МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ВЕБЗАСТОСУНОК ВІКТОРИНИ ЗАСОБАМИ ASP.NET CORE»

Рівень «бакалавр»

Галузь знань 12 «Інформаційні технології»

Спеціальність 122 «Комп’ютерні науки»

Виконав здобувач 4 курсу

Солом’яний Олексій Миколайович

Керівник к.т.н., доцент

Трофименко Олена Григорівна

Рецензент к.пед.н.

Лобода Юлія Геннадіївна

Одеса – 2024

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ”
Факультет кібербезпеки та інформаційних технологій
Кафедра інформаційних технологій
Галузь знань 12 Інформаційні технології
Спеціальність 122 “Комп’ютерні науки”
Назва освітньої програми “Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних технологій
доцент Наталія Логінова_____
“____”_____ 2024 року

ЗАВДАННЯ
на кваліфікаційну (бакалаврську) роботу
здобувачу Солом’яному Олексію Миколайовичу

1. Тема роботи: “Вебзастосунок вікторини засобами ASP.NET CORE”
керівник роботи: к.т.н, доцент Трофименко Олега Григорівна
затверджені наказом НУ “ОЮА” № 809-06 від “02” квітня 2024 року
2. Строк подання здобувачем роботи “20” травня 2024 року
3. Дата видачі завдання “15” вересня 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз вимог до проєкту	15.09.2023 – 15.10.2023	
2	Аналіз конкурентів	16.10.2023 – 30.10.2023	
3	Розробка функціональних вимог	01.11.2023 – 14.11.2023	
4	Проєктування вебзастосунку	15.11.2023 – 20.12.2023	
5	Реалізація проєкту	21.12.2023 – 02.02.2024	
6	Реалізація сутностей та бази даних	03.02.2024 – 24.02.2024	
7	Аналіз результатів та висновки	25.02.2024 – 30.03.2024	
8	Оформлення пояснювальної записки	01.04.2024 – 17.05.2024	

Здобувач _____
(підпис) (ім’я та прізвище)
Керівник роботи _____
(підпис) (ім’я та прізвище)

АНОТАЦІЯ

Солом'яний О. М. «Вебзастосунок вікторини засобами ASP.NET CORE». – Кваліфікаційна робота бакалавра за спеціальністю 122 «Комп'ютерні науки», освітньою програмою «Комп'ютерні науки».

Кваліфікаційна робота викладена на 60 сторінках основного тексту та 66 сторінках загального тексту, містить 3 розділи, 38 рисунків, 10 таблиць, 3 додатки, 23 використаних джерела.

Метою роботи є розробка вебзастосунку для вікторини, який слугуватиме платформою для формування інтерактивного рейтингу на обрану тематику.

Об'єктом дослідження є специфіка розробки вебзастосунку вікторини та організація його інтерфейсу та функціоналу.

Предмет дослідження – засоби розробки вебзастосунку для вікторини засобами ASP.NET Core.

У кваліфікаційній роботі розроблено вебзастосунок для вікторини, для формування інтерактивного рейтингу на обрану тематику. Рейтинг буде формуватися в залежності від вибору користувачів вебзастосунку. Досліджено специфіку розробки вікторин та розробку функціоналу й інтерфейсу засобами ASP.NET Core.

Розроблений вебзастосунок може використовуватись для формування рейтингів в залежності від обраної тематики, які будуть актуальні на сьогоднішній день.

Ключові слова: вебзастосунок, вікторина, ASP.NET Core, інтерактивний рейтинг, авторизація.

ANNOTATION

Solomianyi, O. M. «Web application of the quiz by means of ASP.NET CORE». – Bachelor's thesis in the specialty 122 "Computer Science", educational program "Computer Science".

The thesis is presented on 60 pages of the main text and 66 pages of the general text, contains 3 sections, 3 figures, 10 tables, 3 appendices, 23 references.

The aim of the work is to develop a web application for the quiz, which will serve as a platform for the formation of an interactive rating on the selected topic.

The object of the research is the specifics of the development of the quiz web application and the organization of its interface and functionality.

The subject of the research is the tools for developing a web application for a quiz using ASP.NET Core. In the qualification work, a web application for a quiz has been developed to form an interactive rating on the selected topic. The rating will be formed depending on the choice of users of the web application. The specifics of the development of quizzes and the development of functionality and interface by means of ASP.NET Core are studied.

The developed web application can be used to generate ratings depending on the chosen topic, which will be relevant today.

Keywords: web application, quiz, ASP.NET Core, interactive rating, authorization.

ЗМІСТ

ВСТУП

В сучасному світі розвиток інформаційних технологій створює нові можливості для вдосконалення процесів взаємодії між користувачами та інформаційними системами. Однією з таких можливостей є розробка систем інтерактивних рейтингів, що дозволяють користувачам формувати та використовувати рейтингові списки для різних потреб. В цьому контексті, системи, що дозволяють створювати рейтинги на основі індивідуальних потреб користувача, набувають все більшої популярності. Коли власник вебсайту бажає створити платформу для оцінки та ранжування різних за тематикою рейтингів, така система дозволить користувачам не лише оцінювати різні варіанти, а й отримувати рекомендації на основі зібраних даних. Це підвищить не лише залученість користувачів, а й ефективність їхнього вибору, надаючи можливість враховувати різноманітні фактори та особисті вподобання.

Робота присвячена розробці системи формування інтерактивних рейтингів, яка дозволяє формувати та адаптувати рейтинги на основі потреб власника, зокрема, на прикладі системи рейтингу ідей для побачень.

Метою роботи є розробка вебзастосунку для вікторини, який слугуватиме платформою для формування інтерактивного рейтингу на обрану тематику.

Об’єкт дослідження – специфіка розробки вебзастосунку вікторини та організація його інтерфейсу та функціоналу.

Предмет дослідження – засоби розробки вебзастосунку для вікторини засобами ASP.NET Core.

Відповідно до мети завданнями кваліфікаційної роботи є:

– аналіз предметної області та конкурентів;

– формування функціональних вимог;

– проєктування вебзастосунку;

– розробка клієнтської, серверної частини та бази даних вебзастосунку онлайн-вікторини.

Методи досліджень для розв’язання завдань роботи:

– метод *аналізу* використано для дослідження аналогів ігрових вікторин, а також вибору сучасних засобів розробки;

– методи *аналогій* використовувалися при аналізі наявних застосунків із подібним функціоналом задля виявлення популярних тенденцій;

– методи *абстрагування* та *порівняння* використовувалися для аналізу та узагальнення можливостей, загроз, слабких та сильних сторін серед аналогів програмних застосунків для онлайн-вікторин;

– методи *спостереження*, *пояснень*, *аналізу*# використано під час проєктування архітектури, моделювання предметної області та розробки вебзастосунку, як засоби, що допомагають ґрунтовно охарактеризувати та дослідити той чи інший аспект програмної системи.

Практичне значення здобутих результатів полягає в можливості створення ефективної системи інтерактивних рейтингів, яка може бути адаптована для різноманітних сфер діяльності, зокрема для організації та планування побачень. Запропонована система дозволяє користувачам легко оцінювати та ранжувати варіанти на основі своїх особистих потреб і вподобань, що значно підвищує релевантність і корисність отриманих рекомендацій. Отримані результати можуть бути корисними для розробників програмного забезпечення, маркетологів, організаторів заходів та інших професіоналів, які зацікавлені у впровадженні інтерактивних систем оцінки та ранжування у своїй діяльності.

Результати роботи були апробовані у науковій публікації автора:

Солом’яний О. М. Веброзробка засобами ASP.NET Core. *Кіберпростір в умовах війни та глобальних викликів XXI століття: теорія та практика*: матер. всеукр. наук.-практ. конф. (м. Одеса, 24 листопада 2023 р.). Одеса, 2023. С. 185-187. DOI: 10.32837/11300.27179. URL: https://hdl.handle.net/11300/27179.

1 АНАЛІЗ ВИМОГ ДО ПРОЄКТУ

1.1 Специфіка розробки проєкту вікторини

Специфіка розробки – це сукупність характеристик, властивостей та вимог, які визначають процес розробки програмного забезпечення (вебзастосунку). Вона охоплює різні аспекти, включаючи технічні та організаційні, які впливають на створення кінцевого продукту.

Перелік вимог для онлайн-вікторини:

– *вікторина*: процес вибору авторизованим користувачем однієї з наведених карток;

– *реєстрація:* процес створення нового користувача та присвоєння ролі;

– *автентифікація:* процес підтвердження даних користувача підчас авторизації;

– *рейтинг:* сформований та відсортований набір карток;

– *формування рейтингу:* зміна позицій карток, залежно від вибору користувачів;

– *картка:* об’єкт вікторини, який містить опис картки;

Кожен з цих елементів важливий для створення зручного, ефективного інтерфейсу, який мотивує користувачів до активної участі у вікторині. Відповідне програмування та дизайн вебзастосунку мають велике значення для забезпечення привабливого, інтуїтивно зрозумілого користувацького досвіду.

1.2 Аналіз аналогів

Більшість аналогів у сфері вікторин – це вікторини у форматі опитування, у результаті яких формується остаточна відповідь для користувача (гравця) та статистика на основі відповідей інших користувачів.

Для визначення конкурентоспроможності та актуальності проєкту варто зробити порівняння з аналогами, визначити переваги та недоліки Для цього необхідно провести SWOT-аналіз [1] проєктів-конкурентів. Наразі є багато сайтів

з функціоналом для вікторин, більшість з яких є навчальними. У роботі наведено порівняльний аналіз декількох різних онлайн-вікторин: tezzt.com (табл. 1.1), wordwall.net (табл. 1.2), ukrerudyt.com (табл. 1.3).

tezzt.com – сайт для проходження тестів та вікторин на різні тематики [2].

Таблиця 1.1 – SWOT-аналіз вікторини tezzt.com

Сильні сторони	Слабкі сторони	Можливості	Ризики
Асортимент вікторин на різні теми.	Малий вибір тем для вікторин.	Необмежена можливість проходження вікторин.	Втрата актуальності розділу «вікторини» через неактивний розвиток розділу.
Обрання вікторини в залежності від тривалості вікторини.	Малий вибір вікторин за темами.		
Наявність рейтингу гравців.			

wordwall.net – онлайн-платформа для створення і проходження освітніх вікторин та тестування [3].

Таблиця 1.2 – SWOT-аналіз платформи wordwall.net

Сильні сторони	Слабкі сторони	Можливості	Ризики
Функціонал для створення опитувань.	Обмежені можливості за рахунок платної підписки.	Створення тестів та вікторин для навчальних закладів.	Наявність платної версії може відвернути частину цільової аудиторії.
Наявність шаблонів для створення вікторин.		Зручний редактор тестів та вікторин	
Наявність локалізації на 30+ мов.			

ukrerudyt.com – платформа для створення вікторин для навчального процесу [4].

Таблиця 1.3 – SWOT-аналіз рейтингу ukrerudyt.com

Сильні сторони	Слабкі сторони	Можливості	Ризики
Спеціалізований ресурс для опитувань.	Створення вікторин лише для учнів шкіл.	Розвиток функціоналу для вищих навчальних закладів.	За відсутності просунення ресурсу має обмежену кількість користувачів

Аналіз аналогів виявив недоліки різних за тематикою вікторин, що надало можливості для формування функціоналу вікторини, яка відповідатиме сучасним потребам користувачів. Так, SWOT-аналіз конкурентів дозволив визначити сильні

та слабкі сторони наявних рішень, що допомогло сформулювати ключові вимоги до проєкту: спеціалізована вікторина, актуальність матеріалу на сайті, відсутність платних функцій.

1.3 Розробка функціональних вимог до проєкту

Для успішної реалізації та подальшої підтримки необхідно визначити точні вимоги до функціоналу та оформлення проєкту, що дозволить уникнути можливих проблем під час реалізації. В табл. 1.4 визначено вимоги до функціоналу створюваної вікторини.

Таблиця 1.4 – Опис функціоналу створюваного вебзастосунку

Функція	**Опис**
Базовий функціонал	
Перегляд рейтингу	Кожен користувач може вільно переглянути актуальний рейтинг ідей для побачень
Випадкові картки	Кожен користувач може переглянути 2 випадкові ідеї для побачення, не залежно від рейтингу
Реєстрація	Кожен користувач може зареєструватися на сайті, для отримання додаткових можливостей
Авторизація	Кожен користувач може авторизуватися на сайті, для впливу на рейтинг.
Ролі	Кожен користувач має свою роль, яка дає йому змогу використовувати функціональні можливості в залежності від ролі.
Вікторина	
Вікторина	Кожен авторизований користувач має змогу пройти вікторину, за результатом якого буде формуватися рейтинг ідей для побачень.
Формування рейтингу	У процесі вікторини, картки, які вибрав користувач, будуть підніматися у рейтингу

Функціональні вимоги до проєкту охоплюють: реєстрацію та авторизацію користувачів, проходження вікторини для формування рейтингу, перегляд рейтингу, а також управління ролями користувачів.

