Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКУ З ФУНКЦІЯМИ ПЕРЕГЛЯДУ ХАРАКТЕРИСТИК ЧЕМПІОНІВ ТА АЛГОМИТРІЧНОГО ФОРМУВАННЯ РЕКОМЕНДАЦІЙ У ГРІ LEAGUE OF LEGENS
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-22-3
Яблунівський А. С.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Руденко Д.О.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Яблунівському Артуру Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення вебзастосунку з функціями перегляду характеристик чемпіонів та алгоритмічного формування рекомендацій у грі League of Legends

затверджена наказом університету від 26 травня 2026 року № 435Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 19 червня 2026 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали інтернет-мережі, документація з ASP.NET Core, Razor Pages, C#, JavaScript, HTML, CSS, JSON, відомості про предметну область гри League of Legends, статичні дані про чемпіонів, середовище Visual Studio 2022, програмний проєкт LoLChampionGuide.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз предметної області гри League of Legends та визначення вимог до вебзастосунку інформаційної підтримки гравця.
2. Проєктування архітектури вебзастосунку, структури даних про чемпіонів та алгоритму формування рекомендацій.
3. Програмна реалізація, тестування функціональних можливостей і визначення напрямів подальшого розвитку вебзастосунку.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність роботи, постановка задачі, структура вебзастосунку, інформаційна модель чемпіона, схема алгоритму рекомендацій, інтерфейс інтерактивної карти, каталог чемпіонів, аналізатор матчу, результати тестування.

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

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	13.04.2026	
2	Аналіз завдання, підбір літератури	14.04.26-16.04.26	
3	Аналіз літератури з проблеми, що вирішується	17.04.26-19.04.26	
4	Аналіз існуючих методів розпізнавання	20.04.26-22.04.26	
5	Формування кроків вирішення проблеми	23.04.26-05.05.26	
6	Програмна реалізація	26.04.26-20.05.26	
7	Аналіз отриманих результатів	21.05.26-04.06.26	
8	Оформлення пояснювальної записки	05.06.26-09.06.26	
9	Перевірка на нормоконтроль	10.06.26-19.06.26	
10	Перевірка на плагіат	11.06.26-30.06.26	
11	Рецензування	20.06.26-30.06.26	
12	Підготовка презентації та доповіді	20.06.26-30.06.26	
13	Занесення роботи в електронний архів	30.06.26-09.07.26	
14	Попередній захист кваліфікаційної роботи	30.06.26-09.07.26	

Дата видачі завдання 13 квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ _____доц. Руденко Д.О._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 92 с., 25 табл., 13 рис., 40 джерел.

ВЕБЗАСТОСУНОК, LEAGUE OF LEGENDS, ЧЕМПІОН, РЕКОМЕНДАЦІЙНИЙ АЛГОРИТМ, ASP.NET CORE, RAZOR PAGES, C#, JAVASCRIPT, JSON, ІНТЕРАКТИВНА КАРТА, DATA DRAGON.

Об’єктом роботи є процес перегляду характеристик чемпіонів і формування рекомендацій за складом команд у грі League of Legends.

Метою роботи є створення багатосторінкового вебзастосунку з інтерактивною картою, каталогом чемпіонів, детальними картками та аналізатором матчу.

У роботі використано методи аналізу, об’єктно-орієнтованого проєктування, веброзробки, оброблення JSON-даних і функціонального тестування.

У результаті створено вебзастосунок «LoL Champion Guide & Recommender» засобами ASP.NET Core Razor Pages, C#, HTML, CSS, JavaScript, JSON і Data Dragon.

Практична цінність роботи полягає у навчально-аналітичній підтримці вибору чемпіонів і прийняття рішень у матчі.

WEB APPLICATION, LEAGUE OF LEGENDS, CHAMPION, RECOMMENDATION ALGORITHM, ASP.NET CORE, RAZOR PAGES, C#, JAVASCRIPT, JSON, INTERACTIVE MAP, DATA DRAGON.

The object of this work is the process of reviewing champion characteristics and generating recommendations based on team compositions in the game League of Legends.

The aim of the work is to create a multipage web application with an interactive map, a champion catalog, detailed cards, and a match analyzer.

The methods include analysis, object-oriented design, web development, JSON data processing, and functional testing.

As a result, the “LoL Champion Guide & Recommender” web application was created using ASP.NET Core Razor Pages, C#, HTML, CSS, JavaScript, JSON, and Data Dragon.

The practical value of the work lies in supporting champion selection and in-match decision-making.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface (інтерфейс програмного застосунку, використовується для обміну даними між клієнтом і сервером)

ASP.NET Core – платформа Microsoft для створення вебзастосунків і веб-API мовою C#.

C# – об’єктно-орієнтована мова програмування для реалізації серверної логіки.

CSS – Cascading Style Sheets, мова опису стилів вебсторінок.

Data Dragon – офіційний набір статичних ресурсів Riot Games для League of Legends.

HTML – HyperText Markup Language, мова розмітки вебсторінок.

JavaScript – мова програмування для інтерактивної поведінки сторінок у браузері.

JSON – JavaScript Object Notation, текстовий формат зберігання та обміну даними.

localStorage – механізм браузера для локального збереження даних користувача.

MOBA – Multiplayer Online Battle Arena, жанр командних онлайн-ігор.

Razor Pages – підхід ASP.NET Core для створення сторінок із HTML і серверною логікою.

Summoner’s Rift – основна карта League of Legends для матчів двох команд по п’ять гравців.

ADC – Attack Damage Carry, роль стрільця, що завдає стабільної шкоди з дистанції.

Champion – чемпіон, ігровий персонаж League of Legends з унікальними вміннями.

ВСТУП

Сучасні відеоігри перестали бути лише засобом дозвілля та перетворилися на складні цифрові середовища, у яких користувач взаємодіє з великим обсягом структурованої інформації. Особливо це помітно в командних онлайн-іграх, де результат залежить не тільки від індивідуальної реакції гравця, а й від розуміння ролей, властивостей персонажів, карти, предметів, взаємодії союзників і загроз з боку противників. У таких умовах важливого значення набувають допоміжні вебзастосунки, які можуть подавати ігрову інформацію у зрозумілій, стисло структурованій та зручній для користувача формі.

League of Legends є однією з найвідоміших командних ігор жанру MOBA. Її ігровий процес ґрунтується на протистоянні двох команд, кожна з яких складається з п’яти гравців. Кожен гравець керує окремим чемпіоном, що має власний набір умінь, характеристик, клас, роль і типові варіанти розвитку через предмети. Через велику кількість чемпіонів і різноманітність можливих комбінацій у матчах користувачеві складно швидко оцінити ситуацію, підібрати потрібного персонажа або зрозуміти, які загрози створює склад команди противника.

Існуючі інформаційні ресурси з League of Legends зазвичай орієнтовані на статистику перемог, рейтинги, професійні збірки предметів або великі масиви довідкових даних. Такі сервіси корисні для досвідчених гравців, однак для новачків вони можуть бути перевантаженими. Користувачеві часто потрібна не складна статистична таблиця, а зрозумілий маршрут: переглянути карту, визначити роль, знайти чемпіона, побачити його сильні й слабкі сторони та отримати базову рекомендацію щодо протистояння. Саме на розв’язання такої задачі спрямована ця кваліфікаційна робота.

Актуальність роботи зумовлена потребою у створенні навчально-аналітичного вебзастосунку, який поєднує довідкові та рекомендаційні функції в одному інтерфейсі. Для користувача важливо не лише отримати

перелік характеристик чемпіона, а й зрозуміти, як ці характеристики можуть впливати на конкретний матч. Тому в межах роботи розглядається не просто електронний довідник, а вебзастосунок із модулем алгоритмічного формування рекомендацій за складом команд.

Розроблюваний застосунок має назву «LoL Champion Guide & Recommender». Його функціональність охоплює головну сторінку з поясненням призначення системи, інтерактивну карту Summoner’s Rift, каталог чемпіонів із фільтрами, детальні картки характеристик і аналізатор матчу. Аналізатор приймає склад своєї команди та команди ворога, після чого формує рекомендації щодо предметів, визначає потенційні загрози та надає практичні поради. На відміну від систем, що потребують великих історичних даних, у цій роботі використовується прозорий правиловий підхід, який легко пояснити, перевірити та розширити.

Об’єктом роботи є процес інформаційної підтримки гравця під час ознайомлення з чемпіонами, ролями та базовими рекомендаціями щодо матч-апів у грі League of Legends.

Предметом роботи є структура вебзастосунку, модель зберігання даних про чемпіонів і правиловий алгоритм формування рекомендацій щодо предметів, загроз і поведінки гравця за обраним складом команд.

Метою роботи є розроблення вебзастосунку з функціями перегляду характеристик чемпіонів League of Legends та алгоритмічного формування рекомендацій на основі складу команд. Застосунок має бути реалізований як багатосторінковий вебпроєкт із серверною частиною, клієнтським інтерфейсом, локальною JSON-базою даних і зрозумілою логікою взаємодії користувача з основними функціями.

Для досягнення поставленої мети необхідно виконати такі завдання: проаналізувати предметну область гри League of Legends і визначити основні інформаційні потреби користувача; дослідити підходи до побудови довідкових і рекомендаційних вебзастосунків; сформувати функціональні вимоги до сторінок, навігації, фільтрації, інтерактивної карти та аналізатора

матчу; обґрунтувати вибір технологій ASP.NET Core Razor Pages, C#, HTML, CSS, JavaScript і JSON; спроєктувати модель даних чемпіона та структуру програмного проєкту; реалізувати алгоритм формування рекомендацій за класами ворогів, контрпіками й особливостями складу команди; виконати тестування основних сценаріїв роботи застосунку.

У роботі використано методи системного аналізу, порівняльного аналізу функціональних можливостей, об’єктно-орієнтованого проєктування, структурування даних, правилового формування рекомендацій, клієнт-серверної взаємодії та функціонального тестування. Для реалізації програмного продукту застосовано середовище Visual Studio 2022, платформу ASP.NET Core Razor Pages, мову програмування C#, засоби HTML, CSS і JavaScript, а також статичний JSON-файл як джерело даних про чемпіонів.

Актуальність роботи полягає у створенні програмної основи навчально-аналітичної платформи, яка може бути використана для швидкого ознайомлення з ролями, класами та характеристиками чемпіонів League of Legends. Застосунок допомагає користувачеві не тільки переглядати інформацію, а й отримувати пояснювані рекомендації щодо вибору предметів і поведінки в матчі. Завдяки використанню локальної бази даних і відокремленого сервісу логіки система може бути розширена новими чемпіонами, предметами, правилами та додатковими режимами аналізу.

Структура кваліфікаційної роботи відповідає поставленій меті та складається зі вступу, трьох розділів, висновків, переліку джерел посилання та додатків. У першому розділі аналізується предметна область і формулюється постановка задачі. У другому розділі розглядається проєктування вебзастосунку, структури даних, інтерфейсу та алгоритму рекомендацій. У третьому розділі описується програмна реалізація, тестування функціональних можливостей і напрями подальшого розвитку системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Особливості інформаційної підтримки гравця в League of Legends

Сучасні командні онлайн-ігри є складними цифровими середовищами, у яких користувач постійно працює з великою кількістю інформації. На відміну від простих ігрових застосунків, такі ігри потребують не лише швидкої реакції, а й уміння аналізувати ситуацію, оцінювати дії союзників і противників, порівнювати характеристики персонажів, орієнтуватися на карті та приймати рішення відповідно до зміни ігрового стану. У таких умовах гравцеві потрібна не випадкова сукупність довідкових даних, а зручна система інформаційної підтримки, яка допомагає швидко отримати потрібні відомості та застосувати їх у конкретній ситуації.

League of Legends є командною стратегічною грою жанру MOBA, у якій дві команди по п’ять гравців керують окремими чемпіонами та намагаються першими знищити Nexus противника. Основним полем для класичного режиму гри є Summoner’s Rift, що містить три основні лінії, лісові зони, оборонні споруди та нейтральні об’єкти. Рекомендований склад команди передбачає п’ять основних позицій: Top, Jungle, Mid, Bot та Support. Кожна позиція має власне призначення і передбачає використання чемпіонів із відповідними характеристиками та стилем гри [1].

Особливість League of Legends полягає в тому, що чемпіон не існує ізольовано. Його ефективність залежить від складу союзної команди, складу команди противника, обраної позиції, фази матчу, предметів і стилю гри користувача. Один і той самий чемпіон може бути сильним проти одних опонентів і вразливим проти інших. Наприклад, персонаж із високою дальністю умінь може мати перевагу проти повільних супротивників, але бути слабким проти мобільних ассасинів. Чемпіон із великою кількістю шкоди може швидко усувати слабкі цілі, проте потребувати правильної позиції та захисту від союзників.

Через це інформація про чемпіонів повинна подаватися не лише у вигляді статичного опису. Користувачеві важливо бачити роль чемпіона, його клас, рівень складності, рекомендовану лінію, уміння, предмети, сильні й слабкі сторони. Водночас корисною є можливість отримати пораду щодо конкретної ситуації, наприклад щодо вибору предметів проти певного класу ворогів або щодо обережнішої поведінки проти небезпечного опонента.

Рекомендаційний модуль у такому вебзастосунку можна розглядати як засіб підтримки прийняття рішень. Знаннєво-орієнтовані рекомендаційні системи використовують відомості про характеристики об’єктів, вимоги користувача та встановлені правила для визначення найбільш доречних варіантів із великої кількості альтернатив [2]. У межах розроблюваної системи такими знаннями є ролі й класи чемпіонів, їхні сильні та слабкі сторони, рекомендовані предмети, контрпіки та правила реагування на окремі типи загроз.

Для гравця-початківця найбільш важливими є базові пояснення: призначення карти, ролі, типи чемпіонів, рівень складності та загальні рекомендації. Для користувача з більшим досвідом цінніші стають порівняння складів команд, аналіз небезпечних класів противника, поради щодо предметів і попередження про ризики. Отже, вебзастосунок для цієї предметної області повинен поєднувати довідкову, навчальну та рекомендаційну функції.

У розроблюваному вебзастосунку інформаційна підтримка гравця реалізується через кілька взаємопов’язаних функціональних блоків. Головна сторінка виконує роль точки входу та пояснює основні можливості системи. Інтерактивна карта допомагає користувачеві візуально зрозуміти структуру Summoner’s Rift і призначення основних позицій. Каталог забезпечує пошук, фільтрацію та перегляд детальних карток чемпіонів, а аналізатор матчу формує рекомендації щодо предметів, потенційних загроз і поведінки гравця.

Офіційним джерелом статичних даних та графічних ресурсів League of Legends є Data Dragon, що містить структуровані відомості про чемпіонів,

предмети, руни, заклинання та відповідні зображення [3]. У початковій версії розроблюваного застосунку необхідні відомості зберігаються у локальному файлі champions.json, що спрощує запуск системи та дозволяє доповнювати дані без використання окремої системи керування базами даних.

Функціональну схему вебзастосунку для інформаційної підтримки гравця наведено на рисунку 1.1.

Рисунок 1.1 – Функціональна схема вебзастосунку для інформаційної підтримки гравця

Як видно з рисунка 1.1, вебзастосунок складається з кількох взаємопов’язаних модулів. Головна сторінка спрямовує користувача до основних розділів. Інтерактивна карта забезпечує ознайомлення з позиціями та об’єктами. Каталог чемпіонів надає доступ до характеристик персонажів. JSON-база є джерелом структурованих даних для карти, каталогу й аналізатора. Аналізатор матчу обробляє склад команд і повертає користувачеві результат у вигляді порад щодо предметів, загроз і поведінки.

Такий підхід дозволяє уникнути розрізненості інформації. Користувач не повинен окремо відкривати карту, окремо шукати чемпіона та окремо аналізувати предмети. У межах одного застосунку він може перейти від загального ознайомлення до конкретного вибору та отримати рекомендацію.

Це підвищує зручність роботи з ігровими даними та робить систему корисною як для навчання, так і для швидкої підготовки до умовного матч-апу.

Основні інформаційні потреби користувача вебзастосунку наведено в таблиці 1.1.

Таблиця 1.1 – Основні інформаційні потреби користувача вебзастосунку

Ситуація користувача	**Потрібна інформація**	**Функціональний блок застосунку**
Перше ознайомлення з грою	Пояснення ролей, карти, основних об’єктів	Головна сторінка та інтерактивна карта
Вибір чемпіона	Роль, клас, складність, короткий опис	Каталог чемпіонів із фільтрами
Перегляд характеристик	Уміння, переваги, недоліки, рекомендовані предмети	Детальна картка чемпіона
Підготовка до матчу	Склад союзників і ворогів, потенційні загрози	Аналізатор матчу
Отримання рекомендацій	Предмети, поради щодо поведінки, попередження про ризики	Рекомендаційний модуль

Таким чином, предметна область вимагає створення вебзастосунку, який не лише зберігає довідкову інформацію, а й допомагає користувачеві застосовувати її у практичній ігровій ситуації. Саме тому в роботі передбачено поєднання головної сторінки, інтерактивної карти, каталогу чемпіонів, структурованої бази даних і аналізатора матчу.

1.2 Аналіз існуючих підходів до довідкових і рекомендаційних ігрових сервісів

Для League of Legends існує багато інформаційних ресурсів, які допомагають гравцям отримувати відомості про чемпіонів, предмети, статистику матчів, популярні збірки та рейтинги. Такі сервіси виконують важливу роль, оскільки гра має велику кількість змінних: роль чемпіона, склад команди, склад противника, ігрова фаза, предмети, балансні зміни та рівень досвіду користувача. Проте більшість подібних сервісів орієнтована переважно на досвідчених гравців, які вже вміють самостійно інтерпретувати значний обсяг статистичних даних.

Типові довідкові сервіси містять сторінки чемпіонів, описи умінь, базові характеристики та списки рекомендованих предметів. Їхньою перевагою є повнота інформації. Користувач може знайти конкретного чемпіона, переглянути його здібності, побачити роль і дізнатися про популярні варіанти гри. Водночас такі сервіси не завжди пояснюють, чому певний предмет або стратегія є доцільними в конкретній ситуації. У результаті користувач отримує інформацію, але не завжди розуміє механізм її застосування.

Окремим напрямом розвитку ігрових інформаційних систем є формування рекомендацій щодо вибору персонажа. У роботі К. В. Блащик і Д. Шаєрмана розглянуто систему рекомендування чемпіонів League of Legends та виконано порівняння моделей машинного навчання, призначених для підтримки користувача на етапі вибору персонажа [4]. Це свідчить про можливість формалізації вибору чемпіона як рекомендаційної задачі, у якій результат залежить від характеристик персонажів і конкретних умов формування команди.

Інша група сервісів орієнтована на статистику. Вони аналізують велику кількість матчів і показують відсотки перемог, частоту вибору чемпіонів, популярні предмети, рейтинги ролей і тенденції поточного ігрового

середовища. Такі системи є корисними для досвідчених користувачів, однак мають кілька обмежень у контексті навчального застосунку. По-перше, для їх роботи потрібні великі масиви актуальних даних. По-друге, статистичний результат не завжди є пояснюваним. По-третє, дані можуть швидко змінюватися після оновлень гри, тому система потребує постійної підтримки.

