Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)

Кафедра Інформаційно-мережної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Розробка онлайн гри на декілька гравців з голосовим чатом

(тема)

Виконав:

здобувач 4 року навчання,

групи ТРІМІ-21-1

Володимир Українцев
(власне ім'я, прізвище)

Спеціальність 172 Телекомунікації і радіотехніка
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформаційно-мережна інженерія
(повна назва освітньої програми)

Керівник доцент Андрій Костромицький
(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри _____ Безрук В. М.
(підпис) (прізвище, ініціали)

2025 р.

Не містить відомостей, заборонених до відкритого публікування

Студент _____ / Українцев В.О.
(підпис) (прізвище та ініціали)

Керівник _____ / Костромицький А. І.
(підпис) (прізвище та ініціали)

Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій

Кафедра Інформаційно-мережна інженерія

Рівень вищої освіти перший (бакалаврський)

Спеціальність 172 Телекомунікації та радіотехніка

Тип програми освітньо-професійна

Освітня програма Інформаційно-мережна інженерія

ЗАТВЕРДЖУЮ

Зав. кафедри _____
(підпис)

«___»_____ 20 __ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Українцеву Володимиру Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка онлайн гри на декілька гравців з голосовим чатом

Затверджена наказом по університету від « 23 » травня 20 25 р. № 410 Ст

2. Термін подання студентом роботи до екзаменаційної комісії «15»червня2025р.

3. Вихідні дані до роботи Розробити онлайн гру типу Пінг Понг на двох гравців з голосовим чатом на базі технології WebRTC

4. Перелік питань, що потрібно опрацювати в роботі

Вступ

1 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПЗ

3 РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА

4 РОЗГОРТАННЯ ПРОЕКТА В РОБОЧОМУ СЕРЕДОВИЩІ ЗА ДОПОМОГОЮ DOCKER

5 ТЕСТУВАННЯ ДОДАТКА

Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п. 5 включається до завдання за рішенням випускової кафедри) Слайди у форматі Power Point

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк/терміни виконання етапів роботи	Примітка
1	Підбір і аналіз літератури з теми кваліфікаційної роботи	23.05.2025	
2	Виконання розділу 1	29.05.2025	
3	Виконання розділу 2	31.05.2025	
4	Виконання розділу 3	05.06.2025	
5	Виконання розділу 4	07.06.2025	
6	Підготовка доповіді	10.06.2025	

Дата видачі завдання 23 05 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ доцент Андрій Костромицький
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 51 с., 26 рис., 1 додаток, 11 джерел.

Об’єкт роботи – онлайн гра на декількох гравців з голосовим чатом.

Мета роботи – розробка онлайн гри.

Виконано розробку онлайн гри з голосовим чатом. Розроблено архітектуру програмної системи, інтерфейс користувача, серверну та клієнтську частини, проведено розгортаня програмної системи за допомогою ПЗ Docker в робочому середовищі.

WEB DEVELOPMENT, UI, REACT, EXPRESS, JAVASCRIPT, РОЗРОБКА

ABSTRACT

Explanatory note: 51 p., 26 figures, 1 appendices, 11 sources.

Object of work – development of multiplayer video game with voice chat.

Purpose – to develop a multiplayer video game with voice chat.

The development of an online game with voice chat was completed. The architecture of the software system, user interface, server and client parts were developed, and the software system was deployed using Docker software in the working environment.

WEB DEVELOPMENT, UI, REACT, EXPRESS, JAVASCRIPT

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

Веб – всесвітнє павутиння

ПЗ – програмне забезпечення

SQL (Structured query language) – мова програмування призначення для управління реляційними базами даних

API (application programming interface) – Інтерфейс програмування застосунків

UI (User Interface) – користувацький інтерфейс

JWT (JSON Web Token) – веб-токен у форматі JSON

DOM (Document Object Model) – об’єктна модель документа

CSS (Cascading Style Sheets) – каскадні таблиці стилів

HTML (HyperText Markup Language) – мова розмітки гіпертексту

JS (JavaScript) – мова програмування JavaScript

UML (Unified Modeling Language) – уніфікована мова моделювання

ORM (Object-Relational Mapping) – об’єктно-реляційне відображення

ICE (Interactive Connectivity Establishment) – встановлення інтерактивного з'єднання

STUN (Session Traversal Utilities for NAT) – інструменти проходження сесій через NAT

TURN (Traversal Using Relays around NAT) – проходження через NAT за допомогою ретрансляторів

NAT (Network Address Translation) – трансляція мережевих адрес

SDP (Session Description Protocol) – протокол опису сесії

P2P (Peer-to-Peer) – однорангове з’єднання

TCP (Transmission Control Protocol) – протокол керування передачею

UDP (User Datagram Protocol) – протокол користувацьких дейтаграм

SPA (Single Page Application) – односторінковий застосунок

SSO (Single Sign-On) – єдиний вхід

ВСТУП

У сучасному світі онлайн-ігри набули великої популярності і стали окремої галуззю економіки. Дослідження показують, що станом на 2024 рік, кількість гравці в відео-ігри становить близько 3.3 мільярдів людей по всьому світу, а найбільша частина з них, це люди віком від 18 до 34 років [1].

Найпопулярнішим засобом комунікації в багатокористувацьких іграх є текстовий чат, коли голосовий або вербальний чат є другим за популярністю засобом, хоча голосовий чат дозволяє швидше та зручніше обмінюватись інформацію між гравцями під час ігрового процесу. Такий стан речей склався, через декілька причин, але дві основні з них це технічна складність реалізації голосового чату та дотримання законів про онлайн-ігри компаніями-розробниками онлайн-ігор. Наприклад, в окремих країнах Європейського Союзу, Азії та частинах США, законодавчими органами цих країн були введені закони, що регулюють використання голосового чату серед гравців, що призвело, до збільшення видатків на утримання ігрових-серверів та відключення голосового чату в країнах, де такі закони діють.

Метою моєї роботи є розробка онлайн-гри на двох та більше гравців, що матиме функції голосового чату та пошуку інших гравців для створення нової ігрової сесії.

1 ФОРМУВАННЯ ВИМОГ ДО ВЕБ-ДОДАТКУ

Веб-додаток, що являє собою гру типу Пінг-Понг потребує реалізації надійної архітектури, що забезпечує стабільний ігровий процес та задовільний рівень якості голосового засобу комунікації між гравцями. Веб-додаток повинен включати клієнтську та серверну частини. Клієнтська частина буде забезпечувати взаємодію користувача, тобто гравця, з ігровим процесом. Серверна частина буде забезпечувати ініціацію ігрової сесії, збереження стану ігрової сесії та можливість вербальної комунікації між гравцями.

Основою для серверної частини обрано JavaScript фреймворк Express JS - популярне рішення для вирішення бізнес задачі створення API, який забезпечує високу швидкість роботи серверної частини веб-додатка [2].

Клієнтська частина буде використовувати ReactJS – JavaScript бібліотеку для створення користувацьких інтерфейсів.

Веб-додаток повинен мати наступні можливості:

- ігровий процес;
- керування початком та кінцем ігрової сесії;
- збереження результатів минулих ігрових сесій;
- таблиця лідерів, яка відображає список найбільш результативних гравців;
- підтримка можливості комунікації голосом між гравцями.

2 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПЗ

Вибір архітектура є дуже важливим етапом розробки ПЗ будь-якого рівня складності. Вибір правильної архітектури забезпечує легшу та швидшу розробку ПЗ та його підтримку в майбутньому.

2.1 UML проектування веб-додатка

В процесі розробки веб-додатка було розроблено use case діаграму для відображення функціональних можливостей веб-додатка. Функціональність застосунку з точки зору користувача представлено на рисунку 2.1.