В роботі було вибрано відповідні технології для клієнтської та серверної частин, щоб забезпечити гнучкість і функціональність проєкту.

1.4 Вибір засобів розробки

Для розробки будь-якого вебзастосунку варто чітко визначитися з технологіями, які будуть використовуватися для розробки клієнтської (front-end) та серверної частини (back-end).

Клієнтська частина або фронтенд (front-end) – це частина програмного забезпечення, яка відповідає за взаємодію з користувачем та подання даних та елементів інтерфейсу. Для створення клієнтської частини можна використовувати готові компоненти або рішення у бібліотеках та фреймворках або самостійно розробляти кожен компонент. Приклад елементів для клієнтської частини: кнопки, посилання, меню навігації, форми зв’язку тощо [7].

Серверна частина або бекенд (back-end) – це готова платформа для створення логіки сайту, яка формується з фреймворку та його компонентів, які використовуються для створення окремих операцій у вебзастосунку. Серверна частина також складається з окремих частин для роботи На стороні бекенду проходять усі операції, які користувач не бачить, приклад таких операціє є: переадресування, роутинг, реєстрація користувача, ролі тощо [8].

1.4.1 Засоби розробки серверної частини застосунку

Серверна частина або бекенд (back-end)– це частина майже будь якого вебзастосунку, на боці якої проходять усі операції на сайті, які не бачить користувач, а саме:

- реєстрація;
- авторизація;
- присвоєння ролей;
- робота з базою даних;
- тощо.

Для розробки кваліфікаційної роботи було обрано фреймворк від Microsoft – ASP.NET Core. Опис переваг та недоліків даної платформи приведено у табл. 1.5 нижче.

Таблиця 1.5 – Переваги та недоліки ASP.NET Core [9]

Переваги	Недоліки
Кросплатформеність: підтримка роботи на сучасних операційних системах.	Складність опанування фреймворку.
Висока продуктивність.	Залежність від Microsoft.
Модульність: фреймворк побудовано на модульній архітектурі.	Складність налаштування.
Підтримка клієнтських фреймворків.	Спільнота набагато менше, ніж у інших популярних фреймворків
Вбудована система захисту	Кількість бібліотек для використання.

ASP.NET Core – це потужний і модульний фреймворк від Microsoft для створення back-end вебзастосунків та сервісів. Використання ASP.NET Core для розробки серверної частини застосунку вікторини демонструє прогресивний підхід до використання новітніх технологій у веброзробці. ASP.NET Core підтримує розробку на C# та інших мовах .NET, забезпечуючи високу продуктивність завдяки оптимізованому коду та ефективному керуванню ресурсами, що є критично важливим для вебзастосунків, які обробляють великі обсяги даних та звернень користувачів [9, 10]. C# – це високорівнева об’єктно-орієнтована мова програмування, розроблена компанією Microsoft, яка використовується для створення різноманітних застосунків, від вебсайтів до десктопних програм та мобільних застосунків, зокрема, в рамках платформи .NET.

Microsoft SQL Server Management Studio – це реляційна система керування базами даних (СКБД), розроблена компанією Microsoft, яка використовується для зберігання та управління великими обсягами структурованих даних у корпоративних середовищах. Переваги та недоліки Microsoft SQL Server наведено в табл. 1.6.

Таблиця 1.6 – Переваги та недоліки Microsoft SQL Server

Переваги	Недоліки
Надійність та безпека.	Вартість ліцензії.
Висока продуктивність.	Залежність від платформи.
Інтеграція з продуктами Microsoft.	Ресурсозатратність.
Інструменти адміністрування та розробки.	Складність адміністрування.

Entity Framework (EF) Core є легковаговою, розширюваною, open source версією популярного ORM від Microsoft, яка дозволяє розробникам .NET працювати з базами даних за допомогою .NET об'єктів. Переваги та недоліки Entity Framework Core наведено в табл. 1.7.

Таблиця 1.7 – переваги та недоліки Entity Framework Core

Переваги	**Недоліки**
Надійність та безпека.	Вартість ліцензії.
Висока продуктивність.	Залежність від платформи.
Інтеграція з продуктами Microsoft.	Ресурсозатратність.
Інструменти адміністрування та розробки.	Складність адміністрування.

JetBrains Rider – середа розробки створена виключно для специфіки мови C# та платформи .NET та компонентів, які пов'язані з розробкою під платформу .NET.

Тож вибір ASP.NET Core обгрунтований його кросплатформенністю, високою продуктивністю, модульністю, підтримкою клієнтських фреймворків та вбудованою системою захисту, що є ключовими для створення надійного та ефективного вебзастосунку.

Є деякі виклики, такі як складність освоєння фреймворку, його залежність від Microsoft, складність налаштувань та відносно меншу спільноту порівняно з іншими популярними фреймворками. переваги значно перевищують можливі недоліки, особливо з огляду на інтеграцію з іншими продуктами Microsoft, яка є важливою для корпоративних середовищ.

Додаткові інструменти, такі як Microsoft SQL Server та Entity Framework Core, сприяють ефективній роботі з даними, та JetBrains Rider, який оптимізує процес розробки. Ці інструменти допомагають створити потужну та ефективну розробку, що забезпечує високу надійність і продуктивність вебзастосунку.

1.4.2 Засоби розробки клієнтської частини

Фронтенд (front-end) або клієнтська частина програмного забезпечення відповідає за взаємодію з користувачем та подання інформації. У веброзробці фронтенд стосується компонентів, які відображаються у браузері користувача і з

якими він може взаємодіяти. Фронтенд відповідає за те, щоб вебсторінки були зручними та привабливими для користувачів, а взаємодія із застосунком була інтуїтивно зрозумілою та ефективною.

Основні засоби розробки фронтенду для створюваного застосунку:

1. HTML (Hypertext Markup Language) – мова розмітки, яка використовується для створення структури сторінок вебсайту;

2. CSS (Cascading Style Sheets) – каскадні таблиці стилів, що відповідають за оформлення і зовнішній вигляд елементів HTML;

3. JavaScript – мова програмування, яка використовується для спілкування клієнтської частини із серверною;

4. бібліотека Bootstrap – набір готових рішень для створення адаптивного інтерфейсу для вебзастосунків, чим спрощує розробку клієнтської частини.

Bootstrap – бібліотека класів для розробки простих, зручних, адаптованих інтерфейсів під різні пристрої. Вона має набір CSS і JavaScript компонентів, які полегшують створення сучасних вебсторінок і вебдодатків. Переваги та можливі недоліки наведено в табл. 1.8 [11].

Таблиця 1.8 – Переваги та недоліки Bootstrap

Переваги	**Недоліки**
Bootstrap надає готові UI-компоненти, які дозволяють швидко створювати стильні вебсторінки.	Можливість використання готових компонентів може призвести до того, що дизайн сайту буде схожий на інші сайти, які також використовують Bootstrap.
Можливість створення адаптивного та мобільно-дружнього дизайну без зайвих зусиль.	Не завжди всі компоненти та функціонал, які надає Bootstrap, потрібні для конкретного проєкту, що може зробити його великим та повільним у завантаженні.
Сумісність із різними браузерами, що дозволяє забезпечити однаковий вигляд сторінки у різних браузерах і на різних пристроях користувачів.	Для досягнення унікального дизайну часто потрібно додатково налаштовувати та кастомізувати компоненти Bootstrap, що може забрати багато часу та зусиль.

Фронтенд відіграє вирішальну роль у взаємодії користувачів з вебсторінками. Він забезпечує не лише візуальне подання контенту через HTML і CSS, а й інтерактивність за допомогою JavaScript. Одним із популярних інструментів для полегшення розробки фронтенду є бібліотека Bootstrap, яка допомагає створювати

адаптивні, стильні інтерфейси. Хоча Bootstrap пропонує широкий асортимент готових UI-компонентів, що спрощують процес розробки, це може призвести до одноманітності в дизайні. Крім того, використання цієї бібліотеки може збільшити загальний розмір проєкту, що негативно позначиться на швидкості завантаження сторінок. Тому, попри значні переваги, при виборі Bootstrap потрібно враховувати всі можливі недоліки і специфіку проєкту для досягнення оптимального результату.

1.5 Маркетингові інструменти в створенні та просуванні програмного продукту

Аналіз цільової аудиторії є ключовим етапом при плануванні та розробці вебзастосунку, оскільки дозволяє краще зрозуміти потреби, бажання та поведінку потенційних користувачів. Це забезпечує створення продукту, який відповідає очікуванням користувачів і має високу конкурентоспроможність на ринку.

Основні сегменти цільової аудиторії

1. Студенти

Опис: основний сегмент аудиторії складають студенти та молодь у віці від 18 до 25 років, які активно користуються інтернетом та мають інтерес до інтерактивних та розважальних онлайн-активностей.

Потреби та очікування: молодь шукає цікаві та захоплюючі способи провести час онлайн, тому інтерактивність та елементи змагання у квізах дуже привабливі для них. Вони також очікують простоту у використанні та швидкість доступу до контенту.

Маркетингові стратегії: реклама в соціальних мережах (Instagram, TikTok, Facebook), співпраця з популярними блогерами та інфлюенсерами, організація онлайн-конкурсів та викликів.

2. Люди середнього віку

Опис: сегмент людей у віці від 26 до 40 років, які мають стабільний доступ до інтернету та зацікавлені в інтелектуальних та розважальних заходах.

Потреби та очікування: вони шукають способи відпочити після роботи, отримати нові знання або випробувати свої інтелектуальні здібності. Важливі функції включають зручний інтерфейс, можливість грати разом з друзями чи родиною.

Маркетингові стратегії: контент-маркетинг у блогах та на тематичних форумах, таргетована реклама у соціальних мережах (Facebook, LinkedIn), організація вікторин та інтелектуальних турнірів.

3. Інтелектуали та любителі вікторин

Опис: група людей різного віку, які захоплюються квізами, головоломками та інтелектуальними іграми.

Потреби та очікування: Цей сегмент шукає складні, але цікаві завдання, які допоможуть розвивати їхні навички та знання. Важливим є наявність різноманітних тем і рівнів складності, а також можливість змагатися з іншими.

Маркетингові стратегії: співпраця з платформами, що спеціалізуються на інтелектуальних іграх, просування через тематичні спільноти та форуми, проведення регулярних турнірів та змагань.

4. Освітні заклади

Опис: Школи, коледжі та університети, які можуть використовувати вебзастосунок для освітніх цілей.

Потреби та очікування: освітні заклади шукають інструменти для інтерактивного навчання, які можуть зацікавити студентів та підвищити їхню мотивацію. Важливими є можливості для створення власних квізів та оцінювання результатів.

Маркетингові стратегії: презентації та демонстрації продукту для освітніх установ, розробка спеціальних освітніх програм та матеріалів, співпраця з педагогами та адміністрацією закладів.

Потреби та бажання цільової аудиторії

1. Інтерактивність та залученість

Користувачі очікують, що вебзастосунок буде інтерактивним та захоплюючим. Важливо забезпечити можливість взаємодії з контентом та іншими користувачами, що підвищує рівень залученості.

2. Доступність та зручність використання

Важливими є простий та інтуїтивно зрозумілий інтерфейс, швидкий доступ до контенту та адаптивний дизайн, який дозволяє зручно користуватися застосунком на різних пристроях.

3. Різноманітність контенту

Користувачі очікують, що вікторини будуть різноманітними за темами та рівнями складності. Це дозволяє задовольнити інтереси різних категорій користувачів та підтримувати їхню зацікавленість.

4. Соціальні функції

Можливість змагатися з друзями, обмінюватися результатами та досягненнями в соціальних мережах є важливою для підвищення залученості та популярності вебзастосунку.

5. Освітня цінність

Для освітніх закладів важливо, щоб вебзастосунок мав освітню цінність та сприяв розвитку знань та навичок студентів. Це може включати можливість створення спеціалізованих освітніх квізів та інтеграцію з навчальними програмами.

Ключовими елементами стратегії є оптимізація для пошукових систем (SEO) та маркетинг у соціальних мережах (SMM). Розглянемо детально кожний з цих аспектів.

Оптимізація для пошукових систем (Search Engine Optimization) є однією з найважливіших складових стратегії просування вебзастосунку. Перш за все, необхідно забезпечити, щоб зміст і структура сайту були належним чином налаштовані для пошукових систем. Це включає використання релевантних ключових слів, які відповідають пошуковим запитам потенційних користувачів. Важливо створювати якісний контент, який не тільки приваблює користувачів, але й відповідає їхнім запитам, забезпечуючи високу інформативність і актуальність.

