Міністерство освіти і науки України

Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Шеліпова Кирила Ігоровича*
(ПІБ)

академічної групи *122-17-2*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка веб-додатку для пошуку товаришів по команді в іграх на базі фреймворка React*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Сироткіна О.І.*			
розділів:				
спеціальний	*доц. Сироткіна О.І.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2021

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2021 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-17-2* *Шеліпова Кирила Ігоровича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка веб-додатку для пошуку товаришів по команді в іграх на базі фреймворка React*

затверджена наказом ректора НТУ «ДП» від *07.06.2021* № *317-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2021 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2021 р.***

Завдання видав _____ *доц. Сироткіна О.І.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Шеліпов К. І.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2021 р.*

Термін подання кваліфікаційної роботи до ЕК: *11.06.2021 р.*

РЕФЕРАТ

Пояснювальна записка: 93с., 51 рис., 3 дод., 25 джерела.

Об'єкт розробки: веб-додаток у вигляді веб-сайту з оптимізацію під мобільні пристрої для пошуку товаришів по команді в різних іграх.

Мета кваліфікаційної роботи: створити зручний простір для пошуку однодумців та товаришів по команді для різних ігор, орієнтуючись на систему взаємного вибору.

У вступі розглядається сучасний стан проблеми, конкретизується мета кваліфікаційної роботи, актуальність та галузь її застосування, уточнюється постановка завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформи для розробки, виконано проєктування і розробка програми, описана робота програми, алгоритм і структура її функціонування, а також виклик та завантаження програми, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення полягає у веб-додатку, що дає змогу шукати товаришів по команді для різних ігор.

Актуальність інформаційної системи визначається великим попитом на подібні розробки, що дають можливості для пошуку товаришів по команді.

Список ключових слів: КОМП’ЮТЕР, ПОШУК, ВЕБ-САЙТ, ІГРИ, ДОДАТОК.

ABSTRACT

Explanatory note: 93p., 51 figs., 3 add., 25 sources.

Development object: web application in the form of a website with optimization for mobile devices to find teammates in various games.

The purpose of the diploma project: create a convenient space for finding like-minded people and teammates for different games, focusing on the system of mutual choice.

The introduction examines the current state of the problem, specifies the purpose of the qualification work, relevance and the field of its application, specifies the task statement.

In the first section, the analysis of the subject area was carried out, the relevance of the task and the designation of the development was determined, the task statement was developed, the requirements for the software implementation, the technologies and software tools were developed.

The second section provides: the purpose of the program, the description of the applied mathematical methods, the description of the used technologies and programming languages, description of the structure of the program and algorithms of its operation, and detailed description of the work of the developed software product.

In the economic section the complexity of the developed information system is determined, the calculation of the cost of work on the creation of the program is calculated and the time for its creation is calculated.

The practical value is the creation of the web application, which allows you to search for teammates for different games.

The relevance of the information system is determined by the high demand for such developments, which provide opportunities to find teammates.

Keywords: COMPUTER, SEARCH, WEBSITE, GAMES, APP.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

HTML – HyperText Markup Language (мова розмітки на веб-сторінках);

CSS – Cascading Style Sheets (мова стилів на веб-сторінках);

JS – JavaScript (мова програмування);

Dota 2 – Defense of the Ancients 2 (комп’ютерна гра);

CS GO – Counter Strike Global Offensive (комп’ютерна гра);

VS Code – Visual Studio Code (редактор коду);

SPA – Single Page Application (односторінкова веб-сторінка);

SASS (SCSS) – Syntactically Awesome Style Sheets (препроцесор CSS);

LESS – Leaner Style Sheets (препроцесор CSS).

ВСТУП

У сучасному світі дуже важко знайти людину, яка не користується інтернетом. Веб-сайти стали звичайним інструментом для отримання та використання інформації. Навіть деякі комп'ютерні програми переходять у інтернет та створюють свої веб-версії, наприклад Skype, Microsoft Office, Discord, Telegram та багато інших. Це природньо, бо веб-додаток чи сайт не потребує від користувача мати потужний комп'ютер чи задовольняти системним вимогам. Від користувача потребується лише підключення до мережі інтернет та наявність браузеру.

Інша дуже популярна річ у світі, а особливо серед молоді – це комп'ютерні ігри, які у деяких країнах (Китаї, Південної Кореї) вважаються навіть спортивними дисциплінами. Багато людей по всьому світу кожен день грають в такі ігри, як Dota 2, Counter Strike Global Offensive, League of Legends, Valorant, Fortnite. Але що об'єднує ці ігри? Усі вони розраховані на багато користувачів, тобто мають мультіплеєр. В усіх цих іграх гравець змагається не один, а з командою. Але проблемою може стати те, що людина не має з ким створити команду. У цьому випадку людина вимушена грати з випадковими товаришами по команді, що значно зменшує шанси на перемоги через незіграність, язиковий бар'єр, проблеми у спілкування тощо.

Саме такі люди змогли б вирішити проблему пошуку команди, використовуючи додаток або веб-сайт, який дає їм змогу підбирати товаришів по команді, яких людина потребує.

При створенні веб-додатку необхідно враховувати оптимізацію коду проекту, швидкість завантаження сторінки, адаптивність сайту під мобільні пристрої, безпеку даних користувача.

Актуальність веб-сайту полягає в тому, що люди зможуть легко, а головне –

швидко, шукати товаришів по команді для будь-яких цілей у різних іграх. Зручний інтерфейс програми дасть змогу користувачу не загострювати увагу на проблемах використання додатку. Після короткої реєстрації користувач вже матиме змогу швидко і зручно шукати товаришів по команді з будь-якої точки світу.

Метою даної роботи є створення веб-сайту, який дасть змогу гравцям (або навіть кібер-скаутам) шукати товаришів по команді з усього світу.

Для досягнення поставленої мети треба вирішити наступні питання:

- аналіз предметної області;
- уточнення вимог до параметрів веб-сайту;
- уточнення вимог до продуктивності веб-сайту;
- розробка архітектури компонентів веб-сайту.

Відповідно до проведеного аналізу та завдання в роботі поставлені такі функціональні задачі та вимоги до веб-додатку, а саме:

- зручний і інформативний інтерфейс;
- користувач має мати змогу створити аккаунт та використовувати його;
- можливість здійснення пошуку серед інших користувачів сайту, які знаходяться офлайн;
- окремий пошук для користувачів, які знаходяться онлайн;
- можливість опису свого профілю у різних іграх;
- можливість додавати, видаляти ігри зі свого пошуку та профілю;
- наявність адаптивності для використання сайту на мобільних пристроях;
- наявність можливості керування сайтом для людей з обмеженими можливостями.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

У сучасному світі комп’ютерні ігри здобули небувалої популярності. Багато гравців з усього світу щодня б’ють рекорди з кількості одночасно граючих людей у тих чи інших дисциплінах. На сьогодні ігрова індустрія – дуже розвинута та прогресивна як для бізнесу, так і для самих гравців. Так, сьогодні людина може спостерігати за улюбленою кіберспортивною командою через безліч різних стрімінгових сервісів, таких як Twitch, Youtube, Megogo, Huya. Кіберспорт вже давно став рівним конкурентом звичайному спорту і за кількістю спостерігачів, і за кількістю грошей, яка є в обороті індустрії. А у деяких країнах, наприклад Китаї чи Південній Кореї, кіберспорт навіть популярніше звичайного спорту.

Кожного дня середній онлайн у грі Dota 2 – 600000 гравців, у Counter Strike Global Offensive – 1000000 гравців, у грі League of Legends – 800000 гравці. І це вже понад два с половиною мільйони гравця тільки у трьох іграх. Важливо зауважити, що ще понад 200000 людей у той самий час спостерігають за іншими гравцями у ці ігри на стрімінгових сервісах. На рис. 1.1 ви можете побачити статистику про онлайн в іграх тільки на сервісі цифрової дистрибуції Steam. Зауважимо, що окрім Steam, ще є інші популярні сервіси, які піднімуть статистику ще вище (такі як GOG, Epic Games, Battle.net).

Рис. 1.1. Кількість гравців у різних іграх на сервісі Steam

Як можемо бачити, індустрія дуже розвинулася за останні роки. Безліч гравців з усього світу кожну хвилину шукають новий матч. Безліч кіберспортивних команд щодня змагаються на різних турнірах. Мільярди доларів витрачаються на ігри щонеділі. Але у індустрії є одна дуже велика проблема, яка зачіпає і гравців, і кіберспортивні команди, і спільноту ігор у цілому.

Ця проблема – пошук товаришів по команді. Як ви можете бачити зі статистики, усі найпопулярніші ігри сьогодення – це мультіплеєрні ігри, тобто ігри, де метою є гра у команді. Але у кожній грі з топу пошук товаришів по команді реалізований дуже скудно та погано. Ви не можете обрати мову, на якій ви будете спілкуватися. Ви не можете обрати навіть країну, з якої будуть більшість у вашій команді. Ви не можете обрати собі товариша по команді за необхідними навичками. І за цієї причини дуже часто виходить, що гравець потрапляє у команду, наприклад, з чотирьох польських гравців, хоча сам є

українцем. Або гравець потрапляє у команду, де його навички значно більші або менші за навички його товаришів по команді. Або найгірше – його товариші по команді виявилися зовсім не товаришами та мають дуже негативну поведінку, що може впливати як на гравця, так і на спільноту в цілому.

На рис. 1.2 зображено вікно пошуку у грі Dota 2.

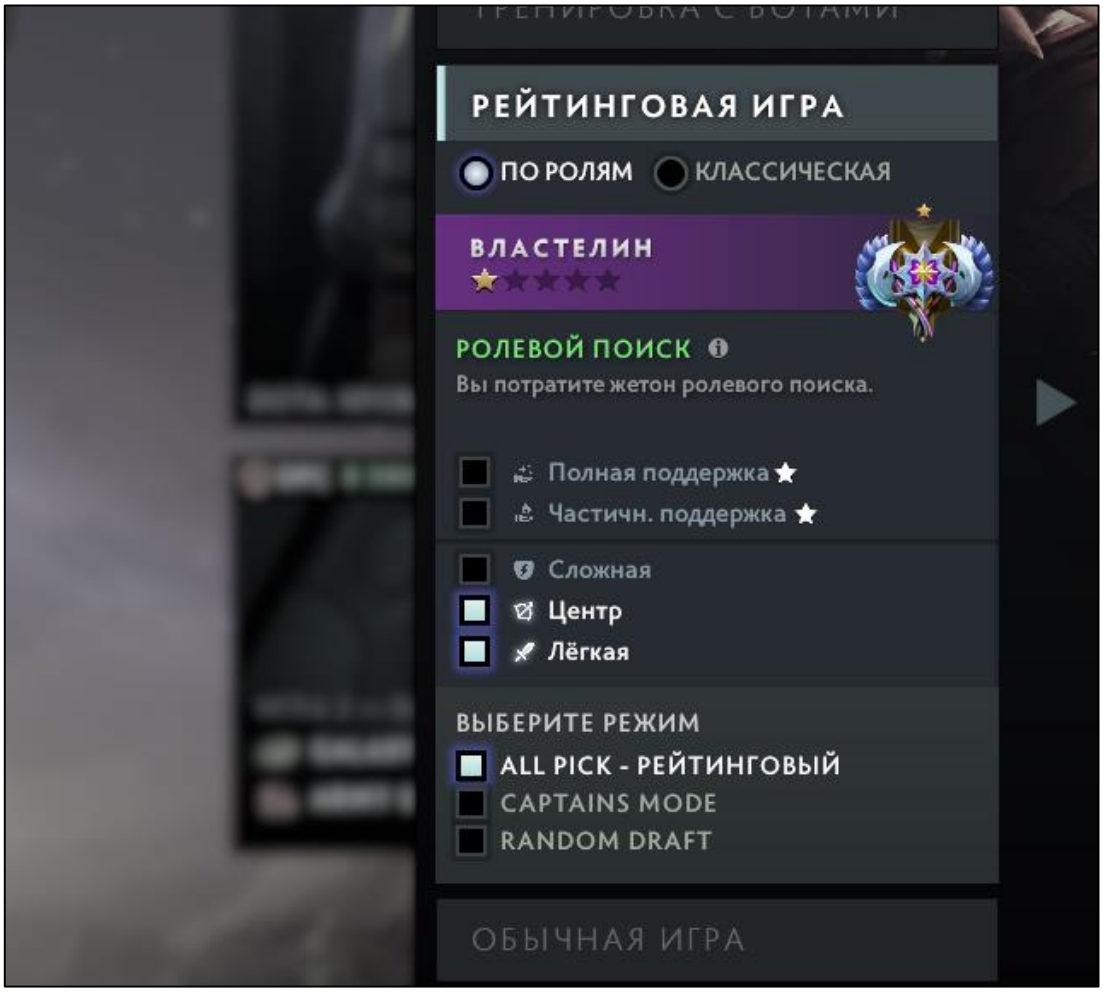

Рис. 1.2. Вікно пошуку матчу у грі Dota 2

Як ми можемо бачити, єдиний критерій, за яким ми можемо фільтрувати пошук у грі Dota 2 – це ігрова роль. Але зауважимо, що навіть використовуючи цю модель пошуку, гра змушує вас раз у декілька ігор грати на ролі, яка вам не

подобається та на якій ви грати не вмієте. Також гра підбирає гравців за рівнем їх навичок, але ця система дуже часто працює не правильно. Так, в один матч можуть потрапити гравці, рівень яких відрізняється як рівень першорозрядника та майстру спорту з шахів. Приклад такого матчу зображено на рис. 1.3.

Рис. 1.3. Приклад нерівного підбору гравців

Як бачимо, в одну команду потрапили гравці зовсім різного рівня навичок, що можна бачити за різними позначками рангу або рейтингу цих гравців.

Але ще гірше, коли в одну команду потрапляють гравці, які не подобаються одне одному. Тоді нормально грати, а тим паче виграти матч не має жодного шансу. Вашому товаришу по команді може не сподобатися мова, на якій ви спілкуєтеся, ігрова роль, яку ви обрали або навіть просто ваш псевдонім у грі.

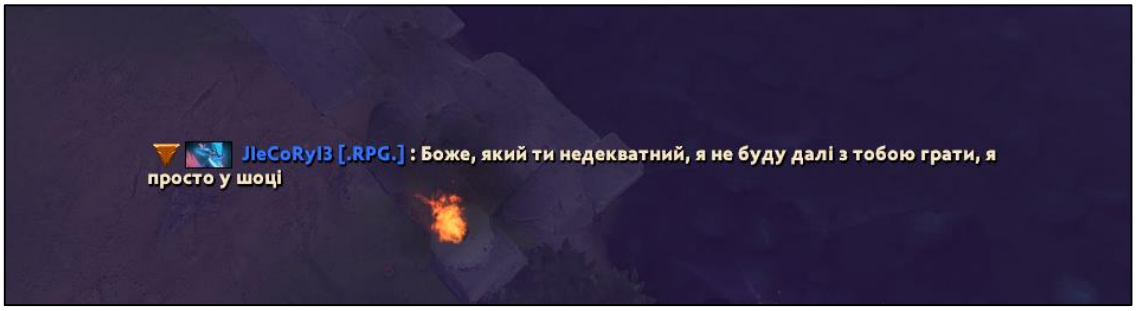

Рис. 1.4. Приклад непорозуміння між командою

І ця проблема торкається не тільки гри Dota 2. Так, у грі Counter Strike Global Offensive ситуація ще гірше. Як можемо бачити на рис. 1.5, у грі можливість пошуку обмежується лише вибором ігрової карти. Ви не можете обрати навіть ігрову роль, яку ви виконуєте у команді (снайпер, гравець з гвинтівками або помічник).

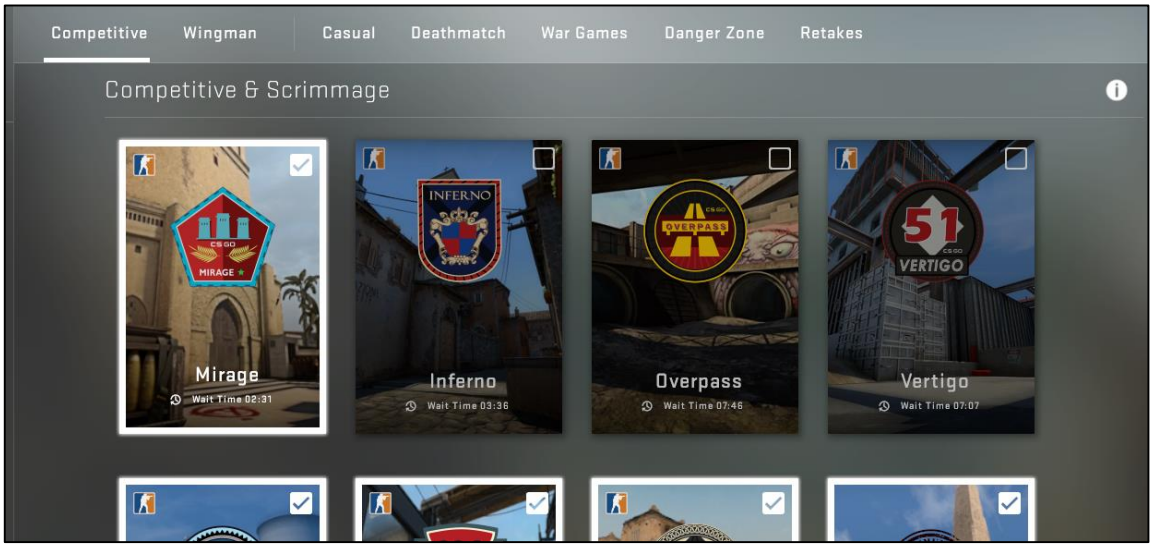

Рис. 1.5. Вікно пошуку у грі Counter Strike Global Offensive

У комп’ютерній грі Paladins людина може почати пошук матчу лише за регіоном. Та це навіть не регіон, звідки будуть гравці з вашої команди. Це просто регіон з ігровими серверами, тобто регіон, де гра буде відчуватися стабільнішою з точки зору інтернету. На рис. 1.6 зображено вікно пошуку у комп’ютерній грі Paladins. До речі, ця гра тісно зав’язана на комунікації у команді та ігрових ролях. Але на відміну від тієї ж Dota 2, у Paladins не можна шукати товаришів по команді навіть за ігровою роллю. Таким чином, кожен матч ви починаєте з сутичок з командою з приладу того, хто саме буде грати на тій чи іншій ролі.