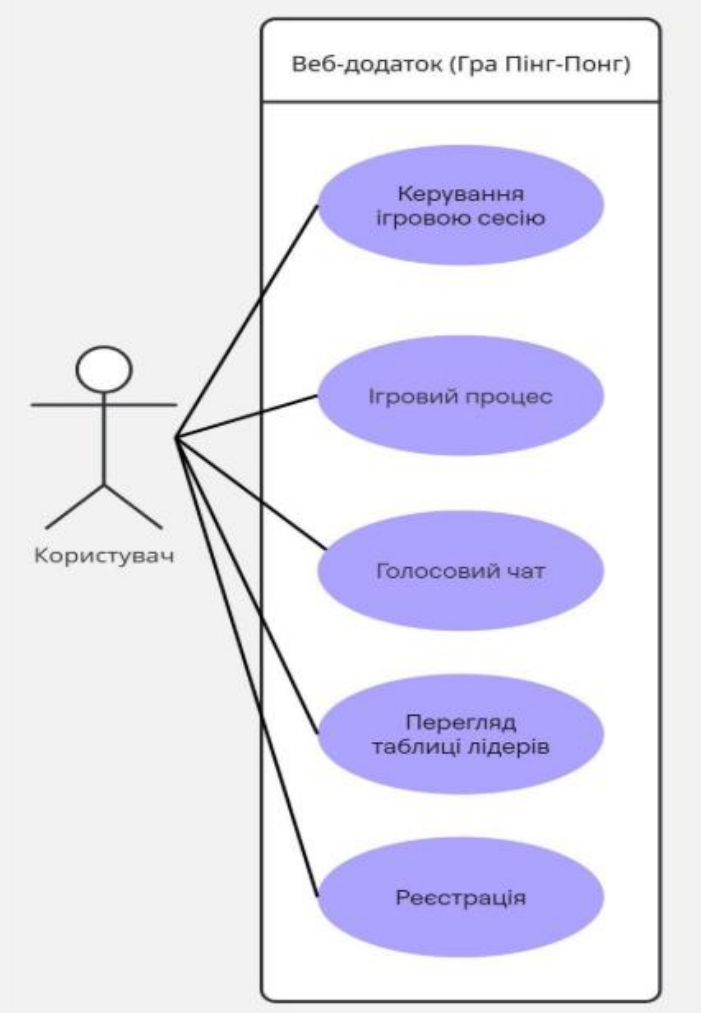


Рисунок 2.1 - Use case діаграма веб-додатка

Користувач (гравець) має такі можливості:

- реєстрація нового аккаунта;
- доступ до ігрового процесу;
- доступ до голосового чату;
- перегляд таблиці лідерів;
- керування ігровою сесією.

2.2 Проектування архітектури веб-додатка

Під час розробки був вибір між двома популярними видами архітектур для веб-додатків: дворівневою та трирівневою архітектурами.

Дворівнева архітектура представляє собою два рівні: клієнтський рівень та рівень бізнес-логіки. З переваг такої архітектури можна виділити її простоту, бо вона не включає рівень доступу до даних на відміну від трирівневої архітектури. З недоліків такої системи можна виділити додаткові складності та ризики під час роботи з базою даних, через відсутність рівня доступу до даних, який відповідає за роботу з базою даних. На рисунку 2.2 представлено схему дворівневої архітектури веб-додатка.

Рисунок 2.2 - Схема дворівневої архітектури веб-додатку

Через недолік двохрівневої архітектури, під час розробки веб-додатка було обрано трьохрівневу архітектуру, яка складається з клієнтської частини, рівень бізнес-логікі та рівня доступу до даних, що працює з базою даних (виконує запити та мутації бази даних). На відміну від попередньої архітектури, ця архітектура позбавлена недоліків при роботі з базою даних. Схема трьохрівневої архітектури представлено на рисунку 2.3.

Рисунок 2.3 - Схема трирівневої архітектури веб-додатку

Клієнтська частина - це те з чим взаємодіє кінцевий користувач, тобто інтерфейс додатку. Для цього рівня обрано технологію ReactJS та мову програмування JavaScript.

React — це безкоштовна бібліотека JavaScript з відкритим вихідним кодом, метою якої є зробити створення користувацьких інтерфейсів на основі компонентів більш «безшовним» [3]. Вона підтримується Meta (раніше Facebook) та спільнотою окремих розробників і компаній. React можна використовувати для розробки односторінкових, мобільних або серверних додатків за допомогою фреймворків, таких як Next.js та Remix. Оскільки React займається лише інтерфейсом користувача та компонентами візуалізації в DOM, додатки React часто покладаються на бібліотеки для маршрутизації та іншої функціональності на стороні клієнта. Ключовою перевагою React є те, що він

повторно візуалізує лише ті частини сторінки, які змінилися, уникаючи непотрібного повторного візуалізації незмінних елементів DOM.

Для покращення зовнішнього вигляду кінцевого інтерфейсу, буде використано CSS-фреймворк Bootstrap [4].

Bootstrap — це бібліотека HTML, CSS та JS, яка зосереджена на спрощенні розробки інформативних веб-сторінок (на відміну від веб-додатків). Основна мета її додавання до веб-проекту — застосувати до цього проєкту вибрані Bootstrap варіанти кольору, розміру, шрифту та макета. Таким чином, основним фактором є те, чи відповідають ці варіанти відповідальним розробникам. Після додавання до проєкту Bootstrap надає базові визначення стилів для всіх елементів HTML . Результатом є однаковий вигляд тексту, таблиць та елементів форм у всіх веббраузерах . Крім того, розробники можуть скористатися перевагами класів CSS, визначених у Bootstrap, для подальшого налаштування зовнішнього вигляду свого вмісту. Наприклад, Bootstrap передбачив світлі та темні таблиці, заголовки сторінок, більш помітні цитати та текст з виділенням.

Bootstrap також постачається з кількома компонентами JavaScript, які не потребують інших бібліотек, таких як jQuery . Вони надають додаткові елементи інтерфейсу користувача, такі як діалогові вікна , підказки , індикатори виконання, випадаючі меню навігації та каруселі. Кожен компонент Bootstrap складається з HTML-структури, CSS-оголошень та, в деяких випадках, супровідного коду JavaScript. Вони також розширюють функціональність деяких існуючих елементів інтерфейсу, включаючи, наприклад, функцію автозаповнення для полів введення.

Рівень бізнес-логікі - це серверна частина додатка, яка відповідає за прийому HTTP-запитів та створення відповіді на них. Основою для рівня було обрано фреймворк ExpressJS. Цей фреймворк має всі необхідні можливості для побудови надійного серверного додатку. Як особливість можна виділити, те що цей фреймворк використовує мову програмування JavaScript, тобто один і той самий розробник може створювати як клієнтську частину так і серверну.

Рівень доступу до даних - це додаткова технологія, що використовується на серверній частині веб-додатку і забезпечує доступ до бази даних та операції з нею. В даному випадку було обрано технологію Prisma [5]. Це ORM (Object-Relational mapping) система, що дозволяє маніпулювати сутностями, які присутні в базі даних (наприклад, записи користувачів), як звичайними об’єктами в мові програмування, що спрощує процес розробки та підтримки проекту.

Її особливістю, є те що розробник може описати структуру бази даних в одну місці, виконати декілька консольних команд і отримати готову до використання базу даних. Структура бази даних описується в файлі schema.prisma в кореневій папці проекту. На рисунку 2.4 продемонстровано приклад структури бази даних, що описана в файлі schema.prisma.

schema.prisma prisma

```
model Product {
  id          String   @id @default(cuid())
  name        String
  description String
  price       Int
  reviews     Review[]
}

model Review {
  id        String   @id @default(cuid())
  text      String
  rating    Int
  Product   Product? @relation(fields: [productId], references: [id])
  productId String?
}
```

Рисунок 2.4 - Приклад schema.prisma файлу, який містить опис структури бази даних

В якості бази даних було обрано MySQL [6], як перевірене часом рішення яке використовується в багатьох компаніях світу для зберігання чутливої інформації. MySQL - це система управління базами даних, що представляє собою програму яка працює в пам’яті сервера на якому вона встановлена і обробляє

TCP-запити через порт 3306. TCP-запити до бази даних представляють собою строчки з SQL-запитами. На рисунку 2.5 зображено приклад такого SQL

В процесі розробку було виділено дві сутності необхідні для функціонування веб-додатка: користувачі та завершені ігрові сесії.