Рекомендаційні ігрові сервіси можуть використовувати різні підходи до формування порад. Найпростішим є правиловий підхід, коли рекомендація створюється внаслідок перевірки певної умови. Наприклад, якщо у ворожій команді є ассасин, система радить обережну гру та захисні предмети. Якщо у ворогів багато танків, система пропонує предмети на пробивання захисту. Якщо у складі противника є чемпіони з лікуванням, доцільними стають предмети або ефекти проти відновлення здоров’я.

Перевагою правилового підходу є прозорість. Користувач може зрозуміти, чому система сформувала певну пораду. Для навчального вебзастосунку це особливо важливо, оскільки метою є не лише отримання результату, а й демонстрація логіки прийняття рішення. Крім того, правила можна реалізувати без зовнішніх статистичних сервісів і без навчання складної моделі.

Статистичний підхід до формування рекомендацій передбачає використання даних про реальні матчі, характеристики гравців, вибраних чемпіонів, отримані ресурси, виконані дії та підсумковий результат гри. На основі таких даних можна визначати фактори, що впливають на перемогу, оцінювати складність матч-апів і прогнозувати результат матчу. Зокрема, у дослідженні Хітар-Гарсії, Моран-Фернандес і Болон-Канедо запропоновано методи прогнозування результату професійних матчів League of Legends на основі інформації, доступної до початку гри [5].

Подальше ускладнення статистичного підходу пов’язане із застосуванням моделей машинного навчання. У роботі С. Чоудхурі, М. Ахсана та П. Барракло досліджено логістичну регресію, випадковий ліс, градієнтний бустинг і XGBoost для прогнозування команди-переможця за

передматчевими та внутрішньоігровими показниками [6]. Такі системи можуть враховувати значну кількість взаємопов’язаних факторів, однак потребують отримання великих наборів актуальних даних через Riot Games API, їх очищення, підготовки ознак, навчання моделей і подальшої перевірки точності.

Для початкової версії навчального вебзастосунку такий підхід є надмірним, оскільки метою роботи є створення зрозумілого й пояснюваного механізму формування базових рекомендацій, а не точне прогнозування результату професійного матчу. Крім того, статистичні закономірності можуть змінюватися після оновлень гри, що створює потребу в регулярному отриманні нових даних і повторному навчанні моделей.

У межах цієї роботи доцільним є використання правилового підходу. Він відповідає навчальній меті проєкту, легко реалізується мовою C#, не потребує зовнішньої бази статистики та дозволяє прозоро пояснити, чому система формує ту чи іншу пораду. Крім того, правилову модель можна поступово розширювати: додавати нові класи загроз, типи предметів, особливості чемпіонів, контрпіки та синергії.

Методологічно близькими до задачі алгоритмічного формування рекомендацій є дослідження, присвячені інтелектуальному опрацюванню структурованих і потокових даних. У роботі А. Ю. Шафроненко, Є. В. Бодянського та Д. О. Руденко запропоновано модифікований рекурентний метод достовірної нечіткої кластеризації, орієнтований на послідовне уточнення результатів оброблення даних [7]. Адаптивні методи онлайн-кластеризації та сегментації потоків даних розглянуто в роботах Є. В. Бодянського, А. Ю. Шафроненко, Д. О. Руденко та співавторів [8, 9]. Підходи до застосування гібридних інтелектуальних систем для аналізу динамічних даних досліджено І. С. Творошенко та В. О. Гороховатським [10].

Зазначені роботи не спрямовані безпосередньо на рекомендування чемпіонів у League of Legends, проте демонструють загальні принципи структурованого аналізу даних, виявлення прихованих залежностей і

формування результату на основі заданих моделей. У розроблюваному вебзастосунку використано простіший правиловий підхід: характеристики чемпіонів, класи загроз і відповідні рекомендації заздалегідь описуються у базі знань та програмній логіці. Такий підхід не потребує навчальної вибірки та дає змогу пояснити причину формування кожної рекомендації.

Порівняння основних підходів до формування рекомендацій наведено в таблиці 1.2.

Таблиця 1.2 – Порівняння підходів до формування рекомендацій

Підхід	**Переваги**	**Обмеження**	**Доцільність у роботі**
Правиловий підхід	Простота, прозорість, контрольованість	Потребує ручного оновлення правил	Основний підхід
Статистичний підхід	Може враховувати реальні дані матчів	Потребує великого набору актуальних даних	Можливий напрям розвитку
Машинне навчання	Може знаходити складні закономірності	Потребує навчання та валідації моделі	Надмірне для початкової версії
Гібридний підхід	Поєднує правила й дані	Має складнішу архітектуру	Перспективний напрям розвитку

Окрему увагу слід приділити способу зберігання даних. У великих інформаційних системах часто використовують серверні бази даних, які дають змогу зберігати користувачів, історію дій, статистику, коментарі та персональні налаштування. Однак для початкової версії навчального вебзастосунку використання повноцінної системи керування базами даних не є обов’язковим.

У цій роботі використовується статична JSON-база даних. Вона містить структуровані записи про чемпіонів, їхні ролі, класи, складність,

характеристики, уміння, предмети, сильні та слабкі сторони. Такий підхід має кілька переваг. По-перше, JSON легко редагувати та перевіряти. По-друге, дані можуть бути використані як серверною частиною, так і клієнтським JavaScript-кодом. По-третє, проєкт не потребує складного налаштування середовища й може бути швидко запущений у Visual Studio 2022.

Водночас JSON-база має обмеження. Якщо кількість даних значно зросте, підтримувати їх вручну буде складніше. Також статичний файл не забезпечує автоматичного оновлення після змін у грі. Проте для навчального вебзастосунку це не є критичним недоліком, оскільки головне завдання полягає не в промисловій точності статистики, а в демонстрації структури системи, логіки роботи з даними та принципу формування рекомендацій.

Розроблюваний вебзастосунок займає проміжне місце між простим довідником і складною статистичною платформою. З одного боку, він містить каталог чемпіонів і картки характеристик. З іншого боку, він має аналізатор матчу, який враховує склад команд і повертає рекомендації. Така комбінація робить застосунок корисним як для навчання, так і для демонстрації основ веброзробки, роботи з даними та алгоритмічного мислення.

Порівняння розроблюваного вебзастосунку з типовими інформаційними сервісами наведено в таблиці 1.3.

Таблиця 1.3 – Порівняння розроблюваної системи з типовими інформаційними сервісами

Критерій	**Типові статистичні сервіси**	**Розроблюваний вебзастосунок**
Основний акцент	Статистика, рейтинги, популярні збірки	Навчальна структура, карта, пояснювані рекомендації
Складність для новачка	Висока через велику кількість показників	Нижча завдяки карткам, фільтрам і коротким поясненням

Продовження таблиці 1.3

Критерій	**Типові статистичні сервіси**	**Розроблюваний вебзастосунок**
Джерело рекомендацій	Статистика або закрита логіка	Прозорі правила в програмному коді
Зберігання даних	Зовнішня база даних або API	Локальний JSON-файл
Мета використання	Оптимізація ігрових показників	Навчання, довідка та базова підтримка вибору

Отже, аналіз існуючих підходів показує, що для поставленої задачі найбільш доцільним є створення вебзастосунку з локальною структурованою базою даних і правиловим алгоритмом рекомендацій. Такий підхід дозволяє реалізувати функціональний програмний продукт у межах бакалаврської роботи та водночас залишити можливість для подальшого розвитку.

1.3 Вимоги до вебзастосунку та користувацьких сценаріїв

Формування вимог до вебзастосунку є важливим етапом перед його проєктуванням і реалізацією. Вимоги визначають, які функції повинна виконувати система, як користувач буде взаємодіяти з інтерфейсом, які дані потрібно зберігати та які результати має повертати програма. Для розроблюваного вебзастосунку вимоги були сформовані з урахуванням предметної області League of Legends, потреб користувача та обмежень навчального проєкту.

Першою вимогою є наявність зрозумілої головної сторінки. Користувач, який відкриває застосунок, повинен одразу зрозуміти його призначення. Головна сторінка має містити короткий опис системи, акцент на основних можливостях і кнопки переходу до ключових розділів: карти,

каталогу чемпіонів та аналізатора матчу. Такий підхід зменшує час ознайомлення й робить застосунок зручнішим для нових користувачів.

Другою вимогою є інтерактивна карта Summoner’s Rift. Вона повинна допомагати користувачеві зрозуміти просторову організацію гри. На карту доцільно додати інтерактивні точки, що відповідають основним лініям, лісу, нейтральним об’єктам і базовим цілям. Після натискання на точку користувач має отримувати коротке пояснення ролі відповідної області та рекомендації щодо типових чемпіонів або ігрового значення об’єкта.

Третьою вимогою є каталог чемпіонів. Він має забезпечувати перегляд списку персонажів, пошук за іменем і фільтрацію за основними параметрами. До таких параметрів належать роль, клас і складність. Картка чемпіона повинна містити іконку, ім’я, роль, клас і коротку оцінку складності. Детальна інформація має відкриватися окремо, щоб не перевантажувати основну сітку карток.

Четвертою вимогою є наявність детальної картки чемпіона. У ній потрібно відображати характеристики, уміння, рекомендовані предмети, сильні та слабкі сторони. Така структура дозволяє користувачеві не просто знайти персонажа, а й оцінити його можливості. У межах навчального застосунку ці дані зберігаються у JSON-файлі та виводяться через клієнтський інтерфейс.

П’ятою вимогою є аналізатор матчу. Це ключовий модуль, який реалізує алгоритмічне формування рекомендацій. Користувач повинен мати змогу вибрати чемпіонів для своєї команди та команди ворога. Після запуску аналізу система має порівняти склади, визначити основні загрози, підібрати рекомендації щодо предметів і сформувати короткі поради щодо поведінки в матчі.

Шостою вимогою є адаптивність користувацького інтерфейсу. Адаптивний вебдизайн передбачає створення гнучкої структури сторінки, яка коректно відображається на пристроях із різними розмірами та роздільною здатністю екрана. Для цього можуть використовуватися гнучкі сітки,

технології CSS Grid і Flexbox, медіазапити, адаптивні зображення та налаштування області перегляду [11]. У розроблюваному вебзастосунку картки чемпіонів, інтерактивна карта, навігація та блоки аналізатора повинні автоматично змінювати своє розташування залежно від ширини екрана.

Сьомою вимогою є збереження стану користувача в аналізаторі матчу. Для цього використовується механізм localStorage, який дає змогу зберігати дані в браузері між окремими сеансами роботи [12]. У localStorage доцільно записувати лише ідентифікатори чемпіонів, вибраних для позицій союзної та ворожої команд. Після повторного відкриття або оновлення сторінки ці ідентифікатори використовуються для відновлення попередньо сформованого складу.

Під час проєктування інтерфейсу необхідно також враховувати вимоги вебдоступності. Інтерактивні компоненти повинні мати достатній контраст, зрозумілі текстові позначення, помітний стан фокуса та достатній розмір області натискання. Рекомендації WCAG 2.2 спрямовані на підвищення доступності вебвмісту для користувачів із різними порушеннями та водночас сприяють покращенню загальної зручності використання вебсистем [13].

Функціональні вимоги до основних сторінок вебзастосунку наведено в таблиці 1.4.

Таблиця 1.4 – Функціональні вимоги до основних сторінок вебзастосунку

Сторінка	**Основні функції**	**Очікуваний результат**
Головна сторінка	Опис призначення, кнопки переходу, блоки можливостей	Користувач розуміє структуру застосунку
Карта	Інтерактивні точки, інформаційна панель, рекомендації за позиціями	Користувач отримує пояснення ролей і об’єктів карти

Продовження таблиці 1.4

Сторінка	**Основні функції**	**Очікуваний результат**
Чемпіони	Пошук, фільтрація, картки, модальне вікно	Користувач знаходить і переглядає чемпіона
Аналізатор	Вибір складів команд, запуск аналізу, збереження вибору	Користувач отримує рекомендації за матчем
Спільна навігація	Бургер-меню, переходи між сторінками	Користувач швидко переміщується між розділами

Користувацький сценарій починається з головної сторінки. Користувач ознайомлюється з коротким описом вебзастосунку та переходить до одного з розділів. Якщо користувач не знайомий із картою, він може відкрити розділ карти, натиснути на позначку лінії або нейтрального об’єкта й отримати пояснення. Якщо користувач хоче знайти конкретного чемпіона, він переходить до каталогу, використовує пошук або фільтри та відкриває детальну картку.

Інший сценарій пов’язаний із підготовкою до умовного матчу. Користувач відкриває аналізатор, вибирає чемпіонів для своєї команди та команди противника. Після цього система аналізує введені дані та формує результат. Результат має бути не суцільним текстом, а структурованою відповіддю: блок загроз, блок рекомендованих предметів, блок переваг і блок порад. Така структура полегшує сприйняття інформації.

Узагальнений сценарій переходу користувача між основними функціональними блоками вебзастосунку наведено на рисунку 1.2.

Рисунок 1.2 – Узагальнений користувацький сценарій роботи із вебзастосунком

На рисунку 1.2 показано послідовність взаємодії користувача із системою: вхід на сайт, перегляд карти, робота з каталогом, вибір складів команд у Team Builder, збереження стану та отримання результату аналізу. Окремо показано роль localStorage, який зберігає вибір користувача між сеансами. Це дозволяє зробити роботу з аналізатором зручнішою, оскільки після оновлення сторінки раніше вибрані чемпіони не втрачаються.

Для реалізації інтерфейсу важливо врахувати принципи зручності. Текст має бути читабельним на темному фоні, інтерактивні елементи – помітними, кнопки – зрозумілими, а модальні вікна – зручними для закриття. Візуальна стилістика може відповідати ігровій тематиці, але вона не повинна заважати сприйняттю інформації. Основна мета дизайну полягає не лише у створенні привабливого вигляду, а й у підтримці функціональності.

Нефункціональні вимоги також мають важливе значення. Застосунок повинен запускатися у середовищі Visual Studio 2022, працювати як ASP.NET Core-проєкт, коректно завантажувати статичні ресурси з папки wwwroot і не вимагати складного налаштування зовнішньої бази даних. Це особливо важливо для перевірки роботи, оскільки програмний продукт має бути зрозумілим і відтворюваним.

До обмежень початкової версії належить відсутність авторизації, персональних профілів, реальної статистики матчів і автоматичного оновлення балансу гри. Ці обмеження не знижують цінність роботи, оскільки мета полягає у створенні навчально-аналітичної платформи з базовим алгоритмом рекомендацій. У майбутньому ці функції можуть бути додані як напрями розвитку.

Критерії якості початкової версії вебзастосунку наведено в таблиці 1.5.

Таблиця 1.5 – Критерії якості початкової версії вебзастосунку

Критерій	**Зміст критерію**	**Спосіб забезпечення**
Зрозумілість	Користувач розуміє призначення розділів	Головна сторінка, короткі описи, навігація
Цілісність	Усі модулі використовують спільні дані	JSON-база та серверний сервіс
Адаптивність	Інтерфейс коректно працює на різних екранах	CSS-сітки, медіазапити, бургер-меню
Пояснюваність	Рекомендації мають зрозумілу причину	Правиловий алгоритм
Розширюваність	Можна додавати чемпіонів і правила	Структурований JSON і окремий сервіс логіки

Отже, вимоги до вебзастосунку охоплюють як функціональні можливості, так і якість користувацького досвіду. Система повинна забезпечувати перегляд інформації про чемпіонів, роботу з інтерактивною картою, аналіз складів команд і формування рекомендацій. Саме ці вимоги визначають подальше проєктування архітектури, структури даних та програмної реалізації.

1.4 Постановка задачі дослідження та розроблення

На основі аналізу предметної області та вимог до системи ставиться задача розробити вебзастосунок для перегляду характеристик чемпіонів і алгоритмічного формування рекомендацій у грі League of Legends. Застосунок має поєднувати довідкову інформацію, інтерактивні елементи та модуль аналізу складів команд. Його призначення полягає у спрощенні роботи користувача з ігровими даними та наданні базових порад щодо вибору предметів і поведінки в матчі.

Об’єктом роботи є процес інформаційної підтримки гравця під час ознайомлення з чемпіонами, ролями, картою та базовими ігровими ситуаціями в League of Legends.

Предметом роботи є методи організації вебзастосунку, структура зберігання даних про чемпіонів і правиловий алгоритм формування рекомендацій щодо предметів, загроз і поведінки гравця на основі складу команд.

Метою роботи є розроблення багатосторінкового вебзастосунку «LoL Champion Guide & Recommender» з функціями перегляду характеристик чемпіонів, інтерактивною картою Summoner’s Rift, каталогом персонажів і аналізатором матчів, який формує рекомендації за допомогою правилового алгоритму.

Для досягнення поставленої мети необхідно розв’язати такі завдання:

– проаналізувати предметну область гри League of Legends та визначити основні інформаційні потреби користувача;

– дослідити підходи до побудови довідкових і рекомендаційних ігрових сервісів;

– сформувати вимоги до вебзастосунку, його сторінок, навігації, адаптивності та інтерфейсу;

– обґрунтувати вибір технологічного стеку для реалізації програмного продукту;

– спроєктувати структуру даних чемпіона, включаючи роль, клас, складність, характеристики, уміння, предмети, переваги та недоліки;

– розробити інтерактивну карту з точками основних ліній, нейтральних об’єктів і пояснювальною інформаційною панеллю;

– створити каталог чемпіонів із пошуком, фільтрами та детальними картками;

– реалізувати аналізатор матчу для вибору складів команд і формування рекомендацій;

– передбачити збереження вибраного складу команд у браузері;

– виконати тестування основних сценаріїв роботи вебзастосунку;

– визначити можливості подальшого розширення програмного продукту.

Очікуваним результатом роботи є вебзастосунок, який можна запустити як ASP.NET Core Razor Pages-проєкт у середовищі Visual Studio 2022. Застосунок повинен містити головну сторінку, інтерактивну карту, каталог чемпіонів і аналізатор матчів. Дані про чемпіонів мають зберігатися у статичному JSON-файлі, а логіка рекомендацій – реалізовуватися на основі правил у серверній частині.

З технічної точки зору система повинна складатися з клієнтської та серверної частин. Клієнтська частина відповідає за відображення сторінок, роботу фільтрів, модальних вікон, інтерактивної карти та виведення результату аналізу. Серверна частина відповідає за завантаження даних, пошук чемпіонів і формування рекомендацій. Такий поділ дозволяє зробити програму зрозумілою, підтримуваною та придатною для подальшого розвитку.

Практична цінність розроблення полягає у створенні навчально-аналітичної платформи, яка може бути використана для швидкого ознайомлення з чемпіонами League of Legends і базового аналізу матч-апів. Користувач отримує можливість не лише переглядати характеристики

персонажів, а й бачити, які загрози може створювати команда противника та які предмети або поведінкові рішення варто розглянути.

