конфліктів у разі одночасної взаємодії кількох користувачів.

Окремо слід згадати алгоритм фільтрації власного відгуку користувача. При завантаженні сторінки з відгуками система виділяє відгук, залишений поточним користувачем, і виводить його на окремому фоні з додатковими кнопками «Редагувати» та «Видалити». Це досягається через наявність порівняння ID поточного користувача з автором кожного з відгуків. Така логіка є простою на перший погляд, однак при великій кількості записів і складній пагінації вона вимагає ефективної обробки масиву відгуків, що реалізується через фільтрацію на бекенді або client-side умовне відображення.

У модулі створення новин для адміністратора застосовано механізм перевірки обов’язкових полів перед відправкою запиту. Тут реалізовано класичний алгоритм валідації форм, коли перед надсиланням даних до API перевіряється, що жодне з полів не є порожнім, заголовок не перевищує заданої кількості символів, а тіло новини містить мінімальний обсяг контенту. У разі невідповідності користувач отримує зрозуміле повідомлення про помилку.

Ще одним прикладом внутрішньої логіки є робота з пагінацією або поступовим підвантаженням відгуків. Тут система спочатку завантажує лише частину даних, і при натисканні кнопки "Завантажити ще" відправляє наступний запит з offset або page-параметром. Це дозволяє уникнути перевантаження інтерфейсу, економить ресурси клієнта та сервера й робить взаємодію з контентом більш плавною. Алгоритм простий, однак передбачає правильну реалізацію кешування, щоб при оновленні сторінки не дублювались уже відображені записи.

Загалом, навіть у невеликих задачах, таких як логіка зміни мови інтерфейсу, застосовується системний підхід. При перемиканні локалі оновлюються всі текстові компоненти, при цьому використовуються ключі для перекладу, які зберігаються в окремих словниках. Це дозволяє швидко масштабувати підтримку нових мов без переписування коду.

Ще одним важливим алгоритмом, реалізованим у системі, є відправка листа на електронну пошту користувача з посиланням на гру Brick Prince. Ця логіка має два сценарії використання: перший — для авторизованих користувачів, другий — для гостей, які вказують свою пошту у відповідній формі у футері. У обох випадках необхідно провести валідацію email-адреси, сформувати HTTP-запит до бекенд-сервісу, де за допомогою стороннього API (SendGrid) відбувається розсилка повідомлень. Система забезпечує зворотній зв’язок: після надсилання користувач отримує повідомлення про успішну доставку або про помилку, що дозволяє уникнути невизначеності та покращує досвід користування платформою.

Незважаючи на простоту інтерфейсу, ця логіка охоплює декілька етапів — перевірку стану користувача (залогінений/гість), валідацію email (на фронті), формування тіла POST-запиту та обробку відповіді від серверної частини. На бекенді ж реалізується перевірка формату email, обробка винятків, пов’язаних із мережею, а також правильна робота з ключем API. Сам алгоритм було побудовано з урахуванням повторного використання компонентів: функція відправки використовується у декількох точках застосунку.

Також варто згадати логіку керування ролями користувачів у системі. У проєкті реалізовано базовий механізм розмежування прав доступу, при якому користувачі поділяються на звичайних, верифікованих та адміністраторів. Відповідні інтерфейсні компоненти, а також API-запити, доступні лише після перевірки ролі користувача. Наприклад, доступ до адміністративної панелі, до функції редагування новин або модерації користувачів обмежений лише для тих, у кого встановлено прапорець role = admin. Цей функціонал реалізовано не тільки на рівні фронтенду (відображення інтерфейсу), а й на рівні бекенду, де при кожному запиті відбувається перевірка прав користувача. Така подвійна перевірка підвищує безпеку застосунку та виключає несанкціонований доступ через підміну запитів.

Крім того, взаємодія з профілем користувача також вимагає певної логіки: при завантаженні сторінки профілю здійснюється запит до API для отримання даних про поточного користувача. У формі редагування автоматично заповнюються поля поточними значеннями, що дає змогу користувачеві швидко внести зміни. Валідація нових даних, перевірка їх коректності та уникнення дублювання email (якщо змінюється поштова адреса) — усе це реалізовано з урахуванням UX-практик.

Загалом, логіка додатка, хоча й не передбачає складних математичних алгоритмів, базується на численних умовних перевірках, сценаріях обробки подій, перевірці ролей та синхронізації стану даних між клієнтською та серверною частинами.

Таким чином, у рамках створення веб застосунку були реалізовані численні алгоритми та логічні блоки, які забезпечують коректну, надійну та інтуїтивно зрозумілу взаємодію користувача з системою. Вони охоплюють як дії з інтерфейсом, так і поведінку бекенду, зокрема перевірку прав доступу, обробку взаємодій, підтримку унікальності дій та збереження консистентності даних. Навіть незначні на перший погляд механізми, такі як реакції на відгуки, несуть у собі важливу логіку, що підвищує якість використання платформи.

3.5 Проєктування UI/UX або іншого дизайну системи

Візуальна складова веб застосунку "Brick Prince" відіграє ключову роль у створенні привабливого та інтуїтивно зрозумілого інтерфейсу, який забезпечує комфортну навігацію, швидке сприйняття інформації та емоційне залучення користувача. Дизайн нашої системи ґрунтується на принципах доступності, адаптивності та мінімалізму, що дозволяє зосередити увагу відвідувача на основному контенті без зайвого перевантаження візуальними елементами[6].

Перш за все, веб інтерфейс було побудовано з урахуванням логічної структури розділів та взаємозв'язків між ними. Всі основні сторінки (головна, новини, відгуки, FAQ, про проєкт, профіль) доступні через верхню навігаційну панель. Такий підхід забезпечує користувачу відчуття контролю над навігацією та дозволяє швидко орієнтуватися в системі. Навігація адаптована під мобільні пристрої завдяки використанню гнучкої верстки (Flexbox та Grid), що забезпечує плавне масштабування контенту на різних екранах[7].

Кольорова палітра базується на темно-синьому фоні з контрастними яскравими акцентами (помаранчеві кнопки дій, білий текст), що формує виразний стиль і підкреслює геймерську тематику проєкту. Кнопки мають м’яке заокруглення, тінь та зміну стану при наведенні, що покращує зворотний зв'язок із користувачем. Типографіка використовує шрифт Poppins із різною товщиною гарнітур для створення ієрархії контенту[8].

Особлива увага приділялася головній сторінці, оскільки вона формує перше враження про платформу. На головному екрані представлено слоган гри, заклик до дії (оформити гру) та стилізоване зображення, яке одразу вказує на тематику середньовічного світу, побудованого з цегли. Кнопка "Оформити гру" динамічно реагує на авторизаційний стан користувача: у разі, якщо він не авторизований, система скролить до форми в футері; в іншому випадку — надсилає листа.

Сторінка "Профіль" реалізована як особистий кабінет користувача, де він може переглянути свої дані, редагувати ім’я або електронну пошту, а також у майбутньому — бачити активність у системі. Важливою деталлю є реалізація можливості редагування профілю без перезавантаження сторінки, що досягається використанням сучасних підходів JavaScript.

На сторінці "Відгуки" реалізовано механізм відображення усіх залишених користувачами відгуків із рейтингом, текстом, датою та можливістю залишати лайки/дизлайки. Власний відгук автоматично виділяється візуально, а поруч з ним з'являються кнопки "Редагувати" та "Видалити". Кнопка "Створити відгук" доступна лише авторизованим користувачам, і при її натисканні відкривається модальне вікно з формою, що дозволяє зручно додати нову думку без залишення сторінки.

Сторінка "Новини" має лаконічне карткове представлення. Кожна новина включає заголовок, дату публікації та короткий зміст. Структура карток та використання білих блоків на темному фоні створюють візуальну чистоту та полегшують сприйняття тексту.

Інтерфейс адміністратора реалізовано в рамках окремого маршруту та адаптовано до потреб керування користувачами та новинами. Було передбачено пошук за email, зміну статусу верифікації, видалення користувачів із підтвердженням дії, а також додавання, редагування та видалення новин. Інтерфейс дозволяє адміністратору зручно виконувати свої завдання без необхідності використовувати зовнішні інструменти.

Інтернаціоналізація інтерфейсу (UA/EN) реалізована за допомогою бібліотеки i18next. Всі текстові елементи переведені у формат ключів, і зміна мови виконується миттєво без перезавантаження сторінки. Це дозволяє забезпечити зручність для широкого кола користувачів.

Узагальнюючи, UI/UX дизайну в розробленому веб застосунку приділено особливу увагу. Кожен елемент ретельно спроєктовано для досягнення максимальної зручності, доступності, стилістичної єдності та приємного користувацького досвіду. Завдяки цьому сайт не лише функціонально виконує свої задачі, а й виглядає сучасно, привабливо та інтуїтивно зрозуміло.

**4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ**

4.1 Середовище розробки та особливості запуску

Розробка веб застосунку для гри Brick Prince здійснювалася на локальному комп’ютері із використанням Visual Studio Code[9] як основного інструменту для програмування. Це середовище було обрано через свою легкість, гнучкість, велику екосистему плагінів, зручну інтеграцію з терміналом, git-репозиторіями та чудову підтримку для мов програмування, що використовувалися у проєкті: JavaScript для фронтенду та Python (FastAPI) для серверної частини.

У якості локального веб-сервера під час розробки інтерфейсної частини застосовувалося розширення Live Server, що дозволяло миттєво переглядати результати змін без необхідності вручну оновлювати сторінку або перезапускати сервер. Це значно пришвидшувало цикл розробки та тестування зовнішнього вигляду сайту, забезпечуючи ефективну перевірку взаємодії між компонентами.

Для створення серверної логіки було обрано FastAPI — сучасний асинхронний Python-фреймворк, що надає інтуїтивний синтаксис для побудови RESTful API. Серверна частина запускалася локально за допомогою команди uvicorn main:app --reload, що дозволяла автоматично оновлювати сервер при внесенні змін до коду.

У якості бази даних використовувалася SQLite — вбудована реляційна СУБД, що не потребує окремого серверного розгортання. Вона була обрана через простоту, відсутність необхідності у складному налаштуванні та чудову інтеграцію з Python через бібліотеку sqlite3[10]. Це забезпечило можливість миттєво створювати й модифікувати базу даних локально без додаткового навантаження на систему. У майбутньому система може бути легко адаптована до масштабованої СУБД, такої як PostgreSQL, але на етапі прототипу SQLite була оптимальним вибором.

Також у серверній частині використовувався модуль smtplib для реалізації надсилання листів на електронну пошту, що стало корисним при додаванні можливості зворотного зв’язку. Запуск відповідного функціоналу вимагав наявності інтернет-з’єднання, коректного API-ключа (наприклад, через сервіс SendGrid) та налаштованої автентифікації відправника.

Характеристики пристрою, на якому виконувалась розробка та тестування вебзастосунку:

* Процесор: 11th Gen Intel® Core™ i5-1135G7, 2.40GHz;
* Оперативна пам’ять: 8 GB;
* Операційна система: Windows 11 Home x64.

Програмне забезпечення:

* + Visual Studio Code (розширення Python, Live Server, ESLint, SQLite Viewer, REST Client);
  + Python 3.11;
  + SQLite;
  + Git;
  + Google Chrome (для перевірки фронтенду);
  + FastAPI + Uvicorn (локальний сервер).

Таке середовище дозволить забезпечити повноцінну паралельну розробку як клієнтської, так і серверної частини, зберігаючи зручність розгортання, швидкість тестування та високу адаптивність до змін. Завдяки легкості запуску кожного з компонентів розробка проходила без потреби у складному CI/CD або віртуальному середовищі. Однак структура проєкту вже спроєктована так, щоб за необхідності її можна було легко перенести на віддалений сервер із підтримкою Docker або подальшим CI/CD у хмарі.

4.2 Описання фізичної моделі бази даних

Був створений окремий файл для ініціації БД(див. додаток А)

Таблиця users зберігає основну інформацію про зареєстрованих користувачів вебзастосунку. Вона має первинний ключ id з автоінкрементом, який унікально ідентифікує кожного користувача. Атрибути name, email та password є обов’язковими для заповнення, при цьому email має унікальне значення для уникнення дублювання облікових записів. Атрибут role обмежується значеннями 'user' або 'admin', що дозволяє диференціювати звичайних користувачів та адміністраторів. Поле created\_at автоматично фіксує дату та час реєстрації нового користувача.

Таблиця news відповідає за збереження новин, що публікуються адміністраторами сайту. Вона має первинний ключ id, а також поля title та content, які містять відповідно заголовок та основний текст новини. Час створення новини автоматично записується у полі created\_at. Для встановлення авторства використовується зовнішній ключ author\_id, що посилається на таблицю users, забезпечуючи зв’язок між новиною та конкретним користувачем-автором. Це дозволяє при необхідності реалізовувати фільтрацію або виведення новин конкретного адміністратора.

Таблиця feedback зберігає відгуки користувачів про гру. Кожен запис має унікальний id як первинний ключ. Через атрибут user\_id встановлюється зовнішній зв’язок з таблицею users, що забезпечує можливість ідентифікувати автора відгуку. Кожен користувач може залишити лише один відгук, що забезпечується обмеженням UNIQUE(user\_id). Поле text містить основний зміст відгуку, а rating обмежений значеннями від 1 до 10, що дозволяє кількісно оцінювати враження від гри. Атрибут created\_at автоматично фіксує момент публікації.

Таблиця likes реалізує функціонал реакцій на відгуки, дозволяючи користувачам ставити лайки або дизлайки. Первинний ключ id створюється автоматично. Поля user\_id та feedback\_id встановлюють відповідні зовнішні зв’язки до таблиць users та feedback. Важливою особливістю є обмеження UNIQUE(user\_id, feedback\_id), яке гарантує, що кожен користувач може оцінити конкретний відгук лише один раз. Поле is\_like є логічним типом і вказує, чи є реакція позитивною (True) або негативною (False).

4.3 Опис програмної реалізації

Після відкриття головної сторінки веб застосунку Brick Prince користувач потрапляє на інтерфейс, що поєднує сучасний дизайн, зручну навігацію та прямий доступ до основного контенту. Незареєстрованому відвідувачу одразу стає доступною інформація про гру, можливість переглянути блог, а також — отримати саму гру, ввівши свою електронну пошту у відповідне поле. Цей механізм дозволяє поширювати дистрибутив гри максимально зручно, без необхідності реєстрації чи зайвих дій з боку користувача (див. рис. 4.1).

