Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ВЕБДОДАТКУ ДЛЯ ВІДОБРАЖЕННЯ РЕЙТИНГУ ГРАВЦІВ ГРИ «STARCRAFT: REMASTERED» З ВИКОРИСТАННЯМ REACT ТА POSTGRESQL

DEVELOPMENT OF A WEB APPLICATION FOR DISPLAYING PLAYER RANKINGS OF THE GAME «STARCRAFT: REMASTERED» USING REACT AND POSTGRESQL

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-42
Мельник Максим Вікторович

Керівник:
Самчук Людмила Михайлівна

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«*20*» *12* 20*24* р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Мельнику Максиму Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка вебдодатку для відображення рейтингу гравців гри «Starcraft: Remastered» з використанням React та PostgreSQL»*

Керівник роботи: *к.т.н., доцент Самчук Л. М.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *React, ASP.NET Core, Vite, HTML, CSS, Javascript, C#, Visual Studio 2022, Visual Studio Code, методичні вказівки до виконання кваліфікаційної роботи бакалавра.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз ігрової індустрії, визначення завдань на кваліфікаційну роботу, аналіз та формування вимог до програмного забезпечення, вибір засобів, методів і алгоритмів для розробки, розробка вебдодатку, розробка Web API, отримання даних із API гри, створення бази даних, тестування та налагодження вебдодатку.*

5. Перелік графічного матеріалу: *5 додатків, 7 таблиць, 25 рисунків.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Самчук Л. М*		
Специфікація вимог до розробленої системи	*Самчук Л. М*		
Розробка об'єкта проектування	*Самчук Л. М*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	3,04 %		
Академічна доброчесність	*Самчук Л. М*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	Виконано
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	Виконано
3	Обгрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	Виконано
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	Виконано
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	Виконано
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	Виконано
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	Виконано

Здобувач вищої освіти _____ Максим МЕЛЬНИК

Керівник кваліфікаційної роботи _____ Людмила САМЧУК

АНОТАЦІЯ

Мельник М. В. Розробка вебдодатку для відображення рейтингу гравців гри «Starcraft: Remastered» з використанням React та PostgreSQL. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі здійснено аналіз предметної області і сформульовано завдання. В другому розділі було проаналізовано аналоги, визначено інструменти розробки, а також визначено основні вимоги до програмного забезпечення. У третьому розділі було описано розробку вебдодатку. У висновках було підведено підсумки виконання кваліфікаційної роботи.

Ключові слова: Front-end, Back-end, API, Starcraft:Remastered, ASP.NET Core, C#, UML, React, PostgreSQL.

ABSTRACT

Melnyk M. Development of a Web Application for Displaying Player Rankings of the Game "Starcraft: Remastered" Using React and PostgreSQL. Manuscript. Qualification work of the bachelor's program "Software engineering" in the specialty "Software engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, a conclusion, a list of references, and appendices.

The first chapter presents an analysis of the subject area and formulates the objectives. The second chapter describes the analysis of analogues and defines the requirements for web application and development tools. The third chapter describes the development of the system. The conclusions summarize the results of the qualification work.

Keywords: Front-end, Back-end, API, Starcraft: Remastered, ASP.NET Core, UML, C#, React, PostgreSQL.

ЗМІСТ

ВСТУП

Актуальність теми. На даний момент галузь відеоігор активно розвивається. Із розвитком Інтернету, швидкості роботи комп'ютерів, а також із появою та активний розвиток стрімінгових платформ аудиторія відеоігор стрімко розширюється, що створює попит не лише для забезпечення ігрового процесу, а й для інструментів аналізу ігор. Невід'ємною частиною галузі відеоігор є стратегії у реальному часі, зокрема «Starcraft:Remastered», яка навіть на сьогоднішній день має велику аудиторію.

Метою кваліфікаційної роботи бакалавра є розробка вебдодатку з клієнт-серверною архітектурою, який дозволяє переглядати таблицю лідерів у грі «Starcraft:Remastered», переглядати найкращого гравця кожної країни у грі, а також відображати гравців лише певної країни, раси, за яку гравець найбільше грає, а також ліги.

Об'єктом кваліфікаційної роботи бакалавра є процес перегляду, фільтрації, та аналізу статистичних даних гравців у грі «Starcraft:Remastered».

Предметом кваліфікаційної роботи бакалавра є методи та засоби розробки вебдодатку.

Для досягнення мети було визначено такі завдання:

- аналіз ігрової індустрії;
- визначення завдань на кваліфікаційну роботу;
- аналіз та формування вимог до програмного забезпечення;
- вибір засобів, методів і алгоритмів для розробки;
- розробка вебдодатку;
- тестування та налагодження вебдодатку;
- розробка Web API;
- отримання даних із API гри;
- створення бази даних.

Робота апробована шляхом участі у X міжнародній науково-практичній конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)» (додаток А).

РОЗДІЛ 1
АНАЛІЗ ІГРОВОЇ ІНДУСТРІЇ ТА ВИЗНАЧЕННЯ ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Поява стрімінгових платформ стало причиною того, що відеоігри стали найбільшою частиною в індустрії розваг. Контент по відеоіграм вже має більшу аудиторію, ніж Netflix, HBO, і Hulu разом узяті. Глядацька аудиторія кіберспортивних дисциплін, як серед випадкових, так і серед постійних глядачів, у 2019 році оцінювалася в 454 мільйони. Чемпіонат світу в 2018 році по грі League of Legends в Південній Кореї зібрав понад 99,6 мільйонів унікальних глядачів. Для порівняння, Superbowl LIII, фінальна гра в сезоні 2018 року в професійній лізі з американського футболу, загалом зібрав 98,2 мільйони глядачів [3].

Сучасний кіберспорт визначається як «форма спорту, де основні аспекти діяльності за допомогою електронних систем, введення гравців і команд, а також виведення результатів через інтерфейси взаємодії людини з комп'ютером» [14].

Ключовим аспектом будь якої кіберспортивної дисципліни є система підбору гравців (Matchmaking system) – алгоритм, який підбирає гравців за різними характеристиками, наприклад, регіон, рівень гри тощо.

В переважній більшості кіберспортивних ігор рейтинг гравця, тобто його рівень гри базується на Matchmaking Rating (MMR), який в свою чергу базується на системі Ело, що використовується Міжнародною федерацією шахів (ФІДЕ).

Дана рейтингова система працює наступним чином:

– якщо гравець перемагає слабшого суперника, він отримує менше очок рейтингу відносно тої кількості, яку має отримати гравець відповідно до норми за матч. Суперник при цьому також втратить меншу кількість очок;

– якщо гравець перемагає сильнішого суперника, він отримує більше очок рейтингу, відповідно і супернику знімає більше очок [3].

Розподіл Гаусса на прикладі платформи для гри в шахи онлайн «Chess.com» зображено на рисунку 1.1.

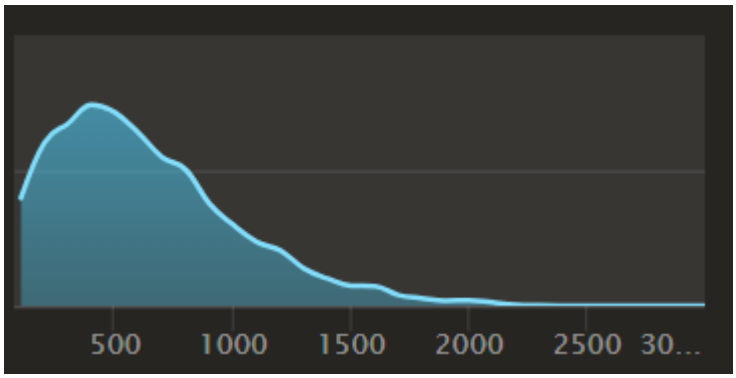

Рисунок 1.1 – Приклад нормального розподілу гравців [4]

На відміну від шахів, де система Ело створює нормальний (Гауссів) розподіл гравців, в більшості ігор гравці розподіляються нерівномірно. Так, у League of Legends на північноамериканському сервері у 2013 році понад 88 % гравців перебували у двох найнижчих рангах. Саме тому системи рейтингу в деяких іграх були модифіковані, для більш рівномірного розподілу гравців. Riot Games, розробники League of Legends, замінили чистий MMR на систему League Points (LP), яку часто оновлюють для точнішого відображення даних [3].

Для покращення навичок гравця необхідно виконувати постійно такі дії:

– створити оптимальні умови для гри, тобто, налаштувати нормальне Інтернет-з’єднання;

– забезпечити мінімальну затримку між комп’ютером та сервером;

– аналізувати записи гри професійних гравців;

– тренувати механіку гри;

– аналізувати свої записи гри.

Затримка між ком’ютером та сервером – це час, за який інформація про дії з комп’ютеру гравця доходить до сервера та відображається. Нормальною затримкою вважається 40 мілісекунд та нижче.

Затримка між комп’ютером гравця та серверу (більше відомий як пінг) негативно відзначається на якості ігрового досвіду та продуктивності

комп’ютера. Зазвичай гравці намагаються зменшити пінг або використовують спеціальні технології для зменшення впливу [13].

Високий пінг зумовлений наступними чинниками:

– проблеми з комп’ютером користувача, що може бути зумовлено слабким проходженням сигналу від мережевої карти до файлу з грою;

– проблеми з Інтернет-з’єднанням користувача;

– несправність ігрового сервера;

– велика кількість користувачів, які підключені до ігрового сервера;

– велика відстань між комп’ютером користувача та сервером гри.

Для мінімалізації впливу цих проблем розробники гри створюють декілька серверів у різних регіонах, щоб у користувачів по всьому світу була мала відстань до серверу.

Отже, на даний момент галузь кіберспортивних ігор дуже сильно розвивається. Проаналізувавши всі вимоги та проблеми, що існують у даній галузі, наступним етапом роботи буде розвиток локальних спільнот, що дозволить ще більше розширити аудиторію даних ігор, а також об’єднувати гравців.

У даній кваліфікаційній роботі було обрано гру «Starcraft:Remastered» у якості об’єкта дослідження. Це оновлена версія гри «Starcraft» та його доповнення Brood War з покращеною графікою та підтримкою розширення екрану 16:9. Суть даної гри полягає у розвитку економіки, розбудівлі бази, створенні армії та знищення ворога. Основний вигляд гри продемонстровано на рисунку 1.2.

Рисунок 1.2 – Приклад основного вигляду гри [8]

У грі існує три раси:

– террани – це раса людей, що перемагає за рахунок грамотного позиціонування армії;

– зерги – це раса інсектоїдних інопланетян, що перемагає за рахунок дешевої, але массової армії;

– протосси – це раса інопланетян, що перемагає за рахунок дорогої та якісної армії.

Розподілення гравців по рангам зображено на рисунку 1.3.

RANKS	WEEKS 1 & 2	STARTING WEEK 3
S	2139 +	Top 1%
A	1926 - 2138	7%
B	1670 - 1925	21%
C	1519 - 1669	21%
D	1376 - 1518	21%
E	1176 - 1375	21%
F	0 - 1175	8%

Рисунок 1.3 – Розподілення гравців по лігам [9]

Кожні півроку стартує новий сезон, на якому змінюють карти, а також обнуляється рейтинг усіх гравців. Усі ранги, крім рангу «S», можливо отримати завдяки кваліфікаційним матчам. Пороги входу до рангів обчислюються відповідно до кількості гравців, які знаходяться у них, завдяки чому створюється нормальний розподіл гравців.

У даній грі існують такі сервери:

– європейський;

– азійський;

– корейський;

– східноамериканський;

– західноамериканський.

Однак, оскільки в рейтингових іграх з’єднання проходить безпосередньо між гравцями (за допомогою P2P Connection), а не на ігровому сервері, система може створити гру між гравцями із різних серверів. По цій же причині пінг у

обох гравців завжди буде однаковим, і якщо у одного з гравців будуть проблеми з Інтернет-з’єднанням, то це буде відображатись у обох.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Сформовано мету роботи, якою є розробка вебдодатку з таблицею лідерів у грі «Starcraft:Remastered», який дозволяє переглядати найкращих гравців у кожній країні, з можливістю завантажувати записи матчів гравця.

Проаналізувавши інформацію, описану вище, під час кваліфікаційної роботи повинні виконуватися такі завдання:

- аналіз ігрової індустрії;
- визначення завдань на кваліфікаційну роботу;
- аналіз та формування вимог до програмного забезпечення;
- вибір засобів, методів і алгоритмів для розробки;
- розробка вебдодатку;
- тестування та налагодження вебдодатку;
- розробка Web API;
- отримання даних із API гри;
- створення бази даних.

Проведено аналіз сучасного стану речей у обраному середовищі та актуальність вище згаданої розробки, охарактеризовано предметну область та її подальший розвиток.

На етапі аналізу предметної області було проаналізовано поточні проблеми та технології, що впливають на ігровий досвід гравця. Було визначено основні дані, які необхідні для створення таблиці рейтингів.

Вибір інструментів вже безпосередньо буде залежати від аналізу предметної області, тому що через це буде залежати метод реалізації структури кваліфікаційної роботи та обмеження.

Наступним етапом є формулювання вимог, що є основою для подальшої розробки. На даному етапі наводяться вимоги до програми, які зображуються у

вигляді текстового опису, UML-моделей у вигляді таблиць з класами, функціями, сценаріями тощо. Крім того, на даному етапі визначається структура даних, що відображається на різних вкладках, а також функціональні можливості, що повинен виконувати сайт (наприклад, пошук гравців за лігою, країною або расою).

Виконавши попередні завдання, велика увага буде приділятися розробці програмного забезпечення. На даному етапі розробляється серверна частина сайту (Back-end), у якій збираються дані гравців, базі даних гравців, користувацької частини сайту (Front-end), та зв’язках між ними.

Завершивши розробку вище описаного вебсайту, заключним етапом є тестування програмного забезпечення. Даний етап охоплює великий спектр перевірок: від перевірки відповідності системи вимогам, до оцінки продуктивності.

Висновки до розділу 1

Було проаналізовано аналіз сучасного стану речей у індустрії кіберспортивних іграх, визначено ключові дані, що впливають на ігровий досвід, та наведено напрям для подальшого розвитку.