Науково-практична значущість роботи полягає у поєднанні методів веброзробки, структурування даних і правилового формування рекомендацій у межах єдиного програмного продукту. Розроблюваний застосунок демонструє, як довідкові дані можуть бути використані не лише для відображення, а й для прийняття простих алгоритмічних рішень.

Правиловий модуль розроблюваного вебзастосунку можна розглядати як спрощений засіб підтримки прийняття рішень. У технологіях прийняття рішень в інформаційних системах важливими є формування множини можливих альтернатив, визначення критеріїв їх оцінювання та вибір результату відповідно до наявної інформації [14]. У розроблюваній системі альтернативами є можливі предмети й варіанти поведінки, критеріями – класи та властивості чемпіонів противника, а результатом – сформований набір рекомендацій. На відміну від складних багатокритеріальних моделей, у початковій версії застосунку вибір виконується за допомогою наперед визначених програмних правил.

Важливою вимогою до рекомендаційного модуля є пояснюваність сформованого результату. У знаннєво-орієнтованих рекомендаційних системах вибір варіанта здійснюється на основі явно визначених відомостей про об’єкти, встановлених обмежень і правил прийняття рішень [2]. Тому кожна рекомендація розроблюваного вебзастосунку повинна супроводжуватися коротким обґрунтуванням, наприклад указівкою на наявність у команді противника танка, ассасина, чемпіона з високим рівнем лікування або персонажа, який є контрпіком для вибраного чемпіона.

Критеріями успішного розв’язання поставленої задачі є коректне завантаження структурованих даних, стабільна робота основних сторінок, правильне застосування фільтрів, збереження вибраного складу команд і формування відтворюваних рекомендацій. За однакових вхідних даних правиловий алгоритм повинен повертати однаковий результат, що спрощує

його тестування та перевірку. Така контрольованість відповідає принципам підтримки прийняття рішень, за яких результат визначається на основі заданих критеріїв, альтернатив і формалізованих правил [14].

Початкова версія системи має певні обмеження. Вона не підключається до реальної статистики матчів, не враховує персональний рівень користувача, не виконує авторизацію та не оновлює дані автоматично. Проте ці обмеження є прийнятними для бакалаврської роботи, оскільки головна мета полягає у створенні функціональної основи, яка може бути розширена в майбутньому.

Подальший розвиток проєкту може передбачати підключення офіційного Riot Games API для отримання динамічних даних про профілі користувачів, історію та результати матчів, статистичні показники й склади команд [15]. На відміну від статичного Data Dragon, використання API дасть змогу доповнити систему актуальними відомостями та створити статистичний модуль рекомендацій. Водночас така інтеграція потребуватиме реєстрації програмного продукту, використання API-ключа, контролю обмежень кількості запитів і оброблення можливих помилок отримання даних. Іншими напрямами розвитку є розширення бази чемпіонів і предметів, додавання вагових коефіцієнтів загроз, збереження історії аналізів та реалізація персональних налаштувань користувача. Таким чином, у першому розділі було розглянуто предметну область League of Legends, визначено потреби користувача, проаналізовано підходи до побудови довідкових і рекомендаційних сервісів, сформовано вимоги до вебзастосунку та поставлено задачу розроблення. Подальший розділ присвячено проєктуванню архітектури системи, структури даних, користувацького інтерфейсу та алгоритму формування рекомендацій.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ТА АЛГОРИТМУ ФОРМУВАННЯ РЕКОМЕНДАЦІЙ

2.1 Архітектура вебзастосунку та вибір технологічних засобів

Проєктування вебзастосунку є одним із ключових етапів розроблення програмного продукту, оскільки саме на цьому етапі визначається загальна структура системи, спосіб організації даних, взаємодія між клієнтською та серверною частинами, а також принципи подальшого розширення. Для теми кваліфікаційної роботи важливо було створити не окрему статичну сторінку, а повноцінний багатосторінковий вебзастосунок, який поєднує довідкову інформацію про чемпіонів League of Legends, інтерактивну карту Summoner’s Rift та модуль алгоритмічного формування рекомендацій.

Розроблюваний вебзастосунок «LoL Champion Guide & Recommender» спроєктовано як навчально-аналітичну систему, що працює за принципом клієнт-серверної взаємодії. Клієнтська частина відповідає за відображення сторінок, оброблення дій користувача, фільтрацію елементів інтерфейсу, відкриття модальних вікон, роботу інтерактивної карти та виведення результатів аналізу. Серверна частина відповідає за завантаження структурованих даних, пошук чемпіонів, оброблення запитів і формування рекомендацій на основі складу команд.

Загальна архітектура системи побудована за модульним принципом. Кожна функціональна частина вебзастосунку має власне призначення та взаємодіє з іншими частинами через спільну модель даних. Головна сторінка виконує інформаційно-навігаційну функцію, сторінка карти забезпечує візуальне ознайомлення з ігровою картою, каталог чемпіонів дозволяє переглядати та фільтрувати персонажів, а аналізатор матчу реалізує рекомендаційну логіку.

Для реалізації серверної частини обрано платформу ASP.NET Core Razor Pages. Razor Pages є сторінково орієнтованою моделлю створення

серверних вебінтерфейсів, у якій кожна сторінка представлена файлом розмітки та пов’язаною з ним моделлю оброблення запитів [16]. Така організація підходить для розроблюваного багатосторінкового вебзастосунку, оскільки його функціональні можливості природно поділяються між окремими сторінками: Index відповідає за головний екран, Map – за інтерактивну карту, Champions – за каталог персонажів, а Analyzer – за аналіз складів команд.

Спільні елементи інтерфейсу, зокрема навігаційне меню, підключення таблиць стилів, сценаріїв і шрифтів, винесено до макета _Layout.cshtml. Це зменшує дублювання розмітки та забезпечує однаковий вигляд усіх сторінок. Серверна частина мовою C# відповідає за роботу з моделями предметної області, завантаження даних про чемпіонів, оброблення запитів і формування результатів аналізу.

Статичні ресурси вебзастосунку розміщено в каталозі wwwroot. До них належать CSS-стилі, JavaScript-сценарії, зображення, файл champions.json та інші ресурси, які безпосередньо передаються браузеру. ASP.NET Core передбачає окремий механізм обслуговування статичних ресурсів, що дає змогу надавати клієнтові файли, які не створюються динамічно під час оброблення запиту [17].

Для відокремлення сторінок і маршрутів від логіки роботи з даними використовується сервіс GameDataService. Його реєстрація здійснюється через вбудований у ASP.NET Core механізм впровадження залежностей. Такий механізм забезпечує передавання готового екземпляра сервісу компонентам, які його потребують, та зменшує прямі залежності між класами [18]. Завдяки цьому спосіб отримання даних у майбутньому можна змінити без повного переписування сторінок і клієнтської частини.

Мова програмування C# використовується для реалізації серверної логіки. Вона дозволяє описати моделі даних, створити сервіс для роботи з JSON-файлом, обробляти запити та формувати відповідь для клієнтської частини. Перевагою C# є строга типізація, що зменшує кількість помилок під

час роботи з об’єктами предметної області. Наприклад, чемпіон, його характеристики, уміння, предмети та списки переваг можуть бути представлені як окремі класи з чітко визначеними властивостями.

Для клієнтської частини використано HTML, CSS і JavaScript. HTML відповідає за структуру сторінок, CSS – за зовнішній вигляд, адаптивність, кольорову палітру, розміщення блоків та анімації, а JavaScript – за динамічну поведінку інтерфейсу. До такої поведінки належать відкриття бургер-меню, оброблення кліків по карті, завантаження списку чемпіонів, фільтрація карток, відкриття модального вікна та надсилання даних аналізатора на сервер.

Особливістю обраної архітектури є використання статичної JSON-бази даних. Дані про чемпіонів зберігаються у файлі champions.json, розміщеному в папці wwwroot/data. Такий підхід є доцільним для початкової версії застосунку, оскільки не потребує окремої системи керування базами даних, спрощує запуск проєкту та дозволяє легко редагувати записи. JSON-файл може містити як прості текстові поля, так і вкладені об’єкти та масиви, що робить його зручним для опису складної структури чемпіона.

Обрана архітектура передбачає чітке розділення рівнів системи. Рівень представлення включає Razor-сторінки, HTML-розмітку та CSS-стилі. Рівень клієнтської логіки включає JavaScript-файли, що відповідають за інтерактивність. Рівень серверної логіки представлений сервісом роботи з даними та API-маршрутами. Рівень даних представлений статичним JSON-файлом і зображеннями, що використовуються в інтерфейсі.

Архітектуру вебзастосунку на основі ASP.NET Core Razor Pages наведено на рисунку 2.1.

На рисунку 2.1 показано, що клієнтський рівень взаємодіє зі сторінками Razor Pages, а сторінки, у свою чергу, використовують сервісний рівень і моделі C#. Сервісний рівень отримує дані зі статичних джерел, зокрема з champions.json і ресурсів Data Dragon. Такий підхід дозволяє

відокремити інтерфейс від логіки роботи з даними та зробити систему більш зрозумілою для подальшого супроводу.

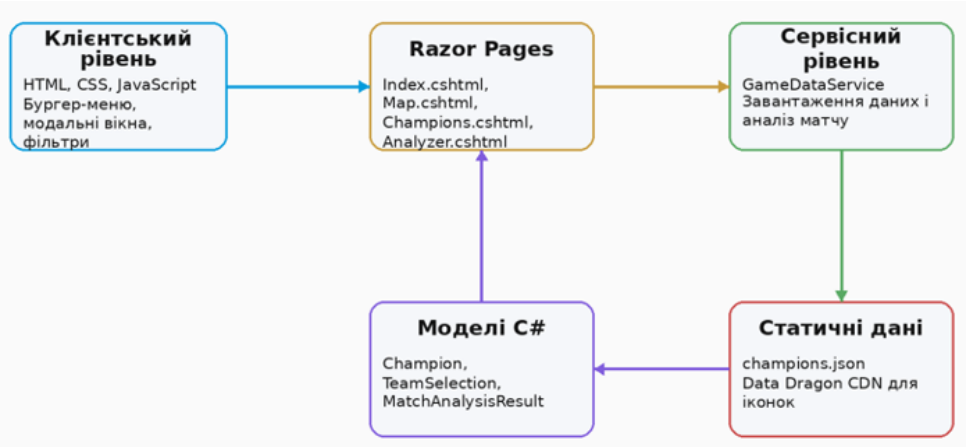

Рисунок 2.1 – Архітектура вебзастосунку на основі ASP.NET Core Razor Pages

Архітектурні рівні вебзастосунку наведено в таблиці 2.1.

Таблиця 2.1 – Архітектурні рівні вебзастосунку

Рівень системи	**Основні елементи**	**Призначення**
Рівень представлення	Razor Pages, HTML, CSS	Відображення сторінок і візуальних компонентів
Рівень клієнтської логіки	JavaScript-файли	Оброблення дій користувача та динамічне оновлення інтерфейсу
Рівень серверної логіки	C#-сервіси, API-маршрути	Завантаження даних, пошук, аналіз і формування рекомендацій
Рівень даних	champions.json, статичні зображення	Зберігання інформації про чемпіонів, предмети та ресурси
Рівень збереження стану	localStorage	Збереження вибраного складу команд у браузері

Для розроблення застосунку обрано середовище Visual Studio 2022. Воно забезпечує зручне створення ASP.NET Core-проєктів, роботу з C#-кодом, запуск локального сервера, налагодження помилок, підсвічування синтаксису та інтеграцію з файловою структурою проєкту. Visual Studio 2022 є зручним середовищем для навчальної розробки, оскільки дозволяє швидко перевіряти зміни та запускати вебзастосунок без додаткового складного налаштування.

Для візуального оформлення застосунку було обрано темну ігрову стилістику. Основними кольорами інтерфейсу є темні сині й чорні відтінки, золоті акценти та світлові ефекти. Такий стиль відповідає атмосфері League of Legends і водночас допомагає виділити важливі інтерактивні елементи. Картки, кнопки, модальні вікна та маркери карти повинні бути достатньо контрастними, щоб користувач міг легко сприймати інформацію.

Перевагою обраного технологічного стеку є його простота та достатня функціональність. ASP.NET Core Razor Pages дозволяє створити серверну частину мовою C#, JavaScript забезпечує інтерактивність, CSS відповідає за адаптивний дизайн, а JSON-файл виконує роль легкої бази даних. Усі ці технології добре поєднуються між собою та дають змогу реалізувати поставлені функціональні вимоги.

Вибір технологічного стеку наведено в таблиці 2.2.

Таблиця 2.2 – Обґрунтування вибору технологічних засобів

Компонент	**Обраний засіб**	**Причина вибору**
Серверна частина	ASP.NET Core Razor Pages	Підтримка C#, проста структура сторінок, можливість створення API
Мова серверної логіки	C#	Строга типізація, зручна робота з моделями та сервісами
Клієнтська структура	HTML	Формування розмітки сторінок

Продовження таблиці 2.2

Компонент	**Обраний засіб**	**Причина вибору**
Візуальне оформлення	CSS	Темна палітра, адаптивність, стилізація карток і меню
Клієнтська інтерактивність	JavaScript	Фільтрація, модальні вікна, карта, запити до API
Дані	JSON	Просте редагування, структурованість, відсутність потреби в СКБД
Середовище розробки	Visual Studio 2022	Зручний запуск і налагодження ASP.NET Core-проєкту
Збереження стану	localStorage	Збереження вибору команд без серверної авторизації

Важливою перевагою архітектури є її розширюваність. Якщо в майбутньому виникне потреба додати нових чемпіонів, достатньо розширити JSON-файл. Якщо потрібно змінити логіку рекомендацій, зміни вносяться до серверного сервісу. Якщо потрібно додати нову сторінку, її можна створити в папці Pages і підключити до спільної навігації. Такий підхід відповідає принципу розділення відповідальностей, коли кожна частина системи виконує власну функцію.

Недоліком використання статичного JSON-файлу є те, що система не отримує автоматичні оновлення балансу гри та не зберігає персональні дані користувача на сервері. Проте для початкової версії це обмеження є

прийнятним. Основна мета роботи полягає у створенні функціональної основи вебзастосунку, демонстрації роботи з даними та реалізації алгоритмічного формування рекомендацій, а не в побудові промислової статистичної платформи.

Таким чином, для реалізації вебзастосунку обрано архітектуру ASP.NET Core Razor Pages із використанням C#, HTML, CSS, JavaScript і JSON. Такий підхід забезпечує достатню функціональність, простоту запуску, зрозумілу структуру проєкту та можливість подальшого розвитку системи.

2.2 Проєктування структури даних і програмних модулів

Структура даних є основою вебзастосунку, оскільки саме від неї залежить коректність роботи каталогу чемпіонів, детальних карток, інтерактивної карти та аналізатора матчів. У межах розроблюваної системи необхідно зберігати різні типи інформації: загальні відомості про чемпіона, його роль, клас, рівень складності, характеристики, уміння, предмети, сильні й слабкі сторони, контрпіки та синергії. Уся ця інформація повинна мати таку структуру, щоб її можна було легко використовувати як на сервері, так і на клієнті.

Файлова структура програмного проєкту організована відповідно до типової логіки ASP.NET Core. У корені проєкту розміщується файл Program.cs, який відповідає за налаштування вебзастосунку, підключення статичних файлів, реєстрацію сервісів і створення маршрутів. Папка Models містить класи предметної області, папка Services – серверний сервіс роботи з даними, папка Pages – Razor-сторінки, а папка wwwroot – статичні ресурси, зокрема CSS, JavaScript, JSON-файл і зображення.

Файлову структуру програмного проєкту у середовищі Visual Studio 2022 наведено на рисунку 2.2.

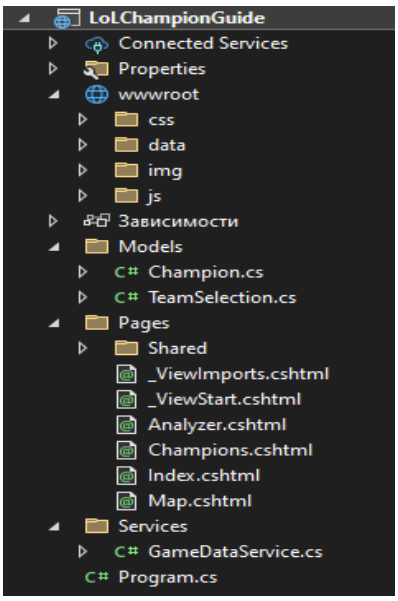

Рисунок 2.2 – Файлова структура програмного проєкту у Visual Studio 2022

Як видно з рисунка 2.2, структура проєкту поділена на логічні групи. Папка Models містить моделі Champion.cs і TeamSelection.cs, які описують основні дані предметної області. Папка Pages містить сторінки Index.cshtml, Map.cshtml, Champions.cshtml та Analyzer.cshtml, що відповідають основним розділам вебзастосунку. Папка Services містить GameDataService.cs, який відповідає за завантаження даних і формування рекомендацій. Папка wwwroot містить клієнтські ресурси: стилі, скрипти, зображення й JSON-базу.

Призначення основних файлів і папок проєкту наведено в таблиці 2.3.

Таблиця 2.3 – Призначення основних файлів і папок проєкту

Файл або папка	**Призначення**
Program.cs	Налаштування застосунку, маршрути, підключення сервісів
Models/Champion.cs	Опис структури чемпіона
Models/TeamSelection.cs	Модель вибору складів команд
Services/GameDataService.cs	Завантаження даних і формування рекомендацій

Продовження таблиці 2.3

Файл або папка	**Призначення**
Pages/Index.cshtml	Головна сторінка
Pages/Map.cshtml	Сторінка інтерактивної карти
Pages/Champions.cshtml	Каталог чемпіонів
Pages/Analyzer.cshtml	Аналізатор матчу
Pages/Shared/_Layout.cshtml	Спільний макет і навігація
wwwroot/css/site.css	Глобальні стилі вебзастосунку
wwwroot/js/site.js	Загальна клієнтська логіка
wwwroot/js/map.js	Логіка інтерактивної карти
wwwroot/js/champions.js	Логіка каталогу чемпіонів
wwwroot/js/analyzer.js	Логіка аналізатора матчу
wwwroot/data/champions.json	Статична база даних чемпіонів
wwwroot/img	Зображення, що використовуються в інтерфейсі

Для початкової версії системи обрано зберігання даних у файлі champions.json. Формат JSON дозволяє подавати інформацію у вигляді об’єктів, масивів, рядків, числових і логічних значень, а також створювати вкладені структури. Для перетворення JSON-тексту на типізовані об’єкти C# використовується простір імен System.Text.Json, який підтримує серіалізацію об’єктів у JSON і зворотну десеріалізацію отриманих даних [19]. Завдяки цьому серверна частина може завантажувати записи з файлу та працювати з ними як із колекцією об’єктів Champion.

Кожен запис про чемпіона містить ідентифікатор, ім’я, роль, клас, рівень складності, опис, шлях до зображення, характеристики, уміння, предмети, сильні та слабкі сторони, контрпіки й синергії. Єдина структура запису використовується в різних функціональних модулях. Каталог отримує ім’я, зображення, роль, клас і складність; детальна картка використовує

характеристики, уміння та предмети; аналізатор працює з класами, контрпіками, перевагами й недоліками чемпіонів.