Сутність користувача повинна мати наступні характеристики:

- ідентифікатор (id), який буде використовуватись для унікальності записів користувачів і уникнення появи дуплікатів, а також для індексації бази даних користувачів, що збільшить швидкість запитів вибірки до бази даних;
- ім'я користувача (username);
- електронна пошта користувача (email);
- пароль (password);
- дата створення акаунта користувача (createdAt);
- дата оновлення даних акаунта користувача (updatedAt);
- зв'язок з ігровими сеансами де гравець був під номером 1 (gamesAsPlayer1);
- зв'язок з ігровими сеансами де гравець був під номером 2 (gamesAsPlayer2);

На рисунку 2.5 зображено опис сутності User (користувач) з prisma.schema файлу.

```
model User {
    id          Int      @id @default(autoincrement())
    username    String   @unique
    email       String   @unique
    password    String
    createdAt   DateTime @default(now())
    updatedAt   DateTime @updatedAt

    gamesAsPlayer1 GameResult[] @relation("Player1Games")
    gamesAsPlayer2 GameResult[] @relation("Player2Games")
}
```

Рисунок 2.5 - Структура сутності User

Також представлено UML схему сутності User на рисунку 2.6.

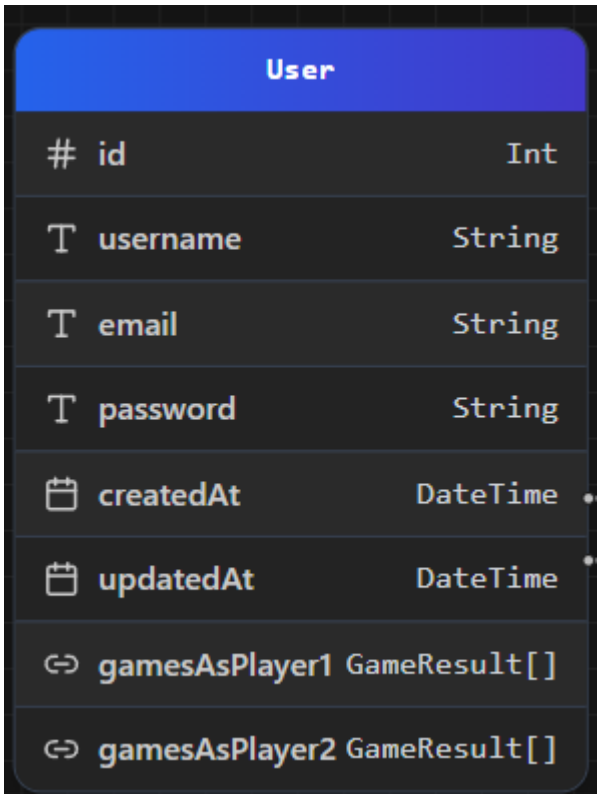


Рисунок 2.6 - UML схема сутності User

Сутність ігрової сесії повинна мати наступні характеристики:

- ідентифікатор (id), який буде використовуватись для унікальності записів ігрових сесій і уникнення появи дуплікатів, а також для індексації бази даних ігрових сесій, що збільшить швидкість запитів вибірки до бази даних;
- ідентифікатор першого гравця (player1Id);
- ідентифікатор другого гравця (player2Id);
- кількість очок першого гравця (player1Score);
- кількість очок другого гравця (player2Score);
- дата ігрової сесії (playedAt);

- ідентифікатор переможця (winnerId);
- зв'язок з гравцями під номером 1 (player1);
- зв'язок з гравцями під номером 2 (player2).

На рисунку 2.7 зображено опис сутності GameResult з prisma.schema файлу.

```
model GameResult {
    id           Int      @id @default(autoincrement())
    player1Id    Int
    player2Id    Int
    player1Score Int
    player2Score Int
    playedAt     DateTime @default(now())
    winnerId     Int?

    player1 User @relation("Player1Games", fields:
    [player1Id], references: [id])
    player2 User @relation("Player2Games", fields:
    [player2Id], references: [id])
}
```

Рисунок 2.7 - Структура сутності GameResult

Також представлено UML схему сутності GameResult на рисунку 2.8.

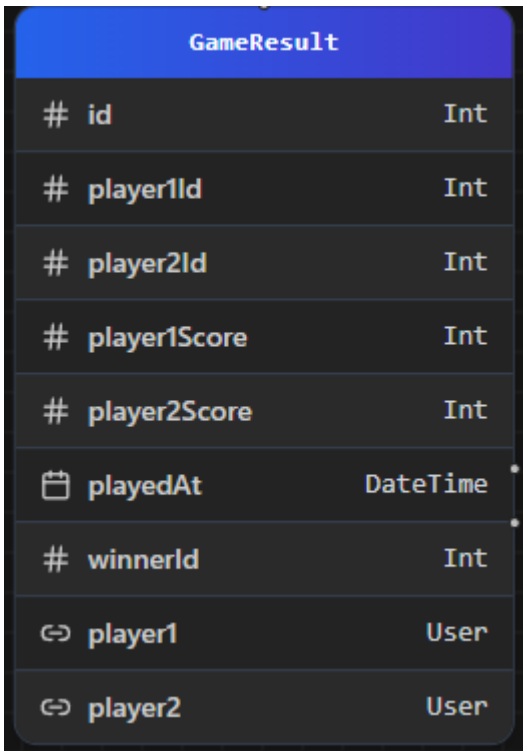


Рисунок 2.8 - UML схема сутності GameResult

На рисунку 2.9 представлено загальну UML схему обох сутностей з бази даних.

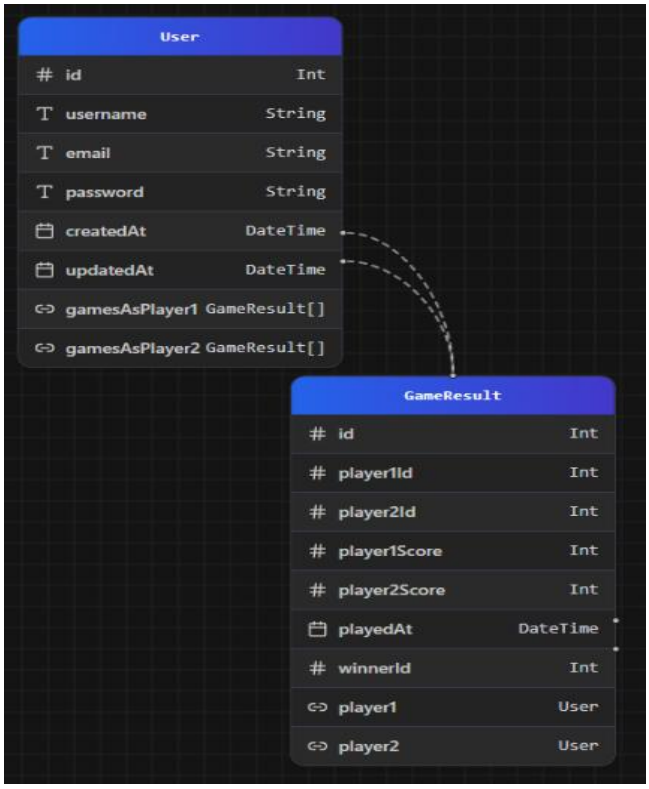


Рисунок 2.9 - Загальна UML схема бази даних

2.3 Проектування голосового чата

Голосовий чат є важливою частиною гри. Для реалізації цієї можливості було обрано технологію Web Real-Time Communication (WebRTC) [7], в перекладі - Веб-комунікація в реальному часі.

WebRTC - це технологія, яка дозволяє веб-додаткам та веб сайтам захоплювати та транслювати аудіо та відео дані, а також обмінюватись цими даними між браузерами без посередників. WebRTC дозволяє обмінюватися даними і проводити телеконференції в режимі peer-to-peer, тобто від користувача до користувача, не вимагаючи встановлення від користувача встановлення плагінів або будь-якого іншого стороннього програмного забезпечення.