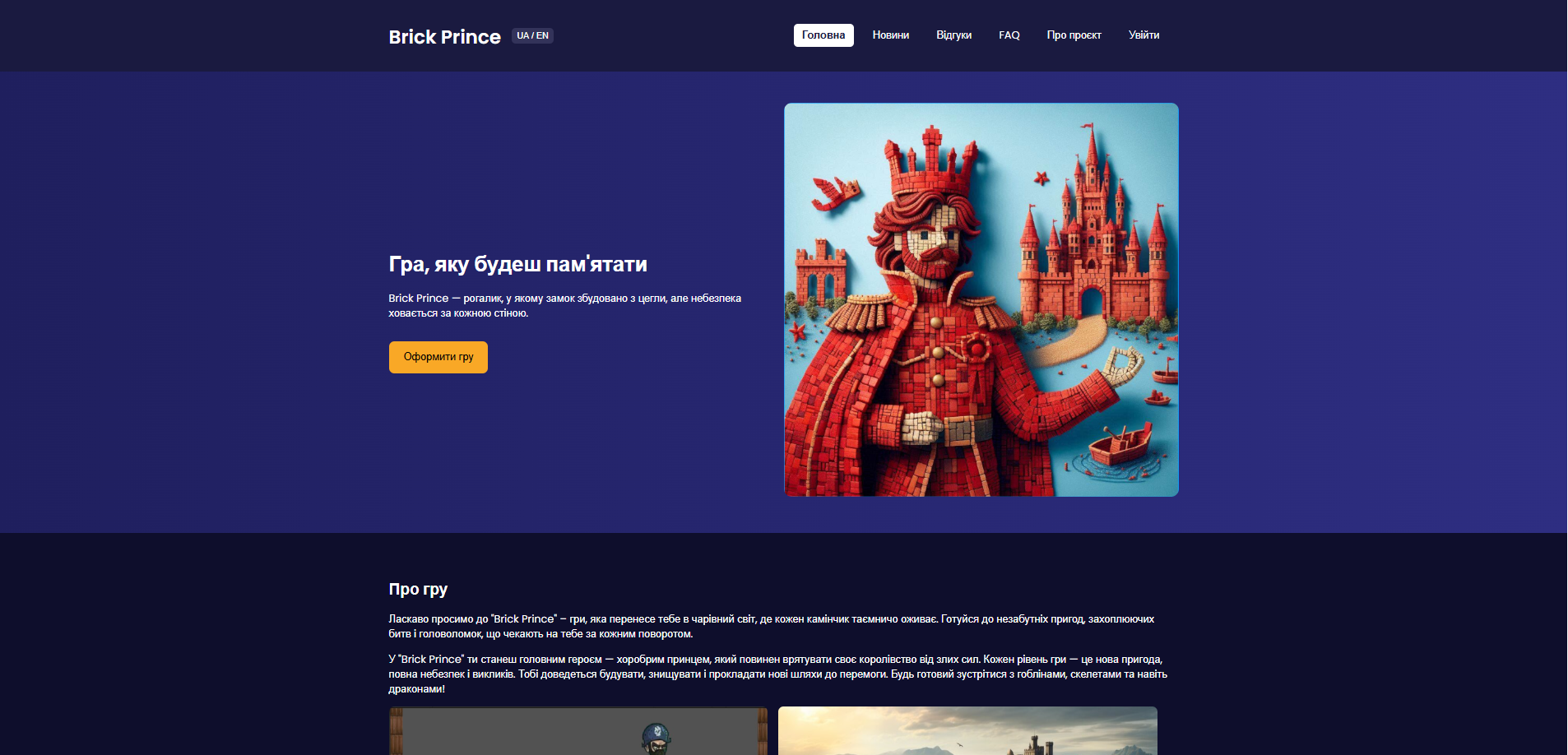


Рисунок 4.1 – Головна сторінка

Незареєстрований користувач при натисканні головної кнопки «Оформити гру» буде примусово перенесений до футеру на головній сторінці.

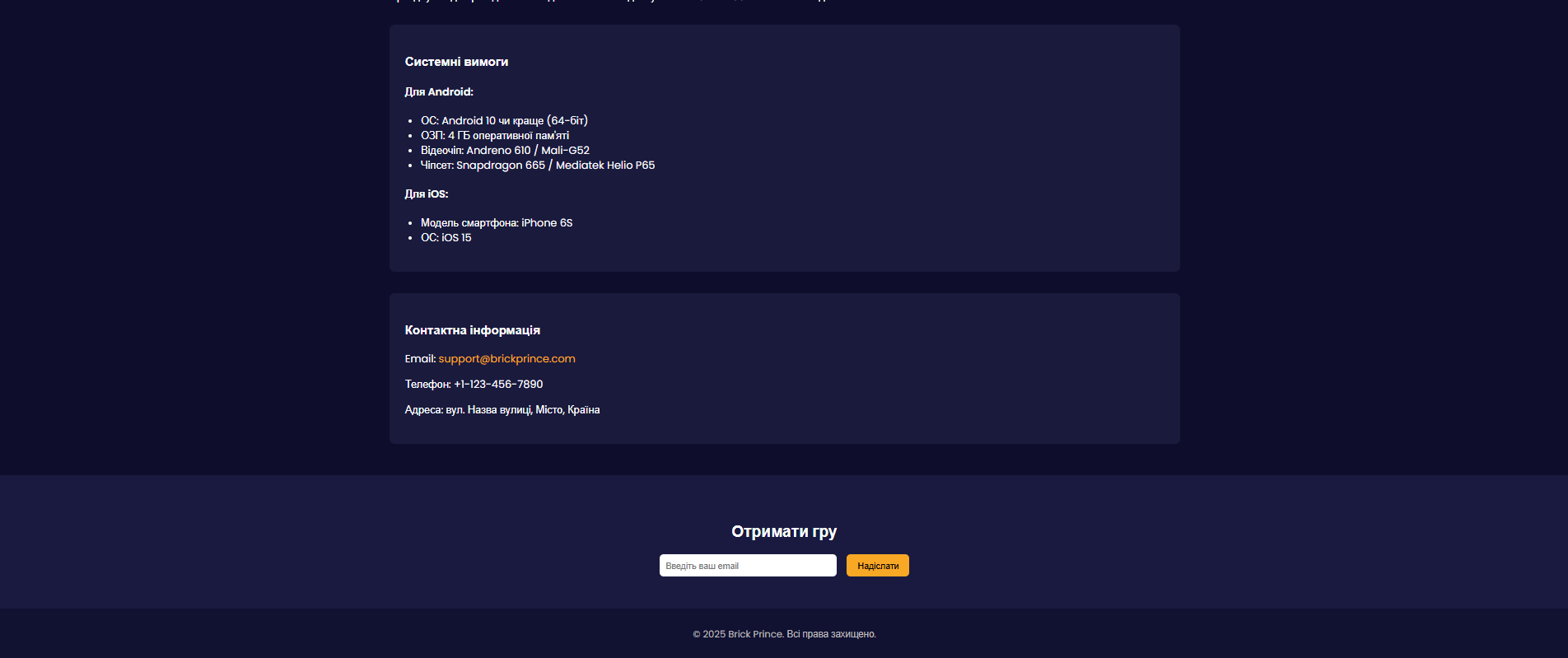


Рисунок 4.2 – Футер

Після введення email та натискання кнопки підтвердження, веб застосунок активує серверний механізм, який ініціює перевірку правильності формату введеної адреси, а далі — викликає функцію для відправки листа з прикріпленим файлом гри.

Відразу варто зазначити що якщо користувач залогінений то футтер буде прихований та отримати гру можна буде за основною кнопкою “Оформити гру”

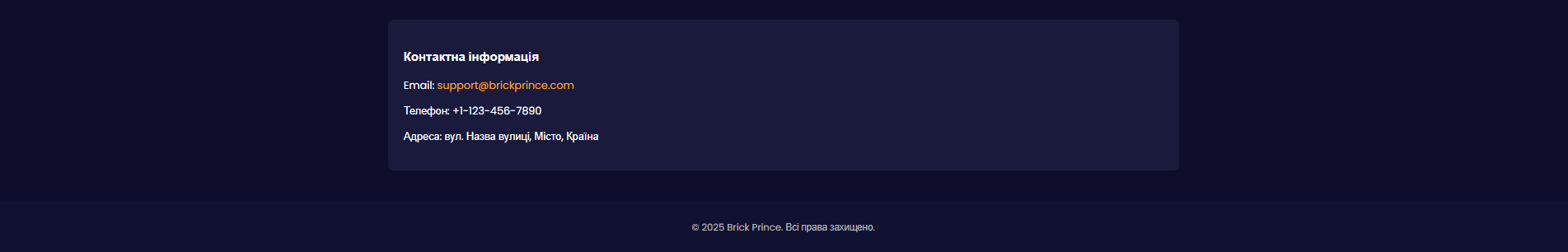


Рисунок 4.2 – Футер для залогіненого користувача

Функціонал відправки листів на пошту реалізовано за допомогою сервісу SendGrid, який забезпечує надійну та швидку доставку електронних листів. У бекенді було реалізовано обробник, що приймає POST-запит, у якому міститься email користувача. Відправка листа відбувається лише у випадку, якщо адреса є валідною.

Скрипт JavaScript:  
Якщо користувач залогінений — сховати форму у футері:

**if (user) {**

**if (formSection) formSection.style.display = "none";**

**}**

Обробка кнопки "Оформити гру" у верхній секції:

**if (orderBtn) {**

**orderBtn.addEventListener("click", (e) => {**

**e.preventDefault();**

**if (!user) {**

**if (formSection) {**

**formSection.scrollIntoView({ behavior: "smooth" });**

**emailInput.focus();**

**}**

**} else {**

**sendGameEmail(user.email);**

**}**

**});**

**}**

Обробка відправки форми футера:

**if (form) {**

**form.addEventListener("submit", (e) => {**

**e.preventDefault();**

**const email = emailInput.value.trim();**

**if (!validateEmail(email)) {**

**alert("Будь ласка, введіть коректний email");**

**emailInput.focus();**

**return;**

**}**

**sendGameEmail(email);**

**});**

**}**

Функція відправки email на бекенд:

**async function sendGameEmail(email) {**

**try {**

**const res = await fetch("http://127.0.0.1:8000/send-game-email", {**

**method: "POST",**

**headers: { "Content-Type": "application/json" },**

**body: JSON.stringify({ email }),**

**credentials: "include",**

**});**

**if (!res.ok) {**

**const err = await res.json();**

**throw new Error(err.error || "Помилка при відправці листа");**

**}**

**alert(`Лист з грою успішно надіслано на: ${email}. Перевірте вашу пошту!`);**

**if (!user && formSection) {**

**emailInput.value = "";**

**}**

**} catch (err) {**

**alert(err.message);**

**}**

**}**

Перевірка email:

**function validateEmail(email) {**

**return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);**

**}**

**});**

API запит на бекенді:  
**ssl.\_create\_default\_https\_context = lambda: ssl.create\_default\_context(cafile=certifi.where())**

**def send\_game\_email(to\_email: str):**

**message = Mail(**

**from\_email="ihor.balytskyi1@nure.ua",**

**to\_emails=to\_email,**

**subject="Ваш Brick Prince — гра готова!",**

**html\_content="""**

**<h1>Дякуємо, що оформили гру Brick Prince!</h1>**

**<p>Ось посилання для завантаження гри:</p>**

**<a href="https://\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*">Завантажити гру</a>**

**""",**

**)**

**try:**

**sg = SendGridAPIClient(os.getenv("SENDGRID\_API\_KEY"))**

**response = sg.send(message)**

**print(f"SendGrid response status: {response.status\_code}")**

**return True**

**except Exception as e:**

**print(f"SendGrid error: {e}")**

**return False**

Незареєстрованому користувачу також доступні наступні сторінки з інформацією та описом гри, до яких він може перейти по навігаційні панелі зверху.