Загальний принцип подання складного об’єкта як множини взаємопов’язаних компонентів та ознак застосовується в інтелектуальних системах аналізу даних. У монографії В. О. Гороховатського та І. С. Творошенко розглянуто структурні технології аналізу багатовимірних даних і моделі їх подання у вигляді множини компонентів [20]. Підходи до кластерного подання структурованих описів і пошуку об’єктів за сукупністю ознак досліджено В. О. Гороховатським, І. С. Творошенко, О. А. Кобиліним та співавторами [21].

Методи статистичного перетворення структурованих описів та визначення релевантності їхніх компонентів наведено в роботі Ю. І. Дарадке, В. О. Гороховатського, І. С. Творошенко та співавторів [22]. Перетворення первинного опису об’єкта на компактнішу систему класифікаційних ознак розглянуто В. О. Гороховатським, І. С. Творошенко та О. В. Яковлевою [23]. Зазначені праці стосуються переважно аналізу зображень, однак методологічно підтверджують доцільність структурованого подання об’єкта через набір формалізованих характеристик.

У розроблюваному вебзастосунку цей принцип застосовується до предметної області League of Legends. Чемпіон розглядається як структурований об’єкт, властивості якого можуть бути використані для відображення інформації, фільтрації, порівняння та формування правилових рекомендацій. На відміну від складних моделей класифікації, у цій роботі не виконується автоматичне виділення ознак: необхідні характеристики визначено заздалегідь і явно збережено у JSON-базі.

Наприклад, для каталогу достатньо відобразити ім’я, іконку, роль, клас і складність. Для детальної картки потрібні характеристики, уміння, предмети, сильні та слабкі сторони. Для аналізатора матчу важливими є роль, клас, предмети, контрпіки та загальні властивості чемпіона. Завдяки цьому

дані не дублюються, а різні сторінки застосунку працюють зі спільним джерелом інформації.

Інформаційну модель чемпіона та пов’язаних із ним сутностей наведено на рисунку 2.3.

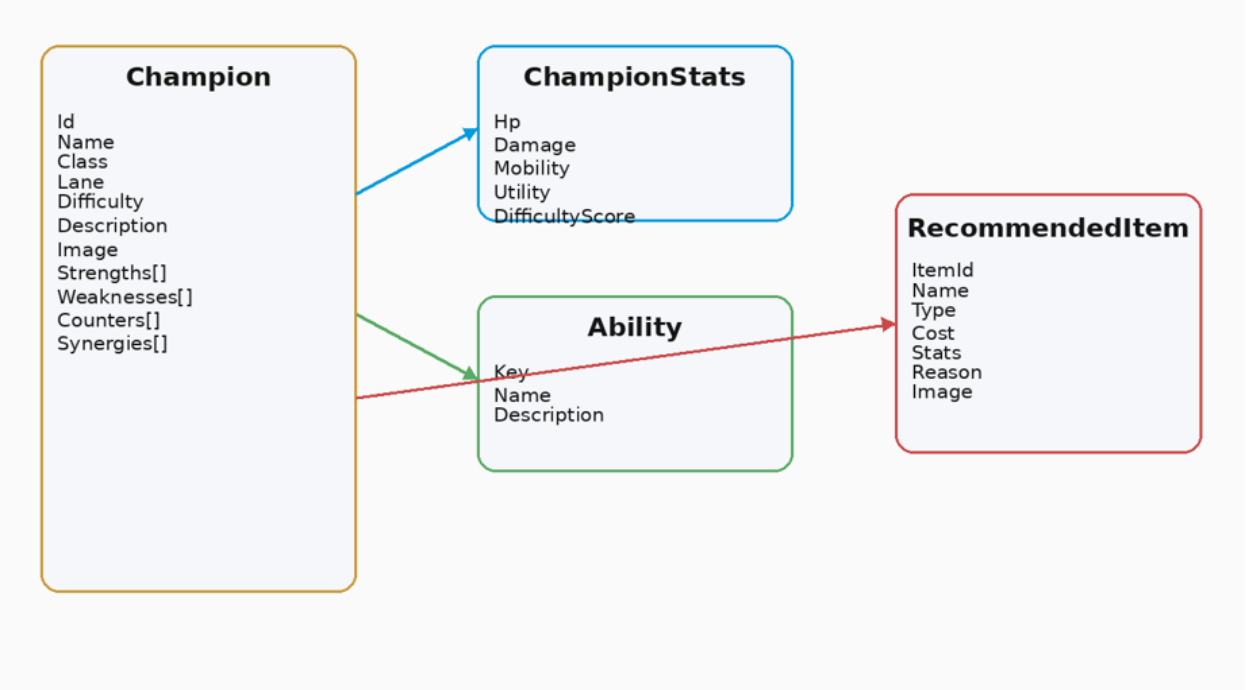

Рисунок 2.3 – Інформаційна модель чемпіона та пов’язаних сутностей

На рисунку 2.3 показано, що основною сутністю є Champion. Вона пов’язана з ChampionStats, Ability та RecommendedItem. Сутність ChampionStats описує числові характеристики персонажа, Ability містить дані про вміння, а RecommendedItem описує предмети, які можуть бути рекомендовані для певного чемпіона. Така модель дозволяє зберігати дані структуровано та використовувати їх у різних частинах застосунку.

Інформаційна модель чемпіона повинна бути достатньо гнучкою. У майбутньому до неї можна додати тип шкоди, рівень мобільності, наявність контролю, фазу сили чемпіона, рекомендовані руни або додаткові пояснення для новачків. Тому структура JSON має бути не випадковим набором полів, а продуманою моделлю предметної області.

Основні поля моделі чемпіона наведено в таблиці 2.4.

Таблиця 2.4 – Основні поля моделі чемпіона

Поле	**Призначення**
id	Унікальний ідентифікатор чемпіона
name	Ім’я чемпіона
role	Основна позиція на карті
class	Ігровий клас чемпіона
difficulty	Рівень складності
description	Короткий опис стилю гри
image	Посилання на іконку або зображення
stats	Набір основних характеристик
abilities	Масив умінь чемпіона
items	Список рекомендованих предметів
strengths	Сильні сторони
weaknesses	Слабкі сторони
counters	Чемпіони або класи, що створюють загрозу
synergies	Чемпіони або класи, з якими є корисна взаємодія

Окрему увагу потрібно приділити характеристикам чемпіона. Вони подані у вигляді вкладеного об’єкта ChampionStats. Доцільно зберігати такі показники, як Hp, Damage, Mobility, Utility та DifficultyScore. Ці значення можна виводити у детальній картці у вигляді прогрес-барів або числових оцінок. Така візуалізація допомагає користувачеві швидко оцінити стиль гри чемпіона.

Уміння чемпіона зберігаються як масив об’єктів Ability. Кожен об’єкт уміння містить клавішу, назву та опис. Клавішами можуть бути Q, W, E і R, що відповідає типовій структурі вмінь у League of Legends. Така модель дозволяє автоматично формувати блок умінь у модальному вікні без ручного створення HTML-розмітки для кожного персонажа.

Рекомендовані предмети зберігаються як масив RecommendedItem. Кожен предмет може містити ідентифікатор, назву, тип, вартість,

характеристики, причину рекомендації та зображення. Причина рекомендації є особливо важливою, оскільки застосунок має не лише показати предмет, а й пояснити, чому його варто розглянути в певній ситуації. Це підтримує загальну ідею пояснюваних рекомендацій.

Окрім моделі чемпіона, у проєкті потрібна модель вибору команд. Для цього використовується клас TeamSelection. Він описує склад своєї команди та команди ворога за п’ятьма позиціями: Top, Jungle, Mid, ADC і Support. Кожна позиція може містити ідентифікатор обраного чемпіона або бути порожньою. Така структура зручна для передавання даних з клієнтської частини на сервер під час запуску аналізу.

Структура вибору команд наведена в таблиці 2.5.

Таблиця 2.5 – Структура моделі TeamSelection

Поле	**Призначення**
allyTop	Чемпіон своєї команди на верхній лінії
allyJungle	Чемпіон своєї команди в лісі
allyMid	Чемпіон своєї команди на середній лінії
allyAdc	Чемпіон своєї команди на позиції ADC
allySupport	Чемпіон своєї команди на позиції Support
enemyTop	Чемпіон ворожої команди на верхній лінії
enemyJungle	Чемпіон ворожої команди в лісі
enemyMid	Чемпіон ворожої команди на середній лінії
enemyAdc	Чемпіон ворожої команди на позиції ADC
enemySupport	Чемпіон ворожої команди на позиції Support

У серверній частині центральним модулем є сервіс GameDataService. Він відповідає за завантаження JSON-файлу, десеріалізацію даних, кешування списку чемпіонів, пошук чемпіона за ідентифікатором, фільтрацію даних і формування рекомендацій. Винесення цієї логіки в окремий сервіс дозволяє не перевантажувати сторінки й API-маршрути зайвим кодом.

Сервісна архітектура має важливі переваги. Якщо логіка роботи з даними зміниться, наприклад JSON-файл буде замінено на базу даних, основні сторінки не потребуватимуть значного переписування. Достатньо буде змінити реалізацію сервісу. Такий підхід робить проєкт більш підтримуваним і відповідає принципу розділення відповідальностей.

API-маршрути потрібні для взаємодії клієнтської частини із сервером. Наприклад, один маршрут може повертати повний список чемпіонів, інший – дані конкретного чемпіона, а третій – результат аналізу матчу. Клієнтський JavaScript звертається до цих маршрутів за допомогою fetch-запитів, отримує JSON-відповідь і оновлює інтерфейс.

Таким чином, структура даних вебзастосунку базується на JSON-моделі чемпіона, а програмні модулі розділені за функціональними зонами. Серверна логіка зосереджена в GameDataService, клієнтська інтерактивність – у JavaScript-файлах, сторінки – у папці Pages, а статичні ресурси – у wwwroot. Це забезпечує логічну організацію проєкту та створює основу для подальшої реалізації.

2.3 Проєктування користувацького інтерфейсу та візуальної стилістики

Користувацький інтерфейс є важливою частиною вебзастосунку, оскільки саме через нього користувач взаємодіє з функціональністю системи. Навіть якщо програмна логіка працює коректно, незручний або

перевантажений інтерфейс може ускладнити використання застосунку. Тому під час проєктування «LoL Champion Guide & Recommender» було враховано не лише наявність функцій, а й спосіб їх подання, порядок переходів між сторінками, читабельність тексту, адаптивність і загальну візуальну стилістику.

Інтерфейс застосунку повинен відповідати трьом основним вимогам. По-перше, він має бути зрозумілим для користувача, який не знайомий із внутрішньою структурою програми. По-друге, він має відповідати ігровій тематиці League of Legends, щоб створювати візуальну цілісність. По-третє, він має бути достатньо простим і не перевантаженим, оскільки головне завдання системи – допомогти користувачеві швидко отримати потрібну інформацію.

Головна сторінка виконує роль першої точки взаємодії користувача із системою. Вона має містити великий заголовок, короткий опис призначення застосунку та кнопки переходу до основних розділів. На ній доцільно розмістити блоки, які коротко пояснюють можливості системи: інтерактивна карта, каталог чемпіонів і аналізатор матчу. Така структура дозволяє користувачеві одразу зрозуміти, що застосунок не є просто статичним довідником.

Глобальна навігація реалізується через бургер-меню. Такий підхід зручний для адаптивного інтерфейсу, оскільки не займає багато місця у верхній частині сторінки й однаково добре працює на широкому та вузькому екрані. При натисканні на кнопку меню відкривається бічна панель з посиланнями на головну сторінку, карту, каталог чемпіонів і аналізатор матчу. Активний пункт меню доцільно виділяти кольором або індикатором, щоб користувач бачив, у якому розділі перебуває.

Сторінка інтерактивної карти призначена для візуального ознайомлення користувача з картою Summoner’s Rift. На сторінці відображається зображення карти, на яке накладаються інтерактивні маркери. Маркери можуть позначати верхню лінію, ліс, середню лінію,

нижню лінію, позицію підтримки, Baron Nashor, Dragon, Rift Herald і Nexus. Коли користувач натискає на маркер, праворуч або нижче від карти відкривається інформаційна панель.

Інформаційна панель карти повинна містити назву обраної області, короткий опис її ролі та додаткові рекомендації. Якщо користувач обирає лінію, система може показати типових чемпіонів для цієї позиції. Якщо користувач обирає нейтральний об’єкт, панель може пояснити його стратегічне значення. Такий підхід перетворює карту на навчальний інструмент, а не просто декоративне зображення.

Сторінка каталогу чемпіонів є довідковим центром застосунку. Вона повинна містити фільтри, пошук і сітку карток. Користувач може знайти конкретного чемпіона за іменем, обрати роль або клас, а також відфільтрувати персонажів за складністю. Кожна картка повинна показувати лише основну інформацію: іконку, ім’я, роль, клас і складність. Детальна інформація відкривається в модальному вікні.

Модальне вікно детальної картки дозволяє показати розширену інформацію про чемпіона без переходу на окрему сторінку та втрати поточного стану каталогу. У ньому відображаються характеристики персонажа, опис умінь, рекомендовані предмети, сильні та слабкі сторони. Під час відкриття модального вікна основний вміст сторінки візуально затемнюється, а взаємодія користувача зосереджується на активному діалоговому блоці.

Під час реалізації модального вікна потрібно забезпечити можливість його закриття за допомогою окремої кнопки та клавіші Escape, а також правильно керувати клавіатурним фокусом. Відповідно до рекомендацій WAI-ARIA, фокус після відкриття діалогового вікна має переміщуватися всередину нього, не виходити за його межі під час переходу клавішею Tab і повертатися до елемента, який відкрив вікно, після його закриття [24]. Це забезпечує зручність роботи як за допомогою миші, так і клавіатури.

Основні елементи картки чемпіона наведено в таблиці 2.6.

Таблиця 2.6 – Основні елементи картки чемпіона

Елемент картки	**Призначення**
Іконка чемпіона	Візуальна ідентифікація персонажа
Ім’я	Назва чемпіона
Роль	Позиція на карті
Клас	Тип ігрової поведінки
Складність	Орієнтовний рівень складності для користувача
Кнопка деталей	Відкриття повної інформації
Теги	Швидке сприйняття категорій чемпіона

Аналізатор матчу має складніший інтерфейс, ніж інші сторінки, оскільки користувач повинен вибрати кілька чемпіонів і отримати структурований результат. Доцільно поділити інтерфейс аналізатора на дві колонки: «Моя команда» і «Команда ворога». У кожній колонці розміщуються п’ять позицій: Top, Jungle, Mid, ADC і Support. Біля кожної позиції є кнопка вибору чемпіона.

Після натискання кнопки вибору відкривається модальне вікно зі списком чемпіонів. У ньому має бути пошук, щоб користувач міг швидко знайти потрібного персонажа. Після вибору чемпіона відповідна позиція оновлюється: з’являється іконка, ім’я та, за потреби, клас або роль. Це дозволяє користувачеві бачити сформований склад команд до запуску аналізу.

Результат аналізу не повинен бути суцільним текстом. Його доцільно поділити на логічні блоки: короткий підсумок, основні загрози, рекомендовані предмети, переваги, слабкі сторони та практичні поради. Такий спосіб подання полегшує сприйняття та робить рекомендації більш зрозумілими. Наприклад, блок загроз може виділяти небезпечних ворогів, а блок предметів – пояснювати, чому певний предмет корисний.

Візуальна стилістика всього застосунку повинна бути єдиною. Для цього в CSS-файлі визначаються загальні змінні кольорів, стилі кнопок,

карток, заголовків, модальних вікон і таблиць. Єдина палітра створює відчуття цілісності. Якщо кожна сторінка матиме власний стиль, користувач сприйматиме застосунок як набір непов’язаних елементів.

Кольорову палітру та стилістичні акценти інтерфейсу вебзастосунку наведено на рисунку 2.4.

Рисунок 2.4 – Кольорова палітра та стилістичні акценти інтерфейсу

На рисунку 2.4 наведено основні кольори інтерфейсу. Темний фон зменшує візуальне навантаження, темно-сині картки створюють глибину інтерфейсу, золотий акцент підкреслює кнопки та активні елементи, блакитне світіння виділяє інтерактивні точки карти, а світлий основний текст забезпечує читабельність.

Адаптивність інтерфейсу забезпечується за допомогою CSS-сіток, flex-контейнерів і медіазапитів. На широкому екрані картки чемпіонів можуть розміщуватися у кілька колонок, карта й інформаційна панель можуть розташовуватися поруч, а аналізатор – у дві колонки. На вузькому екрані ці

блоки повинні переходити один під одним, щоб не виникало горизонтального прокручування [11].

Особливо важливо забезпечити зручність натискання на мобільних пристроях. Кнопки не повинні бути занадто малими, відстань між інтерактивними елементами має бути достатньою, а модальні вікна повинні коректно відкриватися на малих екранах. Бургер-меню має закриватися як кнопкою, так і кліком поза панеллю.

Під час проєктування інтерфейсу потрібно враховувати не лише естетику, а й доступність. Текст має мати достатній контраст із фоном. Кнопки повинні мати зрозумілі підписи. Інтерактивні елементи мають візуально реагувати на наведення або натискання. Повідомлення про помилки повинні бути зрозумілими й не повинні залишати користувача без пояснення.

Таким чином, проєктування користувацького інтерфейсу включає розроблення головної сторінки, глобальної навігації, інтерактивної карти, каталогу чемпіонів, детальної картки та аналізатора матчу. Усі ці компоненти повинні мати єдину стилістику, бути адаптивними, зрозумілими та зручними для користувача.

2.4 Формалізація алгоритму рекомендацій та взаємодії клієнта і сервера

Алгоритм формування рекомендацій є центральним елементом розроблюваного вебзастосунку, оскільки саме він відрізняє систему від звичайного довідника. Якщо каталог чемпіонів лише показує інформацію, то аналізатор матчу намагається застосувати ці дані до конкретної ситуації. У межах цієї роботи рекомендації формуються не на основі машинного навчання, а за допомогою правилового алгоритму, який перевіряє склад команд і визначає типові загрози.

Правиловий алгоритм є доцільним для початкової версії застосунку з кількох причин. По-перше, він прозорий і легко пояснюється. По-друге, він не потребує великого набору статистичних даних. По-третє, він може бути реалізований у серверному C#-сервісі без підключення зовнішніх аналітичних систем. По-четверте, правила можна поступово розширювати й уточнювати.

На вхід алгоритму надходить модель TeamSelection, яка містить склад своєї команди та склад команди ворога. Кожна команда складається з п’яти позицій: Top, Jungle, Mid, ADC і Support. Користувач може заповнити повний склад або лише окремі позиції. Якщо певна позиція не заповнена, алгоритм пропускає її або використовує лише доступну інформацію. Це робить аналізатор гнучким.