На основі вище проведеного аналізу було сформовано мету кваліфікаційної роботи, яка заключається у розробці вебдодатку з таблицею лідерів у грі «Starcraft:Remastered», з можливістю завантажити запис гри, а також сортування гравців за країнами. Даний вебдодаток дозволить новачкам краще аналізувати записи матчів професійних матчів, об’єднувати національні спільноти по даній грі та розширювати їх. Крім того, це зручний інструмент для організаторів місцевих та міжнаціональних турнірів, який дозволяє безпосередньо запрошувати гравців. Окреслено конкретні завдання для реалізації рішення, а саме вибір інструментів для розробки, створення вимог до програмного забезпечення, розробка програмного забезпечення та його тестування.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Проаналізувавши сучасний стан речей в галузі кіберспортивних дисциплін, а також усю інформацію про гру StarCraft:Remastered, було вирішено створити вебсайт з неофіційним рейтингом гравців у вище згаданій грі. В основі лежить збір даних з локального API, обробка даних, збереження їх в базі даних, передача даних у мережу, та їх коректне відображення на вебдодатку.

На даний момент існують такі аналоги:

– «RepMastered» – це платформа для аналізу ігор гравців, та загальної статистики. Має дуже великий функціонал, такий, як перегляд детальної інформації про гравців (статистики по картам, статистика ігор проти інших рас, аккаунти гравця), завантаження записів матчу, та таблицю лідерів на конкретних картах. Однак, дизайн є застарілим. Головне меню сайту зображено на рисунку 2.1;

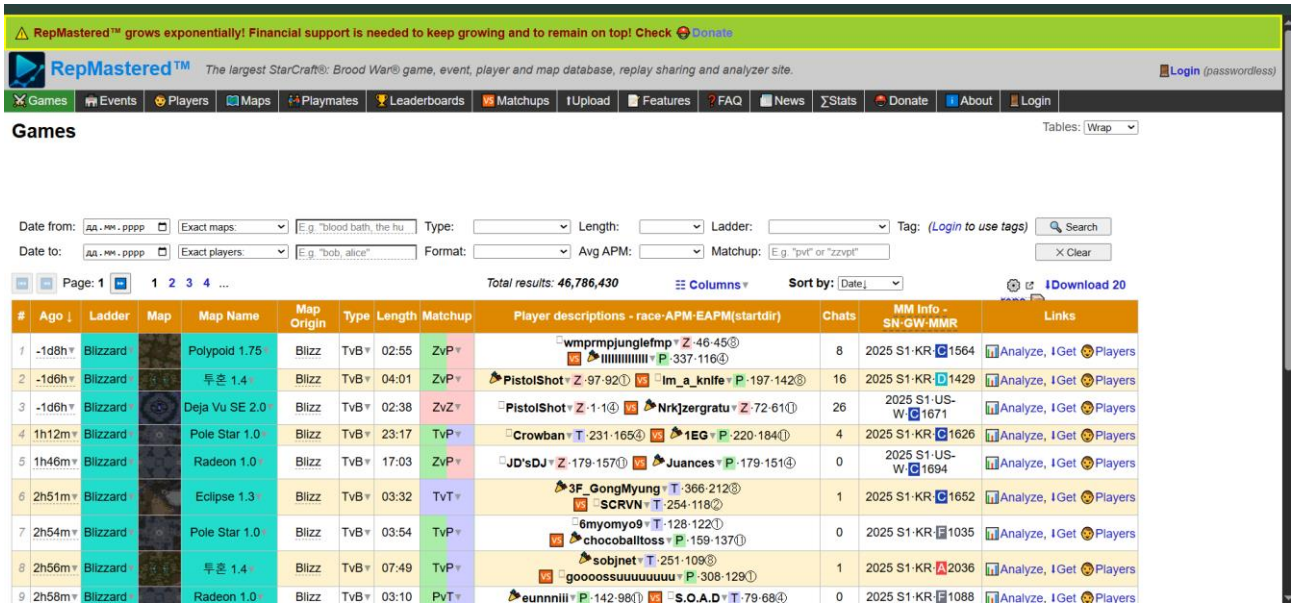

Рисунок 2.1 – Головне меню сайту Repmastered [6]

– «Cwal.gg» – це сайт, який відображає таблицю лідерів по MMR, а також дозволяє переглядати історію матчів гравця та завантажити його ігри. Головна сторінка сайту зображена на рисунку 2.2.

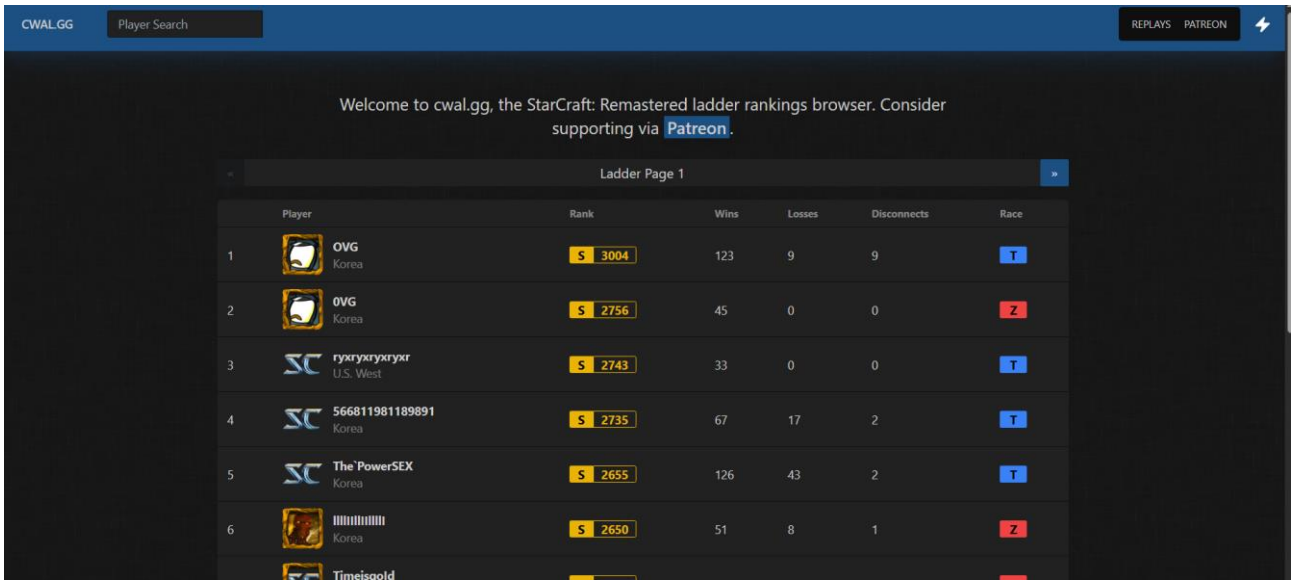

Рисунок 2.2 – Головна сторінка сайту cwal.gg [5]

Вебдодаток, який буде розроблено під час виконання кваліфікаційної роботи, повинен надавати можливість відслідковувати рейтинг гравців у реальному часі, та надавати можливість відображати лише гравців з певної країни, ліги, а також гравців лише за певну расу, а також надавати можливість завантажувати записи гри. Особливістю вебдодатку, який буде розроблено у порівнянні з іншими є можливість переглядати найкращого гравця кожної країни. Це дозволить організаторам національних турнірів легше знаходити гравців, а також об’єднувати локальні спільноти. Крім того, сайт повинен надавати можливість відображати лише корейських та некорейських гравців. Це дозволить організаторам турнірів для гравців не із Південної Кореї легше знаходити собі гравців.

Для зображення структури даних використовується UML-діаграма, а саме діаграма класів. Це зумовлено тим, що діаграма класів чітко показує структуру класів в програмі, а також їх зв’язки та атрибути.

Діаграма класів для даної кваліфікаційної роботи зображена на рисунку 2.3. На ній показано, що у базі даних зберігається країна, ранг, історія матчів кожного гравця, а також історія повідомлень, яка надсилалась під час матчу. Запис матчу зберігається за допомогою посилання, за яким можна його завантажити.

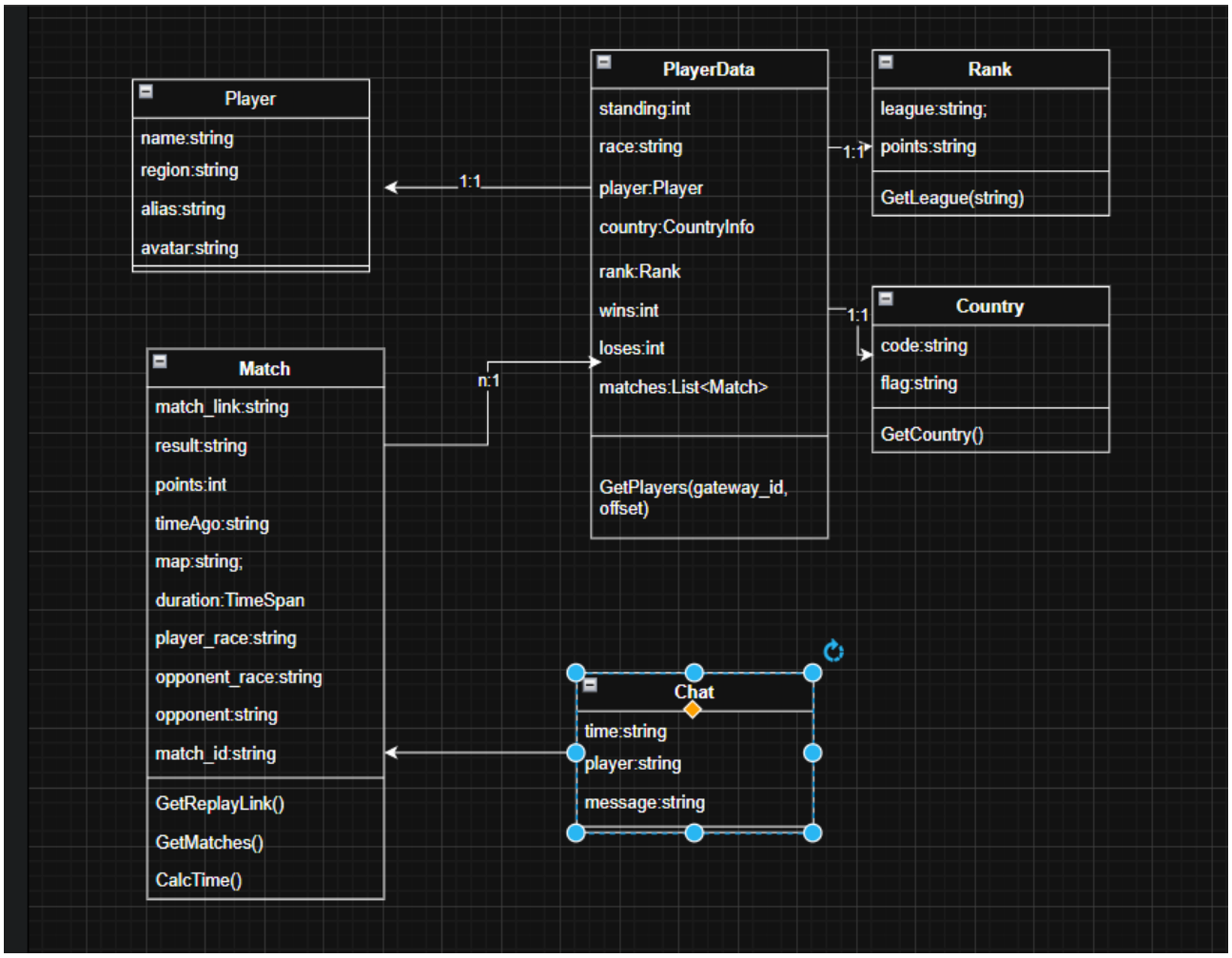

Рисунок 2.3 – Діаграма класів для кваліфікаційної роботи

Опис класів, які відображають структуру збереження даних у вебдодатку наведено в таблиці 2.1.

Таблиця 2.1 – Опис класів

Клас	Опис
PlayerData	Основний клас, що зберігає усі дані про гравця
Player	Клас, що зберігає основні дані про гравця

Продовження таблиці 2.1

Клас	Опис
Country	Клас, що зберігає дані про країну гравця
Rank	Клас, що зберігає дані ранг гравця
Matches	Клас, що відображає основні дані про матчі
Chat	Клас, що зберігає основні дані про чат

Опис екземплярів класу «PlayerData», який зберігає усі дані про гравця, наведено в таблиці 2.2.

Таблиця 2.2 – Опис екземплярів класу «PlayerData»

Дані	Опис
Standing	Місце гравця у світовому рейтингу
Race	Раса, за яку гравець найбільше зіграв матчів
Player	Екземпляр класу Player,який описує основні дані про гравця
Placement	Місце гравця у рейтингу серверу
Country	Екземпляр класу Country, який описує дані про країну гравця
rank	Ранг, в якому знаходиться гравець
wins	Кількість перемог
Loses	Кількість поразок
matches	Список екземпляру класу Matches, що відображає усі дані про матчі гравця
GetPlayers()	Функція для отримання даних про гравців з API розробників

Опис екземплярів класу «Player», який зберігає базові дані про гравця, наведено у таблиці 2.3.

Таблиця 2.3 – Опис екземплярів класу «Player»

Дані	Опис
name	Нікнейм гравця
Region	Сервер, на якому зареєстрований аккаунт
Alias	Battle-Tag гравця
avatar	Посилання на аватарку гравця

Опис екземплярів класу «Country», які отримують та зберігають усю необхідну інформацію для правильного відображення країни, наведено у таблиці 2.4.

Таблиця 2.4 – Опис екземплярів класу «Country»

Дані	Опис
code	Код країни
flag	Прапор країни гравця
GetCountry()	Отримує код країни, перетворює кодування країни з Alpha-3 в Alpha-2

Опис екземплярів класу «Rank», які відповідають за правильне відображення ліги гравця, наведено у таблиці 2.5.

Таблиця 2.5 – Опис екземплярів класу «Rank»

Дані	Опис
league	Ліга, в якій знаходиться гравець
points	Кількість очок рейтингу (MMR) гравця
GetLeague()	Обчислює лігу гравця

Опис екземплярів класу «Matches», який зберігає усю інформацію про історію матчів гравця наведено у таблиці 2.6.

Таблиця 2.6 – Екземпляри класу «Matches»

Дані	Опис
result	Результат матчу
points	Кількість очок рейтингу, що отримав гравець за матч
timeAgo	Час, що пройшов з моменту матчу
map	Карта, на якій був зіграний матч
duration	Тривалість матчу
Player_race	Раса гравця в даному матчі
Opponent_race	Раса оппонента в даному матчі
Opponent	Нікнейм оппонента
Match_id	Id, за яким в базі даних зберігається матч
CalcTime()	Переводить час з UNIX - формату в звичайний
GetMatches()	Отримує історію матчів з API розробників
DownloadMatch()	Отримує посилання з API розробників, за яким можна завантажити матч

Опис екземплярів класу «Chat», який зберігає усю інформацію про історію повідомлень у матчі, наведено у таблиці 2.7.