Метадані відіграють ключову роль у SEO. Правильно налаштовані мета-теги, заголовки та описи дозволяють підвищити видимість сайту в пошукових системах. Це означає, що кожна сторінка сайту повинна мати унікальний заголовок та опис, які чітко відображають її зміст і містять ключові слова.

Крім того, внутрішні та зовнішні посилання є важливими для підвищення авторитету сайту. Внутрішні посилання допомагають користувачам і пошуковим системам краще орієнтуватися на сайті, а зовнішні посилання, особливо з авторитетних ресурсів, підвищують довіру до сайту з боку пошукових систем.

Мобільна оптимізація також є важливою складовою SEO-стратегії. У сучасному світі більшість користувачів використовують мобільні пристрої для доступу до інтернету, тому сайт повинен завантажуватися швидко та бути зручним у використанні на всіх типах пристроїв. Це вимагає адаптивного дизайну та оптимізації швидкості завантаження.

Аналітика грає вирішальну роль у визначенні ефективності SEO-стратегії. Використання інструментів для відстеження та аналізу трафіку на сайті дозволяє зрозуміти, які стратегії працюють найкраще, і вносити необхідні корективи для покращення видимості та залучення більшої кількості користувачів.

Маркетинг у соціальних мережах (Social Media Marketing) є не менш важливим аспектом просування вебзастосунку. Соціальні мережі надають унікальні можливості для взаємодії з цільовою аудиторією та залучення нових користувачів. Створення та активне ведення сторінок у популярних соціальних мережах, таких як Facebook, Instagram, Twitter та LinkedIn, є першим кроком до побудови сильної присутності в соціальних медіа.

Контент-маркетинг є центральним елементом SMM-стратегії. Він включає розробку та публікацію якісного контенту, який цікавить цільову аудиторію. Це можуть бути пости, статті, відео та інший медіа-контент, який сприяє взаємодії з користувачами. Важливо, щоб контент був не тільки інформативним, але й розважальним та інтерактивним.

Реклама в соціальних мережах дозволяє ефективно залучати нових користувачів та збільшувати охоплення. Використання таргетованої реклами

дозволяє досягти саме тієї аудиторії, яка найбільш зацікавлена у вашому продукті. Це може включати як звичайні рекламні оголошення, так і спеціальні акції або пропозиції.

Інтерактивні елементи, такі як вікторини, опитування та конкурси, сприяють підвищенню залученості аудиторії. Вони не тільки збільшують взаємодію з користувачами, але й створюють позитивний імідж бренду, роблячи його більш привабливим та запам'ятовуваним.

Аналіз та звітність є важливою частиною SMM-стратегії. Використання аналітичних інструментів дозволяє відстежувати ефективність кампаній, розуміти, що працює найкраще, і вносити необхідні корективи. Це дозволяє оптимізувати SMM-стратегію та забезпечити максимальну ефективність рекламних зусиль.

Аналіз цільової аудиторії є критично важливим для успішного планування та розробки вебзастосунків, оскільки він дозволяє розуміти та задовольняти потреби користувачів, що в кінцевому підсумку забезпечує високий рівень задоволеності та лояльності. Розуміння вікових груп, їхніх інтересів та поведінкових паттернів дозволяє створити більш ефективні маркетингові стратегії, які спрямовані на максимально точне досягнення та залучення користувачів. Наприклад, залучення студентів через соціальні медіа та блогерів, організація інтелектуальних ігор для людей середнього віку, а також співпраця з освітніми закладами для використання вебзастосунків у навчальних цілях. Важливим є створення інтерактивного та залучаючого контенту, який є доступним та зручним для користування на різних пристроях. Крім того, використання SEO та SMM стратегій сприяє підвищенню видимості вебзастосунку та його популярності серед цільової аудиторії, забезпечуючи таким чином конкурентну перевагу на ринку.

1.6 Висновки до розділу 1

У розділі проаналізовано вимоги до проєкту вебзастосунку вікторини на основі ASP.NET Core.

Аналіз аналогів та SWOT-аналіз конкурентних вікторин виявили недоліки наявних рішень, що дозволило визначити ключові вимоги для створення вікторини,

яка відповідає сучасним потребам користувачів. Використання ASP.NET Core обґрунтоване його кросплатформністю та високою продуктивністю, а також інтеграцією з іншими продуктами Microsoft, що є важливим у корпоративних середовищах. Попри залежність від Microsoft та невелику спільноту, порівняно з іншими фреймворками, переваги ASP.NET Core значно перевищують його можливі недоліки. Важливо також відзначити роль Bootstrap для фронтенд-розробки і створення адаптивних, стильних інтерфейсів.

Аналіз цільової аудиторії виявився критично важливим для планування та розробки, дозволяючи задовольнити потреби користувачів та забезпечити високий рівень їхньої задоволеності та лояльності. Використання маркетингових стратегій SEO та SMM сприятиме підвищенню видимості та популярності вебзастосунку, забезпечуючи йому конкурентну перевагу на ринку. Всі ці елементи разом формують міцну основу для створення ефективного вебзастосунку, що відповідатиме сучасним вимогам та викликам.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ

2.1 Специфіка проєктування

Для вебзастосунків, розроблених засобами ASP.NET Core, є підхід проєктування Domain-Driven Design (DDD) [12] та шарова архітектура (Layered Architecture)] [13]. Використання цих концепцій дозволять розробити застосунок, який відповідатиме реальним потребам користувачів. Окремі шари проєкту виконуватимуть свої задачі. Вони не пов’язані з іншими шарами на пряму, обмежуючи доступ від шару до шару.

2.1.1 Domain-Driven Design

Domain-Drive Design (DDD) або предметно-орієнтоване програмування – це філософія розробки програмного забезпечення, яка орієнтована на важливість розуміння та моделювання сфери бізнесу. Ця стратегія спрямована на покращення якості програмного забезпечення шляхом його більшої відповідності потребам бізнесу, які воно обслуговує [12].

При створенні вебзастосунку з використанням DDD проєкт ділиться на головний проєкт та додаткові шари у вигляді бібліотек, а саме: Domain, Infrastructure (My101Romance, Data Access Layer, API) та Services (рис. 2.1) [12].

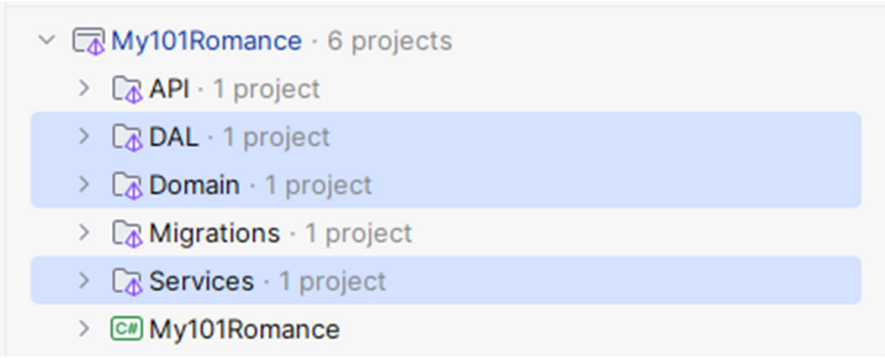

Рисунок 2.1 – Структура Domain-Drive Design. Domain, Infrastructure, Services

Головний проєкт має назву My101Romance та містить точку запуску вебзастосунку. За термінологією DDD Infrastructure – це шар в який входить бібліотеки API, DAL, Migrations.

Domain – це шар, який відповідає за Domain (сутності), а Services відповідає за логіку роботи вебзастосунку.

2.1.2 Багаторівнева архітектура застосунку

Багатошарова (багаторівнева) архітектура (Layered Architecture) – це архітектура, при використання якої застосунок ділиться на рівні, кожен з яких відповідає за своє завдання. Ця архітектура дозволяє зробити застосунок більш гнучким. У Layered Architecture зазвичай використовуються такі шари [13]:

1. DAL (Data Access Layer) – шар, який відповідає за доступ до даних та взаємодію з базою даних. Він має класи і методи для виконання запитів до бази даних, вставки, оновлення та видалення даних;
2. Domain Layer (Шар домена) – шар формує доменну модель застосунку, включаючи сутності, агрегати, бізнес-правила та логіку. Тут визначається основна функціональність застосунку;
3. Services: Шар сервісів використовується для виконання логіки, яка не підходить для розміщення у доменному шарі. Це може поєднувати сервіси для обробки запитів користувачів, логіки взаємодії з даними, обробки подій тощо.
4. API (Application Programming Interface) використовується для взаємодії з зовнішніми системами чи клієнтами. Це може бути вебAPI, яке надає доступ до функціональності застосовування через мережу [13].

API може бути частиною архітектури застосунку, але він не є безпосередньою складовою Layered Architecture.

2.1.3 Model-View-Controller

При розробці проєкту використано підхід для побудови застосунку DDD, шарову архітектуру та патерн Model-View-Controller. Всі вони поєднанні між собою. Весь проєкт поділено на декілька шарів, де кожен виконує свою логіку та не має прямого доступу до даних інших шарів.

За філософією DDD проєкт має три рівні [12]:

- доменний рівень (Domain);

– рівень інфраструктури (My101Romance, Data Access Layer, API);

– рівень подання (Services).

DDD, як і шарова архітектура, ділить проєкт на додаткові шари, тому в шаровій архітектурі є ще окремі рівні:

– My101Romance – головний проєкт;

– Data Access Layer (DAL) – рівень доступу до даних (взаємодія з базою даних);

– API.

Model-View-Controller (MVC) [14] – це архітектурний шаблон, який використовується для розробки програмного забезпечення та застосунків. У MVC система поділяють на три основні компоненти (рис. 2.2).

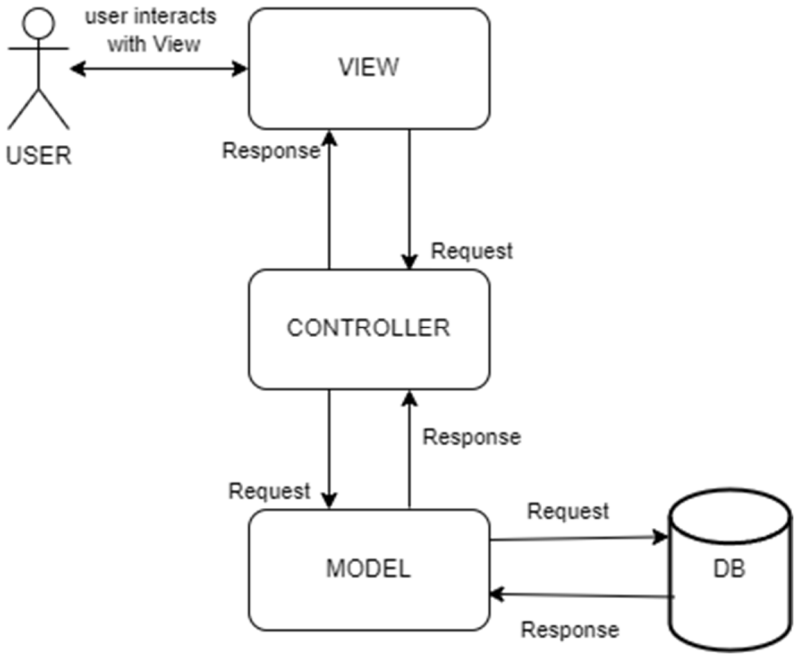

Рисунок 2.2 – Model-View-Controller

Модель (Model) відповідає за управління даними та логікою програми. Сюди входять об'єкти даних, класи, які забезпечують доступ до бази даних, та логіка, що визначає, як дані обробляються та змінюються. У поєднанні з DDD та шаровою архітектурою шар Model перетворюється на шари Domain, DAL, Services.

Подання (View) подає інтерфейс користувача, який відображає дані та сприймає дії користувача. Сюди входить все, що стосується клієнтської частини, подання усієї інформації.

Контролер (Controller) керує взаємодією між Моделлю та Подання. Він відповідає за оброблення запитів користувача, виклик методів Моделі для отримання чи змінення даних та відправлення відповіді до View. Зберігається в одному шарі з View [14].