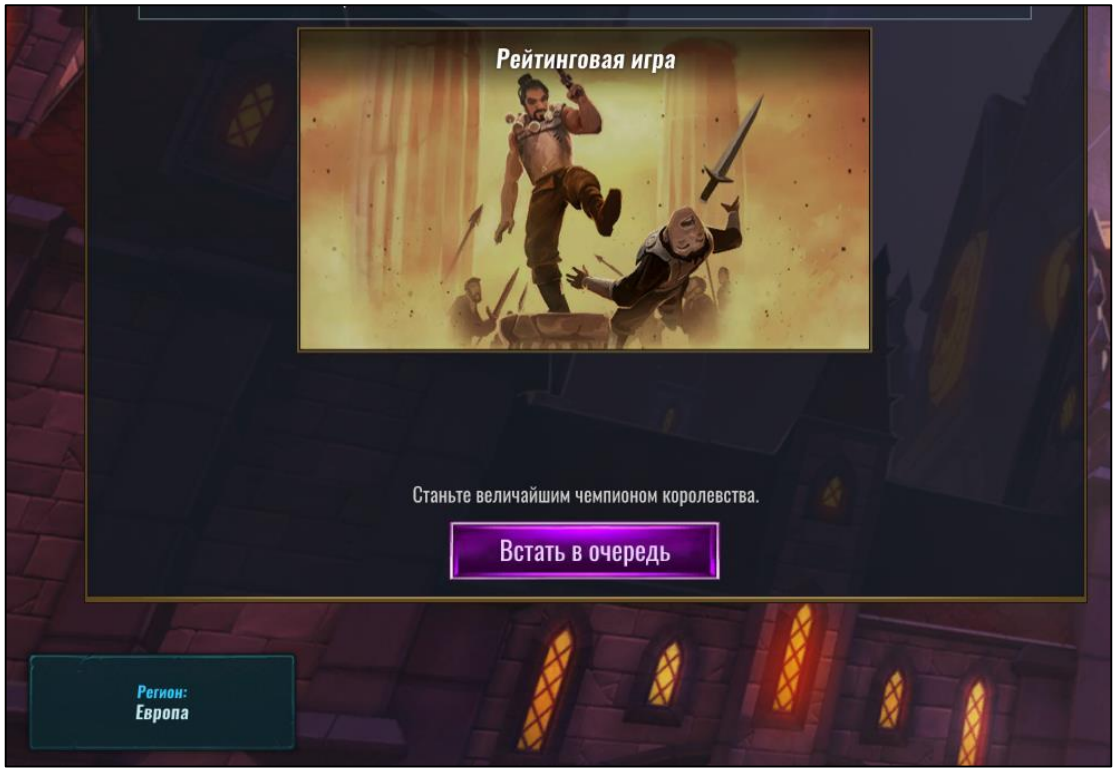

Рис. 1.6. Вікно пошуку у комп’ютерній грі Paladins

Можна подумати, що така проблема присутня лише у безкоштовних або умовно безкоштовних іграх. Але це не так. Така проблема присутня усюди. Наприклад, у комп’ютерній грі Call of Duty: Modern Warfare 2019, яка коштує 690 гривень, у пошуку зовсім нічого, окрім ігрового режиму, відфільтрувати не можна. Таким чином, навіть у платній грі, яка, здавалося б, мала б мати більш розвинутий інтерфейс та більш досвідчену команду розробників, проблема пошуку товаришів по команді є такою ж гострою. Вікно пошуку у грі Call of Duty: Modern Warfare 2019 зображено на рис. 1.7.

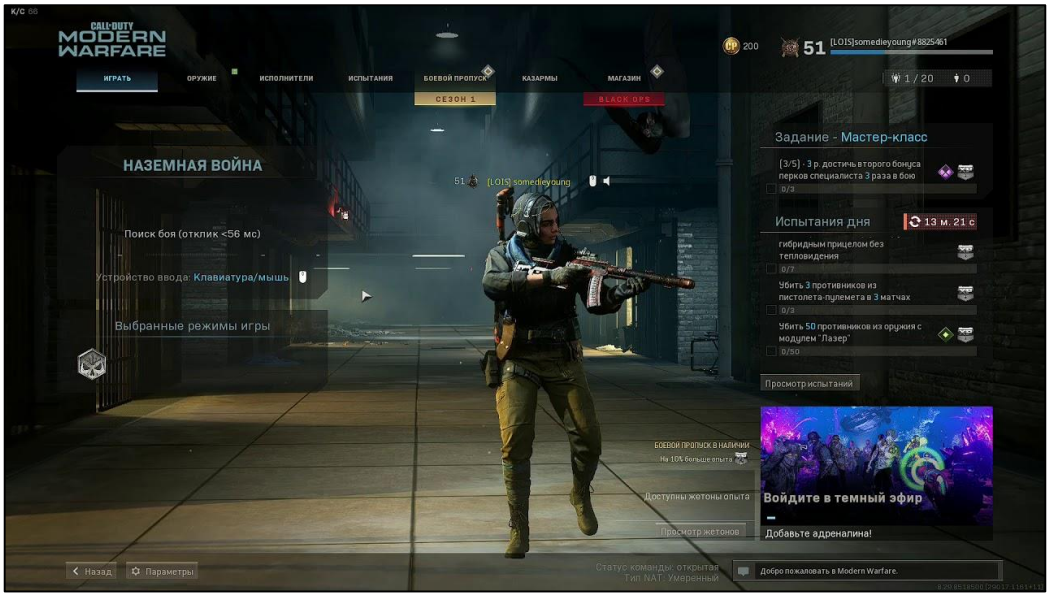

Рис. 1.7. Вікно пошуку у грі Call of Duty: Modern Warfare 2019

Але проблема пошуку товаришів по команді торкається не лише звичайних гравців, а й кіберспортивних команд. Скаутам команд дуже важко знайти нову зірку, новий талант на кіберспортивній сцені, бо можливість пошуку гравців межує лише з можливістю пошуку серед вже існуючих команд. А команди, які збирають свою команду навмання, майже завжди мають негативні результати. Так, дивлячись на топ різних гравців у дисципліні Counter Strike Global Offensive, можемо побачити, що топ 100 гравців із року в рік майже не змінюється, а верхівка топу є незмінною зовсім. Приладом для гордості, звичайно, є те, що топ 1 рейтингу у кожному році займає український гравець Олександр «s1mple» Костилєв. На рис. 1.8 зображено топ гравців дисципліни за останні три роки.

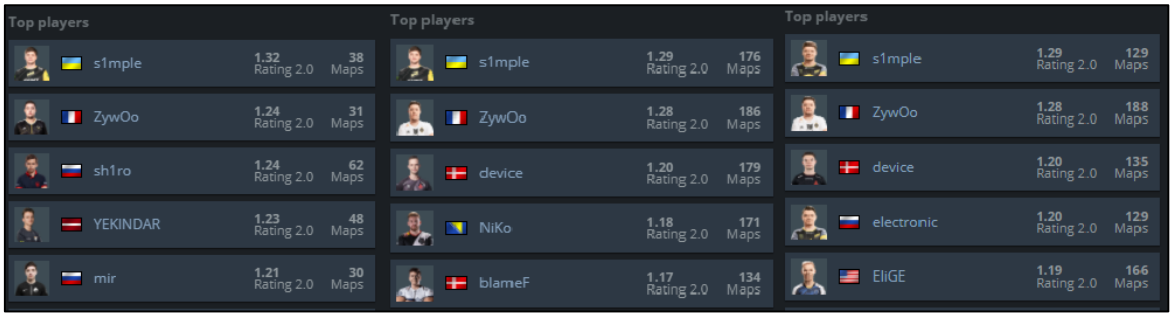

Рис. 1.8. Топ гравців у дисципліні Counter Strike GO останні 3 роки

Однак, ситуація залишається незмінною вже протягом п’яти років. У жодній дисципліни, типу футболу чи баскетболу, жоден гравець не досягав таких результатів. А причина у тому, що вибірка гравців дуже невелика. У футболі є ігрові школи, про різні види спорту розповідають з дитинства у навчальних закладах, скаути можуть знайти майбутню зірку навіть на подвір’ї на стадіоні. Але як шукати гравців скаутам у кіберспорті? Для цього необхідно створити простір або додаток для пошуку товаришів по команді. Це допоможе і гравцям, і скаутам, і спільноті у цілому.

Чи є вже якісь приклади ресурсів для пошуку товаришів по команді? Так. Але всі ресурси, які існують на даний момент, користуються дуже малим попитом, бо вони створенні у форматі форумів або чатів. Гравець залишає свою анкету на такому сайті, а потім вимушений чекати, поки хто-небудь на неї відповість. Часто гравці зовсім забувають про такий ресурс, бо команда була їм потрібна тут і зараз.

Прикладом такого ресурсу є сайт турнірного організатора Starladder. Гравці залишають малу інформацію про себе, свої контакти і все. Далі слід лише чекати. Довго чекати поки хто-небудь напише тобі. Або можна почати самому шукати товариша по команді, але зручного способу зробити це немає – лише шукати гравців серед безлічі сторінок сайту. Зовнішній вид сайту зображено на рис. 1.9.

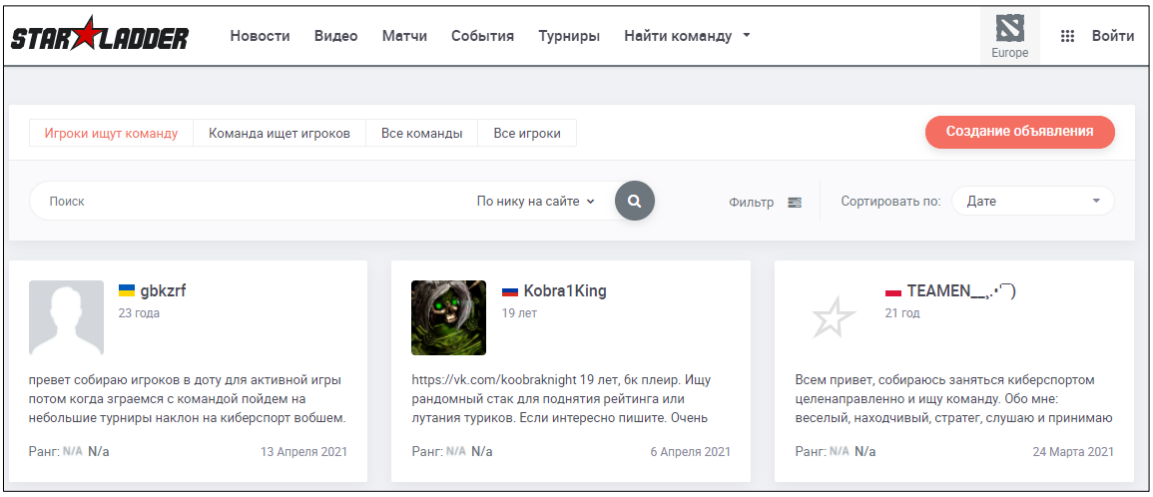

Рис. 1.9. Зовнішній вигляд сайту Starladder

Усі інші платформи для пошуку користуються ще меншим попитом. Та проблема полягає у тому, що під кожну гру потрібно шукати окремий сервіс для пошуку, а для деяких ігор таких сервісів зовсім не існує.

Розробка зручного централізованого додатку, який дав би змогу шукати товаришів по команді у великій кількості ігор, вирішує усі описані вище проблеми. Але треба зрозуміти, який додаток для пошуку людей буде дійсно зручним для використання.

Зробити аналіз додатків для пошуку або знайомств людей достатньо просто. Необхідно лише зайти у магазин додатків Play Market та подивитися, які додатки є найпопулярнішими. Найбільш популярні на сьогоднішній день додатки – це Badoo та Tinder. Вони мають велику аудиторію та гнучкий і вдалий інтерфейс. Інші додатки, які можна знайти на ринку, копіюють ідею основних та найпопулярніших. Основна ідея цих додатків полягає у тому, що кожна людина у пошуку відображається у вигляді карточки. На карточці зображена інформація про людину, її фото, вік, інтереси тощо. Якщо людина сподобалася користувачу, він перегортає карточку управо. Якщо не сподобалася – уліво. Таким чином людина може швидко та зручно шукати саме тих, хто їй потрібен. Але

найголовніше трапляється, коли людина, яку користувач перегорнув управо також перегорнула користувача управо. Таким чином будується пара та люди можуть писати одне одному та почати спілкуватися. Приклад інтерфейсу додатку Badoo зображено на рис. 1.10.

Рис. 1.10. Інтерфейс мобільного додатку Badoo

А тепер уявимо, що подібний додаток буде орієнтований лише на ігри. Ви заходите у додаток, починаєте пошук за окремою грою, продивляєтесь безліч гравців с усього світу за різними критеріями, які вам потрібні і швидко знаходите необхідних членів команди для матчу у, наприклад, Dota 2 або для турніру з CS GO. Це вирішить багато проблем пошуку геймерів з усього світу. Люди зможуть збирати команди швидко та зручно. Можливість пошуку за різними іграми в одному місці – дуже зручна альтернатива усім форумам за окремою грою.

1.2. Призначення розробки та галузь застосування

Призначення розробки – створення веб-додатку, адаптивного для мобільних пристроїв, для реалізації можливості пошуку товаришів по команді у різноманітних ігрових дисциплінах.

Додаток може бути використаний звичайними гравцями у різні комп’ютерні ігри та скаутами кіберспортивних команд для пошуку гравців за критеріями, які їм необхідні. Також додаток може сприяти створенню спільноти навколо окремої гри шляхом формування рейтингу гарних гравців, що сприятиме поліпшенню психологічного клімату у тій чи іншій грі серед спільноти.

1.3. Підстави для розробки

Підставою для розробки кваліфікаційної роботи на тему «Розробка веб-додатку для пошуку товаришів по команді в іграх на базі фреймворка React» є наказ ректору по Національному технічному університету «Дніпровська політехніка» №317-с від 07.06.2021.

1.4. Постановка завдання

Метою даної роботи є створення веб-додатку, який дозволяє шукати товаришів по команді у різноманітних іграх шляхом створення інтерфейсу, схожого на зручний інтерфейс додатків для знайомств.

Продукт повинен працювати в усіх актуальних браузерах на усіх популярних платформах (Windows, MacOS, Linux) та на усіх актуальних пристроях, тобто має бути адаптивним під мобільні пристрої. Продукт повинен

давати можливість користувачу зареєструватися та має належним чином захищати інформацію, надану користувачем.

1.5. Вимоги до програми або програмного виробу

Головною вимогою роботи є можливість використання додатку на мобільних пристроях та швидкість роботи продукту. Алгоритми рендеренгу сторінки мають задіяти якомога менше ресурсів комп’ютеру або мобільного пристрою, на якому використовується додаток.

1.5.1. Вимоги до функціональних характеристик

До веб-додатку, який розроблюється в ході кваліфікаційної роботи, висовуються наступні функціональні вимоги:

- можливість реєстрації користувача;
- можливість логіну користувача;
- можливість редагування свого профілю;
- пошук товаришів по команді за різними іграми серед користувачів, які знаходяться оффлайн;
- пошук товаришів по команді за різними іграми серед користувачів, які знаходяться онлайн;
- можливість використання сайту на мобільних пристроях та його адаптивність;
- можливість використовувати сайт для людей з обмеженими можливостями;
- зручний та зрозумілий інтерфейс.

1.5.2. Вимоги до інформаційної безпеки

Веб-додаток має передавати інформацію користувача використовуючи спеціальні техніки, щоб зловмисники не могли отримати доступ до особистих даних користувача. Для зберігання інформації, яка може бути підв’язана до браузера користувача мають використовуватись cookie браузера.

Профіль користувача має бути захищений паролем та логіном, який буде знати лише користувач. Пароль та логін користувача мають зберігатися лише на серверній частині додатку.

1.5.3. Вимоги до складу та параметрів технічних засобів

Веб-додаток не вимогливий до потужності пристрою, на якому його запущено. Головна вимога для роботи веб-додатку – наявність інтернету та браузеру для відтворення сторінки у ньому. Швидкість повільного 3G інтернету має бути достатньою для роботи та завантаження веб-додатку на мобільних та комп’ютерних пристроях, а також планшетах.

1.5.4. Вимоги до інформаційної та програмної сумісності

Для запуску коду веб-додатка необхідні наступні компоненти:

- будь-яка операційна система (Windows, Linux, MacOS);
- встановлене середовище Node.js;
- інструмент керування пакетами NPM;
- інструмент запуску файлів проекту serve.

Або, при використанні вже завантаженої у хмарові сервіси версії сайту необхідно лише перейти за посиланням веб-додатку.

Для розробки веб-додатка необхідні наступні компоненти:

- редактор коду (VS Code, WebStorm або будь-який інший);
- середа запуску серверу Node.js;
- середа запуску пакетів та скриптів NPM;
- бібліотека React;
- бібліотека React Router;
- бібліотека styled-components;
- бібліотека Redux;
- складальник файлів проекту Webpack (або будь-який інший, наприклад, Rollup).

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Призначення розробки системи – створення веб-додатку для пошуку товаришів по команді в іграх.

Розроблений в ході виконання кваліфікаційної роботи веб-додаток має наступні функції та призначення:

- зручний та інтуїтивно зрозумілий інтерфейс;
- адаптивність під мобільні пристрої для зручного використання додатку на усіх пристроях;
- можливість пошуку товаришів по команді серед користувачів, які не знаходяться зараз на сайті;
- можливість пошуку товаришів по команді серед користувачів, які зараз онлайн;
- можливість реєстрації користувача та створення ним облікового запису (профілю);
- можливість налаштування профілю після його створення;

Веб-додаток може бути використаний будь-який користувачем або скаутом кіберспортивної команди для пошуку товаришів по команді в іграх або для пошуку нового гравця у кіберспортивну команду.

2.2. Опис застосованих математичних методів

Для коректної роботи веб-додатку не було застосовано жодних математичних методів.

2.3. Опис використаних технологій та мов програмування

Сучасні веб-сайти створюються за допомогою цілої низки технологій. Вибір цих технологій обґрунтовується вимогами сайту. Але три основні технології є незмінними: HTML, CSS та JavaScript. Для створення веб-сайту необхідно створити розмітку сайту за допомогою мови розмітки HTML (Hypertext Markup Language), далі потрібно оформити стилі сайту за допомогою CSS (Cascading Style Sheets) та у кінці потрібно додати сайту функціонал за допомогою мови JavaScript.