Таблиця 2.7 – Клас «Chat»

Дані	Опис
time	Час відправлення повідомлення в грі
player	Гравець, який відправив повідомлення
message	Основний текст повідомлення

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Для розробки вебдодатку, який відображає неофіційний рейтинг гравців, необхідно дістати дані із API розробників. У StarCraft: Remastered до нього

можна отримати доступ лише під час запущеної гри. Для цього необхідно запустити гру, відслідкувати порт, до якого під'єднаний комп'ютер до серверу, та відправити необхідний GET – запит на сервер, який буде включати порт.

Для відслідковування порту використовується Fiddler. Це програма, яка відслідковує усі HTTP запити із комп'ютера, та дозволяє детально переглянути кожен із них. Головне меню зображено на рисунку 2.4.

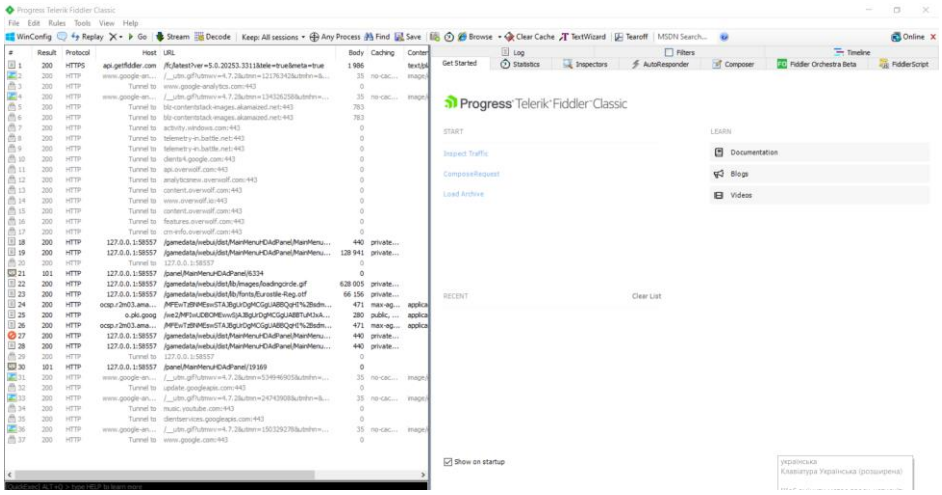

Рисунок 2.4 – Головне меню Fiddler

Вебдодаток буде побудований на клієнт-серверній архітектурі, що значить, що він буде включати в себе три частини – для взаємодії з користувачем (Front-end), взаємодії з сервером (Back-end) та базу даних. Для створення цих частин використовуються різні технології.

Для створення Front-end частини було проаналізовано такі технології: HTML, CSS, JavaScript, React, Vue, Angular.

Для створення Back-end частини було проаналізовано такі технології:Node.js, Laravel, ASP.NET Core.

HTML – це мова гіпертекстової розмітки. Дозволяє визначати структуру вебсайту, а також ієрархію елементів. Основним елементом даної розмітки є так звані теги, які визначають тип елементу. Крім того, у HTML можна кожному елементу присвоїти один або більше класів, що дозволяє зручно відслідковувати,

за що відповідає той чи інший елемент. Приклад HTML коду зображено на рисунку 2.5.

```
</div>
  </td>
  <td className="table-cell country-top-cell player-cell">
    <Link to={`/player-page/${encodeURIComponent(row.player.name)}`} className="player-link">
     <div className="player-container-flex">
     <div style={{ display: "flex", alignItems: "center" }}>
     <img
    src={row.player.avatar}
    alt="Avatar"
    onError={(e) => {
      e.target.onerror = null;
      e.target.src = 'https://p1.hiclipart.com/preview/716/196/996/blizzard-flat-iconset-starcraft-remastered-png-clipart.jpg
    }}
    width="50"
    height="50"
    />
        <div>
          <p className="table-text">{row.player.name}</p>
          <p className="table-text" >
            {row.player.region}
          </p>
        </div>
      </div>
      </div>
```

Рисунок 2.5 – Приклад HTML розмітки [10]

CSS – це мова стилів, яку використовують для зображення зовнішнього вигляду HTML – документів. Вона дозволяє змінювати шрифти, кольори, розміри, відстань між різними елементами, анімацію, а також змінювати розміри відповідно до розмірів екрану, що дозволяє створити адаптивний дизайн. Стилі в даній мові є каскадними, тобто стилі, які є притаманні батьківським елементам, будуть притаманні і для дочірніх елементів. Ця мова працює зі стилями через селектори, тобто, набір елементів, до яких саме буде застосований цей стиль. Селекторами можуть бути типи елементів (<p>), так і назви класів. Кожен стиль складається з властивості та її значення. Приклад CSS стилів зображено на рисунку 2.6.

JavaScript – це мова програмування, що використовується для написання логічної частини веб – додатку. Виконується безпосередньо в браузері. Синтаксис даної мови програмування включає в себе змінні, цикли, умовні оператори, функції, об’єкти та класи. Однією з важливих особливостей JavaScript є робота з структурою HTML – документів. Завдяки цьому JavaScript дозволяє динамічно додавати або видаляти елементи, змінювати текст, атрибути, стилі та інші

властивості сторінки. Мова постійно розвивається, і сучасний JavaScript включає багато нових можливостей, таких як асинхронне програмування, модулі, лямбда функції та класи.

```
.table-text
{
  font-family: 'TT Firs Neue Trl', sans-serif;
  color: white;
  text-align: start;
  padding-left: 10px;
}
.table-standing
{
color: gray;
padding:0;
text-align: center;
}
```

Рисунок 2.6 – Приклад CSS коду [11]

React – це бібліотека JavaScript , яка призначена для створення клієнтської частини вебдодатку. Вона дозволяє розробникам створювати динамічні вебдодатки, де вміст змінюється без перезавантаження сторінки. Основною складовою даної бібліотеки є компоненти. Це незалежна частина інтерфейсу, що містить свої елементи, логіку та стиль. Це дозволяє легко маштабувати додаток, а також використовувати вже існуючі компоненти, що зменшує кількість коду. React працює з файлами у форматі JSX, що дозволяє працювати з HTML розміткою у одному файлі з Javascript, що спрощує розуміння коду. Також React включає в себе стани, які дозволяють відслідковувати змінні значення та дозволяє підключати інші бібліотеки, що робить React гнучким.

Vue.js – це бібліотека для JavaScript, яка призначена для створення інтерфейсів користувача, а також односторінкових додатків. Його особливістю є простота та зрозумілість, саме тому він є найпопулярнішим вибором серед новачків. Як і React, Vue базується на компонентах. Однією з головних особливостей Vue є реактивність, коли змінюється стан даних, інтерфейс автоматично оновлюється. Розробнику не потрібно вручну змінювати DOM – Vue

самостійно стежить за змінами й застосовує їх. Має просту інтеграцію з існуючими проектами.

Angular – це повноцінний фреймворк, який використовується для масштабних вебдодатків. У його арсеналі з коробки є все необхідне: маршрутизація, робота з формами, здійснення HTTP-запитів, система модулів, механізми залежностей, інструменти для тестування та багато іншого. Це дає змогу розробляти великі корпоративні системи, не вдаючись до залучення сторонніх бібліотек. Центральним елементом Angular є компоненти, що описують окремі блоки інтерфейсу. Кожен компонент містить HTML-шаблон, логіку, написану на TypeScript, а також стилі. Компоненти об'єднуються в модулі, які слугують для організації коду та забезпечення масштабованості застосунку. Angular застосовує двостороннє прив'язування даних (two-way data binding), що реалізує автоматичну синхронізацію змін між моделлю та інтерфейсом. Зміна, внесена користувачем у формі, призводить до автоматичного оновлення моделі, і навпаки [1].

Отже, для малих і середніх проєктів варто використовувати Vue, для великих проектів – Angular, якщо необхідна гнучкість – React.

Проаналізувавши вище описану інформацію, для даної кваліфікаційної роботи було обрано React для написання Front-end частини, який включає в себе HTML розмітку та CSS стилі, оскільки він є гнучким та підтримує безліч бібліотек, що дозволяє легко змінити архітектуру проекту відповідно до бажань та потреб.

Так, у даній кваліфікаційній роботі до React буде встановлено MUI. Це бібліотека, яка включає в себе уже готові компоненти, такі як таблиці, діалогові вікна, випадаючі меню, тощо. Це дозволяє швидко та зручно додавати та налаштовувати за допомогою дизайну уже існуючі елементи. Зв'язок з Back-end частиною буде здійснюватися за допомогою Rest API. Це технологія, що дозволяє отримувати дані із серверів через HTTP-запити (GET, POST, PUT, DELETE тощо). Проект буде базуватись на Vite. Це інструмент для збирання проєктів та

створення фундаменту проєктів, який набирає популярність завдяки швидкій компіляції коду та гарячій заміні модулів (HRM).

Node.js – це сучасний інструмент для розробки серверної частини сайту, що базується на мові програмування JavaScript. Це середовище виконання, який надає інструменти для роботи з файловою системою, мережею, потоками тощо. Виконує все в одному потоці, що робить його незручним для систем, які вимагають багато роботи із процесором, але робить даний інструмент хорошим вибором для розробки систем із великою кількістю запитів. Має підтримку NPM (Node Package Manager), що надає доступ для великого набору бібліотек.

Laravel – це фреймворк для мови програмування PHP, який дотримується шаблону MVC (Model-View-Controller). Це значить, що структура проекту повинна складатися з таких елементів:

- модель (Model) – клас, що відображає, як саме повинні зберігатись дані;
- контроллер (Controller) – це функції, які відповідають за логічну частину;
- вигляд (View) – відповідають за основний вигляд.

Однією з головних переваг Laravel є велика кількість готових рішень, таких як: маршрутизація, ORM для роботи з базою даних (Eloquent), шаблонізатор Blade, система аутентифікації, робота з поштою, чергами, подіями, кешуванням, тестуванням тощо [12].

ASP.NET Core – це фреймворк від компанії Microsoft на базі мови програмування C#, що славиться своєю універсальністю. Є кросплатформенним, тобто, розробляти додатки на даному фреймворку можна на більшості сучасних операційних системах, таких як Windows, MacOS тощо. Містить велику кількість шаблонів, від Web API, до повноцінних вебдодатків на базі архітектури MVC. Також фреймворк підтримує Razor Pages, Blazor, які дозволяють створювати інтерфейси користувача, використовуючи C# замість JavaScript [2]. Для роботи з базами даних ASP.NET Core зазвичай використовує Entity Framework Core – об’єктно-реляційну модель (ORM), яка дозволяє працювати з БД через класи. Підтримуються SQL Server, PostgreSQL, MySQL, SQLite та інші СУБД. Має

високу продуктивність, що робить його хорошим вибором для сервісів, які створюють високе навантаження.

Проаналізувавши дані інструменти для створення Back-end частини, було прийнято рішення використовувати ASP.NET Core у даній кваліфікаційній роботі через його універсальність. Для зручної роботи з базами даних до ASP.NET Core було встановлено Entity Framework Core, який дозволяє автоматично створити базу даних залежно від існуючих моделей. Для керування даними було обрано PostgreSQL у якості системи керування базами даних, оскільки вона є повністю безкоштовною.

Оскільки для створення Back-end частини обрано ASP.NET Core, тому у якості IDE (інтегрованого середовища розробки) обираємо Visual Studio 2022. Він дозволяє створити новий проект уже з існуючого шаблону без терміналу, що дозволяє зручно почати роботу. Крім того, він має інтеграцію з Docker, що дозволяє зручно запускати проєкт у ізольованому середовищі. Visual Studio має вбудований менеджер пакетів NuGet, що дозволяє швидко встановити у проект бібліотеки та залежності відповідно до вимог проєкту. Вигляд головного інтерфейсу Visual Studio 2022 зображено на рисунку 2.7.

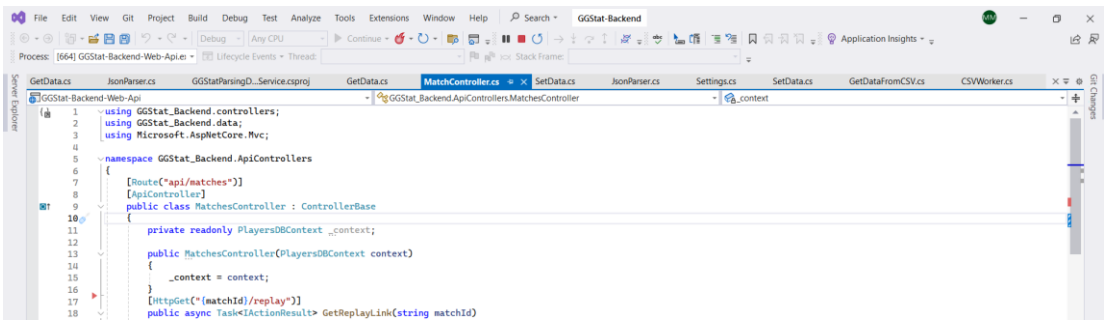

Рисунок 2.7 – Звичайний вигляд Visual Studio 2022

Однак, для розробки Front-end частини Visual Studio 2022 є незручним, оскільки він заточений на розробку Back-end частини, а також в ньому немає інтеграцій з сучасними інструментами розробки користувацької частини сайту. Саме тому для розробки Front-end частини було обрано Visual Studio Code у якості IDE. Це гнучкий редактор коду, який підтримує сучасні інструменти

розробки Front-end частини. Має величезний набір плагінів, що дозволяє використовувати його для різних потреб, а також змінити основний вигляд інтерфейсу відповідно до побажань користувача. Приклад головного меню Visual Studio Code зображено на рисунку 2.8.