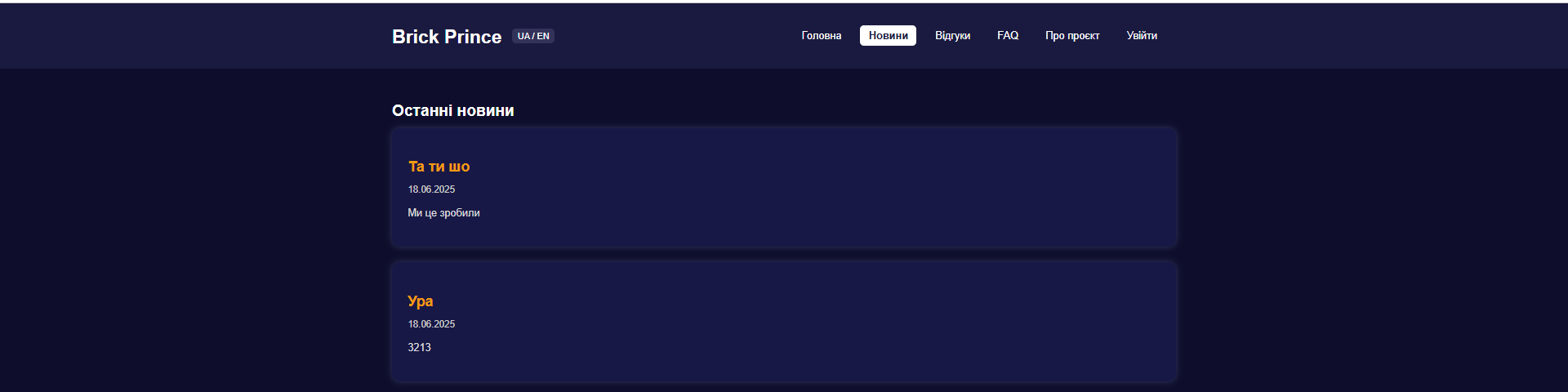


Рисунок 4.3 – Сторінка новин

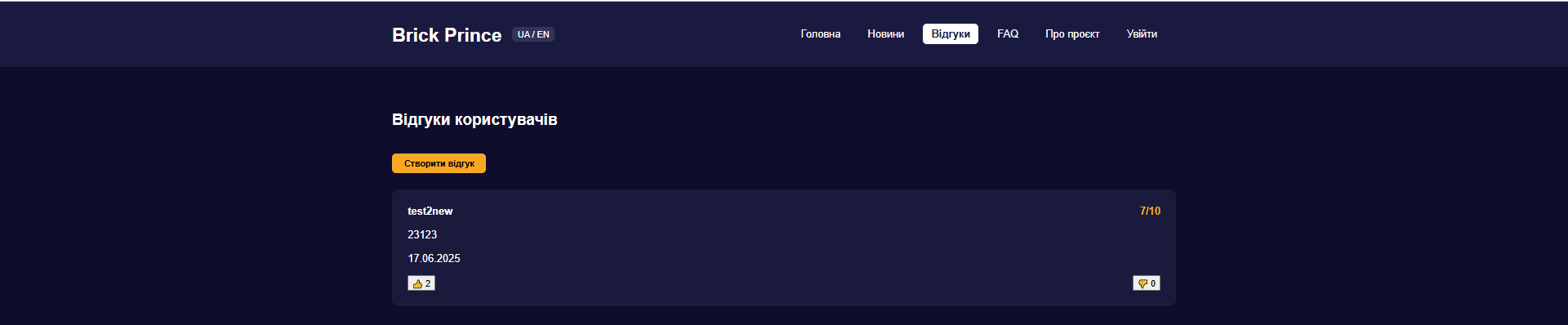


Рисунок 4.4 – Сторінка відгуків

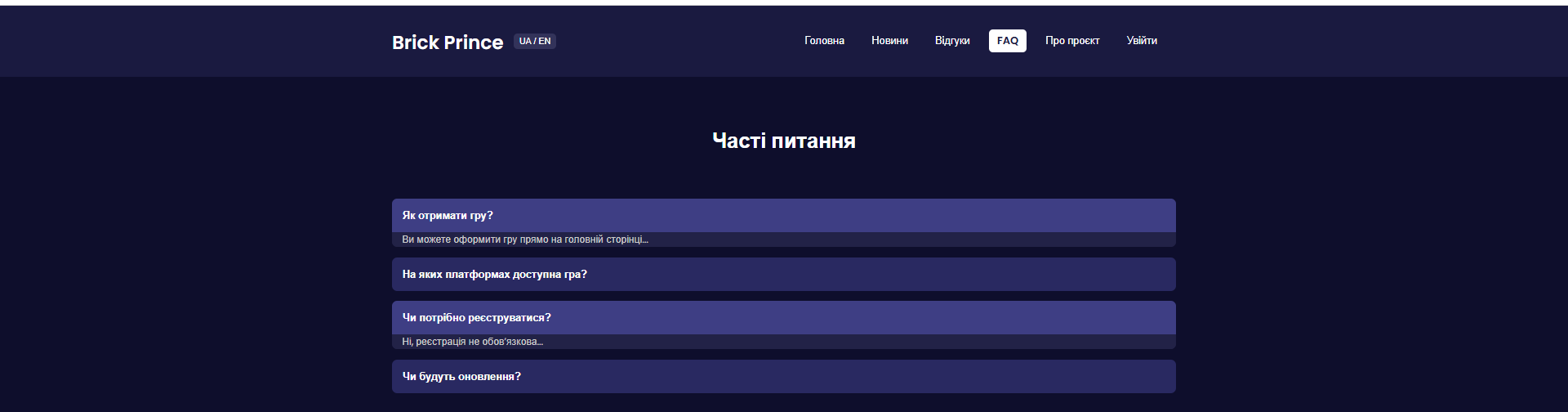


Рисунок 4.5 – Сторінка FAQ

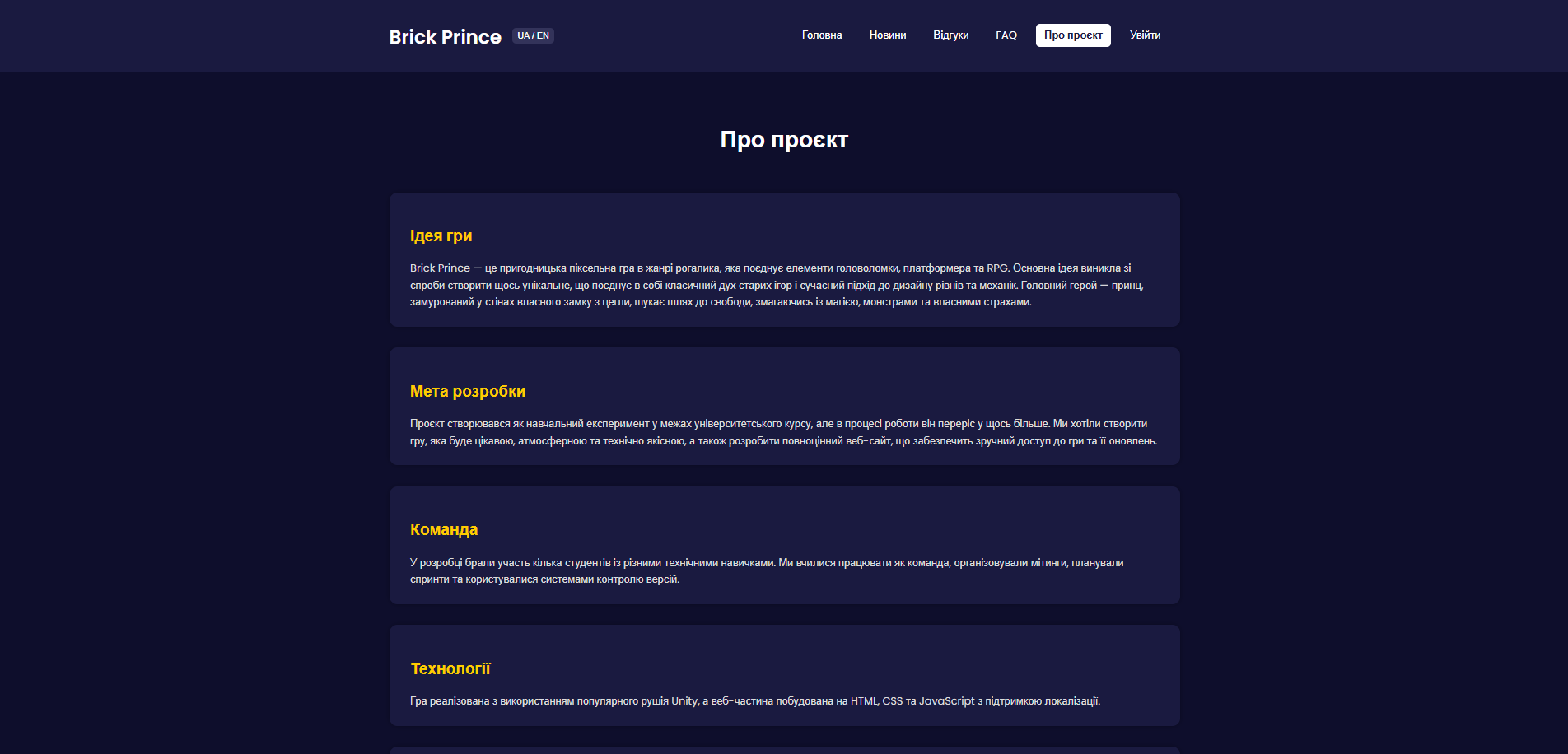


Рисунок 4.6 – Сторінка про проєкт

Незареєстрований користувач має змогу за бажанням, ніхто його не примушує, якщо йому так хочеться, перейти до форми реєстрації/авторизації.

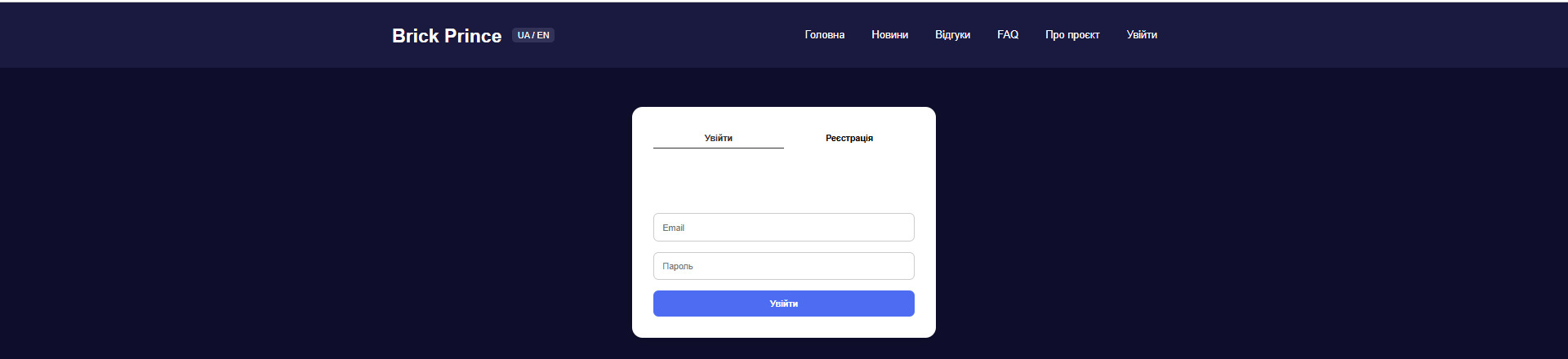


Рисунок 4.7 – Логін

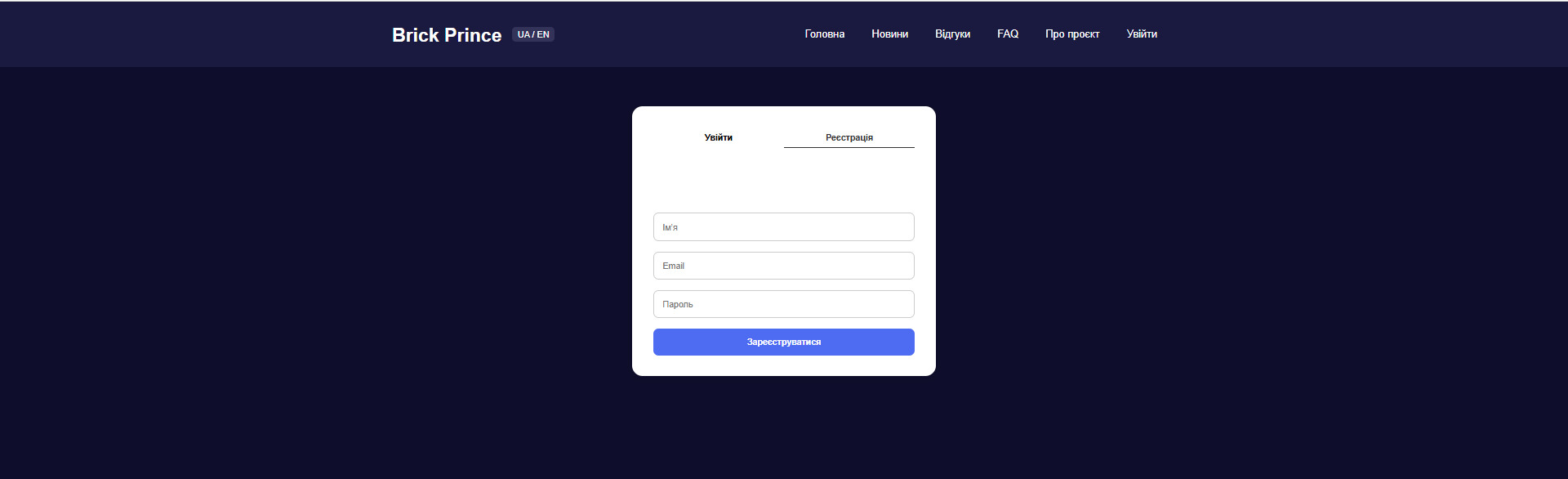


Рисунок 4.8 - Реєстрація

Далі(див.рис.4.8) користувач може увійти в свій аккаунт, який він перед цим зареєстрував, або зареєструвати новий.

Програмна обробка реєстрації:  
 **registerForm.addEventListener("submit", async (e) => {**

**e.preventDefault();**

**const name = document.getElementById("register-name").value.trim();**

**const email = document.getElementById("register-email").value.trim();**

**const password = document.getElementById("register-password").value;**

**if (!name || !email || !password) {**

**alert("Будь ласка, заповніть усі поля");**

**return;**

**}**

**const formData = new FormData();**

**formData.append("name", name);**

**formData.append("email", email);**

**formData.append("password", password);**

**try {**

**const res = await fetch("http://127.0.0.1:8000/register", {**

**method: "POST",**

**body: formData,**

**credentials: "include"**

**});**

**if (!res.ok) {**

**const err = await res.json();**

**alert("Помилка реєстрації: " + (err.error || res.status));**

**return;**

**}**

**const loginRes = await fetch("http://127.0.0.1:8000/login", {**

**method: "POST",**

**body: new URLSearchParams({ email, password }),**

**credentials: "include"**

**});**

**if (loginRes.ok) {**

**const meRes = await fetch("http://127.0.0.1:8000/me", {**

**credentials: "include"**

**});**

**const user = await meRes.json();**

**localStorage.setItem("user", JSON.stringify(user));**

**window.location.href = "/index.html";**

**} else {**

**alert("Не вдалося увійти після реєстрації");**

**}**

**} catch (err) {**

**console.error(" Register error:", err);**

**alert("Помилка з'єднання при реєстрації");**

**}**

**});**

Обробка входу:  
**loginForm.addEventListener("submit", async (e) => {**

**e.preventDefault();**

**const email = document.getElementById("login-email").value.trim();**

**const password = document.getElementById("login-password").value;**

**if (!email || !password) {**

**alert("Будь ласка, введіть email і пароль");**

**return;**

**}**

**const formData = new FormData();**

**formData.append("email", email);**

**formData.append("password", password);**

**try {**

**const res = await fetch("http://127.0.0.1:8000/login", {**

**method: "POST",**

**body: formData,**

**credentials: "include"**

**});**

**if (!res.ok) {**

**const err = await res.json();**

**alert("Помилка входу: " + (err.error || res.status));**

**return;**

**}**

**const meRes = await fetch("http://127.0.0.1:8000/me", {**

**credentials: "include"**

**});**

**const user = await meRes.json();**

**localStorage.setItem("user", JSON.stringify(user));**

**window.location.href = "/index.html";**

**} catch (err) {**

**console.error(" Login error:", err);**

**alert("Помилка з'єднання при вході");**

**}**

**});**

**});**

Запити на бекенді:  
Реєстрація:

**@app.post("/register")**

**async def register(**

**name: str = Form(...),**

**email: str = Form(...),**

**password: str = Form(...)**

**):**

**conn = get\_db()**

**cursor = conn.cursor()**

**hashed\_pw = hashlib.sha256(password.encode()).hexdigest()**

**cursor.execute("SELECT id FROM users WHERE email = ?", (email,))**

**if cursor.fetchone():**

**conn.close()**

**return JSONResponse(status\_code=400, content={"error": "Користувач уже існує"})**

**cursor.execute(**

**"INSERT INTO users (name, email, password, role) VALUES (?, ?, ?, ?)",**

**(name, email, hashed\_pw, "user")**

**)**

**conn.commit()**

**conn.close()**

**return {"message": "Реєстрація успішна"}**

Запит логіну:  
  
**@app.post("/login")**

**async def login(request: Request, email: str = Form(...), password: str = Form(...)):**

**conn = get\_db()**

**cursor = conn.cursor()**

**hashed\_pw = hashlib.sha256(password.encode()).hexdigest()**

**cursor.execute("SELECT \* FROM users WHERE email = ? AND password = ?", (email, hashed\_pw))**

**user = cursor.fetchone()**

**conn.close()**

**if user:**

**request.session["user\_id"] = user["id"]**

**request.session["role"] = user["role"]**

**return {"message": "Вхід успішний"}**

**else:**

**return JSONResponse(status\_code=401, content={"error": "Неправильний email або пароль"})**

Після того як користувач зареєструвався або залогінився його перекидає назад на головну сторінку, де вже прихований футер та можна отримати гру на свою пошту при натисканні на просту кнопку “Оформити гру”.