2.2 Моделювання програмної системи

Моделювання програмної системи є ключовим етапом у розробці вебзастосунків, оскільки воно дозволяє детально продумати архітектуру, визначити основні компоненти і взаємодії між ними. В цьому розділі ми зосередимося на структурному та функціональному моделюванні вебзастосунку вікторини, що включає в себе створення діаграм класів, використання випадків і компонентів, які визначають як вебзастосунок буде реагувати на користувацькі запити та обробляти дані. Моделювання відіграє важливу роль у забезпеченні високої якості кінцевого продукту та сприяє ефективній комунікації між членами команди розробників. Особлива увага буде приділена вибору архітектурного стилю, забезпеченню масштабованості та безпеки системи, що є критичним для вебзастосунків, доступних в інтернеті.

2.2.1 Сценарії використання

Unified Modeling Language (UML) – це мова візуального моделювання, використовувана для опису, проєктування та документування програмних систем. UML забезпечує набір графічних нотаток для створення діаграм класів, діаграм випадків використання, діаграм послідовностей, діаграм станів та інших типів діаграм, які допомагають інженерам програмного забезпечення зрозуміти, проєктувати та документувати структуру і поведінку системи. UML широко застосовується в методологіях розробки програмного забезпечення, таких як об'єктно-орієнтоване програмування і Domain-Driven Design (DDD) [15].

Діаграма сценаріїв використання (UseCase) для вебзастосунків описує сценарії використання та можливості користувачів в залежності від їх ролі [16].

Діаграма на рис. 2.3 має 4 дійові особи: гість (guest), користувач (user), адміністратор (admin) та суперадміністратор (root).

Гість – неавторизований користувач, який немає прав на взаємодію з вікториною, має права лише на перегляд рейтингу та випадкових карток.

Користувач – авторизований користувач, який має право на проходження вікторини.

Адміністратор – привілейований користувач, який має право на створення нових карток.

Суперадміністратор – привілейований користувач з найвищим рівнем доступом, не обмежений у правах. Має змогу створювати нових адміністраторів. Може бути 1 у системі.

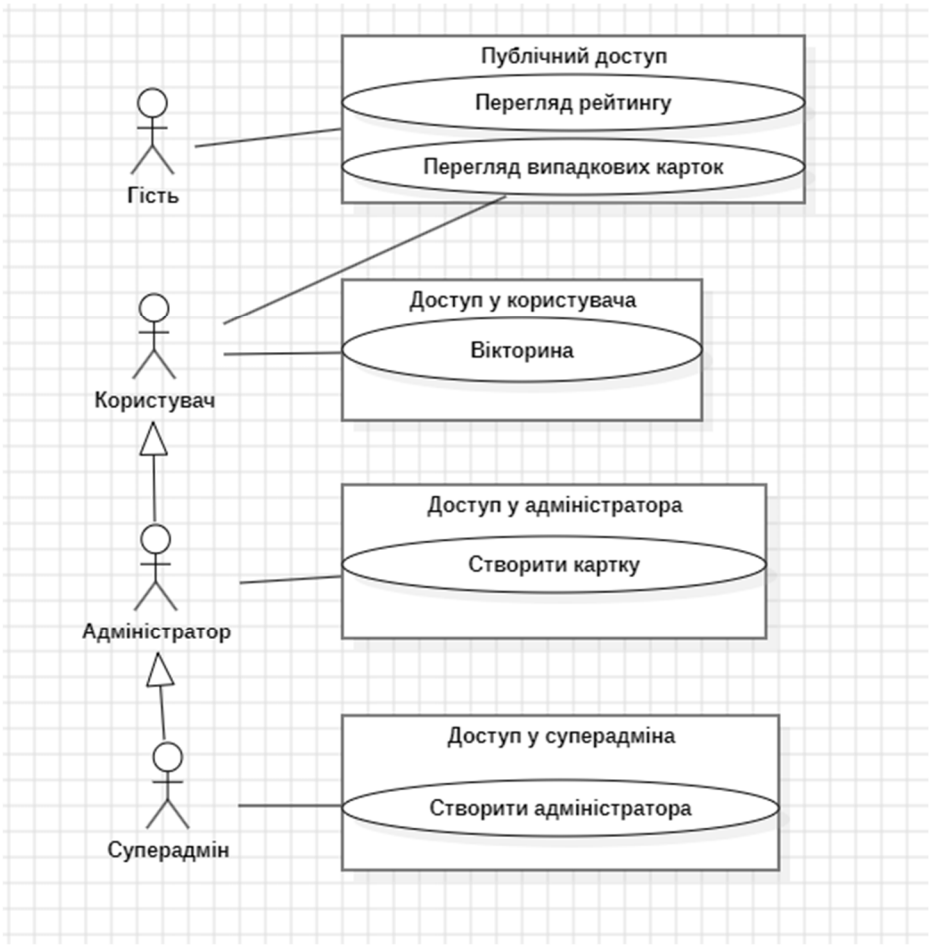

Рисунок 2.3 – UseCase діаграма

Кожен з них має свої можливості та права, наприклад, суперадміністратор може переглядати випадкові картки, дивитися рейтинг, додавати нових адміністраторів, додавати картки та грати у вікторину. Гість може лише дивитися випадкові картки або рейтинг, а зареєстрований користувач може брати участь у вікторині. Адміністратор може виконувати всі ці активності разом, окрім реєстрації нових адміністраторів.

2.2.2 MindMap

MindMap (ментальна карта) – це візуальний інструмент для організації інформації, який використовується для генерації, структурування та класифікації ідей. Цей метод часто використовується для мозкового штурму, планування проєктів, навчання та прийняття рішень [17]. На рис. 2.4 зображено MindMap для DDD.

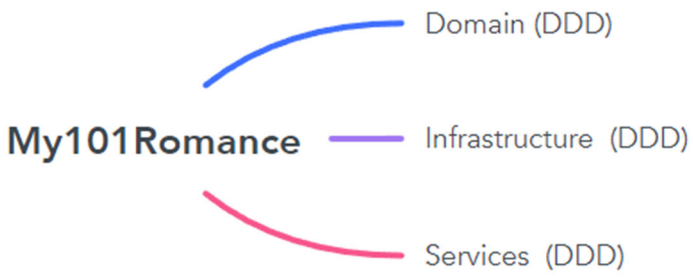

Рисунок 2.4 – MindMap ієрархії DDD

Структуру проєкту зображено у вигляді MindMap на рис. 2.5.

Рисунок 2.5 – Загальна MindMap проєкту

На рис. 2.6 – 2.8 деталізовано MindMap для кожної з гілок структури застосунку.