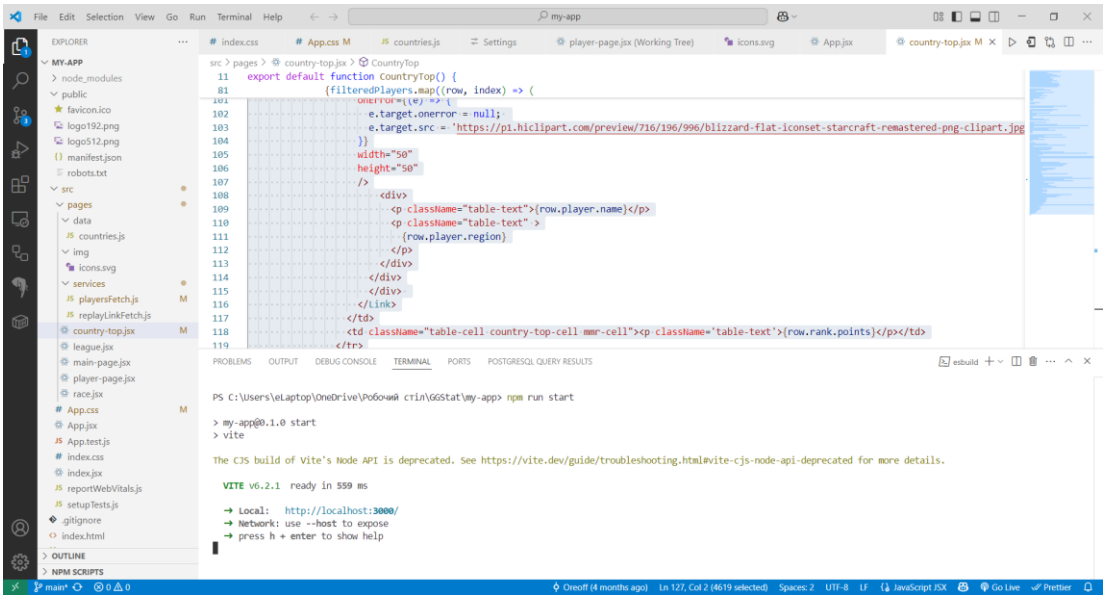

Рисунок 2.8 – Головне меню Visual Studio Code

Висновки до розділу 2

Проаналізувавши сучасний стан речей у грі StarCraft: Remastered, а також аналоги вебдодатку, який було розроблено під час виконання даної кваліфікаційної роботи, було вирішено, що даний вебдодаток повинен містити в собі таблицю найкращих гравців, яка оновлюється у реальному часі. Вебдодаток повинен надавати можливість відображати гравців лише певної країни, ліги, та раси, а також завантажувати записи матчів конкретного гравця.

Для моделювання системи було вирішено використовувати модель класів, оскільки вона найбільш точно відображає структуру даних, яка повинна бути у кваліфікаційній роботі бакалавра.

Після аналізу різних інструментів, було обрано React для Front-end частини вебдодатку. Це бібліотека для мови програмування JavaScript, яка дозволяє працювати з HTML у одному файлі з JavaScript, що робить розробку зручною.

Крім того, React підтримує безліч інших бібліотек. Серед них буде використовуватись MUI для використання вже готових компонентів, а також Rest API для отримання даних із Back-end частини. Для швидшого старту розробки у даній кваліфікаційній роботі буде використовуватись збірник Vite. У якості IDE було вирішено використовувати Visual Studio Code.

Для Back-end частини було обрано ASP.NET Core у якості інструмента розробки. До нього було вирішено Entity Framework Core, для роботи з базами даних. У якості системи керування базами даних було обрано PostgreSQL. У якості IDE було вирішено використовувати Visual Studio 2022.

РОЗДІЛ 3
РОЗРОБКА ВЕБДОДАТКУ З ТАБЛИЦЕЮ ЛІДЕРІВ В ГРІ «STARCRAFT:REMASTERED»

3.1 Практична реалізація об’єкта проєктування

Розробка вебдодатку почалася із розробки клієнтської частини. За допомогою команди npm create-react-app було ініціалізовано проєкт на базі React, що дозволило створити базову структуру проєкту. Після цього, проєкт було перенесено на Vite для підтримки Hot Reloading, а також оптимізації продуктивності.

Структура клієнтської частини зображена на рисунку 3.1.

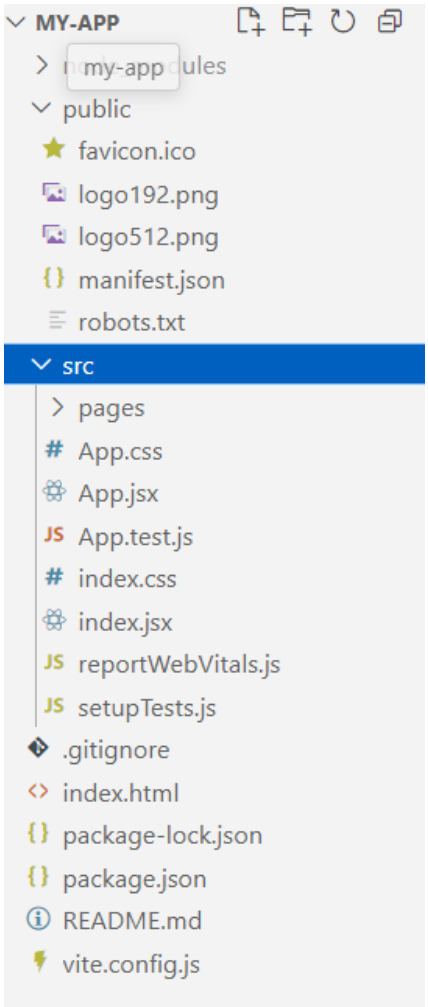

Рисунок 3.1 – Структура клієнтської частини вебдодатку

Усі стилі елементів інтерфейсу визначені в файлі App.css. У директорії Pages знаходяться усі основні компоненти для створення структури сторінок вебдодатку, а також SVG-спрайт, який зберігає усі іконки, які використовуються для візуалізації елементів вебдодатку. App.jsx є головним компонентом, в якому знаходиться верхня частина додатку (header), нижня частина додатку (footer), а також визначає логіку маршрутизації проєкту.

Загалом додаток включає в себе три сторінки: головна сторінка, сторінка гравця, а також сторінка «country-top», яка відображає найкращого гравця кожної країни. Для кожної сторінки було створено окремий компонент: «main-page», «country-top» та «player-page», які знаходяться у різних файлах відповідно до назви.

Для відображення раси та ліги гравця використовуються окремі компоненти, для того, щоб забезпечити гнучкість у стилізації елементів, зокрема, зміни кольору тексту відповідно до вмісту. Також це дозволяє зменшити кількість коду, оскільки ці самі елементи використовуються на головній сторінці та сторінці гравця. Компонент раси гравця відображено у лістингу 3.1.

Лістинг 3.1 – Код компоненту раси гравця

```
export default function Race ({ text}) {
    let className = '';
    if (text.includes("terran") || text.includes("T")||
text.includes("t")) {
      className = 'terran';
    } else if (text.includes("zerg")|| text.includes("Z") ||
text.includes("z")) {
      className = 'zerg';
    } else if (text.includes("protoss")|| text.includes("P") ||
text.includes("p")) {
      className = 'protoss';}
  return (
    <div className="race-container">
<p className={className}>{text}</p>
<p className="race-full-text">{className}</p>
    </div>
  );};
```

кінець лістингу 3.1

Компонент для відображення ліги гравця відображено на лістингу 3.2. Даний компонент повертає MMR гравця, та лігу гравця, які змінюють колір відповідно до ліги, в якій знаходиться гравець.

Лістинг 3.2 – Код компоненту для відображення ліги гравця

```
export default function League({ text = '', MMR = '' }) {
  let textColor = '#000';
  if (typeof text !== 'string') {
    console.warn('Expected "text" to be a string');
    text = String(text);
  }
  if (text.includes("F")) {
    textColor = '#9E9E9E';
  } else if (text.includes("E")) {
    textColor = '#10FE40';
  } else if (text.includes("D")) {
    textColor = '#0BFEE6';
  } else if (text.includes("C")) {
    textColor = '#0B1BFE';
  } else if (text.includes("B")) {
    textColor = '#EE0BFE';
  } else if (text.includes("A")) {
    textColor = '#FE0606';
  } else if (text.includes("S")) {
    textColor = '#DAA520';
  }
  return (
    <div className='league-container'>
    <p className="rating" style={{ backgroundColor: textColor }}>
      {text}
    </p>
    <p className="mmr" style={{ color: textColor }}>
    {MMR}
  </p>
    </div>
  );
}
League.propTypes = {
  text: PropTypes.string,
  MMR: PropTypes.string,
};
League.defaultProps = {
  text: '',
  MMR: '',
};
```

кінець лістингу 3.2

Для реалізації адаптивності вебдодатку використовуються так звані медіа-запити, які дозволяють змінювати стилі елементів відповідно до параметрів пристрою, зокрема ширини екрану. Приклад застосування медіа-правил відображено у лістингу 3.3.

Лістинг 3.3 – Приклад медіа-правил

```
@media (max-width: 1000px) {
  .race-full-text
  {
    display:none;
  }
}
```

кінець лістингу 3.3

Для реалізації пагінації у таблицях, а також для створення випадаючого меню для вибору гравців певної країни було використано бібліотеку MUI, яку було підключено до проєкту за допомогою команди «npm install @mui/material @emotion/react @emotion/style».

Основний вигляд головного меню зображено на рисунку 3.2.

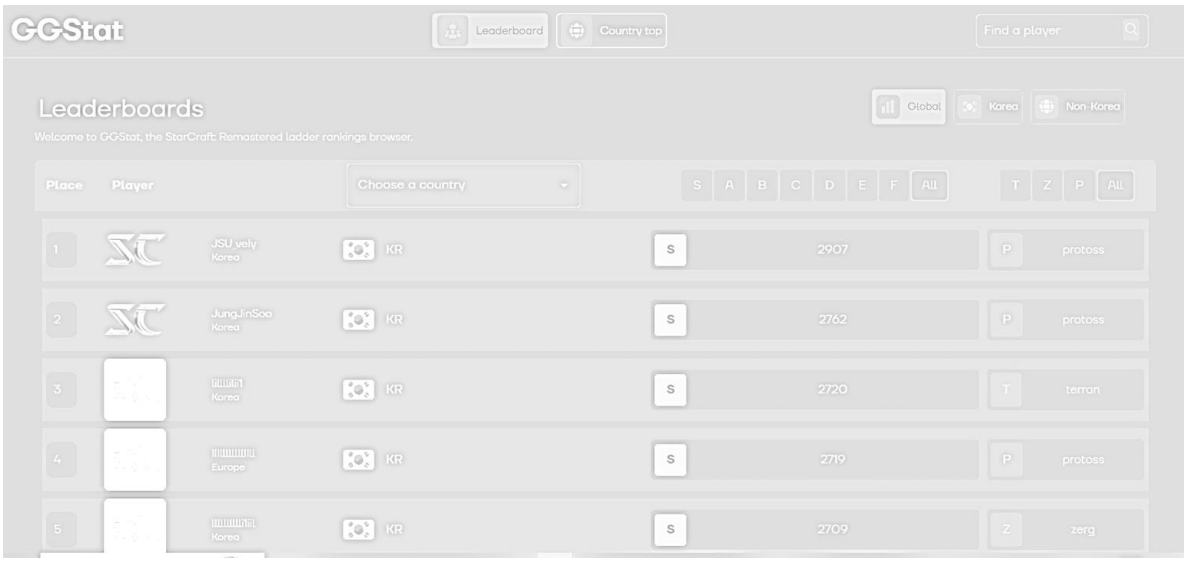

Рисунок 3.2 – Головна сторінка

У даній таблиці є можливість відобразити всіх гравців із Південної Кореї, а також усіх гравців, які не походять із цієї країни. Приклад відображення гравців не із Південної Кореї зображено у рисунку 3.3.

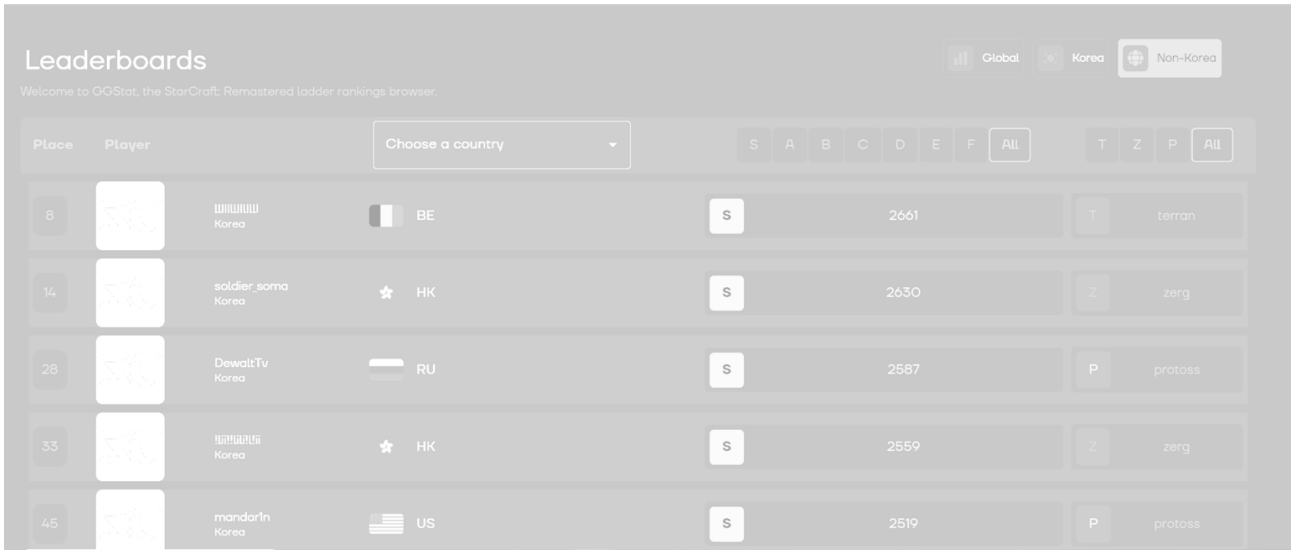

Рисунок 3.3 – Приклад відображення некорейських гравців

Крім того, таблиця дозволяє відобразити гравців лише певної країни, раси, або ліги. На рисунку 3.4 зображено приклад роботи селекторів, а саме відображення всіх гравців з України з С ліги, які грають за расу протосів.