Але сучасні веб-додатки вже давно зробили крок уперед. Для більшої адаптивності, структурованості розробки, більшої кількості можливостей та інших переваг у розробці веб-сайтів використовують різні бібліотеки та фреймворки. Так, для CSS є ціла низка популярних бібліотек, що прискорюють та полегшують розробку стилів для сайтів, такі як Bootstrap, Semantic UI та різні препроцесори, наприклад SCSS або LESS. Для керуванням сайтом та додаванням на нього різного функціоналу використовують три основні дуже популярні фреймворки: React, Angular, Vue. Хоча часто React називають бібліотекою, що вказано навіть на офіційному сайті React, цей інструмент дає дуже багато можливостей для керування компонентами сайту, керуванням їх поведінки, створенням SPA (Single page application).

Кожна бібліотека та фреймворк мають свої переваги та недоліки. Для цієї роботи було обрано наступний стек технологій, який відповідає вимогам для створення сучасного та зручного веб-додатку.

Для створення структури сайту було обрано бібліотеку React. По-перше, ця бібліотека дає змогу створювати SPA додаток. Чому це важливо? Сайт, який є однією сторінкою, дозволяє використовувати себе без перезавантаження при

переході на різні вкладки сторінки. Це економить і трафік користувача, і його час та підвищує швидкість відгуку сайту та його продуктивність. Схему роботи SPA та різницю між традиційною сторінкою можемо побачити на рис. 2.1.

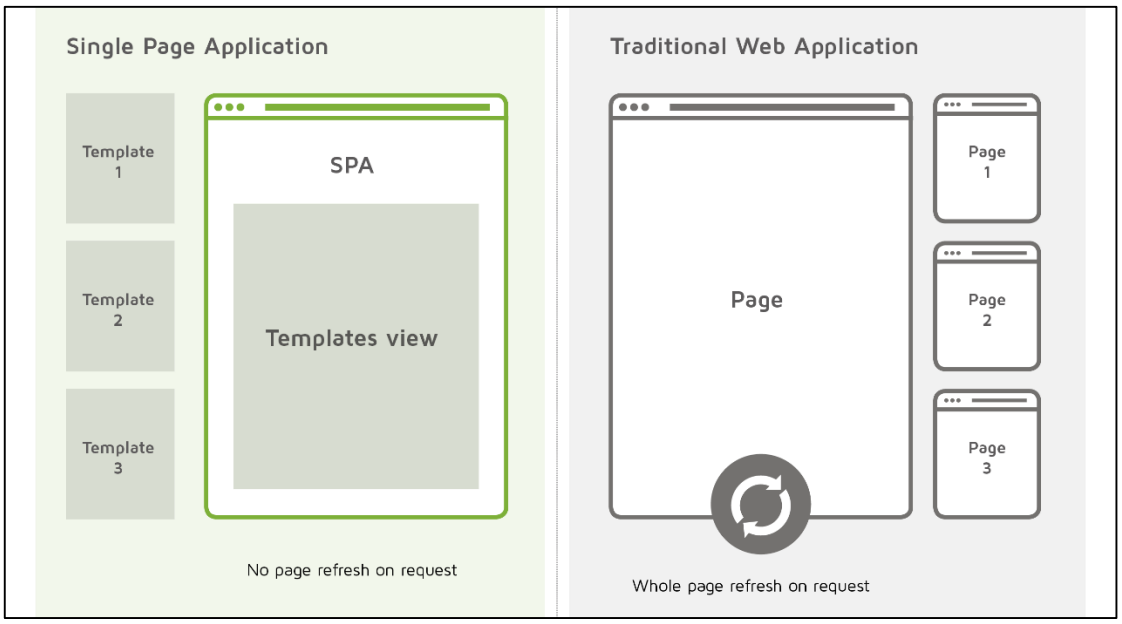

Рис 2.1. Схема роботи SPA

По-друге, React дозволяє створювати сайти з динамічними компонентами, що означає, що якщо один з компонентів сайту оновиться, то для його оновлення React застосує свої алгоритми та не буде перезавантажувати усю сторінку, а оновить лише конкретний компонент. Але ця перевага також змушує нас правильно підійти до розбиття сайту на різні компоненти. Правильна структура та архітектура сайту прискорює та спрощує роботу, але неправильна структура може тільки зіпсувати процес розробки.

По-третє, на відміну від Angular, React дуже гнучка бібліотека з великою кількістю різних модулів, що дозволяє реалізувати за допомогою React будь-яке рішення. Ми можемо зручно налаштування перехід між вкладками сайту, налаштувати стилі сайту, розробити сховище для даних сайту, створити зручній

API-клієнт для роботи з сервером.

Та по-четверте, React є самою популярною бібліотекою JavaScript, що спрощує роботу з ним через велику кількість інформації про нього на спеціалізованих ресурсах та у книгах.

Для запуску веб-додатку необхідно використовувати інструмент для збору додатку, тобто такі інструменти, як Webpack, Rollup. У цій роботі використовується Webpack, бо за допомогою зручної утіліти для налаштування базової комплектації React CRA (create-react-app) цей інструмент збору встановлюється за замовченням. Також необхідно використовувати середу запуску додатку. Це Node.js, через використання системою контролю пакетів NPM (Node Package Manager). Система контролю пакетів – це інструмент, який дозволяє керувати пакетами бібліотек. За допомогою NPM ми можемо зручно встановлювати, видаляти пакети бібліотек, завантажувати додатки, тестувати сайт та навіть налаштовувати роботу з git та іншими пакетними утілітами.

Для контролю коду та версій сайту використовується git, а для контролю репозиторію роботи – GitLab. Git дозволяє розбивати функціонал сайту на різні гілки та структурувати код роботи. Наприклад, як на рис. 2.2, де master та testing є різними гілками роботи з різним функціоналом, який потім об’єднується в одну гілку.

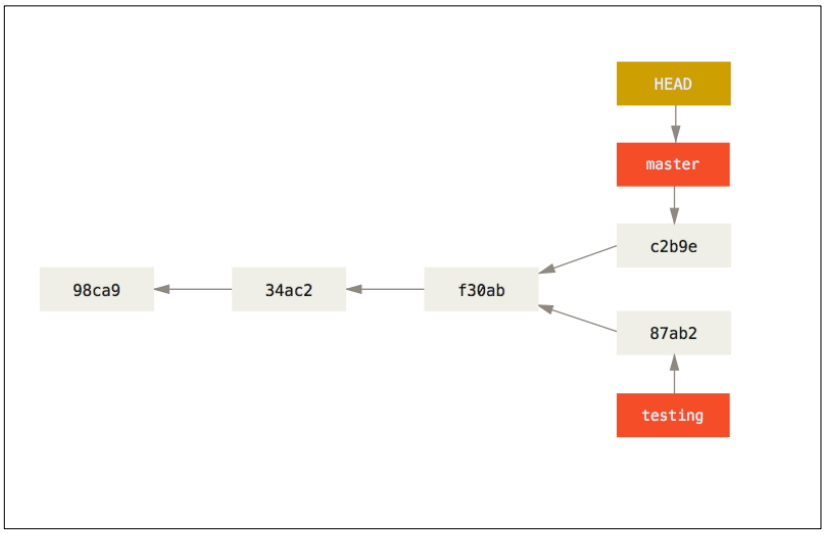

Рис. 2.2. Схема гілок git

Для переходу між сторінками сайту та навігації по ньому використовується бібліотека React – React Router, як найбільш зручний та розповсюджений інструмент.

Для контролю сховищем сайту використовується система контролю сховищем Redux, основною ідеєю якої є те, що ми створюємо окреме сховище додатку, до якого звертаємося за допомогою подій, що генерують наші компоненти. Сховище приймає подію, розуміє за типом події, що саме потрібно роботи зі сховищем, після чого ред'юсер сховища оновлює його та повертає у додаток. Схему роботи Redux можна побачити на рис. 2.3.

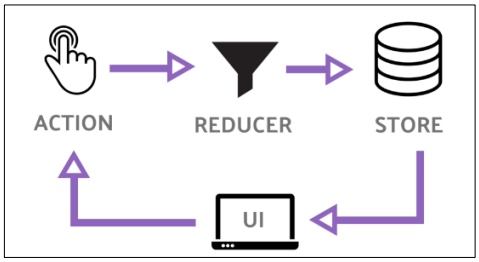

Рис. 2.3. Схема роботи Redux

Для оформлення зовнішнього виду сайту використовується CSS-модулі, що відрізняються від стандартних файлів CSS тим, що вони дають змогу інкапсулювати стилі всередині їх компонентів. Тобто стилі окремого компоненту будуть мати спеціальний згенерований модулем стиль, який буде застосовуватися лише для конкретного компоненту, не змішуючи стилі різних компонентів, навіть якщо вони мають однакове ім’я. Якщо потрібно використати якийсь стиль у декількох компонентах, можемо створити спільний звичайний файл CSS, та використовувати стилі з нього паралельно з використанням CSS-модулів.

Для того, щоб забезпечити безпечність даних користувача використовуються технології шифрування при передачі даних. Таким чином, при реєстрації користувача у кукі-файли браузера користувача встановлюється спеціальний токен, що забезпечує підключення до серверу з даними. Без даного токену зловмисник не має можливості підключення до серверу, а отримати сам токен можна тільки після авторизації, тобто за допомогою логіну та паролю. При кожному запиті з клієнтської сторони додатку надсилається і сам токен, сервер перевіряє чи правильний токен має користувач і тільки після цього надсилає необхідні дані. Схему роботи цього захисного механізму зображено на рис. 2.4.

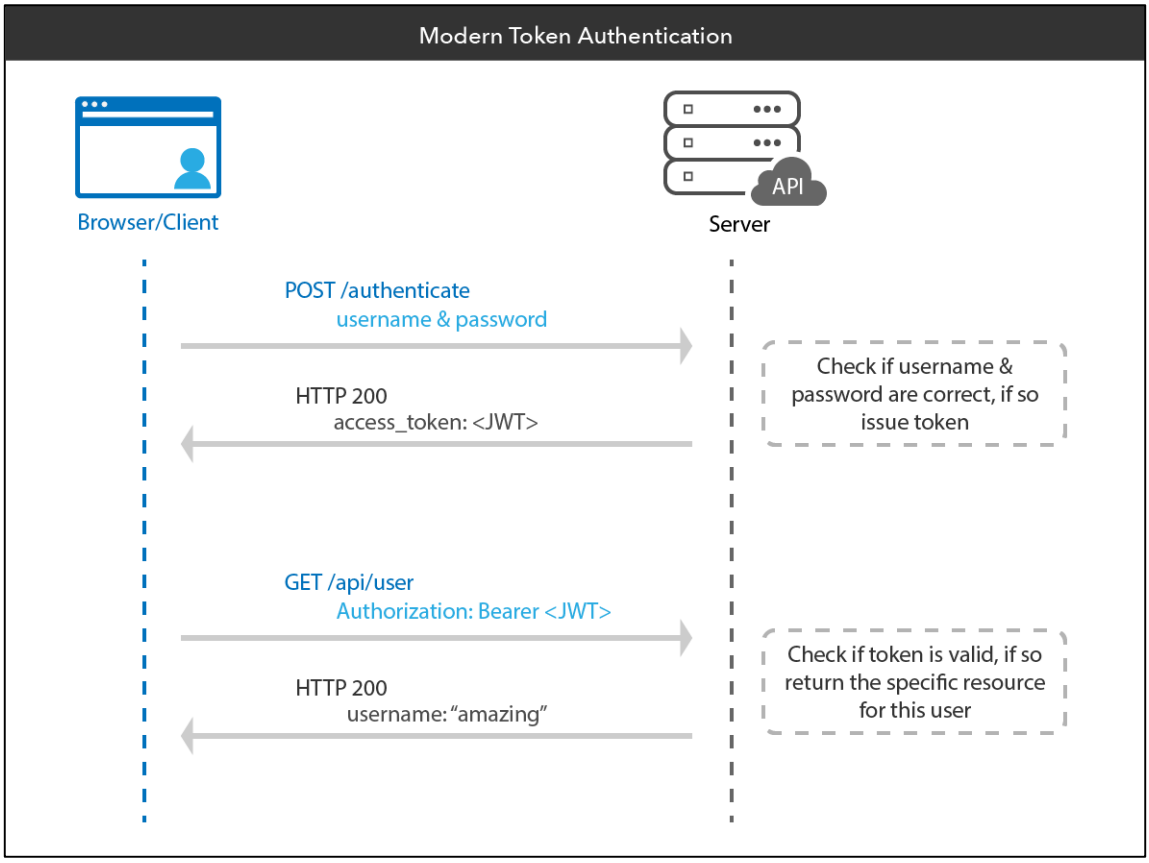

Рис. 2.4. Схема передачі даних при використанні токену

Для роботи з сервером використовується Fetch API, як найзручніший та найбільш відповідний до вимог інструмент. Fetch API надає інтерфейс для роботи з мережевими запитами на сервер та відмінно підходить для роботи з сервером через архітектуру REST (Representational State Transfer). У додатку створено API-клієнт, який є обгорткою над Fetch API. API-клієнт має надавати зручний інтерфейс для роботи з функціями запиту на сервер для нашого додатку.

Архітектура REST для моделювання роботи з сервером була обрана як найпопулярніша та найбільш зручна. Суть цієї архітектури полягає в тому, що кожен запит ми створюємо на конкретні так звані ендпоінти, а для різного функціоналу та роботи з даними застосовуємо різні методи доступу до цих ендпоінтів. Наприклад, для доступу до даних тільки для читання ми створюємо

запит на сервер через метод GET (отримати). Для запису даних – метод POST (встановити). Для оновлення даних – метод PATCH (оновити). Схема роботи ендпоінтів REST архітектури наведена на рис. 2.5.

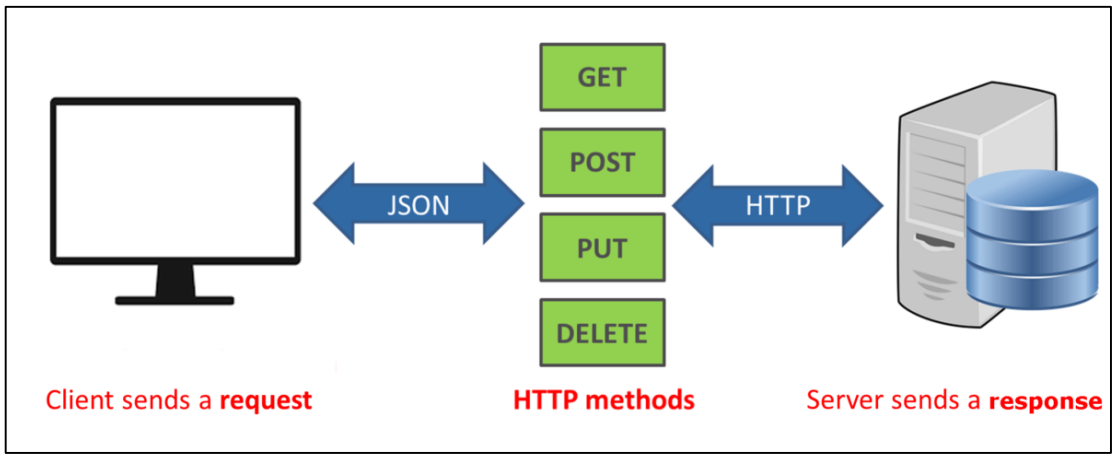

Рис. 2.5. Схема роботи REST архітектури

2.4. Опис структури системи та алгоритмів її функціонування

Структура проекту роботи складається з папок, що мають різне функціональне значення, та файлів, що відповідають за конфігурацію проекту роботи. Структура проекту наведена на рис. 2.6, рис. 2.7 та рис 2.8.

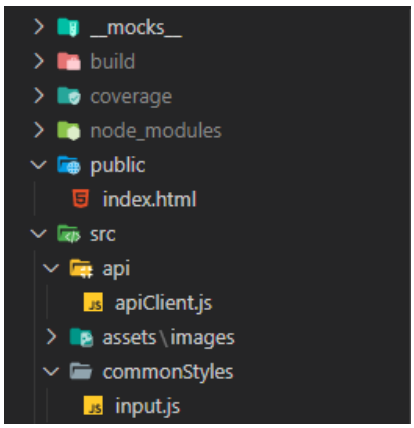

Рис. 2.6. Структура файлів проекту

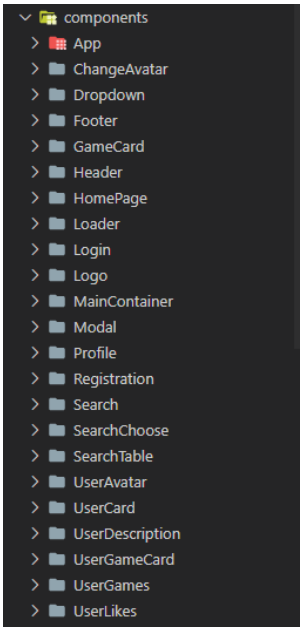

Рис. 2.7. Структура файлів проекту (папка компонентів)

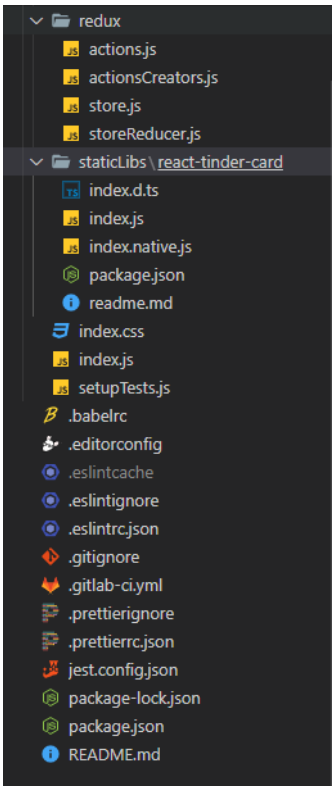

Рис. 2.8. Структура файлів проекту

Опис файлів структури:

1. Папки _mocks_, build, node_modules, coverage мають службове призначення та не несуть сенсу для бізнес логіки проекту.
2. Папка public, що має єдиний файл index.html – головний файл додатку, у який React монтує усю логіку проекту.
3. Папка src – основна папка проекту. У ній зберігаються файли компонентів, службових файлів, додаткових бібліотек.
4. Папка api – має у собі файл apiClient.js, що є АПІ-клієнтом веб-додатку. Він опрацьовує роботу з сервером та вказує, які дані додаток відправляє та отримує при роботі з сервером.
5. Папка assets, у якій є папка images – це папки, що зберігають статичні файли картинок.
6. Папка commonStyles, у якій файл index.js – файл, з якого експортуються спільні для усього проекту стильові компоненти, які застосовуються у декількох місцях на сайті.
7. Папка components – папка, що зберігає усі робочі компоненти додатку
8. Папка App – папка, що зберігає файли головного компоненту додатку, що збирає усі компоненти у цілу систему.
9. Папка ChangeAvatar – папка, що зберігає компонент, який відтворює кнопку для зміни картинки користувача.
10. Папка Dropdown – папка, що зберігає компонент стилізованого випадаючого списку.
11. Папка Footer – папка, що зберігає компонент нижньої частини веб-додатку.
12. Папка GameCard – папка, що зберігає компонент картки гри.
13. Папка Header – папка, що зберігає компонент-шапку сайту.
14. Папка HomePage – папка, що зберігає компонент домашньої сторінки

веб-додатку.