WebRTC складається з декількох взаємопов'язаних API (Application programming interface, Інтерфейс програмування застосунків) і протоколів, які працюють разом для досягнення цієї мети.

З особливостей WebRTC можна виділити:

- низьку затримку передачі аудіо та відео;
- зручну інтеграцію з JavaScript веб-додатками.

В процесі встановлення з’єднання між учасниками конференції (пірами), виділяють чотири етапи:

- сигналізація;
- з'єднання;
- безпека;
- комунікація.

Переходи між етапами проходять послідовно, а обов’язковою умовою для переходу до наступного етапу конференції є успішне завершення минулого етапу.

Під час першого етапу, пір не знає з ким буде проходити комунікація. Задача цього етапу - підготовка виклику, для того щоб обидва піра WebRTC змогли розпочати комунікацію.

Сигналізація здійснюється за допомогою існуючого протокола SDP (Session Description Protocol, Протокол опису сесії). SDP - це мережевий протокол, призначений для опису сеансу передачі потокових даних, включаючи телефонію. Повідомлення SDP, що передається від одного піра до іншого можуть вказувати на:

- адреси місця призначення;
- номери UDP портів для відправника та одержувача;
- медіа-формати, які можуть застосовуватися під час сесії;
- час старту і зупинки.

Під час етапу сигналізації піри WebRTC отримують достатньо сигналізації для того, щоб спробувати розпочати з’єднання. Для цього використовується технологія ICE (Interactive Connectivity Establishment, Встановлення інтерактивного з'єднання) [9]. ICE - це метод, що використовується в комп'ютерних мережах для пошуку способів якомога безпосереднього зв'язку двох комп'ютерів один з одним у однорангових мережах. Найчастіше це використовується для інтерактивних медіа, таких як голос через Інтернет-протокол (VoIP), одноранговий зв'язок, відео та обмін миттєвими повідомленнями. У таких програмах зв'язок через центральний сервер був би повільним і дорогим, але прямий зв'язок між клієнтськими програмами в Інтернеті є дуже складним через транслятори мережевих адрес (NAT), брандмауери та інші мережеві бар'єри.

NAT (від англ. Network Address Translation — «перетворення мережевих адрес») — це механізм у мережах TCP/IP, котрий дозволяє змінювати IP-адресу у заголовку пакету, що проходить через пристрій маршрутизації трафіку. Також має назви IP Masquerading, Network Masquerading і Native Address Translation. Уявімо ситуація, коли два піра, що знаходяться у двох різних мережах, намагаються встановити з’єднання між собою: Пір1 з мережі А,з адресою 5.0.0.1, має адресу 192.168.0.1 та Пір2 з мережі Б з мережею 5.0.0.2, має адресу 192.168.0.1. Для забезпечення можливості комунікації треба запобігти до трансляції NAT. Пір1 використовує порт 7000 для встановлення WebRTC

з’єднання з Пір2. 192.168.0.1:7000 прив'язується (bind) до 5.0.0.1.1:7000. Це дає змогу Agent 2 «досягати» Agent 1, надсилаючи пакети за адресою 5.0.0.1.1:7000. Створення відображення NAT схоже на автоматичну версію переадресації портів (port forwarding) у роутері.

Через існування різних типів NAT може виникати проблема під час встановлення з’єднання між пірами. Для вирішення цієї проблеми існує STUN (Session Traversal Utilities for NAT) – це інструмент, який використовується іншими протоколами, такими як Interactive Connectivity Establishment (ICE), Session Initiation Protocol (SIP) та WebRTC . Він надає хостам інструмент для виявлення наявності транслятора мережевих адрес та виявлення зіставленої, зазвичай публічної, IP -адреси та номера порту, які NAT виділив для потоків UDP (User Datagram Protocol) програми до віддалених хостів. Протокол потребує допомоги від стороннього мережевого сервера (STUN-сервера), розташованого на протилежній (публічній) стороні NAT, зазвичай у публічному Інтернеті.

На теперішній час існує багато безоплатних STUN сервері, наприклад, від американської компанії Google. В моїй роботі я буду використовувати STUN сервер Google з адресою stun:stun.l.google.com:13902.

Перший етап ініціації з’єднання на стороні користувача можна побачити на рисунку 2.10. Додаток користувача створює нове WebRTC з’єднання, вказуючи в параметрах з’єднання адресу STUN сервера Google.

```
peerConnection = new RTCPeerConnection({
  iceServers: [
    {
      urls: "stun:stun.l.google.com:13902",
    },
  ],
});
```

Рисунок 2.10 - Ініціації WebRTC з’єднання додатком користувача

Щоб визначити, чи перебуває пір за NAT, і отримати публічну IP-адресу, якщо це можливо, агент ICE надішле запит на вказаний в конфігурації WebRTC сервер STUN. Якщо NAT є, він встановить свою публічну адресу і порт у заголовку повідомлення. Сервер STUN спробує виконати команду ping на цю публічну адресу з різних IP-адрес, щоб перевірити, чи перебуває пір за NAT, і якщо так, то який це тип. Якщо все піде правильно, то сервер STUN поверне адресу і порт.

Не всі NAT реалізовані однаково і можуть відрізнятися в тому, як вони дозволяють пакетам проходити. Деякі реалізації NAT, такі як NAT один до одного, дозволять встановити P2P-з'єднання. Деякі, наприклад симетричні, цього не роблять.

У реалізації NAT один до одного (або Full Cone NAT), щойно внутрішню IP-адресу/порт duo зіставляють із зовнішньою IP-адресою/портом duo, усі пакети, що надходять на зовнішню адресу/порт, незалежно від того, звідки вони надходять, буде надіслано через вихідну внутрішню адресу/порт. Якщо за таким NAT стоять однорангові вузли, то для встановлення P2P-з'єднання достатньо отримати публічну IP-адресу і порт обох однорангових вузлів.

У симетричних NAT'ах зовнішня IP-адреса/порт залежить від внутрішньої IP-адреси/порту duo і призначення. Запити до сервера TURN зіставляються із заданою IP-адресою і портом зовнішнього джерела. Але якщо один і той самий внутрішній хост надсилає пакет в інше місце призначення, наприклад в одноранговий вузол, з яким він намагається зв'язатися, зовнішня IP-адреса/порт відрізнятиметься. Крім того, зовнішній хост повинен отримати пакет від внутрішнього хоста, перш ніж він зможе відправити пакет назад.

Якщо контакти знаходяться за симетричним NAT, вони не зможуть спілкуватися. В такому випадку потрібне інше рішення: TURN.

Traversal Using Relays around NAT (TURN) [10] - це протокол, який допомагає обходити транслятори мережевих адрес (NAT) або брандмауери для мультимедійних додатків. Він може використовуватися з протоколом керування передачею (TCP) і протоколом користувацьких дейтаграм (UDP). Найбільш

корисний для клієнтів у мережах, замаскованих симетричними пристроями NAT. TURN не допомагає запускати сервери на добре відомих портах у приватній мережі через NAT; він підтримує з'єднання користувача за NAT лише з одним одноранговим пристроєм, як, наприклад, у телефонії.

Агент ICE спочатку спробує встановити з'єднання безпосередньо між одноранговими вузлами та переключиться на опцію TURN тільки в тому разі, якщо це не спрацює. Вам не потрібно піклуватися про це самостійно, потрібно тільки слухати подію onicecandidate RTCPeerConnection. Вона спрацьовуватиме щоразу, коли виявляється кандидат ICE. Потім вам потрібно надіслати кандидата своєму контакту через свій сигнальний механізм:

На рисунку 2.11 зображені два обробника подій onmessage та onicecandidate.

```
pc.onicecandidate = (event) => {
  if (event.candidate) {
    signaling.send(event.candidate);
  }
}

signaling.onmessage = async (message) => {
  const data = JSON.parse(message.data);
  if (data) {
    const { message_type, content } = data;
    // ...
    if (message_type === MESSAGE_TYPE.CANDIDATE && content) {
      await pc.addIceCandidate(content);
    }
  }
}
```

Рисунок 2.11 - Обробники подій onmessage та onicecandidate.