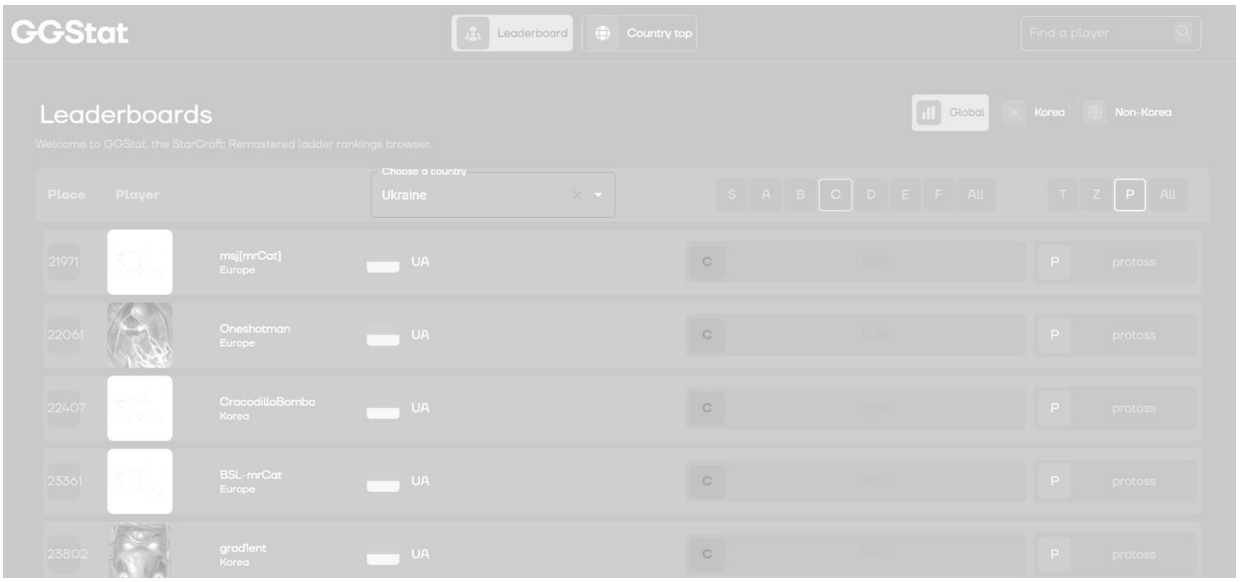

Рисунок 3.4 – Приклад роботи селекторів

Мобільна версія даної таблиці зображена на рисунку 3.5. На мобільній версії усі елементи вибору (селектори) згруповані у модальному меню, яке відкривається за допомогою кнопки «Filter», що дозволяє зручно користуватися вебдодатком на мобільних пристроях, при цьому зберігаючи весь функціонал.

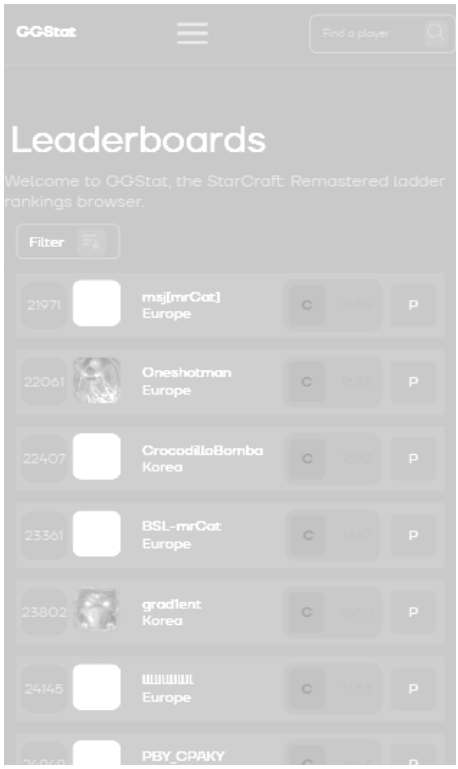

Рисунок 3.5 – Мобільна версія головної сторінки

Модальне меню в якому знаходяться усі фільтри у мобільній версії вебдодатку, зображено на рисунку 3.6.

Рисунок 3.6 – Модальне меню селекторів

На одній сторінці може відображатись не більше ніж 25 гравців. Для навігації між сторінками таблиці використовується пагінація, яку було створено із готового шаблону в MUI. Пагінація, а також нижня частина вебдодатку (footer) зображена на рисунку 3.7.

Рисунок 3.7 – Пагінація та нижня частина вебдодатку

Приклад роботи панелі для пошуку гравців відображено на рисунку 3.8. При натисканні на кнопку пошуку, а також при активації поля введення тексту відображається список гравців, який фільтрується за допомогою введення у це поле тексту. У списку відображаються лише ті гравці, псевдонім (нікнейм) яких містить частину тексту, яка була написана у полі введення. Кожен елемент списку є посиланням на сторінку цього гравця.

Рисунок 3.8 – Приклад роботи панелі пошуку

Основний вигляд сторінки для перегляду найкращих гравців кожної країни (country-top) відображена на рисунку 3.9.

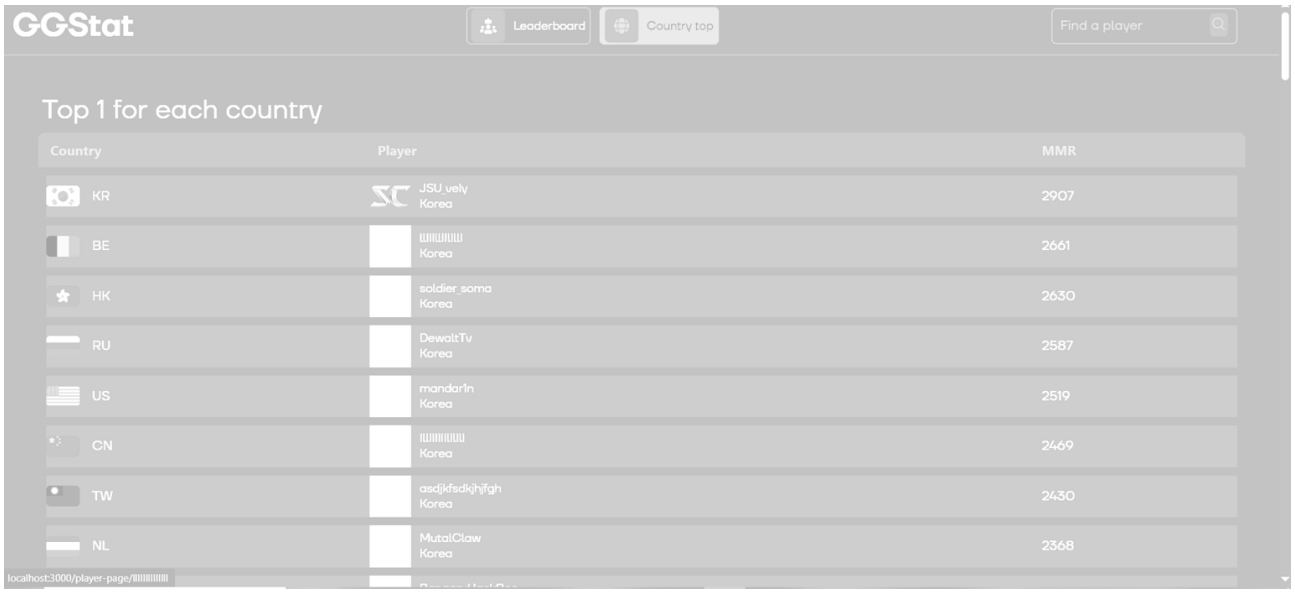

Рисунок 3.9 – Основний вигляд сторінки «country-top»

Версія даної сторінки для мобільних пристроїв не відрізняється від звичайної версії. Приклад сторінки, на якій зберігається уся інформація про гравця, а також його історію матчів, зображено на рисунку 3.10.

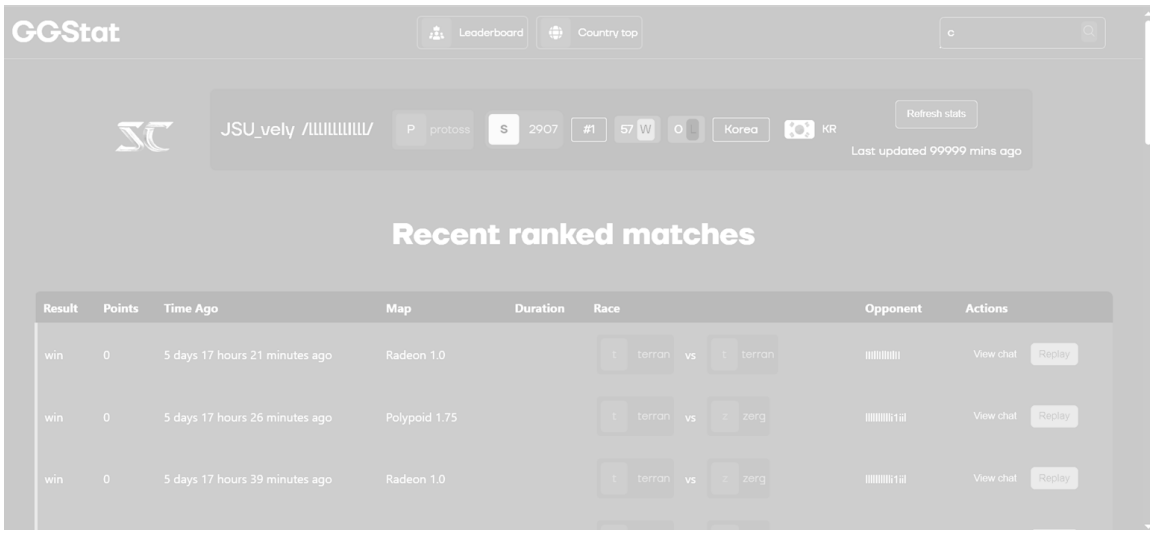

Рисунок 3.10 – Сторінка гравця

При натисканні кнопки «View Chat» відобразяться всі повідомлення, які надсилали гравці один одному під час гри. Приклад відображено на рисунку 3.11.

Рисунок 3.11 – Приклад відображення чату

Мобільна версія даної сторінки зображена на рисунку 3.12.

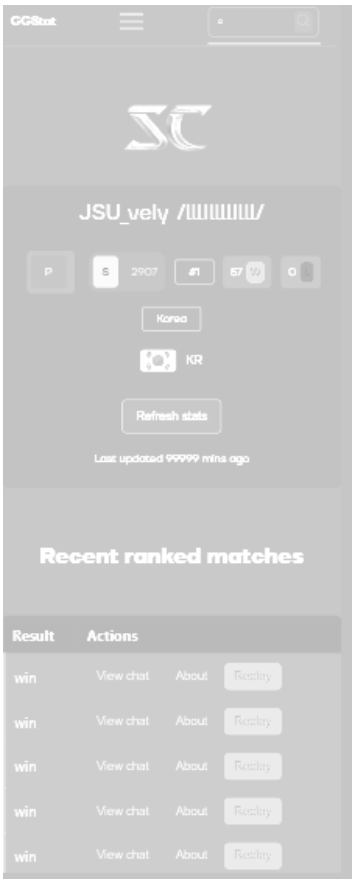

Рисунок 3.12 – Мобільна версія сторінки гравця

На мобільній версії усю інформація про матч є прихованою, та відображається у окремому полі, яке відображається при натисканні кнопки «About», що дозволяє відобразити усю інформацію у зручному для користувача

вигляді при обмеженому екрані. Приклад відображення інформації про матч зображено на рисунку 3.13.

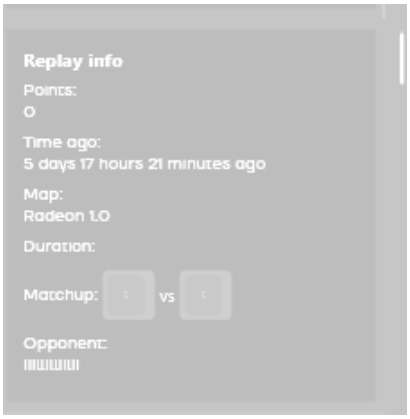

Рисунок 3.13 – Основна інформація про матч на мобільній версії

Для взаємодії із серверною частиною було обрано архітектурний шаблон Rest API. Отримання інформації відбувається за допомогою HTTP запитів до серверної частини. У відповідь сервер надсилає дані у форматі JSON. Код для отримання даних відображений на лістингу 3.4.

Лістинг 3.4 – Код для отримання даних

```
export default async function fetchPlayers() {
  try {

    const getResponse = await fetch('http://localhost:5000/api/players',
{
      method: 'GET',

    });
    if (!getResponse.ok) {
      throw new Error(`HTTP error! status: ${getResponse.status}`);
    }
    const data = await getResponse.json();
    console.log("Fetched players:", data);
    return data;
  } catch (error) {
    console.error('Error fetching data:', error);
  }
}
```

кінець лістингу 3.4

Після розробки клієнтської частини вебдодатку було розроблено серверну частину. На основі попереднього аналізу було прийнято рішення використовувати мікросервісну архітектуру для розробки серверної частини. Це передбачає розподіл системи на незалежні сервіси, кожен з яких виконує свою функцію. Це дозволить тестувати кожну частино окремо, а також забезпечити роботу проєкту у разі відмови функціоналу, наприклад, відмови роботи ігрового серверу, що робить отримання нових даних неможливим.

У даній кваліфікаційній роботі бакалавра було створено три сервіси:

– консольний додаток для отримання даних із самої гри, та запису їх у CSV файл;

– консольний додаток для отримання даних з CSV файлу та їх запису у базу даних;

– Web API сервіс, який відправляє дані у JSON форматі у клієнтську частину.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Завдяки використанню Vite у якості збірника для React додатку, у процесі розробки було забезпечено підтримку HMR (Hot Module Replacement). Це надало можливість автоматично переглядати оновлений інтерфейс без перезавантаження сторінки, що значно підвищило зручність та ефективність розробки. Для перегляду змін достатньо зберегти зміни у коді за допомогою комбінації клавіш Ctrl + S.

Для регулювання ширини екрану, а також детального відображення властивостей елементів структури вебдодатку, використовувались вбудовані інструменти розробника браузера (DevTools).

Дані спочатку були записані на клієнтській частині вебдодатку, у файлі «players.json». Після того, як було створено Web API, файл «players.json» було перенесено у даний сервіс.

Для перевірки чи правильний порт знаходить функція GetPort було використано Fiddler. Для запуску програми у ізольованому середовищі використовувався Docker. У сервісі, який передає дані на клієнтську частину, а також у сервісі, який створює базу даних та записує туди дані, було створено Dockerfile для створення контейнерів, а у кореневій папці проєкту – docker-compose.yml файл, який збирає мультиконтейнер. Visual Studio 2022 дозволяє автоматично створити DockerFile, а також docker-compose файл та автоматично запускати проєкт із них. Однак, для під'єднання бази даних до мультиконтейнеру, сервісу для імпортування даних з CSV файлу у базу даних, а також файлів CSV необхідно доповнити код. Повний код docker-compose.yml відображено у додатку Б.

3.3 Розробка Web API

Із серверної частини дані можна відправити за допомогою Web API застосунку. Його було створено за допомогою команди «dotnet new webapi» у терміналі папки, в якій повинен зберігатися проєкт. Доступ до даних реалізовується за допомогою контроллерів, які визначають кінцеві точки, а також логіку обробки HTTP запитів.

У лістингу 3.5 зображено контроллер для зберігання усіх даних, а також для запису даних.