15. Папка Loader – папка, що зберігає компонент завантаження.
16. Папка Login – папка, що зберігає сторінку логіну користувача у веб-додаток.
17. Папка Logo – папка, що зберігає компонент значка веб-додатка.
18. Папка MainContainer – папка, що зберігає службовий компонент, який використовується для стилізації усіх сторінок сайту.
19. Папка Modal – папка, що зберігає компонент модального вікна, що використовується на сайті.
20. Папка Profile – папка, що зберігає сторінку користувача.
21. Папка Registration – папка, що зберігає сторінку реєстрації користувача.
22. Папка Search – папка, що зберігає компонент для пошуку товаришів по команді.
23. Папка SearchChoose – папка, що зберігає сторінку вибору типу пошуку.
24. Папка SearchTable – папка, що зберігає сторінку вибору гри, за якою користувач буде вести пошук.
25. Папка UserAvatar – папка, що зберігає компонент картинки користувача.
26. Папка UserCard – папка, що зберігає компонент картки користувача.
27. Папка UserDescription – папка, що зберігає компонент опису інформації користувача.
28. Папка UserGameCard – папка, що зберігає компонент картки гри у користувача для зміни та налаштування параметрів користувача у даній грі.
29. Папка UserGames – папка, що зберігає компонент, що відображає усі

ігри користувача та кнопку додавання нової гри.

30. Папка UserLikes – папка, що зберігає сторінку вподобань користувача.
31. Папка redux – папка, що зберігає усі налаштування бібліотеки Redux.
32. Папка staticLibs – папка, що зберігає у собі необхідні для роботи додатку статичні бібліотеки.
33. Файл index.css – загальний файл стилів для усього додатку.
34. Файл index.js – загальний функціональний файл, у якому відбувається монтування усього додатку.
35. Файл .babelrc – файл налаштування Babel.
36. Файли .editorconfig, .eslintignore, .eslintrc.json – файли конфігурації середовища розробки.
37. Файл .gitignore – файл налаштування системи контролю версій Git
38. Файл .gitlab-ci.yml – файл налаштування розгортання проекту на сайті GitLab.
39. Файли .prettierignore, .prettierrc.json – файли конфігурації бібліотеки Prettier, що використовується для форматування коду.
40. Файли package-lock.json, package.json – файли конфігурації усього проекту, які зберігають зв’язки з необхідними бібліотеками, що пакет NPM встановлює для коректної роботи веб-додатку.

Загальний алгоритм роботи веб-додатку можна поділити на декілька етапів:

а) збір необхідних компонентів та бібліотек за допомогою NPM;
б) скрипт бібліотеки React починає монтування веб-додатку у файл єдиної фізичної сторінки веб-додатку;
в) бібліотека styled-components, яка використовує підхід CSS-in-JS, монтує стилі у звичайні CSS правила;

г) React спостерігає за змінами під час роботи веб-додатку та монтує нові сторінки, коли необхідно;

д) Redux під час роботи додатку зберігає дані, необхідні для роботи веб-додатку;

е) усі дані, що надходять з серверу, обробляє АПІ-клієнт, після чого ці дані обробляє або Redux, або безпосередньо React.

Спочатку за допомогою ручного або автоматичного при використанні розгортання веб-додатку способу викликається команда npm start або npm run build для запуску проекту або створення зібраного каталогу проекту відповідно.

Після чого завдяки WebPack усі зібрані файли починають взаємодію між собою, під час його React починає процедуру створення віртуального DOM дерева для монтування компонентів на сторінку.

Усі супутні бібліотеки також починають монтуватися у проект. Так, CSS modules перетворюються у звичайні CSS правила, але генерують ключі до відповідних компонентів. Styled components перетворює стильові компоненти у звичайний набір CSS правил, що одразу ж застосовуються для відповідних компонентів.

Redux завдяки даним, які додаток отримує від серверу застосовуючи АПІ-клієнт, зберігає дані у своїй базі та обробляє усі запити на зміну та отримання даних від React.

На цьому етапі завантаження сторінки завершується та відбувається фаза пасивного «слухання» сайту. React реагує на зміні, які користувач відтворює на веб-додатку та відображає відповідні дані або сторінки.

Під час роботи додатку кожні чотири хвилини з клієнту користувача надсилається запит на оновлення його статусу у базі даних. Таким чином можна ідентифікувати користувача, який знаходиться онлайн.

Розглянемо деякі ключові елементи функціювання веб-додатку.

Файл index.js у папці src відтворює монтування бібліотекою React усіх компонентів у компонент div, що знаходиться у головному та єдиному html файлі проекту.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './components/App';
import { BrowserRouter } from 'react-router-dom';
import { Provider } from 'react-redux';
import store from './redux/store';

ReactDOM.render(
  <Provider store={store}>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </Provider>,
  document.getElementById('root')
);
```

Файл index.html у папці public має у собі елемент, у який React монтує усі компоненти. Також цей файл містить усі необхідні підключення та налаштування для сторінки, яка буде відображатися у браузері.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta content="width=device-width, initial-scale=1" name="viewport" />
    <script
      src="https://kit.fontawesome.com/2da37c001d.js"
      crossorigin="anonymous"
    ></script>
    <title>AKY App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

У файлі apiClient.js у папці api відбувається обробка усіх запитів користувача. АПІ-клієнт розроблений таким чином, що він має дві головні функції – для створення запиту та для створення параметрів, необхідних для запиту. Далі усі інші функції АПІ-клієнту використовують основні для створення

та відправки запиту, щоб потім передати дані, які сервер відправить користувачеві, у відповідні компоненти, що будуть відображати ці дані.

```
import Cookies from 'js-cookie';

const API = {
  apiBase: 'https://uaky.herokuapp.com/api/v1/',
  isRefreshing: false,

  async createRequest(endpoint, method, headers, data, isFormData = false) {
    const options = this.createOptions(method, headers, data, isFormData);

    const request = fetch(this.apiBase + endpoint, options);

    return request.then((error) => {
      if (error?.status === 403) {
        if (!this.isRefreshing) {
          this.isRefreshing = true;
          return this.refreshToken().then((res) => {
            if (res?.status === 200) {
              return res.json().then((tokens) => {
                Cookies.set('accessToken', tokens.accessToken);
                Cookies.set(
                  'refreshToken',
                  tokens.refreshToken
                );
                this.isRefreshing = false;
                const newHeaders = headers;
                newHeaders.Authorization = `Bearer ${tokens.accessToken}`;
                const newOptions = this.createOptions(
                  method,
                  newHeaders,
                  data,
                  isFormData
                );
                return fetch(
                  this.apiBase + endpoint,
                  newOptions
                );
              });
            }
          });
        } else {
          return new Promise((resolve) => {
            const intervalId = setInterval(() => {
              if (!this.isRefreshing) {
                clearInterval(intervalId);
                const newHeaders = headers;
                newHeaders.Authorization = `Bearer ${Cookies.get(
                  'accessToken'
                )}`;
```

```
                        const newOptions = this.createOptions(
                            method,
                            newHeaders,
                            data,
                            isFormData
                        );
                        resolve(
                            fetch(this.apiBase + endpoint, newOptions)
                        );
                    }
                }, 100);
            });
        }
    }
    return request;
  });
},

async refreshToken() {
  return await this.createRequest('auth/refresh', 'POST', {
    Authorization: `Bearer ${Cookies.get('refreshToken')}`,
  });
},

createOptions(method, headers, data, isFormData) {
  const options = {};
  if (headers) options.headers = headers;
  if (data && !isFormData) options.body = JSON.stringify(data);
  if (data && isFormData) options.body = data;
  options.method = method;
  return options;
},

async getGames() {
  return await this.createRequest('game', 'GET');
},

async commonSearchUsers(id) {
  return await this.createRequest(`common-search/${id}`, 'GET', {
    Authorization: `Bearer ${Cookies.get('accessToken')}`,
  });
},

async onlineSearchUsers(id) {
  return await this.createRequest(`online-search/${id}`, 'GET', {
    Authorization: `Bearer ${Cookies.get('accessToken')}`,
  });
},

async refreshUsers(id) {
  return await this.createRequest(`common-search/${id}/views`, 'DELETE', {
    Authorization: `Bearer ${Cookies.get('accessToken')}`,
```

```
    });
},

async onlineRefreshUsers(id) {
    return await this.createRequest(`online-search/${id}/views`, 'DELETE', {
        Authorization: `Bearer ${Cookies.get('accessToken')}`,
    });
},

async registration(data) {
    return await this.createRequest(
        'register',
        'POST',
        { Accept: '*/*', 'Content-Type': 'application/json' },
        data
    );
},

async login(data) {
    return await this.createRequest(
        'auth/login',
        'POST',
        { Accept: '*/*', 'Content-Type': 'application/json' },
        data
    );
},

async getUserInfo() {
    return await this.createRequest('me', 'GET', {
        Authorization: `Bearer ${Cookies.get('accessToken')}`,
    });
},

async updateUserGame(game_id, data) {
    return await this.createRequest(
        `me/games/${game_id}`,
        'PUT',
        {
            Authorization: `Bearer ${Cookies.get('accessToken')}`,
            Accept: '*/*',
            'Content-Type': 'application/json',
        },
        data
    );
},

async deleteUserGame(game_id) {
    return await this.createRequest(`me/games/${game_id}`, 'DELETE', {
        Authorization: `Bearer ${Cookies.get('accessToken')}`,
    });
},
```

```
async addUserGame(data) {
   return await this.createRequest(
      'me/games',
      'POST',
      {
         Authorization: `Bearer ${Cookies.get('accessToken')}`,
         Accept: '*/*',
         'Content-Type': 'application/json',
      },
      data
   );
},

async editNickname(data) {
   return await this.createRequest(
      'me/nickname',
      'PATCH',
      {
         Authorization: `Bearer ${Cookies.get('accessToken')}`,
         Accept: '*/*',
         'Content-Type': 'application/json',
      },
      data
   );
},

async editPassword(data) {
   return await this.createRequest(
      'me/password',
      'PATCH',
      {
         Authorization: `Bearer ${Cookies.get('accessToken')}`,
         Accept: '*/*',
         'Content-Type': 'application/json',
      },
      data
   );
},

async editUserPicture(data) {
   return await this.createRequest(
      'me/image',
      'PATCH',
      {
         Authorization: `Bearer ${Cookies.get('accessToken')}`,
         Accept: '*/*',
         'Content-Type': 'application/json',
      },
      data
   );
},
```

```
    async uploadPicture(payload) {
        return await this.createRequest(
            'images/users',
            'POST',
            {
                Authorization: `Bearer ${Cookies.get('accessToken')}`,
                Accept: '*/*',
            },
            payload,
            true
        );
    },

    async getUserLikes() {
        return await this.createRequest('me/likes', 'GET', {
            Authorization: `Bearer ${Cookies.get('accessToken')}`,
        });
    },

    async likeUser(userId, gameId) {
        return await this.createRequest(
            `users/${userId}/likes/${gameId}`,
            'POST',
            {
                Authorization: `Bearer ${Cookies.get('accessToken')}`,
                Accept: '*/*',
                'Content-Type': 'application/json',
            }
        );
    },

    async keepOnline() {
        return await this.createRequest('me/last-visited-date', 'PATCH', {
            Authorization: `Bearer ${Cookies.get('accessToken')}`,
            Accept: '*/*',
            'Content-Type': 'application/json',
        });
    },
};

export default API;
```

Далі компоненти відображають дані, які приходять до них з АПІ-клієнту або з Redux-сховища. Також компоненти монтують свої стилі. Для прикладу розглянемо компонент Profile, що відображає сторінку користувача. Цей компонент має виклик АПІ-клієнта, після чого, коли отримує відповідь від серверу, передає отримані дані у свої дочірні компоненти. У той же час стилі його

компоненту зберігає створений бібліотекою styled-components стилізований компонент. Після монтажу цього компоненту на сторінку усі стилі, що використовує цей компонент, перетворюються на звичайні CSS правила.

```
import React, { useEffect, useState } from 'react';
import styled from 'styled-components';
import API from '../../api/apiClient';
import UserDescription from '../UserDescription';
import UserGames from '../UserGames';

const StyledProfile = styled.div`
  display: flex;
  width: 100%;
  flex-direction: column;
  min-height: 100vh;
  align-items: center;
`;

export default function Profile() {
  const [userGames, setUserGames] = useState([]);
  const [nickname, setNickname] = useState();
  const [avatar, setAvatar] = useState();

  useEffect(() => {
    API.getUserInfo()
      .then((data) => data.json())
      .then((body) => {
        setUserGames(body.games);
        setNickname(body.nickname);
        setAvatar(body.image);
      });
  }, []);

  return (
    <StyledProfile>
      <UserDescription nickname={nickname} avatar={avatar} />
      <UserGames userGames={userGames} />
    </StyledProfile>
  );
}
```

Бібліотека Redux у свою чергу обробляє події зміни сховища та оновлює його. Так, якщо до центру обробки подій Redux потрапить подія LOAD_GAMES, то Redux завантажить ігри, які надійшли до нього з цією подією у своє сховище.

```
import { LOAD_GAMES, LOAD_USERS } from './actions';

const initialState = {
```

```
  games: [],
  users: [],
};

export default function mainReducer(store = initialState, action) {
  switch (action.type) {
    case LOAD_GAMES:
      return {
        ...store,
        games: action.payload,
      };
    case LOAD_USERS:
      return {
        ...store,
        users: action.payload,
      };
    default:
      return store;
  }
}
```

У компонентах, де необхідно використати дані з бібліотеки Redux створюється метод для виклику сховища та отримання конкретних даних від нього, не завантажуючи при цьому зайві. Наприклад, у файлі компоненту SearchTable.jsx ми підключаємося до нашого сховища завдяки методу connect, що огортає експорт нашого компоненту. Метод connect надає бібліотека Redux. Після чого ми можемо використовувати дані з сховища Redux як атрибути нашого компоненти, щоб відобразити їх користувачу.

```
import React, { useEffect, useState } from 'react';
import { connect } from 'react-redux';
import { v4 as uuid } from 'uuid';
import styles from './SearchTable.module.css';
import PropTypes from 'prop-types';
import GameCard from '../GameCard';
import { Link } from 'react-router-dom';
import API from '../../api/apiClient';

function SearchTable(props) {
  const [games, setGames] = useState([]);

  useEffect(() => {
    API.getUserInfo()
      .then((data) => data?.json())
      .then((body) => {
        console.log(body);
```

```
            const games = props.games.filter((game) => {
              return body.games?.some(
                (userGame) => userGame.id === game.id
              );
            });

            setGames(games);
          });
    }, [props.games]);

    return (
      <div className={styles.SearchTable}>
        {games.map((game) => {
          return (
            <Link key={uuid()} to={'/search/' + game.id}>
              <GameCard
                gameTitle={game.title}
                imageLarge={game.imageLarge}
                icon={game.icon}
                smallImage={game.imageSmall}
              />
            </Link>
          );
        })}
      </div>
    );
}

SearchTable.propTypes = {
    games: PropTypes.array,
};

const mapStateToProps = (state) => {
    return {
      games: state.games,
    };
};

export default connect(mapStateToProps)(SearchTable);
```

У файлі App.jsx ми збираємо усі компоненти веб-додатку. Тут ми налаштовуємо бібліотеку React Router, що дозволяє зв’язувати сторінки сайту з відповідними адресами. Також ми налаштовуємо різні види доступу до сторінок сайту за допомогою ідентифікації користувача за Cookies браузеру. Таким чином ми маємо два типи сторінок: AuthorizedRoute та UnAuthorizedRoute, тобто сторінка, яка доступна лише користувачам, які увійшли на сайт, та сторінка, яка

доступна лише користувачам, які не зайшли на сайт відповідно. Також при завантаженні усього веб-додатку ми повинні запустити процедуру оновлення статусу користувача кожні чотири хвилини, щоб можна було ідентифікувати користувача як онлайн користувача. Також у цьому файлі при завантаженні усього сайту ми отримує необхідні дані, які далі відсилаємо до сховища Redux, щоб надалі використовувати дані звідти.