Перший етап алгоритму – перевірка вхідних даних. Система визначає, чи є у запиті хоча б один обраний чемпіон. Якщо даних недостатньо, користувачеві повертається повідомлення про необхідність вибрати чемпіонів. Якщо дані є, сервер шукає відповідні об’єкти чемпіонів у JSON-базі.

Другий етап – визначення основного чемпіона користувача. У простій реалізації система може орієнтуватися на першу заповнену позицію своєї команди або на окремо визначену роль, якщо вона передається з інтерфейсу. Основний чемпіон потрібен для формування персоналізованого блоку рекомендацій: сильні сторони, слабкі сторони, рекомендовані предмети та порівняння з опонентом.

Третій етап – пошук прямого опонента на тій самій позиції. Наприклад, якщо користувач вибрав чемпіона на середній лінії, система перевіряє, хто обраний у ворога на Mid. Це дозволяє сформувати блок матч-апу. Якщо прямий опонент відомий, система може порівняти класи чемпіонів, перевірити контрпіки або видати попередження про можливі ризики.

Четвертий етап – аналіз усієї ворожої команди. Алгоритм проходить по кожному обраному ворогу та визначає його клас. Клас чемпіона дозволяє

зробити базове припущення про характер загрози. Наприклад, Assassin створює загрозу швидкого усунення слабких цілей, Tank має високу витривалість, Mage може завдавати великої магічної шкоди або контролювати зону, Marksman небезпечний стабільною шкодою в пізній фазі гри, Support може забезпечувати лікування, щити або контроль.

П’ятий етап – формування рекомендацій щодо предметів і поведінки. Для кожного типу загрози можна визначити типову відповідь. Проти ассасинів доцільно радити обережне позиціонування та захисні предмети. Проти танків – предмети на пробивання захисту. Проти лікування – anti-heal ефекти. Проти великої кількості контролю – обережнішу гру, очищення ефектів або уникнення непідготовлених боїв.

Послідовність роботи алгоритму формування рекомендацій наведено на рисунку 2.5.

Рисунок 2.5 – Послідовність роботи алгоритму формування рекомендацій

На рисунку 2.5 показано, що алгоритм починається з отримання моделі TeamSelection, після чого система визначає чемпіона користувача, порівнює його з ворогом на лінії, аналізує склад команди противника та формує результат. Окремо виділено блок правил і шкалу загроз, які використовуються для прийняття рішення.

Приклади правил формування рекомендацій наведено в таблиці 2.7.

Таблиця 2.7 – Приклади правил формування рекомендацій

Умова	**Ризик**	**Рекомендація**
У команді ворога є Assassin	Швидке усунення слабкої цілі	Тримати дистанцію, розглянути захисний предмет
У ворогів багато Tank або Fighter	Висока витривалість команди	Планувати предмети на пробивання захисту
У ворогів є сильне лікування	Складно завершити бій	Використати предмети або ефекти проти лікування
Ворожий Marksman має перевагу	Небезпечна стабільна шкода	Обмежувати його позиціонування та уникати затяжних боїв
У ворогів багато контролю	Ризик втрати мобільності	Не починати бій без бачення й позиційної переваги
Прямий опонент є контрпіком	Складний матч-ап	Грати обережніше на ранніх етапах

Важливим елементом алгоритму є усунення дублювання рекомендацій. Якщо в команді ворога кілька чемпіонів одного класу, система не повинна кілька разів повторювати однакову пораду. Для цього можна використовувати колекції з унікальними значеннями або перевіряти, чи вже додано певну рекомендацію до списку. Це робить результат компактним і читабельним.

Після формування рекомендацій сервер повертає клієнту структурований JSON-об’єкт. Він може містити такі поля: summary, threats, recommendedItems, advantages, weaknesses, tips. Кожне поле містить окремий текст або масив повідомлень. Такий формат зручний, оскільки клієнтська частина може відобразити кожен блок окремо.

Формат JSON є текстовим і незалежним від конкретної мови програмування форматом обміну структурованими даними. Він підтримує подання рядків, чисел, логічних значень, порожнього значення null, а також

складених структур у вигляді об’єктів і масивів [27]. Це дозволяє серверній частині формувати результат аналізу засобами C#, а клієнтській частині обробляти той самий результат засобами JavaScript.

Відповідь аналізатора містить окремі поля summary, threats, recommendedItems, advantages, weaknesses і tips. Поділ результату на структуровані компоненти спрощує його оброблення та дає змогу відображати кожну групу рекомендацій в окремому візуальному блоці.

Структура відповіді аналізатора наведена в таблиці 2.8.

Таблиця 2.8 – Структура відповіді аналізатора

Поле відповіді	**Зміст**
summary	Загальний підсумок аналізу
playerChampion	Основний чемпіон користувача
laneOpponent	Прямий опонент на лінії
threats	Список основних загроз
recommendedItems	Рекомендовані предмети
advantages	Сильні сторони обраного чемпіона
weaknesses	Слабкі сторони або ризики
tips	Практичні поради щодо поведінки

Взаємодія клієнтської та серверної частин відбувається за допомогою HTTP-запитів. Для їх надсилання у клієнтському JavaScript-коді використовується Fetch API, який надає браузерний інтерфейс для отримання мережевих ресурсів і роботи з об’єктами Request та Response [26]. Метод fetch() повертає об’єкт Promise, що дозволяє асинхронно очікувати відповідь сервера без повного перезавантаження вебсторінки.

Під час відкриття каталогу клієнт надсилає GET-запит для отримання списку чемпіонів. Після цього отримані дані використовуються для формування карток, фільтрів і модальних вікон. Під час запуску аналізатора

застосовується POST-запит, у тілі якого передається модель TeamSelection зі складами союзної та ворожої команд.

Клієнтська частина не повинна самостійно містити всю логіку рекомендацій. Її завдання полягає у збиранні даних від користувача, передаванні їх на сервер і відображенні результату. Серверна частина виконує аналіз, оскільки саме там зосереджено доступ до моделі даних і правил. Такий поділ робить систему більш зрозумілою.

Використання окремого програмного інтерфейсу для взаємодії клієнтської та серверної частин відповідає загальним принципам API-орієнтованої архітектури. У роботі Д. О. Руденко та В. В. Маренича проаналізовано сучасні способи інтеграції програмних застосунків через API, зокрема модель «запит–відповідь», REST-підхід, питання масштабованості, версіонування та обмеження кількості запитів [25]. Для розроблюваного вебзастосунку використовується спрощений варіант такої взаємодії: клієнт передає серверу вибраний склад команд, а сервер повертає сформований результат аналізу.

Послідовність взаємодії під час аналізу матчу можна описати так: користувач вибирає чемпіонів у інтерфейсі, JavaScript формує об’єкт TeamSelection, дані надсилаються POST-запитом на сервер, GameDataService знаходить чемпіонів у базі, виконує правила аналізу, формує результат, сервер повертає JSON-відповідь, клієнт відображає блоки рекомендацій.

Послідовність роботи аналізатора наведено в таблиці 2.9.

Таблиця 2.9 – Послідовність роботи аналізатора матчу

Крок	**Дія**	**Результат**
1	Користувач вибирає чемпіонів	Формується склад команд
2	JavaScript зберігає вибір	Дані записуються у внутрішній стан і localStorage

Продовження таблиці 2.9

Крок	Дія	Результат
3	Користувач натискає кнопку аналізу	Формується JSON-запит
4	Сервер отримує TeamSelection	Запит передається до GameDataService
5	Сервіс шукає чемпіонів	Дані беруться з JSON-бази
6	Алгоритм перевіряє правила	Формуються загрози й рекомендації
7	Сервер повертає JSON-відповідь	Клієнт отримує результат
8	JavaScript відображає результат	Користувач бачить рекомендації

Для збереження стану аналізатора використовується localStorage. Коли користувач вибирає чемпіона, JavaScript оновлює об’єкт стану та записує його в localStorage у вигляді JSON-рядка. Після перезавантаження сторінки скрипт перевіряє, чи є збережені дані, і відновлює вибір. Це покращує зручність користування, оскільки користувач не втрачає вже введену інформацію.

При використанні localStorage важливо не зберігати зайву інформацію. Достатньо зберігати ідентифікатори чемпіонів за позиціями. Повні дані про чемпіонів можна повторно завантажити із сервера або JSON-файлу. Такий підхід зменшує обсяг збережених даних і дозволяє завжди відображати актуальну інформацію з бази.

Алгоритм рекомендацій повинен враховувати, що дані можуть бути неповними. Користувач може вибрати лише одного чемпіона або заповнити лише ворожу команду. У такій ситуації система не повинна завершувати роботу помилкою. Вона має виконати частковий аналіз або повідомити, яких даних не вистачає. Це важливо для стабільності застосунку.

Можливі напрями розширення алгоритму наведено в таблиці 2.10.

Таблиця 2.10 – Напрями подальшого розширення алгоритму

Напрям розширення	**Очікуваний ефект**
Додавання вагових коефіцієнтів	Пріоритетне ранжування загроз
Урахування фази гри	Точніші поради для ранньої, середньої та пізньої гри
Окрема база предметів	Менше дублювання та гнучкіші рекомендації
Підключення зовнішніх даних	Оновлення інформації відповідно до актуального стану гри
Аналіз синергій союзників	Рекомендації не лише проти ворогів, а й за складом своєї команди
Історія аналізів	Можливість повторного перегляду попередніх результатів

Таким чином, алгоритм формування рекомендацій у розроблюваному вебзастосунку базується на правилах, які аналізують склад команд, класи чемпіонів і типові загрози. Взаємодія клієнта і сервера реалізується через JSON-запити, що забезпечує зручний обмін даними між JavaScript-кодом і C#-сервісом. Такий підхід є достатнім для початкової версії системи та створює основу для подальшого ускладнення рекомендаційної логіки.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

3.1 Реалізація серверної частини та роботи з даними

Програмну реалізацію вебзастосунку «LoL Champion Guide & Recommender» виконано відповідно до спроєктованої архітектури ASP.NET Core Razor Pages. Основною метою реалізації було створення багатосторінкового вебзастосунку, який забезпечує перегляд довідкової інформації про чемпіонів League of Legends, роботу з інтерактивною картою Summoner’s Rift, вибір складів команд і формування рекомендацій на основі правилового алгоритму.

Серверна частина застосунку відповідає за запуск вебпроєкту, обслуговування Razor-сторінок, підключення статичних ресурсів, завантаження JSON-бази даних, оброблення запитів клієнтської частини та формування результатів аналізу. Саме серверна логіка забезпечує зв’язок між структурованими даними про чемпіонів і рекомендаційним модулем.

У корені проєкту розміщується файл Program.cs, який є точкою входу вебзастосунку. У ньому налаштовується робота Razor Pages, підключається обслуговування статичних файлів із папки wwwroot, реєструється сервіс роботи з даними та визначаються маршрути API. Така структура відповідає типовій організації ASP.NET Core-проєкту та дозволяє зосередити базові налаштування в одному місці.

Файл Program.cs виконує кілька важливих функцій. По-перше, він створює об’єкт вебзастосунку. По-друге, підключає сервіси, необхідні для роботи сторінок і оброблення даних. По-третє, визначає порядок оброблення HTTP-запитів. По-четверте, запускає застосунок на локальному сервері. Завдяки цьому користувач може відкрити програму в браузері після запуску проєкту у Visual Studio 2022.

У проєкті передбачено використання статичних файлів. До них належать CSS-стилі, JavaScript-файли, JSON-база даних, зображення карти та інші ресурси, які безпосередньо завантажуються браузером. Для цього в ASP.NET Core використовується механізм обслуговування статичних файлів. Усі такі ресурси розміщуються у папці wwwroot, що є стандартним місцем для публічних файлів вебзастосунку.

Окремим елементом серверної реалізації є моделі даних. Вони розміщуються у папці Models і описують структуру об’єктів, з якими працює програма. Основною моделлю є Champion, яка відображає інформацію про чемпіона League of Legends. Вона містить поля для ідентифікатора, імені, ролі, класу, рівня складності, опису, зображення, характеристик, умінь, предметів, сильних і слабких сторін.

Використання моделей є важливим, оскільки дозволяє працювати з даними не як із довільним текстом, а як із типізованими об’єктами C#. Це зменшує кількість помилок, спрощує пошук потрібних властивостей і робить код зрозумілішим. Наприклад, замість ручного звернення до рядкових ключів JSON-коду сервер працює з властивостями об’єкта Champion.

Крім моделі Champion, у проєкті використовується модель TeamSelection. Вона потрібна для передавання складу команд із клієнтської частини на сервер. Модель містить позиції своєї команди та команди ворога: Top, Jungle, Mid, ADC і Support. Кожна позиція зберігає ідентифікатор обраного чемпіона. Саме ця модель використовується аналізатором матчу.

Логіку роботи з даними винесено до окремого сервісу GameDataService, який є одним із центральних компонентів серверної частини. Сервіс відповідає за завантаження JSON-файлу, перетворення його в колекцію об’єктів Champion, пошук чемпіонів за ідентифікатором, отримання повного списку даних і формування рекомендацій за складом команд. Завдяки цьому логіка опрацювання даних не дублюється в окремих сторінках і маршрутах.

Поділ програмної системи на окремі компоненти з чітко визначеною відповідальністю відповідає загальним принципам SOLID та чистої архітектури. У дослідженні Є. О. Дацок, О. М. Дацока та Д. О. Руденко показано, що ізоляція логіки, застосування інтерфейсів та інверсії залежностей сприяють підвищенню підтримуваності, гнучкості й розширюваності інформаційних систем [28]. У розроблюваному вебзастосунку ці принципи реалізовано шляхом відокремлення моделей, сервісу роботи з даними, API-маршрутів і клієнтського інтерфейсу.

GameDataService реєструється в Program.cs за допомогою вбудованого механізму впровадження залежностей ASP.NET Core. Після реєстрації екземпляр сервісу може автоматично передаватися компонентам, які потребують доступу до ігрових даних. Це дає змогу не створювати об’єкт сервісу вручну під час кожного запиту та спрощує подальшу заміну способу зберігання даних [18]. Завантаження даних із JSON-файлу відбувається під час першого звернення до сервісу або під час ініціалізації застосунку. Файл champions.json містить масив об’єктів, кожен з яких описує окремого чемпіона. Для перетворення JSON-тексту в C#-об’єкти використовується десеріалізація. У результаті сервер отримує колекцію об’єктів Champion, з якою можна працювати програмно.

Структура взаємодії серверної частини з даними наведена в таблиці 3.1.

Таблиця 3.1 – Основні етапи роботи серверної частини з даними

Етап	**Дія**	**Результат**
1	Запуск застосунку	Ініціалізація ASP.NET Core-проєкту
2	Підключення статичних файлів	Браузер отримує CSS, JavaScript, JSON і зображення
3	Реєстрація GameDataService	Сервіс доступний для оброблення запитів
4	Завантаження champions.json	Дані зчитуються зі статичного файлу

Продовження таблиці 3.1

Етап	Дія	Результат
5	Десеріалізація JSON	Дані перетворюються в об’єкти Champion
6	Оброблення API-запиту	Сервер повертає список чемпіонів або результат аналізу

Під час реалізації важливо було забезпечити правильну відповідність між JSON-структурою та C#-моделями. Якщо назви полів у JSON-файлі не відповідають властивостям моделі, частина даних може не завантажитися. Тому структура Champion повинна повторювати основні поля, передбачені у файлі champions.json. Для вкладених об’єктів і масивів використовуються окремі допоміжні класи, наприклад для характеристик, умінь і предметів.

API-маршрути потрібні для взаємодії клієнтської частини із сервером. Вони дають змогу отримувати дані без повного перезавантаження сторінки. Наприклад, сторінка каталогу може надіслати запит на отримання всіх чемпіонів, після чого JavaScript сформує картки. Аналізатор може надіслати POST-запит зі складом команд, а сервер поверне готовий результат аналізу у форматі JSON.

Такий підхід забезпечує динамічність застосунку. Користувач натискає кнопку, змінює фільтр або запускає аналіз, а сторінка оновлює лише потрібну частину інтерфейсу. Це робить застосунок зручнішим і наближає його поведінку до сучасних вебсистем.

Для реалізації API-запиту аналізатора використовується передавання JSON-об’єкта з клієнта на сервер. JavaScript формує об’єкт TeamSelection, у якому зберігаються вибрані чемпіони. Потім цей об’єкт передається на сервер методом POST. Сервер приймає дані, знаходить відповідних чемпіонів у базі та запускає логіку формування рекомендацій.

Під час отримання POST-запиту серверна частина перетворює JSON-вміст тіла запиту на типізований об’єкт TeamSelection. В ASP.NET Core цей

процес виконується механізмом прив’язування моделі, який отримує дані з тіла HTTP-запиту, параметрів маршруту або рядка запиту та перетворює їх на властивості відповідного об’єкта C# [29]. Після цього сформована модель передається до методу аналізу сервісу GameDataService.

Перед виконанням аналізу перевіряється коректність отриманої моделі: наявність об’єкта запиту, правильність ідентифікаторів чемпіонів та допустима кількість вибраних позицій. Механізм валідації ASP.NET Core працює після прив’язування моделі та дозволяє виявити помилки перетворення даних і порушення встановлених правил до початку виконання основної серверної логіки [30]. У разі некоректного запиту сервер повертає відповідний код помилки та повідомлення для клієнтської частини.

Результат аналізу також повертається у форматі JSON. Він містить структуровані блоки: загальний підсумок, список загроз, рекомендовані предмети, переваги, слабкі сторони та поради. Завдяки цьому клієнтська частина може окремо вивести кожен блок у відповідному елементі інтерфейсу.

У серверній реалізації важливо передбачити оброблення помилок. Якщо JSON-файл відсутній або має неправильну структуру, застосунок не повинен завершувати роботу без пояснення. Якщо користувач передає порожній склад команд, сервер має повернути відповідне повідомлення. Якщо чемпіона не знайдено за ідентифікатором, система повинна пропустити його або повідомити про некоректний вибір.

Основні серверні ситуації, які необхідно обробляти, наведено в таблиці 3.2.

Таблиця 3.2 – Оброблення типових серверних ситуацій

Ситуація	**Причина**	**Реакція системи**
JSON-файл не знайдено	Неправильний шлях або відсутній файл	Повернення повідомлення про помилку завантаження
Неправильна структура JSON	Помилка у файлі даних	Повідомлення про неможливість десеріалізації

Продовження таблиці 3.2

Ситуація	**Причина**	**Реакція системи**
Чемпіон не знайдений	Некоректний ідентифікатор	Пропуск позиції або повідомлення про відсутність даних
Порожній запит аналізатора	Користувач не вибрав чемпіонів	Повернення підказки щодо заповнення складу
Відсутні предмети	Неповні дані чемпіона	Формування загальних порад без предметних рекомендацій

Алгоритмічна частина GameDataService аналізує не лише одного чемпіона, а й увесь склад ворожої команди. Для цього сервіс отримує список обраних ворогів, визначає їхні класи та формує набір загроз. Якщо у ворожій команді є Assassin, система може додати попередження про ризик швидкого усунення слабкої цілі. Якщо є Tank або Fighter, система радить звернути увагу на предмети проти витривалих ворогів. Якщо є Support із лікуванням, доцільною може бути рекомендація щодо anti-heal ефектів.