Після отримання кандидата від свого контакту, потрібно доставити його агенту ICE, викликавши addIceCandidate. Інша частина переговорів і остаточний відбір кандидатів потім здійснюється агентом ICE. Наприкінці переговорів про

кандидата, у разі успіху, колеги можуть почати спілкування.

2.4 Проектування аутентифікації користувачів додатка

Для забезпечення доступу до додатка, буде реалізована реєстрацію, що вимагає введення надійного пароля, пошти користувача, ім’я користувача. Пошта та ім’я користувача повинні бути унікальними. На рисунку 2.5 зображено схему сутності користувача в базі даних, навпроти полів email та username прописані модифікатори @unique, що вимагають від поля містити унікальне значення, що зроблено для уникання дублікатів записів користувачів.

Під час реєстрації пароль користувача хешується за допомогою JavaScript бібліотеки bcrypt. Бібліотека bcrypt приймає пароль користувача і змінює вигляд в якому він буде зберігатись в базі даних. Це зроблено для безпечного зберігання паролів в базі даних. На рисунку 2.12 зображена спрощена схема роботи bcrypt.

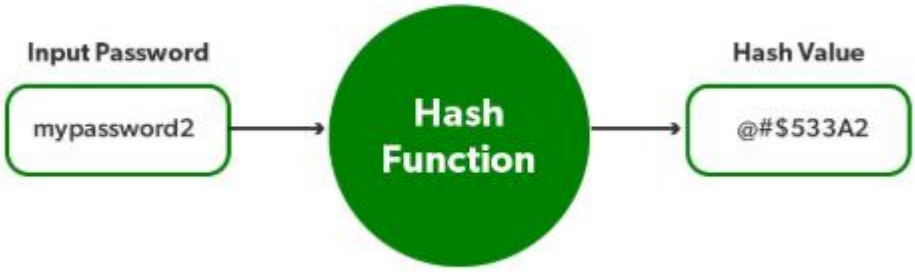


Рисунок 2.12 - Спрощена схема роботи bcrypt.

Після успішного хешування пароля, сервер створює новий запис в базі даних в таблиці Users, а додаток користувача отримує JWT-токен та зберігається як cookie в браузері. Додаток користувача надсилає JWT-токен разом з кожним

запитом до сервера, щоб підтвердити особистість користувача. Також JWT-токен має строк дії обмежений у три години реального часу. Після трьох годин, користувачу треба буде знов увійти до свого облікового запису в веб додатку.

JSON Web Token (JWT) - це запропонований інтернет-стандарт для створення даних з необов'язковим підписом та/або необов'язковим шифруванням, корисне навантаження якого містить JSON, що стверджує певну кількість тверджень. Токени підписуються за допомогою приватного секретного або публічного/приватного ключа [11].

Наприклад, сервер може згенерувати токен із твердженням "увійшов як адміністратор" і надати його клієнту. Клієнт може використовувати цей токен, щоб довести, що він увійшов в систему від імені адміністратора. Токени можуть бути підписані закритим ключем однієї сторони (зазвичай сервера), щоб будь-яка сторона могла згодом перевірити, чи є токен легітимним. Якщо інша сторона володіє відповідним відкритим ключем, вона також може перевірити легітимність токена за допомогою відповідних засобів, що заслуговують на довіру, то вона також може перевірити легітимність токена. Токени розроблені таким чином, щоб бути компактними, безпечними для URL-адрес, і зручними у використанні, особливо в контексті єдиного входу у веб-браузері (SSO). Твердження JWT зазвичай можуть використовуватися для передачі ідентифікаційних даних автентифікованих користувачів між постачальником ідентифікаційних даних і постачальником послуг, або будь-якого іншого типу тверджень, як того вимагають бізнес-процеси. JWT спирається на інші стандарти на основі JSON: JSON Web Signature та JSON Web Encryption.

cookie - це невеликий блок даних, створений веб-сервером під час перегляду користувачем веб-сайту і розміщений на комп'ютері або іншому пристрої користувача за допомогою веб-браузера користувача. Файли cookie розміщуються на пристрої, який використовується для доступу до веб-сайту, і протягом сеансу на пристрої користувача може бути розміщено більше одного файлу cookie.

2.5 Проектування інтерфейсу користувача

Під час розробки було визначено які екрани мають бути в додатку:

- екран аутентифікації користувача;
- екран реєстрації користувача;
- екран пошуку гри;
- екран гри;
- екран налаштувань облікового запису користувача;
- екран з результатами минулих ігор користувача;
- екран з результатами найбільш результативних гравців.

Екран реєстрації повинен містити чотири текстових поля для введення необхідної для реєстрації інформації (пошта, ім’я, пароль та підтвердження пароля) та кнопку відправки даних.

Екран аутентифікації повинен містити два поля для введення інформації для аутентифікації (пошта, пароль) та кнопку відправки даних.

Екран пошуку гри повинен надавати користувачу можливість пошуку гри з випадковим гравцем, можливість приєднатись до конкретного гравця за ідентифікатором гри та можливість почати власну ігрову сесію.

Екран гри повинен надавати користувачу можливість скопіювати ідентифікатор поточної ігрової сесії за яким інший гравець може доєднатись до гри, можливість увімкнути або вимкнути мікрофон.

На екрані налаштувань користувач повинен мати можливість змінювати дані свого облікового запису.

На екрані результатів минулих ігор користувач повинен бачити всі завершені ігри в яких він брав участь.

На екрані з результатами найбільш результативних гравців повинні відображатись всі найбільш результативні гравці по порядку спадання набраних за результатами ігор очок.

3 РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА

Створено екран реєстрації нових користувачів. На рисунку 3.1 зображено екран реєстрації.

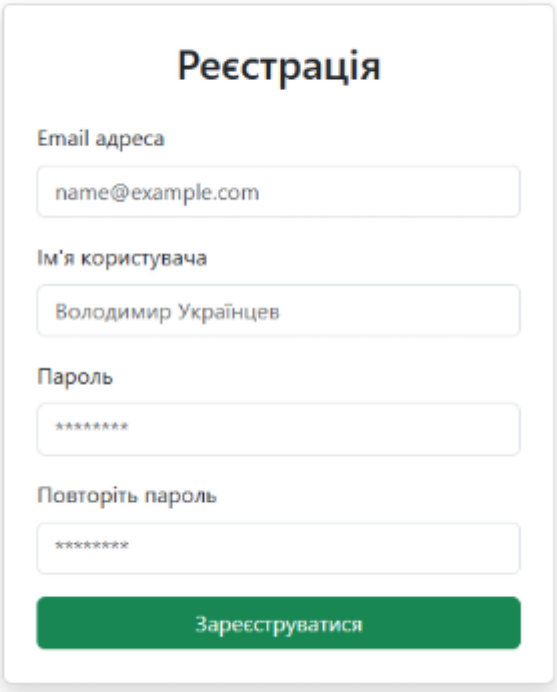


Рисунок 3.1 - Екран реєстрації нових користувачів

Було додано валідацію текстових полів на випадок якщо користувач увів дані в неправильному форматі, а також вивід помилок при спробі створення нового запису користувача на сервері. На рисунку 3.2 зображено екран користувача з активованими повідомленнями валідації.