```
import React, { useEffect } from 'react';
import MainContainer from '../MainContainer/MainContainer';
import { Route, Switch, useLocation, Redirect } from 'react-router-dom';
import { TransitionGroup, CSSTransition } from 'react-transition-group';
import { connect } from 'react-redux';
import Cookies from 'js-cookie';

import Header from '../Header';
import Footer from '../Footer';
import Login from '../Login';

import loginBG from '../../assets/images/loginBG.jpg';
import patternBG from '../../assets/images/pattern.png';
import pattern1BG from '../../assets/images/pattern1.png';
import homeBG from '../../assets/images/homeBg.jpg';
import registerBG from '../../assets/images/registerBG.jpg';
import Registration from '../Registration';
import loadData from './connectStore';
import PropTypes from 'prop-types';
import SearchTable from '../SearchTable';
import SearchChoose from '../SearchChoose';
import Search from '../Search';
import Profile from '../Profile/Profile';
import UserLikes from '../UserLikes';
import API from '../../api/apiClient';
import OnlineSearch from '../Search/OnlineSearch';
import HomePage from '../HomePage';

const AuthorizedRoute = ({ path, children }) => {
  return (
    <Route path={path}>
      {Cookies.get('accessToken') ? children : <Redirect to="/login" />}
    </Route>
  );
};

const UnAuthorizedRoute = ({ path, children }) => {
  return (
    <Route path={path}>
      {Cookies.get('accessToken') ? <Redirect to="/home" /> : children}
```

```
      </Route>
    );
};

UnAuthorizedRoute.propTypes = {
    path: PropTypes.string,
    children: PropTypes.node,
};

AuthorizedRoute.propTypes = {
    path: PropTypes.string,
    children: PropTypes.node,
};

function App(props) {
    const location = useLocation();

    const startTimer = () => {
        if (Cookies.get('accessToken')) {
            API.keepOnline();
            setInterval(() => {
                API.keepOnline();
            }, 1000 * 60 * 4);
        }
    };

    useEffect(() => {
        props.loadData();
        startTimer();
    }, []);

    return (
        <div className="appWrapper">
            <Header />
            <TransitionGroup>
                <CSSTransition
                    key={location.key}
                    classNames="fade"
                    timeout={300}
                >
                    <Switch location={location}>
                        <Route path="/home" exact>
                            <MainContainer bg={homeBG}>
                                <HomePage />
                            </MainContainer>
                        </Route>

                        <AuthorizedRoute path="/search" exact>
                            <MainContainer isCentered={false} bg={pattern1BG}>
                                <SearchTable />
                            </MainContainer>
                        </AuthorizedRoute>
```

```
            <AuthorizedRoute path="/search/:id" exact>
              <MainContainer bg={pattern1BG}>
                <SearchChoose />
              </MainContainer>
            </AuthorizedRoute>

            <AuthorizedRoute path="/me" exact>
              <MainContainer bg={pattern1BG}>
                <Profile />
              </MainContainer>
            </AuthorizedRoute>

            <AuthorizedRoute path="/search/:id/common" exact>
              <MainContainer bg={patternBG}>
                <Search />
              </MainContainer>
            </AuthorizedRoute>

            <AuthorizedRoute path="/search/:id/online" exact>
              <MainContainer bg={patternBG}>
                <OnlineSearch />
              </MainContainer>
            </AuthorizedRoute>

            <AuthorizedRoute path="/likes" exact>
              <MainContainer bg={pattern1BG}>
                <UserLikes />
              </MainContainer>
            </AuthorizedRoute>

            <UnAuthorizedRoute path="/login" exact>
              <MainContainer bg={loginBG}>
                <Login />
              </MainContainer>
            </UnAuthorizedRoute>

            <UnAuthorizedRoute path="/registration" exact>
              <MainContainer bg={registerBG}>
                <Registration />
              </MainContainer>
            </UnAuthorizedRoute>

            <Route path="/">
              <Redirect to="/home" />
            </Route>
          </Switch>
        </CSSTransition>
      </TransitionGroup>
      <Footer />
    </div>
  );
```