У межах поточної реалізації правила мають навчальний характер. Вони не використовують складну статистику, але дозволяють показати принцип алгоритмічного формування рекомендацій. Кожна порада виникає не випадково, а внаслідок перевірки певної умови. Це робить систему пояснюваною та зручною для опису в кваліфікаційній роботі.

Перевагою серверної реалізації є централізація рекомендаційної логіки. Клієнтська частина не повинна дублювати правила або самостійно аналізувати класи чемпіонів. Вона лише передає дані та відображає відповідь. Якщо в майбутньому потрібно змінити правила, достатньо оновити серверний сервіс, не переписуючи інтерфейс.

У процесі реалізації було враховано, що JSON-база може бути неповною. Початкова версія не обов’язково містить усіх чемпіонів League of Legends. Проте структура даних спроєктована так, щоб базу можна було

поступово розширювати. Додавання нового чемпіона полягає у створенні нового запису в JSON-файлі за встановленим шаблоном.

Таким чином, серверна частина вебзастосунку реалізує запуск проєкту, роботу з моделями, завантаження структурованих даних, API-взаємодію та правилове формування рекомендацій. Вона є основою всієї системи, оскільки забезпечує зв’язок між даними про чемпіонів і функціями користувацького інтерфейсу.

3.2 Реалізація клієнтської частини та інтерактивних компонентів

Клієнтська частина вебзастосунку відповідає за взаємодію користувача із системою. Саме вона визначає, як користувач бачить сторінки, як переходить між розділами, як натискає на елементи карти, як фільтрує чемпіонів, як відкриває детальні картки та як запускає аналіз матчу. Тому реалізація клієнтської частини має таке саме значення, як і серверна логіка.

Основою клієнтської частини є Razor-сторінки, HTML-розмітка, CSS-стилі та JavaScript-файли. Razor Pages використовуються для створення структури сторінок, HTML визначає розміщення основних елементів, CSS відповідає за зовнішній вигляд і адаптивність, а JavaScript забезпечує інтерактивність. Такий розподіл дозволяє підтримувати зрозумілу структуру проєкту.

Після завантаження сторінки клієнтський JavaScript-код надсилає асинхронний GET-запит до серверного маршруту для отримання списку чемпіонів. Для виконання мережевого запиту використовується Fetch API, який дає змогу отримувати ресурси без повного перезавантаження вебсторінки та асинхронно опрацьовувати відповідь сервера [26]. Отримана JSON-відповідь перетворюється на масив JavaScript-об’єктів, на основі якого формуються картки каталогу, елементи пошуку, списки вибору в аналізаторі та вміст модальних вікон.

Усі сторінки використовують спільний макет _Layout.cshtml. У цьому файлі підключаються глобальні стилі, JavaScript-файли, шрифти, а також розміщується загальна навігація. Завдяки цьому всі сторінки мають однакову структуру: спільне меню, єдиний фон, однакові відступи, стилі кнопок і карток. Це створює враження цілісного програмного продукту.

Головна сторінка Index.cshtml реалізує презентаційний екран застосунку. Вона містить заголовок, короткий опис призначення системи, кнопки переходу до карти й каталогу, а також блоки з описом основних можливостей. Ця сторінка не перевантажується складною логікою, оскільки її головне завдання – зорієнтувати користувача та показати, які функції доступні в системі.

Реалізовану головну сторінку вебзастосунку наведено на рисунку 3.1

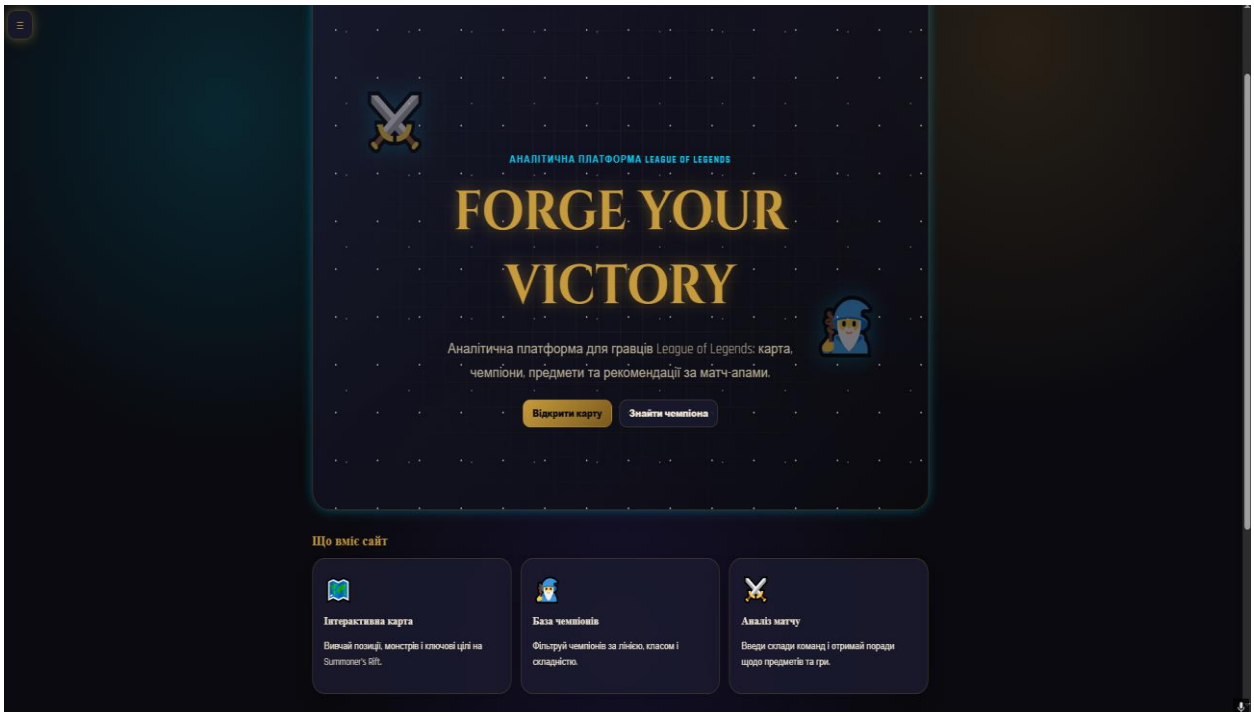

Рисунок 3.1 – Головна сторінка вебзастосунку

На рисунку 3.1 показано головний екран застосунку з центральним заголовком, коротким поясненням призначення системи, кнопками переходу до основних розділів і блоками можливостей. Такий вигляд дозволяє

користувачеві швидко зрозуміти, що застосунок призначений для роботи з картою, чемпіонами та рекомендаціями.

Навігація реалізована через бургер-меню. Користувач натискає кнопку меню, після чого відкривається бічна панель із пунктами переходу. Меню можна закрити повторним натисканням, кнопкою закриття або кліком поза панеллю. Такий підхід зручний для адаптивного дизайну, оскільки меню не займає багато місця на екрані та залишається доступним на мобільних пристроях.

Логіка відкриття та закриття меню реалізується у JavaScript. Скрипт додає або прибирає CSS-клас, який відповідає за видимість бічної панелі. Цей спосіб простий, зрозумілий і не потребує сторонніх бібліотек. Він також дозволяє легко додати анімацію відкриття меню через CSS-переходи.

Сторінка Map.cshtml реалізує інтерактивну карту Summoner’s Rift. Основним елементом сторінки є контейнер із зображенням карти. Поверх цього зображення розміщуються маркери, які відповідають важливим позиціям: Top, Jungle, Mid, ADC, Support, Baron Nashor, Dragon, Rift Herald і Nexus. Кожен маркер є клікабельним елементом.

Для маркерів карти використовується абсолютне позиціонування у відсотках. Це означає, що координати кожної точки задаються відносно ширини та висоти контейнера карти. Завдяки цьому маркери залишаються на правильних місцях навіть тоді, коли карта масштабується на різних екранах. Такий підхід є кращим, ніж фіксовані координати в пікселях.

Після натискання на маркер JavaScript визначає, який саме елемент було обрано, і оновлює інформаційну панель. Якщо користувач натиснув на лінію, панель показує її опис і рекомендованих чемпіонів. Якщо користувач натиснув на нейтральний об’єкт, панель показує його значення для матчу. Така реалізація робить карту навчальним і навігаційним елементом.

Реалізацію інтерактивної карти з маркерами позицій та нейтральних об’єктів наведено на рисунку 3.2.

Рисунок 3.2 – Реалізація інтерактивної карти з маркерами позицій та об'єктів

На рисунку 3.2 показано карту Summoner's Rift з позначеними інтерактивними точками. Користувач може натиснути на відповідну позицію або об'єкт і отримати пояснення в інформаційній панелі. Такий підхід поєднує візуальне подання предметної області та навчальну функцію.

Сторінка Champions.cshtml реалізує каталог чемпіонів. Після завантаження сторінки JavaScript отримує список чемпіонів із сервера або JSON-файлу, після чого створює картки. Кожна картка формується динамічно на основі даних. Це означає, що розробнику не потрібно вручну створювати HTML для кожного чемпіона. Достатньо додати запис до бази даних, і він буде відображений у каталозі.

Картка чемпіона містить іконку, ім'я, роль, клас і складність. Для складності використовується візуальне позначення у вигляді зірочок. Картка також має кнопку для відкриття детальної інформації. При наведенні на картку може змінюватися тінь, рамка або колір, що показує її інтерактивність.

Фільтрація чемпіонів реалізується на клієнтському боці. Користувач може обрати роль, клас, складність або ввести ім'я в поле пошуку. JavaScript порівнює введені значення з даними кожного чемпіона та оновлює сітку карток. Якщо чемпіон не відповідає обраним умовам, він не відображається.

Такий спосіб працює швидко для невеликої навчальної бази даних і не потребує повторних запитів до сервера після кожної зміни фільтра.

Сторінку каталогу чемпіонів із фільтрами та картками персонажів наведено на рисунку 3.3.

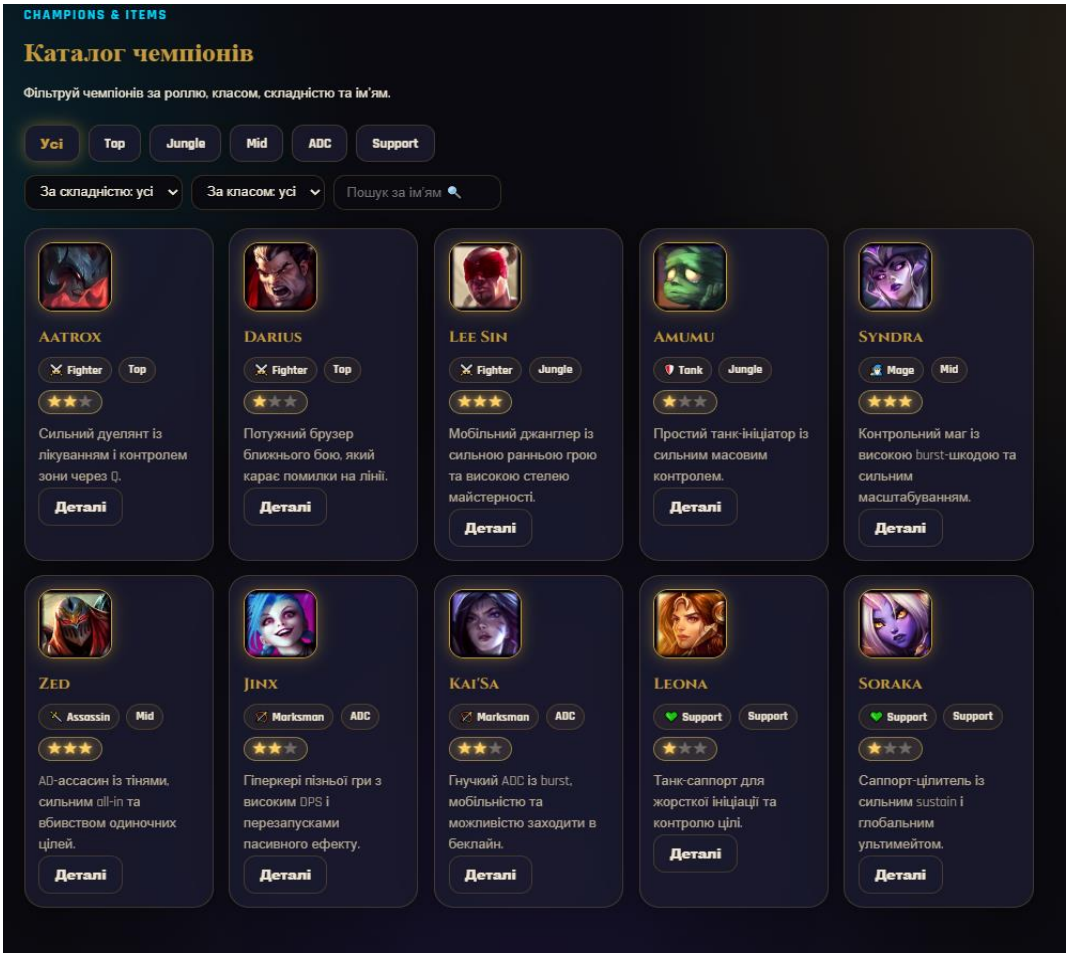

Рисунок 3.3 – Сторінка каталогу чемпіонів із фільтрами та картками

На рисунку 3.3 показано каталог чемпіонів, у якому користувач може переглядати картки, фільтрувати їх за ролями й класами та виконувати пошук за іменем. Такий інтерфейс дозволяє швидко знайти потрібного персонажа без перегляду всього списку вручну.

Модальне вікно детальної картки відкривається після вибору чемпіона. У ньому відображається повніша інформація: характеристики, уміння, предмети, сильні та слабкі сторони. Модальне вікно має затемнений фон і

кнопку закриття. Також воно може закриватися при натисканні поза його областю. Це відповідає звичній поведінці сучасних вебінтерфейсів.

Детальна картка чемпіона є важливим компонентом, оскільки вона демонструє, як структуровані дані з JSON-файлу перетворюються на зрозумілий інтерфейс. Характеристики подаються у вигляді прогрес-барів, уміння – у вигляді окремих блоків Q, W, E, R, предмети – у вигляді карток з іконками, а сильні та слабкі сторони – у вигляді списків.

Приклад детальної картки чемпіона з характеристиками, уміннями та рекомендованими предметами наведено на рисунку 3.4.

Рисунок 3.4 – Детальна картка чемпіона Syndra

На рисунку 3.4 показано модальне вікно чемпіона Syndra. У ньому відображено клас, роль, складність, характеристики, опис умінь, рекомендовані предмети, сильні та слабкі сторони. Такий формат подання дозволяє користувачеві швидко отримати повну інформацію про обраного чемпіона, не переходячи на іншу сторінку.

Основні інтерактивні компоненти клієнтської частини наведено в таблиці 3.3.

Таблиця 3.3 – Інтерактивні компоненти клієнтської частини

Компонент	**Сторінка**	**Призначення**
Бургер-меню	Усі сторінки	Перехід між основними розділами
Інтерактивні маркери	Карта	Вибір ліній і об’єктів Summoner’s Rift
Інформаційна панель	Карта	Виведення опису вибраної області
Фільтри	Каталог чемпіонів	Пошук і відбір персонажів
Картки чемпіонів	Каталог чемпіонів	Коротке представлення персонажів
Модальне вікно	Каталог і аналізатор	Показ деталей або вибір чемпіона
Вибір команди	Аналізатор	Формування складу союзників і ворогів
Блок результату	Аналізатор	Відображення рекомендацій

Сторінка Analyzer.cshtml реалізує аналізатор матчу. Вона має дві основні колонки: «Моя команда» і «Команда ворога». У кожній колонці розміщується п’ять позицій. Біля кожної позиції користувач може натиснути кнопку вибору чемпіона. Після натискання відкривається модальне вікно зі списком доступних чемпіонів.

Вибір чемпіона реалізується через JavaScript. Користувач натискає на картку в модальному вікні, після чого скрипт записує ідентифікатор чемпіона у відповідну позицію команди, оновлює візуальний блок і закриває модальне вікно. У результаті користувач бачить, які позиції вже заповнені. Під час відкриття модального вікна фокус користувача переноситься до активного

діалогового блока, а після закриття повертається до елемента, за допомогою якого вікно було відкрито, що відповідає рекомендаціям щодо доступної реалізації модальних діалогів [24].

Інтерфейс вибору складів своєї команди та команди противника наведено на рисунку 3.5.

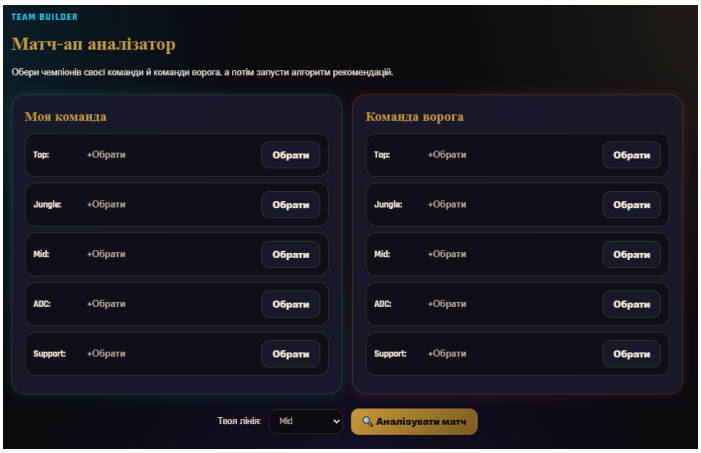

Рисунок 3.5 – Інтерфейс вибору складів команд в аналізаторі матчу

На рисунку 3.5 показано, що аналізатор поділений на дві логічні частини: склад своєї команди та склад команди противника. Така структура зручна для користувача, оскільки відповідає реальній логіці матчу League of Legends. Після вибору чемпіонів користувач може натиснути кнопку аналізу та отримати результат.

Після вибору складу користувач натискає кнопку аналізу. JavaScript формує JSON-об’єкт із даними про команди та надсилає його на сервер. Після отримання відповіді скрипт очищує попередній результат і створює нові блоки з рекомендаціями. Якщо сервер повертає список загроз, кожна загроза виводиться окремим елементом. Якщо є рекомендовані предмети, вони також відображаються у відповідному блоці.

Збереження стану аналізатора реалізується за допомогою localStorage, що забезпечує зберігання даних між оновленнями та повторними відкриттями вебсторінки [12]. Після кожного вибору чемпіона поточний склад команд перетворюється на JSON-рядок і записується в localStorage. Під

час повторного відкриття сторінки скрипт перевіряє, чи існує збережений стан. Якщо так, він відновлює обраних чемпіонів у відповідних позиціях.

CSS-оформлення застосунку реалізоване у файлі site.css. У ньому задаються основні кольори, шрифти, фон, стилі кнопок, карток, таблиць, модальних вікон, меню та адаптивних елементів. Темна палітра поєднується із золотими й блакитними акцентами. Це допомагає підкреслити ігрову тематику й зробити інтерфейс візуально цілісним.