Лістинг 3.5 – ApiController

```
[Route("api/players")]
[ApiController]
public class SetData : ControllerBase
    {
        private readonly PlayersDBContext _context;
        public SetData(PlayersDBContext context)
        {
            _context = context;
        }
        [HttpPost("save")]
          public async Task<IActionResult> SavePlayersToDatabase()
          {
```

```
                try
                {
                     int offset = 0;
                     var playersFromApi = await
GetData.GetPlayersAsync(offset);
                     var allPlayers = await
_context.PlayerDatas.ToListAsync();
                     _context.PlayerDatas.RemoveRange(allPlayers);
                     await _context.SaveChangesAsync();
                     _context.ChangeTracker.Clear();
                     await
_context.PlayerDatas.AddRangeAsync(playersFromApi);
                     await _context.SaveChangesAsync();
                     return Ok("Database has been cleared and new
data inserted successfully.");
                }
                catch (Exception ex)
                {
                     return StatusCode(500, $"Error resetting and
saving data: {ex.Message}");
                }
          }
          [HttpGet]
        public async Task<IActionResult> GetPlayersFromDatabase()
        {
            try
            {
                var players = await _context.PlayerDatas
                     .Include(pd => pd.player)
                     .Include(pd => pd.country)
                     .Include(pd => pd.rank)
                     .Include(pd => pd.matches)
                     .ThenInclude(m => m.chat)
                     .ToListAsync();
                return Ok(players);
            }
            catch (Exception ex)
            {
                return StatusCode(500, $"Error reading data:
{ex.Message}");
            }
        }
     }
```

кінець лістингу 3.5

Для завантаження записів гри створений окремий контроллер, який відображено у лістингу 3.6.

Лістинг 3.6 – Контролер для завантаження записів гри

```
namespace GGStat_Backend.ApiControllers
{
     [Route("api/matches")]
     [ApiController]
     public class MatchesController : ControllerBase
     {
          private readonly PlayersDBContext _context;

          public MatchesController(PlayersDBContext context)
          {
               _context = context;
          }

          [HttpGet("{matchId}/replay")]
          public async Task<IActionResult> GetReplayLink(string
matchId)
          {
               var replayLink = await GetData.GetLinkAsync(matchId);
               if (string.IsNullOrEmpty(replayLink))
               {
                    return NotFound("Replay link not found.");
               }
               return Ok(new { replayLink });
          }
     }
}
```

кінець лістингу 3.6

Для завантаження матчів використовується окрема функція GetLinkAsync, яка завантажує посилання на матч із серверу гри. Вона відображена у лістингу 3.7.

Лістинг 3.7 – Функція для завантаження матчів

```
public static async Task<string> GetLinkAsync(string match_id)
          {

               if (string.IsNullOrWhiteSpace(match_id))
                    return " ";

               var json = await
JsonParser.GetRequest($"http://127.0.0.1:{Settings.Port}/web-
api/v1/matchmaker-gameinfo-playerinfo/{match_id}");
               var jsonDoc = JsonDocument.Parse(json);
```

```
                if (jsonDoc.RootElement.TryGetProperty("replays", out
var replaysElement) && replaysElement.ValueKind ==
JsonValueKind.Array)
                {
                     foreach (var replay in
replaysElement.EnumerateArray())
                     {

                          if (replay.TryGetProperty("url", out var
urlElement))
                          {
                               var match_link =
urlElement.GetString();
                               if
(!string.IsNullOrWhiteSpace(match_link))
                               {
                                    return match_link;
                               }
                          }
                     }
                }
                return "";
                                        }
```

кінець лістингу 3.7

Головний файл Program.cs, який створює та запускає Web API відображено у лістингу 3.8.

Лістинг 3.8 – Головний файл сервісу з Web API

```
namespace GGStat_Backend
{
     public class Program
     {
          public static void Main(string[] args)
          {
               var builder = WebApplication.CreateBuilder(args);
     builder.Services.AddDbContext<PlayersDBContext>(options =>
     options.UseNpgsql(builder.Configuration.GetConnectionString("De
faultConnection")));
               builder.Services.AddCors(options =>
               {
                    options.AddPolicy("AllowLocalhost", policy =>
                    {
                         policy.WithOrigins("http://localhost:3000")
                               .AllowAnyHeader()
                               .AllowAnyMethod();
```

```
                    });
                });
                builder.Services.AddEndpointsApiExplorer();
                builder.Services.AddSwaggerGen();
     builder.Services.AddControllers().AddNewtonsoftJson();
                var app = builder.Build();
                using (var scope = app.Services.CreateScope())
                {
                    var dbContext =
scope.ServiceProvider.GetRequiredService<PlayersDBContext>();
                }
                app.UseCors("AllowLocalhost");
                if (app.Environment.IsDevelopment())
                {
                    app.UseSwagger();
                    app.UseSwaggerUI();
                }
                app.MapControllers();
                app.Run();
            }
     }
     }
```

кінець лістингу 3.8

У даному коді було підключено усі контроллери, які створені у програмі, а також налаштовано правила доступу до них за допомогою CORS. До даних заборонений доступ усім адресам, крім «http://localhost:3000», під якою працює клієнтська частина. Крім того, у даний сервіс було інтегровано Swagger UI, який дозволяє відобразити роботу endpoint-ів та переглядати структуру запитів і відповідей безпосередньо через інтерфейс. Вигляд Swagger UI зображено на рисунку 3.14.

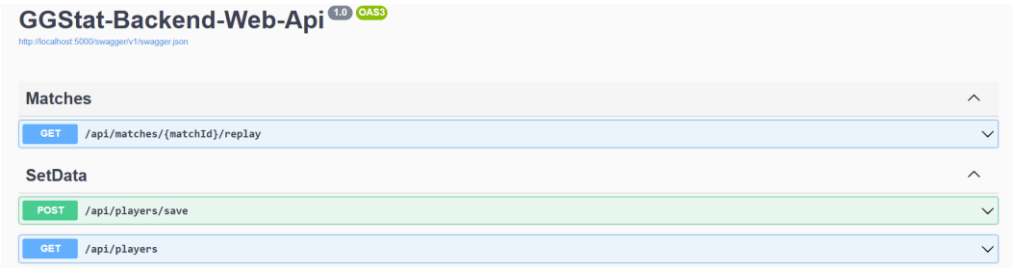

Рисунок 3.14 – Swagger UI

3.4 Отримання даних з API гри

Для того, щоб отримати дані про гравців, спочатку необхідно запустити гру. Це зумовлене тим, що у гри немає Web API, до якого можна отримати доступ за допомогою браузера, але при запуску гри створюється локальний сервер, із якого можна отримати дані, надсилаючи запити до endpoint-ів, які описані на рисунку 3.15.

Endpoint	`SCApi` method	Notes
`/v1/file-set/classic.files.global.maps-1v1`	`classicFilesGlobalMaps1v1()`	List of ladder maps by season
`/v1/gateway`	`gateway()`	Gateways, ids, online player counts
`/v1/leaderboard/{ladder}?offset={offset}&length={length}`	`leaderboardEntity(ladder, offset, length)`	Paginated player rankings
`/v1/leaderboard-name-search/{ladder}/{toon}`	`leaderboardNameSearch(toon)`	Name search of ranked players
`/v1/leaderboard-rank-by-toon/{ladder}/{toon}/{gateway}`	`leaderboardRankByToon(ladder, toon, gateway)`	Ranked data for a given profile
`/v1/leaderboard`	`leaderboard()`	List of all leaderboards
`/v1/map-stats-by-toon/{toon}/{gateway}`	`mapStatsByToon(toon, gateway)`	Win rate by map and season
`/v1/matchmaker-gameinfo-playerinfo/{matchId}`	`matchMakerGameInfoPlayerInfo(matchId)`	Ranked match info and link to replay
`/v2/aurora-profile-by-toon/{toon}/{gateway}?request_flags=scr_mmgameloading`	`auroraProfileByToonv2(toon, gateway, 'scr_mmgameloading')`	Minimal acct info
`/v2/aurora-profile-by-toon/{toon}/{gateway}?request_flags=scr_mmtooninfo`	`auroraProfileByToonv2(toon, gateway, 'scr_mmtooninfo')`	Minimal acct info + recent game played count
`/v2/aurora-profile-by-toon/{toon}/{gateway}?request_flags=scr_profile`	`auroraProfileByToonv2(toon, gateway, 'scr_profile')`	Full acct info
`/v2/aurora-profile-by-toon/{toon}/{gateway}?request_flags=scr_tooninfo`	`auroraProfileByToonv2(toon, gateway, 'scr_tooninfo')`	Full acct info minus game history

Рисунок 3.15 – Список endpoint-ів для отримання даних з гри [7]

У даній кваліфікаційній роботі бакалавра будуть використовуватися три із них:

– «/v1/leaderboard/{ladder}?offset={offset}&length={length}» – для отримання загальних даних про гравців. Параметр «Length» визначає кількість гравців, що отримуються за один запит, offset – кількість гравців, яких необхідно пропустити перед отриманням даних;

– «/v2/aurora-profle-toon/{toon}/{gateway}?request_flags=scr_tooninfo» – для отримання країни гравця, а також історію матчів. Параметр «Toon» є унікальним псевдонім (нікнейм) гравця, gateway – ідентифікатор серверу, на якому зареєстрований аккаунт;

– «/v1/matchmaker-gameinfo-playerinfo/{matchId}» – для отримання посилання на матч у локальному сховищі. «matchId» – це ідентифікатор матчу, який отримується із попереднього endpoint-у.

Для отримання даних із серверу необхідно спочатку запустити куплену копію гри. Після чого необхідно отримати порт, на якому було встановлено з’єднання з локальним сервером гри. Для відстеження порту використовується функція GetPort, яка відображена у лістингу 3.9.

Лістинг 3.9 – Функція для відстеження порту

```
public static async Task<int> GetPort()
     {
          using var httpClient = new HttpClient();
          var loopbackPorts = IPGlobalProperties
.GetIPGlobalProperties()
.GetActiveTcpListeners()
.Where(ep =>
     ep.Address.Equals(IPAddress.Loopback) ||
     ep.Address.Equals(IPAddress.IPv6Loopback))
.Select(ep => ep.Port)
.ToHashSet();
          var localConnections = IPGlobalProperties
               .GetIPGlobalProperties()
               .GetActiveTcpConnections()
               .Where(conn =>
     (conn.LocalEndPoint.Address.Equals(IPAddress.Loopback) &&
                    loopbackPorts.Contains(conn.LocalEndPoint.Port)
))
               .ToList();
```

```
          foreach (var endpoint in localConnections)
          {
               int port = endpoint.LocalEndPoint.Port;
               Console.WriteLine(port);
               string url = $"http://127.0.0.1:{port}/web-
api/v1/leaderboard/{Settings.LeaderboardId}?offset=0&length={Setting
s.BatchSize}";
               try
               {
                    var response = await httpClient.GetAsync(url);
                    if (response.IsSuccessStatusCode)
                    {
                         Console.WriteLine($"StarCraft local web UI
found on port {port}");
                         return port;
                    }
               }
               catch (HttpRequestException)
               {}
               catch (TaskCanceledException)
               {}
          }
          Console.WriteLine(" Could not find SCR web UI server.");
          return 0;
     }
     }
```

кінець лістингу 3.9

Дана функція перебирає усі активні порти, та надсилає запит на сервер, який видобуває загальну інформацію про гравців. Якщо сервер повертає код 200, тобто запит є успішним, функція повертає значення цього порту.

Далі необхідно власне здійснити запити до серверу. Код для створення запиту на сервер відображено у лістингу 3.10.

Лістинг 3.10 – Код для створення запиту на сервер

```
private static readonly HttpClient client = new HttpClient();
public static async Task<string> GetRequest(string url)
{
     try
     {
          HttpResponseMessage response = await client.GetAsync(url);
          response.EnsureSuccessStatusCode();
          string responseBody = await
response.Content.ReadAsStringAsync();
```

```
            return responseBody;
        }
        catch (HttpRequestException e)
        {
            return $"Error: {e.Message}";
        }
                                    }
```

кінець лістингу 3.10

Після цього було створено класи, які відображають основні дані. Класи будуть створені на основи діаграми класів, яка була реалізована у попередньому розділі. Клас PlayerData, який зберігає усю інформацію про гравця, описано на лістингу 3.11.

Лістинг 3.11 – Клас «PlayerData»

```
public class PlayerData
{
    [Key]
    public int Id { get; set; }
    public int standing { get; set; }
    public int PlayerId { get; set; }
    public Player player { get; set; }
    public int CountryId { get; set; }
    public CountryInfo country { get; set; }
    public int RankId { get; set; } /
    public Rank rank { get; set; }
    public string race { get; set; }
    public int wins { get; set; }
    public int loses { get; set; }
    public List<Match>? matches { get; set; } = new List<Match>();
}
```

кінець лістингу 3.11

Клас Player, який відображає базову інформацію про гравця, відображено на лістингу 3.12.

Лістинг 3.12 – Клас «Player»

```
[Key]
 public int Id { get; set; }
 public string? name { get; set; }
```

```
 public string? region { get; set; }
 public string? alias { get; set; }
 public string? avatar { get; set; }
```

кінець лістингу 3.12

Клас Country відображено у лістингу 3.13. З метою уникнення конфліктів із зовнішніми бібліотеками, які використовуються у даному проєкті було вирішено змінити ім’я класу на CountryInfo.

Лістинг 3.13 – Клас «CountryInfo»

```
namespace GGStatParsingDataService.models
{
    public class CountryInfo
    {
        [Key]
        public int id { get; set; }
        public string? code { get; set; }
        public string? flag { get; set; }
    }
}
```

кінець лістингу 3.13

Клас Rank, за допомогою якого зберігаються дані про MMR гравця, а також його ліга, зображено на лістингу 3.14.

Лістинг 3.14 – Клас «Rank»

```
public class Rank
{
    [Key]
    public int Id { get; set; }
    public int points { get; set; }
    public string league { get; set; }
}
```

кінець лістингу 3.14

Клас Match, за допомогою якого зберігається історія матчів, відображено у лістингу 3.15.

Лістинг 3.15 – Клас «Match»

```
public class Match
{
     [Key]
     public int Id { get; set; }
     public string? match_id { get; set; }
     public string? match_link { get; set; }
     public string? result { get; set; }
     public int points { get; set; }
     public string? timeAgo { get; set; }
     public string? map { get; set; }
     public string? duration { get; set; }
     public string? player_race { get; set; }
     public string? opponent_race { get; set; }
     public string? opponent { get; set; };
}
```

кінець лістингу 3.15

Клас Chat за допомогою якого зберігається історія повідомлень показано у лістингу 3.16.

