МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО – НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні

ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Програмне забезпечення комп'ютерної рольової гри з використанням React та Node.js**

Виконав: студент 4 курсу, групи 6.1210-пзс
Спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
А.А. Хлівний
(ініціали та прізвище)
Керівник доцент, А.І. Безверхий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дискус» Р.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____перший (бакалавський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Програмне забезпечення систем_____
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри _____Тетяна КРИТСЬКА
« 01 » березня 2024 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Хлівному Андрію Анатолійовичу
(прізвище, ім'я, по батькові)

1.Тема роботи Програмне забезпечення комп'ютерної рольової гри з використанням React та Node.js
керівник роботи _____Безверхий Анатолій Ігорович, доцент_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 26.12.2023 № 2215-с
2. Строк подання студентом кваліфікаційної роботи ____07.06.2024_____
3. Вихідні дані кваліфікаційної роботи бакалавра
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково – пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- аналіз сучасних технологій для створення клієнтської частини Веб застосунків;
- проєктування Веб-застосунку;
- програмна реалізація застосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
14 Слайдів презентації
6. Консультанти розділів бакалаврської роботи

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

7. Дата видачі завдання 01.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	02.05-04.05.24	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	04.05-05.05.24	виконано
3	Аналіз існуючих аналогів	05.05-06.05.24	виконано
4	Розробка структури бази даних для зберігання даних	06.05-07.05.24	виконано
5	Узгодження подальших дій з науковим керівником	07.05-08.05.24	виконано
6	Налаштування стартових CRUD ендпоінтів для роботи з таблицями бази даних	08.05-12.05.24	виконано
7	Розробка API для роботи фронтенду	12.05-16.05.24	виконано
8	Розбиття бекенду на окремі модулі	16.05-19.05.24	виконано
9	Створення дизайну майбутнього веб застосунку	19.05-22.05.24	виконано
10	Розробка стартового проекту та налагодження взаємодії з даними через API	22.05-29.06.24	виконано
11	Реалізація функціональності фронтенду та верстка всіх необхідних сторінок	29.06-06.06.24	виконано
12	Остаточна перевірка працездатності всіх складових	06.06-07.06.24	виконано
13	Створення презентації	07.06-08.06.24	виконано
14	Оформлення звіту	08.06-16.06.24	виконано

Студент _____ А.А. Хлівний
(підпис) (прізвище та ініціали)

Керівник роботи _____ А.І. Безверхий
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок — 85

Рисунків — 28

Джерел — 12

Хлівний А.А. Програмне забезпечення комп’ютерної рольової гри з використанням React та Node.js: кваліфікаційна робота бакалавра спеціальності 121 «Програмне забезпечення систем» / наук. керівник А. І. Безверхий. Запоріжжя : ЗНУ, 2024. 85 с.

Кваліфікаційна робота присвячена розробці програмного забезпечення для комп'ютерної рольової гри «Мафія» з використанням сучасних веб-технологій React та Node.js. Метою даної роботи є створення функціонального та інтерактивного застосунку, здатного забезпечити зручний та захоплюючий ігровий процес для користувачів. В рамках проекту проведено аналіз вимог до гри «Мафія», визначено основні функціональні компоненти та архітектуру програми. Використання React для фронтенду дозволило створити динамічний інтерфейс користувача, що забезпечує високу чуйність і інтерактивність. Node.js був обраний для серверної частини програми завдяки його продуктивності та здатності обробляти велику кількість запитів у реальному часі. У роботі описані основні етапи розробки: проектування архітектури, реалізація клієнтської та серверної частин, інтеграція з базою даних, а також тестування та налагодження програми. Результатом виконання цієї роботи є повноцінний веб-додаток, який дозволяє користувачам створювати та брати участь в ігрових сесіях «Мафії», взаємодіяти один з одним у режимі реального часу та отримувати задоволення від ігрового процесу. Додаток пройшов тестування та показав стабільну роботу в різних умовах.

Ключові слова: «*Мафія», комп'ютерна рольова гра, React, Node.js,* NestJS *веб-розробка, веб-додаток, інтерактивний інтерфейс, реальний час.*

ABSTRACT

Pages — 85

Drawings — 28

Sources — 12

Khlivnyi A.A. Software of a computer role-playing game using React and Node.js: qualification work of a bachelor of specialty 121 «Software of systems» / Science. manager A. I. Bezverkhy. Zaporizhzhia: ZNU, 2024. 85 p.

The thesis is devoted to the development of software for the computer role-playing game «Mafia» using modern React and Node.js web technologies. The purpose of this work is to create a functional and interactive application capable of providing a convenient and exciting gameplay for users.As part of the project, the requirements for the game «Mafia» were analyzed, the main functional components and the architecture of the program were determined. Using React for the frontend made it possible to create a dynamic user interface that provides high responsiveness and interactivity. Node.js was chosen for the backend of the application due to its performance and ability to handle large number of requests in real time. The work describes the main stages of development: architecture design, implementation of the client and server parts, integration with the database, as well as testing and debugging of the program. Special attention is paid to issues of security and protection of user data, which is a critical aspect for multiplayer games. The result of this work is a full-fledged web application that allows users to create and participate in Mafia game sessions, interact with each other in real time and enjoy the gameplay. The application has been tested and has shown stable performance in various conditions.

Keywords: «Mafia», computer role-playing game, React, Node.js, NestJS, web development, web application, interactive interface, real time.

ЗМІСТ

ВСТУП 8
1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ 11
1.1 Правила та особливості рольової гри «Мафія» 11
1.2 Загальний опис веб-додатків 16
1.3 Аналіз існуючих аналогів веб-додатку гри «Мафія» 20
1.4 Висновки першого розділу 22
2 ДОСЛІДЖЕННЯ ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ 24
2.1 Аналіз методів рендерингу веб-додатків 24
2.1.1 Client-Side Rendering (CSR) 24
2.1.2 Server-Side Rendering (SSR) 26
2.1.3 Static Site Generation (SSG) 27
2.1.4 Incremental Static Regeneration (ISR) 29
2.2 Аналіз сучасних технологій для створення клієнтської частини 30
2.2.1 React 30
2.2.2 Vue.js 33
2.2.3 Angular 35
2.3 Аналіз сучасних технологій для створення серверної частини 37
2.3.1 NestJS 37
2.3.2 Django 39
3 РОЗРОБКА КОМП'ЮТЕРНОЇ РОЛЬОВОЇ ГРИ 44
3.1 Архітектура веб-додатку 44
3.2 Проектування 47
3.2.1 Проектування бази даних 47
3.2.2 Діаграми використання (Use-Case) 49
3.3 Програмна реалізація серверної частини 52
3.3.1 Огляд серверної архітектури 52

3.3.2 Реалізація модуля IAM 54
3.3.3 Реалізація модуля Users 56
3.3.4 Реалізація модуля Players 56
3.3.4 Реалізація модуля Lobby 57
3.3.5 Реалізація модуля Game 58
3.4 Програмна реалізація клієнтської частини 60
3.4.1 Огляд клієнтської архітектури 60
3.4.2 Реалізація компонентів React 61
3.4.3 Використання Redux та Redux-Saga для управлянні станом 70
3.4.4 Реалізація кастомних хуків для забезпечення логіки гри 71
ВИСНОВКИ 82
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 84

ВСТУП

Актуальність теми

Зараз людям потрібне спілкування більше, ніж будь-коли. І ігри стали новим способом комунікації один з одним навіть на різних кінцях світу. Після пандемії Covid-19 більше третини геймерів по всьому світу почали грати частіше, щоб проводити більше часу зі своєю родиною та друзями. Гра «Мафія» є популярною інтелектуально-психологічною грою на вечірках у всьому світі.

Гра «Мафія» - це стратегічна рольова гра в жанрі детективу, що вимагає інтелектуального та психологічного аналізу. У ній відбувається протистояння між організованою меншістю (мафією) та неорганізованою більшістю (мирними мешканцями міста). У сюжеті жителі міста, втомлені беззаконням мафії, вирішують ув'язнити всіх злочинців; у відповідь мафія починає війну, спрямовану на повне знищення мирного населення [17]. Наразі існує чимало онлайн варіацій цієї гри, у жанрі RPG (role play game), пригод та інше, але немає безкоштовної версії з відео чатом та автоматизованим процесом гри.

У 2023 році існує приблизно 2 мільярди веб-сайтів, з яких щонайменше 13,441,975 використовують React.js для створення своїх інтерфейсів користувача [19]. Тому клієнтська частина гри реалізована за допомогою одного з найпопулярніших JavaScript-фреймворків – React.js, розробленого компанією Facebook. Він був розроблений для створення компонентів веб-інтерфейсу і використовується великими компаніями, такими як Netflix, Airbnb і Instagram. Основною перевагою React.js є віртуальний DOM (Document Object Model), який дозволяє більш ефективно виконувати зміни та оновлення веб-сторінок. React.js також пропонує компонентний підхід, що спрощує повторне використання коду та загальне керування проектом [18].

Мета дослідження

Розробка програмного забезпечення комп’ютерної рольової гри з використанням react та node.js Розробка та реалізація веб-додатку гри «Мафія». Цей додаток призначений для автоматизації процесу проведення гри «Мафія». Головною метою розробки є створення зручного та ефективного інструменту для гри «Мафія», який забезпечить правильний хід гри онлайн.

Завдання дослідження

- Реалізувати клієнтську частину гри з використанням React.js для забезпечення ефективної роботи інтерфейсу користувача та спрощення розробки.
- Створити швидкий, масштабований та ефективний сервер з використанням сучасних підходів до програмування на базі NestJS.
- Забезпечити взаємодію між гравцями у реальному часі за допомогою бібліотеки Socket.IO.

Об’єкт дослідження

Об'єктом дослідження є технології розробки комп’ютерної рольової гри з використанням react та node.js

Предмет дослідження

Предметом дослідження є процес проведення гри «Мафія», взаємодія між гравцями та розробки програмного забезпечення комп’ютерної рольової гри.

Методи дослідження

Аналіз літератури та існуючих рішень:

Вивчення наукових статей, книг та інших джерел інформації, пов'язаних із розробкою ігор, використанням React та Node.js, а також архітектурою веб-додатків. Аналіз існуючих аналогічних додатків та ігор, виявлення їх переваг та недоліків.

Проектування та моделювання:

Розробка архітектури програми, включаючи клієнтську і серверну частини. Створення моделей даних та взаємодії між компонентами системи.

Програмування та розробка:

Реалізація клієнтської частини програми за допомогою React.

Розробка серверної частини на Node.js. Інтеграція клієнтської та серверної частин, забезпечення їхньої коректної взаємодії.

Тестування:

Юніт-тестування окремих компонентів програми для перевірки їхньої коректної роботи. Інтеграційне тестування з метою оцінки взаємодії між різними частинами системи. Тестування користувача із залученням реальних користувачів для отримання зворотного зв'язку та виявлення можливих проблем.

Ці методи допоможуть систематично та послідовно провести дослідження, розробку та оцінку програмного забезпечення для гри «Мафія» на базі React та Node.j

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Правила та особливості рольової гри «Мафія»

Ігри стали новим способом комунікації між людьми навіть на різних кінцях світу. Під час пандемії Covid-19 світ переосмислив важливість спілкування один з одним, у цей складний період онлайн ігри стали помічником та подарували можливість багатьом людям проводити більше часу зі своєю родиною та друзями навіть на великих відстанях. «Мафія» не стала винятком.

Гра «Мафія» - це стратегічна рольова гра в жанрі детективу, що вимагає інтелектуального та психологічного аналізу. У ній відбувається протистояння між організованою меншістю (мафією) та неорганізованою більшістю (мирними мешканцями міста). У сюжеті жителі міста, втомлені беззаконням мафії, вирішують ув'язнити всіх злочинців; у відповідь мафія починає війну, спрямовану на повне знищення мирного населення [1]. Наразі існує чимало онлайн варіацій цієї гри, у жанрі RPG (role play game), пригод та інше, але немає безкоштовної версії з відео чатом та автоматизованим процесом гри

У сьогоденні «Мафія» стала всесвітньо відомою грою, яку знають і чули практично всі.

Історія створення: Прототипом гри є європейська гра «Кілер», де замість команди мафії грає лише одна людина. Однак з появою «мафії», де замість однієї людини грає ціла команда, ця гра швидко змінила напрям. Основні правила і основна ідея гри були створені в 1986 році, і гра набула популярності серед студентів. Зростання популярності «Мафії» тривало протягом 21-го століття, коли в багатьох країнах почали виникати спортивні клуби та клуби любителів гри. Люди все частіше грали в «Мафію» і розповідали про це іншим.

Загальні правила гри:
Кількість гравців: від 4 до 10 чоловік. Якщо менше 4 — гра стає менш цікава,

більше 10 — гра стає ускладненою через збільшення кількості обговорень і стає складно керованою.

Початок гри:

Ніч Долі: Розподіл Ролей

Вибір ролей здійснюється за допомогою грального карт, кожна з яких позначає певну роль у грі. Перед початком гри із загального комплекту карток вибирається необхідна кількість, що дорівнює кількості учасників, і формується ігрова колода. Учасники займають будь-які місця за столом. Ведучий тасує колоду і просить усіх гравців заплющити очі, позначаючи початок фази під назвою «Ніч Долі», коли всі мешканці міста «засинають» Підходячи до кожного гравця по черзі, ведучий просить розплющити очі і випадково витягнути карту з колоди, що визначає статус гравця згідно з обраною картою.

День спокою

Ведучий просить усіх розплющити очі, оголошуючи початок «Дня Спокою». Гравці починають «знайомство» один з одним, представляючись мешканцями міста. Кожен учасник дивиться свою карту, щоб дізнатися про свою ігрову роль. Демонструвати картку іншим гравцям суворо забороняється. Протягом цього дня обговорення не допускаються.

Ніч Знайомства

Після завершення знайомства ведучий оголошує «Ніч Знайомства» і всі гравці заплющують очі. Ведучий повідомляє: «Мафія прокинулася та знайомиться». На цьому етапі мафіозі розплющують очі і впізнають один одного, визначаючи приналежність до клану «мафії». Це слід робити дуже обережно, щоб уникнути підозр з боку інших гравців. Потім ведучий оголошує: «Мафія познайомилася і заснула», і мафіозі знову заплющують очі.

Основний ігровий цикл:

День

Ведучий оголошує: «Настав день. Усі мешканці міста прокинулися». Всі гравці розплющують очі, і починається обговорення. Учасники мають таку інформацію: поведінку при роздачі карт, висловлювання напередодні, рухи та дії протягом ночі, а також зміни настрою та поведінки інших гравців. На підставі цих спостережень гравці можуть робити висновки та припущення. Допускається використання будь-яких аргументів, як правдивих, і помилкових, для обґрунтування своїх висновків.

Ведучий пропонує висунути підозрюваних у причетності до «мафії». Висуваються кандидатури, після чого кожному «кандидату» надається останнє слово, яке може бути вирішальним під час голосування. Проводиться голосування. Гравець, визнаний винним у результаті голосування, відкриває та показує свою карту всім та залишає ігрове поле.

Ніч

Після того, як «убитий» гравець залишив ігрове поле, ведучий оголошує: «Настала ніч. Усі мешканці міста заснули», і всі гравці заплющують очі. Вночі ведучий по черзі будить спеціальних персонажів гри, і вони таємно повідомляють свої дії.

- Хід «мафії»: Ведучий: «Мафія прокинулася і обирає свою жертву». «Мафія» розплющує очі і домовляється знаками, кого треба «вбити». Після того, як мафіозі вказують на жертву ведучому, він оголошує: «Мафія зробила свій вибір і заснула», і мафіозі заплющують очі.
- Хід «шерифа»: Ведучий оголошує: «Шериф виходить на полювання». «Шериф» розплющує очі і вказує на гравця, якого він підозрює в причетності до «мафії». Ведучий підтверджує дію і оголошує: «Шериф зробив постріл і заснув», і «шериф» заплющує очі.
- Хід комісара: Ведучий оголошує: «На розслідування виходить комісар». Комісар розплющує очі та вказує на одного з гравців. Ведучий показує

палець вгору, якщо гравець чесний, і вниз, якщо це мафія. Після цього ведучий оголошує: «Комісар провів розслідування та заснув», і комісар заплющує очі.

- Хід лікаря: Ведучий оголошує: «Доктор виходить, щоб допомогти». «Доктор» відкриває очі і вказує на будь-якого гравця. Після цього ведучий оголошує: «Доктор надав допомогу і заснув», і «лікар» заплющує очі.

Ранкові новини

Ведучий оголошує: «Настав день. Усі мешканці міста прокинулися», і всі гравці розплющують очі. Потім ведучий розповідає про події, що відбулися вночі, оформляючи мову у стилістиці гри. Далі ігровий процес повторюється, але з урахуванням нових аргументів та інформації, що з'явилися у гравців. Важливо запам'ятовувати, хто кого висував та хто за кого голосував.