```
}
const mapDispatchToProps = {
  loadData,
};
App.propTypes = {
  loadData: PropTypes.func,
};
export default connect(null, mapDispatchToProps)(App);
```

2.5. Обґрунтування та організація вхідних та вихідних даних програми

У якості вхідних даних веб-додаток може приймати дані користувача, які він може надати на сторінці реєстрації, логіну та на сторінці профілю. У якості вихідних даних користувач отримує сторінки веб-додатку, на якій він може здійснювати стандартні браузерні операції.

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Веб-додаток завдяки розповсюдженню веб-браузерів на усіх можливих платформах може бути відкритий на будь-якій операційній системі. Також веб-додаток адаптований під користування на різних пристроях, таких як планшети, телефони, настільні комп’ютери.

За навантаженням додаток є дуже праце спроможним, бо всю роботи для користувача виконує браузер та система має завантаження не більше, чим під час звичайного використання браузеру. Якщо пристрій користувача спроможній працювати з обраним браузером без проблем, тоді додаток буде працювати стабільно та швидко.

2.6.2. Використані програмні засоби

Для зручного користуванням веб-додатком важливо відкривати його у браузерах, що підтримують роботу React. Такими є усі сучасні браузери (Chrome, Mozilla, Brave, Edge, Opera). Але додаток може не працювати у старих браузерах, таких як Internet Explorer.

Користувач може мати будь-яку операційну систему на своєму пристрої (Window, MacOS, Linux та дистрибутиви, Android, iOS), якщо вона підтримує хоча б один необхідний для роботи додатку браузер.

2.6.3. Виклик та завантаження програми

Для завантаження веб-додатку на локальному пристрої необхідно у папці проекту використовуючи Git Bash, консоль розробника або будь-яку іншу програмну консоль викликати команду npm start, після чого чекати завершення збірки додатку. Після збірки файлів додаток відкриється у браузері на локальному сервері.

Для завантаження веб-додатку у звичайний спосіб достатньо лише перейти за прямим посиланням на веб-додаток (https://uaky-client.herokuapp.com/).

2.6.4. Опис інтерфейсу користувача

Після відкриття веб-додатку користувач потрапляє на домашню сторінку сайту, що зображена на рис. 2.9.

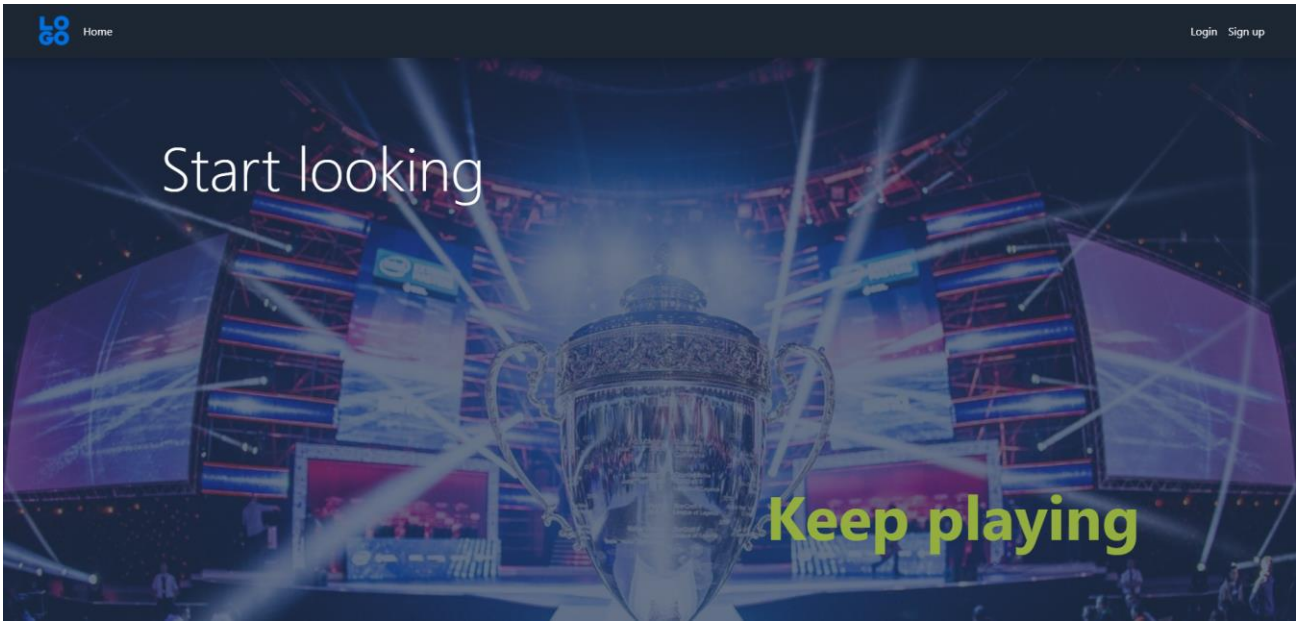

Рис. 2.9. Домашня сторінка сайту

Користувач може використовувати шапку сайту для навігації. На рис. 2.10 зображено шапку сайту з елементами управління на версії для комп’ютерів.

Рис. 2.10. Шапка сайту на комп’ютерній версії

На телефонах шапка сайту виглядає інакше. Замість шапки сайту користувач може натиснути на спеціальну кнопку-гамбургер, що зображено на рис. 2.11.

Рис. 2.11. Кнопка-гамбургер на мобільній версії додатку

При натисканні на кнопку гамбургер користувач може використовувати панель навігації, що має абсолютно такі ж пункти, як і шапка сайту для навігації по додатку. Панель навігації на мобільних пристроях зображено на рис. 2.12.

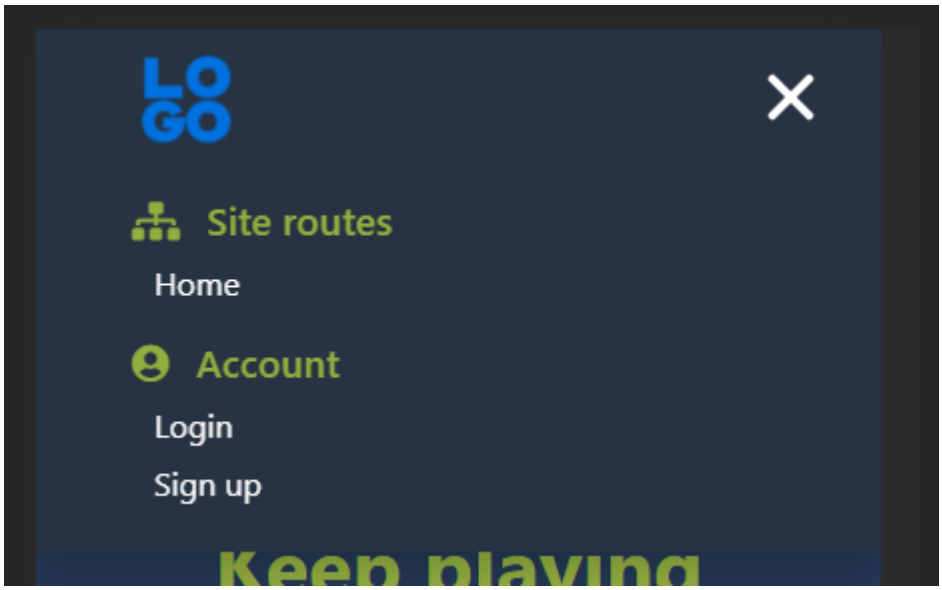

Рис. 2.12. Панель навігації по сайту на мобільних пристроях

Щоб закрити панель, користувач може знову натиснути на кнопку-гамбургер, що у відкритому стані панелі виглядає як кнопка закриття.

Для переходу на сторінку реєстрації користувач може обрати у навігаційному меню опцію Sign up. Після цього користувач потравляє на сторінку реєстрації. Сторінка реєстрації зображена на рис. 2.13 для комп’ютерної версії та на рис. 2.14 на мобільній версії.

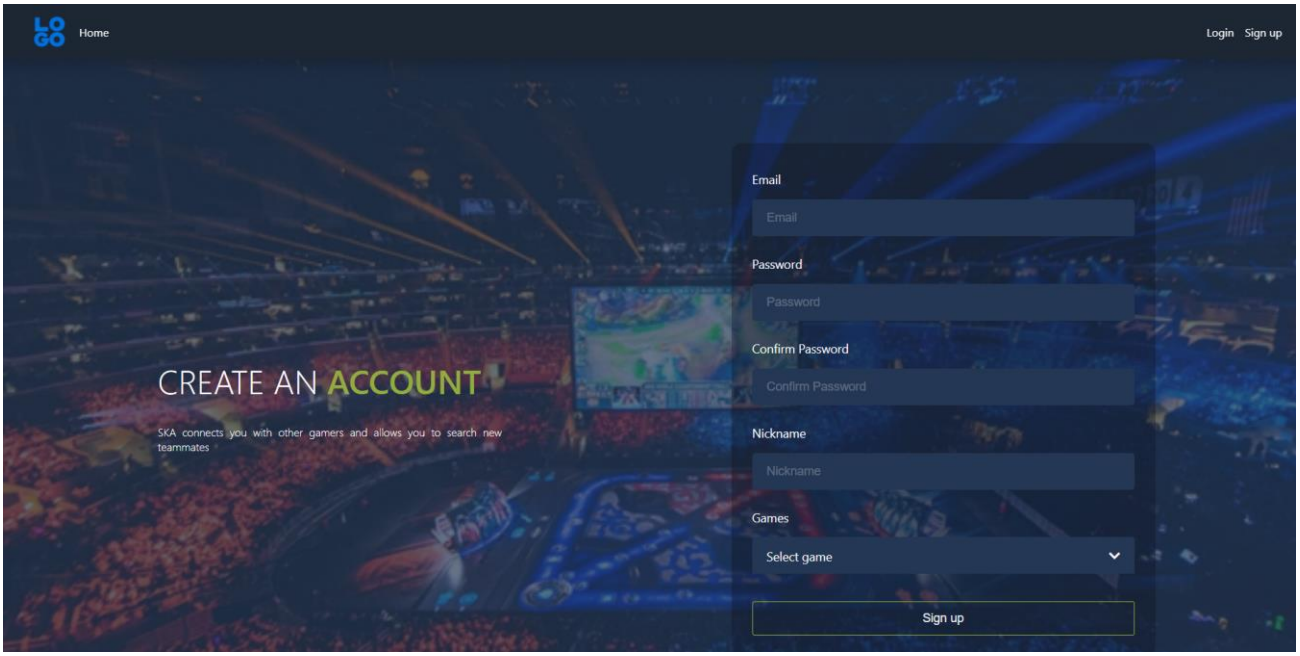

Рис. 2.13. Сторінка реєстрації на комп’ютерній версії

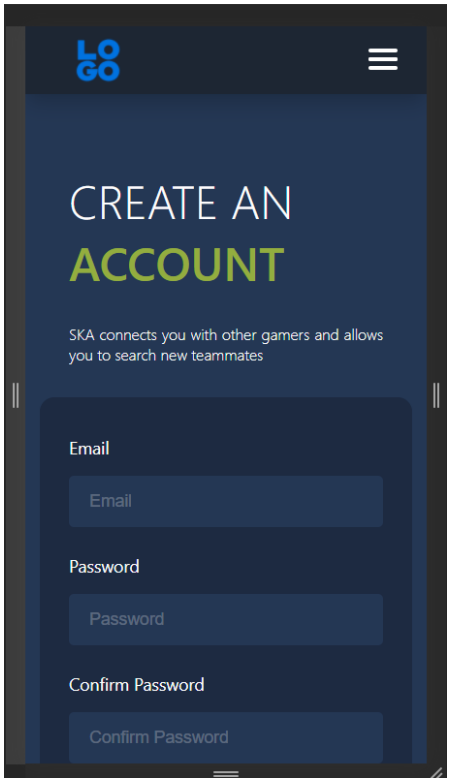

Рис. 2.14. Сторінка реєстрації на мобільній версії

Після переходу на сторінку реєстрації користувач може почати заповнювати форму реєстрації, де кожне поле форми має свою інтуїтивно зрозумілу назву, завдяки чому для користувача немає проблеми для заповнення цих полів. Наприклад, користувач може ввести адресу своєї електронної пошти у полі Email. Коли користувач обирає поле, воно відповідно відсвічується, щоб користувач міг відчувати наслідки своїх дій, що зображено на рис. 2.15.

Рис. 2.15. Поле, яке обирає користувач, підсвічується.

Коли користувач заповнив усі поля, він може додати ігри, за якими у майбутньому він зможе шукати товаришів по команді. Для цього він з випадаючого списку (рис. 2.16) обирає необхідну гру, після чого відкривається додаткова форма для заповнення інформації про гру (рис 2.17).

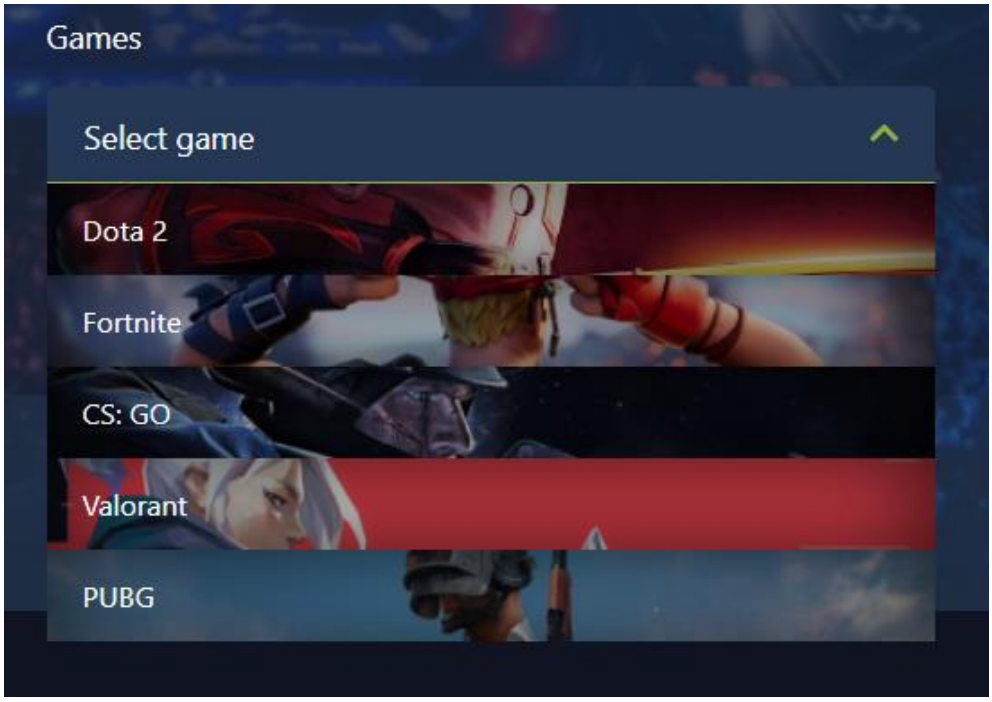

Рис. 2.16. Список ігор, які може додати користувач

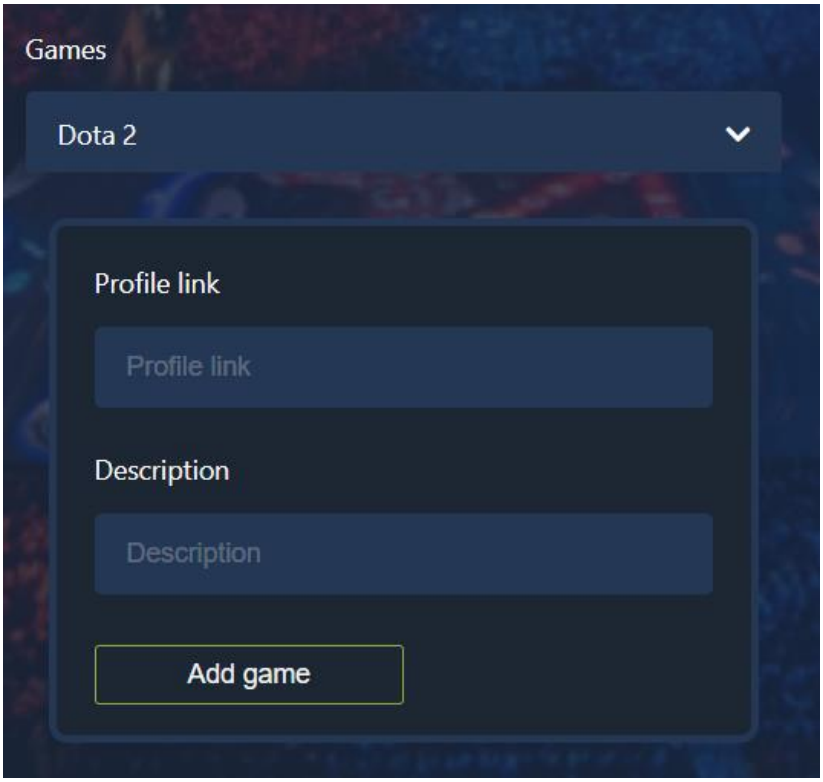

Рис. 2.17. Форма для інформації про гру.

Після заповнення інформації про гру користувач може додати її натиснувши на кнопку Add game. Після цього під випадаючим списком з’явиться назва та іконка відповідної гри (рис 2.18), щоб користувач міг зрозуміти, які ігри він вже додав. Після додавання гри користувач може повторити описані вище дії для додавання безлічі ігор.

Рис. 2.18. Назва та іконка доданих ігор

Якщо користувач хочу видалити вже додану гру або видалити її, щоб додати інші дані, він може натиснути на значок закриття поряд з назвою гри, що зображено на рис 2.19.

Рис. 2.19. Значок видалення доданої гри

На будь-якому етапі реєстрації якщо користувач уведе невірні або некоректні дані у форму він зможе побачити після відправки форми, що саме пішло не так у ході заповнення. Помилки відображаються під відповідним полем. Наприклад, користувач побачить помилку, якщо повтор паролю не співпадає з основним паролем, що зображено на рис. 2.20.

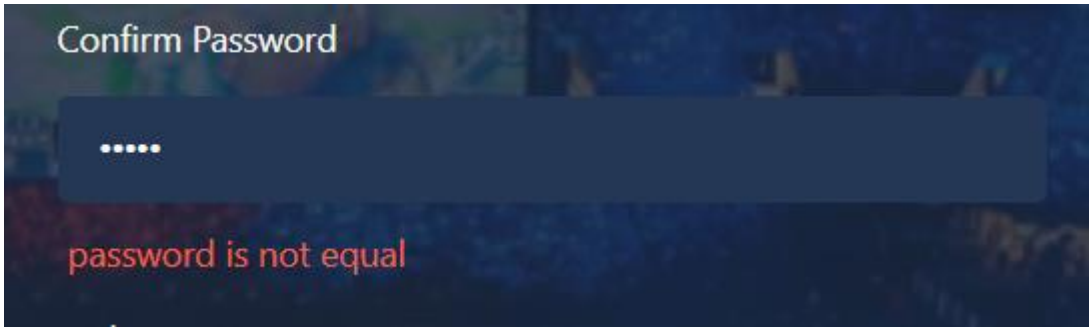

Рис. 2.20. Зображення помилки для користувача

Після вдалого заповнення користувач може натиснути кнопку Sign up, щоб завершити реєстрацію. Якщо реєстрація завершена успішно, користувач опиниться на сторінці логіну (на рис. 2.21 зображено комп'ютерну версію сторінки логіну, на рис. 2.22 – мобільну).

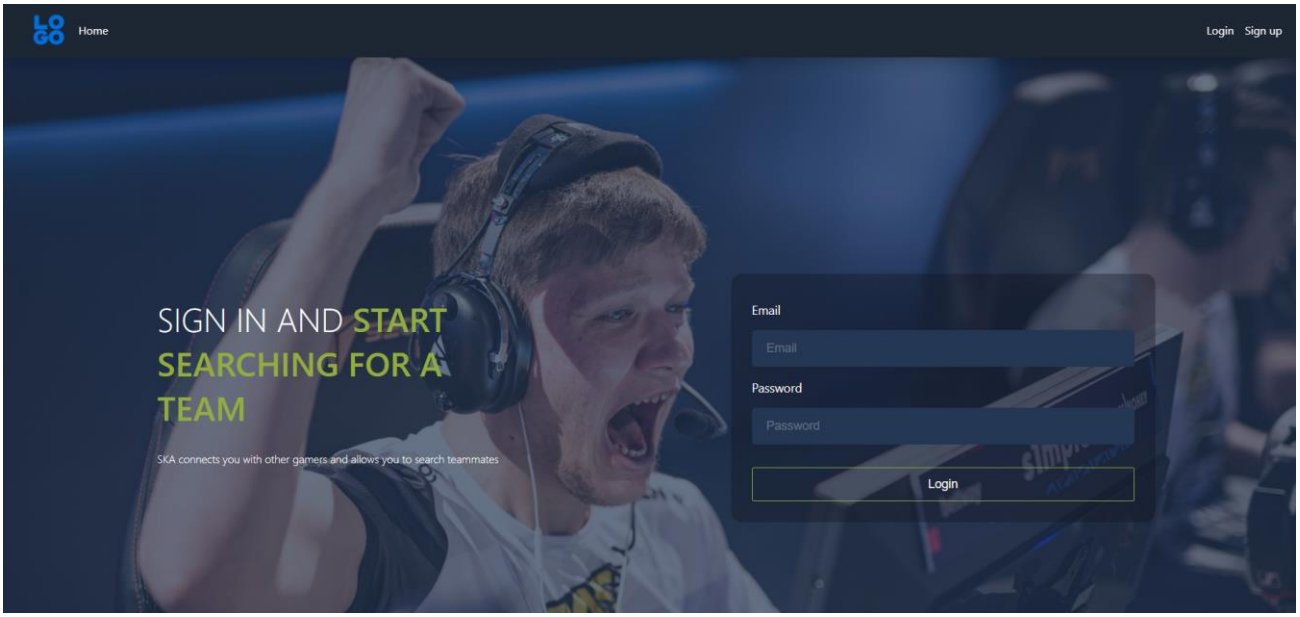

Рис. 2.21. Комп'ютерна версія сторінки логіну

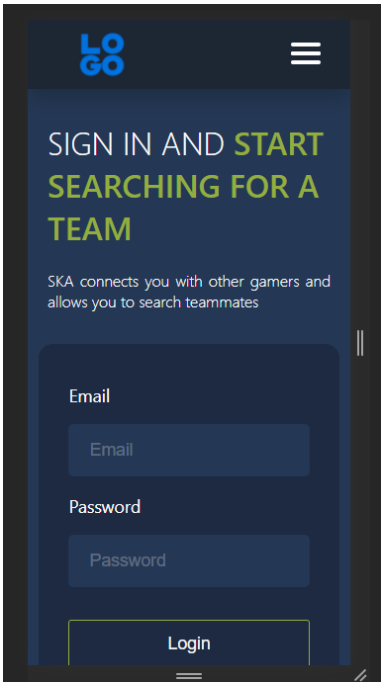

Рис. 2.22. Мобільна версія сторінки логіну

Процес логіну відбувається таким же чином, як на сторінці реєстрації. Після заповнення відповідної форми логіну, користувач може натиснути на кнопку Login, після чого при вдалому логіні користувач опиниться на головній сторінці.

Після логіну шапка сайту (та відповідно меню навігації у мобільній версії) змінюються. Кнопки Login та Registration зникають. З’являються кнопки Search Teammates, My likes, Profile та Logout. Вид шапки сайту після логіну зображено на рис. 2.23. Вид меню навігації на мобільних пристроях зображено на рис. 2.24.

Рис. 2.23. Шапка сайту після логіну

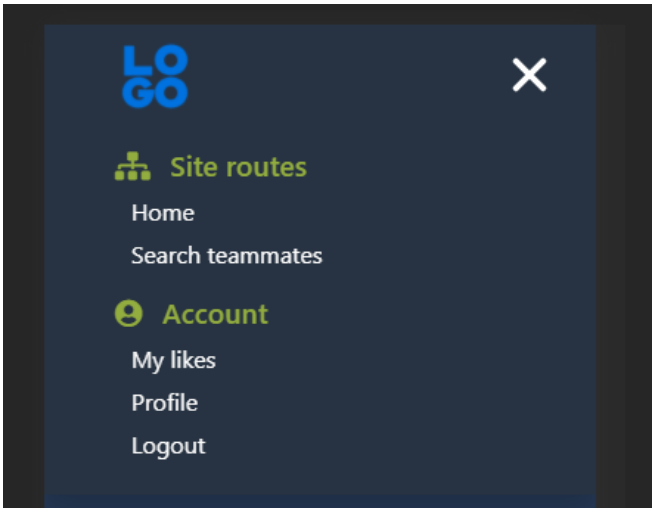

Рис. 2.24. Меню навігації на мобільній версії після логіну

Якщо користувач хоче вийти зі свого профілю, він може натиснути на кнопку Logout. Після цього стан сайт знову буде як при першому відвідуванні сайту.

Для переходу на сторінку профілю користувача можна натиснути на кнопку Profile. Сторінка профілю користувача зображена на рис. 2.25 для комп'ютерної версії та на рис. 2.26 для мобільної версії.

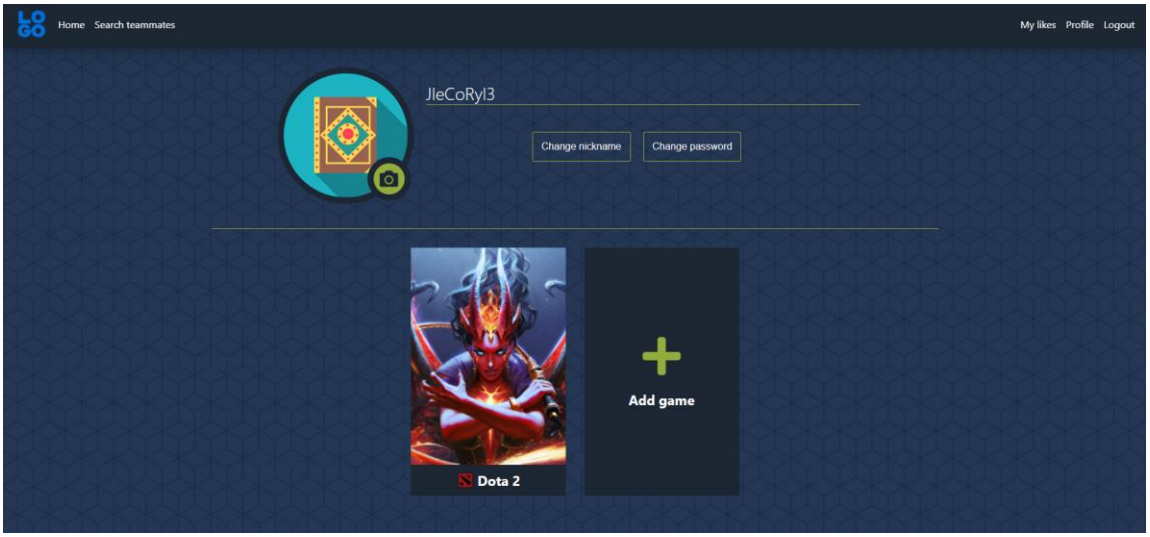

Рис. 2.25. Сторінка профілю на комп'ютерній версії сайту

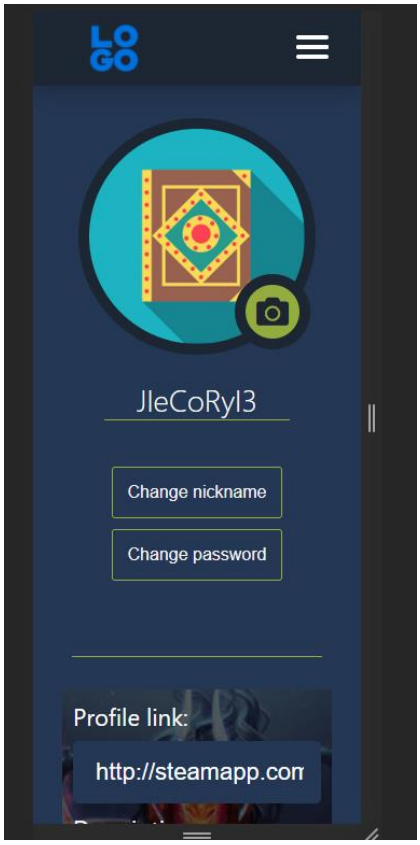

Рис. 2.26. Сторінка профілю користувача на мобільній версії

На цій сторінці зображено картинку користувача, псевдонім та ігри, які користувач собі додав для пошуку. За замовченням користувач отримує одну з восьми випадкових картинок для свого профілю (рис. 2.27).

Рис. 2.27. Картинка профілю користувача

Якщо користувач хоче змінити картинку, він може натиснути на кнопку зміни, що зображено на рис. 2.28.

Рис. 2.28. Кнопка зміни користувача

Після чого користувач може обрати картинку зі свого пристрою. Якщо картинку вдалося змінити, сторінка перезавантажиться з новою картинкою, що зображено на рис. 2.29 та рис. 2.30.

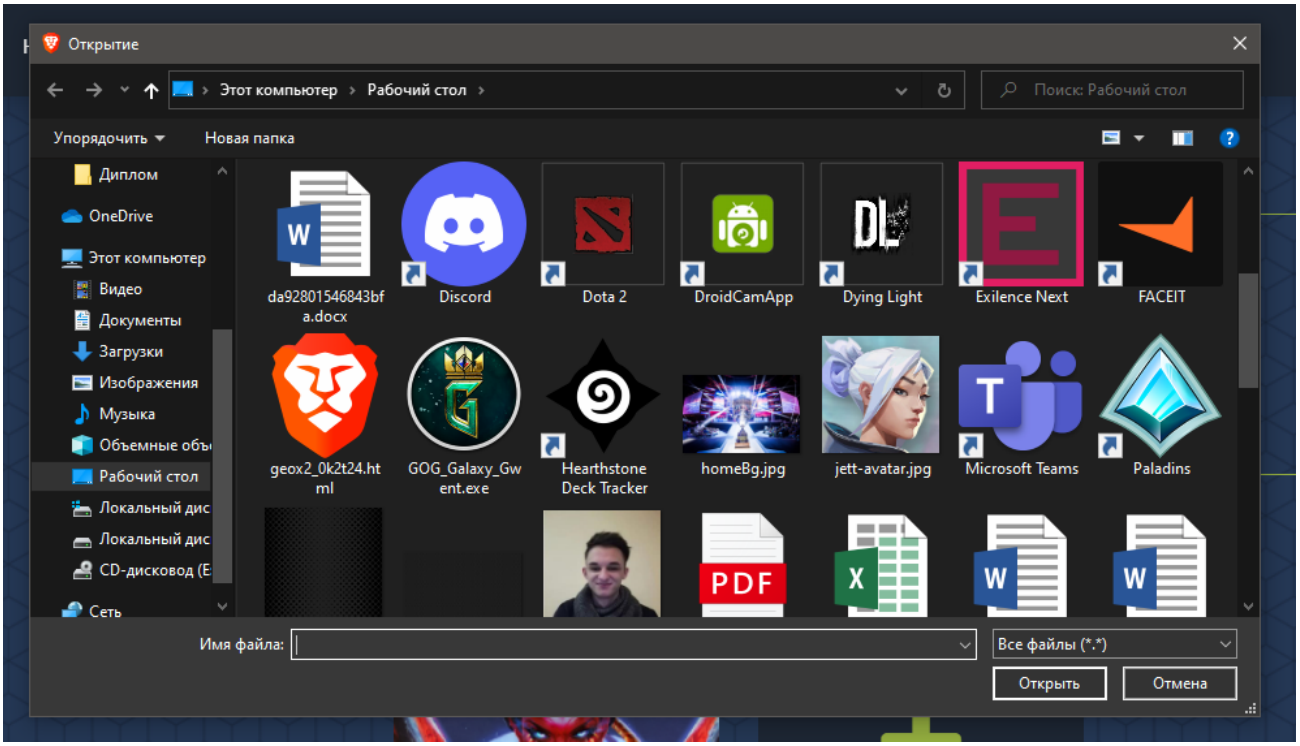

Рис. 2.29. Меню вибору картинки

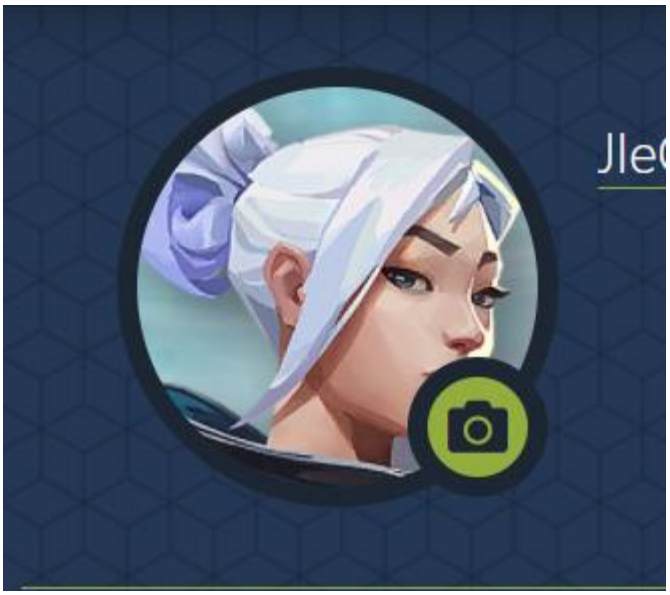

Рис. 2.30. Змінена картинка

Також користувач може змінити пароль або псевдонім, натиснувши на відповідні кнопки. Після натискання відкриється контекстне меню, де користувач

може змінити що потрібно (рис. 2.31).

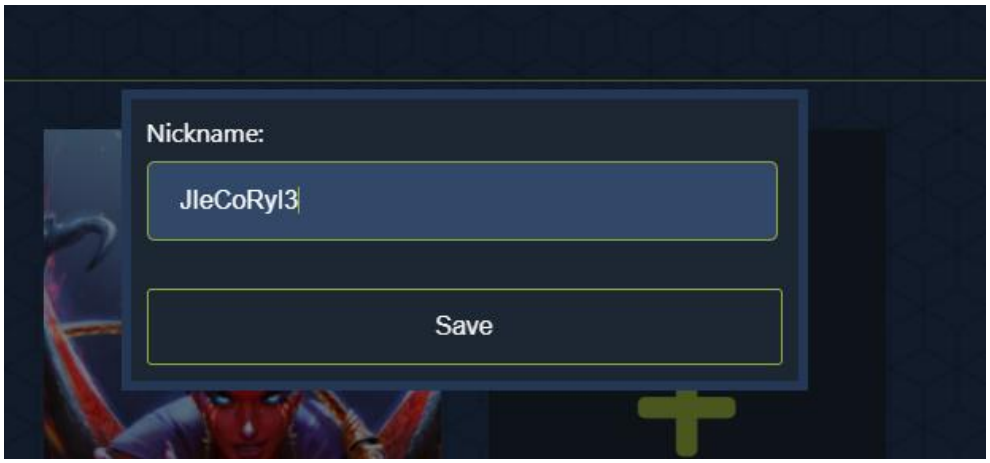

Рис. 2.31. Контекстне меню зміни псевдоніму

Якщо користувач хоче закрити контекстне меню, він може натиснути просто за межами цього контекстного меню.

Якщо користувач хоче змінити дані гри, яку він вже додав, він може навести на картку гри, після чого відобразяться відповідні поля для зміни (рис. 2.32). На мобільній версії ці поля показуються за замовченням.

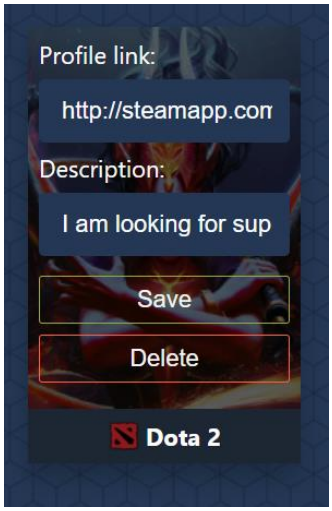

Рис. 2.32. Поля для редагування гри

Після зміни інформації про гру користувач може натиснути кнопку Save щоб зберегти зміни.

Якщо користувач хоче видалити гру, він може натиснути на кнопку Delete.

Якщо користувач хоче додати іншу гру, він може натиснути на кнопку додавання нової гри, що зображено на рис. 2.33.

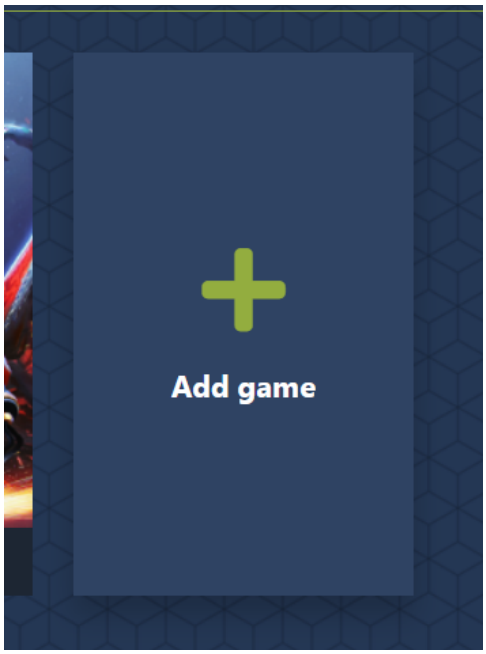

Рис. 2.33. Кнопка додавання картинки

Після натискання відкривається контекстне меню з випадаючим списком, таким самим як на формі реєстрації, та полями для заповнення. Після заповнення користувач може натиснути Add, після чого сторінка перезавантажиться та нова гра буде додана до списку вже існуючих. Форма додавання гри зображена на рис. 2.34.

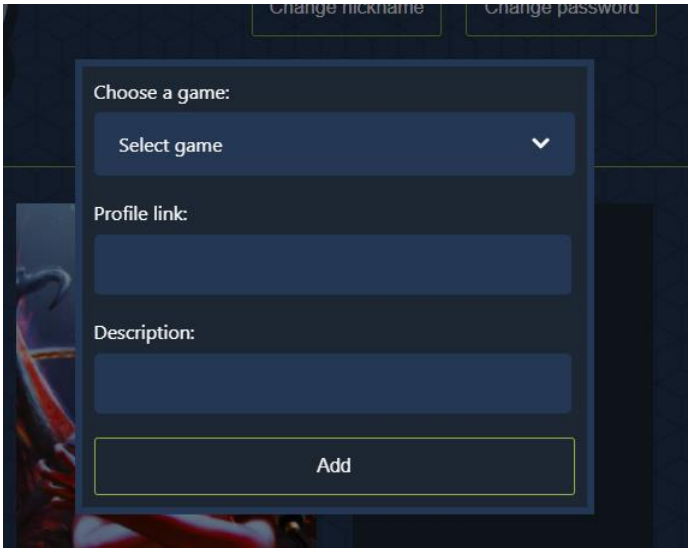

Рис. 2.34. Форма додавання гри

Для того, щоб почати пошук товаришів по команді користувач може натиснути на кнопку Search Teammates. На сторінці пошуку будуть відображатися ігри, які додані у користувача. Сторінка пошуку зображена на рис. 2.35.

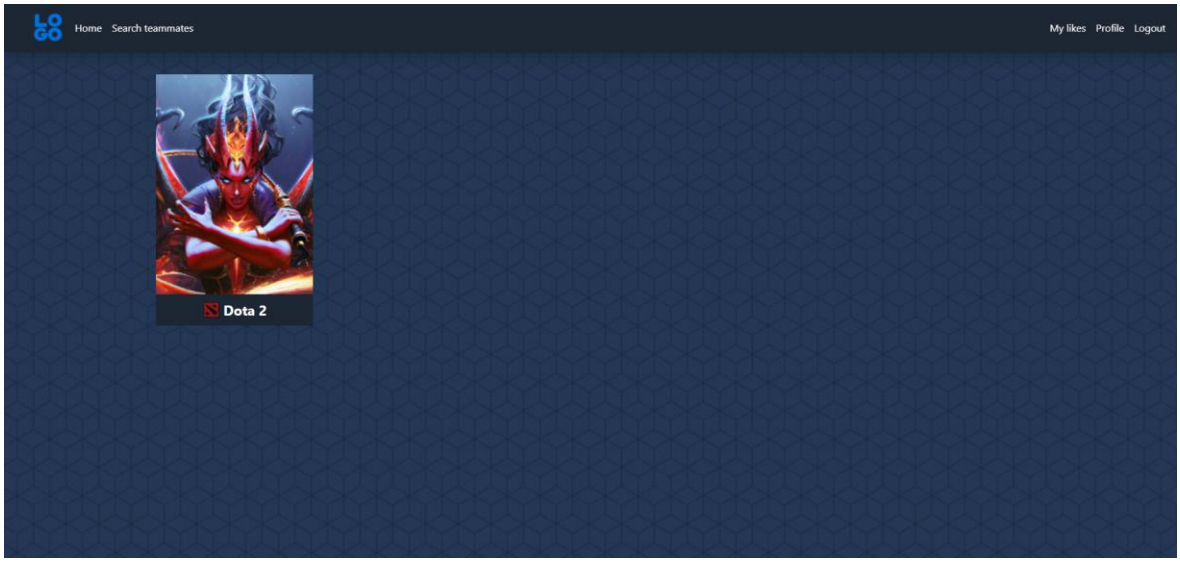

Рис. 2.35. Сторінка вибору ігор для пошуку

Після обрання гри, за якою користувач буде шукати товаришів по команді він потравляє на сторінку вибору типу пошуку, що зображено на рис. 2.36 для комп’ютерної версії та рис. 2.37 для мобільної версії.

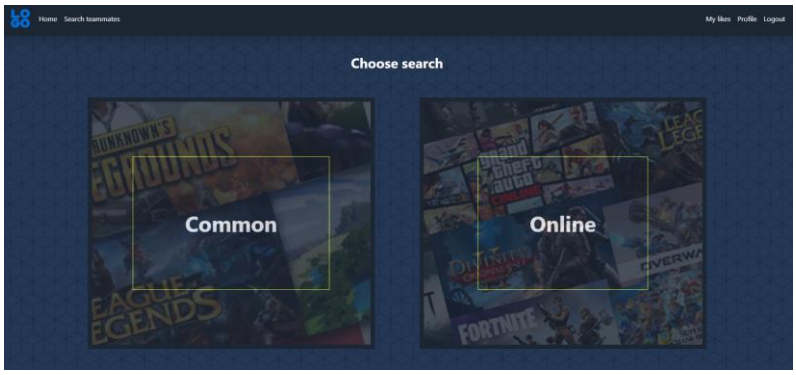

Рис. 2.36. Сторінка вибору типу пошуку на комп’ютерній версії

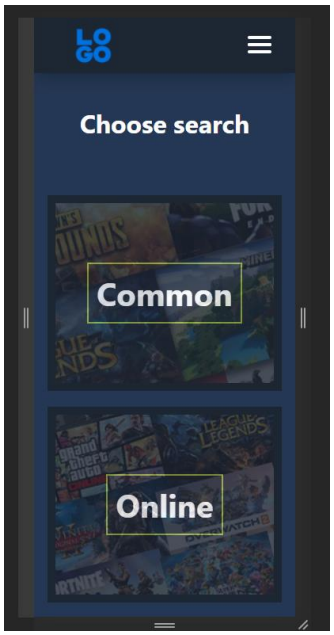

Рис. 2.37. Сторінка вибору типу пошуку на мобільній версії

Користувач може обрати один з двох типів пошуку: Common та Online. Звичайний пошук дозволяє шукати товаришів по команді серед користувачів, які

у цілому зареєстровані на сайті. Онлайн пошук дозволяє шукати товаришів по команді серед користувачів, які саме під час пошуку знаходяться на сайті.