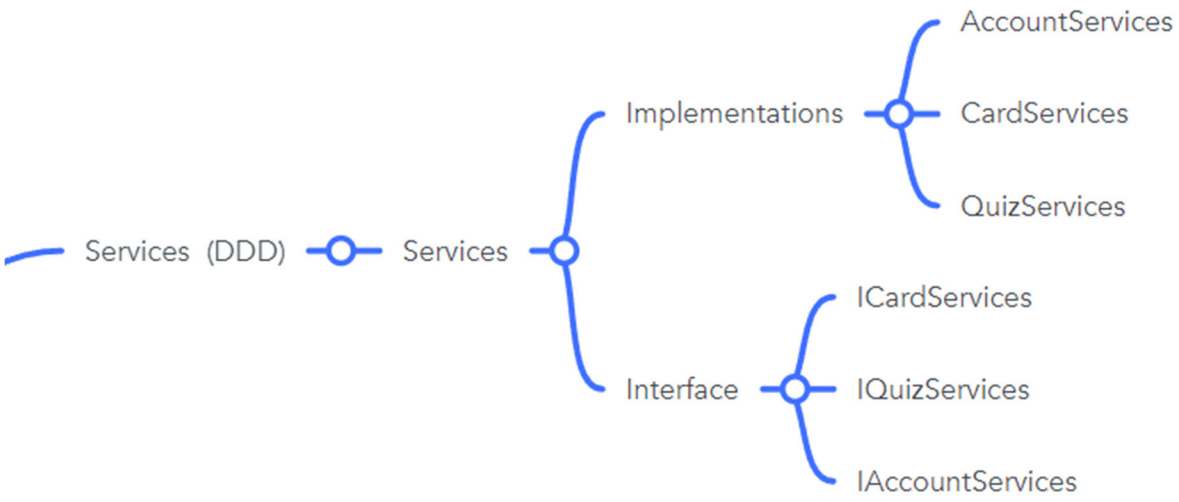

Рисунок 2.6 – MindMap Services

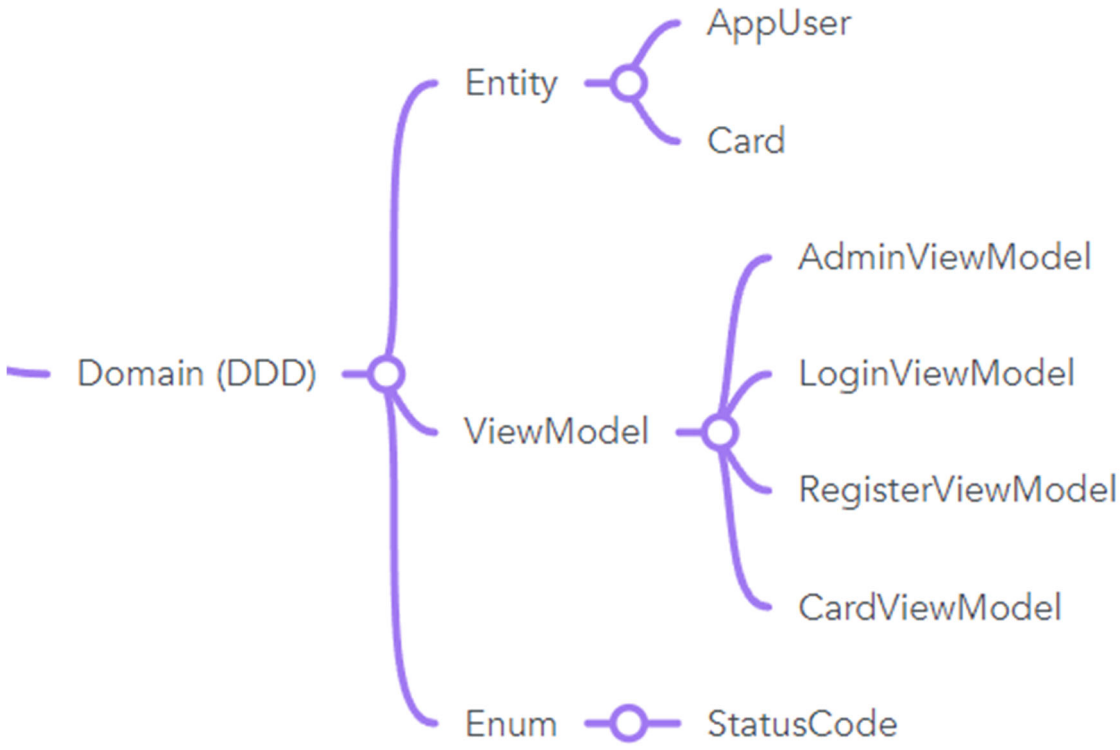

Рисунок 2.7 – MindMap Domain

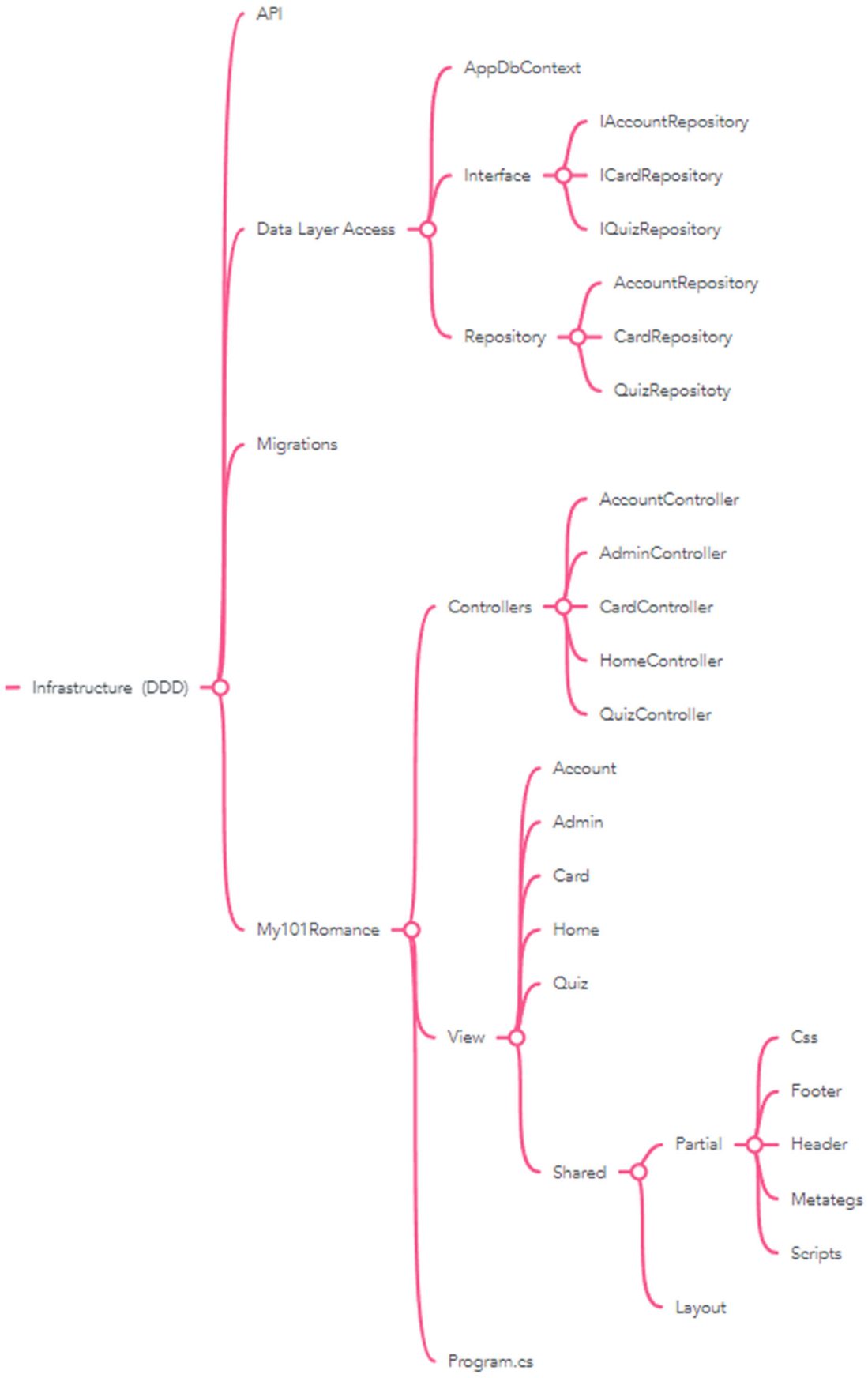

Рисунок 2.8 – MindMap Infrastructure

Завдяки MindMap, можна побачити графічне поєднання DDD та шарової архітектури, а також з чого складається кожен шар.

2.2.3 ER-діаграма

ER-діаграма (ERD, Entity-Relationship Diagram) – це графічне подання сутностей, атрибутів і зв'язків між ними, яке використовується для моделювання структури баз даних. ER-діаграми допомагають розробникам і аналітикам зрозуміти логічну структуру даних і їх взаємозв'язки в системі. ER-діаграму зображено на рис 2.9.

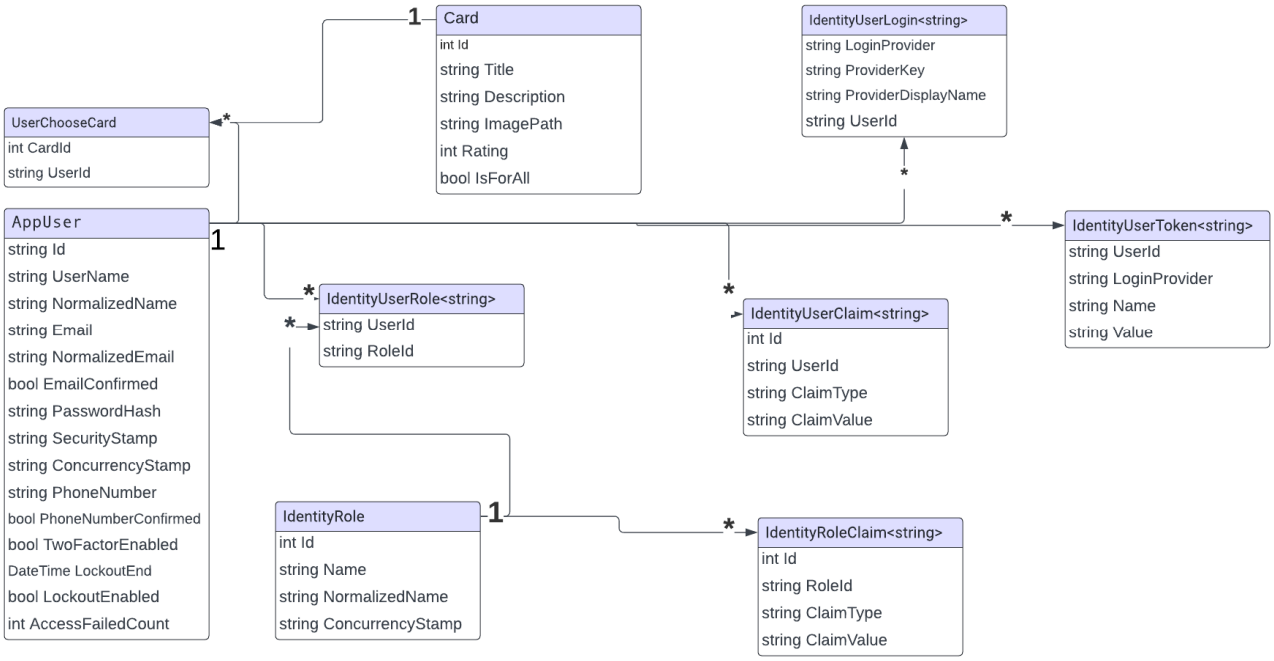

Рисунок 2.9 – ER-діаграма бази даних

Таблиця 2.1 – Типи зв’язків у базі даних

Таблиця А	Таблиця Б	Тип зв’язку
AppUser	IdentityUserRole	Один до багатьох
IdentityRole	IdentityUserRole	Один до багатьох
AppUser	IdentityUserClaim	Один до багатьох
AppUser	IdentityRoleClaim	Один до багатьох
AppUser	IdentityUserToken	Один до багатьох
AppUser	IdentityUserLogin	Один до багатьох
Card	UserChooseCard	Один до багатьох
AppUser	UserChooseCard	Один до багатьох

ER-діаграма бази даних для кваліфікаційної роботи складається з 9 таблиць:

1. AppUser:
   - string Id;
   - string UserName;
   - string NormalizedUserName;
   - string Email;
   - string NormalizedEmail;
   - bool EmailConfirmed;
   - string PasswordHash;
   - string SecurityStamp;
   - string ConcurrencyStamp;
   - string PhoneNumber;
   - bool PhoneNumberConfirmed;
   - bool TwoFactorEnabled;
   - DateTimeOffset LockoutEnd;
   - bool LockoutEnabled;
   - int AccessFailedCount.
2. IdentityRole:
   - string Id;
   - string Name;
   - string NormalizedName;
   - string ConcurrencyStamp.
3. IdentityUserRole:
   - string UserId;
   - string RoleId.
4. IdentityUserClaim:
   - int Id;
   - string UserId;

- string ClaimType;
- string ClaimValue.

5. IdentityUserLogin:
   - string LoginProvider;
   - string ProviderKey;
   - string ProviderDisplayName;
   - string UserId.
6. IdentityRoleClaim:
   - int Id;
   - string RoleId;
   - string ClaimType;
   - string ClaimValue.
7. IdentityUserToken:
   - string UserId;
   - string LoginProvider;
   - string Name;
   - string Value.
8. Card:
   - int Id;
   - string Title;
   - string Description;
   - string ImagePath;
   - int Rating;
   - bool IsForAll.
9. UseChooseCard:
   - int CardId;
   - string UserId.

2.3 Висновки до розділу 2

У розділі проведено огляд технологій та інструментів, які використовувалися для розробки вебзастосунку. Використання сучасних технологій HTML, CSS, Bootstrap для фронтенду та ASP.NET Core для бекенду дозволило створити високонавантажений вебзастосунок з інтерактивними елементами.

Архітектура проєкту має DDD, шарову архітектуру та паттерн Model-View-Controller (MVC). Вона забезпечує модульність і легкість у підтримці та подальшому розвитку системи.

Створено MindMap та діаграму сценаріїв використання (UseCase), що описує можливості користувачів та їх ролі в системі. Це допомогло визначити функціональні вимоги і забезпечити користувачам інтуїтивно зрозумілий та ефективний інтерфейс.

3 РЕАЛІЗАЦІЯ ПРОЄКТУ

3.1 Реалізація MVC, Domain-Drive Design, Layered Architecture

При першому запуску IDE JetBrains Rider буде запропоновано активізувати ліцензію або придбати її. Було використано навчальну ліцензію за допомогою університетською пошти @onua.edu.ua [18].

У діалоговому вікні вибираємо «New solution», після вибираємо тип проєкту, а саме «Web» і налаштовуємо. Проєкт отримав назву «My101Romance». Назва проєкту (Project name) та назва рішення (Solution name) мають збігатися. Вибираємо шлях, де буде зберігатися проєкт, актуальну версію .NET (target framework), мову (C#) та шаблон (template) «Web App (Model-View-Controller)» і вибираємо «create». Діалогове вікно New Solution зображено на рис. 3.1.

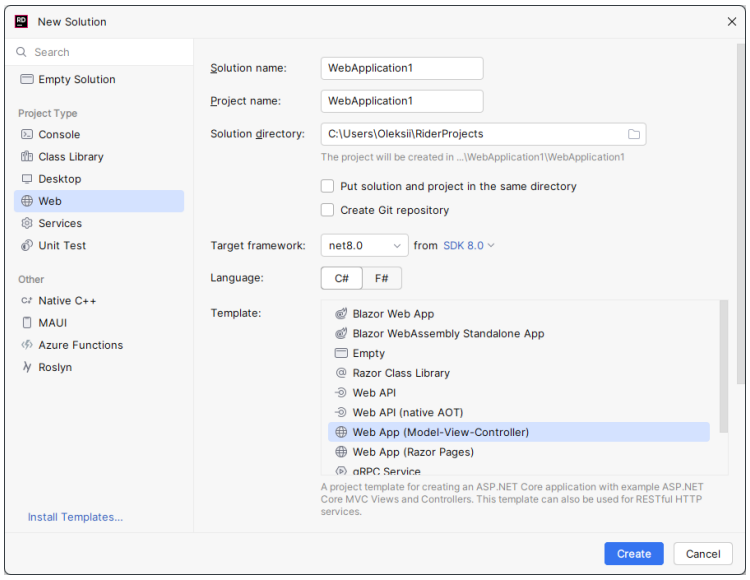

Рисунок 3.1 – New Solution

Після чого створюється пустий шаблон MVC, який необхідно привести до DDD та шарової архітектури.

Складові проєкту: Properties, Wwwroot, Controllers, Models, View, appsetings.json, appsettings.Development.json, Program.cs. На рис. 3.2 зображено усі файли проєкту.

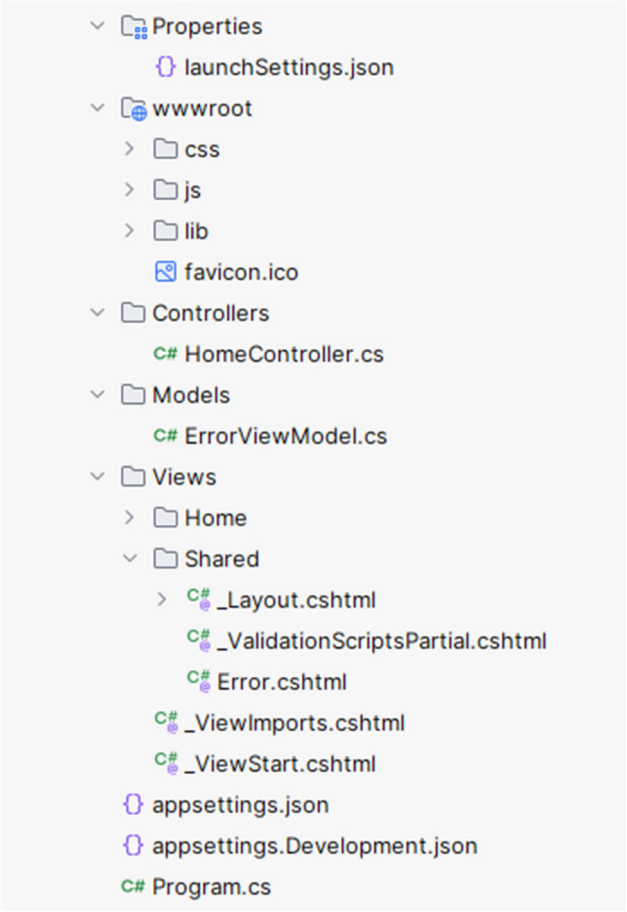

Рисунок 3.2 – Компоненти проєкту

У проєкті є файл Program.cs. Це стартовий файл або точка входу у застосунок. Цей файл запускає застосунок і містить налаштування проєкту.