Особливості Проведення Голосування

При голосуванні гравці, які бажають віддати свій голос за підозрюваного у причетності до «мафії», мають підняти руку і тримати її доти, доки ведучий дорахує до трьох, назве кількість голосів та запропонує проголосувати за наступного кандидата.

Ігрові персонажі:

- Мирні жителі

Це спокійна роль у грі. Вночі мирні жителі просто сидять із заплющеними очима, чекаючи своєї долі. Вдень їхнє завдання — якнайшвидше виявити та усунути мафію. Чим швидше місто переможе, тим швидше почнеться нова гра, в якій гравець може отримати більш цікаву роль.

- Комісар

Комісар грає за «чесних». Після оголошення ведучого: «На розслідування виходить комісар», комісар розплющує очі та вказує на одного з гравців. Ведучий показує пальцем вниз, якщо це мафія, або пальцем вгору, якщо це є чесний житель. Після цього комісар заплющує очі. Комісар повинен приховувати свою особу, щоб уникнути швидкої розправи з боку мафії.

- Сержант

Сержант – помічник комісара, який грає за «чесних». Він разом із комісаром стежить за сусідами ночами і може бути його рупором для оголошення виявлених порушень. У разі смерті комісара сержант займає його місце, одержуючи підвищення та вступаючи в активну гру. Поки комісар живий, сержант спостерігає, розмірковує та прикриває його тили. Після підвищення сержант активно грає вдень, прикидаючись чесним жителем або видаючи комісара лише у крайньому випадку.

- Шериф

Шериф грає за чесних. Вночі він може вбити будь-якого гравця, але може утриматися від вбивства. Роль шерифа вимагає дотримуватися закону, що робить її гідною, але не вдячною.

- Лікар

Вдень лікар намагається визначити, кого мафія захоче вбити вночі. Вночі він лікує вибраного навмання гравця. Вранці оголошується, кого саме вилікував лікар. Якщо лікар двічі вилікує одного й того самого гравця, цей гравець вмирає від передозування.

- Мафія

Роль мафії є однією з найважчих і водночас найцікавіших у грі. Чисельність мафії невелика, тому важливо, щоб інші гравці не здогадалися, хто є мафією. Мафії доводиться бути особливо уважною, щоби не видати себе. Вночі мафії надається можливість усувати гравців, що заважають. Основні цілі - комісар, який може перевірити будь-якого гравця у ведучого, якщо запідозрить недобре. Мафії треба пам'ятати, що за одну ніч можна вбити лише одного городянина. Швидке ухвалення рішення про вибір жертви зменшує ймовірність зробити необережний рух, який міг би видати мафіозі.

- Фатальна жінка

Фатальна жінка грає за мафію і має унікальну здатність. Вночі вона може вибрати одного гравця та «заткнути йому рот». Цей гравець не зможе голосувати та висловлювати свою думку наступного дня.

Гра «Мафія» спонукає до рефлексії щодо людських взаємин та самоаналізу. У процесі ігрової взаємодії виявляються особисті недоліки та переваги, іноді зовсім несподіваним чином. Гра сприяє кращому розумінню та інтерпретації мотивів поведінки інших учасників. По завершенні гри зберігається бажання подумки аналізувати події, що відбулися. Згадуються епізоди, в яких людина, яка не викликає довіри протягом усієї гри, виявлялася «лікарем», який рятує вас вночі, а учасник, що захищається, — «мафіозі». Виникає питання про причини таких помилкових суджень, що стимулює бажання продовжувати грати з метою глибшого розуміння соціальних взаємодій та механізмів прийняття рішень.

1.2 Загальний опис веб-додатків

В наш час, велика кількість звичних настільних ігор має свої аналоги в онлайн форматах (додатки, веб-сайти та веб-додатки), що дає можливість грати з друзями навіть в різних кінцях світу.

Веб-додаток (Web Application) — це програмне забезпечення, яке працює у браузері та надає користувачеві доступ до різних функцій та сервісів через інтернет. Веб-програми можуть бути доступні на різних пристроях, таких як:

- комп'ютери,
- смартфони,
- планшети,

Вони не вимагають встановлення на пристрої користувача, оскільки вони запускаються у веб-браузері.

Веб-додатки використовують комбінацію серверних та клієнтських сценаріїв для виконання завдань користувача. З огляду на те, що мільйони

компаній використовують Інтернет як свою маркетингову стратегію, веб-додаток має надавати інформацію, зберігати дані, обробляти транзакції та взаємодіяти з потенційними споживачами [1]

Мета створення веб-додатку різна. Вона полягає у взаємодія з користувачами, налагодження бізнес-процесів всередині компанії, організація навчального процесу, проведення фінансових операцій, синхронізація роботи над спільними документами.

Прикладами веб-додатків є:

- Інтернет-пошта (Gmail).
- Хмарні сховища (Dropbox).
- Текстові редактори (Google Документи) та онлайн-нотатки (Evernote).
- Соціальні мережі (Facebook, Tik-Tok, Instagram).
- Магазини електронної комерції (Amazon, Rozetka, Foxtrot, Comfy).
- Системи замовлень та продажів (Booking.com,Tickets.ua, Proizd.ua).
- Системи управління проектами (Trello).
- Онлайн-банкінги (Privat24, Monobank).

Особливості веб-додатку полягає в прийманні запиту від клієнта та проведенні обчислення. Потім воно формує веб-сторінку та відправляє її клієнту через мережу, використовуючи протокол HTTP. До того ж, веб-програма може бути клієнтом для інших служб, таких як база даних або інша веб-програма, розташована на іншому сервері.

Останнім часом набув популярності новий підхід до розробки веб-додатків, - Ajax(Asynchronous JavaScript and XML.). Суть підходу полягає в тому, що сторінки веб-застосунку не перезавантажуються повністю, а завантажують лише змінені дані з сервера. Це робить сторінки більш інтерактивними та підвищує продуктивність програми.

Вимоги до веб-додатку.

- Зручність у використанні. Відвідувач веб-додатку має зрозуміти, як користуватися ним протягом 10-15 секунд.
- Реакція на дії користувача. Від натиску до появи результату не більше 10 секунд.
- Оптимізація швидкості виконання програми.
- Безпека.

Переваги веб-додатків:

1. Доступність. Як вище було зазначено, доступ до програми може здійснюватися з будь-якого пристрою та будь-якої точки світу, за умови доступу до інтернету.
2. Адаптивність. web-application не вимагають конкретної операційної системи. Вони працюють на всіх ОС і у всіх існуючих браузерах:
   - Opera;
   - Chrome;
   - Firefox та ін.
3. Відсутність програмного забезпечення для клієнтів. Не потрібно нічого завантажувати і встановлювати, що спрощує процес користування та суттєво оптимізує процес обслуговування, модернізації інтерфейсу. Всі оновлення активуються автоматично при завантаженні сторінки.
4. Захист даних. Хмарне сховище забезпечує зберігання всієї особистої інформації користувачів та їх дані.
5. Високий рівень безпеки. Забезпечення захисту та налаштування відбувається централізовано завдяки єдиній точці входу, яка характеризується високим рівнем безпеки.

Ще один важливий пункт дослідження – етапи розробки. Вони включають в себе такі пункти[4]:

Перший етап, дослідження та аналіз. На цьому етапі відбувається збір даних, а саме функції які буде виконувати додаток, побажання замовника для

якого створюється додаток. Аналіз схожих вже існуючих додатків, аналіз цільової аудиторії.

Другий етап – це дизайн. При розробці дизайну інтерфейсу враховується тренди і вимоги, легкість сприйняття, простота, зручність та доступність використання додатку.

Наступний етап – розробка та програмування. Тут важливим є детальне опрацювання архітектури веб-додатку та швидкість завантаження. Основним на цьому етапі є вибір технологій:

- Фронтенд-технології: Вибір мов і фреймворків для розробки інтерфейсу користувача (наприклад React, Angular, Vue.js).
- Бекенд-технології: Визначення серверної частини програми (наприклад, Node.js, Python, Ruby, PHP, Java, .NET).
- Бази даних: Рішення, яку СУБД використовуватиме зберігання даних (наприклад, MySQL, PostgreSQL, MongoDB).

Четвертим етапом є тестування. Під час даного етапу розробки виявляються помилки, недоліки та невідповідності, після чого поправляються. Низка пунктів за якими відбувається правильне тестування додатку:

- Юніт-тестування: Перевірка окремих компонентів на коректність роботи.
- Інтеграційне тестування: Тестування взаємодії між різними компонентами програми.
- Функціональне тестування: Переконайтеся, що всі функції програми працюють правильно.
- Тестування навантаження: Оцінка продуктивності програми під високим навантаженням.

Фінальний етап – впровадження та завершення. Додаток випроваджується та передається замовнику [3]. Звертаємо увагу на такі пункти:

- Вибір хостингу: Визначення сервера або хмарного провайдера для розміщення веб-програми.
- Налаштування CI/CD: Автоматизація процесів збирання, тестування та розгортання.
- Моніторинг та підтримка: Налаштування систем моніторингу для відстеження роботи програми та оперативного реагування на проблеми.

Створення веб-програми - це багатоетапний процес, що вимагає ретельного планування та виконання. Важливо враховувати всі аспекти, починаючи від аналізу вимог та вибору технологій, до розробки, тестування та розгортання. Успішне виконання кожного етапу гарантує створення якісного та функціонального веб-додатку, що задовольняє потреби користувачів.

1.3 Аналіз існуючих аналогів веб-додатку гри «Мафія»

Сьогодні існує певна кількість веб-версій гри. Для розробки власної рольової комп'ютерної гри «Мафія» треба провести аналіз аналогів вже існуючих ігор. Для аналізу обиралися веб-додатки, що схожі за принципом гри на оригінальну карткову версію. Під час особистого аналізу аналогів було виявлено один аналог:

Версія веб-додатку «Мафія онлайн» (див. Рис. 3)

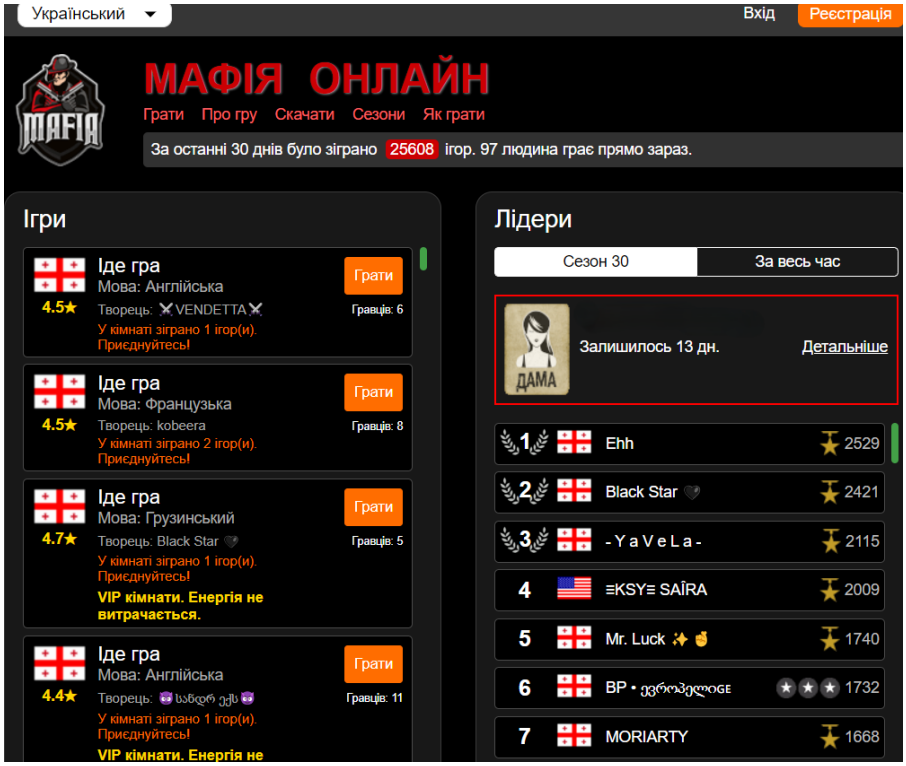

Рисунок *1.1 — «Мафія онлайн»*

На даному зображенні видно інтерфейс веб-програми «Мафія Онлайн».

Загальний інтерфейс: Інтерфейс підтримує українську мову, що видно з меню мови у верхньому лівому кутку.

Навігація: У верхній частині екрана розташоване меню з основними розділами:

- Грати
- Про гру
- Скачати
- Сезони
- Як грати

Користувальницький доступ: У верхньому правому куті є кнопки «Вхід» і «Реєстрація», що дозволяє користувачам увійти в систему або зареєструватися.

Основні елементи э статистика та список ігор.

Статистика відображає інформацію про кількість зіграних ігор за останні 30 днів (25608) та кількість активних гравців на даний момент (97).

У лівій частині екрана знаходиться список активних ігор із зазначенням:

- Назви гри та мови (наприклад, англійська, французька, грузинська).
- Рейтинг гри (4.5 зірки).
- Творця гри.
- Кількість гравців у кімнаті.
- Кнопки для приєднання до гри («Грати»).

У правій частині екрана відображається поточний сезон (30) та лідери сезону із зазначенням їх рейтингу та країни.

Візуальні та функціональні аспекти.

Колірна схема: Інтерфейс виконаний у темних тонах із використанням червоних та помаранчевих акцентів для виділення важливих елементів.

Зручність використання: Усі основні елементи навігації та взаємодії знаходяться в межах одного екрана, що спрощує доступ до потрібної інформації та дій.

Веб-додаток «Мафія Онлайн» пропонує інтуїтивно зрозумілий інтерфейс із чіткою структурою, що дозволяє легко орієнтуватися користувачам. Воно підтримує різні мови, надає інформацію про поточні ігри та лідерів, а також включає елементи для привернення уваги до нових та популярних функцій.

1.4 Висновки першого розділу

У цьому розділі було проведено аналіз предметної області де було зазначено принцип гри «мафія», поетапний розбір ходу гри та персонажів, що є необхідним для створення онлайн версії гри. У другому підрозділі було визначено основні положення та вимоги до створення власного веб-додатку. Це

дало чітке уявлення, які задачі повинні бути виконанні під час розробки веб-додатку і на що потрібно звернути особливу увагу.

У третьому підрозділі було проведено дослідження та огляд вже існуючих веб-додатків «мафія», тобто аналогів. Цей аналіз дозволив виявити недоліки та переваги вже існуючих додатків, що дає можливість визначити та покращити власну розробку веб-ресурсу.

2 ДОСЛІДЖЕННЯ ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Аналіз методів рендерингу веб-додатків

2.1.1 Client-Side Rendering (CSR)

Client-Side Rendering (CSR) є підходом, при якому більшість або весь рендеринг веб-сторінок здійснюється на стороні клієнта. В цьому підході сервер надсилає мінімальний HTML-документ, а решта вмісту завантажується та обробляється за допомогою JavaScript (див. Рис. 2.1), що зазвичай здійснюється через фреймворки, такі як React, Vue або Angular [2].