Після обрання типу пошуку користувач опиняється на сторінці пошуку з картками користувачів (рис. 2.38). Якщо користувачеві сподобався гравець, він може свайпнути його управо, якщо не сподобався – уліво.

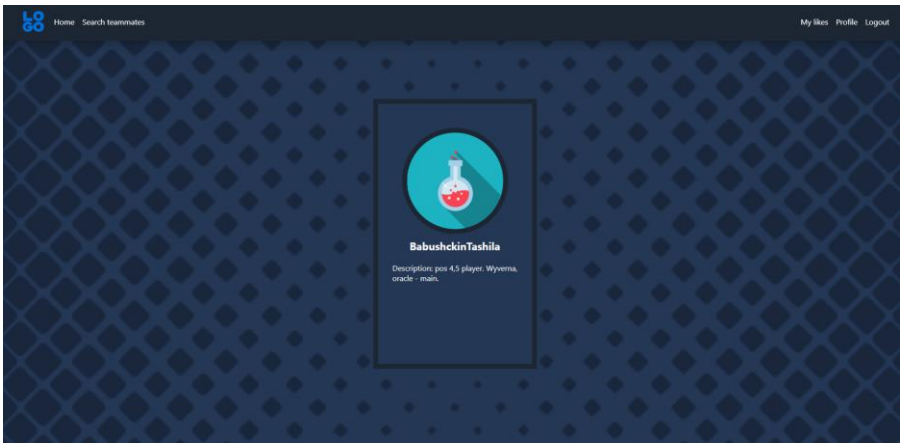

Рис. 2.38. Сторінка пошуку з картками користувачів

Якщо користувач перегорнув вже усіх користувачів, він може запустити пошук серед тих користувачів, що йому не сподобалися, наново. Кнопка перезавантаження зображена на рис. 2.39. Якщо користувачеві сподобався гравець, він більше не з’явиться у нього у пошуку.

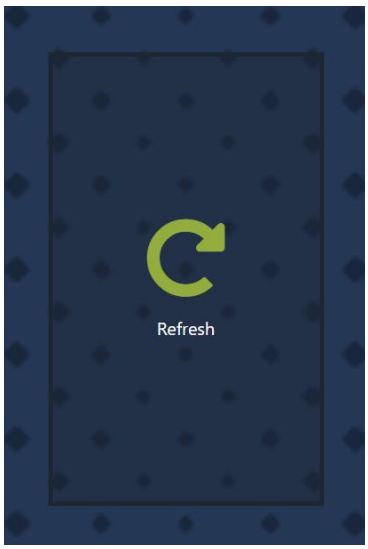

Рис. 2.39. Кнопка скидання користувачів

Щоб подивитися, яким гравцям сподобався користувач, можна натиснути на кнопку My likes у шапці сайту. Після чого користувач потрапляє на сторінку вподобань, яка зображена на рис. 2.40.

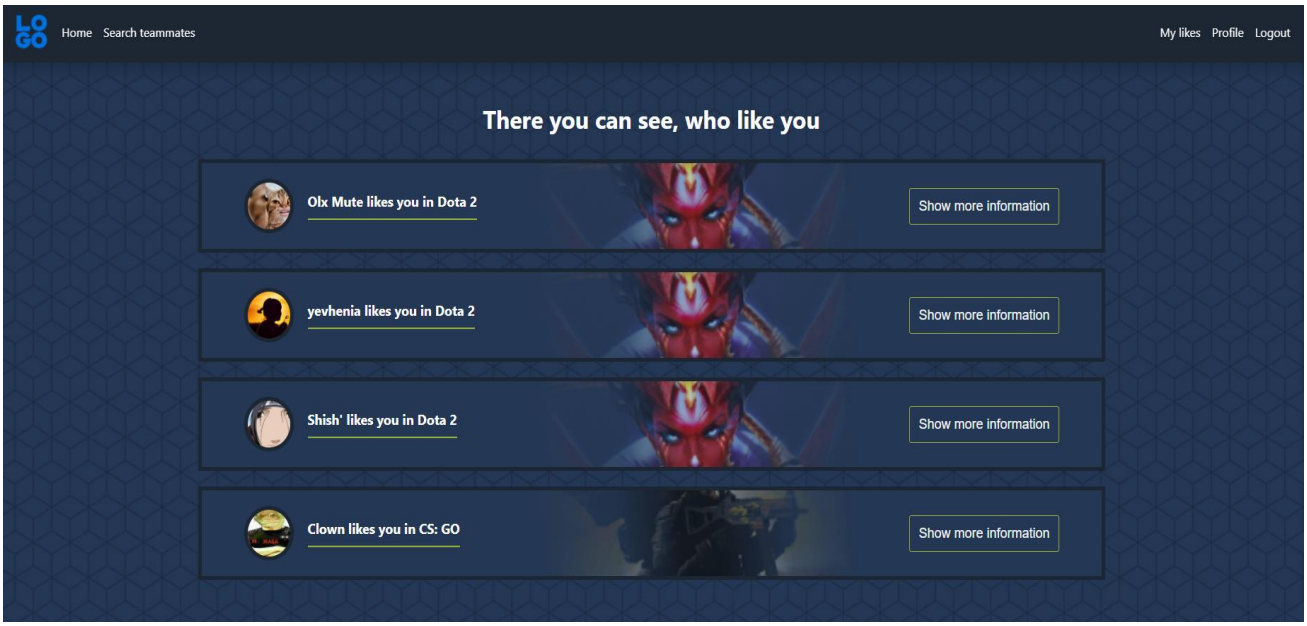

Рис. 2.40. Сторінка вподобань користувача

На сторінці вподобань користувач може бачити кому саме та в якій грі він сподобався. Щоб подивитися більш детальну інформацію про користувача можна натиснути на кнопку Show more information. Після цього з’являється модальне вікно з детальною інформацію про користувача, що зображено на рис. 2.41.

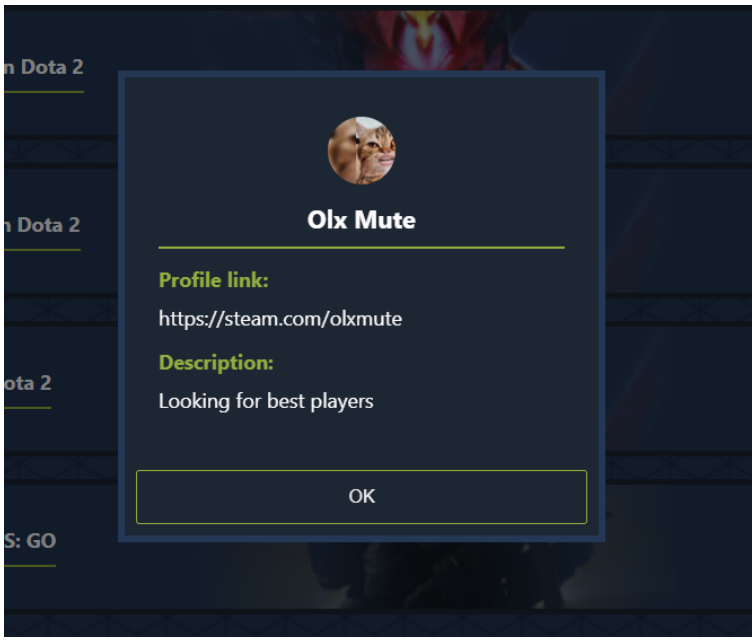

Рис. 2.41. Модальне вікно детальної інформації

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

Вихідні дані розробки програмного забезпечення:

а) передбачуване число операторів – 1730;

б) коефіцієнт складності програми – 1,75;

в) коефіцієнт кореляції програми в ході її розробки – 0,06;

г) середня годинна заробітна плата програміста, грн/год – 111,27;

д) коефіцієнт кваліфікації програміста, обумовлений від стажу – 1,2;

е) вартість машино-години ЕОМ, грн/год – 4,2.

3.1. Визначення трудомісткості розробки програмного забезпечення

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 60 людино-годин);

t_u – витрати праці на дослідження алгоритму рішення задачі,

t_a – витрати праці на розробку блок-схеми алгоритму,

t_n – витрати праці на програмування по готовій блок-схемі,

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ,

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C(1 + p), \text{ людино-годин,} \qquad (3.2)$$

де q – передбачуване число операторів,

C – коефіцієнт складності програми,

p – коефіцієнт корекції програми в ході її розробки.

$$Q = 1730 \cdot 1{,}75 \cdot (1 + 0{,}06) = 3209{,}15$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}, \text{ людино-годин,} \qquad (3.3)$$

де B, яке дорівнює 1,2, – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі,

k, яке дорівнює 1,2, – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності.

$$t_u = \frac{3209{,}15 \cdot 1{,}2}{82 \cdot 1{,}2} = 39{,}13,$$ людино-годин.

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20..25) \cdot k}; \quad (3.4)$$

$$t_a = \frac{3209{,}15}{22 \cdot 1{,}2} = 121{,}55,$$ людино-годин.

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20..25) \cdot k}, \quad (3.5)$$

$$t_n = \frac{3209{,}15}{21 \cdot 1{,}2} = 127{,}34,$$ людино-годин.

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5) \cdot k}; \quad (3.6)$$

$$t_{отл} = \frac{3209{,}15}{5 \cdot 1{,}2} = 534{,}85, \text{ людино-годин,}$$

– за умови комплексного налагодження завдання:

$$t_{отл}^{к} = 1{,}4 \cdot t_{отл}; \quad (3.7)$$

$$t_{отл}^{к} = 1{,}4 \cdot 534{,}85 = 748{,}79, \text{ людино-годин.}$$

Витрати праці на підготовку документації:

$$t_{д} = t_{др} + t_{до}, \quad (3.8)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису

$$t_{др} = \frac{Q}{(15..20) \cdot k}; \quad (3.9)$$

$$t_{др} = \frac{3209{,}15}{20 \cdot 1{,}2} = 133{,}71, \text{ людино-годин.}$$

де $t_{до}$ – трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0{,}75 \cdot t_{др}; \tag{3.10}$$

$$t_{до} = 0{,}75 \cdot 133{,}71 = 100{,}28 \text{ , людино-годин.}$$

$$t_{д} = 133{,}71 + 100{,}28 = 233{,}99 \text{ , людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

$$t_{д} = 60 + 39.13 + 121{,}55 + 127{,}34 + 534{,}85 + 233{,}99 = 1116{,}86 \text{ , людино-годин.}$$

У результаті ми розрахували, що в загальній складності необхідно 1116,86 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного для налагодження програми на ЕОМ.

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \quad (3.11)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн,} \quad (3.12)$$

де t – загальна трудомісткість, людино-годин,

$C_{ПР}$ – середня годинна заробітна плата програміста, грн/година. Середня заробітна плата становить 700$ [2], що у перерахунку на грн/год виходить 111,27.

$$З_{ЗП} = 1116,86 \cdot 111.27 = 124273 \text{ , грн.}$$

$З_{МВ}$ – вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot C_{МЧ}, \text{ грн,} \quad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$C_{МЧ}$ – вартість машино-години ЕОМ, грн/год., дорівнює 4,2 [1].

$$З_{МВ} = 534,85 \cdot 4,2 = 2246, \text{ грн,}$$

$$K_{ПО} = 123273 + 2246 = 125519 \text{ , грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ мес}, \tag{3.14}$$

де B_k – число виконавців,

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні Fp = 176 годин).

$$T = \frac{1116,86}{1 \cdot 176} \approx 6,34 \text{ міс.}$$

Висновки: час розробки даного програмного забезпечення складає 1116,86 людино-годин. Таким чином, очікувана тривалість розробки складе 6,34 місяця при 40 годинному робочому тижні (місячний фонд робочого часу 176 годин), а витрати на створення програмного забезпечення складатимуть 125519 грн.

ВИСНОВКИ

Метою даної роботи є створення веб-сайту, який дасть змогу гравцям (або навіть кібер-скаутам) шукати товаришів по команді з усього світу.

Додаток може бути використаний звичайними гравцями у різні комп'ютерні ігри та скаутами кіберспортивних команд для пошуку гравців за критеріями, які їм необхідні. Також додаток може сприяти створенню спільноти навколо окремої гри шляхом формування рейтингу гарних гравців, що сприятиме поліпшенню психологічного клімату у тій чи іншій грі серед спільноти.

Продукт працює в усіх актуальних браузерах на усіх популярних платформах (Windows, MacOS, Linux) та на усіх актуальних пристроях, тобто є адаптивним під мобільні пристрої. Продукт дає можливість користувачу зареєструватися та належним чином захищає інформацію, надану користувачем.

Для досягнення поставленої мети вирішено наступні питання:

- аналіз предметної області;
- уточнення вимог до параметрів веб-сайту;
- уточнення вимог до продуктивності веб-сайту;
- розробка архітектури компонентів веб-сайту.

Відповідно до проведеного аналізу та завдання в роботі поставлені та вирішені такі функціональні задачі та вимоги до веб-додатку, а саме:

- зручний і інформативний інтерфейс;
- користувач має мати змогу створити аккаунт та використовувати його;
- можливість здійснення пошуку серед інших користувачів сайту, які знаходяться офлайн;
- окремий пошук для користувачів, які знаходяться онлайн;
- можливість опису свого профілю у різних іграх;

– можливість додавати, видаляти ігри зі свого пошуку та профілю;
– наявність адаптивності для використання сайту на мобільних пристроях;
– наявність можливості керування сайтом для людей з обмеженими можливостями.

Визначено трудомісткість розробленої інформаційної системи (1116,86 людино-годин), проведений підрахунок вартості роботи по створенню програми (125519 грн) та розраховано час на його створення (6,34 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Расчет стоимости машино-часа ЭВМ. studbooks.net. URL: https://studbooks.net/1786806/geografiya/raschet_stoimosti_mashino_chasa.

2. Зарплаты программистов в Украине. ДОУ. URL: https://jobs.dou.ua/salaries/#period=dec2020&city=Dnipro&title=Junior%20Software%20Engineer&language=JavaScript&spec=&exp1=0&exp2=10.

3. Flanagan D. JavaScript: the definitive guide. O'Reilly Media, 2020. 706 p.

4. Haverbeke M. Eloquent JavaScript: A Modern Introduction to Programming. 3rd ed. San Francisco, California, United States of America : No Starch Press, 2018. 472 p.

5. Head First JavaScript Programming: A Brain Friendly Guide. O'Reily, 2014.

6. Crockford D. JavaScript: The Good Parts: Working with the Shallow Grain of JavaScript. O'Reilly Media, Inc., 2008. 250 p.

7. Banks A., Porcello E. Learning react: modern patterns for developing react apps. O'Reilly Media, 2020. 310 p.

8. DuRocher D. HTML and CSS quickstart guide: the simplified beginners guide to developing a strong coding foundation, building responsive websites, and mastering ... of modern web design. ClydeBank Media LLC, 2021. 361 p.

9. Frain B. Responsive Web Design with HTML5 and CSS: Develop future-proof responsive websites using the latest HTML5 and CSS techniques, 3rd Edition. Packt Publishing, 2020. 408 p.

10. Freeman A. Pro react 16. Apress, 2019. 745 p.

11. Grant K. J. CSS in depth. Manning Publications, 2018. 472 p.

12. Meloni J. C., Kyrnin J. HTML, CSS, and javascript all in one, sams teach yourself (3rd edition). Sams Publishing, 2018. 800 p.

13. Meyer E. A., Weyl E. CSS: the definitive guide: visual presentation for the web. O'Reilly Media, 2017. 1090 p.

14. Roldan C. S. React Design Patterns and Best Practices: Design, build and deploy production-ready web applications using standard industry practices, 2nd Edition. Packt Publishing, 2019. 350 p.

15. Verou L. CSS secrets: better solutions to everyday web design problems. 2015. 354 p.

16. Wieruch R. The road to react: your journey to master plain yet pragmatic react.js. Independently published, 2018. 226 p.

17. Bugl D. Learning Redux: write maintainable, consistent, and easy-to-test web applications. Packt Publishing, 2017. 374 p.

18. Clean architecture: a craftsman's guide to software structure and design. Pearson, 2017. 432 p.

19. Freeman S. Growing object-oriented software, guided by tests. Upper Saddle River, NJ : Addison Wesley, 2010. 358 p.

20. Garreau M., Faurot W. Redux in action. Manning Publications, 2018. 312 p.

21. Introduction to algorithms. 3rd ed. Cambridge, MA : The MIT Press, 2009.

22. Mardan A. React quickly: painless web apps with react, JSX, redux, and graphql. Manning Publications, 2017. 528 p.

23. McLaughlin B. Head first object-oriented analysis and design. Beijing : O'Reilly, 2007. 600 p.

24. Pattern-oriented software architecture. Chichester [England] : Wiley, 2000.

25. Skiena S. S. The algorithm design manual. Springer, 2020. 810 p.

ДОДАТОК А

КОД ПРОГРАМИ

App.jsx