Файли launchSettings.json, appsettings.json, appsettings.Development.json є файлами налаштувань. launchSettings відповідає за запуск проєкту, там зберігається налаштування для запуску: адреса сервера, ознака потреби запуску браузера, тип сервера (http, https, IIS).

Папки Controllers, Models, Views відносяться до паттерна MVC, але папку Models буде винесено до окремого шару. Для цього у проєкті створюємо папку, а у папці створюємо проєкт типу бібліотека з назвою Domain, у ній одразу формуємо папки Entity та ViewModels. Entity – це сутності, які будуть використовуватися для роботи з базою даних, ViewModels – це зменшена версія Entity для роботи з

контролерами, щоб не передавати усі параметри з Entity, будемо використовувати лише необхідні як ViewModel [19].

WWWROOT – це коренева папка для клієнтської частини, де зберігаються усі зображення, JavaScript-скрипти та CSS-таблиці. Також тут зберігається бібліотеки, наприклад Bootstrap, яка встановлена усталено [20].

Для реалізації треба створити додаткові папки з проєктами, за які вони відповідають (рис. 3.3), шар API не використовується.

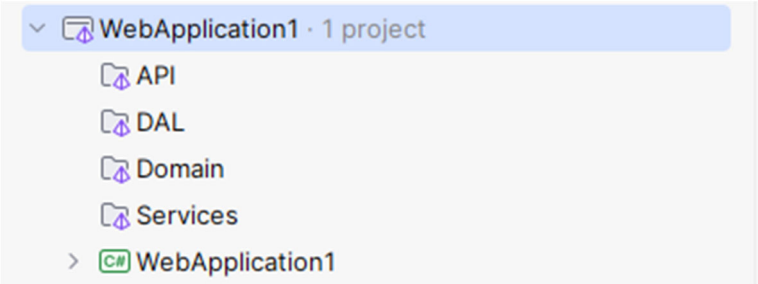

Рисунок 3.3 – Шари проєкту

Створення нового рішення у JetBrains Rider з використанням шаблону MVC закладає міцний фундамент для проєкту. Вибір шаблону "Web App (Model-View-Controller)" дозволяє дотримуватися принципів розділення логіки подання, контролю та моделі, що сприяє зручності обслуговування і масштабованості застосунку.

Проєкт "My101Romance" має ключові компоненти MVC архітектури (Properties, Wwwroot, Controllers, Models, Views) та файли налаштувань, зокрема appsettings.json та appsettings.Development.json. Файл Program.cs слугує стартовою точкою для запуску застосунку, що забезпечує його конфігурацію та ініціалізацію.

Винесення папки Models до окремого шару Domain з подальшим розподілом на Entity та ViewModels відображає підхід Domain-Driven Design (DDD). Це дозволяє краще організувати код та знизити зв'язність між компонентами, що підвищує гнучкість і підтримуваність проєкту.

Розміщення статичних файлів у Wwwroot сприяє організованості клієнтської частини застосунку, а використання бібліотек, таких як Bootstrap, полегшує створення сучасного та зручного інтерфейсу користувача.

3.2 Реалізація сутностей та бази даних

У створюваному застосунку є 2 головні сутності (класи) на рівні Domain – користувач (AppUser) і картка (Card). Кожна сутність (клас) має параметри, які характеризують поведінку сутності, тобто атрибути. Приклад класу Card наведено на рис. 3.4.

```
94 usages  dolbolesya
public class Card
{
    [Required]
    [Key]
    10 usages
    public int Id { get; set; }

    [Display(Name = "Заголовок")]
    13 usages
    public string? Title { get; set; }

    [Display(Name = "Опис")]
    12 usages
    public string? Description { get; set; }

    [Display(Name = "Зображення")]
    8 usages
    public string? ImagePath { get; set; } =
        "https://images.prom.ua/1065612508_w640_h640_vafelnaya-kartinka-lyubov.jpg";

    [Display(Name = "Рейтинг")]
    13 usages
    public int Rating { get; set; }

    [Display(Name = "Відображення для усіх користувачів")]
    4 usages
    public bool IsForAll { get; set; }

}
```

Рисунок 3.4 – Клас Card

Для зручної роботи було створено анотацію даних (Data Annotation) [21] для сутності, анотація потрібна для зручної валідації даних. Анотація [Key] вказує, що

при створені таблиці, атрибут Id є ключем, а анотація [Required] вказує, що це поле є обов'язковим.

Після реалізації сутностей на рівні (шар) Data Access Layer (DAL) було створено клас AppDbContext. Це файл для взаємодії з базою даних та підключення її (рис. 3.5).

```
11 usages  dolbolesya *
public sealed class AppDbContext : IdentityDbContext<AppUser>
{
    dolbolesya
    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
    {
        Database.EnsureCreated();
    }

    12 usages
    public DbSet<Card> Card { get; set; }

    6 usages
    public DbSet<AppUser?> User { get; set; }

    3 usages
    public DbSet<Card> UserChooseCard { get; set; }
}
```

Рисунок 3.5 – Клас AppDbContext

За допомогою команди Database.EnsureCreated() у проєкті створено базу даних при першому запуску, якщо її не було. Команди DbSet<клас> вказують на створення таблиць за описом класу [22].

Далі, у головному проєкті необхідно підключити посилання на базу даних. Для цього необхідно отримати дані підключення до бази даних. Для цього треба встановити на робочу машину ядро бази даних Microsoft SQL Server та систему керування базою даних (СКБД) Microsoft SQL Server Management Studio. Після цього треба запустити її, екран запуску зображено на рис. 3.6.

Якщо все правильно було встановлено, то при запуску СКБД з'явиться вікно підключення до сервера, де в полі Server Name буде адреса для підключення. Її

необхідно вставити у JSON-файл appsettings.json, де є поле ConnectingString з DefaultConnection, який містить рядок з ключем «Server=».

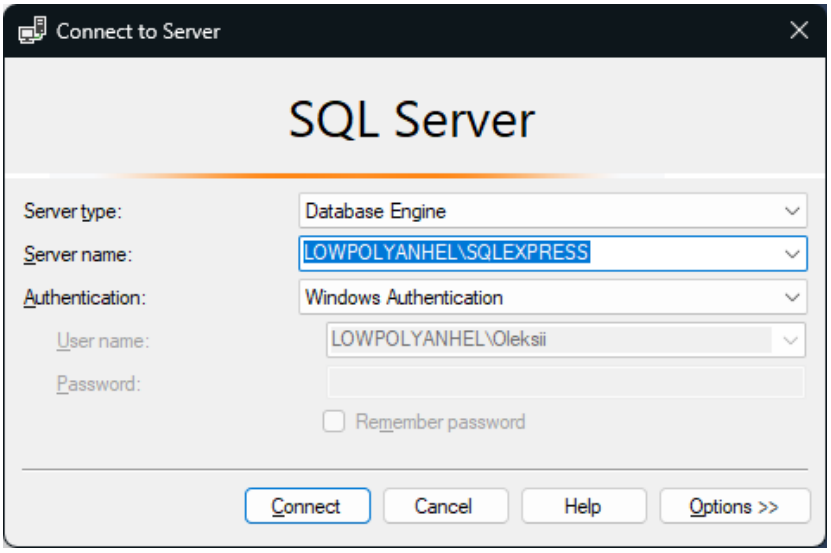

Рисунок 3.6 – Microsoft SQL Server Management Studio

Підставляємо адресу нашого сервера «Server=LOWPOLYANHEL\SQLEXPRESS;» і в цьому ж рядку прописуємо назву майбутньої бази «Database=romance». Також прописуємо налаштування для з'єднання.

На кінцевому етапі маємо рядок підключення, де вказано всі параметри:

```
"DefaultConnection": "Server=Ім'я_комп'ютера\\SQLEXPRESS;
Database=назва_бази_даних; Trusted_Connection=True;
MultipleActiveResultSets=true; TrustServerCertificate=True"
```

Далі у файлі Program.cs було виконано з'єднання з базою даних, використано DefaultConnection як рядок для підключення (рис. 3.7).

```
builder.Services.AddDbContext<AppDbContext>( optionsAction: options =>
{
    options.UseSqlServer(builder.Configuration.GetConnectionString( name: "DefaultConnection"));
});
```

Рисунок 3.7 – З'єднання з базою даних

Для створення бази даних у ASP.NET Core треба встановити певні nuget-pakage на різні шари проєкту [23] (табл. 3.1).

Таблиця 3.1 – Nuget-pakages

Рівень	Nuget-pakage
DAL	– Microsoft.AspNetCore.Identity.EntityFrameworkCore – Microsoft.EntityFrameworkCore – Microsoft.EntityFrameworkCore.Design – Microsoft.EntityFrameworkCore.SqlServer – Microsoft.EntityFrameworkCore.Tools
My101Romance	– Microsoft.EntityFrameworkCore.Design – Microsoft.EntityFrameworkCore.SqlServer – Microsoft.EntityFrameworkCore.Tools

При першому запуску проєкта буде створена нова база даних, у випадку, якщо її не було.

На рис. 3.8 зображено 7 додаткових таблиць з приставкою AspNet, які відносяться до Identity. Ці таблиці створюються за шаблоном, який прописано у IdentityDbContext, а також зв’язки між ними також створені за шаблоном.

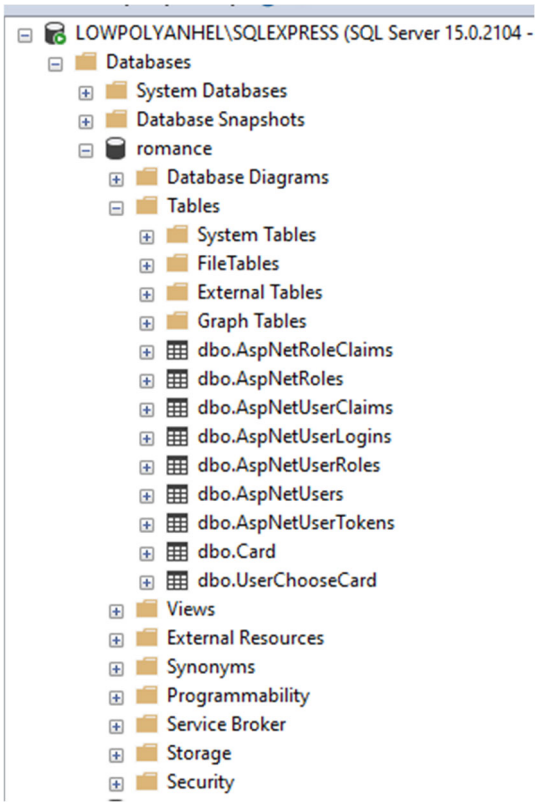

Рисунок 3.8 – Вигляд утворених таблиць

Можна побачити, що в нас є таблиця Card, яку створили, але немає таблиці AppUser. Це пов’язано з тим, клас AppUser повністю успадковується від класу IdentityUser, який належить до Microsoft.AspNetCore.Identity, тому усі користувачі будуть автоматично зберігатися у таблиці AspNetUsers. Клас AppUser є для кастомних параметрів для користувача, які не були створені у IdentityUser [20].

У Identity є 2 головні таблиці, які мають зв’язки з іншими таблицями, а саме AppUser (IdentityUser) та IdentityRole, тип зв’язків у табл. 3.2 [20].

3.3 Реалізація рейтингу вікторини

Формування рейтингу проходить під час проходження вікторини шляхом почергового вибору однієї з двох карток. Картка, яку вибрали, збільшує значення рейтингу. Тим самим формується рейтинг від картки з найбільшим показником до карток з найменшим рейтингом.