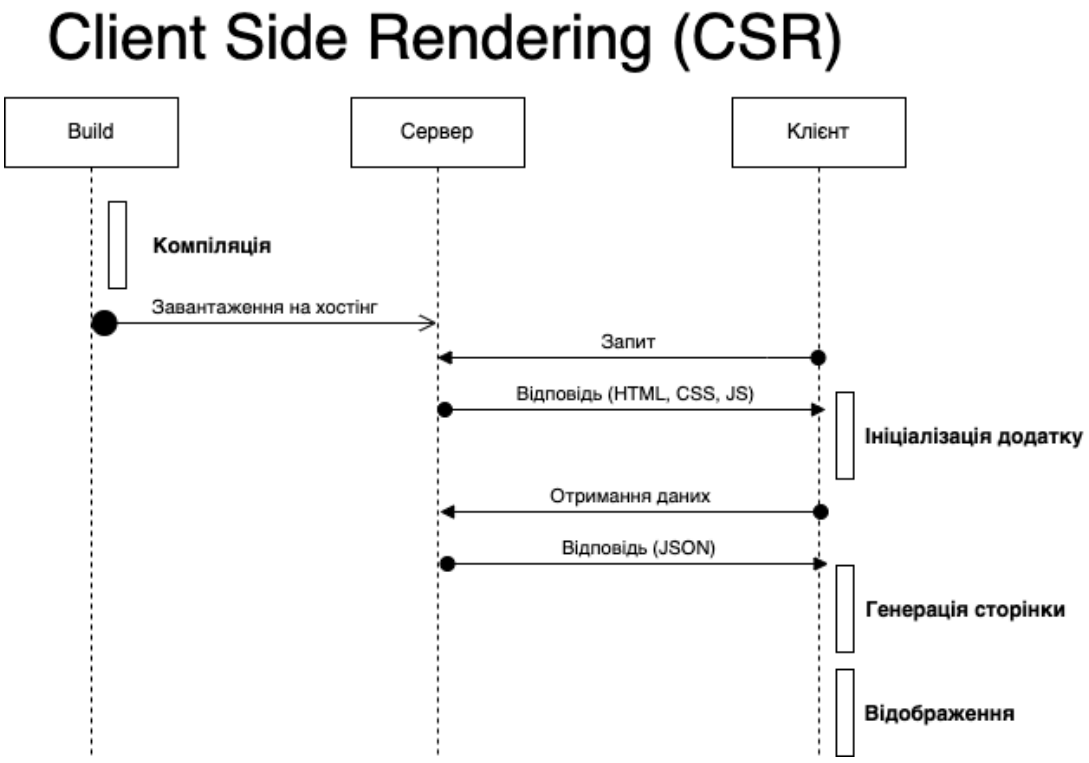

Рисунок *2.1 — Client-Side Rendering*

Переваги:

- Легкість реалізації: додаток складається з кількох файлів: файлу стилів, скриптів та умовно порожньої HTML-сторінки, в якій підключаються стилі та скрипти.
- Мінімальне навантаження на сервер: не потрібно рендерити сторінку на кожен запит, а лише повернути статичні файли.
- Актуальність інформації: при переході між сторінками змінюється URL-адреса в браузері та, за потреби, підтягуються дані з сервера.
- Відсутність необхідності у сервері: можна створити окремий RESTful сервер, який повертатиме дані в JSON форматі.
- Легкість масштабування.
- Недорогий запуск у виробництво: не потрібно багато ресурсів для повернення декількох файлів, що дозволяє зекономити на хостингу.

Недоліки:

- Проблеми з SEO: пошукові системи не можуть коректно індексувати сторінки з мінімальним HTML.
- Навантаження на клієнтські пристрої: рендеринг сторінки може зайняти значний час на старих або слабких пристроях через великий обсяг JavaScript.
- Розмір JavaScript файлу: при першому завантаженні розмір JavaScript файлу може бути значним.

Інструменти:

- Create React App (CRA): інструмент для швидкого створення та налаштування React-додатків з мінімальними конфігураціями.
- Vue CLI: командний інтерфейс для створення та налаштування додатків на базі Vue.js.
- Angular CLI: офіційний інструмент для створення, налаштування та управління Angular-додатками.

- Vite: сучасний інструмент для збірки фронтенд-додатків, який пропонує надзвичайно швидкий старт та гаряче перезавантаження, оптимізований для роботи з сучасними фреймворками, такими як React та Vue.

2.1.2 Server-Side Rendering (SSR)

Основною особливістю цього підходу є те, що вся веб-сторінка рендериться на сервері при першому запиті, після чого в браузері ініціалізується React, і далі додаток функціонує як звичайний SPA, виконуючи, за потреби, AJAX-запити до сервера (див. Рис. 2.2). Дані завжди завантажуються актуальні, оскільки при кожному перезавантаженні сторінки відбувається запит до сервера. З точки зору SEO, цей підхід є ідеальним, оскільки повертається повноцінна HTML-сторінка. Користувачі також задоволені, адже вони працюють з додатком, як із звичайним SPA, не помічаючи різниці[2].

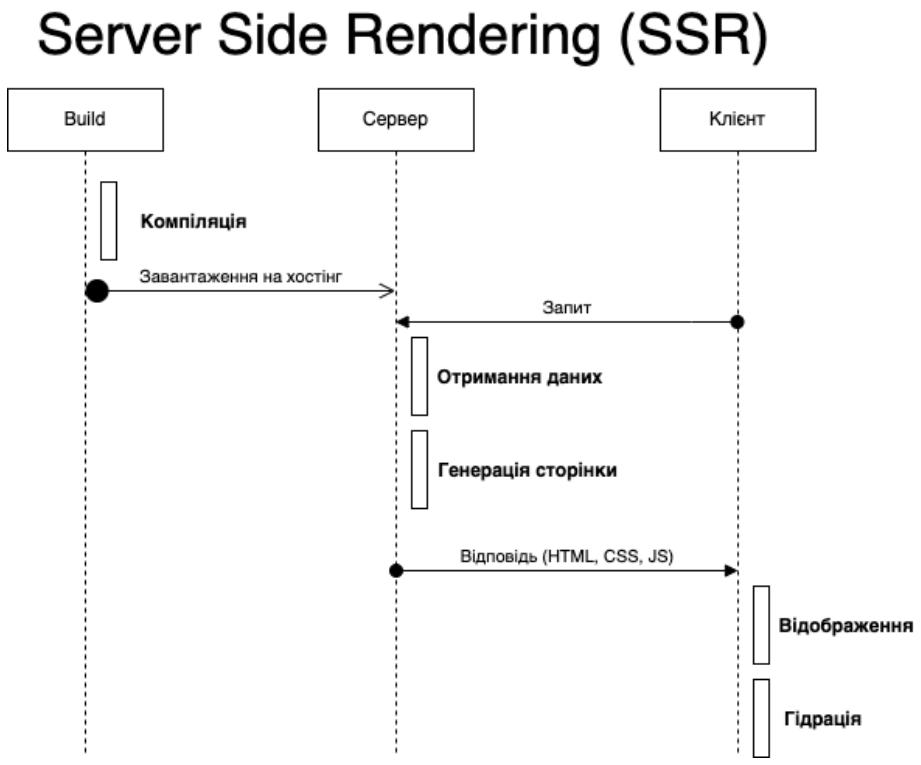

Рисунок *2.2 — Server-Side Rendering*

Переваги:

- Ідеальна реалізація SEO: повноцінні HTML-сторінки забезпечують високу видимість у пошукових системах.
- Актуальність інформації: дані завжди актуальні завдяки постійним запитам до сервера.
- Висока швидкість завантаження та відображення контенту.
- Налаштування кеша на сервері: можливість налаштування кешування для покращення продуктивності.

Недоліки:

- Реалізація є складнішою у порівнянні з клієнтським рендерингом.
- Високе навантаження на сервер: кожен запит потребує обробки.
- Збільшення розміру файлів, які завантажуються з сервера.
- Довша відповідь від сервера: збільшений час до першого байта (TTFB).

Інструменти:

- NodeJS: платформа для серверного JavaScript, яка дозволяє реалізувати серверний рендеринг.
- Next.js: фреймворк для React, який спрощує реалізацію SSR.
- Remix: сучасний фреймворк для побудови веб-додатків, який підтримує SSR.

2.1.3 Static Site Generation (SSG)

Якщо дані оновлюються не часто, то немає необхідності кожного разу генерувати цілу сторінку. В таких випадках застосовується підхід статичної генерації сайтів (SSG). Замість того, щоб генерувати сторінки при кожному запиті, можна один раз при побудові додатку згенерувати всі сторінки. Це передбачає створення HTML, CSS та JS файлів заздалегідь, які потім розміщуються на простому файловому сервері або навіть в CDN, і повертаються

при кожному запиті (див. Рис. 2.3). Такий підхід усуває потребу в сервері для обробки запитів чи генерування сторінок, забезпечуючи максимальну швидкість завантаження та безпеку. Найбезпечніший сервер — це той сервер, якого немає.

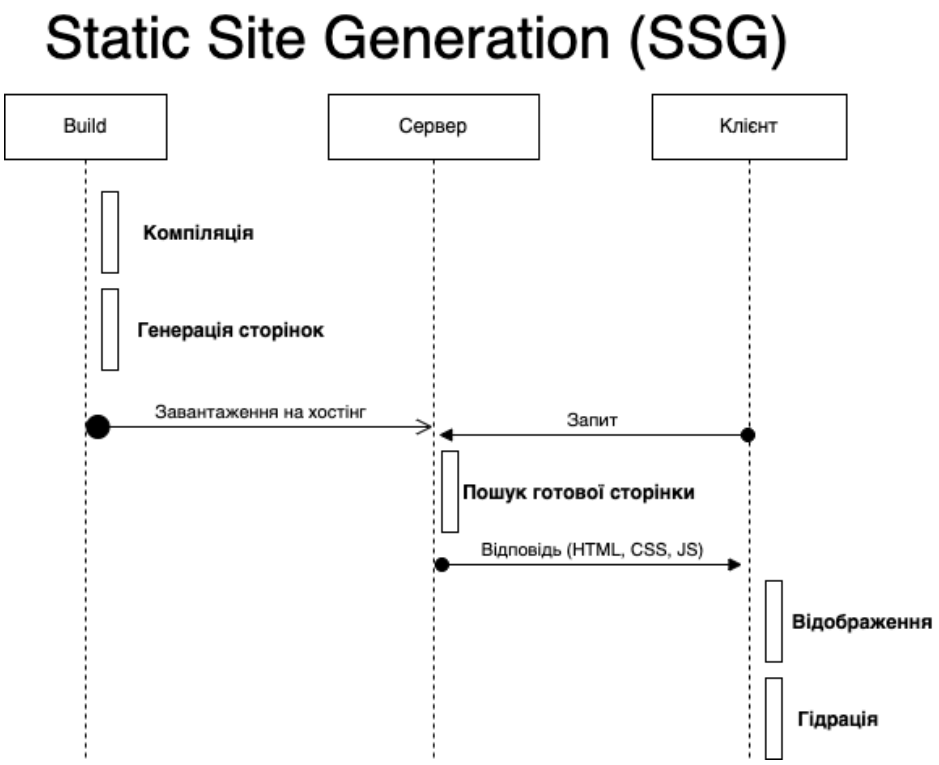

Рисунок *2.3 — Static Site Generation*

З точки зору SEO, це ідеальний варіант. Для користувача сторінка відображається ще швидше ніж при SSR, оскільки не треба очікувати обробки даних на сервері , що зменшує час до першого байта (TTFB). Для користувачів все залишається незмінним, оскільки після завантаження сторінки ініціалізується React, і додаток працює як звичайний SPA.

Недоліком цього підходу є те, що при оновленні даних необхідно повністю перезібрати проєкт і відрендерити всі сторінки. Хоча це більше особливість, ніж недолік, варто зазначити, що генерація великої кількості сторінок, наприклад для сайту з 100 000 сторінками, може зайняти декілька годин.

Можуть виникнути питання щодо взаємодії користувачів з сайтом, наприклад, при відправці форм або зборі інформації. В таких випадках можна запустити окремий RESTful сервер, який оброблятиме різноманітні запити від клієнтів. Також варто звернути увагу на серверлесс або «хмарні функції», такі як AWS Lambda або GCP Cloud Functions[2].

Переваги:

- Ідеальна реалізація SEO: повноцінні HTML-сторінки забезпечують високу видимість у пошукових системах.
- Висока швидкість завантаження та відображення.
- Відсутність сервера для обробки запитів підвищує безпеку.
- Дешевий хостинг: мінімальні вимоги до серверних ресурсів.

Недоліки:

- Оновлення контенту: необхідно перезбирати проєкт.
- Час на генерацію: генерація великої кількості сторінок може зайняти значний час, наприклад, для 100 000 продуктів.
- Інструменти:
- GatsbyJS: фреймворк для створення статичних сайтів на основі React.
- NextJS: фреймворк для React, який підтримує як SSG, так і SSR.

2.1.4 Incremental Static Regeneration (ISR)

Цей метод поєднує переваги двох попередніх підходів, Server-Side Rendering (SSR) і Static Site Generation (SSG). Не завжди доцільно генерувати сторінку при кожному запиті, і заздалегідь згенеровані сторінки також не є ідеальним рішенням, якщо контент оновлюється час від часу.

Рішенням є розділення сторінок. Ті, що частіше оновлюються, будуть рендеритися за допомогою SSR, тоді як ті, що рідко оновлюються, будуть згенеровані за допомогою SSG[2].

Переваги:

- Ідеальна реалізація SEO: забезпечує високу видимість у пошукових системах завдяки повноцінним HTML-сторінкам.
- Висока швидкість завантаження та відображення контенту.
- Гнучкість застосування: Можливість застосування підходу для деяких сторінок, що потребують частих оновлень.
- Відносна актуальність інформації: Сторінки оновлюються за необхідністю, забезпечуючи актуальність даних.

Недоліки:

- Реалізація є складнішою порівняно з іншими методами.
- Необхідність сервера: обробка запитів і оновлення сторінок.

Інструменти:

- Next.js: фреймворк для React, який підтримує інкрементальну регенерацію статичних сторінок і дозволяє комбінувати SSG та SSR підходи.

2.2 Аналіз сучасних технологій для створення клієнтської частини

2.2.1 React

React є бібліотекою для побудови інтерфейсів користувача, що функціонує виключно на рівні представлення. Відмінною рисою React є його вузька спеціалізація, яка контрастує з повноцінними фреймворками, такими як Angular. React забезпечує інструменти для створення компонентів, які визначають, як відображатиметься HTML, але не включає функціональні можливості, притаманні більш комплексним фреймворкам.

React надає мову шаблонів і callback-функції для відтворення HTML. Результатом роботи React є генерація HTML-коду, а компоненти React, що об'єднують HTML та JavaScript, зберігають свій внутрішній стан у пам'яті, але кінцевий продукт завжди залишається у форматі HTML. Таким чином, для

створення повноцінного динамічного застосунку необхідно використовувати додаткові бібліотеки та інструменти[5].

Переваги:

- Код компонента React чітко показує, як буде виглядати інтерфейс залежно від стану. Такий підхід дозволяє легко розуміти та налагоджувати код, особливо у великих командах.
- Комбінація HTML та JavaScript у JSX робить компоненти легшими для розуміння. Незважаючи на те, що традиційно вважається найкращою практикою розділення логіки (JavaScript) та представлення (HTML), JSX дозволяє органічно поєднувати їх, що сприяє кращій організації коду.
- React дозволяє рендерити компоненти на сервері, що є важливим для оптимізації продуктивності та покращення SEO. Це забезпечує швидке завантаження сторінок та покращує користувацький досвід.

Недоліки:

- React не надає системи подій, роботи з AJAX, шару даних, promises або повноцінного фреймворка. Для створення повноцінного застосунку необхідно використовувати додаткові бібліотеки та інструменти, що може призвести до необхідності розробки власних рішень для багатьох завдань.
- Документація React може бути заплутаною для новачків, особливо через наявність кількох конкуруючих туторіалів для початківців та складність у навігації.
- Розмір бібліотеки є досить великою з огляду на обмежений набір функцій. Наприклад, мінімізований файл React займає близько 35 KB, що є значним обсягом для бібліотеки, яка займається лише рендерингом.

Однією з ключових концепцій, пов'язаних з React, є Flux — патерн, що забезпечує односторонній потік даних. Це означає, що події, викликані користувачем, змінюють модель, а потім модель оновлює представлення (див.

Рис. 2.4). Такий підхід допомагає підтримувати синхронність станів моделі та представлення, але також призводить до різних реалізацій та складності у виборі правильної бібліотеки[6].

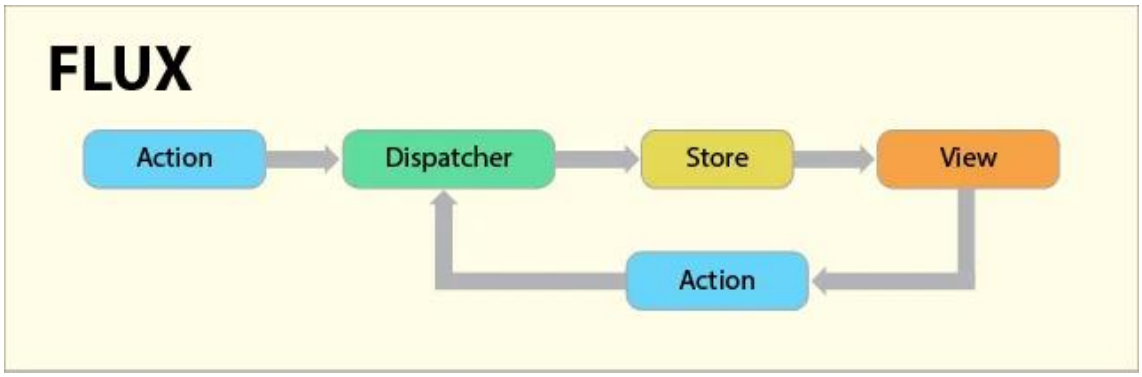

Рисунок *2.4 — Архітектура Flux*

Однією з ключових технологій, що відрізняє React від інших бібліотек і фреймворків, є концепція віртуального DOM (Document Object Model).