```
import React, { useEffect } from 'react';
import MainContainer from '../MainContainer/MainContainer';
import { Route, Switch, useLocation, Redirect } from 'react-router-dom';
import { TransitionGroup, CSSTransition } from 'react-transition-group';
import { connect } from 'react-redux';
import Cookies from 'js-cookie';

import Header from '../Header';
import Footer from '../Footer';
import Login from '../Login';

import loginBG from '../../assets/images/loginBG.jpg';
import patternBG from '../../assets/images/pattern.png';
import pattern1BG from '../../assets/images/pattern1.png';
import homeBG from '../../assets/images/homeBg.jpg';
import registerBG from '../../assets/images/registerBG.jpg';
import Registration from '../Registration';
import loadData from './connectStore';
import PropTypes from 'prop-types';
import SearchTable from '../SearchTable';
import SearchChoose from '../SearchChoose';
import Search from '../Search';
import Profile from '../Profile/Profile';
import UserLikes from '../UserLikes';
import API from '../../api/apiClient';
import OnlineSearch from '../Search/OnlineSearch';
import HomePage from '../HomePage';

const AuthorizedRoute = ({ path, children }) => {
  return (
    <Route path={path}>
      {Cookies.get('accessToken') ? children : <Redirect to="/login" />}
```

```
        </Route>
    );
};

const UnAuthorizedRoute = ({ path, children }) => {
    return (
        <Route path={path}>
            {Cookies.get('accessToken') ? <Redirect to="/home" /> : children}
        </Route>
    );
};

UnAuthorizedRoute.propTypes = {
    path: PropTypes.string,
    children: PropTypes.node,
};

AuthorizedRoute.propTypes = {
    path: PropTypes.string,
    children: PropTypes.node,
};

function App(props) {
    const location = useLocation();

    const startTimer = () => {
        if (Cookies.get('accessToken')) {
            API.keepOnline();
            setInterval(() => {
                API.keepOnline();
            }, 1000 * 60 * 4);
        }
    };

    useEffect(() => {
        props.loadData();
        startTimer();
    }, []);
```

```
return (
  <div className="appWrapper">
    <Header />
    <TransitionGroup>
      <CSSTransition
        key={location.key}
        classNames="fade"
        timeout={300}
      >
        <Switch location={location}>
          <Route path="/home" exact>
            <MainContainer bg={homeBG}>
              <HomePage />
            </MainContainer>
          </Route>

          <AuthorizedRoute path="/search" exact>
            <MainContainer isCentered={false} bg={pattern1BG}>
              <SearchTable />
            </MainContainer>
          </AuthorizedRoute>

          <AuthorizedRoute path="/search/:id" exact>
            <MainContainer bg={pattern1BG}>
              <SearchChoose />
            </MainContainer>
          </AuthorizedRoute>

          <AuthorizedRoute path="/me" exact>
            <MainContainer bg={pattern1BG}>
              <Profile />
            </MainContainer>
          </AuthorizedRoute>

          <AuthorizedRoute path="/search/:id/common" exact>
            <MainContainer bg={patternBG}>
              <Search />
```

```
                    </MainContainer>
                </AuthorizedRoute>

                <AuthorizedRoute path="/search/:id/online" exact>
                    <MainContainer bg={patternBG}>
                        <OnlineSearch />
                    </MainContainer>
                </AuthorizedRoute>

                <AuthorizedRoute path="/likes" exact>
                    <MainContainer bg={pattern1BG}>
                        <UserLikes />
                    </MainContainer>
                </AuthorizedRoute>

                <UnAuthorizedRoute path="/login" exact>
                    <MainContainer bg={loginBG}>
                        <Login />
                    </MainContainer>
                </UnAuthorizedRoute>

                <UnAuthorizedRoute path="/registration" exact>
                    <MainContainer bg={registerBG}>
                        <Registration />
                    </MainContainer>
                </UnAuthorizedRoute>

                <Route path="/">
                    <Redirect to="/home" />
                </Route>
            </Switch>
        </CSSTransition>
    </TransitionGroup>
    <Footer />
  </div>
 );
}
```

```
const mapDispatchToProps = {
  loadData,
};

App.propTypes = {
  loadData: PropTypes.func,
};

export default connect(null, mapDispatchToProps)(App);
```

apiClients.js

```
import Cookies from 'js-cookie';

const API = {
  apiBase: 'https://uaky.herokuapp.com/api/v1/',
  isRefreshing: false,

  async createRequest(endpoint, method, headers, data, isFormData = false) {
    const options = this.createOptions(method, headers, data, isFormData);

    const request = fetch(this.apiBase + endpoint, options);

    return request.then((error) => {
      if (error?.status === 403) {
        if (!this.isRefreshing) {
          this.isRefreshing = true;
          return this.refreshToken().then((res) => {
            if (res?.status === 200) {
              return res.json().then((tokens) => {
                Cookies.set('accessToken', tokens.accessToken);
                Cookies.set(
                  'refreshToken',
                  tokens.refreshToken
                );
                this.isRefreshing = false;
                const newHeaders = headers;
                newHeaders.Authorization = `Bearer ${tokens.accessToken}`;
                const newOptions = this.createOptions(
```

```
                        method,
                        newHeaders,
                        data,
                        isFormData
                    );
                    return fetch(
                        this.apiBase + endpoint,
                        newOptions
                    );
                });
            }
        });
    } else {
        return new Promise((resolve) => {
            const intervalId = setInterval(() => {
                if (!this.isRefreshing) {
                    clearInterval(intervalId);
                    const newHeaders = headers;
                    newHeaders.Authorization = `Bearer ${Cookies.get(
                        'accessToken'
                    )}`;
                    const newOptions = this.createOptions(
                        method,
                        newHeaders,
                        data,
                        isFormData
                    );
                    resolve(
                        fetch(this.apiBase + endpoint, newOptions)
                    );
                }
            }, 100);
        });
    }
}
return request;
});
},
```

```
async refreshToken() {
    return await this.createRequest('auth/refresh', 'POST', {
        Authorization: `Bearer ${Cookies.get('refreshToken')}`,
    });
},

createOptions(method, headers, data, isFormData) {
    const options = {};
    if (headers) options.headers = headers;
    if (data && !isFormData) options.body = JSON.stringify(data);
    if (data && isFormData) options.body = data;
    options.method = method;
    return options;
},

async getGames() {
    return await this.createRequest('game', 'GET');
},

async commonSearchUsers(id) {
    return await this.createRequest(`common-search/${id}`, 'GET', {
        Authorization: `Bearer ${Cookies.get('accessToken')}`,
    });
},

async onlineSearchUsers(id) {
    return await this.createRequest(`online-search/${id}`, 'GET', {
        Authorization: `Bearer ${Cookies.get('accessToken')}`,
    });
},

async refreshUsers(id) {
    return await this.createRequest(`common-search/${id}/views`, 'DELETE', {
        Authorization: `Bearer ${Cookies.get('accessToken')}`,
    });
},
```

```
async onlineRefreshUsers(id) {
    return await this.createRequest(`online-search/${id}/views`, 'DELETE', {
        Authorization: `Bearer ${Cookies.get('accessToken')}`,
    });
},

async registration(data) {
    return await this.createRequest(
        'register',
        'POST',
        { Accept: '*/*', 'Content-Type': 'application/json' },
        data
    );
},

async login(data) {
    return await this.createRequest(
        'auth/login',
        'POST',
        { Accept: '*/*', 'Content-Type': 'application/json' },
        data
    );
},

async getUserInfo() {
    return await this.createRequest('me', 'GET', {
        Authorization: `Bearer ${Cookies.get('accessToken')}`,
    });
},

async updateUserGame(game_id, data) {
    return await this.createRequest(
        `me/games/${game_id}`,
        'PUT',
        {
            Authorization: `Bearer ${Cookies.get('accessToken')}`,
            Accept: '*/*',
            'Content-Type': 'application/json',
```

```
        },
        data
    );
},

async deleteUserGame(game_id) {
    return await this.createRequest(`me/games/${game_id}`, 'DELETE', {
        Authorization: `Bearer ${Cookies.get('accessToken')}`,
    });
},

async addUserGame(data) {
    return await this.createRequest(
        'me/games',
        'POST',
        {
            Authorization: `Bearer ${Cookies.get('accessToken')}`,
            Accept: '*/*',
            'Content-Type': 'application/json',
        },
        data
    );
},

async editNickname(data) {
    return await this.createRequest(
        'me/nickname',
        'PATCH',
        {
            Authorization: `Bearer ${Cookies.get('accessToken')}`,
            Accept: '*/*',
            'Content-Type': 'application/json',
        },
        data
    );
},

async editPassword(data) {
```

```
    return await this.createRequest(
        'me/password',
        'PATCH',
        {
            Authorization: `Bearer ${Cookies.get('accessToken')}`,
            Accept: '*/*',
            'Content-Type': 'application/json',
        },
        data
    );
},

async editUserPicture(data) {
    return await this.createRequest(
        'me/image',
        'PATCH',
        {
            Authorization: `Bearer ${Cookies.get('accessToken')}`,
            Accept: '*/*',
            'Content-Type': 'application/json',
        },
        data
    );
},

async uploadPicture(payload) {
    return await this.createRequest(
        'images/users',
        'POST',
        {
            Authorization: `Bearer ${Cookies.get('accessToken')}`,
            Accept: '*/*',
        },
        payload,
        true
    );
},
```

```
    async getUserLikes() {
        return await this.createRequest('me/likes', 'GET', {
            Authorization: `Bearer ${Cookies.get('accessToken')}`,
        });
    },

    async likeUser(userId, gameId) {
        return await this.createRequest(
            `users/${userId}/likes/${gameId}`,
            'POST',
            {
                Authorization: `Bearer ${Cookies.get('accessToken')}`,
                Accept: '*/*',
                'Content-Type': 'application/json',
            }
        );
    },

    async keepOnline() {
        return await this.createRequest('me/last-visited-date', 'PATCH', {
            Authorization: `Bearer ${Cookies.get('accessToken')}`,
            Accept: '*/*',
            'Content-Type': 'application/json',
        });
    },
};

export default API;
```

Весь інший код можна знайти в файлах проекту.

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу бакалавра

на тему:

«Розробка веб-додатку для пошуку товаришів по команді на базі фреймворка React»

студента групи 122-17-2 Шеліпова Кирила Ігоровича

Керівник економічного розділу

доцент каф. ПЕП та ПУ, к.е.н **Л. В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
Появнювальні документи	
Shelipov_Diploma.docx	Пояснювальна записка кваліфікаційної роботи. Документ Word.
Shelipov_Diploma.pdf	Пояснювальна записка кваліфікаційної роботи в форматі PDF.
Програма	
Shelipov_Diploma.zip	Архів. Містить коди програми.
Презентація	
Shelipov_Diploma.pptx	Презентація кваліфікаційної роботи.