Лістинг 3.16 – Клас «Chat»

```
public class Chat
     {
          [Key]
          public int id { get; set; }
          public string? time { get; set; }
          public string? player { get; set; }
          public string? message { get; set; }
}
```

кінець лістингу 3.16

Сервіс для збору даних працює поетапно. На першому етапі сервіс збирає загальні дані про гравців, крім історії матчів, та країни гравця, та записує ці дані у файл players.csv. На наступному етапі дані імпортуються з цього файлу, після чого на основі кожного запису робляться запити до API гри про країну походження гравця, та його історію матчів. На основі цих даних доповнюється список, який вже записується в players_with_countries.csv. Код для запису основних даних про гравця описано у додатку В. У цьому додатку також

включені функції для обчислення ліги гравця на основі його MMR, а також функція «GetRegion», яка повертає регіон гравця у зрозумілому для користувача вигляді відповідно до ідентифікатора.

Код, який записує дані в players.csv зображено у лістингу 3.17.

Лістинг 3.17 – Код для запису даних в players.csv

```
public static async Task WriteToCsvAsync(List<PlayerData> data,
string filePath)
{
     using (var writer = new StreamWriter(filePath))
     using (var csv = new CsvWriter(writer,
CultureInfo.InvariantCulture))
     {
          csv.Context.RegisterClassMap<PlayerDataMap>();
          csv.WriteHeader<PlayerData>();
          csv.NextRecord();
          await csv.WriteRecordsAsync(data);
     }
}
```

кінець лістингу 3.17

Клас «PlayerDataMap», який було використано у попередньому коді, відображений у лістингу 3.18.

Лістинг 3.18 – Клас «PlayerDataMap»

```
public class PlayerDataMap : ClassMap<PlayerData>
{public PlayerDataMap(){
          Map(p => p.standing);
          Map(p => p.player.name);
          Map(p => p.player.alias);
          Map(p => p.player.region);
          Map(p => p.player.avatar);
          Map(p => p.country.code);
          Map(p => p.country.flag);
          Map(p => p.rank.points);
          Map(p => p.rank.league);
          Map(p => p.race);
          Map(p => p.wins);
          Map(p => p.loses);
          Map(p => p.matches);
}}
```

кінець лістингу 3.18

Код для запису країни описано у лістингу 3.19. Оскільки більшість сервісів для генерації прапорів використовують Alpha-2 кодування (тобто, код країни записується двома літерами), а у грі країна гравця зберігається у Alpha-3 кодуванні (тобто, код країни записується трьома літерами), у даній функції було використано бібліотеку «Nager.Country», яка зберігає Alpha-3 та Alpha-2 кодування країни, що дозволяє швидко перетворити код країни у правильний формат.

Лістинг 3.19 – Код для запису країни

```
public static async Task<string> GetCountry(string json)
{
     var jsonDoc = JsonDocument.Parse(json);
     var alpha3Code =
jsonDoc.RootElement.GetProperty("country_code").GetString();
     var countryProvider = new CountryProvider();
     var countries = countryProvider.GetCountries();
     var alpha2Code = countries.FirstOrDefault(c =>
c.Alpha3Code.ToString() == alpha3Code);
     if (alpha2Code != null) return
alpha2Code.Alpha2Code.ToString();
     else return "Unknown";
     }
```

кінець лістингу 3.19

Для отримання даних про матчі було створено окрему функцію, яку відображено у додатку Г. Також у додатку відображено функцію GetTime, яка перетворює час у вигляд, зрозумілий кінцевому користувачу.

Далі уже доповнений список було записано у players_with_countries.csv за допомогою функції із лістингу 3.17, але замість класу «PlayerDataMap» використовувався клас «PlayerDataMapWithCountry», який відображений у лістингу 3.20.

Лістинг 3.20 – Клас «PlayerDataMapWithCountry»

```
public class PlayerDataMapWithCountry : ClassMap<PlayerData>
{
     public PlayerDataMapWithCountry()
```

```
    {
         Map(p => p.standing);
         Map(p => p.player.name);
         Map(p => p.player.alias);
         Map(p => p.player.region);
         Map(p => p.player.avatar);
         Map(p => p.country.code);
         Map(p => p.country.flag);
         Map(p => p.rank.points);
         Map(p => p.rank.league);
         Map(p => p.race);
         Map(p => p.wins);
         Map(p => p.loses);
         Map(p => p.matches).Convert(p =>
              string.Join(" | ", p.Value.matches.Select(m =>

    $"{m.match_id},{m.match_link},{m.result},{m.points},{m.timeAgo}
," +

    $"{m.map},{m.duration},{m.player_race},{m.opponent_race},{m.opp
onent}")));
    }
    }
```

кінець лістингу 3.20

Далі сервіс для запису у базу даних зчитує дані з файлу «players_with_countries.csv», та записує їх у список. Для цього використовувався код, ідентичний до лістингу 3.17. Після цього було створено базу даних та додано туди дані зі списку.

3.5 Розробка бази даних

База даних була розроблена за допомогою Entity Framework Core, використовуючи міграції. Для початку було створено контекст, який відображено у додатку Д.

Після цього необхідно створити початкову міграцію за допомогою команди «Add-Migration InitialCreate» у консолі менеджеру пакетів NuGet. Далі необхідно було налаштувати рядок підключення до бази даних, який містить усі необхідні параметри для створення зв’язку з базою даних, зокрема, назву бази даних,

пароль, ім’я користувача, порт підключення, та код доступу. Рядок підключення знаходиться в файлі appsettings.json, який відображений у лістингу 3.21.

Лістинг 3.21 – Appsettings.json

```
{
  "exclude": [
    "**/bin",
    "**/bower_components",
    "**/jspm_packages",
    "**/node_modules",
    "**/obj",
    "**/platforms"
  ],
  "ConnectionStrings": {
    "DefaultConnection":
"Host=postgres_db;Database=GGStatDB;Username=postgres;Password=maxim
12345"
  }
}
```

кінець лістингу 3.21

Після цього було додано код, який зображено на лістингу 3.22 для створення зв’язку з системою керування базами даних.

Лістинг 3.22 – Код для зв’язку з базою даних

```
var configuration = new ConfigurationBuilder()
     .SetBasePath(Directory.GetCurrentDirectory())
     .AddJsonFile("appsettings.json", optional: false,
reloadOnChange: true)
     .Build();
var services = new ServiceCollection();
services.AddDbContext<PlayersDBContext>(options =>
     options.UseNpgsql(configuration.GetConnectionString("DefaultCon
nection")));
                    services.AddTransient<SetData>();
```

кінець лістингу 3.22

Після запуску проєкту буде автоматично створено базу даних згідно існуючих моделей, якщо її не існує.

Код для додавання даних відображено у лістингу 3.23.

Лістинг 3.23 – Код для заповнення бази даних

```
public SetData(PlayersDBContext context)
{
     _context = context;
}

public async Task SavePlayersToDatabase()
{
     var playersFromCsv = GetDataFromCSV.SaveData();
     Console.WriteLine($"📄 Players loaded from CSV:
{playersFromCsv.Count}");

     foreach (var player in playersFromCsv)
     {
          var existingPlayer = await _context.PlayerDatas
               .Include(p => p.matches)
               .FirstOrDefaultAsync(p => p.standing ==
player.standing);

          if (existingPlayer != null)
          {
               _context.Matches.RemoveRange(existingPlayer.matches);
               await _context.SaveChangesAsync();
               player.Id = existingPlayer.Id;
               _context.PlayerDatas.Update(player);
          }
          else
          {
               await _context.PlayerDatas.AddAsync(player);
          }
     }
     await _context.SaveChangesAsync();
     }
```

кінець лістингу 3.23

Далі, використовуючи контекст для розробки бази даних, було зчитано звідти усі дані, та записані в список, який і передається через Web API.

Висновки до розділу 3

У даному розділі було розроблено вебдодаток з клієнт-серверною архітектурою для відображення таблиці рейтингів у грі «Starcraft:Remastered». Програма містить таблицю лідерів в грі, найкращого гравця кожної країни, а

також профіль гравця, в якому можна завантажити матч. Дизайн усіх сторінок є адаптивним, тобто, має здатність підлаштовуватися під екран користувача. Крім цього, у таблиці існує можливість відобразити всіх гравців лише певної країни, ліги та раси за яку найбільше ігор зіграно у гравця.

ВИСНОВКИ

Під час виконання першого розділу було проаналізовано розвиток сучасної індустрії відеоігор, а також було досліджено, які саме характеристики впливають на якість ігрового процесу.

На основі аналізу було визначено завдання, які повинні бути виконані для успішної реалізації.

На етапі проєктування було проаналізовано аналоги, та на основі цього було визначено основні вимоги до вебдодатку, а також було розроблено UML-діаграму класів, яка відображає структуру, в якій повинні зберігатися дані.

Для реалізації вебдодатку було обрано стек технологій. У якості інструмента для розробки Front-end частини вебдодатку було обрано React на базі збірника Vite, та підключеною бібліотекою MUI. У якості Back-end частини вебдодатку було обрано ASP.NET Core, та Entity Framework Core для підключення до бази даних. У якості системи керування базами даних було обрано PostgreSQL.

У практичній частині було розроблено вебдодаток для відображення таблиці рейтингів гравців. Реалізовано відображення гравців лише певної раси, ліги та країни, а також завантаження запису гри. Було створено адаптивний дизайн, який дозволяє зручно відобразити усі дані на екранах з невеликим розміром.

Було проведено тестування вебдодатку з метою перевірки правильності функціонування, а також відповідності вимогам, які були попередньо описані. Вебдодаток правильно функціонує, та відповідає поставленим вимогам.

Було створено Web API, який контролює передачу даних із Back-end частини у Front-end частину. Дані передаються за допомогою двох контролерів, для завантаження усіх даних та для завантаження запису гри.

Для отримання даних було проаналізовано структуру endpoint-ів, визначено механізм отримання даних із гри, а також їх запису. На основі цього аналізу було визначено основний механізм зберігання даних.

За допомогою Entity Framework Core, а саме міграцій, які створюються на базі контекстів, було створено базу даних. Використовуючи connectionString, було налаштовано підключення до бази даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Angular. URL: https://v17.angular.io/guide/what-is-angular (дата звернення: 18.03.2025).

2. ASP.NET core, an open-source web development framework. URL: https://dotnet.microsoft.com/en-us/apps/aspnet (дата звернення: 18.03.2025).

3. Boosting: Rank and skill deception in esports / E. Conroy et al. Entertainment Computing. 2021. Vol. 36. P. 100393. URL: https://doi.org/10.1016/j.entcom.2020.100393 (дата звернення: 01.03.2025).

4. Chess.com. URL: https://www.chess.com/leaderboard/live/rapid (дата звернення: 01.03.2025).

5. Cwal.gg. URL: https://cwal.gg/ (дата звернення: 17.03.2025).

6. Games RepMastered. URL: https://repmastered.icza.net/ (дата звернення: 17.03.2025).

7. GitHub. URL: https://github.com/evanandrewrose/bw-web-api (дата звернення: 01.04.2025).

8. Google-Ergebnis. URL: https://images.app.goo.gl/boZtbJ51ryosV2aHA (дата звернення: 01.03.2025).

9. Google-Ergebnis. URL: https://images.app.goo.gl/rewJ7s95zbG1Cu4b6 (дата звернення: 16.03.2025).

10. Google-Ergebnis. URL: https://images.app.goo.gl/oTL8ySRtVDRJZYPa (дата звернення: 18.03.2025).

11. Invalid Dynamic Link. URL: https://images.app.goo.gl (дата звернення: 18.03.2025).

12. Laravel. The PHP Framework For Web Artisans. URL: https://laravel.com/docs/11.x (дата звернення: 18.03.2025).

13. The Effects of Network Latency on Competitive First-Person Shooter Game Players / S. Liu et al. URL: https://doi.org/10.1109/qomex51781.2021.9465419 (дата звернення: 01.03.2025).

14. The impact of eSports and online video gaming on lifestyle behaviours in youth: A systematic review / G. Chan et al. Computers in Human Behavior. URL: https://doi.org/10.1016/j.chb.2021.106974 (дата звернення: 01.03.2025).

ДОДАТКИ

Додаток А – Підтвердження апробації результатів кваліфікаційної роботи

Міністерство освіти і науки України
Волинська обласна рада
Луцька міська рада
Луцький національний технічний університет
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Національний університет «Львівська політехніка»
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут (м. Київ)
Тернопільський національний педагогічний університет імені В. Гнатюка
Рівненський державний гуманітарний університет
Навчально-науковий інститут «Українська інженерно-педагогічна академія» Харківського національного університету ім. В.Н. Каразіна
Люблінська політехніка (Польща)
Фрайберзька гірнича академія (Німеччина)
Гронінгенський університет (Нідерланди)
Кавказький університет (Грузія)
Політехнічний університет Браганси (Португалія)

ІТОНВ

ТЕЗИ ДОПОВІДЕЙ
X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)

23-24 травня 2025 р.

Луцьк 2025

УДК 004.41

ВИБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ ОНЛАЙН ТАБЛИЦІ ЛІДЕРІВ В ГРІ «STARCRAFT:REMASTERED»

Мельник Максим Вікторович
Луцький національний технічний університет, студент,
бакалавр інженерії програмного забезпечення, maximmelnik37@gmail.com
Самчук Людмила Михайлівна
Луцький національний технічний університет, кандидат технічних наук,
доцент кафедри прикладної механіки та мехатроніки, samchuk204@gmail.com

CHOOSING TOOLS FOR DEVELOPMENT OF LEADERBOARD IN GAME «STARCRAFT:REMASTERED»

Melnyk Maxym Viktorovych
Lutsk National Technical University, Student, Bachelor of Software Engineering,
maximmelnik37@gmail.com
Samchuk Liudmyla Mykhailivna
Lutsk National Technical University, PhD inTechnical Sciences,
AssociateProfessorofthe Department ofAppliedMechanicsandMechatronics,
samchuk204@gmail.com

In modern era of digital technologies, online videogames became a big part of entertainment industry. Online multiplayer games, such as Starcraft:Remastered, require data management system to track gameplay, statistics, and match history to increase skill of players. This system requires flexible and reliable data structure.This paper describes analyzing of existing tools and chosing most efficient tools for developing online leaderboard.

***Keywords**: MMR, videogame, front-end, back-end, CIL*

У сучасному ігровому середовищі система лідерборду (таблиці лідерів) виконує важливу функцію мотивації гравців, формування спільноти та підтримки змагального духу. Зважаючи на популярність гри «StarCraft: Remastered» та її орієнтацію на PvP-контент, розробка інтерактивної, надійної та масштабованої онлайн таблиці лідерів потребує грамотного вибору технологічного стеку.