Реальний DOM є представленням HTML-структури веб-сторінки у браузері. Коли користувач взаємодіє з веб-сторінкою, наприклад, натискаючи кнопку або заповнюючи форму, браузер оновлює Реальний DOM для відображення змін. Після цього браузер повторно відображає сторінку з оновленим HTML.

Віртуальний DOM є абстракцією над Реальним DOM. Він є легковаговою копією Реального DOM, що дозволяє здійснювати швидші оновлення та підвищувати продуктивність. Коли користувач взаємодіє з веб-сторінкою, React оновлює Віртуальний DOM, порівнює його з попередньою версією та оновлює Реальний DOM лише необхідними змінами (див. Рис. 2.5). Цей процес відомий як узгодження (Reconciliation)[7].

Рисунок *2.5 — Взаємодія Віртуального DOM з Реальним DOM*

Віртуальний DOM і Реальний DOM відрізняються наступними ключовими аспектами:

- Реальний DOM значно повільніший за Віртуальний DOM, оскільки кожне оновлення вимагає від браузера перерахунку всієї структури документа. Віртуальний DOM, будучи легковаговою копією Реального DOM, дозволяє здійснювати швидші оновлення та підвищувати продуктивність.
- Реальний DOM має деревоподібну структуру, і кожен вузол має прив'язаний слухач подій. Коли користувач взаємодіє з веб-сторінкою, браузер виконує відповідні слухачі подій, що може бути повільним і ресурсоємним процесом. Віртуальний DOM, навпаки, забезпечує більш ефективне оновлення, оскільки слухачі подій прив'язані лише до кореневого вузла, а не до кожного окремого вузла.

2.2.2 Vue.js

Vue.js є прогресивним фреймворком для JavaScript, який використовується для розробки веб-інтерфейсів та односторінкових додатків. Фреймворк Vue.js

спирається на архітектуру Model-View-Controller (MVC), зосереджуючи увагу на рівні представлення (див. Рис. 2.6). Проте, він також підтримує інші архітектурні підходи, зокрема компонентну архітектуру, що дозволяє розробникам ефективно організовувати свої проекти.

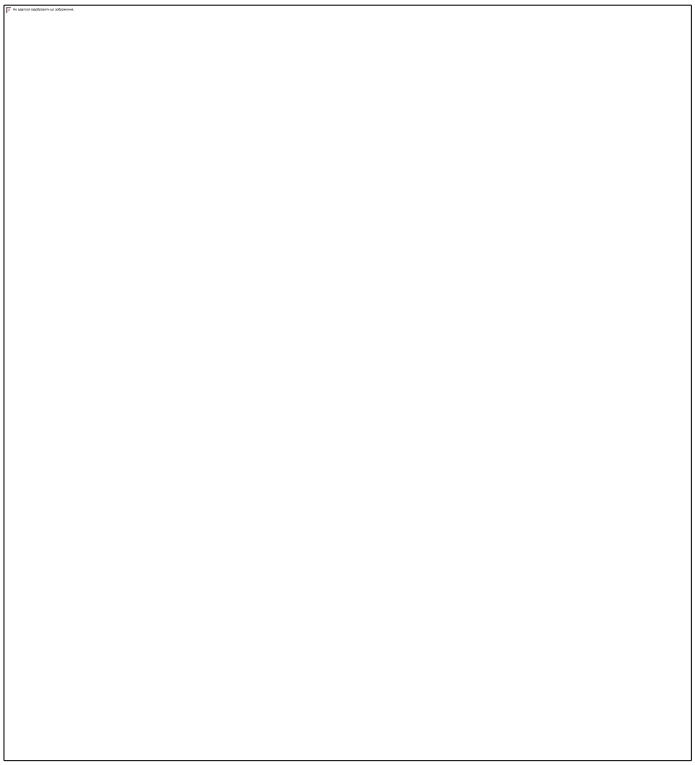

Рисунок *2.6 — Архітектура Model-View-Controller*

Однією з основних переваг Vue.js є його простота використання та навчання. Він не вимагає глибоких знань з боку розробника і може бути використаний з базовими HTML, CSS і JavaScript. Крім того, Vue.js відомий своєю високою продуктивністю та легкістю інтеграції з існуючими проектами.

Переваги:

- Простота використання: не вимагає глибоких знань з боку розробника і може бути використаний з базовими HTML, CSS і JavaScript.

- Продуктивність: забезпечує високу продуктивність та масштабованість, має невеликий розмір та оптимізовану швидкість роботи.
- Легкість розуміння: використовую чітко визначену архітектуру.
- Проста інтеграція: легке включення компонентів до існуючих проектів.

Недоліки:

- Мовний бар'єр: основна спільнота розташована в Китаї, що може ускладнити розуміння деяких пакетів та документації для англомовних розробників.
- Ризик надмірної гнучкості: пропонує багато свободи у використанні різних підходів, що може призвести до розбіжностей у коді та складнощів у роботі над великими проектами.
- Менша кількість компонентів і плагінів: може відчуватися деяка нестача плагінів та бібліотек, проте, основні компоненти присутні, а широкий вибір плагінів стає доступним з ростом популярності фреймворку.

У цілому, Vue.js є потужним та ефективним фреймворком для розробки веб-інтерфейсів та односторінкових додатків. Його простота використання та висока продуктивність роблять його привабливим вибором для багатьох розробників[8].

2.2.3 Angular

Фреймворк Angular пропонує інтеграцію декларативних шаблонів, передових практик, впровадження залежностей та інші інструменти, що сприяють швидкому і ефективному розвитку програмного забезпечення[9].

Переваги:

- Архітектура MVC: пропонує архітектурний підхід, що базується на модель-подання-контролер, сприяючи ефективній організації та розділенню логіки програми та інтерфейсу користувача.

- Розширена архітектура дизайну: спрощує процес управління багатьма компонентами, роблячи його доступним навіть для початківців програмістів.
- Модульність: використовує модульний підхід, що сприяє зручному поділу функціональності на окремі модулі.
- Прив'язка даних: забезпечує двосторонню прив’язку даних, що спрощує взаємодію між моделлю та представленням.
- Сервіси та впровадження залежностей: вбудований механізм ін’єкції залежностей, що полегшує створення модульних та ефективних додатків.
- Директиви: дозволяє розширювати HTML за допомогою директив, що забезпечує більшу гнучкість у взаємодії з користувацьким інтерфейсом.
- TypeScript: забезпечує переваги статичної типізації, зменшуючи кількість помилок під час розробки та полегшуючи читабельність коду.

Недоліки:

- Великий розмір: порівняно з іншими фреймворками, має досить значний розмір, що може призвести до більшого часу завантаження для користувачів і займати більше місця на сервері.
- Складність навчання: може здатися складним через велику кількість концепцій та відмінностей від стандартного JavaScript.
- Часті зміни: версії оновлюються досить часто, що може створювати проблеми з сумісністю між різними версіями та потребує постійного оновлення додатків.
- Важкість SEO: у порівнянні з іншими фреймворками, Angular може мати проблеми з оптимізацією для пошукових систем, оскільки спочатку весь контент рендериться на клієнтському боці, що може ускладнити індексацію вмісту.

2.3 Аналіз сучасних технологій для створення серверної частини

2.3.1 NestJS

Необхідність написання серверного коду на мові програмування JavaScript з використанням платформи Node.js виникає у зв'язку з різноманітними потребами в інтеграції бізнес-логіки між існуючим бекендом та фронтендом. Зазвичай код на чистому Node.js мало хто пише, в основному використовуються фреймворки. Одним з найпопулярніших фреймворків на сьогоднішній момент є Express — простий та мінімалістичний фреймворк для Node.js, який широко використовується. Він простий у використанні, гнучкий та, здається, здатний легко здійснювати будь-які завдання, однак має наступні недоліки:

- Гнучкість: без належних навичок використання є великий ризик допустити помилки.
- Імпорт: ця функція реалізована не надто нативно, тому якщо помилитися з архітектурою, заплутування уникнути не вдасться.
- Типізація: більшість перевірок на тип та порожність потрібно виконувати вручну, оскільки використовується JavaScript, а не TypeScript.
- Взаємодія з фронтендом: додаткова робота з кукі та іншими серверними взаємодіями з боку користувальницького інтерфейсу.

Все вищезазначене свідчить про те, що в Express використовується застарілий патерн з використанням функцій зворотного виклику (callback functions), що часто призводить до неприємних наслідків. Використання Express у чистому вигляді практично неможливе, за винятком деяких простих додатків. Тому буде розглянуто фреймворк NestJS.

NestJS — це фреймворк для створення серверних додатків на Node.js. Він написаний на TypeScript і повністю його підтримує (навіть сучасні версії). Це надає можливість перевірки типів даних: як простих, так і складних. Enum та інші опції допомагають у цьому. Декоратори — це функціонал декларативного

програмування, що дозволяє розширювати будь-які методи за бажанням. Класичні імпорти TypeScript дозволяють прискорити розробку та писати код великими командами без перешкод. Чіткий розподіл функціонального навантаження на компоненти:

- Interceptor — відповідає за додавання, перехоплення запитів.
- Guards — відповідає за перевірки доступності за певними критеріями.
- Pipes — виконує дві функції. Перша — перетворення одного типу в інший. Наприклад, коли очікується рядок для ID, але отримується число, і для того, щоб не довелося перетворювати типи на клієнтському інтерфейсі, pipe реалізує це на бекенді. Друга функція полягає у перевірці відповідності типів.
- Custom route decorators — потужний інструмент для керування маршрутизацією веб-запитів. Вони спрощують маршрутизацію, групуючи маршрути та пов’язану з ними логіку в одному місці, а також можуть бути використані для встановлення додаткових правил обробки запитів.
- Exception filters — це всі можливі винятки (згідно з мережевою взаємодією). У тих випадках, коли помилка не була оброблена розробником, він все одно отримає дані про неї.
- Middleware — попереднє оброблення запитів з доступом до всієї інформації запиту.
- Modules — використовуються для організації коду та архітектури додатка.
- Controllers — відповідають за обробку вхідних запитів і надання відповідей клієнту, а також за маршрутизацію, розбір параметрів і загалом за все, що стосується прийняття та віддачі за запитом.

NestJS підтримує будь-які протоколи на додачу до HTTP, наприклад, на основі RabbitMQ, Nats, Kafka або навіть звичайного TCP-протоколу[13].

Усе це доступно просто з коробки, без необхідності встановлення модулів, плагінів, розширень та іншого. Крім того, є командний рядок інтерфейсу користувача (CLI), що означає, що за допомогою командного рядка код буде генеруватися так, як потрібно, і розміщуватися там, де це потрібно, що, в свою чергу, значно прискорить роботу.

Переваги:

- Легкість використання, навчання та освоєння.
- Потужний інтерфейс командного рядка для підвищення продуктивності та спрощення розробки.
- Детальна та налагоджена документація з прикладами.
- Відкритий вихідний код.
- Прості додатки для модульного тестування.
- Створений для монолітів та мікросервісів.

Недоліки:

- Потрібно вивчити великий обсяг інформації, щоб користуватися правильно.
- Доведеться багато писати DTO, щоб у запиті дані оброблялися коректно.
- Іноді доводиться робити багатократні перевірки за типами, наприклад, Enum та інші.

2.3.2 Django

Згідно статистичній платформі: Python є другою за популярністю мовою програмування для розробки серверних додатків на 2023 рік[10]. Ця скриптова мова має досить простий синтаксис. Python має динамічну типізацію. Для веб-розробки. Python часто обирають для серверної частини розробки. Серед великомасштабних веб-сайтів і програм, створених за допомогою цієї мови, є Google, Facebook, Instagram, YouTube, Dropbox і Reddit[11].

Django — це високорівневий веб-фреймворк на мові програмування Python, розроблений для швидкої та зручної розробки веб-додатків. Його основні принципи включають можливість швидкої розробки, забезпечення високої продуктивності, чистоту та простоту коду, а також можливість масштабування.

Основні характеристики та компоненти фреймворку Django:

- Принцип «Батарейки в комплекті»: постачається з великою кількістю вбудованих компонентів, які вирішують типові завдання веб-розробки, такі як автентифікація користувачів, управління сесіями, робота з формами, адміністрування сайту, і багато іншого.
- Чистота коду: використовує принципи чистої архітектури та шаблон проектування Model-View-Template (MVT) (див. Рис. 2.7), що допомагає розробникам створювати чистий і підтримуваний код.
- Вбудований адміністративний інтерфейс: має потужний адміністративний інтерфейс, який генерується автоматично на основі моделей даних. Це дозволяє легко керувати контентом сайту без необхідності створення окремих адміністративних панелей.
- Можливість масштабування: розроблений для обслуговування як малих, так і великих додатків. Архітектура дозволяє легко масштабувати додаток за рахунок розподілення навантаження між декількома серверами.
- Висока безпека: включає багато вбудованих механізмів безпеки для захисту від поширених загроз, таких як SQL-ін'єкції, міжсайтовий скриптинг (XSS), міжсайтові підробки запитів (CSRF) та інші.

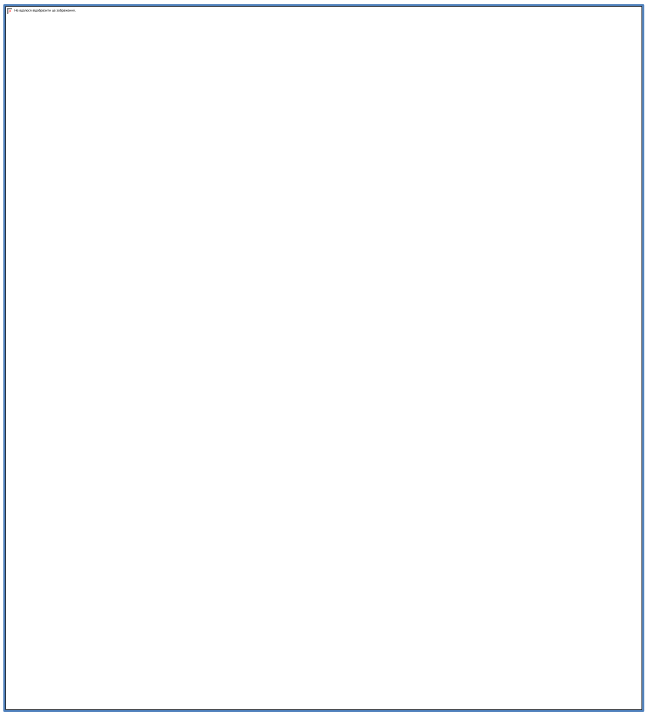

Рисунок *2.7 — Архітектура Model-View-Template*

Основні компоненти Django:

- Моделі (Models): визначають структуру бази даних. Вони представляють собою класи Python, які успадковуються від django.db.models.Model, і містять поля (атрибути), що відповідають стовпцям у таблиці бази даних.
- Види (Views): відповідають за обробку HTTP-запитів і повернення відповідей. Вони визначають, як дані повинні відображатися користувачеві. Види можуть бути як функціями, так і класами.
- Шаблони (Templates): використовуються для генерації HTML-вмісту. Django має власний механізм шаблонів, який дозволяє вставляти змінні та логіку прямо в HTML.
- Маршрутизація (URLconf): використовується файл конфігурації URL (URLconf) для зв'язування URL-адрес з відповідними видами. Це дозволяє легко управляти URL-структурою додатку.

- Форми (Forms): використовуються для обробки вводу користувача. Вони можуть бути автоматично згенеровані на основі моделей або створені вручну для більш складних випадків використання.
- Система автентифікації (Auth): вбудована система автентифікації, яка підтримує створення, авторизацію та управління користувачами.

Переваги:

- Швидка розробка: дозволяє швидко створювати повнофункціональні веб-додатки завдяки багатому набору вбудованих інструментів та функцій.
- Масштабованість: підходить для розробки як малих, так і великих проектів. Він може легко масштабуватися для обслуговування великої кількості користувачів.
- Висока продуктивність: Завдяки оптимізації та можливостям кешування, забезпечує високу продуктивність додатків.
- Активна спільнота: має велику та активну спільноту розробників, що забезпечує постійну підтримку та розвиток фреймворку.
- Багата екосистема: існує багато сторонніх пакетів та додатків, які можна легко інтегрувати в проект, розширюючи його функціональність.

Недоліки:

- Строга структура проекту: вимагає дотримання певної структури проекту, що може обмежувати гнучкість розробників.
- Монолітність: спрямований на розробку монолітних додатків. Хоча він підтримує модульність і дозволяє створювати мікросервіси, це не є основною перевагою фреймворку, і для цього можуть знадобитися додаткові інструменти та конфігурація.
- Продуктивність: Хоча він і забезпечує високу продуктивність для більшості застосувань, в окремих випадках він може бути повільнішим у порівнянні з деякими іншими фреймворками, особливо для додатків, які потребують обробки великої кількості одночасних запитів.