Адаптивність реалізується за допомогою медіазапитів. На широких екранах каталог чемпіонів може показувати кілька карток у ряд, карта й інформаційна панель можуть розміщуватися поруч, а аналізатор – у дві колонки. На вузьких екранах блоки переходять один під одним, щоб користувач не мав горизонтального прокручування. Це важливо, оскільки вебзастосунок може відкриватися не лише на комп’ютері, а й на телефоні.

Основні сценарії клієнтської взаємодії наведено в таблиці 3.4.

Таблиця 3.4 – Основні сценарії клієнтської взаємодії

Сценарій	**Дія користувача**	**Результат**
Відкриття меню	Натискання на кнопку бургер-меню	Відкривається бічна навігаційна панель
Перегляд карти	Натискання на маркер	Оновлюється інформаційна панель
Пошук чемпіона	Введення імені в поле пошуку	Сітка карток фільтрується
Перегляд деталей	Натискання на картку чемпіона	Відкривається модальне вікно
Вибір команди	Натискання на позицію в аналізаторі	Обраний чемпіон додається до складу
Запуск аналізу	Натискання кнопки аналізу	Виводиться результат із рекомендаціями

Продовження таблиці 3.4

Сценарій	**Дія користувача**	**Результат**
Перезавантаження сторінки	Оновлення браузера	Вибір команд відновлюється з localStorage

Таким чином, клієнтська частина вебзастосунку реалізує всі основні елементи взаємодії користувача із системою. Вона забезпечує навігацію, перегляд карти, фільтрацію чемпіонів, відкриття деталей, вибір складів команд, збереження стану та відображення рекомендацій. Адаптивність сторінок забезпечується використанням CSS Grid, Flexbox і медіазапитів, які змінюють кількість колонок, розміри карток та розташування функціональних блоків відповідно до ширини екрана [11]. Поєднання HTML, CSS і JavaScript дозволило створити інтерактивний адаптивний інтерфейс без використання складних сторонніх фреймворків.

3.3 Тестування функціональних можливостей вебзастосунку

Тестування є важливим етапом розроблення вебзастосунку, оскільки воно дозволяє перевірити, чи відповідає реалізована система поставленим вимогам. Для вебзастосунку «LoL Champion Guide & Recommender» тестування охоплює запуск проєкту, перевірку сторінок, роботу навігації, завантаження даних, фільтрацію чемпіонів, відкриття модальних вікон, взаємодію з картою, вибір складів команд, роботу localStorage і формування рекомендацій.

Метою тестування є підтвердження працездатності основних функцій системи. Оскільки застосунок має навчальний характер, основна увага приділяється функціональному тестуванню та перевірці користувацьких сценаріїв. Автоматизоване тестування може бути додане в майбутньому, але для початкової версії достатньо ручної перевірки ключових сценаріїв.

Першим етапом тестування є перевірка запуску проєкту. Для цього проєкт відкривається у Visual Studio 2022, після чого запускається локальний сервер ASP.NET Core. У браузері повинна відкритися головна сторінка вебзастосунку. Якщо застосунок запускається без помилок, це підтверджує коректність базових налаштувань Program.cs, структури проєкту та підключення Razor Pages.

Другим етапом є перевірка підключення статичних файлів. Необхідно переконатися, що CSS-стилі застосовуються, JavaScript-файли завантажуються, зображення карти доступне, а JSON-файл може бути прочитаний. Якщо стилі не підключені, сторінка відображатиметься без оформлення. Якщо не завантажуються скрипти, інтерактивні елементи не працюватимуть. Якщо не доступний JSON-файл, каталог чемпіонів і аналізатор не зможуть отримати дані.

Третім етапом є перевірка глобальної навігації. Користувач повинен мати змогу відкрити бургер-меню, перейти на головну сторінку, сторінку карти, каталог чемпіонів і аналізатор. Меню повинно відкриватися та закриватися без помилок. На мобільному екрані меню також має залишатися доступним.

Четвертим етапом є перевірка головної сторінки. На ній мають коректно відображатися заголовок, опис, кнопки переходу та блоки можливостей. Кнопки повинні вести до відповідних розділів. Текст має бути читабельним, а декоративні елементи не повинні перекривати основну інформацію.

П’ятим етапом є тестування інтерактивної карти. Потрібно натиснути на кожен маркер і перевірити, чи оновлюється інформаційна панель. Для маркерів ліній повинні відображатися пояснення відповідної позиції. Для нейтральних об’єктів повинна показуватися інформація про їхню роль у матчі. Також потрібно перевірити, чи не зміщуються маркери при зміні розміру екрана.

Шостим етапом є перевірка каталогу чемпіонів. Після відкриття сторінки список чемпіонів має завантажитися автоматично. Картки повинні містити іконки, імена, ролі, класи та складність. Пошук за іменем має фільтрувати список відповідно до введеного тексту. Фільтри за роллю, класом і складністю мають працювати окремо та в комбінації.

Сьомим етапом є перевірка модального вікна детальної картки. Після натискання на чемпіона має відкриватися вікно з повною інформацією. У ньому повинні відображатися характеристики, уміння, предмети, сильні та слабкі сторони. Вікно повинно закриватися кнопкою або кліком поза ним. Якщо інформації багато, вікно повинно коректно прокручуватися.

Восьмим етапом є тестування аналізатора матчу. Користувач повинен мати змогу вибрати чемпіона для кожної позиції своєї команди та команди ворога. Після вибору позиція має оновлюватися. Якщо користувач змінює вибір, старий чемпіон має замінюватися новим. Якщо сторінку оновити, вибрані чемпіони повинні відновитися з localStorage.

Дев’ятим етапом є перевірка запуску аналізу. Після натискання кнопки аналізу система повинна надіслати дані на сервер і отримати відповідь. Результат має бути структурованим і містити рекомендації, а не випадковий або порожній текст. Якщо склад не заповнений, система повинна вивести підказку про необхідність вибору чемпіонів.

Приклад сформованого результату аналізу матчу наведено на рисунку 3.6.

Рисунок 3.6 – Приклад результату аналізу матчу

На рисунку 3.6 показано результат роботи аналізатора. Система визначає чемпіона користувача, прямого опонента, формує рекомендації щодо предметів, попереджає про загрози, показує переваги та надає поради щодо гри. Такий формат є зручним, оскільки користувач отримує не суцільний текст, а структуровану відповідь.

Десятим етапом є перевірка правил рекомендацій. Для цього потрібно створити кілька тестових ситуацій. Наприклад, якщо у ворожій команді є Assassin, система повинна сформувати попередження про високий ризик швидкого ураження. Якщо у ворогів є Tank або Fighter, повинна з’явитися рекомендація щодо предметів на пробивання захисту. Якщо обрано неповний склад, система повинна виконати частковий аналіз.

Контрольні сценарії тестування наведено в таблиці 3.5.

Таблиця 3.5 – Контрольні сценарії тестування вебзастосунку

№	**Сценарій**	**Очікуваний результат**
1	Запустити проєкт у Visual Studio 2022	Відкривається головна сторінка
2	Відкрити бургер-меню	З’являється навігаційна панель
3	Перейти на сторінку карти	Відображається карта Summoner’s Rift
4	Натиснути маркер Mid	Панель показує опис середньої лінії
5	Відкрити каталог чемпіонів	Завантажується сітка карток
6	Виконати пошук за іменем	Відображаються відповідні чемпіони
7	Застосувати фільтр за роллю	Список оновлюється відповідно до ролі
8	Відкрити детальну картку	З’являється модальне вікно з інформацією
9	Вибрати чемпіонів у аналізаторі	Позиції команд оновлюються
10	Натиснути кнопку аналізу	З’являються рекомендації

Продовження таблиці 3.5

№	Сценарій	Очікуваний результат
11	Оновити сторінку аналізатора	Вибір команд відновлюється
12	Запустити аналіз без вибору	Система показує підказку

Під час тестування важливо перевіряти не лише правильність функцій, а й зручність використання. Наприклад, якщо кнопка є, але її складно помітити, це погіршує інтерфейс. Якщо модальне вікно відкривається, але його важко закрити, це також є проблемою. Якщо текст рекомендацій занадто довгий або неструктурований, користувачеві буде важко його сприймати.

Окремо потрібно перевірити адаптивність. Для цього застосунок відкривається на різних розмірах екрана або перевіряється через інструменти розробника браузера. На мобільному екрані картки повинні переходити в одну колонку, меню має залишатися доступним, карта повинна масштабуватися, а аналізатор не повинен створювати горизонтальне прокручування.

Результати тестування основних функцій наведено в таблиці 3.6.

Таблиця 3.6 – Результати перевірки функціональних можливостей

Функція	Стан	Результат
Запуск проєкту	Виконано	Застосунок відкривається в браузері
Глобальна навігація	Виконано	Переходи між сторінками працюють
Інтерактивна карта	Виконано	Маркери реагують на натискання
Каталог чемпіонів	Виконано	Картки завантажуються та відображаються
Фільтрація	Виконано	Пошук і фільтри працюють коректно
Модальне вікно	Виконано	Деталі чемпіона відкриваються
Аналізатор	Виконано	Склади команд обираються

Продовження таблиці 3.6

Функція	**Стан**	**Результат**
Рекомендації	Виконано	Система формує структурований результат
localStorage	Виконано	Вибір команд зберігається
Адаптивність	Виконано	Сторінки коректно змінюють структуру

Тестування рекомендаційного алгоритму виконувалося методом функціональних сценаріїв, у кожному з яких задавався певний склад союзної та ворожої команд, після чого перевірялася відповідність отриманого результату наперед визначеним правилам. Для цього використовувалися як окремі матч-апи, наприклад Aatrox проти Darius або Syndra проти Zed, так і склади команд із переважанням певного класу чемпіонів.

Якщо у складі противника наявні танки, система повинна рекомендувати предмети або ефекти, пов’язані з пробиванням захисту. За наявності чемпіонів із відновленням здоров’я має формуватися рекомендація щодо використання anti-heal. Якщо ворожа команда містить мобільного ассасина, результат повинен містити попередження про необхідність безпечного позиціювання та використання захисних предметів. Окремо перевірялося виявлення контрпіка, якщо ідентифікатор вибраного ворога міститься у відповідному списку моделі Champion.

Приклади тестування рекомендацій наведено в таблиці 3.7.

Таблиця 3.7 – Приклади перевірки рекомендаційного алгоритму

Тестова ситуація	**Очікувана рекомендація**
У ворогів є Assassin	Тримати дистанцію, розглянути захисний предмет
У ворогів кілька Tank/Fighter	Планувати предмети проти витривалих цілей

Продовження таблиці 3.7

Тестова ситуація	**Очікувана рекомендація**
У ворогів є лікування	Розглянути предмети або ефекти проти відновлення здоров’я
У ворогів сильний Marksman	Контролювати позиціонування та уникати затяжних боїв
Склад команди неповний	Виконати частковий аналіз або показати підказку
Чемпіон відсутній у базі	Не завершувати роботу помилкою, а пропустити позицію

Крім позитивних сценаріїв, було перевірено граничні та помилкові ситуації. До них належать запуск аналізу з частково заповненим складом, повторне додавання одного чемпіона, передавання невідомого ідентифікатора, отримання порожньої відповіді та недоступність JSON-файлу. Серверна частина повинна перевіряти коректність моделі TeamSelection після прив’язування даних запиту та не передавати некоректні значення до основного алгоритму [29, 30].

Для клієнтської частини перевірено збереження і відновлення вибраних складів за допомогою localStorage [12], відкриття та закриття модального вікна, повернення клавіатурного фокуса [24], відображення повідомлення про помилку й повторне надсилання запиту. Також перевірено, що швидке послідовне застосування пошуку та фільтрів не призводить до появи карток, які не відповідають установленим умовам.

Результат тестування вважається успішним, якщо фактична реакція системи відповідає очікуваному результату, інтерфейс не переходить у некоректний стан, а користувач отримує зрозуміле повідомлення замість необробленої програмної помилки.

Тестування показує, що розроблений вебзастосунок виконує основні функціональні вимоги. Він запускається у середовищі Visual Studio 2022, відображає сторінки, завантажує дані, дозволяє переглядати чемпіонів, працювати з картою та формувати рекомендації. Основні сценарії користувача реалізовані та можуть бути використані для демонстрації роботи програмного продукту.

Водночас тестування підтверджує наявність обмежень початкової версії. Система не використовує реальні статистичні дані, не має авторизації, не зберігає історію аналізів на сервері та не оновлює баланс гри автоматично. Проте ці обмеження відповідають поставленій задачі й можуть бути розглянуті як напрями подальшого розвитку.

Таким чином, тестування функціональних можливостей підтвердило працездатність основних компонентів вебзастосунку. Перевірено запуск, навігацію, карту, каталог, фільтрацію, модальні вікна, аналізатор, збереження стану та формування рекомендацій. Це дає підстави вважати розроблений програмний продукт готовим до демонстрації та подальшого вдосконалення.

3.4 Аналіз результатів роботи та напрями подальшого розвитку

У результаті виконання кваліфікаційної роботи було розроблено вебзастосунок «LoL Champion Guide & Recommender», призначений для перегляду характеристик чемпіонів League of Legends і формування базових рекомендацій на основі складу команд. Програмний продукт поєднує кілька функціональних частин: головну сторінку, інтерактивну карту, каталог чемпіонів, детальні картки та аналізатор матчу.

Розроблений застосунок демонструє можливість створення навчально-аналітичної платформи, яка працює з ігровими даними та надає користувачеві структуровану інформацію. На відміну від звичайного довідника, система не лише показує відомості про чемпіонів, а й

використовує ці дані для формування рекомендацій. Це підвищує практичну цінність застосунку.

Одним із перспективних напрямів розвитку системи є перехід від правилового до гібридного рекомендаційного алгоритму. Для цього необхідно сформувати набір даних про матчі, виконати його попереднє опрацювання та визначити інформативні ознаки. Загальні принципи створення, навчання та перевірки нейронних моделей у задачах опрацювання даних розглянуто Д. О. Руденко, В. В. Лотвіновою та Є. В. Безверхою [31]. Окремою проблемою під час роботи з реальною статистикою можуть бути пропущені або неповні значення, підходи до нейромережевого відновлення яких досліджено в роботі [32].

Подальше вдосконалення головної сторінки може передбачати чіткіше структурування інформаційних блоків, виділення основної дії користувача, оптимізацію закликів до переходу та скорочення кількості другорядних елементів. Підходи до побудови й оцінювання цільових вебсторінок проаналізовано Д. О. Артьомовим і Д. О. Руденко [33]. У розроблюваному застосунку ці принципи можуть бути використані для покращення першого ознайомлення користувача з картою, каталогом і аналізатором матчу.

Для статистичного модуля також важливими є швидкість оброблення даних та обмеження обсягу ознак. У працях викладачів кафедри інформатики досліджено колективне використання нейронних моделей [34], скорочення структурованих описів [35] і зменшення обчислювальних витрат шляхом вилучення надлишкових компонентів [36]. Ці методи розроблялися для задач класифікації зображень і не переносяться безпосередньо на рекомендації League of Legends, однак демонструють загальний принцип: зі збільшенням обсягу даних необхідно контролювати складність моделі, кількість ознак і час формування результату.

Іншим напрямом розвитку є створення діалогового помічника, який пояснюватиме рекомендації природною мовою. Дослідження гібридної контекстної пам’яті для чатботів показує можливість поєднання поточного

контексту діалогу з пошуком релевантної інформації у зовнішньому сховищі [37]. Підходи до комбінованого використання великих мовних моделей для аналізу та генерації тексту розглянуто в роботі [38], а можливість формування відповідей на основі попередньо наданого спеціалізованого вмісту – у роботі [39].

У межах подальшого розвитку вебзастосунку мовна модель могла б отримувати структурований результат правилового або статистичного аналізатора та перетворювати його на коротке пояснення. При цьому сама модель не повинна самостійно визначати характеристики чемпіонів або вигадувати предмети: фактичні дані мають надходити з перевіреної бази, а мовний компонент повинен використовуватися лише для формування зрозумілого текстового представлення результату.

Одним із головних результатів роботи є реалізація каталогу чемпіонів. Користувач може переглядати список персонажів, знаходити потрібного чемпіона за іменем, фільтрувати за роллю, класом або складністю, а також відкривати детальну картку. Такий функціонал допомагає швидко орієнтуватися в базі даних і отримувати потрібну інформацію без перегляду великої кількості зовнішніх ресурсів.

Другим важливим результатом є створення інтерактивної карти Summoner’s Rift. Вона дозволяє користувачеві візуально ознайомитися з основними позиціями та об’єктами гри. Натискання на маркери відкриває пояснювальну інформацію, що робить карту не просто ілюстрацією, а навчальним елементом.

Третім результатом є реалізація аналізатора матчу. Користувач може вибрати склад своєї команди та команди ворога, після чого система формує рекомендації щодо можливих загроз, предметів і поведінки. Аналізатор є основним модулем, який реалізує алгоритмічне формування рекомендацій.

Четвертим результатом є створення єдиної клієнт-серверної структури. Серверна частина працює з даними й формує рекомендації, а клієнтська

частина відповідає за інтерфейс та взаємодію користувача. Такий поділ робить систему підтримуваною та придатною до розширення.

Основні результати розроблення наведено в таблиці 3.8.

Таблиця 3.8 – Основні результати розроблення вебзастосунку

Результат	**Зміст**
Головна сторінка	Реалізовано інформаційний екран із переходами до основних розділів
Інтерактивна карта	Додано маркери ліній і нейтральних об’єктів
Каталог чемпіонів	Реалізовано картки, пошук і фільтрацію
Детальна картка	Відображаються характеристики, уміння, предмети, переваги й недоліки
Аналізатор матчу	Реалізовано вибір складів команд і формування рекомендацій
JSON-база	Створено структуроване джерело даних про чемпіонів
Серверний сервіс	Реалізовано завантаження даних і правилову логіку
Адаптивний інтерфейс	Сторінки пристосовуються до різних розмірів екрана

Практична цінність роботи полягає в тому, що створений вебзастосунок може використовуватися як навчальний інструмент для гравців, які хочуть швидше орієнтуватися в ролях, характеристиках і базових взаємозв’язках між чемпіонами. Він також може бути використаний як демонстраційний програмний продукт для вивчення веброзробки, роботи з JSON-даними, клієнт-серверної взаємодії та правилових алгоритмів.

Розроблена система має кілька переваг. Вона проста для запуску, не потребує складної бази даних, має зрозумілу структуру проєкту, використовує популярні вебтехнології та може бути розширена. Дані про чемпіонів зберігаються в JSON-файлі, що спрощує їх редагування. Серверна

логіка винесена в окремий сервіс, що полегшує супровід. Клієнтська частина розділена за сторінками, що також покращує підтримуваність.

Водночас система має обмеження. Вона не підключається до реальної статистики матчів, не враховує актуальні патчі гри автоматично, не має серверної авторизації, не зберігає історію аналізів і не використовує машинне навчання. Рекомендації формуються на основі правил, тому їх точність залежить від якості закладеної бази знань.