Перше, що треба зробити для участі у вікторині, – зареєструватися. Для цього на рівні Domain є класи RegisterViewModel та LoginViewModel. Це класи, які описують дані, що треба отримати для реєстрації або авторизації. Усі моделі ViewModel зберігаються у шарі Domain. RegisterViewModel та LoginViewModel (рис. 3.9 та 3.10).

```
12 usages  dolbolesya
public class RegisterViewModel
{
    [Required]
    7 usages
    public required string UserName { get; set; }

    [Required]
    [EmailAddress]
    8 usages
    public required string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    8 usages
    public string? Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    [Compare( otherProperty: "Password", ErrorMessage = "The password and confirmation password do not match.")]
    3 usages
    public required string ConfirmPassword { get; set; }


    [RegularExpression( pattern: "^[a-zA-Z0-9]*$", ErrorMessage = "Username can only contain letters and digits.")]
    public string ErrorMessage { get; set; } = "";
}
```

Рисунок 3.9 – Клас RegisterViewModel

Для реєстрації треба перейти за посиланням /Account/Register. При реєстрації користувачу треба заповнити дані: UserName, Email, Password, ConfirmPassword, після чого ці дані (Model) відправляються з подання (View) до контролеру (Controller) AccountController, де в нас існує два методи Register. Перший метод повертає подання для форми реєстрації, а другий вже приймає цю форму реєстрації і обробляє дані.

```
5 usages  dolbolesya
public class LoginViewModel
{
    [Required(ErrorMessage = "Email address is required!")]
    [Display(Name = "email")]
    4 usages
    public string Email { get; set; }

    [Required(ErrorMessage = "Password is required!")]
    [DataType(DataType.Password)]
    4 usages
    public string Password { get; set; }
}
```

Рисунок 3.10 – Клас LoginViewModel

В першу чергу ці дані перевіряються на валідність, тобто правильно вказані, якщо ні, користувач отримує про це повідомлення. Якщо дані валідні, то проходить перевірка, пошук користувача з такою поштою, як вказано у формі. У випадку, якщо користувачів не знайдено, ми створюємо нового користувача і робимо присвоєння даних з форми до користувача (рис. 3.11).

```
var existingUser = await _userManager.FindByEmailAsync(model.Email);
if (existingUser != null)
{
    //user with this email
    ModelState.AddModelError(key: nameof(RegisterViewModel.Email), errorMessage: "User with this email already exists.");
    return View("auth/Register", model);
}
```

Рисунок 3.11 – Пошук по пошті

Наступним кроком перевіряється пароль, якщо він підходить під умови Identity, то в поле пароля для користувача записується хєш пароля. Якщо все вдалося, то проходить збереження користувача та присвоєння йому ролі «User» за допомогою інструменту UserManager.

Для реєстрації необхідно вказати пароль, який буде валідним по правилам Identity, в нашому випадку, довжина пароля 4 символи, 1 знак, символи обох регістрів. Вимоги до пароля вказані на рис. 3.12.

```
builder.Services.AddIdentity<AppUser, IdentityRole>( setupAction: options =>
{
    options.Password.RequiredLength = 4;
    options.Password.RequireLowercase = true;
    options.Password.RequireUppercase = true;
    options.Password.RequireNonAlphanumeric = true;
})
```

Рисунок 3.12 – Вимоги до паролю Identity

Для автентифікації та авторизації необхідно перейти по шляху /Accoun/Login. Метод Login також має 2 реалізації, подання форми авторизації, а також метод який

приймає дані з форми і робить автентифікацію. У випадку збігу даних, користувач авторизується на сайті.

Якщо цей користувач є адміністратором або root, в нього з'являється додаткове меню – меню адміністратора, де можна створити нові ролі або надати користувачам нові права. Identity також надає інструменти для перевірки, яка роль у користувача і таким чином відобразити контент (рис. 3.13).

Тепер, коли користувач зареєстрований, він може перейти за посиланням /Quiz/Play та розпочати гру.

Перед користувачем з'являється пара з двох випадкових карток, де необхідно вибрати одну з двох, 8 разів. Картки, які були вибрані, піднімаються у рейтингу. Після 3 спроб по 8 пар користувач не зможе грати добу, це зроблено в цілях безпеки та для запобігання накручування.

```
@if (User.IsInRole("admin") || User.IsInRole("root"))
{
    <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#"
           id="adminDropdown" role="button" data-toggle="dropdown"
           aria-haspopup="true" aria-expanded="false">
            Адмін
        </a>
        <div class="dropdown-menu" aria-labelledby="adminDropdown">
            <a class="dropdown-item" href="/admin/addrole">Add Role</a>
            <a class="dropdown-item" href="/admin/rolelist">Role List</a>

        </div>
    </li>
}
```

Рисунок 3.13 – Identity. Подання контенту в залежності від ролі

Отримання карток на рівні Data Access Layer (DAL) реалізується через спеціальний метод, який взаємодіє з базою даних для отримання випадкових карток. Спочатку в DAL створюється інтерфейс, який має метод для пошуку випадкових карток. Цей інтерфейс визначає, які саме методи мають бути реалізовані для взаємодії з базою даних. Наприклад, інтерфейс може містити метод `GetRandomCards`, який має параметр `count`, що визначає кількість карток.

Потім клас, який реалізує цей інтерфейс, містить конкретну реалізацію методу для отримання випадкових карток. У даному випадку метод `GetRandomCards` використовує LINQ для вибору випадкових карток з бази даних. Використання `Guid.NewGuid()` у `OrderBy` дозволяє отримати випадковий порядок карток, а метод `Take` обмежує кількість карток, які будуть повернуті.

Далі на рівні бізнес-логіки або сервісу викликається метод з репозиторію для отримання карток. Наприклад, метод `SelectEightCards` в сервісі викликає метод з репозиторію для отримання двох випадкових карток.

Нарешті, контролер, який обробляє запити користувача, звертається до сервісу для отримання карток і передає їх у вигляді відповіді на запит. Контролер обробляє HTTP-запит користувача до `/Quiz/Play`, отримує картки через сервіс і передає їх у вигляді моделі до подання.

На рівні DAL є метод, який реалізує інтерфейс для пошуку випадкових карток, метод для отримання карток зображено на рис. 3.14.

```
0+2 usages   dolbolesya
public async Task<List<Card>> SelectEightCards()
{
    var randomCards List<Card> = await _db.Card.OrderBy(x Card => Guid.NewGuid()).Take(2).ToListAsync();
    return randomCards;
}
```

Рисунок 3.14 – Опис методу для отримання карток

Далі ці картки потрапляють до шару Services, звідки цей метод виклкається з контролера і передається до View.

Для зручності у файлі Program.cs зареєстровано нові шляхи до контролерів та методів, які викликаються у контролерах:

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

Усі базові шляхи для зручності було винесено (рис 3.15).

```
app.MapControllerRoute(
    name: "quiz",
    pattern: "Quiz/SelectCard",
    defaults: new { controller = "Quiz", action = "SelectCard" });

app.MapControllerRoute(
    name: "login",
    pattern: "login",
    defaults: new { controller = "Account", action = "Login" });

app.MapControllerRoute(
    name: "register",
    pattern: "register",
    defaults: new { controller = "Account", action = "Register" });

app.MapControllerRoute(
    name: "quiz",
    pattern: "quiz",
    defaults: new { controller = "Quiz", action = "Play" });

app.MapControllerRoute(
    name: "top",
    pattern: "top",
    defaults: new { controller = "Card", action = "Top" },
    constraints: new { isAuthenticated = new IsNotAuth() });

app.MapControllerRoute(
    name: "random",
    pattern: "random",
    defaults: new { controller = "Card", action = "ShowRandomCards" });
```

Рисунок 3.15 – Реєстрація нових маршрутів.

Попри нові маршрути, весь функціонал доступний за старими маршрутами.

3.4 Функціонал програмного застосунку

Вебзастосунок вікторини для вибору можливих ідей для побачень, насамперед має реєстрацію та авторизацію користувачів. Зареєстровані користувачі при вході матимуть змогу проходити вікторину, під час якої формуватиметься рейтинг можливих ідей для побачень.

Для авторизації потрібно зареєструвати нового користувача. За посиланням шляхом «/register» користувача очікує форма реєстрації (рис 3.16). Для процедури реєстрації необхідно вказати ім’я користувача у полі «Username», пошту у полі «Email», пароль «Password» та підтвердити його «Confirm password».

Дані нового користувача:

– Username – r00t;
– Email – root@root;

– Password – Qq1!;

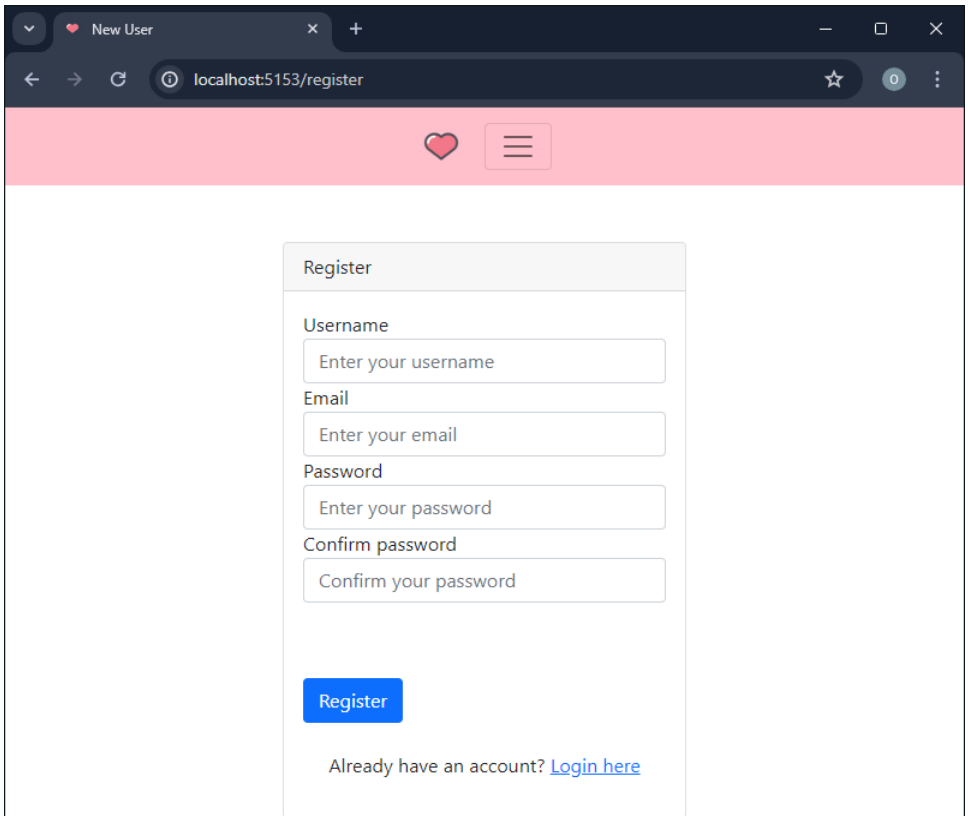

Рисунок 3.16 – Форма реєстрації.

Після реєстрації, оновивши базу даних, там з'являється новий користувач (рис 3.17).

Results Messages

	Id	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed	PasswordHash
1	5a3cc...	root	ROOT	root@mail	ROOT@MAIL	0	AQAAAAIAAYagAAAAE...
2	62923...	test	TEST	test@test	TEST@TEST	0	AQAAAAIAAYagAAAAE...
3	94c90...	r00t	R00T	root@root	ROOT@ROOT	0	AQAAAAIAAYagAAAAE...

Рисунок 3.17 – Новий користувач у базі даних

Для авторизації та автентифікації необхідно ввести дані користувача у форму авторизації за шляхом /login (рис. 3.18).

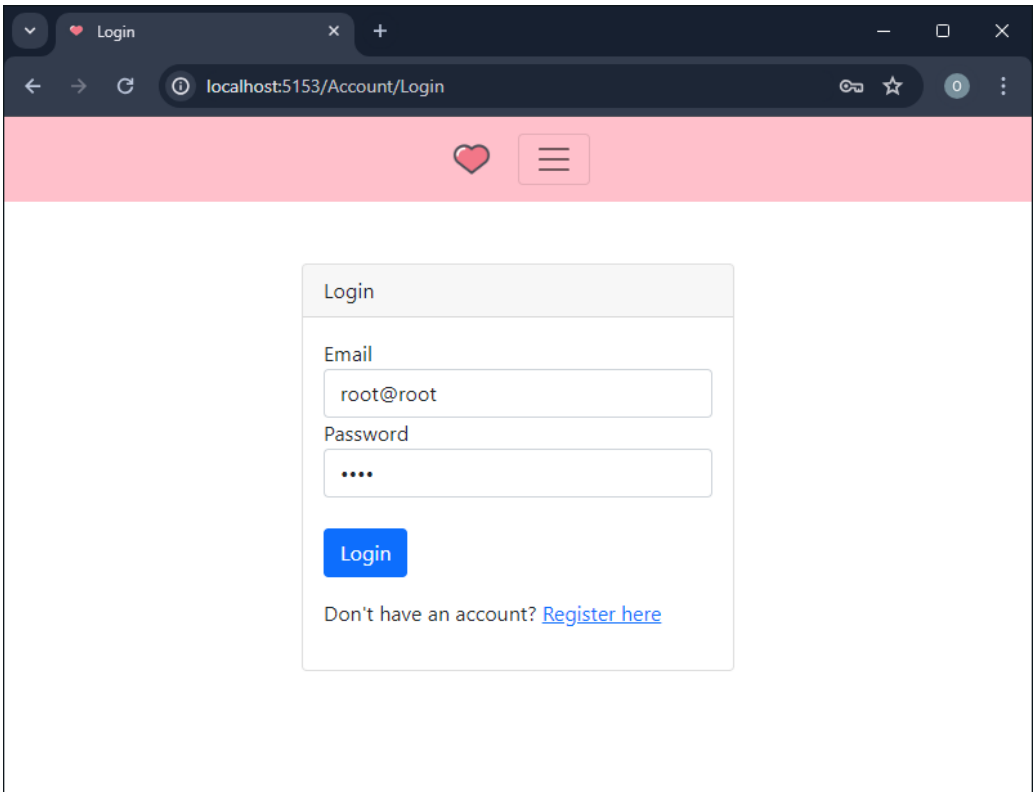

Рисунок 3.18 – Форма авторизації

Після авторизації у навігаційному меню з'явиться Username, який використано для реєстрації (рис. 3.19).