- Розмір і складність: є великим фреймворком з багатьма вбудованими функціями.
- Малопридатний для односторінкових додатків (SPA): фреймворк більше підходить для традиційних веб-додатків, ніж для сучасних односторінкових додатків (SPA), які часто вимагають значної взаємодії з клієнтським інтерфейсом. Хоча він може працювати разом з фреймворками на кшталт React або Angular, це потребує додаткової конфігурації.
- Обмежена асинхронна підтримка: в останніх версіях Django додано підтримку асинхронних запитів, але вона ще не повністю розвинена в порівнянні з іншими фреймворками, такими як Node.js або FastAPI, які зосереджені на асинхронній обробці.
- Менеджмент міграцій бази даних: система міграцій є потужною, але вона може бути складною у використанні для великих проектів з багатьма змінами в схемі бази даних. Неправильне управління міграціями може призвести до проблем з узгодженістю даних.
- Велика кількість абстракцій: фреймворк використовує багато рівнів абстракцій, що може ускладнити відладку та розуміння коду. Це може призводити до складностей у знаходженні джерела проблеми.

3 РОЗРОБКА КОМП'ЮТЕРНОЇ РОЛЬОВОЇ ГРИ

3.1 Архітектура веб-додатку

Веб-додаток для гри «Мафія» можна розбити на дві основні частини: серверну частину (Back-end) та клієнтську частину (Front-end) застосунку.

Серверна частина застосунку, відповідає за обробку запитів від користувачів, взаємодію з базою даних, а також за реалізацію всієї бізнес-логіки гри «Мафія». Основні функції, які виконує серверна частина, включають:

1. Авторизація та реєстрація користувачів:
   - Створення та управління обліковими записами користувачів.
   - Забезпечення безпечної авторизації та автентифікації користувачів.
2. Управління іграми та користувачами:
   - Створення нових ігрових сесій.
   - Додавання та видалення користувачів з ігрових сесій.
   - Зберігання інформації про хід гри, ролі гравців, та результати ігор.
3. Обробка ігрових подій та логіки:
   - Відстеження ходу гри та управління станом гри.
   - Реалізація правил гри «Мафія», включаючи обробку голосувань, вбивств, та інших ігрових подій.
4. Взаємодія з базою даних:
   - Збереження та отримання даних про користувачів, ігри та інші необхідні дані.
   - Забезпечення цілісності та безпеки даних.

Для реалізації серверної частини використовується NestJS, сучасний фреймворк для Node.js, який надає модульну архітектуру та забезпечує високу масштабованість і ефективність. NestJS підтримує TypeScript, що сприяє написанню більш надійного та масштабованого коду. Для забезпечення взаємодії

у реальному часі та миттєвого обміну даними між користувачами, серверна частина також використовує сокети (WebSockets).

Клієнтська частина застосунку відповідає за коректну взаємодію з користувачем, відправку запитів на серверну частину застосунку, отримання та відображення відповідної інформації. Основні функції, які виконує фронтенд частина, включають:

1. Інтерфейс користувача (UI):

- Забезпечення зручного та інтуїтивно зрозумілого інтерфейсу для користувачів.
- Відображення інформації про гру, включаючи ролі гравців, хід гри та результати.

2. Авторизація та реєстрація користувачів:

- Форми для входу та реєстрації нових користувачів.
- Забезпечення зворотного зв'язку для користувачів про успішність або помилки авторизації.

3. Управління ігровими сесіями:

- Інтерфейси для створення та управління ігровими сесіями.
- Відображення поточного стану гри та можливостей для взаємодії (голосування, обговорення тощо).

4. Обмін даними з сервером:

- Відправка запитів на сервер для отримання та відправки даних.
- Обробка відповідей від сервера та оновлення інтерфейсу відповідно до отриманих даних.
- Використання WebSockets оновлень у реальному часі і комунікацій.

Для реалізації фронтенд частини використовується React, популярна бібліотека для створення користувацьких інтерфейсів. React дозволяє створювати компонентний підхід до розробки, що сприяє легкості в розширенні та підтримці коду.

Взаємодія між фронтенд та бекенд частинами застосунку здійснюється за допомогою REST API. Серверна частина надає необхідні ендпоінти для обробки запитів від фронтенд частини, забезпечуючи обмін даними у форматі JSON. Фронтенд частина надсилає запити до цих ендпоінтів для отримання чи відправки даних, необхідних для роботи користувацького інтерфейсу. Крім цього, для забезпечення оновлень у реальному часі та комунікації між гравцями, використовується WebSockets, що дозволяє негайно передавати інформацію між клієнтами та сервером.

Для реалізації веб-додатку «Мафія» з використанням React та NestJS, доцільно застосувати паттерн MVC (Model-View-Controller) в поєднанні з Component-Based Architecture на фронтенді та Modular Monolith на бекенді.

MVC — це паттерн, який дозволяє чітко розділити відповідальності в застосунку, що робить його легшим для розробки, тестування та підтримки.

- Model: відповідає за управління даними, логікою та правилами бізнесу. Включає в себе взаємодію з базою даних та управління станом гри.
- View: відповідає за відображення даних користувачеві та прийом від нього вводу. У випадку використання React, View буде реалізований у вигляді компонентів.
- Controller: обробляє запити від користувача, взаємодіє з моделлю для отримання чи зміни даних, та обирає відповідну View для відображення.

На фронтенді, з використанням React, застосовується Component-Based Architecture:

- Компоненти: дрібні, ізольовані частини інтерфейсу, які можуть бути повторно використані в різних частинах застосунку. Вони відповідають за конкретні функціональні частини інтерфейсу, наприклад, форми авторизації, списки гравців, ігрові таблиці тощо.

- Контейнерні компоненти: виконують роль контролерів, взаємодіючи з серверами через API або WebSockets, та передаючи дані до презентаційних компонентів.

На бекенді використовується Modular Monolith: підхід, при якому бекенд залишається єдиним застосунком, але розділяється на модулі, кожен з яких відповідає за конкретну функціональність. Це зберігає простоту монолітного додатку, одночасно дозволяючи структурувати код та розділяти відповідальності.

Для реалізації взаємодії у реальному часі використовується паттерн Observer: паттерн, при якому об'єкти (спостерігачі) підписуються на події іншого об'єкту (суб'єкта) та отримують повідомлення при зміні стану суб'єкта. В даному випадку, серверний додаток надсилає повідомлення клієнтам через WebSockets, коли відбуваються зміни в стані гри, наприклад, новий хід, голосування, результати тощо.

3.2 Проектування

3.2.1 Проектування бази даних

Для реалізації гри «Мафія» з використанням MongoDB та Mongoose, проектування бази даних включає створення схем даних для основних об'єктів гри, таких як користувачі, гравці, лобі та ігрові сесії. MongoDB як нереляційна база даних дозволяє зберігати дані у форматі документів, що забезпечує гнучкість у структуруванні даних та масштабованість.

MongoDB обрана для зберігання даних через її гнучкість, можливість горизонтального масштабування та високу продуктивність при роботі з великими обсягами даних. Використання Mongoose як ODM (Object Data Modeling) бібліотеки для Node.js забезпечує зручний інтерфейс для роботи з MongoDB, включаючи визначення схем, валідацію даних та роботу з моделями.

Основні схеми даних, які використовуються в базі даних, включають:

1. Колекція «User»
   - username: Унікальне ім'я користувача.
   - displayName: Відображуване ім'я користувача.
   - password: Захешований пароль користувача.
2. Колекція «Player»
   - user: Унікальний ідентифікатор користувача, який є гравцем.
   - role: Роль гравця в грі (наприклад, «Мафія», «Мирний житель»).
   - ready: Статус готовності гравця.
   - alive: Статус гравця (живий, вбитий мафією або вигнаний на голосуванні).
   - socketId: Ідентифікатор WebSocket підключення гравця.
3. Колекція «Lobby»
   - name: Назва лобі.
   - owner: Ідентифікатор гравця, який є власником лобі.
   - participants: Масив ідентифікаторів гравців, які беруть участь у лобі.
4. Колекція «Game»
   - stage: Стадія гри (наприклад, «Розподіл ролей»).
   - players: Масив гравців, які беруть участь у грі.
   - mafiaMet: Масив гравців, які є мафією і зустрілися.
   - donCheck: Гравець, перевірений доном.
   - mafiaVotes: Голоси мафії.
   - doctorSave: Гравець, якого врятував лікар.
   - sheriffCheck: Гравець, перевірений шерифом.
   - dayVotes: Голоси, подані під час денного голосування.
   - currentSpeaker: Поточний промовець.

В базі даних MongoDB немає суворої необхідності дотримуватись нормалізації, як у реляційних базах даних, проте варто дотримуватись принципів, що забезпечують цілісність та зручність обробки даних:

Автозаповнення (Autopopulate) — використовується для автоматичного заповнення референційних полів, що дозволяє уникнути додаткових запитів для отримання пов'язаних даних.

Зв’язки між схемами реалізовані через використання ObjectId типу в Mongoose, що забезпечує референційні зв’язки між документами:

User - Player: Один до одного (кожен гравець є користувачем).

Lobby - Player: Один до багатьох (кожне лобі має багато гравців).

Game - Player: Один до багатьох (кожна ігрова сесія має багато гравців).

3.2.2 Діаграми використання (Use-Case)

Діаграми використання (Use-Case diagrams) є важливим інструментом для визначення функціональності веб-додатку та взаємодії користувачів із системою. Вони допомагають візуалізувати різні сценарії використання додатку та визначити основні вимоги до системи. Для гри «Мафія» були створені діаграми використання, що охоплюють основні функціональні можливості додатку.

Діаграма використання для користувача (див. Рис. 3.1):

Актор: Користувач (User)

Сценарії використання:

- Реєстрація — користувач вводить дані; система зберігає дані в базі даних та створює новий обліковий запис.
- Авторизація — користувач вводить дані для входу; система перевіряє дані та надає доступ до системи у випадку успішної автентифікації.
- Редагування профілю — користувач змінює дані профілю; система перевіряє дані та оновлює документ у базі даних.

- Створення лобі — користувач створює нове лобі; система створює новий документ та додає гравця до нового лобі.
- Приєднання до лобі — користувач обирає існуюче лобі; система додає гравця до обраного лобі та оновлює інформацію про учасників.
- Вихід з системи — користувач виходить з облікового запису.

Рисунок *3.1 — Діаграма використання User*

Діаграма використання для гравця(див. Рис. 3.2):

Актор: Гравець (Player)

Сценарії використання:

- Приєднання до гри — гравець знаходиться у лобі до початку гри; система додає гравця до обраної гри та оновлює інформацію про учасників.
- Отримання ролі — гравець отримує роль на початку гри; система зберігає
- Підтвердження готовності — гравець підтверджує свою готовність до початку гри; система оновлює статус гравця та перевіряє готовність усіх учасників перед початком гри.

- Взаємодія в грі — гравець виконує дії, в залежності від своєї ролі; система перевіряє стадію гри та роль гравця та виконує дію, а разі успішної перевірки.
- Чат в грі — гравець запитує з’єднання з іншими гравцями; система надсилає запит до інших гравців.

Рисунок *3.2 — Діаграма використання Player*

Діаграма використання для гравця(див. Рис. 3.3):

Актор: Власник лобі (Lobby owner)

Сценарії використання:

- Видалення лобі — власник видаляє лобі; система видаляє документ та повідомляє усіх учасників про видалення лобі.
- Вихід з лобі — учасник виходить з лобі; система видаляє учасника та повідомляє інших учасників.
- Управління учасниками лобі — власник назначає нового власника лобі; система змінює власника та повідомляє інших учасників.

- Початок гри — власник починає гру; система створює нову гру, призначає кожному учаснику роль та повідомляє усіх учасників лобі.

Рисунок *3.3 — Діаграма використання Lobby owner*

3.3 Програмна реалізація серверної частини

3.3.1 Огляд серверної архітектури

Основні компоненти серверної архітектури включають використання NestJS для розробки серверного додатку, MongoDB як базу даних та Mongoose для роботи з базою даних.

1.NestJS — прогресивна платформа для створення ефективних і масштабованих серверних додатків Node.js. Вона використовує TypeScript і натхнена концепціями та дизайном Angular, що забезпечує модульність і зручність у розробці.
Основні особливості: висока продуктивність, модульність, підтримка TypeScript, простота інтеграції з різними базами даних і фреймворками.

2.MongoDB — це документо-орієнтована NoSQL база даних, яка зберігає дані у форматі BSON (бінарний аналог JSON). Вона ідеально підходить для додатків, що вимагають гнучкості та швидкої обробки великих обсягів даних. Основні особливості: Гнучка схема даних, висока продуктивність, масштабованість, підтримка реплікації та шардингу.

3.Mongoose — це об'єктно-документний мапер (ODM) для Node.js і MongoDB, який надає інструменти для роботи з документами в MongoDB, включаючи валідацію схем, побудову запитів і кастомну бізнес-логіку. Основні особливості: Схеми з валідацією, потужні інструменти для створення запитів, підтримка middleware і hooks, інтеграція з MongoDB.

Архітектурний патерн.

Для реалізації серверної частини використовується патерн Controller-Service-Model, який є варіацією архітектурного патерну MVC (Model-View-Controller). Цей патерн забезпечує чітке розділення відповідальностей між різними компонентами додатку:

1. Контролери (Controllers).
   - Відповідають за обробку HTTP-запитів та взаємодію з клієнтами.
   - Викликають методи сервісів для виконання бізнес-логіки.
2. Сервіси (Services).
   - Містять основну бізнес-логіку додатку.
   - Взаємодіють з моделями для доступу до бази даних та виконання необхідних операцій.
3. Моделі (Models).
   - Відповідають за структуру даних та взаємодію з базою даних.
   - Використовують Mongoose для визначення схем і виконання CRUD-операцій.

3.3.2 Реалізація модуля IAM

Модуль IAM (Identity and Access Management) відповідає за управління ідентифікацією та доступом користувачів у системі. Основні функції цього модуля включають реєстрацію та автентифікацію.

Цей модуль містить підмодуль Authentication, що відповідає за автентифікацію користувачів (див. Рис. 3.4).

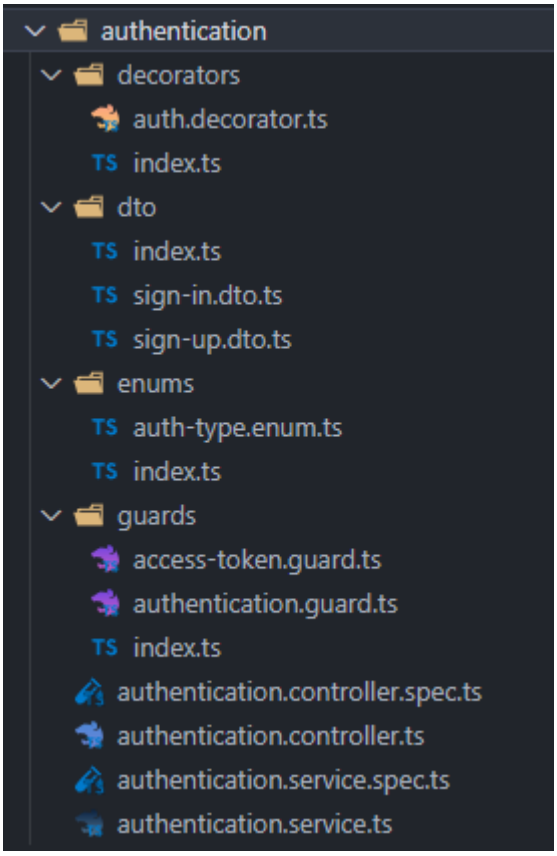

Рисунок *3.4 — «Структура модуля Authentication»*

- Декоратор Auth: відповідає за визначення типу автентифікації.
- dto (Data Transfer Objects) файли: визначають структури даних, які використовуються для передачі даних між клієнтом і сервером.
- Перелік (enum) AuthType: тип автентифікації.

- Охоронець AccessTokenGuard: використовується для перевірки валідності токена доступу. Він може працювати як з HTTP-запитами, так і з WebSocket-з'єднаннями.
- Охоронець AuthenticationGuard: використовується для визначення типу аутентифікації та делегування перевірки відповідному охоронцю.
- Контролер AuthenticationController: обробляє запити автентифікації.
- Сервіс AuthenticationService: відповідає за логіку автентифікації.

Модуль IAM також містить наступні компоненти (див. Рис. 3.5).