Перш за все, для розробки необхідно визначити архітектуру проєкту. Для роботи було обрано клієнт – серверну архітектуру проєкту, тому що необхідно обробляти велику

кількість даних, а також важко їх отримати. Це значить, що проєкт буде складатися з таких частин:

– клієнтська частина(front-end), тобто частина, яка відповідає за основний вигляд веб – додатку;

– серверна частина(back-end), тобто частина, яка відповідає за взаємодію з сервером.

Оскілки обидві частини мають зовсім різне призначення, то і відповідно, інструменти для їх створення та налаштування будуть зовсім різними. Аналіз вимог до системи: збирання та відображення результатів гравців у реальному часі, фільтрація за регіонами, сезоном, расами тощо, велика кількість запитів до бази даних у пікові години, запобігання фальсифікації результатів, можливість інтеграції з Battle.net API (за наявності відкритого доступу до статистики). Метою цієї роботи є обґрунтований вибір інструментів для розробки такої системи з урахуванням вимог до продуктивності, безпеки, масштабованості та зручності використання.

React – це бібліотека для мови JavaScript з відкритим кодом. Його основною ідеєю є розбиття елементів на незалежні компоненти,які мають свою логіку та стан. За допомогою useStatета useEffect можна зберігати стан компонента. React підтримує сторонні бібліотеки, що дозволяє легко змінювати архітектуру проєкту за власним бажанням. Недоліком React ємалий вбудований функціонал. Він підтримує лише безпосередньо розробку інтерфейсів. Якщо під час розробки виникла потреба для розробки інших засобів, необхідно підключати сторонні бібліотеки.

Angular-це повноцінний фреймворк на базі мови TypeScript, що має в собі великий вбудований набір інструментів для забезпечення різних потреб від маршрутизації до роботи з HTTP запитами. У порівнянні з JavaScript, TypeScript забезпечує статичну типізацію, що робить його зручним інструментом для роботи з великим кодом. Однак, для роботи з маленькими та середніми проєктами, Angular є

незручним, через те, що навіть дляпростих завдань необхідна велика кількість коду та структуризація проєкту.

Vue – це бібліотека для мови програмування JavaScript, яка відрізняється своєю простотою. Як і React, складається з компонентів. Особливістю Vue є реактивність, тобто, інтерфейс оновлюється автоматично після оновлення коду. Vue дозволяє самостійно обрати підхід до реалізації логіки, структури, стилів. Це робить даний фреймворк гнучким, але дуже часто призводить до появи так званого «коду – спагетті» - складно читабельного коду без структури з великою кількістю переплетень.Крім того, Vue має меншу кількість доступних сторонніх бібліотек у порівнянні з React, а також меншу кількість вбудованих функцій у порівнянні з Angular.

Проаналізувавши інформацію, описану вище, було прийнято рішення використовувати React для розробки клієнтської частини веб додатку, оскільки у даному проєкті немає необхідності для великого набору вбудованих функцій. Однак, через специфічність проєкту необхідно підключати сторонні бібліотеки, через що і було обрано React.

Node.js – це середовище виконання JavaScript, що дозволяє працювати з даною мовою програмування у серверній частині проєкту. Використовує один потік, в якому всі функції керуються за допомогою асинхронного програмування(тобто, код виконується не по порядку написання), що дозволяє обробляти велику кількість запитів, але не підходить для розробки програмного забезпечення, яке вимагає інтенсивну роботу з процесором.

ASP.NET Core – це фреймворк для мови програмування C# та системи .NET, який був розроблений Microsoft. Має підтримку Entity Framework Core, що дозволяє автоматично створювати бази даних за допомогою міграцій. Підтримує архітектуру MVC(Model-View-Controller). Підтримує Docker, що дозволяє створити контейнер, який буде автоматично налаштований для проєкту. Дозволяє автоматично створювати нові проєкти із уже існуючого набору шаблонів, який є дуже

великим та покриває практично усі потреби для розробки серверних додатків. Його головними недоліками є велика складність у освоєнні, а також повільний перший запуск проєкту через те, що код не одразу компілюється у машинний код, а спочатку компілюється у CIL(Common Intermediate Language), а уже після цього – у машинний код.

Laravel – це фреймворк для мови програмування PHP, який створений для роботи з сервером та працює на архітектурі MVC. Має гарно структурований код у порівнянні з іншими інструментами. Має уже готові рішення для створення аутентифікації, транзакцій, роботою з поштою тощо. Однак, має нижчу продуктивність у порівнянні з ASP.NET Core чи Node.js.

Проаналізувавши технічні характеристики, функціональні можливості та архітектурні особливості представлених інструментів, було обґрунтовано вибір ASP.NET Core як бекенд-основи для розробки серверної частини системи онлайн таблиці лідерів у грі «StarCraft: Remastered». Цей фреймворк демонструє високу продуктивність, асинхронність обробки запитів, підтримку сучасних стандартів розробки. Застосування клієнт-серверної архітектури дає змогу чітко розмежувати обов'язки між клієнтським інтерфейсом (UI) та серверною логікою, що підвищує масштабованість, зручність супроводу та гнучкість при розширенні функціональності. Такий підхід є виправданим у випадку з системою, яка повинна обслуговувати динамічну статистику, підтримувати фільтрацію, обробку великої кількості запитів у реальному часі та зберігати історичні дані.

Список використаних джерел

1. "Гейміфікація в маркетингу: як залучити та утримати клієнтів за допомогою ігрових стратегій" / / О.І. Сидоренко, К.О. Марченко, Л.М. Гончаренко; Видавництво "МаркетГейм", Херсон, 2023. – 110 с.

СЕРТИФІКАТ

Кафедра цифрових освітніх технологій

Кафедра інженерії програмного забезпечення

Даний сертифікат засвідчує, що

Мельник Максим Вікторович

брав(-ла) участь у

X Міжнародній науково-практичній конференції
з проблем вищої освіти і науки
"Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)"
загальним обсягом 15 годин (0,5 кредитів ECTS),

яка проводилася 23-24 травня 2025 року у м.Луцьк
на базі Луцького національного технічного університету

Ректор ЛНТУ _____ **Ірина ВАХОВИЧ**

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

№ ІТОНВ-2025-101

Програма конференції: https://itonv.lntu.edu.ua/files/2025/program-itonv-2025.pdf

Додаток Б

Файл docker-compose.yml

```
version: '3.4'
services:
  ggstat-db-init:
    image: ${DOCKER_REGISTRY-}ggstatdbinit
    build:
      context: .
      dockerfile: src/GGStat-ImporterService/Dockerfile
    volumes:
    - ./db:/app/db
    depends_on:
      postgres_db:
        condition: service_healthy
    environment:
      -
ConnectionStrings__DefaultConnection=Host=postgres_db;Port=5432;Database=GGStatDB;Username=
postgres;Password=maxim12345
    restart: "no"
  ggstat-backend-web-api:
    image: ${DOCKER_REGISTRY-}ggstatbackendwebapi
    build:
      context: .
      dockerfile: src/GGStat-Web-Api/Dockerfile
    depends_on:
      ggstat-db-init:
        condition: service_completed_successfully
    environment:
      -
ConnectionStrings__DefaultConnection=Host=postgres_db;Port=5432;Database=GGStatDB;Username=
postgres;Password=maxim12345
      - ASPNETCORE_URLS=http://+:5000;https://+:5001
    ports:
      - "5000:5000"
      - "5001:5001"
  postgres_db:
    image: postgres:15
    container_name: ggstat_postgres
    restart: always
    environment:
      POSTGRES_DB: GGStatDB
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: maxim12345
    ports:
      - "5432:5432"
    volumes:
      - pg_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 10s
      timeout: 5s
      retries: 5
volumes:
        pg_data:
```

Додаток В

Код для запису даних у список

```
	public static async Task<List<PlayerData>> GetPlayersAsync(int offset)
	{
		string GetRegion(int gateway_id)
		{
			if (gateway_id == 20) return "Europe";
			else if (gateway_id == 30) return "Korea";
			else if (gateway_id == 10) return "US West";
			else if (gateway_id == 11) return "US East";
			else if (gateway_id == 45) return "Asia";
			else return " ";
		}
		string CalcLeague(int points)
		{
			if (points < 1137) return "F";
			else if (points >= 1137 && points < 1426) return "E";
			else if (points >= 1426 && points < 1549) return "D";
			else if (points >= 1549 && points < 1697) return "C";
			else if (points >= 1697 && points < 2014) return "B";
			else if (points >= 2014 && points < 2370) return "A";
			else return "S";
		}
		var json = await
JsonParser.GetRequest($"http://127.0.0.1:{Settings.Port}/web-
api/v1/leaderboard/{Settings.LeaderboardId}?offset={offset}&length={Settings.BatchSize}");
		var jsonDoc = JsonDocument.Parse(json);
		var rows = jsonDoc.RootElement.GetProperty("rows");
		List<PlayerData> players = new List<PlayerData>();

		foreach (var row in rows.EnumerateArray())
		{
			var _player = row[7].GetString();
			var _region = row[2].GetInt32();
			var player_json = await
JsonParser.GetRequest($"http://127.0.0.1:{Settings.Port}/web-api/v2/aurora-profile-by-
toon/{_player}/{_region}?request_flags=scr_profile");
			var _country = "";
			var _points = 0;
			_points = row[3].GetInt32();

			var player = new PlayerData
			{
				standing = row[0].GetInt32(),
				player = new Player
				{
					name = row[7].GetString(),
					alias = row[8].GetString(),
					region = row[2].GetInt32().ToString(),
					avatar = row[9].GetString(),
				},
				country = new CountryInfo
				{
	code = _country,
	flag = $"https://flagcdn.com/w40/{_country.ToLower()}.png",
				},
				rank = new Rank
				{
					points = _points,
					league = CalcLeague(_points),
				},
				race = row[10].GetString().Substring(0, 1).ToUpper(),
				wins = row[4].GetInt32(),
				loses = row[5].GetInt32(),
```

```
                matches = null,
            };
            players.Add(player);
        }
        return players;
}
}
```

Додаток Г

Функція для отримання історії матчів

```
public static async Task<List<Match>> GetMatchHistory(string player, string json)
{
	string GetTime(long unixTime)
	{
		DateTimeOffset dateTime = DateTimeOffset.FromUnixTimeSeconds(unixTime);
		DateTime normalDate = dateTime.UtcDateTime;
		DateTimeOffset currentDateTime = DateTimeOffset.UtcNow;
		TimeSpan timeElapsed = currentDateTime - dateTime;
		return $"{timeElapsed.Days} days {timeElapsed.Hours} hours
{timeElapsed.Minutes} minutes ago";
	}
	var jsonDoc = JsonDocument.Parse(json);
	var games = jsonDoc.RootElement.GetProperty("game_results");
	List<Match> values = new List<Match>();
	foreach (var game in games.EnumerateArray())
	{
		var _match_id = game.GetProperty("match_guid").GetString();
		var attributes = game.GetProperty("attributes");
		var mapName = attributes.GetProperty("mapName").GetString();
		var createTime = game.GetProperty("create_time").GetString();
		var create_time = int.Parse(createTime);
		var timeAgo = GetTime(create_time);
		var players = game.GetProperty("players");
		string playerRace = "";
		string opponentRace = "";
		string opponentName = "";
		string result = "";
		foreach (var playerInfo in players.EnumerateArray())
		{
			var playerAttributes = playerInfo.GetProperty("attributes");
			string toon = playerInfo.GetProperty("toon").GetString();
			string playerResult = playerInfo.GetProperty("result").GetString();
			if (toon == player)
			{
	playerRace = playerAttributes.GetProperty("race").GetString();
	result = playerResult;
			}
			else if (!string.IsNullOrEmpty(toon))
			{
	opponentRace =playerAttributes.GetProperty("race").GetString();
				opponentName = toon;
			}
		}
		var match = new Match
		{
			match_id = _match_id,
			match_link = null,
			map = mapName,
			timeAgo = timeAgo,
			player_race = playerRace,
			opponent_race = opponentRace,
			opponent = opponentName,
			result = result,
			chat = { },
		};
		values.Add(match);
	}
	return values;
	}
```

Додаток Д

Контекст для створення бази даних

```
public class PlayersDBContext : DbContext
{
	public PlayersDBContext(DbContextOptions<PlayersDBContext> options) : base(options)
{ }

	public DbSet<PlayerData> PlayerDatas { get; set; }
	public DbSet<Player> Players { get; set; }
	public DbSet<CountryInfo> CountryInfos { get; set; }
	public DbSet<Rank> Ranks { get; set; }
	public DbSet<Match> Matches { get; set; }
	public DbSet<Chat> Chats { get; set; }
	protected override void OnModelCreating(ModelBuilder modelBuilder)
	{
		modelBuilder.Entity<PlayerData>()
			.HasOne(pd => pd.player)
			.WithMany()
			.HasForeignKey(pd => pd.PlayerId)
			.OnDelete(DeleteBehavior.Cascade);
		modelBuilder.Entity<PlayerData>()
			.HasOne(pd => pd.country)
			.WithMany()
			.HasForeignKey(pd => pd.CountryId)
			.OnDelete(DeleteBehavior.Cascade);
		modelBuilder.Entity<PlayerData>()
			.HasOne(pd => pd.rank)
			.WithMany()
			.HasForeignKey(pd => pd.RankId)
			.OnDelete(DeleteBehavior.Cascade);
		modelBuilder.Entity<PlayerData>()
.HasMany(pd => pd.matches)
.WithOne(m => m.PlayerData)
.HasForeignKey(m => m.PlayerDataId)
.OnDelete(DeleteBehavior.Cascade);
		modelBuilder.Entity<Match>()
.HasMany(m => m.chat)
.WithOne()
.HasForeignKey("MatchId")
.OnDelete(DeleteBehavior.Cascade);
	}
}
public class PlayersDBContextFactory : IDesignTimeDbContextFactory<PlayersDBContext>
{
	public PlayersDBContext CreateDbContext(string[] args)
	{
		var config = new ConfigurationBuilder()
			.SetBasePath(Directory.GetCurrentDirectory())
			.AddJsonFile("appsettings.json")
			.Build();

		var optionsBuilder = new DbContextOptionsBuilder<PlayersDBContext>();
		var connectionString = config.GetConnectionString("DefaultConnection");
		optionsBuilder.UseNpgsql(connectionString);
		return new PlayersDBContext(optionsBuilder.Options);
	}
                                        }
```