З нових змін для нашого користувача це оновлена навігаційна панель, тепер замість кнопки “Увійти” там кнопка “Профіль”(див.рис.4.9).

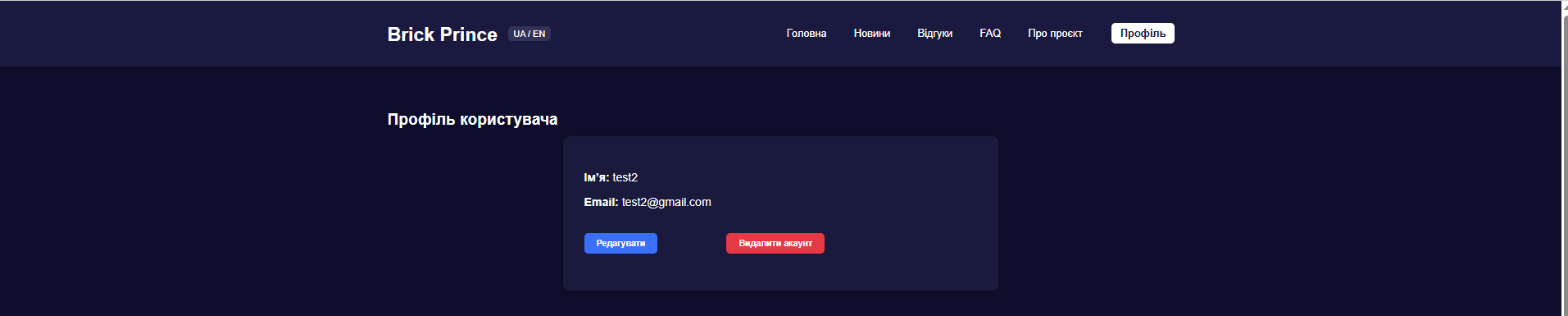


Рисунок 4.9 – Профіль звичайного зареєстрованого користувача

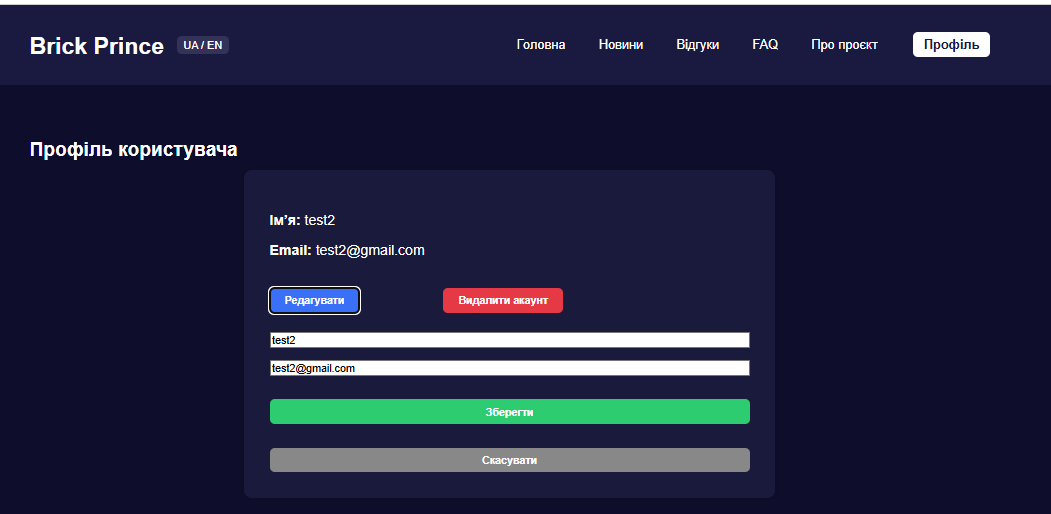


Рисунок 4.10 – Форма редагування профілю

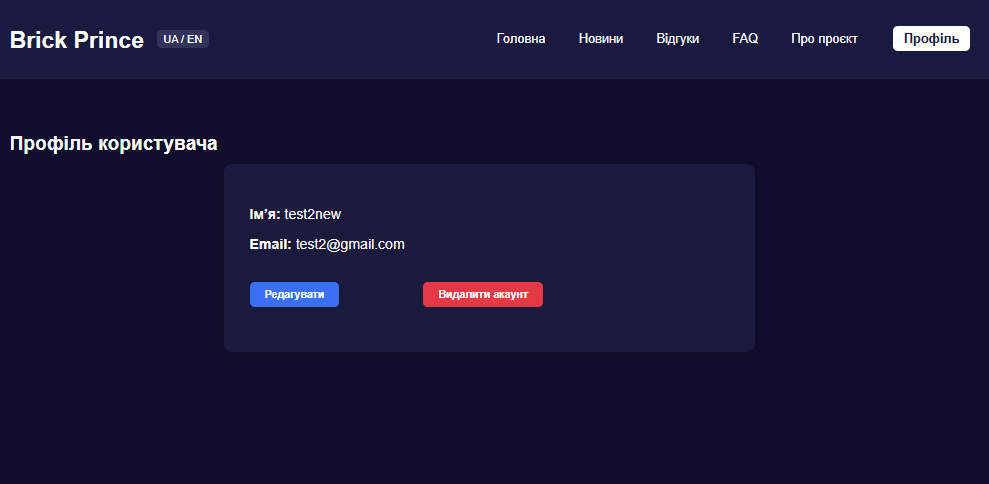


Рисунок 4.11 – Профіль після редагування

Користувачу надається можливість редагувати свій профіль, або видалити свій аккаунт.

Програмна реалізація:  
JS частина:

**editForm.addEventListener("submit", async (e) => {**

**e.preventDefault();**

**const name = newNameInput.value.trim();**

**const email = newEmailInput.value.trim();**

**const formData = new FormData();**

**formData.append("name", name);**

**formData.append("email", email);**

**try {**

**const res = await fetch("http://127.0.0.1:8000/update-profile", {**

**method: "POST",**

**body: formData,**

**credentials: "include"**

**});**

**if (res.ok) {**

**const updated = await res.json();**

**localStorage.setItem("user", JSON.stringify(updated));**

**nameSpan.textContent = updated.name;**

**emailSpan.textContent = updated.email;**

**editForm.classList.add("hidden");**

**alert("Профіль оновлено");**

**} else {**

**const err = await res.json();**

**alert("Помилка оновлення: " + (err.error || ""));**

**}**

**} catch (err) {**

**console.error("Fetch error:", err);**

**alert("Сталася помилка");**

**}**

**});**

Видалення акаунту:

**deleteBtn.addEventListener("click", async () => {**

**const confirmed = confirm("Ви дійсно хочете видалити акаунт?");**

**if (!confirmed) return;**

**const res = await fetch("http://127.0.0.1:8000/me", {**

**method: "DELETE",**

**credentials: "include"**

**});**

**if (res.ok) {**

**localStorage.removeItem("user");**

**alert("Акаунт видалено");**

**window.location.href = "/index.html";**

**} else {**

**alert("Не вдалося видалити акаунт");**

**}**

**});**

API запит на бекенді:  
Редагування профілю:

**@app.post("/update-profile")**

**async def update\_profile(request: Request, name: str = Form(...), email: str = Form(...)):**

**user\_id = request.session.get("user\_id")**

**if not user\_id:**

**return JSONResponse(status\_code=401, content={"error": "Необхідно увійти"})**

**conn = get\_db()**

**cursor = conn.cursor()**

**cursor.execute("SELECT id FROM users WHERE email = ? AND id != ?", (email, user\_id))**

**if cursor.fetchone():**

**conn.close()**

**return JSONResponse(status\_code=400, content={"error": "Цей email вже використовується"})**

**cursor.execute("UPDATE users SET name = ?, email = ? WHERE id = ?", (name, email, user\_id))**

**conn.commit()**

**cursor.execute("SELECT id, name, email, role FROM users WHERE id = ?", (user\_id,))**

**updated\_user = cursor.fetchone()**

**conn.close()**

**return dict(updated\_user)**

Видалення профілю:  
**@app.delete("/me")**

**async def delete\_me(request: Request):**

**user\_id = request.session.get("user\_id")**

**if not user\_id:**

**return JSONResponse(status\_code=401, content={"error": "Ви не авторизовані"})**

**conn = get\_db()**

**cursor = conn.cursor()**

**cursor.execute("DELETE FROM users WHERE id = ?", (user\_id,))**

**conn.commit()**

**conn.close()**

**request.session.clear()**

**return {"message": "Акаунт успішно видалено"}**

Після ознайомлення з можливостями сторінки профілю користувач може перейти на сторінку відгуків та ознайомитися зі своїми новими можливостями. А саме появи кнопки створити відгук, та кнопок редагування видалення після створення відгуку(див.рис.4.12)

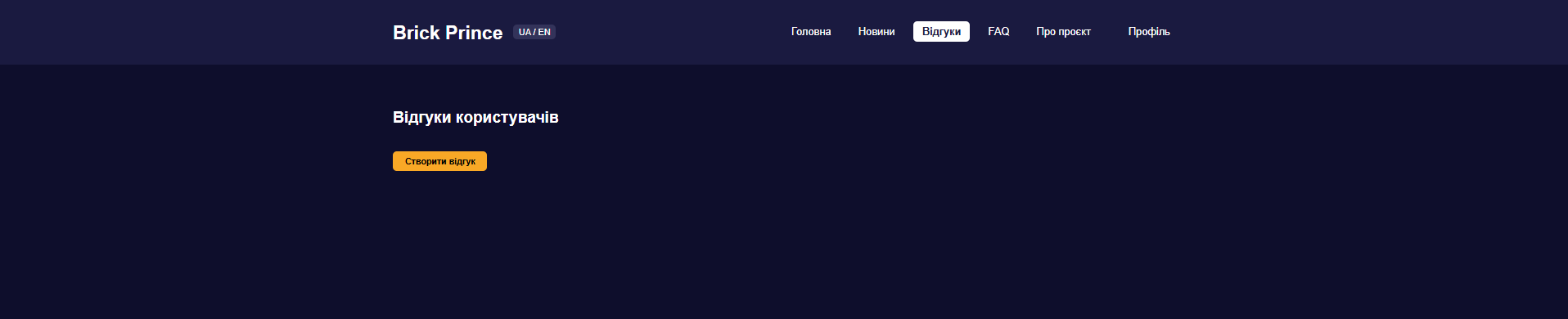


Рисунок 4.12 – Оновлена сторінка відгуків для залогіненого користувача

Користувачу відкривається окреме модульне вікно з полями(див.рис.4.13), які необхідно заповнити.

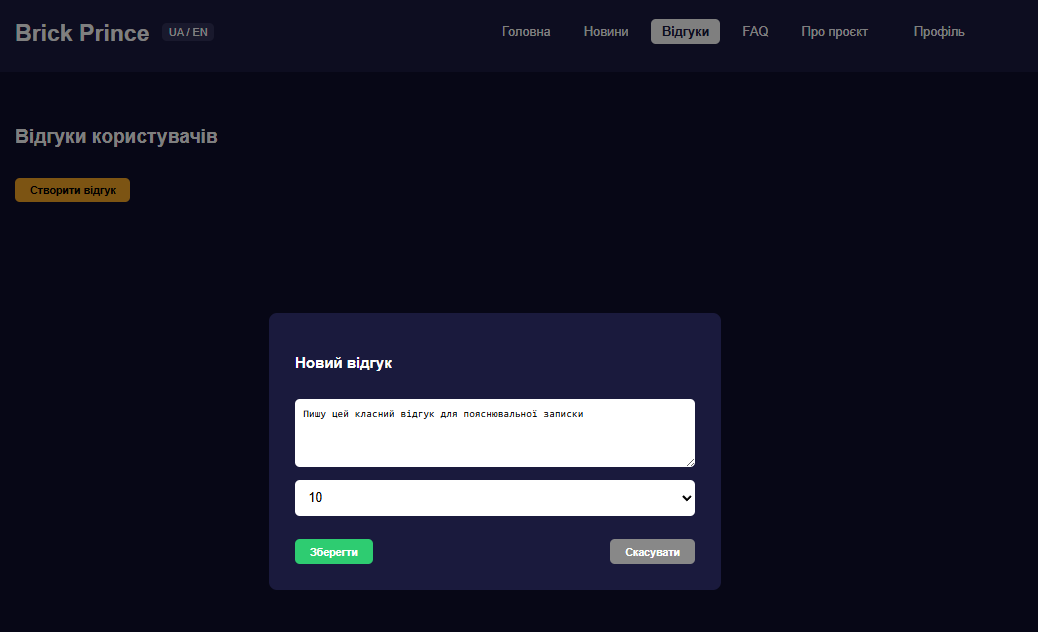


Рисунок 4.13 – Форма створення відгуку

При натисканні кнопки збереження відгук додається до БД, та стає доступним для перегляду в списку відгуків(див.рис.4.14).