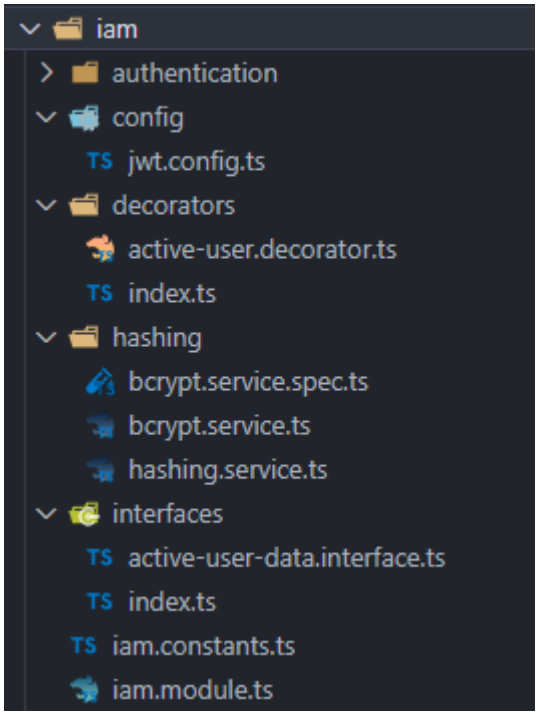

Рисунок *3.5 — «Структура модуля IAM»*

- Декоратор ActiveUser: вилучає із запиту поле з інформацією про користувача.
- Сервіс BcryptService: відповідає за хешування паролів.

3.3.3 Реалізація модуля Users

Модуль Users відповідає за управління існуючими записами користувачів (див. Рис. 3.6). Основні функції цього модуля включають перегляд, редагування та видалення профілю.

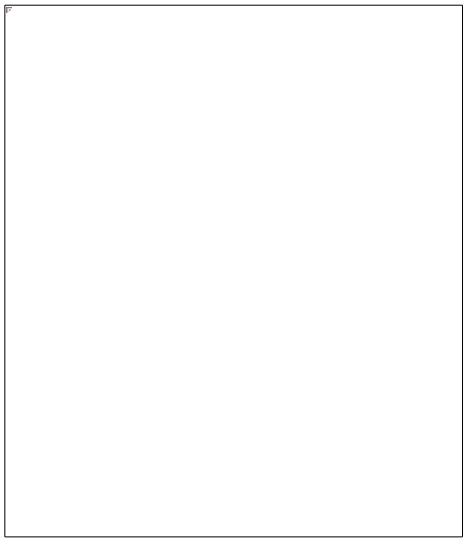

Рисунок *3.6 — «Структура модуля Users»*

- Сутність User: модель визначає структуру даних користувача, що зберігається у базі даних.
- Контролер ProfileController: обробляє запити профілю користувача.
- Сервіс UsersService: відповідає за логіку отримання даних, редагування та видалення користувачів.

3.3.4 Реалізація модуля Players

Модуль Players відповідає за створення та управління гравцями (див. Рис. 3.7).

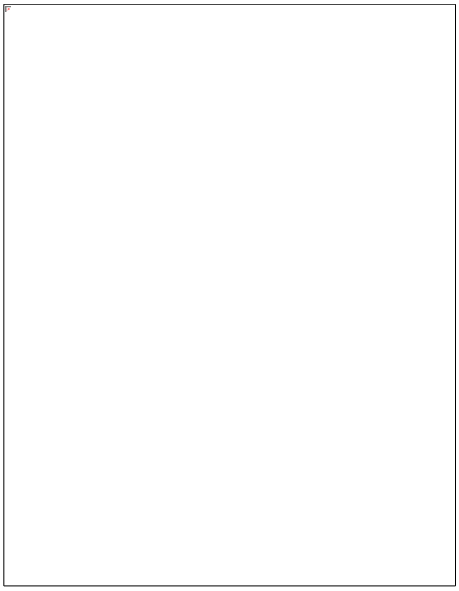

Рисунок *3.7 — «Структура модуля Players»*

- Сутність Player: модель визначає структуру даних гравця, що пов’язана с автентифікованим користувачем та зберігається у базі даних.
- Перелік Role: роль гравця у грі.
- Перелік Team: команда, за яку грає гравець.
- Сервіс PlayersService: відповідає за логіку отримання даних, редагування та видалення гравців.

3.3.4 Реалізація модуля Lobby

Модуль Lobby відповідає за створення та управління лобі (див. Рис. 3.8). Основні функції цього модуля включають перегляд, редагування та видалення лобі.

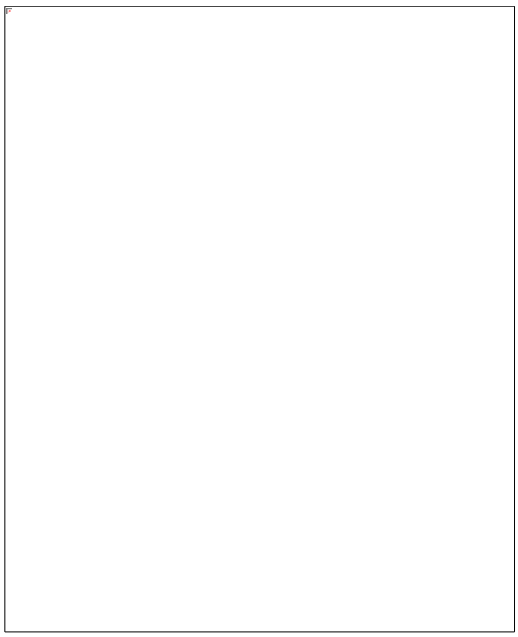

Рисунок *3.8 — «Структура модуля Lobby»*

- Сутність Lobby: модель визначає структуру даних лобі, що містить гравців та зберігається у базі даних.
- Контролер LobbyController: обробляє запити лобі та налаштувань.
- Шлюз LobbyGateway: обробляє повідомлення лобі.
- Сервіс LobbySettingsService: відповідає за зберігання налаштувань лобі.
- Сервіс LobbyService: відповідає за логіку обробки подій лобі.

3.3.5 Реалізація модуля Game

Модуль Game відповідає за створення та управління грою (див. Рис. 3.9).

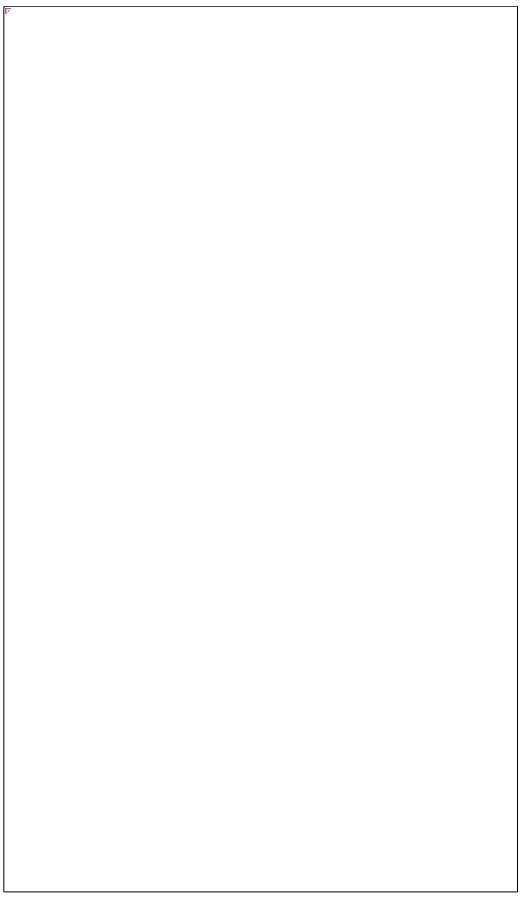

Рисунок *3.9 — «Структура модуля Game»*

- Сутність Game: модель визначає структуру даних гри, що зберігається у базі даних.
- Перелік GameStage: стадія гри.
- Контролер GameController: обробляє запити поточної гри.
- Шлюз GameGateway: обробляє повідомлення гри.
- Сервіс GameService: відповідає за логіку обробки подій гри.

3.4 Програмна реалізація клієнтської частини

3.4.1 Огляд клієнтської архітектури

Клієнтська архітектура гри «Мафія», реалізована з використанням React, включає в себе наступні компоненти та технології:

1. React: бібліотека JavaScript для створення інтерфейсів користувача, яка дозволяє розбити інтерфейс на компоненти та забезпечити їх декларативний інтерфейс. Вона забезпечує швидку та ефективну роботу завдяки використанню віртуального DOM.

2. Redux: це контейнер стану для JavaScript-додатків, який допомагає керувати станом у всьому додатку. Він забезпечує передбачуваність і простоту управління станом додатку.

3. WebSocket: протокол зв'язку, який дозволяє встановлювати двостороннє з'єднання між клієнтом і сервером через одне з'єднання TCP. Використання WebSocket дозволяє реалізувати ігрову логіку в реальному часі, забезпечуючи миттєву взаємодію між гравцями.

4. React Router: React Router - це бібліотека маршрутизації для React, яка дозволяє керувати маршрутами у веб-додатку. Вона дозволяє створювати односторінкові додатки та ефективно керувати навігацією користувача.

5. WebRTC: WebRTC - це технологія, яка дозволяє реалізувати відео- та аудіозв'язок прямо у веб-браузері без необхідності встановлення додаткових програм. Використання WebRTC дозволяє гравцям взаємодіяти між собою в реальному часі без затримок і перешкод.

Для реалізації клієнтської частини використовується патерн Модель-Вид-Контролер (MVC) — це архітектурний патерн, який використовується для побудови веб-додатків з розділеною відповідальністю. Він допомагає відокремити бізнес логіку, користувацький інтерфейс та логіку взаємодії з користувачем, щоб забезпечити більшу структурованість та легкість розробки.

1. Модель (Model).

- Представляє собою частину додатку, яка відповідає за бізнес логіку та обробку даних.
- Взаємодіє з базою даних, обробляє запити та повертає дані контролеру для відображення.

2. Вид (View).

- Відповідає за представлення даних користувачу.
- Отримує дані від контролера та відображає їх у вигляді веб-сторінки.

3. Контролер (Controller).

- Відповідає за обробку запитів користувача та взаємодію з моделлю для отримання необхідних даних.
- Отримує вхідні дані від користувацького інтерфейсу, обробляє їх та передає відповідні запити до моделі для подальшої обробки.
- Після отримання відповіді від моделі передає дані для відображення результатів користувачу.

3.4.2 Реалізація компонентів React

Компонентна архітектура в React дозволяє створювати багаторазові, ізольовані блоки коду, що полегшує розробку, тестування та підтримку проєкту. У цьому розділі буде розглянуто реалізацію основних компонентів для гри «Мафія».

1. CreateLobbyModal: модальне вікно з текстовим полем та кнопкою для створення нового лобі (див. Рис. 3.10).

Рисунок *3.10 — Компонент CreateLobbyModal*

2. LobbyList: список існуючих лобі (див. Рис. 3.11).

Рисунок *3.11 — Компонент LobbyList*

3. LobbyUserList: список гравців у лобі (див. Рис. 3.12).

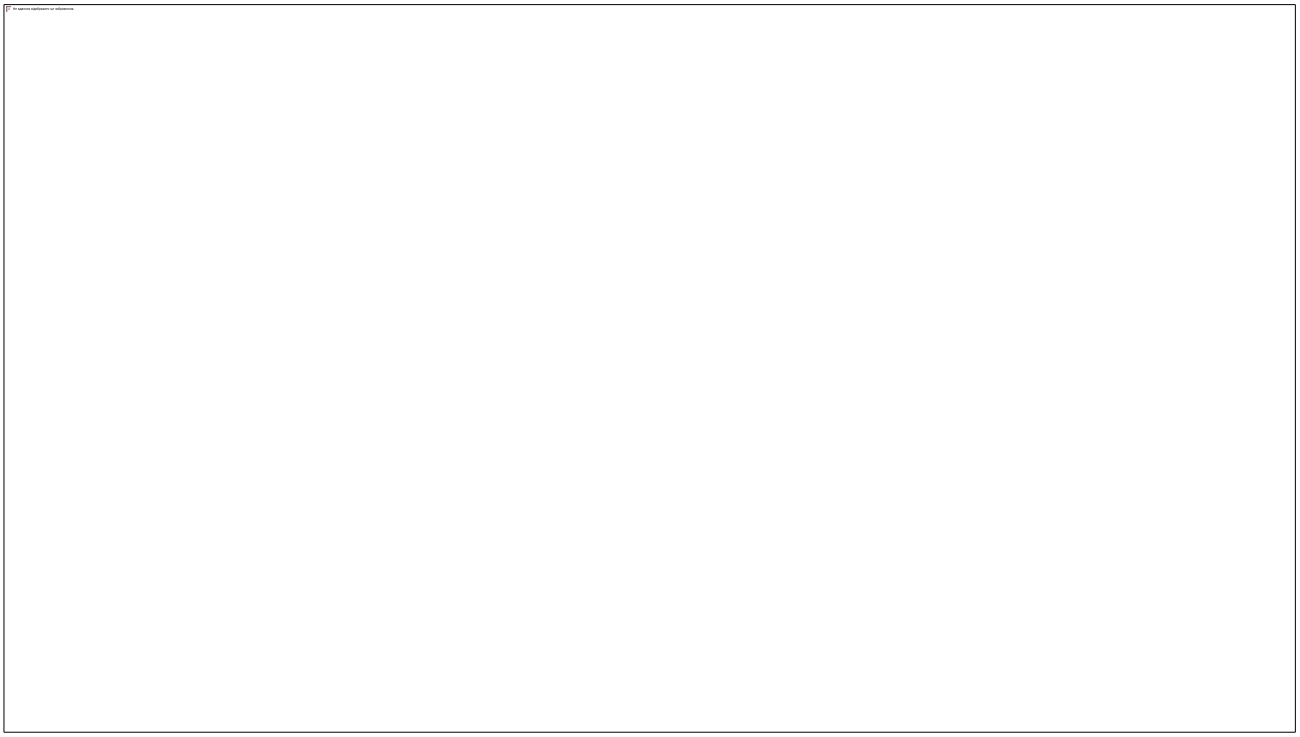

Рисунок *3.12 — Компонент LobbyUserList*

4. PlayerCard: картка гравця під час гри з відображенням відео (див. Рис. 3.13).

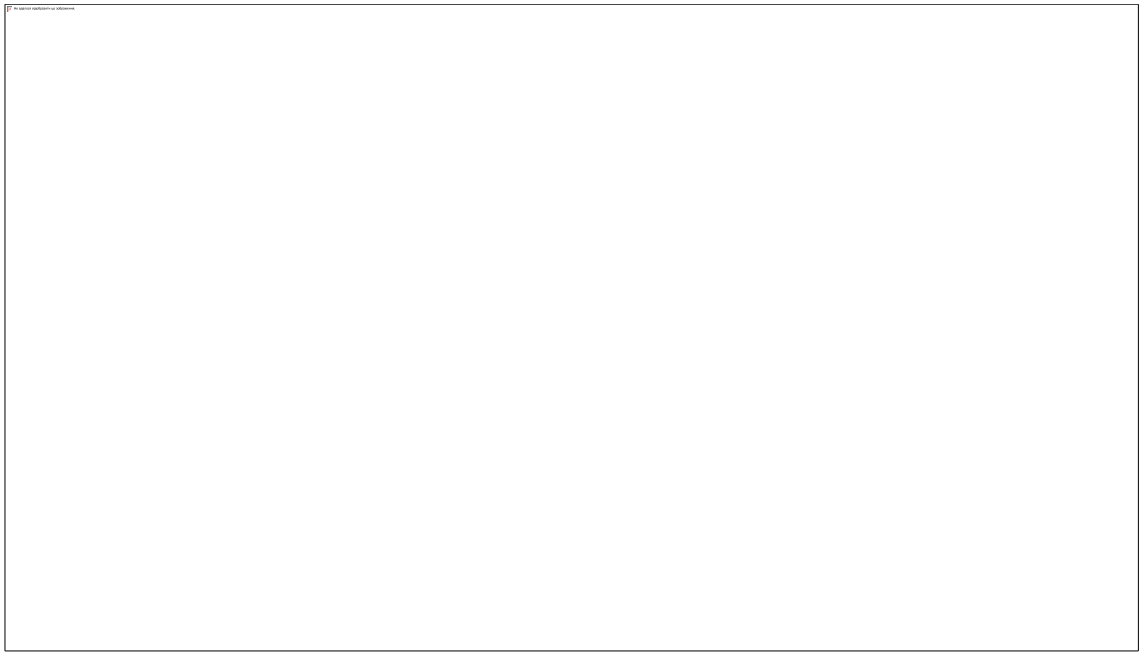

Рисунок *3.13 — Компонент PlayerCard*

Основні сторінки:

1. LoginPage: сторінка для автентифікації користувача (див. Рис. 3.14).

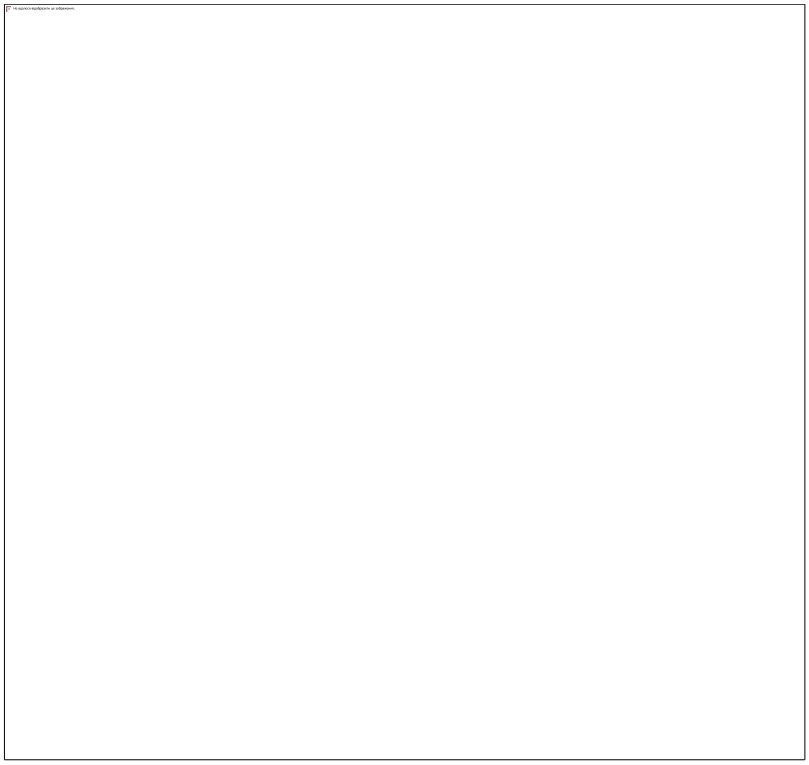

Рисунок *3.14 — Сторінка LoginPage*

2.RegisterPage: сторінка для створення облікового запису користувача (див. Рис. 3.15).