Основні переваги та обмеження системи наведено в таблиці 3.9.

Таблиця 3.9 – Переваги та обмеження розробленого вебзастосунку

Аспект	**Переваги**	**Обмеження**
Дані	Проста JSON-структура, легке редагування	Немає автоматичного оновлення
Інтерфейс	Адаптивність, темна стилістика, інтерактивність	Потребує подальшого UX-удосконалення
Рекомендації	Прозора правилова логіка	Не використовує реальну статистику матчів
Запуск	Простий запуск у Visual Studio 2022	Потрібне середовище .NET
Розширення	Можна додавати нові сторінки й правила	Великі обсяги даних краще переносити до СКБД

Одним із головних напрямів подальшого розвитку є розширення бази чемпіонів. Початкова версія може містити обмежену кількість записів, але в майбутньому доцільно додати повний список чемпіонів League of Legends. Для цього потрібно створити записи за встановленою JSON-структурою, додати іконки, характеристики, уміння, предмети, переваги та слабкі сторони.

Другим напрямом розвитку є створення окремої бази предметів. У поточній версії предмети можуть зберігатися всередині записів чемпіонів. Це зручно для невеликого проєкту, але при розширенні може призвести до дублювання. Окрема база предметів дозволить зв’язувати предмети з чемпіонами, класами ворогів і типами загроз. Це зробить рекомендації гнучкішими.

Третім напрямом є ускладнення алгоритму рекомендацій. До правил можна додати вагові коефіцієнти. Наприклад, кожен ворог може отримувати оцінку загрози за класом, роллю, мобільністю, типом шкоди та наявністю контролю. Після цього система зможе ранжувати загрози й показувати користувачеві, які вороги є найбільш небезпечними.

Четвертим напрямом є підключення зовнішніх джерел даних. Data Dragon може використовуватися не лише для іконок, а й для оновлення базових даних про чемпіонів. Однак при цьому потрібно враховувати, що офіційні дані не завжди містять навчальні пояснення, контрпіки або логіку рекомендацій. Тому власна база знань усе одно залишатиметься важливою.

П’ятим напрямом є додавання сторінки предметів. На ній користувач міг би переглядати предмети, їхні характеристики, вартість, типи ситуацій, у яких вони корисні, а також чемпіонів, яким ці предмети підходять. Це посилило б довідкову частину системи.

Шостим напрямом є реалізація історії аналізів. Користувач міг би зберігати попередні склади команд і результати рекомендацій. Для цього можна використати як localStorage, так і серверну базу даних. У першому випадку дані залишатимуться лише в браузері, а в другому – можуть бути доступними після авторизації.

Сьомим напрямом є додавання персональних налаштувань. Користувач міг би вказати свою основну роль, улюблених чемпіонів, рівень досвіду або бажаний стиль гри. На основі цих даних система могла б формувати більш персоналізовані поради.

Напрями подальшого розвитку наведено в таблиці 3.10.

Таблиця 3.10 – Напрями подальшого розвитку вебзастосунку

Напрям	**Очікуваний результат**
Повна база чемпіонів	Розширення довідкової частини
Окрема база предметів	Зменшення дублювання та гнучкі рекомендації
Вагові коефіцієнти загроз	Пріоритетне ранжування небезпек
Підключення Data Dragon	Автоматичне отримання частини актуальних даних
Сторінка предметів	Повніший довідковий функціонал
Історія аналізів	Можливість повторного перегляду результатів
Персональні налаштування	Більш індивідуальні рекомендації
Авторизація	Збереження даних користувача на сервері
Статистичний модуль	Урахування реальних ігрових даних
Гібридний алгоритм	Поєднання правил і статистики

У перспективі застосунок може перейти від правилової моделі до гібридної. Такий підхід поєднував би ручні правила з актуальними статистичними даними. Наприклад, правила могли б визначати базові типи загроз, а статистика – уточнювати популярні предмети або складність матч-апу. Це підвищило б якість рекомендацій, але потребувало б складнішої архітектури.

Також перспективним є покращення візуальної частини. Можна додати анімації переходів між сторінками, покращені картки предметів, порівняння двох чемпіонів, режим для новачків і розширені підказки. Водночас важливо не перевантажити інтерфейс, оскільки основна перевага застосунку полягає в простоті та зрозумілості.

Підсумовуючи результати, можна зазначити, що розроблений вебзастосунок відповідає поставленій меті. Він забезпечує перегляд характеристик чемпіонів, роботу з інтерактивною картою, вибір складів

команд і формування рекомендацій. Реалізована система має зрозумілу архітектуру, використовує сучасні вебтехнології та може бути розширена в майбутньому.

Таким чином, у третьому розділі було описано програмну реалізацію серверної та клієнтської частин, роботу з даними, інтерактивні компоненти, тестування функціональних можливостей і аналіз результатів. Розроблений вебзастосунок є завершеною початковою версією навчально-аналітичної платформи для гравців League of Legends.

Подальше вдосконалення вебзастосунку доцільно виконувати поетапно. Спочатку можна реалізувати автоматичне оновлення статичних відомостей через Data Dragon та отримання динамічних даних про матчі через офіційний Riot Games API [3, 15]. Після накопичення достатнього набору спостережень правила можуть бути доповнені статистичними коефіцієнтами або моделями машинного навчання, призначеними для рекомендування чемпіонів і прогнозування результатів матчів [4–6]. При цьому правиловий алгоритм доцільно зберегти як базовий рівень системи, що забезпечує роботу застосунку за відсутності зовнішніх даних.

Незалежно від подальшого ускладнення алгоритму система повинна зберігати зрозуміле пояснення кожної сформованої поради. Перспективним є додавання діалогового помічника, який перетворюватиме структурований результат аналізатора на стислий текст і відповідатиме на уточнювальні запитання користувача. Підходи до збереження контексту діалогу, комбінованого використання великих мовних моделей і формування відповідей на основі спеціалізованого вмісту розглянуто в роботах [37–39]. Водночас джерелом фактичних відомостей про чемпіонів, предмети та склад команд повинна залишатися перевірена структурована база даних, а мовна модель має виконувати лише функцію пояснення результату.

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений вебзастосунок з функціями перегляду характеристик чемпіонів та алгоритмічного формування рекомендацій у грі League of Legends. Розроблений програмний продукт поєднує довідкову, навчальну та аналітичну функції, що дозволяє користувачеві отримувати структуровану інформацію про чемпіонів, орієнтуватися на карті Summoner’s Rift та формувати базові рекомендації щодо вибору предметів і поведінки в матчі.

У першому розділі було проаналізовано предметну область League of Legends як командної онлайн-гри жанру MOBA. Визначено, що гравцеві необхідно працювати з великою кількістю взаємопов’язаних даних: ролями, позиціями на карті, характеристиками чемпіонів, предметами, складом союзної команди та складом противників. Обґрунтовано потребу у створенні вебзастосунку, який не лише подає довідкову інформацію, а й допомагає застосовувати її в конкретній ігровій ситуації. Також було сформовано основні функціональні вимоги до системи та визначено задачу розроблення.

У другому розділі було спроєктовано архітектуру вебзастосунку та алгоритм формування рекомендацій. Для реалізації обрано ASP.NET Core Razor Pages, мову C#, HTML, CSS, JavaScript і статичну JSON-базу даних. Такий технологічний стек забезпечив простоту запуску, зрозумілу структуру проєкту та можливість подальшого розширення. Було спроєктовано модель даних чемпіона, структуру вибору складів команд, основні програмні модулі, користувацький інтерфейс і правиловий алгоритм рекомендацій. Алгоритм базується на аналізі складу команд, класів чемпіонів, потенційних загроз і типових відповідей у вигляді предметів або порад щодо поведінки.

У третьому розділі було описано програмну реалізацію та тестування вебзастосунку. Серверна частина забезпечує завантаження даних, роботу з моделями, API-взаємодію та формування рекомендацій. Клієнтська частина реалізує головну сторінку, глобальну навігацію, інтерактивну карту, каталог

чемпіонів, модальні вікна, аналізатор матчу та збереження стану користувача за допомогою localStorage. У процесі тестування перевірено запуск проєкту, роботу сторінок, навігацію, фільтрацію чемпіонів, відкриття детальних карток, вибір складів команд і формування результату аналізу.

Практична цінність розробленого вебзастосунку полягає у створенні навчально-аналітичної платформи, яка допомагає гравцеві швидше орієнтуватися в ролях чемпіонів, їхніх характеристиках, сильних і слабких сторонах, рекомендованих предметах і базових ігрових рішеннях. Застосунок може бути корисним для користувачів, які починають знайомство з League of Legends, а також як демонстраційний проєкт для вивчення веброзробки, роботи з JSON-даними, клієнт-серверної взаємодії та правилових алгоритмів.

Поставлену мету роботи досягнуто: розроблено багатосторінковий вебзастосунок «LoL Champion Guide & Recommender», який містить головну сторінку, інтерактивну карту Summoner’s Rift, каталог чемпіонів із фільтрами, детальні картки характеристик та аналізатор матчів. У системі реалізовано алгоритмічне формування рекомендацій на основі складу команд і класів чемпіонів противника.

Розроблений програмний продукт має можливості для подальшого розвитку. У майбутньому доцільно розширити базу чемпіонів до повного набору персонажів, створити окрему базу предметів, підключити зовнішні джерела даних, додати вагові коефіцієнти для оцінювання загроз, реалізувати історію аналізів, персональні налаштування користувача та складнішу гібридну модель рекомендацій. Це дозволить підвищити точність порад і зробити вебзастосунок більш наближеним до повноцінної ігрової аналітичної системи.

Результати роботи апробовано у вигляді 1 тези доповіді під час XXIV Міжнародної науково-практичної конференції «Modern technologies and innovations as new tools for the development of science» [40].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Riot Games. League of Legends: How to play. URL: https://www.leagueoflegends.com/en-us/how-to-play/ (дата звернення: 15.06.2026).

2. Uta, M., Felfernig, A., Le, V.-M., Tran, T. N. T., Garber, D., Lubos, S., & Burgstaller, T. (2024). Knowledge-based recommender systems: Overview and research directions. Frontiers in Big Data, 7, Article 1304439. https://doi.org/10.3389/fdata.2024.1304439

3. Riot Games. Data Dragon. Riot Developer Portal. URL: https://developer.riotgames.com/docs/lol (дата звернення: 15.06.2026).

4. Błaszczyk, K. W., & Szajerman, D. (2023). Champion recommendation in League of Legends using machine learning. In J. Mikyška, C. de Mulatier, M. Paszynski, V. V. Krzhizhanovskaya, J. J. Dongarra, & P. M. A. Sloot (Eds.), Computational Science – ICCS 2023 (pp. 155–170). Springer. https://doi.org/10.1007/978-3-031-36027-5_12

5. Hitar-García, J. A., Morán-Fernández, L., & Bolón-Canedo, V. (2023). Machine learning methods for predicting League of Legends game outcome. IEEE Transactions on Games, 15(2), 171–181. https://doi.org/10.1109/TG.2022.3153086

6. Chowdhury, S., Ahsan, M., & Barraclough, P. (2025). Applications of linear and ensemble-based machine learning for predicting winning teams in League of Legends. Applied Sciences, 15(10), Article 5241. https://doi.org/10.3390/app15105241

7. Шафроненко, А. Ю., Бодянський, Є. В., & Руденко, Д. О. (2023). Модифікований рекурентний метод достовірної нечіткої кластеризації з використанням оптимізаційної процедури на основі косяків риб. Системи обробки інформації, 1(172), 92–96. https://doi.org/10.30748/soi.2023.172.11

8. Bodyanskiy, Y., Shafronenko, A., Rudenko, D., Polubiekhin, A., & Frolov, D. (2024). Online image segmentation using credibilistic fuzzy clustering.

In Proceedings of the 8th International Conference on Computational Linguistics and Intelligent Systems (pp. 1–10). CEUR Workshop Proceedings, 3664. https://ceur-ws.org/Vol-3664/paper1.pdf

9. Bodyanskiy, Y., Shafronenko, A., Rudenko, D., & Tanianskyi, O. (2024). Online stacking credibilistic fuzzy clustering for data stream mining. In Proceedings of the Seventh International Workshop on Computer Modeling and Intelligent Systems (pp. 340–349). CEUR Workshop Proceedings, 3702. https://ceur-ws.org/Vol-3702/paper28.pdf

10. Tvoroshenko, I., & Gorokhovatskyi, V. (2022). The application of hybrid intelligence systems for dynamic data analysis. International Journal of Engineering and Information Systems, 6(2), 40–48.

11. Mozilla contributors. Responsive web design. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Responsive_Design (дата звернення: 15.06.2026).

12. Mozilla contributors. Window: localStorage property. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage (дата звернення: 15.06.2026).

13. World Wide Web Consortium. Web Content Accessibility Guidelines (WCAG) 2.2. W3C Recommendation. URL: https://www.w3.org/TR/WCAG22/ (дата звернення: 15.06.2026).

14. Творошенко, І. С. (2021). Технології прийняття рішень в інформаційних системах: навчальний посібник. Харків: ХНУРЕ. https://doi.org/10.30837/978-966-659-294-4

15. Riot Games. Riot Developer Portal: Developer documentation. URL: https://developer.riotgames.com/docs/portal (дата звернення: 15.06.2026).

16. Microsoft. Introduction to Razor Pages in ASP.NET Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/ (дата звернення: 15.06.2026).

17. Microsoft. Static files in ASP.NET Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/static-files (дата звернення: 15.06.2026).

18. Microsoft. Dependency injection in ASP.NET Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection (дата звернення: 15.06.2026).

19. Microsoft. How to serialize and deserialize JSON in .NET. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/overview (дата звернення: 15.06.2026).

20. Гороховатський, В. О., & Творошенко, І. С. (2022). Аналіз багатовимірних даних за описом у формі множини компонент: монографія. Харків: ХНУРЕ. 124 с. https://doi.org/10.30837/978-966-659-379-8

21. Gorokhovatskyi, V., Tvoroshenko, I., Kobylin, O., & Vlasenko, N. (2023). Search for visual objects by request in the form of a cluster representation for the structural image description. Advances in Electrical and Electronic Engineering, 21(1), 19–27. https://doi.org/10.15598/aeee.v21i1.4661

22. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., Gadetska, S., & Al-Dhaifallah, M. (2023). Statistical data analysis models for determining the relevance of structural image descriptions. IEEE Access, 11, 126938–126949. https://doi.org/10.1109/ACCESS.2023.3332291

23. Gorokhovatskyi, V., Tvoroshenko, I., & Yakovleva, O. (2024). Transforming image descriptions as a set of descriptors to construct classification features. Indonesian Journal of Electrical Engineering and Computer Science, 33(1), 113–125. https://doi.org/10.11591/ijeecs.v33.i1.pp113-125

24. World Wide Web Consortium. Dialog (Modal) Pattern. WAI-ARIA Authoring Practices Guide. URL: https://www.w3.org/WAI/ARIA/apg/patterns/dialog-modal/ (дата звернення: 15.06.2026).

25. Руденко, Д. О., & Маренич, В. В. (2024). Дослідження методів розробки програмних додатків з інтегрованими API. У матеріалах II Міжнародної науково-практичної конференції «Scientific Research: Modern Challenges and Future Prospects» (23–25 вересня 2024 р., Мюнхен, Німеччина) (с. 144–147).

26. Mozilla contributors. Fetch API. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 15.06.2026).

27. Bray, T. (2017). The JavaScript Object Notation (JSON) Data Interchange Format (RFC 8259). Internet Engineering Task Force. https://doi.org/10.17487/RFC8259

28. Дацок, Є. О., Дацок, О. М., & Руденко, Д. О. (2025). Аналіз ефективності використання ORM Dapper та принципів SOLID у проєктуванні інформаційної системи медичного призначення. Вісник Національного технічного університету «ХПІ». Серія: Інформатика та моделювання, 2(14), 157–171. https://doi.org/10.20998/2411-0558.2025.02.11

29. Microsoft. Model Binding in ASP.NET Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/models/model-binding (дата звернення: 15.06.2026).

30. Microsoft. Model validation in ASP.NET Core MVC and Razor Pages. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/models/validation (дата звернення: 15.06.2026).

31. Руденко, Д. О., Лотвінова, В. В., & Безверха, Є. В. (2023). Навчання нейронної мережі в задачах обробки даних. Collection of Scientific Papers «SCIENTIA» (22 вересня 2023 р., Сінгапур, Республіка Сінгапур), 94–95.

32. Rudenko, D. O., Bezverkha, Y. V., & Lotvinova, V. V. (2023). Neural network recovery of omissions in data sets. In Proceedings of the VIII International Scientific Conference «Development of Science in the XXI Century» (14–15 September 2023, Dortmund, Germany) (pp. 87–88). https://doi.org/10.5281/zenodo.8355620

33. Артьомов, Д. О., & Руденко, Д. О. (2024). Огляд підходів до створення landing page. У матеріалах IV Міжнародної науково-практичної конференції «Scientific Research: Modern Challenges and Future Prospects» (18–20 листопада 2024 р., Мюнхен, Німеччина) (с. 135–138).

34. Gorokhovatskyi, V., Tvoroshenko, I., Yakovleva, O., Hudáková, M., & Gorokhovatskyi, O. (2024). Application a committee of Kohonen neural networks to training of image classifier based on description of descriptors set. IEEE Access, 12, 73376–73385. https://doi.org/10.1109/ACCESS.2024.3404371

35. Gorokhovatskyi, V., Tvoroshenko, I., Yakovleva, O., & Hudáková, M. (2025). Image description compression in classification structural methods. IEEE Access, 13, 43631–43641. https://doi.org/10.1109/ACCESS.2025.3548910

36. Gorokhovatskyi, V., Chmutov, Y., Tvoroshenko, I., & Kobylin, O. (2025). Reducing computational costs by compressing the structural description in image classification methods. Advanced Information Systems, 9(1), 5–12. https://doi.org/10.20998/2522-9052.2025.1.01

37. Bohdan, N., Tvoroshenko, I., Gorokhovatskyi, V., & Kobylin, O. (2025). Development of a hybrid method to enhance context memory for a chatbot application based on large language models. International Journal of Academic Information Systems Research, 9(10), 7–18.

38. Suprun, A., Tvoroshenko, I., Gorokhovatskyi, V., & Yakovleva, O. (2025). Development and research of a method for the combined use of large language models for text generation. International Journal of Academic and Applied Research, 9(10), 249–263.

39. Yakovleva, O., Nebeský, L., & Kirichenko, A. (2023). Using the GPT models for responses based on custom content to develop neural consultant for university applicants. In Proceedings of the V International Scientific and Practical Conference «Trends in Science Regarding the Creation of New Teaching Methods» (16–18 October 2023, Madrid, Spain) (pp. 172–178).

40. Яблунівський, А. С. (2026). Розроблення вебзастосунку для перегляду характеристик чемпіонів та формування рекомендацій у League of

Legends. XXIV Міжнародна науково-практична конференція «Modern technologies and innovations as new tools for the development of science»: Тези доповідей. Мюнхен. С. 92–94.