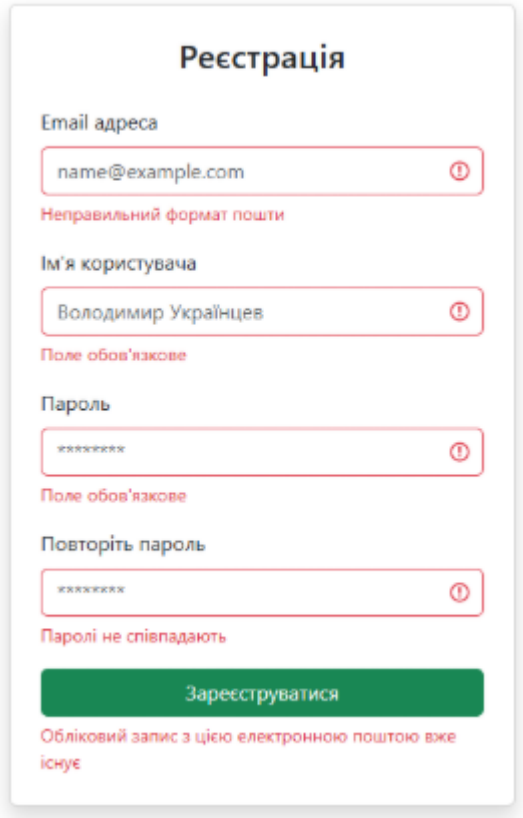


Рисунок 3.2 - Екран реєстрації нових користувачів

На головному екрані до якого користувач переходить після реєстрації містяться кнопки пошуку гри і переходу до топ гравців, а в правому верхньому куті при натисканні на ім’я користувача, з’являється випадний список. На рисунку 3.3 зображено головний екран.

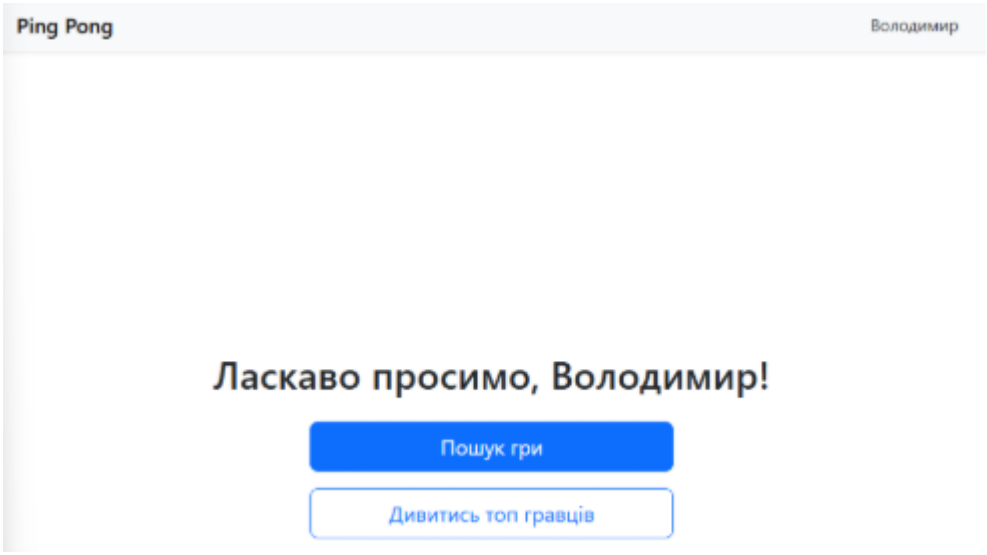


Рисунок 3.3 - Головний екран

На екрані пошуку гри користувач може перейти до пошуку випадкової гри або приєднатись до існуючої гри за ідентифікатором. На рисунку 3.4 зображено екран пошуку гри.

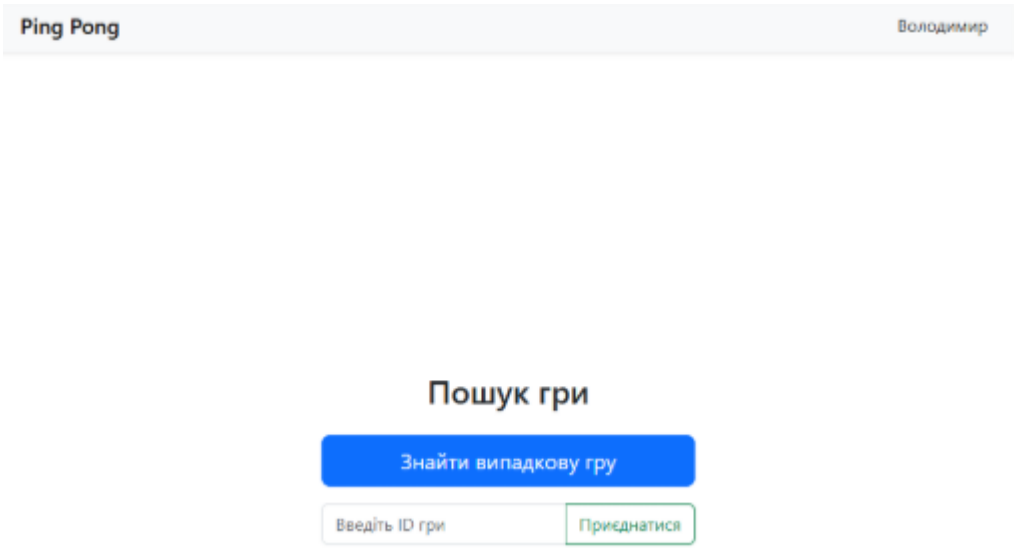


Рисунок 3.4 - Екран пошуку гри

На екрані налаштувань користувач може оновити дані свого облікового запису. На рисунку 3.5 зображено екран налаштувань.

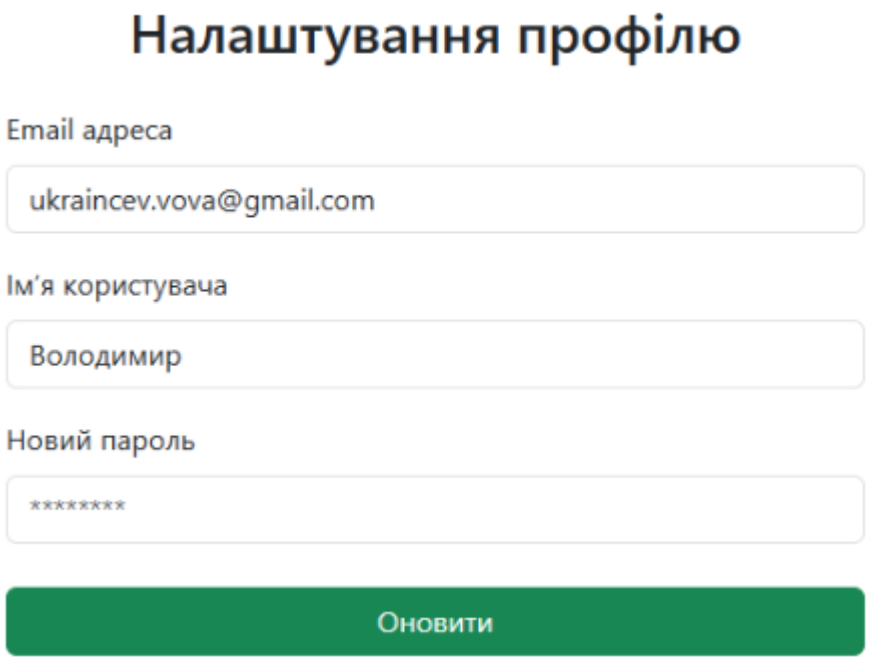


Рисунок 3.5 - Екран налаштувань

На екрані гри є ігрове поле, рахунок гри, і кнопка увімкнення мікрофона. На рисунку 3.6 зображено ігри.

Рисунок 3.6 - Екран ігрового поля

На екрані минулих ігор показані результати минулих ігор. На рисунок 3.7 зображено цей екран.

Рисунок 3.7 - Екран минулих ігор

На екрані екрану з результатами найбільш результативних гравців. На рисунок 3.8 зображено екран.