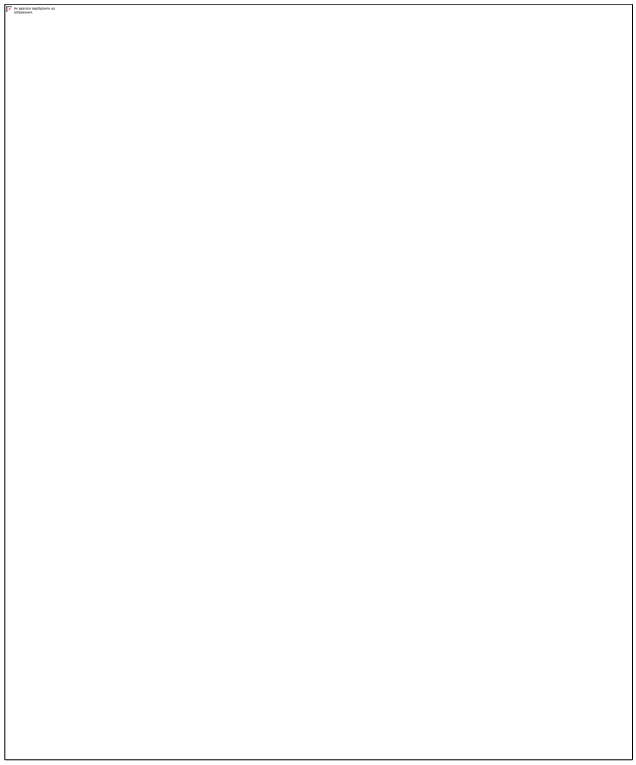

Рисунок *3.15 — Сторінка RegisterPage*

3.HomePage: сторінка для створення лобі та приєднання до існуючих лобі (див. Рис. 3.16).

Рисунок *3.16 — Сторінка HomePage*

4.LobbyPage: сторінка існуючого лобі з можливістю маніпуляцією гравцями та лобі (див. Рис. 3.17).

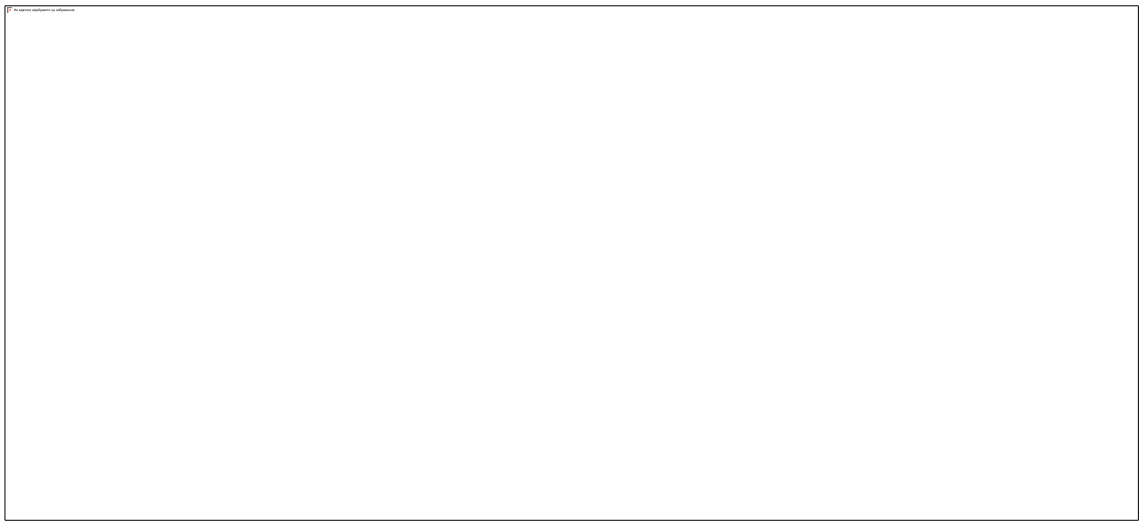

Рисунок *3.17 — Сторінка LobbyPage*

5.GamePage: сторінка процесу гри, яка складається з 3 етапів:

5.1.Розподіл ролей: гравці отримують свої ролі та підтверджують готовність (див. Рис. 3.18).

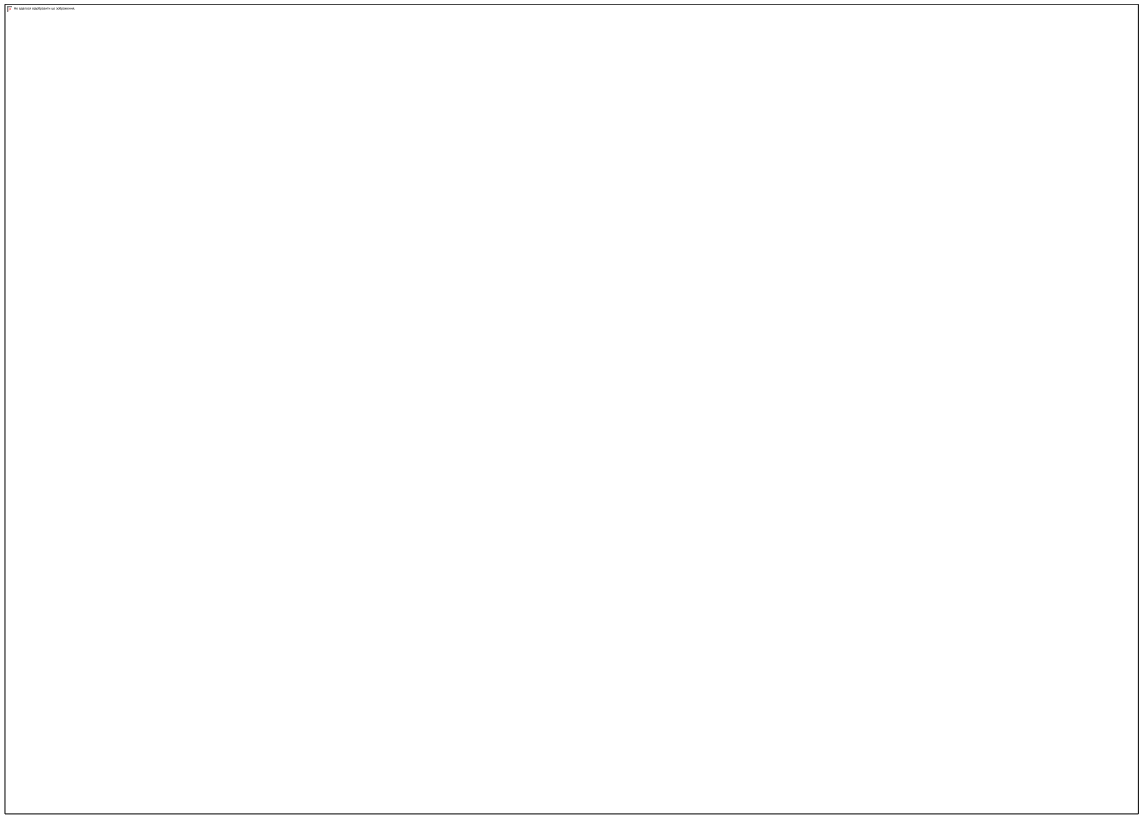

Рисунок *3.18 — Сторінка GamePage (розподіл ролей)*

5.2. Процес гри: гравці по черзі виконують дії своєю ролі та приймають участь у обговореннях та голосуваннях (див. Рис. 3.19).

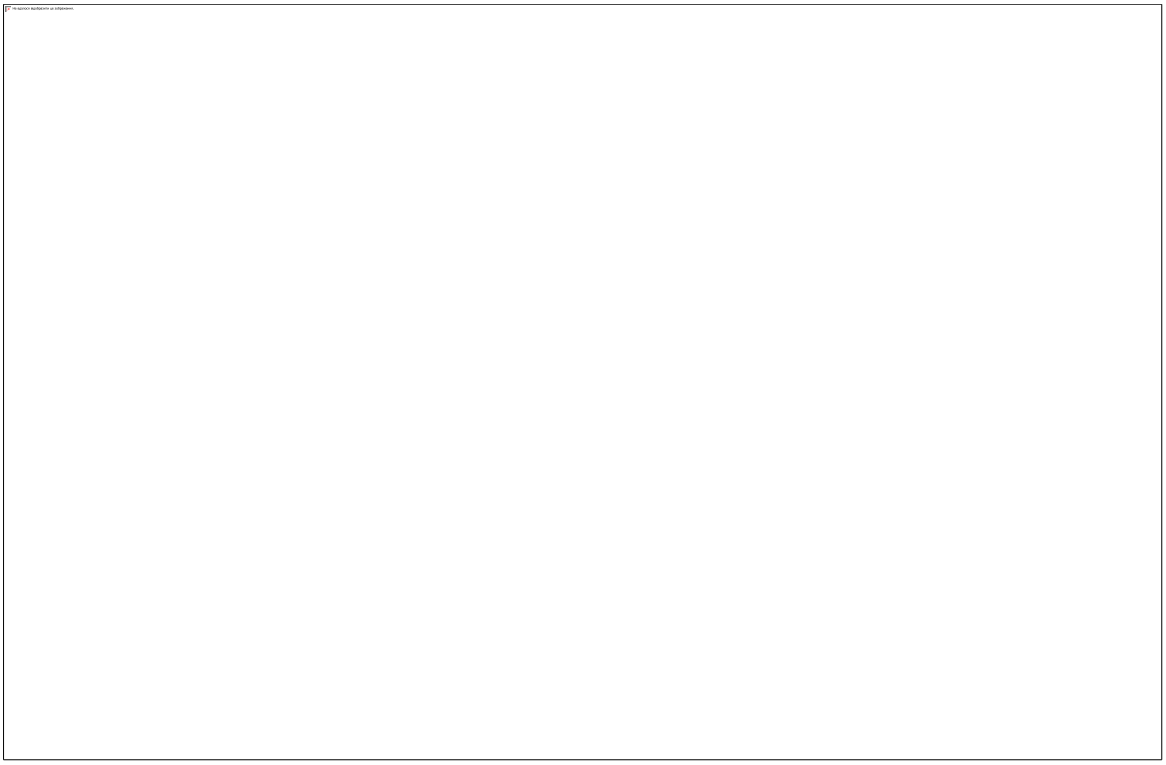

Рисунок *3.19 — Сторінка GamePage (розподіл ролей)*

5.3. Гра завершена: відображається команда-переможець гри (див. Рис. 3.20).

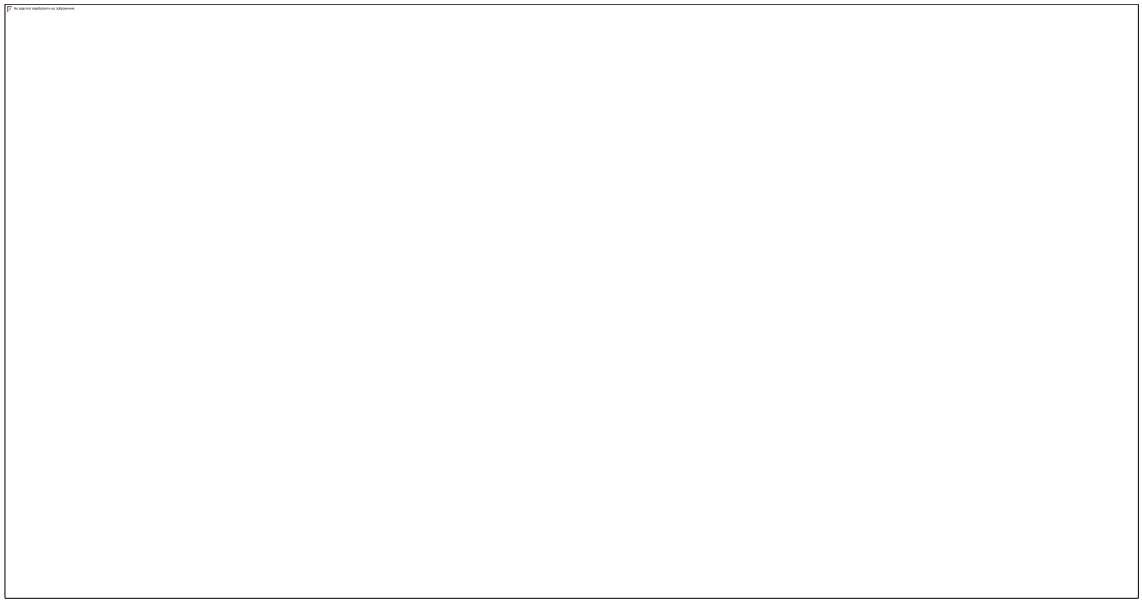

Рисунок *3.20 — Сторінка GamePage (розподіл ролей)*

3.4.3 Використання Redux та Redux-Saga для управлянні станом

Для ефективного управління станом у клієнтській частині додатку «Мафія» використовується Redux у поєднанні з Redux-Saga. Ці технології дозволяють централізувати та впорядкувати стан додатку, а також керувати побічними ефектами, такими як асинхронні запити до сервера.

В рамках управління станом за допомогою Redux, слайси (slices) стейту використовуються для організації коду в більш модульний та зручний для підтримки спосіб. Слайси дозволяють розділяти стан додатку на логічні частини, кожна з яких відповідає за певний аспект додатку:

1. userSlice: слайс з інформацією про автентифікованого користувача
    - user — поле з інформацією про користувача
    - accessToken — JSON Web Token автентифікованого користувача
2. lobbySlice: слайс з інформацією про поточне та існуючі лобі
    - lobbies — список існуючих лобі
    - currentLobbyId — ID поточного лобі
    - settings — налаштування лобі

3. gameSlice: слайс з інформацією про гру
    - currentPlayer — інформація про гравця
    - checkedPlayerId — ID гравця, якого було перевірено
    - currentCheck — результат останньої перевірки

Redux-Saga використовується для управління побічними ефектами у Redux-додатках, яка використовує генератори для написання асинхронного коду, що робить його більш читабельним та керованим:

1. userSaga
    - signInWorker — відповідає за автентифікацію користувача
    - signUpWorker — відповідає за створення облікового запису користувача
    - editProfileWorker — відповідає за редагування даних про користувача
    - logoutWorker — відповідає за вихід з облікового запису користувача
2. lobbySaga
    - getLobbiesWorker — відповідає за отримання списку існуючих лобі
    - getLobbySettingsWorker — відповідає за отримання налаштувань лобі
3. gameSaga
    - getPlayerWorker — відповідає за отримання інформації про поточного гравця

3.4.4 Реалізація кастомних хуків для забезпечення логіки гри

Кастомні хуки (custom hooks) у React дозволяють інкапсулювати логіку, що повторюється, та використовувати її в різних компонентах. У грі «Мафія», використання сокетів для взаємодії у реальному часі між клієнтами є критичним

для інтерактивності гри. Для роботи з сокетами буде використовуватися бібліотека socket.io-client та реалізовано кастомні хуки для забезпечення цієї логіки.