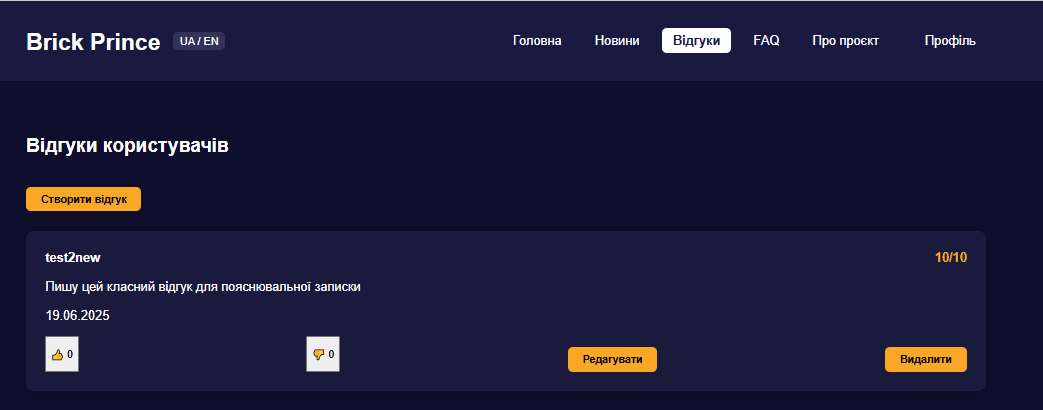


Рисунок 4.14 – Новий створений відгук додався до списку

Адмін панель. Для доступу до адмін панелі користувач має мати роль - адмін, та перейти до профілю. Йому відкриється доступ до прихованої кнопки переходу до адмін панелі.

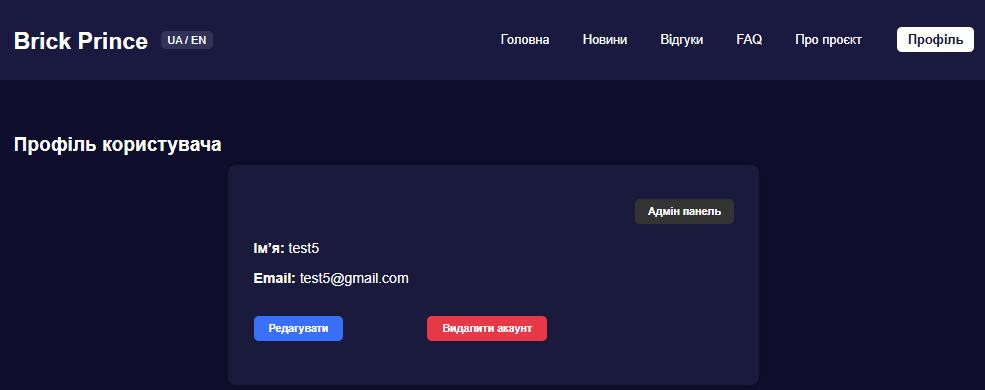


Рисунок 4.15 – Оновлений профіль для ролі адміністратор

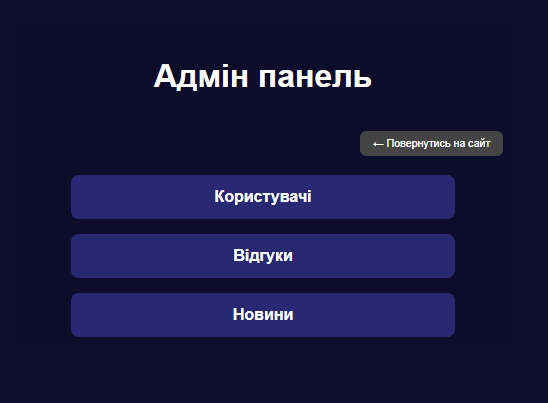


Рисунок 4.16 – Адмін панель(головне меню)

Подорожуючи кнопками адмін меню, користувач може взаємодіяти з записами користувачів(див.рис.4.17), відгуків(див.рис.4.18), новин(див.рис.4.19).

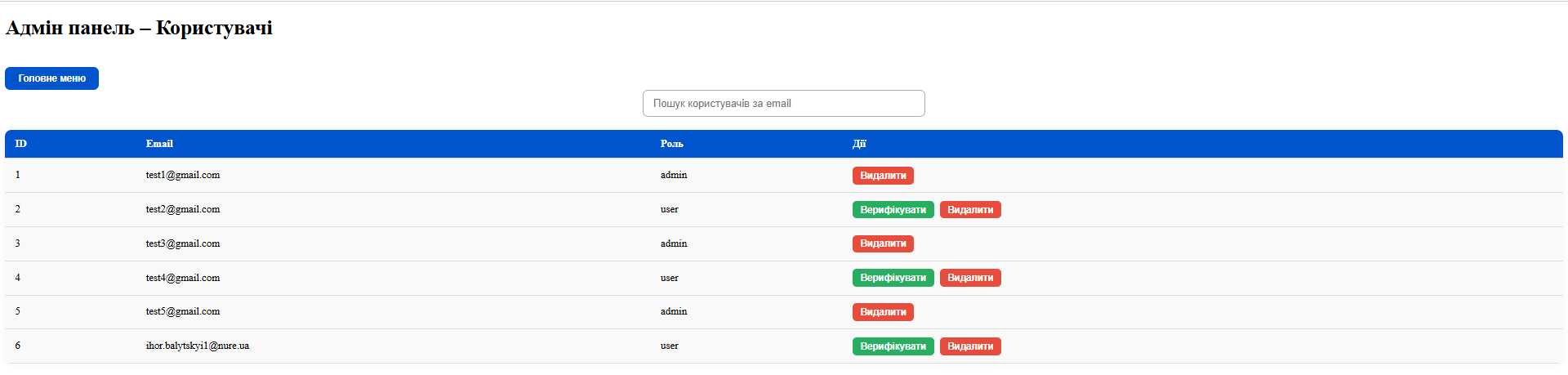


Рисунок 4.17 – Адмін панель(Вкладка користувачі)



Рисунок 4.18 – Адмін панель(Вкладка відгуки)

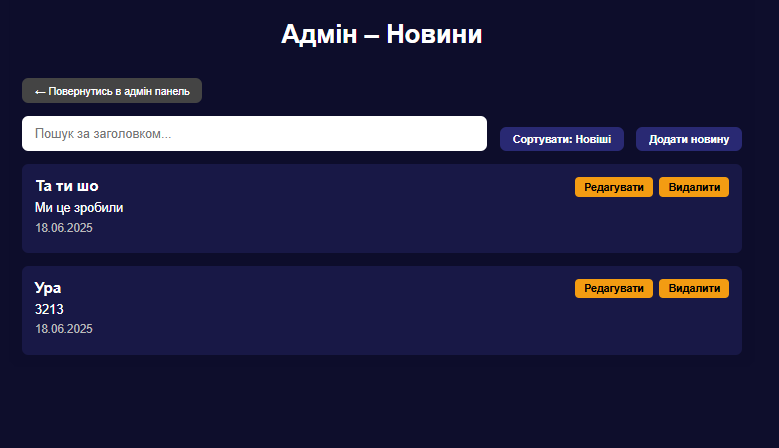


Рисунок 4.19 - Адмін панель (Вкладка новини)

При створенні/редагування новини користувач має заповнити/редагувати відповідні поля у модальному вікні(див.рис.4.20). Далі запис новини додається до БД, та всі користувачі можуть переглянути оновлений список меню на сторінці новин(див.рис.4.23).

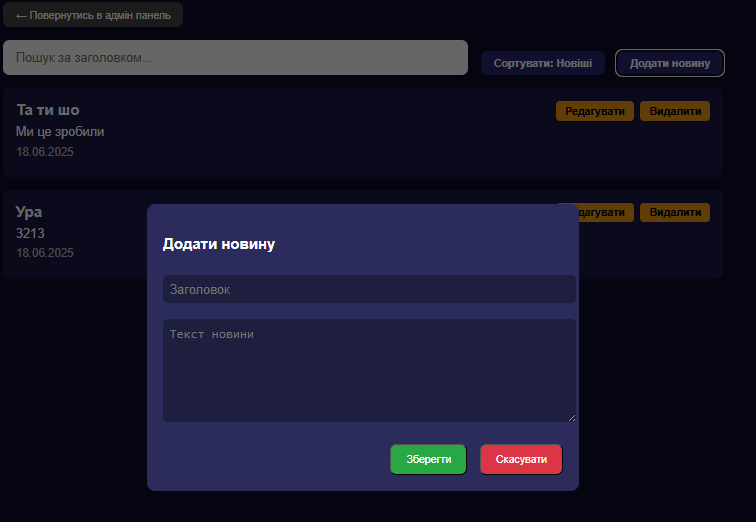


Рисунок 4.20 – Форма додавання новини

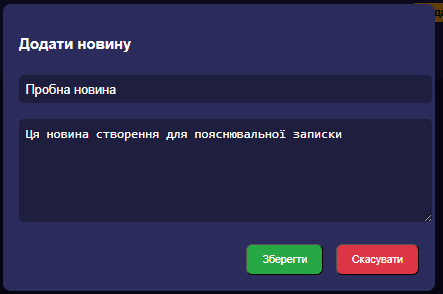


Рисунок 4.21 – Заповнена форма додавання новини

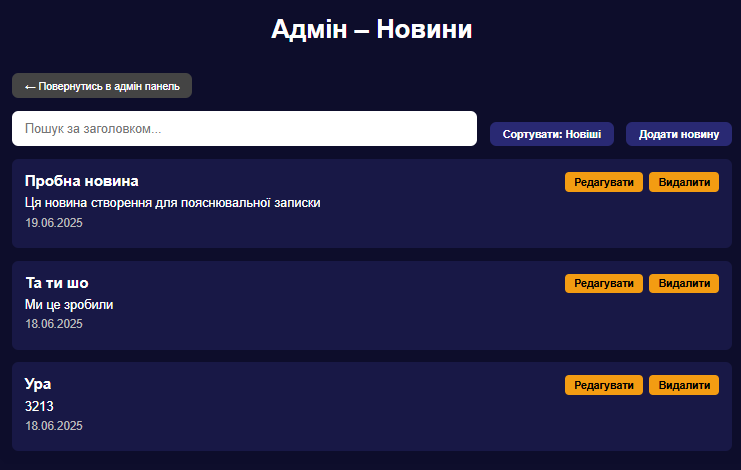


Рисунок 4.22 – Додана нова новина у списку панелі адміністратора

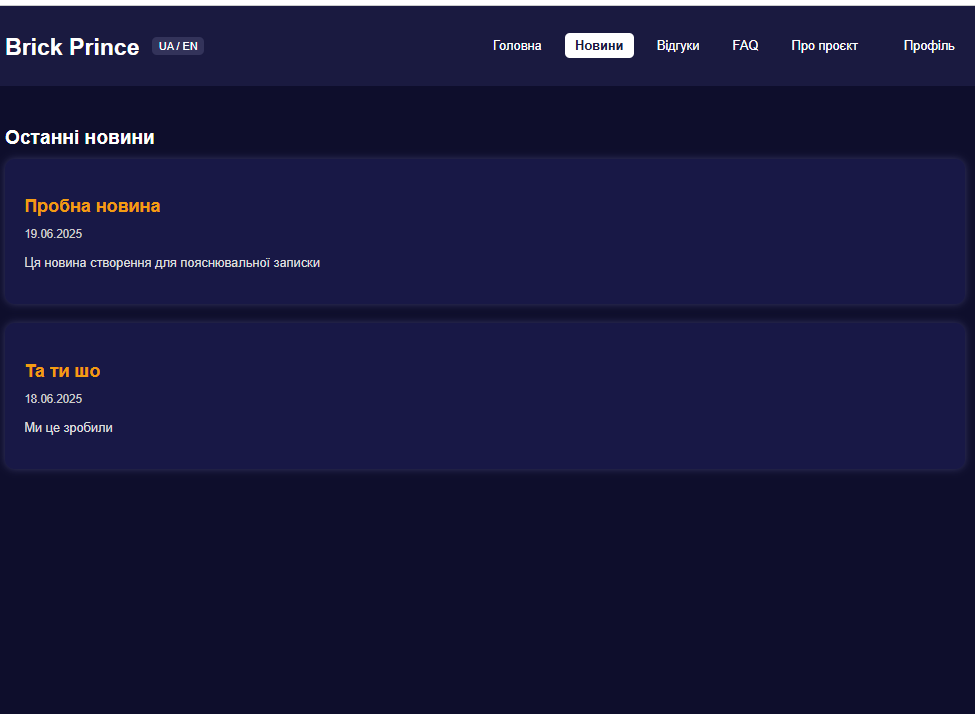


Рисунок 4.23 – Оновлений список усіх новин

Реалізація адмін панелі:  
JS частина(користувачі):

Завантаження користувачів з API:

**let users = [];**

**async function loadUsers() {**

**try {**

**const res = await fetch("http://127.0.0.1:8000/admin/users", { credentials: "include" });**

**if (!res.ok) throw new Error("Не вдалося завантажити користувачів");**

**users = await res.json();**

**renderUsers(users);**

**} catch (err) {**

**alert(err.message);**

**}**

**}**

Відобразити список користувачів:

**function renderUsers(usersList) {**

**usersTableBody.innerHTML = "";**

**if (usersList.length === 0) {**

**noUsersMsg.classList.remove("hidden");**

**return;**

**} else {**

**noUsersMsg.classList.add("hidden");**

**}**

**usersList.forEach(user => {**

**const tr = document.createElement("tr");**

**tr.innerHTML = `**

**<td>${user.id}</td>**

**<td>${user.email}</td>**

**<td>${user.role}</td>**

**<td>**

**${user.role === "user" ? `<button class="btn verify-btn" data-id="${user.id}">Верифікувати</button>` : ""}**

**<button class="btn delete-btn" data-id="${user.id}">Видалити</button>**

**</td>**

**`;**

**usersTableBody.appendChild(tr);**

**});**

**}**

Фільтрація користувачів за email:

**searchInput.addEventListener("input", () => {**

**const query = searchInput.value.toLowerCase();**

**const filtered = users.filter(u => u.email.toLowerCase().includes(query));**

**renderUsers(filtered);**

**});**

Обробник кнопок таблиці:

**usersTableBody.addEventListener("click", async (e) => {**

**if (e.target.classList.contains("delete-btn")) {**

**const userId = e.target.dataset.id;**

**if (confirm("Ви дійсно хочете видалити користувача?")) {**

**try {**

**const res = await fetch(`http://127.0.0.1:8000/admin/users/${userId}`, {**

**method: "DELETE",**

**credentials: "include"**

**});**

**if (!res.ok) throw new Error("Не вдалося видалити користувача");**

**alert("Користувача видалено");**

**loadUsers();**

**} catch (err) {**

**alert(err.message);**

**}**

**}**

**}**

JS частина(відгуки):  
Завантажуємо відгуки:

**async function loadFeedbacks() {**

**try {**

**const res = await fetch("http://127.0.0.1:8000/feedback", { credentials: "include" });**

**if (!res.ok) throw new Error("Помилка завантаження відгуків");**

**const feedbacks = await res.json();**

**renderFeedbacks(feedbacks);**

**} catch (error) {**

**feedbackList.innerHTML = `<p>${error.message}</p>`;**

**}**

**}**

Відображуємо відгуки:

**function renderFeedbacks(feedbacks) {**

**feedbackList.innerHTML = "";**

**if (feedbacks.length === 0) {**

**feedbackList.innerHTML = "<p>Відгуків немає.</p>";**

**return;**

**}**

**feedbacks.forEach(fb => {**

**const card = document.createElement("div");**

**card.className = "feedback-card";**

**card.innerHTML = `**

**<p><strong>Користувач ID:</strong> ${fb.user\_id}</p>**

**<p><strong>Відгук:</strong> ${fb.text}</p>**

**<p><strong>Рейтинг:</strong> ${fb.rating}/10</p>**

**<p><strong>Дата:</strong> ${new Date(fb.created\_at).toLocaleString("uk-UA")}</p>**

**<button class="delete-btn" data-id="${fb.id}">Видалити</button>**

**`;**

**card.querySelector(".delete-btn").addEventListener("click", () => {**

**if (confirm("Ви дійсно хочете видалити цей відгук?")) {**

**deleteFeedback(fb.id);**

**}**

**});**

**feedbackList.appendChild(card);**

**});**

**}**  
  
Видалити відгук:

**async function deleteFeedback(id) {**

**try {**

**const res = await fetch(`http://127.0.0.1:8000/admin/feedback/${id}`, {**

**method: "DELETE",**

**credentials: "include"**

**});**

**if (!res.ok) throw new Error("Не вдалося видалити відгук");**

**await loadFeedbacks(); // Оновити список після видалення**

**} catch (error) {**

**alert(error.message);**

**}**

**}**

JS частина(новини):  
Завантаження новин:

async function fetchNews() {

try {

const res = await fetch("http://127.0.0.1:8000/news", { credentials: "include" });

if (!res.ok) throw new Error("Помилка завантаження новин");

newsData = await res.json();

applyFilters();

} catch (err) {

alert(err.message);

}

}  
  
Фільтр та дії з новинами :

Застосування сортування за датою та пошуку за заголовком новин:

**function applyFilters() {**

**const searchTerm = searchInput.value.trim().toLowerCase();**

**filteredNews = newsData.filter(n => n.title.toLowerCase().includes(searchTerm));**

**filteredNews.sort((a, b) => {**

**const dateA = new Date(a.created\_at);**

**const dateB = new Date(b.created\_at);**

**return sortNewestFirst ? dateB - dateA : dateA - dateB;**

**});**

**renderNews();**

**}**

Функція виведення новин на сторінку: **function renderNews() {**

**newsList.innerHTML = "";**

**if (filteredNews.length === 0) {**

**newsList.innerHTML = "<p>Новини не знайдено.</p>";**

**return;**

**}**

**filteredNews.forEach(n => {**

**const item = document.createElement("div");**

**item.className = "news-item";**

**const date = new Date(n.created\_at).toLocaleDateString("uk-UA");**

**item.innerHTML = `**

**<h3>${n.title}</h3>**

**<p>${n.content}</p>**

**<p class="news-date">${date}</p>**

**<div class="news-actions">**

**<button class="edit-btn" data-id="${n.id}">Редагувати</button>**

**<button class="delete-btn" data-id="${n.id}">Видалити</button>**

**</div>**

**`;**

**newsList.appendChild(item);**

**});**

**document.querySelectorAll(".edit-btn").forEach(btn => {**

**btn.addEventListener("click", () => openModal("edit", btn.dataset.id));**

**});**

**document.querySelectorAll(".delete-btn").forEach(btn => {**

**btn.addEventListener("click", () => deleteNews(btn.dataset.id));**

**});**

**}**

Відкриття модального вікна для редагування та додавання новини:

**function openModal(mode, newsId = null) {**

**modal.classList.remove("hidden");**

**if (mode === "edit") {**

**modalTitle.textContent = "Редагувати новину";**

**const newsItem = newsData.find(n => n.id == newsId);**

**titleInput.value = newsItem.title;**

**contentInput.value = newsItem.content;**

**editNewsId = newsId;**

**} else {**

**modalTitle.textContent = "Додати новину";**

**titleInput.value = "";**

**contentInput.value = "";**

**editNewsId = null;**

**}**

**}**

**function closeModal() {**

**modal.classList.add("hidden");**

**titleInput.value = "";**

**contentInput.value = "";**

**editNewsId = null;**

**}**

**async function saveNews() {**

**const title = titleInput.value.trim();**

**const content = contentInput.value.trim();**

**if (!title || !content) {**

**alert("Заповніть всі поля");**

**return;**

**}**

**const formData = new FormData();**

**formData.append("title", title);**

**formData.append("content", content);**

**try {**

**let res;**

**if (editNewsId) {**

**res = await fetch(`http://127.0.0.1:8000/news/${editNewsId}`, {**

**method: "PUT",**

**body: formData,**

**credentials: "include",**

**});**

**} else {**

**res = await fetch("http://127.0.0.1:8000/news", {**

**method: "POST",**

**body: formData,**

**credentials: "include",**

**});**

**}**

**if (!res.ok) {**

**const err = await res.json();**

**throw new Error(err.error || "Помилка при збереженні");**

**}**

**await fetchNews();**

**closeModal();**

**} catch (err) {**

**alert(err.message);**

**}**

**}**

Запит на видалення новини:

**async function deleteNews(id) {**

**if (!confirm("Видалити новину?")) return;**

**try {**

**const res = await fetch(`http://127.0.0.1:8000/news/${id}`, {**

**method: "DELETE",**

**credentials: "include",**

**});**

**if (!res.ok) {**

**const err = await res.json();**

**throw new Error(err.error || "Помилка при видаленні");**

**}**

**await fetchNews();**

**} catch (err) {**

**alert(err.message);**

**}**

**}**

**searchInput.addEventListener("input", applyFilters);**

**sortToggle.addEventListener("click", () => {**

**sortNewestFirst = !sortNewestFirst;**

**sortToggle.textContent = sortNewestFirst ? "Сортувати: Новіші" : "Сортувати: Старіші";**

**applyFilters();**

**});**

**addNewsBtn.addEventListener("click", () => openModal("add"));**

**cancelBtn.addEventListener("click", closeModal);**

**saveBtn.addEventListener("click", saveNews);**

**fetchNews();**

**});**

**5 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ**

У процесі створення веб застосунку для гри Brick Prince було реалізовано повноцінну систему, яка дозволяє не лише популяризувати гру через зручний інтерфейс взаємодії з користувачем, але й забезпечити базові комунікаційні, інформаційні та адміністративні функції, що відповідають сучасним вимогам до програмних продуктів у сфері ігрової індустрії.

Першочерговою метою проєкту було створення стабільного та доступного інструменту, який дозволяє користувачам дізнатися більше про гру, завантажити її та взаємодіяти з іншими гравцями шляхом публікації відгуків. Аналізуючи реалізовану систему, можна стверджувати, що поставлені цілі були досягнуті повною мірою.

Один із ключових результатів — успішне розділення ролей користувачів. В системі чітко розмежовано можливості звичайного користувача й адміністратора: користувач може залишати лише один відгук, оцінювати інші та змінювати інформацію про себе, тоді як адміністратор має доступ до керування новинами та модерації контенту. Такий підхід дозволив зберегти простоту взаємодії для більшості користувачів, водночас надаючи достатньо інструментів для обслуговування та контролю платформи.

Не менш важливим є те, що веб застосунокповністю працездатний без необхідності встановлення складного програмного забезпечення — користувачеві достатньо будь-якого сучасного браузера. Сторінки коректно відображаються на різних розширеннях екранів, елементи інтерфейсу логічно згруповані, реалізована базова адаптивність. Це сприяє зниженню барʼєру входу для нових користувачів і підвищує охоплення аудиторії.

Функціонал з відправленням гри через email також виявився ефективним засобом поширення. Він дозволяє контролювати кількість завантажень, потенційно забезпечує канал для подальшої взаємодії з користувачами та не вимагає реєстрації, що знижує фрикційність першого контакту з проєктом.

Особливу увагу було приділено реалізації блоку відгуків, який підтримує повноцінну взаємодію спільноти: можливість залишити оцінку, дати текстовий коментар, а також оцінити думки інших користувачів за допомогою реакцій. Це не тільки створює динамічний соціальний елемент, а й формує живу базу зворотного звʼязку, яка може бути корисною для подальшого розвитку гри.

З точки зору стабільності системи, під час тестування веб застосунокпродемонстрував належну надійність. Основні сценарії взаємодії не викликали збоїв, усі ключові маршрути відпрацьовували коректно, а обробка помилок реалізована достатньо грамотно, з відповідними повідомленнями для користувача у разі введення недійсних або неповних даних.

Загалом, функціонал, закладений у проєкт, охоплює більшість базових потреб для презентації гри та підтримки спільноти довкола неї. Незважаючи на відносну простоту архітектури, проєкт має потенціал для масштабування, а всі компоненти створені з урахуванням можливості подальшого розширення. Це стосується як додавання нових мов інтерфейсу, так і впровадження нових розділів (наприклад, галереї, статистики або форуму).

У процесі розробки вебзастосунку Brick Prince виникали різноманітні труднощі, які були пов’язані як з технічними аспектами реалізації, так і з організацією логіки окремих функціональних блоків. Виявлення та вирішення цих проблем дозволило значно покращити якість застосунку, зробити його більш стабільним і зручним для кінцевого користувача.

Однією з перших проблем стала побудова структури бази даних таким чином, щоб кожен користувач міг залишити лише один відгук, при цьому зберігаючи можливість редагування або видалення власного коментаря. Первинно розглядалася ідея дозволити кілька відгуків, однак після аналізу було прийнято рішення обмежити до одного запису на користувача — це вимагало унікального зв’язку user\_id у таблиці feedback, який був реалізований як обмеження на рівні схеми БД. Проблема полягала у правильному врахуванні цього обмеження при оновленні даних, що було вирішено шляхом додаткових перевірок на бекенді перед виконанням запитів.

Ще однією важливою складністю стала реалізація реакцій у вигляді лайків і дизлайків. На перший погляд, завдання виглядало тривіальним, однак логіка взаємодії вимагала контролю виключності: користувач не міг залишити одночасно лайк і дизлайк, а також повинен мати змогу скасувати реакцію. Початковий варіант реалізації спричиняв дублювання записів у базі, що негативно впливало на коректність підрахунків. Це було виправлено шляхом введення комбінованого унікального індексу UNIQUE(user\_id, feedback\_id) у таблиці likes та вдосконаленням логіки на сервері.

У ході реалізації надсилання гри на електронну пошту незареєстрованому користувачеві зʼявилася проблема з відправленням листів із локального середовища розробки. Через обмеження деяких SMTP-серверів виникали помилки автентифікації. Було вирішено використовувати перевірений поштовий сервер із підтримкою TLS, а також додати обробку помилок, яка інформує користувача у разі технічної несправності або введення некоректної адреси.

Нарешті, ще одним викликом стало забезпечення узгодженості між фронтендом і бекендом, особливо в частині обробки динамічних дій, таких як видалення новин, редагування профілю чи завантаження нових відгуків. У деяких випадках зміни, внесені через API, не одразу коректно відображались на клієнтському боці. Це потребувало оптимізації механізмів оновлення сторінки або часткового ререндерінгу DOM, що згодом було реалізовано через повторні запити до сервера після критичних дій.

Загалом, попри низку викликів, усі основні труднощі були успішно подолані. Це стало можливим завдяки поетапному підходу до розробки, регулярному тестуванню та вдумливому рефакторингу коду, що в підсумку забезпечило стабільну й функціональну роботу застосунку Brick Prince.

**ВИСНОВКИ**

У процесі розробки веб застосунку Brick Prince було досягнуто основну мету, а саме створено повноцінну функціональну систему для представлення гри, взаємодії з користувачами через відгуки та новини, а також адміністрування контенту з боку розробників. Розроблений застосунок охоплює ключові потреби інди-ігрового проєкту: презентацію продукту, зворотний зв’язок з гравцями, поширення оновлень та ефективну модерацію взаємодій. Така система значно покращує сприйняття гри кінцевими користувачами й забезпечує розробникам необхідні інструменти для подальшого розвитку.

Веб застосунок для підтримки Brick Prince є легко масштабованим і може бути доповнений новим функціоналом, зокрема: можливістю коментування новин, підтримкою markdown-форматування в текстах, розширеною аналітикою для адміністратора та інтеграцією з іншими платформами поширення ігор. Також можлива подальша адаптація під мобільні пристрої та підключення зовнішніх API для відстеження дій користувачів.

Таким чином, веб застосунок Brick Prince відповідає сучасним вимогам до функціональності, дизайну та надійності, а також є добрим прикладом практичної реалізації повного циклу розробки програмного забезпечення — від проєктування до впровадження та тестування.