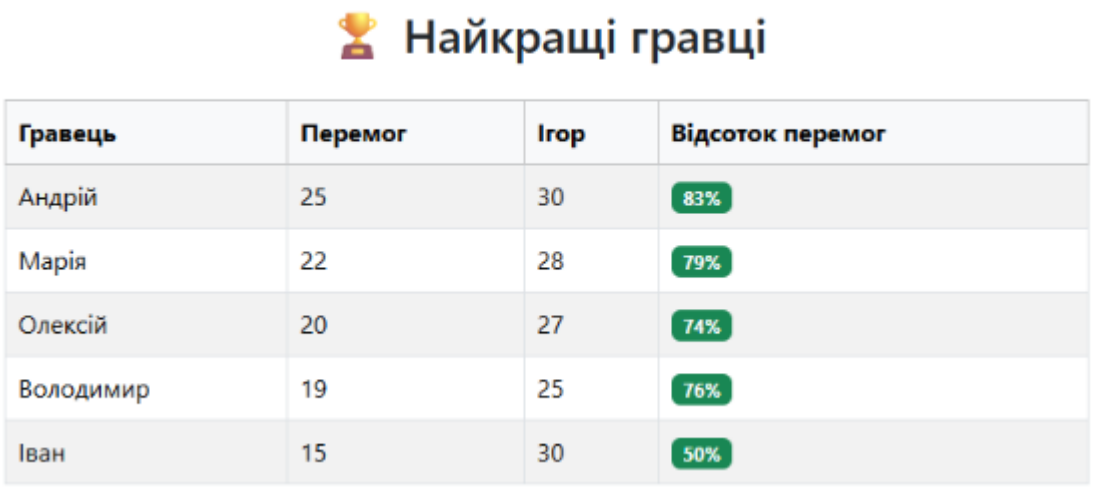


Рисунок 3.8 - Екран з результатами найбільш результативних гравців

4 РОЗГОРТАННЯ ПРОЕКТА В РОБОЧОМУ СЕРЕДОВИЩІ ЗА ДОПОМОГОЮ DOCKER

Для забезпечення стабільної роботи проекта і зручного та швидкого розгортання в робочому середовищі буде використано Docker.

Docker — це програмне забезпечення, яке дає можливість на певній ділянці пам'яті ізольовано встановити необхідну ОС (операційну систему), версію Java, налаштувати змінні оточення, встановити різні залежності і дати доступ тільки за певних умов. При цьому дану програму абсолютно не буде хвилювати, що відбувається навколо.

Для функціонування Docker потрібно лише встановити Docker Engine на хості, який буде запускати готові образи додатків. Також Docker пропонує безкоштовний реєстр образів, який містить таке популярне ПЗ як: Apache, Java, NodeJS. Docker також пропонує приватний реєстр, який можна хостити на власній машині.

Спрощену схему роботи ПЗ Docker наведено на рисунку 4.1.

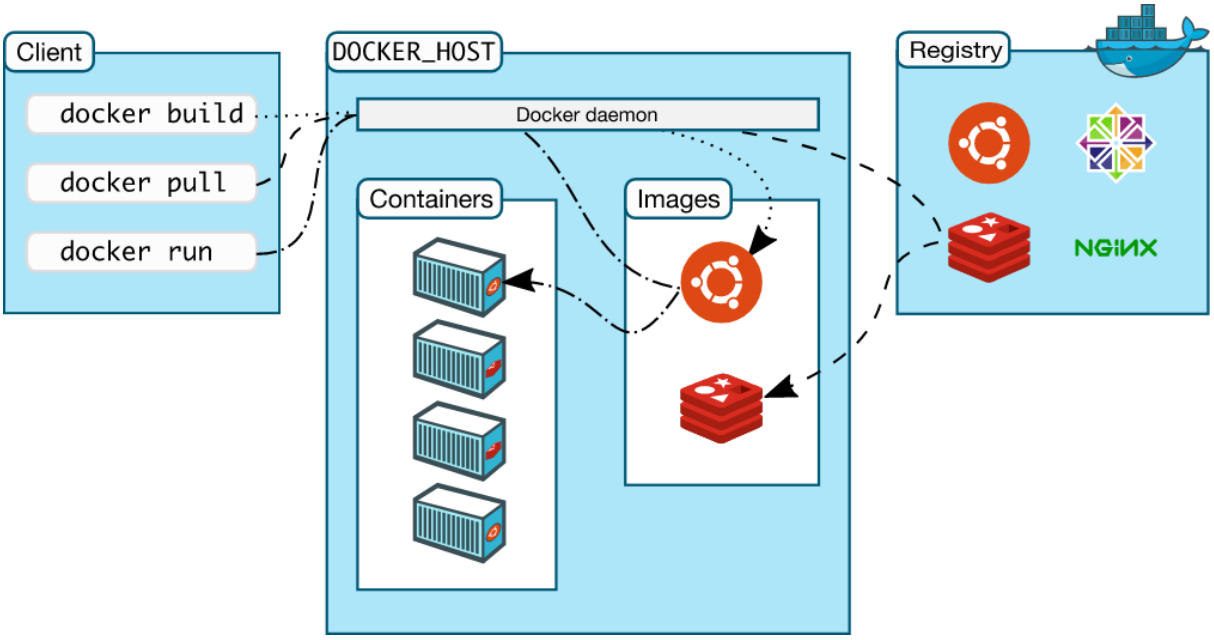


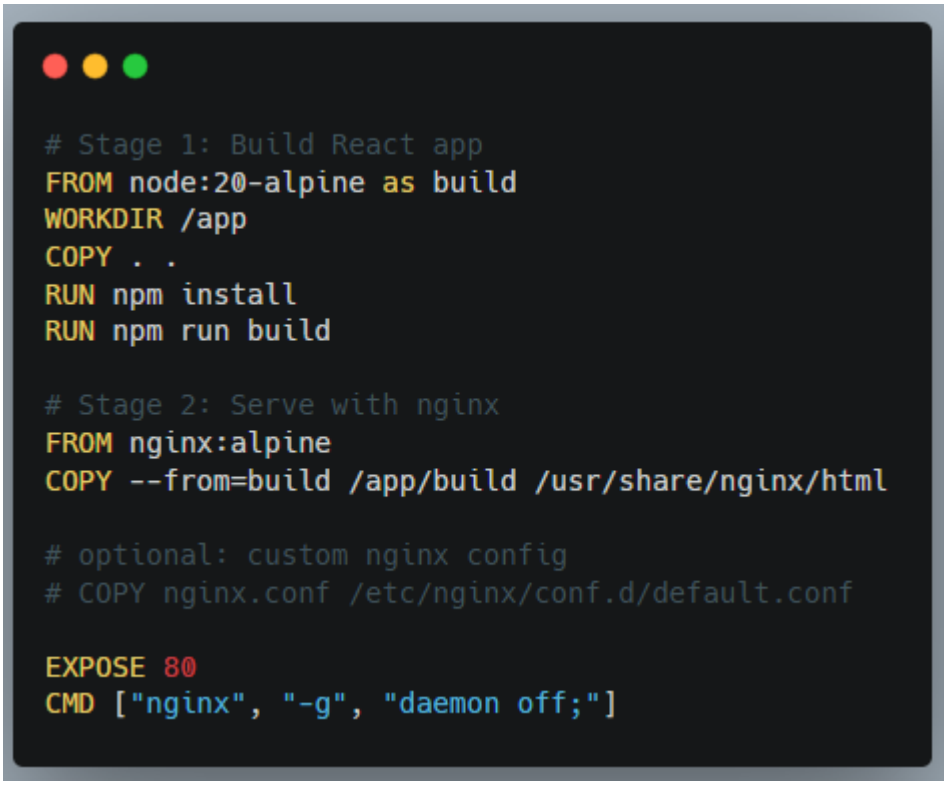
Рисунок 4.1 – спрощена схема роботи ПЗ Docker

Порядок розгортання додатка буде описано в docker-compose файлі, який буде посилатися на Dockerfile, що описують кожний з сервісів (сервер, клієнт, база даних). На рисунку 4.2 зображено docker compose файл.

```yaml
version: '3.8'

services:
  mysql:
    image: mysql:8
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: app_db
      MYSQL_USER: app_user
      MYSQL_PASSWORD: app_pass
    volumes:
      - mysql-data:/var/lib/mysql
    ports:
      - "3306:3306"

  backend:
    build: ./server
    environment:
      DB_HOST: mysql
      DB_USER: app_user
      DB_PASSWORD: app_pass
      DB_NAME: app_db
    ports:
      - "5000:5000"
    depends_on:
      - mysql

  frontend:
    build: ./client
    ports:
      - "3000:80"
    depends_on:
      - backend

volumes:
  mysql-data:
```

Рисунок 4.2 – Docker compose файл.