- useLobbiesSocket — хук на лістингу 3.1 забезпечує управління підключенням до сокетів і обробки подій, пов’язаних з списком лобі.

Лістинг *3.1 Реалізація хука useLobbiesSocket*

```
export const useLobbiesSocket = () => {
  const navigate = useNavigate();
  const { accessToken } = useAppSelector(state =>
state.user);
  const { currentLobbyId } = useAppSelector(state =>
state.lobby);
  const dispatch = useAppDispatch();
  const addListeners = useCallback(() => {
    SocketApi.lobby.onCreated(data => {
      dispatch(lobbySlice.actions.createLobby(data));
    });
    SocketApi.lobby.onRemoved(data => {
      dispatch(lobbySlice.actions.removeLobby(data));
    });
  }, [dispatch]);
  const handleCreateLobby = useCallback(
    (name: string) => {
      if (!name) return;
      SocketApi.lobby.create({ name }, lobby => {
        navigate(Paths.Lobby, {
          state: { lobby },
        });
      });
    },
    [navigate],
  );
  useMount(() => {
    SocketApi.lobby.createConnection(accessToken);
    addListeners();
    if (currentLobbyId) {
      SocketApi.lobby.leave({ lobbyId: currentLobbyId });
```

```
        dispatch(lobbySlice.actions.setCurrentLobbyId(''));
      }
      return () => {
        SocketApi.lobby.disconnect();
      };
    });
    return { handleCreateLobby };
  };
```

- useLobbySocket— хук на лістингу 3.2 забезпечує управління підключенням до сокетів і обробки подій, пов’язаних з поточним лобі.

Лістинг *3.2 Реалізація хука useLobbySocket*

```
  export const useLobbySocket = (lobby: Nullable<Lobby>,
setCurrentLobby: (lobby: Nullable<Lobby>) => void) => {
    const navigate = useNavigate();
    const { accessToken } = useAppSelector(state =>
state.user);
    const dispatch = useAppDispatch();
    const addListeners = useCallback(() => {
      SocketApi.lobby.onJoined(data => {
        dispatch(gameSlice.actions.getPlayer());
        setCurrentLobby(data);
      });
      SocketApi.lobby.onLeft(setCurrentLobby);
      SocketApi.lobby.onRemoved(data => {
        if (data.lobbyId === lobby?.id) {
          navigate(Paths.Root);
        }
      });
      SocketApi.lobby.onGameStarted(game => {
        navigate(`/${Paths.Game}`, {
          state: { game },
        });
      });
      SocketApi.lobby.onOwnerChanged(setCurrentLobby);
    }, [dispatch, lobby?.id, navigate, setCurrentLobby]);
    const handleJoinLobby = useCallback(() => {
      if (!lobby?.id) return;
```

```
        SocketApi.lobby.join({ lobbyId: lobby.id },
setCurrentLobby);

dispatch(lobbySlice.actions.setCurrentLobbyId(lobby.id));
      }, [lobby?.id, setCurrentLobby, dispatch]);
      const handleLeaveLobby = useCallback(() => {
        if (!lobby?.id) return;
        SocketApi.lobby.leave({ lobbyId: lobby.id });
        dispatch(lobbySlice.actions.setCurrentLobbyId(''));
        navigate(Paths.Root);
      }, [dispatch, lobby?.id, navigate]);
      const handleRemoveLobby = useCallback(() => {
        if (!lobby?.id) return;
        SocketApi.lobby.remove({ lobbyId: lobby.id });
      }, [lobby?.id]);
      const handleChangeOwner = useCallback(
        (playerId: string) => {
          if (!lobby?.id) return;
          SocketApi.lobby.changeOwner({ lobbyId: lobby.id,
playerId });
        },
        [lobby?.id],
      );
      const handleStartGame = useCallback(() => {
        if (!lobby?.id) return;
        SocketApi.lobby.startGame({ lobbyId: lobby.id });
      }, [lobby?.id]);
      const handleBeforeUnload = useCallback((event:
BeforeUnloadEvent) => {
        event.preventDefault();
      }, []);
      const handleUnload = useCallback(() => {
        if (lobby) {
          SocketApi.lobby.leave({ lobbyId: lobby.id });
        }
      }, [lobby]);
      useBeforeUnload(handleBeforeUnload);
      useMount(() => {
        SocketApi.lobby.createConnection(accessToken);
        addListeners();
```

```
        handleJoinLobby();
        const heartbeat = setInterval(() =>
SocketApi.lobby.heartbeat(), ONE_MINUTE);
        window.addEventListener('unload', handleUnload);
        return () => {
          clearInterval(heartbeat);
          SocketApi.lobby.disconnect();
        };
      });
      return { handleLeaveLobby, handleRemoveLobby,
handleChangeOwner, handleStartGame };
    };
```

- useGameSocket— хук на лістингу 3.3 забезпечує управління підключенням до сокетів і обробки подій, пов’язаних з поточною грою.

Лістинг *3.3 Реалізація хука useGameSocket*

```
    export const useGameSocket = (currentGame:
Nullable<Game>, setCurrentGame: (game: Game) => void) => {
      const { accessToken } = useAppSelector(state =>
state.user);
      const { currentPlayer: player } = useAppSelector(state
=> state.game);
      const dispatch = useAppDispatch();
      const addListeners = useCallback(() => {
        SocketApi.game.onPlayerReady(setCurrentGame);
        SocketApi.game.onStarted(setCurrentGame);
      }, [setCurrentGame]);
      const handleReady = useCallback(() => {
        if (!currentGame?.id) return;
        SocketApi.game.ready({ gameId: currentGame.id }, ({
game, player: updatedPlayer }) => {
          setCurrentGame(game);

dispatch(gameSlice.actions.setPlayer(updatedPlayer));
        });
      }, [currentGame?.id, setCurrentGame, dispatch]);
      const handleBeforeUnload = useCallback((event:
BeforeUnloadEvent) => {
```

```
        event.preventDefault();
      }, []);
      useBeforeUnload(handleBeforeUnload);
      useMount(() => {
        if (currentGame) {
          SocketApi.game.createConnection(accessToken,
currentGame?.id ?? '', player?.id ?? '');
          addListeners();
          dispatch(gameSlice.actions.getPlayer());
        }
        const heartbeat = setInterval(() =>
SocketApi.game.heartbeat(), ONE_MINUTE);
        return () => {
          clearInterval(heartbeat);
          SocketApi.game.disconnect();
        };
      });

      return { handleReady };
    };
```

- useGameChat— хук на лістингу 3.3 забезпечує керування чатом гри та передачею аудіо- та відеопотоків між гравцями в реальному часі.

Лістинг *3.3 Реалізація хука useGameChat*

```
    export const useGameChat = (
      isGame: boolean,
      stream: MediaStream | undefined,
      setStream: (stream: MediaStream | undefined) => void,
      setPlayersMedia:
Dispatch<SetStateAction<PlayerMedia[]>>,
    ) => {
      const { currentPlayer: player } = useAppSelector(state
=> state.game);
      const peersRef = useRef<{ peerId: string; peer:
Peer.Instance }[]>([]);
      const streamRef = useRef<Nullable<MediaStream>>(null);
      const offerPeer = useCallback(
        async (data: Player) => {
          if (!player || !stream) return;
```

```
          const peer = new Peer({
            initiator: true,
            trickle: false,
            stream,
          });
          peer.on('signal', signal => {
            SocketApi.game.offerSignal({ toPlayer: data.id,
signal });
          });
          peer.on('stream', peerStream => {
            setPlayersMedia(prev => [
              ...prev,
              {
                playerId: data.id,
                stream: peerStream,
              },
            ]);
          });
          peersRef.current.push({
            peerId: data.id,
            peer,
          });
        },
        [player, setPlayersMedia, stream],
      );
      const answerPeer = useCallback(
        ({ fromPlayer, signal: incomingSignal }:
SignalResponse) => {
          if (!stream || !player) return;
          const peer = new Peer({
            initiator: false,
            trickle: false,
            stream,
          });
          peer.on('signal', signal => {
            SocketApi.game.answerSignal({ toPlayer:
fromPlayer, signal });
          });
          peer.on('stream', peerStream => {
            setPlayersMedia(prev => [
```

```
            ...prev,
            {
              playerId: fromPlayer,
              stream: peerStream,
            },
          ]);
        });
        peer.signal(incomingSignal);
        peersRef.current.push({
          peerId: fromPlayer,
          peer,
        });
      },
      [player, setPlayersMedia, stream],
    );
    const confirmPeer = useCallback(({ fromPlayer, signal
}: SignalResponse) => {
      const item = find(peersRef.current, p => p.peerId ===
fromPlayer);
      item?.peer.signal(signal);
    }, []);
    const getMediaStream = useCallback(async () => {
      try {
        const mediaStream = await
navigator.mediaDevices.getUserMedia({
          video: true,
          audio: true,
        });
        setStream(mediaStream);
        streamRef.current = mediaStream;
      } catch (error) {
        console.error(error);
      }
    }, [setStream]);
    useEffect(() => {
      if (stream) {

SocketApi.game.onCreatePlayersConnection(offerPeer);
        SocketApi.game.onOfferSignal(answerPeer);
        SocketApi.game.onAnswerSignal(confirmPeer);
```

```
        }
        // eslint-disable-next-line react-hooks/exhaustive-
deps
      }, [stream]);
      useMount(() => {
        if (isGame) {
          getMediaStream();
        }
        return () => {
          streamRef.current?.getTracks().forEach(track =>
track.stop());
        };
      });
    };
```

- useGameProcess— хук на лістингу 3.4 забезпечує управління різними етапами гри та взаємодії гравців під час гри.

Лістинг *3.4 Реалізація хука useGameProcess*

```
    export const useGameProcess = (currentGame: Game,
setCurrentGame: (game: Game) => void) => {
      const { currentPlayer } = useAppSelector(state =>
state.game);
      const dispatch = useAppDispatch();
      const addListeners = useCallback(() => {
        SocketApi.game.onGameUpdated(game => {
          setCurrentGame(game);

dispatch(gameSlice.actions.setPlayer(game.players.find(player
=> player.id === currentPlayer?.id) ?? null));
          dispatch(gameSlice.actions.setCheckedPlayer(''));
        });
      }, [currentPlayer?.id, dispatch, setCurrentGame]);
      const handleMafiaMet = useCallback(() => {
        SocketApi.game.mafiaMet({ gameId: currentGame.id });
      }, [currentGame]);
      const handleFinishSpeech = useCallback(() => {
        SocketApi.game.finishSpeech({ gameId: currentGame.id
});
      }, [currentGame]);
```

```
      const handleDonCheck = useCallback(
        (playerId: string) => {
          SocketApi.game.donCheck({ gameId: currentGame.id,
playerId }, checkResult => {

dispatch(gameSlice.actions.setCheckedPlayer(playerId));

dispatch(gameSlice.actions.setCurrentCheck(checkResult ? 'yes'
: 'no'));
          });
        },
        [currentGame.id, dispatch],
      );
      const handleMafiaVote = useCallback(
        (playerId: string) => {
          SocketApi.game.mafiaVote({ gameId: currentGame.id,
playerId });
        },
        [currentGame.id],
      );
      const handleSheriffCheck = useCallback(
        (playerId: string) => {
          SocketApi.game.sheriffCheck({ gameId:
currentGame.id, playerId }, checkResult => {

dispatch(gameSlice.actions.setCheckedPlayer(playerId));

dispatch(gameSlice.actions.setCurrentCheck(checkResult ? 'yes'
: 'no'));
          });
        },
        [currentGame.id, dispatch],
      );
      const handleDayVote = useCallback(
        (playerId: string) => {
          SocketApi.game.dayVote({ gameId: currentGame.id,
playerId });

dispatch(gameSlice.actions.setCheckedPlayer(playerId));
        },
```

```
        [currentGame.id, dispatch],
      );
      useMount(() => {
        addListeners();
      });
      return { handleMafiaMet, handleFinishSpeech,
handleDonCheck, handleMafiaVote, handleSheriffCheck,
handleDayVote };
    };
```

ВИСНОВКИ

У кваліфікаційній роботі розглянуто розробку програмного забезпечення комп’ютерної рольової гри з використанням технологій React та Node.js на прикладі гри «Мафія». Основними етапами виконаної роботи були аналіз предметної області, дослідження сучасних програмних засобів реалізації, а також безпосередня розробка гри. Основні висновки за результатами роботи:

1. Гра «Мафія» має чітко визначені правила та структуру, що визначає її особливості як рольової гри. Було детально розглянуто основні аспекти гри, що дозволило зрозуміти вимоги до програмної реалізації.
2. Виконано аналіз існуючих веб-додатків, що імітують гру «Мафія», що дозволило визначити їх переваги та недоліки, а також виявити можливості для покращення.
3. Проаналізовано різні методи рендерингу веб-додатків. Це дозволило визначити оптимальний підхід для розробки гри.
4. Розроблено архітектуру веб-додатку, яка включає серверну та клієнтську частини, що взаємодіють через REST API.
5. Спроектовано базу даних для зберігання інформації про користувачів, ігри, лобі та інші сутності гри «Мафія».
6. Реалізовано серверну частину додатку з використанням NestJS, яка включає модулі для управління автентифікацією (IAM), користувачами, гравцями, лобі та іграми.
7. Розроблено клієнтську частину з використанням React, зокрема реалізовано компоненти для відображення інтерфейсу гри, управління станом за допомогою Redux та Redux-Saga, а також кастомні хуки для забезпечення логіки гри.
8. Створено повнофункціональний веб-додаток, що реалізує гру «Мафія», який забезпечує інтерактивний та зручний інтерфейс для користувачів.

9. Забезпечено належну продуктивність та масштабованість додатку завдяки обраним технологіям та архітектурним рішенням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jazayeri, M. Modern Web Application Development / M. Jazayeri, CS. Mesnage, J. Rose // Emerging Methods, Technologies, and Process Management in Software Engineering. – John Wiley and Sons Inc., 2007. C. 131-147.

2. Способи генерації сторінок URL: https://dou.ua/forums/topic/41585 (дата звернення 02.06.2024 р.).

3. Веб-додаток і його характеристики URL: https://www.centum-d.com/veb-dodatok-yogo-harakteristiki (дата звернення 02.06.2024 р.).

4. Електронний документ «AUTOMATION AND DEVELOPMENT OF ELECTRONIC DEVICES» ADED-2022 Part 2 URL: https://openarchive.nure.ua/server/api/core/bitstreams/689a8d92-140e-4825-b47f-3b07b085929d/content (дата звернення 02.06.2024 р.).

5. «Modern Web-Development using ReactJS», Sanchit Aggarwal et al. International Journal of Recent Research Aspects ISSN: 2349-7688,Vol. 5, Issue 1, March 2018, pp. 133-137 URL: http://ijrra.net/Vol5issue1/IJRRA-05-01-27.pdf (дата звернення 02.06.2024 р.).

6. Flux: An Application Architecture for React URL: https://legacy.reactjs.org/blog/2014/05/06/flux.html (дата звернення 02.06.2024 р.).

7. Virtual DOM and Real DOM URL: https://medium.com/@surksha8/virtual-dom-and-real-dom-understanding-the-differences-da8f3fab4261 (дата звернення 02.06.2024 р.).

8. Vue.js — навіщо він нам? URL: https://medium.com/@jstify.community/vue-js-%D0%BD%D0%B0%D0%B2%D1%96%D1%89%D0%BE-%D0%B2%D1%96%D0%BD-%D0%BD%D0%B0%D0%BC-981b5e17af39 (дата звернення 02.06.2024 р.).

9. Angular and Benefits of Angular URL: https://medium.com/sliit-foss/angular-and-benefits-of-angular-c44603b40460 (дата звернення 02.06.2024 р.).

10. Python Statistics 2023 URL: https://leftronic.com/blog/python-statistics (дата звернення 02.06.2024 р.).

11. ЩО ТАКЕ МОВА ПРОГРАМУВАННЯ PYTHON? URL: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-jazik-programmirovanija-python (дата звернення 02.06.2024 р.).

12. Електроний документ Python: The Most Advanced Programming Language for Computer Science Applications. URL: https://www.scitepress.org/Papers/2020/103079/103079.pdf (дата звернення 02.06.2024 р.).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Хлівний Андрій Анатолійович, студент _4_ курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz20bd-119@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Програмне забезпечення комп'ютерної рольової гри з використанням React та Node.js**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-систем, а також на архівування моєї роботи в базі даних цієї системи.

Дата 14.06.2024 Підпис _____ Хлівний Андрій Анатолійович
(студент)

Дата 14.06.2024 Підпис _____ Безверхий Анатолій Ігорович
(науковий керівник)