Усі результати розробки, включаючи вихідний код, опубліковані у відкритому доступі на GitHub за посиланням:

<https://github.com/NureBalytskyiIhor/CPP/tree/main/2025_%D0%91_%D0%9A%D0%9A%D0%9F_%D0%9F%D0%97%D0%9F%D0%86-22-4_%D0%91%D0%B0%D0%BB%D0%B8%D1%86%D1%8C%D0%BA%D0%B8%D0%B9_%D0%86_%D0%9E>

**ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Krahmer M. FastAPI: Modern Web APIs with Python. 2nd ed. New York : O’Reilly Media, 2023. 280 p. (дата звернення: 10.03.2025)
2. Wikipedia. Internationalization and localization : en.wikipedia.org. – (дата звернення: 10.04.2025).
3. SendGrid : офіційний вебсайт. URL: [https://sendgrid.com/docs/for-developers/sending-email](https://sendgrid.com/docs/for-developers/sending-email%20) (дата звернення: 15.04.2025).
4. Laws of UX : principles that designers should consider. URL: https://lawsofux.com (дата звернення: 20.04.2025).
5. Epic Web Dev. Why you should probably be using SQLite. URL: <https://epicweb.dev/articles/why-you-should-probably-be-using-sqlite> (дата звернення: 03.05.2025).
6. ResearchGate. UI/UX Design: Principles, Trends and Best Practices. URL: [https://www.researchgate.net/publication/369889938](https://www.researchgate.net/publication/369889938%20) (дата звернення: 10.05.2025).
7. NN/g (Nielsen Norman Group). 10 Usability Heuristics for User Interface Design. URL: [https://www.nngroup.com/articles/ten-usability-heuristics](https://www.nngroup.com/articles/ten-usability-heuristics%20) (дата звернення: 12.05.2025).
8. Soegaard, M. UI Color Palette 2025: Best Practices, Tips, and Tricks for Designers [Електронний ресурс] / Mads Soegaard // Interaction Design Foundation. URL: [https://www.interaction-design.org/literature/topics/color-theory](https://www.interaction-design.org/literature/topics/color-theory%20) (дата звернення: 18.05.2025).
9. Microsoft. Visual Studio Code Documentation [Електронний ресурс] // Visual Studio Code Docs. URL: [https://code.visualstudio.com/docs](https://code.visualstudio.com/docs%20%20)  (дата звернення: 20.05.2025)
10. Van Rossum, G. The Python Standard Library: sqlite3 — DB-API 2.0 interface for SQLite databases [Електронний ресурс] / Guido van Rossum // Python documentation. URL: <https://docs.python.org/3/library/sqlite3.html>  
    (дата звернення: 27.05.2025)

ДОДАТОК А

Зміст файлу створення БД

**import sqlite3**

**from pathlib import Path**

**DB\_PATH = Path(\_\_file\_\_).resolve().parent.parent / "brickprince.db"**

**conn = sqlite3.connect(DB\_PATH)**

**cursor = conn.cursor()**

**# SQL-запити для створення таблиць**

**schema = """**

**CREATE TABLE IF NOT EXISTS users (**

**id INTEGER PRIMARY KEY AUTOINCREMENT,**

**name TEXT NOT NULL,**

**email TEXT UNIQUE NOT NULL,**

**password TEXT NOT NULL,**

**role TEXT CHECK(role IN ('user', 'admin')) NOT NULL DEFAULT 'user',**

**created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP**

**);**

**CREATE TABLE IF NOT EXISTS news (**

**id INTEGER PRIMARY KEY AUTOINCREMENT,**

**title TEXT NOT NULL,**

**content TEXT NOT NULL,**

**created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,**

**author\_id INTEGER,**

**FOREIGN KEY (author\_id) REFERENCES users(id)**

**);**

**CREATE TABLE IF NOT EXISTS feedback (**

**id INTEGER PRIMARY KEY AUTOINCREMENT,**

**user\_id INTEGER UNIQUE,**

**text TEXT NOT NULL,**

**rating INTEGER CHECK(rating BETWEEN 1 AND 10) NOT NULL,**

**created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,**

**FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE**

**);**

**CREATE TABLE IF NOT EXISTS likes (**

**id INTEGER PRIMARY KEY AUTOINCREMENT,**

**user\_id INTEGER,**

**feedback\_id INTEGER,**

**is\_like BOOLEAN NOT NULL,**

**UNIQUE(user\_id, feedback\_id),**

**FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE,**

**FOREIGN KEY (feedback\_id) REFERENCES feedback(id) ON DELETE CASCADE**

**);**

**"""**

**cursor.executescript(schema)**

**conn.commit()**

**conn.close()**

ДОДАТОК Б

SRS документ

(Веб застосунок для підтримки гри Brick Prince)

(Балицький Ігор Олександрович)

**Software Requirements Specification**

**Document**

**Version: 1** **Date: (05/20/2025)**

1. Вступ

1.1 Призначення

Цей документ є специфікацією програмних вимог (SRS) до вебзастосунку Brick Prince. Він призначений для розробників, тестувальників, замовників, технічних експертів і всіх зацікавлених сторін, які беруть участь у процесі створення, тестування та підтримки даного програмного забезпечення. Документ визначає функціональні та нефункціональні вимоги до системи, а також встановлює межі та основні припущення про її реалізацію.

Метою документа є надання чітких і зрозумілих вимог до програмного продукту, що слугуватимуть основою для його проєктування, розробки та перевірки.

1.2 Область застосування

Brick Prince — це сучасний вебзастосунок, розроблений як інформаційна платформа для підтримки мобільної гри однойменного ігрового проєкту. Застосунок дозволяє користувачам отримувати гру, стежити за новинами, залишати відгуки, оцінювати інші коментарі, а адміністраторам — модерувати контент і керувати користувачами.

Система охоплює такі функції:

* реєстрація та вхід користувачів;
* перегляд новин про гру;
* завантаження гри за email;
* створення та редагування відгуків;
* оцінка відгуків через лайки/дизлайки;
* панель адміністратора з доступом до користувачів, новин, відгуків;
* підтримка багатомовного інтерфейсу.

Програмне забезпечення призначене для використання через сучасні веб браузери на настільних комп’ютерах і мобільних пристроях. Система не включає безпосередньо геймплей гри, її серверну частину або функції внутрішньоігрового процесу — лише зовнішню підтримку гри з боку спільноти та адміністраторів.

Основною метою є створення зручного й масштабованого інструменту для взаємодії між гравцями та розробниками проєкту Brick Prince.

1.3 Терміни, скорочення та абревіатури

У цьому документі використовуються такі терміни та скорочення:

* SRS — Software Requirements Specification, специфікація програмних вимог
* UI — User Interface, користувацький інтерфейс
* UX — User Experience, досвід користувача
* CRUD — Create, Read, Update, Delete — базові операції з даними
* SQLite — легка реляційна база даних
* REST API — архітектурний стиль прикладного інтерфейсу для взаємодії клієнт-сервер
* HTML/CSS/JS — мови для створення інтерфейсів вебзастосунків
* FastAPI — фреймворк для побудови веб-API на Python
* JWT — JSON Web Token — формат для авторизації
* i18n — Internationalization — підтримка багатомовності
* SMTP — Simple Mail Transfer Protocol — протокол надсилання електронних листів

1.4 Перелік використаних джерел

У документі було використано наступні джерела:

1. ISO/IEC/IEEE 29148:2018. Systems and software engineering — Life cycle processes — Requirements engineering.
2. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання.
3. Офіційна документація FastAPI. URL: https://fastapi.tiangolo.com
4. Офіційна документація SQLite. URL: [https://sqlite.org](%20https://sqlite.org)
5. Mozilla Developer Network (MDN): HTML, CSS, JavaScript Docs. URL: [https://developer.mozilla.org](%20https://developer.mozilla.org)
6. Python Software Foundation. Official Python Docs. URL: [https://docs.python.org](%20https://docs.python.org)
7. Bootstrap Documentation. URL: [https://getbootstrap.com](%20https://getbootstrap.com)
8. Git SCM Documentation. URL: [https://git-scm.com/doc](%20https://git-scm.com/doc)
9. RFC 5321 – Simple Mail Transfer Protocol (SMTP). URL: https://datatracker.ietf.org/doc/html/rfc5321

1.5 Огляд

Даний документ специфікації програмних вимог складається з трьох основних розділів:

* Розділ 1 надає загальну інформацію про призначення, обсяг, скорочення та джерела, що використовуються.
* Розділ 2 описує загальну структуру програмного продукту, включаючи його функціональні можливості, характеристики користувача, обмеження та припущення.
* Розділ 3 детально описує конкретні вимоги до функціоналу, інтерфейсів, продуктивності, безпеки, атрибутів системи тощо. Саме на цей розділ орієнтуються розробники під час реалізації програмного продукту.

Після ознайомлення з цим документом користувачі, розробники та інші зацікавлені особи отримають чітке уявлення про вимоги до вебзастосунку Brick Prince, що дозволить успішно реалізувати його функціональні та нефункціональні компоненти.

2. Загальний опис

Цей розділ описує загальні фактори, що впливають на розробку програмного продукту Brick Prince WebApp, а також пояснює передумови та контекст для кращого розуміння вимог, сформульованих у розділі 3. Інформація, наведена тут, орієнтована на кінцевого замовника або користувача й подається у зрозумілій формі без складної технічної специфікації.

2.1 Перспектива програмного продукту

Веб застосунокBrick Prince є незалежною інформаційною системою, створеною спеціально для підтримки гравців, розробників і адміністрації однойменної гри у жанрі roguelike. Він забезпечує низку функцій: надання доступу до гри для незареєстрованих користувачів, зберігання відгуків, перегляд новин, взаємодію з користувачами, а також адміністрування контенту. Продукт є повністю самостійним і не потребує зовнішніх комерційних сервісів для основної функціональності.

Програмне забезпечення побудовано за клієнт-серверною архітектурою з REST API, де фронтенд реалізовано на HTML/CSS/JS, а серверна частина — на Python з використанням FastAPI. База даних — SQLite. Застосунок розгортається локально або на сервері, не вимагає складного середовища або ліцензованих систем.

Хоча подібні системи управління спільнотою гравців вже існують (наприклад, Steam Community, itch.io profiles), Brick Prince WebApp має більш вузьку спеціалізацію — він націлений на обслуговування однієї конкретної гри з урахуванням її унікальної механіки та специфічного аудиторного фокусу. Це дозволило реалізувати набір функцій, які відповідають саме потребам розробників гри та її користувачів, без перевантаження сторонніми можливостями.

У разі інтеграції з іншими сервісами (наприклад, надсиланням листів через SMTP або оновленням клієнта гри) інтерфейси взаємодії реалізуються відповідно до стандартних протоколів, забезпечуючи масштабованість і майбутню сумісність.

2.1.1 Інтерфейси системи

Система Brick Prince WebApp взаємодіє з зовнішніми службами лише частково. Основною інтеграцією є модуль надсилання електронних листів. Цей функціонал реалізовано через SMTP-протокол, що дозволяє надсилати листи з ключами доступу до гри користувачам, які ввели свій email на головній сторінці. Жодна стороння система типу платіжних шлюзів або соціальних мереж не інтегрується з нашим застосунком. У разі розширення функціональності можливе додавання API взаємодії з ігровим клієнтом або сервісами аналітики, однак наразі система залишається ізольованою та самодостатньою.

2.1.2 Користувацькі інтерфейси

Інтерфейс програми побудований на основі сучасного графічного веб-інтерфейсу (GUI), що дозволяє взаємодіяти із системою за допомогою миші та клавіатури. Особлива увага приділена зручності користування: меню, кнопки та форми розташовані логічно, з дотриманням принципів UX. Інтерфейс доступний українською та англійською мовами, перемикач знаходиться у правому верхньому куті.

Дизайн сайту адаптований до різних розмірів екранів і не містить надлишкових візуальних ефектів, що забезпечує швидке завантаження і комфортну роботу навіть на слабких пристроях. Враховано потреби користувачів з мінімальним досвідом — усі функції інтуїтивно зрозумілі, а помилки вводу супроводжуються поясненнями.

2.1.3 Апаратні інтерфейси

Програмне забезпечення Brick Prince WebApp не вимагає безпосередньої взаємодії з апаратним забезпеченням. Жодних спеціалізованих пристроїв, драйверів або додаткового обладнання не використовується. Система функціонує у веббраузері, що дозволяє уникнути апаратної залежності. Таким чином, вимоги до апаратних інтерфейсів відсутні.

2.1.4 Програмні інтерфейси

Застосунок взаємодіє з низкою програмних компонентів, які є необхідними для його роботи. Основним компонентом є СКБД SQLite, з якою серверна частина на Python/FastAPI працює напряму через відповідні бібліотеки. У частині роботи з поштовим сервером використовується бібліотека smtplib для встановлення SMTP-з'єднання.

Інтерфейс клієнт-сервер реалізовано у форматі REST API. Фронтенд надсилає HTTP-запити на бекенд, отримуючи у відповідь JSON-дані. Для забезпечення кросбраузерності використовуються стандартні технології HTML, CSS і JavaScript без використання специфічних фреймворків.

2.1.5 Комунікаційні інтерфейси

Основною комунікаційною взаємодією є передача даних між клієнтом і сервером через протокол HTTP(S). Всі запити до API реалізовано як REST-запити, відповіді формуються у форматі JSON. Надсилання листів відбувається через SMTP-з'єднання з налаштованим поштовим сервером. Встановлення з'єднання та автентифікація відбуваються відповідно до RFC 5321.

2.1.6 Обмеження пам’яті

Система Brick Prince WebApp не накладає суворих обмежень на використання оперативної чи постійної пам’яті. Оскільки база даних працює на SQLite та всі дані локально зберігаються у вигляді простих таблиць, навіть слабкі сервери або локальні машини можуть успішно запускати застосунок. У мінімальній конфігурації достатньо 512 МБ оперативної пам’яті, проте реальні вимоги можуть бути ще нижчими.

2.1.7 Операції

Застосунок підтримує як інтерактивний режим роботи користувача (авторизація, додавання відгуку, редагування профілю), так і пасивний перегляд публічної інформації (новини, відгуки, отримання гри). Резервне копіювання даних здійснюється вручну — копією файлу бази даних SQLite. Відновлення відбувається шляхом заміни файлу на резервний. Система не потребує обслуговування вночі чи спеціального режиму для виконання фонових задач.

2.1.8 Вимоги до адаптації під конкретну локацію

Для встановлення та запуску застосунку не потрібно жодних спеціальних змін у середовищі користувача. Система не вимагає фізичної адаптації серверного приміщення або закупівлі додаткового обладнання. Єдине обмеження — наявність браузера та Python-оточення на сервері. Перед встановленням необхідно створити та ініціалізувати базу даних, а також налаштувати SMTP, що виконується один раз згідно з інструкцією в документації.

2.2 Функції програмного продукту

Програмне забезпечення Brick Prince WebApp виконує роль вебзастосунку, що забезпечує зручну взаємодію користувача з інформацією про гру «Brick Prince», а також дозволяє залишати відгуки, переглядати новини про гру та отримувати копію гри на електронну пошту.