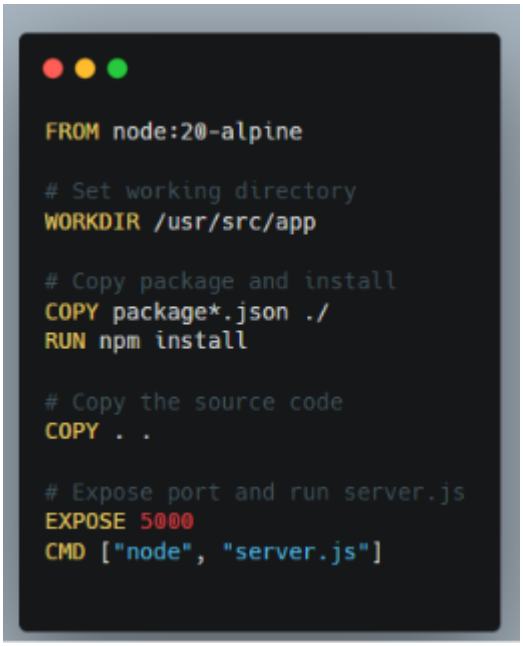
Було створено Dockerfile для клієнта. На рисунку 4.3 зображено Dockerfile для клієнта.



```dockerfile
# Stage 1: Build React app
FROM node:20-alpine as build
WORKDIR /app
COPY . .
RUN npm install
RUN npm run build

# Stage 2: Serve with nginx
FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html

# optional: custom nginx config
# COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Рисунок 4.3 - Dockerfile клієнта

Було створено Dockerfile для сервера. На рисунку 4.4 зображено Dockerfile для сервера.



```dockerfile
FROM node:20-alpine

# Set working directory
WORKDIR /usr/src/app

# Copy package and install
COPY package*.json ./
RUN npm install

# Copy the source code
COPY . .

# Expose port and run server.js
EXPOSE 5000
CMD ["node", "server.js"]
```

Рисунок 4.4 - Dockerfile сервера

Для розгортання через Docker, необхідно встановити ПЗ в систему, передати на сервер Docker compose і Dockerfile і виконати команду docker compose up –d.

5 ТЕСТУВАННЯ ДОДАТКУ

Тестування в JavaScript є важливою практикою, яка гарантує, що ваша кодова база залишається надійною, легко підтримуваною і не містить помилок. Важливо виявляти проблеми на ранніх стадіях розробки, а тестування - один з найефективніших способів досягти цього.

5.1 Тестування клієнтської частини

Мета тестування клієнтської частини полягає в тому щоб бути впевненим, що користувач побачить веб додаток таким, яким його задумував розробник. Тестування буде здійснюватись за допомогою технології React Testing Library.

React Testing Library - це дуже легке рішення для тестування React-компонентів. Вона надає легкі утиліти поверх react-dom та react-dom/test-utils, таким чином заохочуючи кращі практики тестування. Її основним керівним принципом є:

Чим більше ваші тести схожі на те, як використовується ваше програмне забезпечення, тим більше впевненості вони можуть вам дати.

Таким чином, замість того, щоб мати справу з екземплярами відрендерених React-компонентів, тести будуть працювати з реальними DOM-вузлами. Утиліти, які надає ця бібліотека, полегшують запити до DOM так само, як це робив би користувач. Пошук елементів форми за текстом мітки (так само, як це зробив би користувач), пошук посилань і кнопок за текстом (так само, як це зробив би користувач). Вона також розкриває рекомендований спосіб пошуку елементів за тестом даних як «аварійний люк» для елементів, де текстовий вміст і мітка не мають сенсу або не є практичними.

Ця бібліотека заохочує програми бути більш доступними і дозволяє наблизити тести до використання компонентів так, як це буде робити користувач, що дає вам більше впевненості в тому, що програма буде працювати, коли її буде використовувати реальний користувач.

На рисунку 5.1 продемонстровано код для тестування сторінки

авторизації користувача. Тести інших компонентів користувацької частини додатка схожі за своєю суттю.

```
import { render, screen, fireEvent } from "@testing-library/react";
import Login from "./Login";

describe("Login component", () => {
  it("allows user to input email and password and submit the form", () => {

    console.log = jest.fn();

    render(<Login />);

    const emailInput = screen.getByLabelText(/Email адреса/i);
    const passwordInput = screen.getByLabelText(/Пароль/i);
    const submitButton = screen.getByRole("button", { name: /увійти/i });

    fireEvent.change(emailInput, { target: { value: "test@example.com" } });
    fireEvent.change(passwordInput, { target: { value: "123456" } });
    fireEvent.click(submitButton);

    expect(console.log).toHaveBeenCalledWith("Email:", "test@example.com");
    expect(console.log).toHaveBeenCalledWith("Password:", "123456");
  });
});
```

Рисунок 5.1 – Тест сторінки авторизації

5.2 Тестування серверної частини

Тестування серверної частини буде проходити за допомогою бібліотеки Supertest.

Supertest — це популярний інструмент для тестування HTTP API, який дає змогу розробникам легко здійснювати запити до вебсервісів і перевіряти їхні відповіді. Він орієнтований на інтеграційне тестування вебсервісів, що є ключовим компонентом для перевірки правильної взаємодії між клієнтської та серверною частинами, а також внутрішніх сервісів.

На рисунку 5.2 зображено код для тестування серверної частини додатка.

```
const request = require('supertest');
const WebSocket = require('ws');
const { server } = require('./server');

describe('Express and WebSocket Server', () => {
  let httpServer;

  beforeAll((done) => {
    httpServer = server.listen(0, () => done());
  });

  afterAll((done) => {
    httpServer.close(() => done());
  });


  test('WebSocket should echo message', (done) => {
    const address = httpServer.address();
    const url = `ws://localhost:${address.port}`;
    const ws = new WebSocket(url);

    ws.on('open', () => {
      ws.send('Hello');
    });

    ws.on('message', (data) => {
      expect(data).toBe('Echo: Hello');
      ws.close();
      done();
    });
  });
});
```

Рисунок 5.2 – Серверний тест

ВИСНОВКИ

У результаті виконання роботи було проаналізовано вибір архітектури додатка, недоліки та переваги кожної з архітектур. Проведено аналіз роботи протокола для голосового чату, аналіз необхідних веб-сторінок, список необхідних для розробки і тестування технологій. Розроблено інтерфейс користувача і серверну частину. Можна зробити наступні висновки:

1. Для стабільної роботи додатку потрібно визначитись із правильної архітектурою.
2. Для проведення голосового чату посередники не потрібні, але для ініціації сесії потрібне посередник в вигляді сервера ICE.
3. Важливо проводите тестування як клієнтської так і серверної частин додатка.

ПЕРЕЛІК ПОСИЛАНЬ

1. Fabio D. How Many Gamers Are There? (New 2025 Statistics). Exploding Topics. 05.06.2025. URL: https://explodingtopics.com/blog/number-of-gamers.

2. Express - Node.js web application framework. https://expressjs.com/. URL: https://expressjs.com/.

3. React. https://react.dev/. URL: https://react.dev/.

4. Bootstrap · The most popular HTML, CSS, and JS library in the world.. URL: https://getbootstrap.com/.

5. Prisma | Instant Postgres plus an ORM for simpler db workflows. URL: https://www.prisma.io/.

6. MySQL. URL: https://www.mysql.com/.

7. RTCPeerConnection - Web APIs | MDN. URL: https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection.

8. Session Description Protocol - Wikipedia. Wikipedia. URL: https://en.wikipedia.org/wiki/Session_Description_Protocol.

9. Interactive Connectivity Establishment - Wikipedia. Wikipedia. URL: https://en.wikipedia.org/wiki/Interactive_Connectivity_Establishment.

10. Wikipedia. URL: https://en.wikipedia.org/wiki/Traversal_Using_Relays_around_NAT.

11. JSON Web Token. Wikipedia. URL: https://en.wikipedia.org/wiki/JSON_Web_Tok