Основні функції системи охоплюють такі ключові напрями:

1. Надсилання гри незареєстрованим користувачам.  
 На головній сторінці сайту розміщено форму, в яку користувач може ввести адресу електронної пошти. Після надсилання адреси система автоматично генерує лист з посиланням на завантаження гри та надсилає його за допомогою SMTP-сервера. Це дозволяє поширювати гру серед нових гравців без необхідності проходження реєстрації.

2. Реєстрація та авторизація користувачів.  
 Система надає можливість створення облікового запису з унікальною адресою електронної пошти та паролем. Після успішної реєстрації користувач отримує доступ до особистого кабінету, де може переглядати і змінювати свої дані. Авторизовані користувачі отримують додаткові можливості, зокрема — залишати відгуки про гру.

3. Перегляд та керування відгуками.  
 Кожен зареєстрований користувач має право створити один відгук, який можна згодом редагувати або видалити. Всі відгуки виводяться на сторінці з пагінацією. Відгуки містять ім’я автора, текст, дату, оцінку гри (від 1 до 10), а також можливість поставити «лайк» або «дизлайк». Лише зареєстровані користувачі можуть взаємодіяти з реакціями.

4. Реакції на відгуки (лайки/дизлайки).  
 Користувач може поставити одну реакцію на відгук: лайк або дизлайк. Повторне натискання прибирає реакцію, а зміна типу реакції автоматично оновлює стан у базі. Це стимулює прозору оцінку вмісту без спаму.

5. Перегляд новин про гру.  
 На окремій сторінці виводиться стрічка новин, створена адміністраторами. Вона включає заголовок, дату публікації та повний текст кожної новини. Таким чином, користувачі можуть бути поінформовані про останні оновлення, зміни у грі або майбутні події.

6. Панель адміністратора.  
 Користувачі з роллю «admin» мають доступ до адміністративної панелі, де можуть:

* переглядати всіх зареєстрованих користувачів;
* змінювати роль або видаляти користувачів;
* створювати, редагувати або видаляти новини;
* переглядати список усіх відгуків та при потребі модерувати їх.

7. Локалізація інтерфейсу.  
 Інтерфейс сайту підтримує перемикання між українською та англійською мовами. Це забезпечує зручність використання для ширшої аудиторії. Мовне перемикання зберігається під час навігації між сторінками.

8. Адаптивний дизайн.  
 Сайт оптимізовано для перегляду на різних пристроях, включно з мобільними. Навігація та відображення вмісту адаптуються до розміру екрану, що забезпечує зручний доступ до всіх функцій незалежно від типу пристрою.

2.3 Характеристики користувачів

Цільова аудиторія вебзастосунку Brick Prince WebApp охоплює широкий спектр користувачів, від звичайних гравців мобільної гри до адміністраторів, відповідальних за її підтримку та розвиток. Загальні характеристики користувачів мають прямий вплив на побудову інтерфейсу, логіку взаємодії та рівень доступних функцій.

Кінцеві користувачі гри (гравці):  
Це переважно молоді люди віком від 13 до 35 років, з базовим або середнім рівнем цифрової грамотності. Вони мають досвід користування мобільними додатками, знайомі з базовими вебінтерфейсами та очікують просту й інтуїтивну взаємодію. Через це інтерфейс було реалізовано у зрозумілій формі з адаптивним дизайном і логічною структурою навігації, а вся взаємодія з системою відбувається в кілька кроків — наприклад, надсилання гри на пошту або залишення відгуку.

Зареєстровані користувачі:  
Ці користувачі, окрім доступу до публічного вмісту, мають змогу створювати та редагувати відгуки. Для них було реалізовано особистий кабінет та базові можливості керування профілем. Ця категорія не потребує спеціальної підготовки, тому функціонал побудовано максимально прозоро, з використанням знайомих елементів інтерфейсу (кнопки, форми, підтвердження дій).

Адміністратори системи:  
Це обмежене коло осіб, які мають розширені повноваження у вебзастосунку. Вони відповідають за створення новин, управління користувачами та модерування відгуків. Їх рівень технічної підготовки вищий за середній, тому інтерфейс адміністративної панелі передбачає доступ до функцій керування базою даних через прості форми, без потреби писати SQL-запити або взаємодіяти зі складною системою адміністрування.

Таким чином, при проєктуванні інтерфейсу особливу увагу було приділено простоті, зручності та наочності, щоб забезпечити доступність для гравців без технічної підготовки, при цьому не обмежуючи можливості досвідчених користувачів і адміністраторів.

2.4 Обмеження

Під час розробки вебзастосунку Brick Prince WebApp було враховано низку обмежень, які впливають на технічні рішення, архітектуру та реалізацію окремих функціональних можливостей. Ці обмеження не пов'язані безпосередньо з функціоналом, але визначають рамки, в межах яких має працювати програмне забезпечення.

Регуляторні політики та безпека даних.  
Усі дії, пов’язані зі збором і збереженням персональних даних користувачів (наприклад, ім’я, email), мають відповідати основним вимогам захисту конфіденційної інформації.

Обмеження апаратного середовища.  
Веб застосунокорієнтований на запуск у звичайному браузері без використання спеціалізованого обладнання. Він має коректно працювати на більшості сучасних пристроїв — як на стаціонарних ПК, так і на мобільних пристроях. Система не має доступу до специфічного апаратного забезпечення користувача, і тому не взаємодіє з зовнішніми пристроями.

Інтерфейси з іншими застосунками.  
Система використовує лише внутрішні API, які були спеціально реалізовані для цього проєкту. Взаємодії з сторонніми сервісами або застосунками не передбачено, що спрощує архітектуру та підвищує автономність рішення.

Паралельна робота та масштабування.  
Застосунок не підтримує горизонтальне масштабування на рівні користувача — кожен клієнт працює із системою незалежно. Паралельна обробка запитів на бекенді реалізована через стандартні можливості FastAPI.

Контроль та аудит.  
Повноцінна система аудиту не реалізована, однак облік донатів та створення новин фіксується із прив’язкою до користувача, що забезпечує базовий рівень простежуваності дій.

Вимоги до мови розробки.  
Весь серверний код реалізовано на мові Python (версія 3.10+) з використанням фреймворку FastAPI. Клієнтська частина реалізована з використанням HTML, CSS, JavaScript та бібліотеки Fetch API для взаємодії з сервером.

Надійність і критичність.  
Оскільки система не є життєво критичною, жорсткі вимоги до безвідмовності або високої доступності не встановлювалися. Проте передбачено захист від некоректного введення даних, перевірку прав доступу та інші базові механізми забезпечення стабільності.

Таким чином, визначені обмеження не лише задають межі технічної реалізації, а й допомагають зосередити розробку на ключових аспектах взаємодії користувача з системою без перевантаження зайвими складностями.

2.5 Припущення та залежності

У процесі створення вебзастосунку Brick Prince WebApp було зроблено низку припущень, які безпосередньо вплинули на формулювання функціональних і нефункціональних вимог. Зміна будь-якого з цих чинників може призвести до перегляду окремих частин вимог, викладених у цьому документі.

По-перше, передбачається, що користувачі мають стабільне підключення до Інтернету і можуть відкривати веб застосуноку сучасному браузері, що підтримує JavaScript та CSS3. Якщо підключення до мережі відсутнє або браузер застарілий, робота з системою стає неможливою або обмеженою.

По-друге, застосунок передбачає локальний запуск серверної частини або її розгортання на хостингу, що підтримує Python 3.10+, FastAPI та SQLite. Зміна середовища виконання, наприклад, перехід на іншу СУБД (MySQL або PostgreSQL), потребуватиме модифікацій у частині обробки запитів та ініціалізації бази даних.

По-третє, вихідною передумовою є використання тільки власного API без сторонніх сервісів або інтеграцій. Наприклад, поштові повідомлення формуються та надсилаються із сервера за допомогою стандартної бібліотеки, а не сторонніх поштових сервісів (Mailgun, SendGrid тощо). У разі потреби інтеграції з зовнішніми сервісами вимоги до системи значно зміняться.

Також, очікується, що кількість одночасних користувачів буде відносно невеликою, що дозволяє уникнути реалізації повноцінного механізму масштабування чи балансування навантаження. У разі зростання популярності проєкту й потреби обробляти запити великої кількості користувачів, архітектура серверної частини потребуватиме перегляду.

Крім того, передбачено, що адміністрування застосунку здійснюється технічно підготовленими користувачами, що мають прямий доступ до серверної частини або до панелі адміністратора. Автоматизоване резервне копіювання або розподілена система керування ролями наразі не впроваджені.

Таким чином, зазначені припущення і залежності окреслюють межі, у яких проєкт функціонує стабільно, і визначають потенційні точки адаптації системи у разі зміни зовнішніх умов або очікувань користувачів.

2.6 Розподіл вимог

У межах першої реалізації вебзастосунку Brick Prince WebApp було зосереджено увагу на реалізації основного функціоналу, необхідного для повноцінної роботи користувача із системою. Проте, враховуючи обмеження у часі, ресурсах і складності окремих компонентів, частина вимог була перенесена на наступні ітерації або розглядається як потенційне розширення у майбутніх версіях.

Функціонал, реалізований у поточній версії:

* Реєстрація та авторизація користувачів;
* Перегляд новин про гру;
* Перегляд та створення відгуків;
* Оцінювання відгуків (лайки/дизлайки);
* Надсилання гри на email користувачам;
* Панель адміністратора з можливістю керування новинами, відгуками та користувачами.

Функціонал, перенесений до наступних версій:

* Підтримка багатомовності в усьому інтерфейсі. На даному етапі реалізовано часткову локалізацію, однак повне охоплення інтерфейсу ще потребує часу. Планується включення розширеної підтримки української та англійської мов у майбутніх релізах.
* Система повідомлень або сповіщень для користувачів. Вона дозволила б інформувати про зміну статусу відгуку або новини. Цей функціонал наразі відкладений через складність реалізації асинхронної логіки на стороні сервера.
* Розширена система фільтрів для відгуків. На сьогодні реалізовано лише пагінацію та базове сортування. Фільтрація за рейтингом, датою або наявністю реакцій буде додана в наступному оновленні.
* Візуальна система аналітики або статистики в адмінпанелі. Можливість бачити кількість зареєстрованих користувачів, середній рейтинг, активність користувачів тощо запланована для майбутніх версій.

Цей розподіл дає змогу ефективно планувати подальшу розробку, а також адаптувати застосунок до зростаючих потреб цільової аудиторії. Кожен з відкладених компонентів може бути розглянутий окремо під час наступного життєвого циклу проєкту, з урахуванням технічної та часової доцільності.

3. Конкретні вимоги (Specific Requirements)

Цей розділ містить деталізовані функціональні та нефункціональні вимоги до вебзастосунку Brick Prince. Усі вимоги формулюються у формі, придатній для тестування, відстеження, реалізації та перевірки.

3.1 Зовнішні інтерфейси (External Interfaces)

3.1.1 Інтерфейс користувача:

* Сторінки входу, реєстрації, профілю користувача, новин, відгуків, адмінпанелі.
* Формати вводу: текстові поля, паролі, числові значення.
* Формати виводу: HTML-сторінки, таблиці, картки, модальні вікна.

3.1.2 Інтерфейси бази даних:

* SQLite через SQLAlchemy
* Формати: JSON-відповіді API, SQL-запити.

3.1.3 Інтерфейси адміністратора:

* CRUD-функціональність для користувачів, новин, відгуків.

3.1.4 Комунікаційні інтерфейси:

* HTTP(S), RESTful API.

3.2 Функціональні вимоги (Functions)

3.2.1 Реєстрація та автентифікація:

* Система повинна забезпечувати реєстрацію з валідацією email та паролю.
* Користувач може увійти, вийти та змінити свої дані профілю.

3.2.2 Перегляд та створення відгуків:

* Користувач може створити лише один відгук, редагувати чи видалити його.
* Відгуки відображаються з датою, рейтингом, текстом, кількістю лайків/дизлайків.

3.2.3 Реакції на відгуки:

* Користувач може поставити лайк або дизлайк; повторне натискання скасовує дію.
* Можливе лише одне активне оцінювання на один відгук від користувача.

3.2.4 Адмінпанель:

* Можливість редагування/видалення новин, користувачів та відгуків.
* Пошук за email та назвами, експорт/імпорт CSV.

3.2.5 Надсилання гри:

* Незареєстрований користувач може ввести email для отримання гри.

3.3 Вимоги до продуктивності (Performance Requirements)

* 90% запитів до API повинні оброблятись менше ніж за 500 мс.
* Система має підтримувати одночасну роботу щонайменше 50 користувачів.

3.4 Логічні вимоги до бази даних (Logical Database Requirements)

* Таблиці: users, feedback, news, likes, reviews, settings.
* Всі зовнішні ключі мають бути збережені, забезпечено цілісність даних.
* Відгуки та реакції мають бути зв'язані з користувачем.

3.5 Обмеження дизайну (Design Constraints)

* Використання SQLite як БД.
* Реалізація на Python (FastAPI) для бекенду, HTML/CSS/JS для фронтенду.

3.5.1 Дотримання стандартів:

* Формати CSV для експорту/імпорту.
* RESTful структурування API.

3.6 Атрибути програмної системи (Software System Attributes)

3.6.1 Надійність:

* Система повинна функціонувати без збоїв при стандартному використанні.
* При втраті зв'язку відображається повідомлення, без втрати введених даних.

3.6.2 Доступність:

* Доступ до клієнтської частини 24/7, за винятком технічного обслуговування.

3.6.3 Безпека:

* Захист від SQL-ін’єкцій, XSS, CSRF.
* Паролі зберігаються в хешованому вигляді.

3.6.4 Підтримуваність:

* Структура проєкту повинна дозволяти легке доповнення нових модулів.

3.6.5 Портативність:

* Система має бути сумісною з основними браузерами (Chrome, Firefox, Edge).
* Можливість розгортання на будь-якому сервері з підтримкою Python 3.10+.

3.7 Організація специфікації вимог

Вимоги системи організовано за функціональною ієрархією, згруповано за основними підсистемами: користувацька частина, система відгуків, адмінпанель, механізм надсилання гри. Кожна група має чіткі, уніфіковані вимоги у вигляді специфікацій до вводу, обробки та виводу.