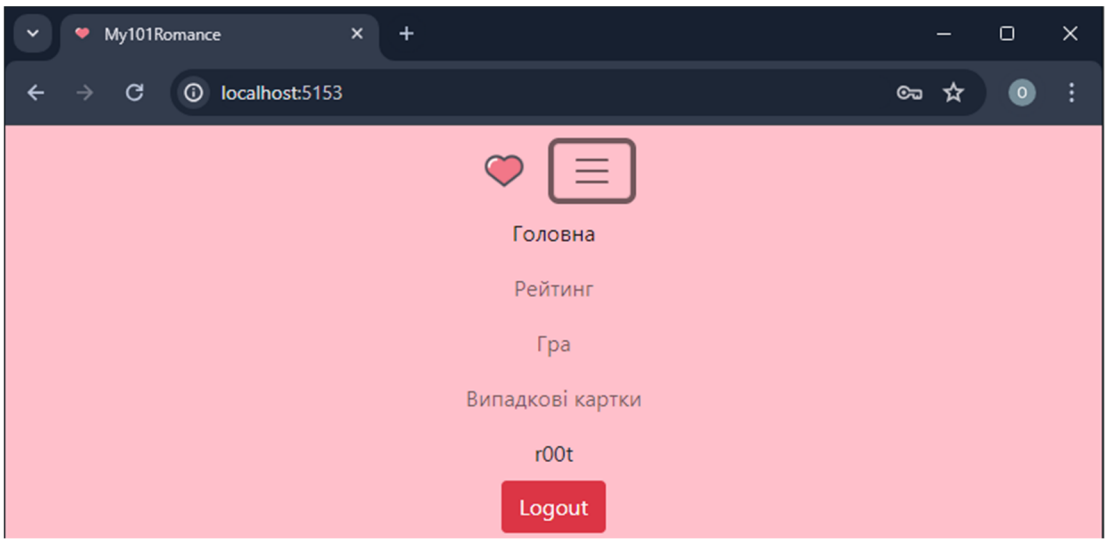

Рисунок 3.19 – Подання Username

За посиланням /admin/rolelist будуть відображені усі доступні ролі у вебзастосунку (рис. 3.20).

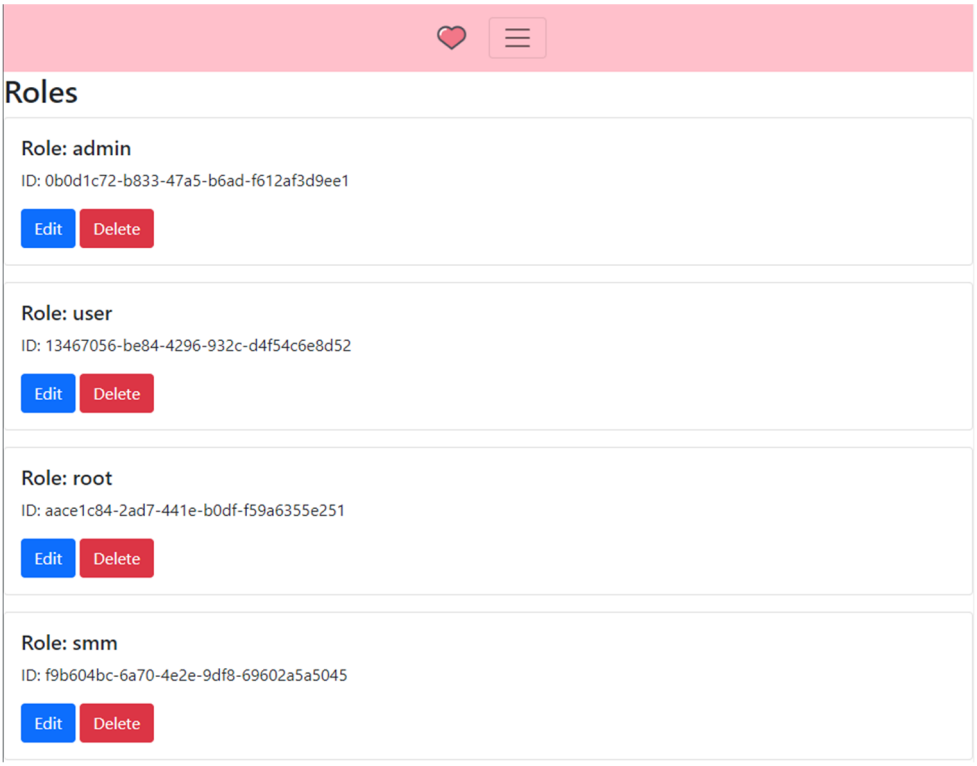

Рисунок 3.20 – Перечень ролей

За посиланням /Admin/EditRole/ID буде відображено усіх користувачів, які мають роль «user» (рис 3.21). Також на цій сторінці можна змінити назву ролі.

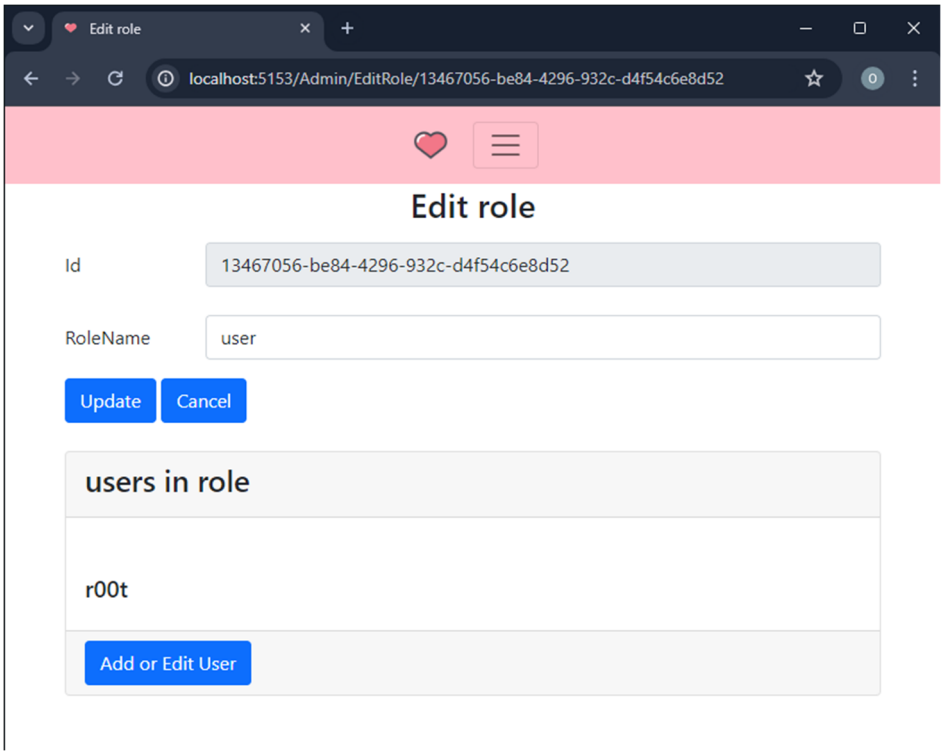

Рисунок 3.21 – Редагування користувача

Для задавання користувачам нових ролей варто вибрати Username користувача або користувачів і натиснути кнопку «Update» (рис. 3.22).

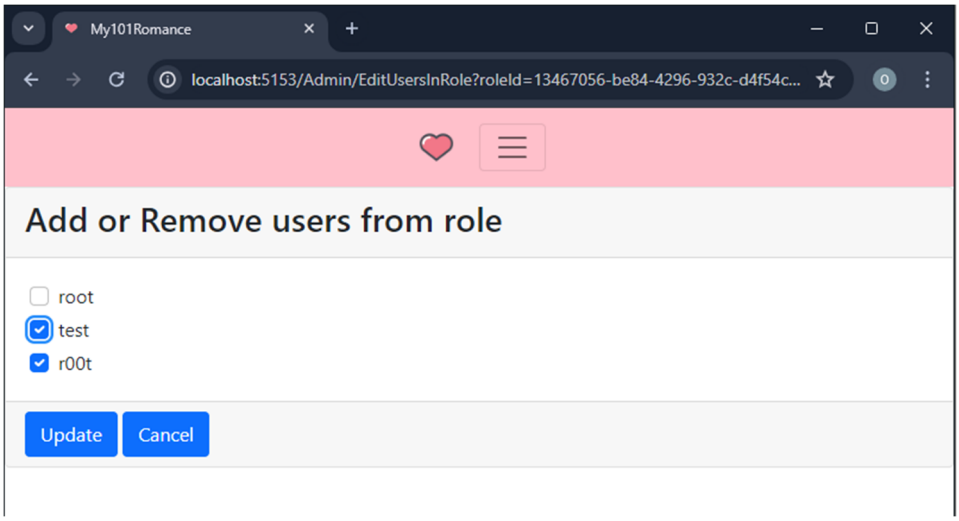

Рисунок 3.22 – Додавання користувачів до ролі

Якщо повернутися до сторінки перегляду ролей користувачів, то буде відображено новий лист користувачів у ролі «user» (рис. 3.23).

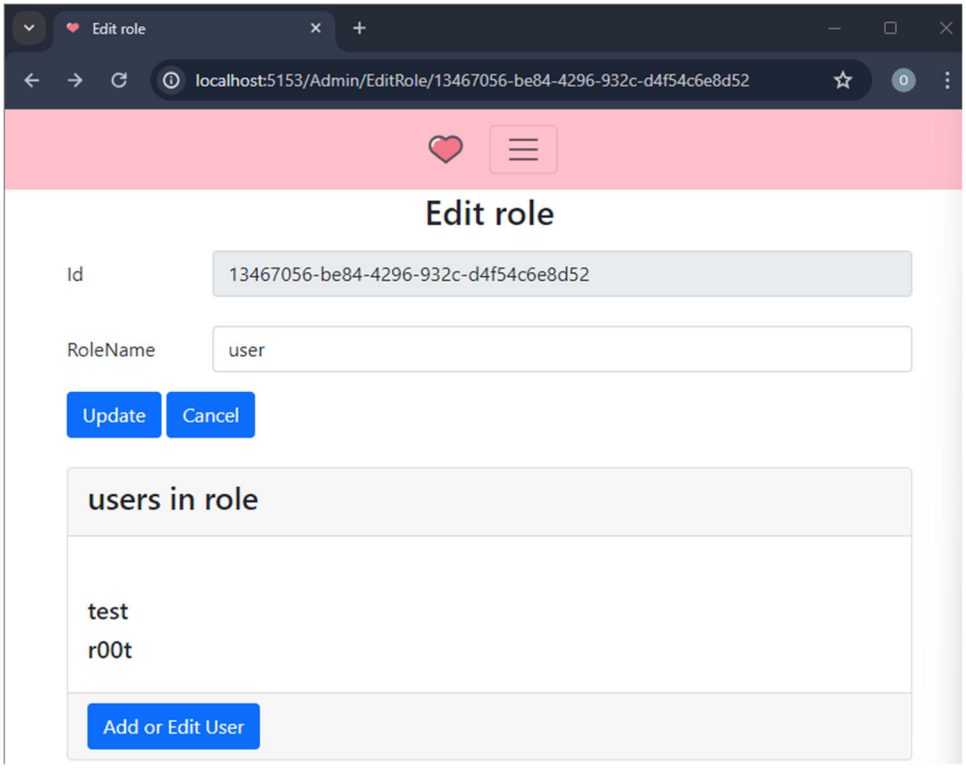

Рисунок 3.23 – Новий лист користувачів у ролі

Якщо розглядати функціонал вікторини, то за адресою /quiz гравцю запропонують вибрати одну з двох карток (рис. 3.24). При наведенні на кожну картку буде відображатися її заголовок (title) та опис (description) (рис. 3.25).

За умови, якщо користувач не авторизувався, йому буде надано доступ лише до двох методів, а саме:

– рейтинг;
– випадкові картки.

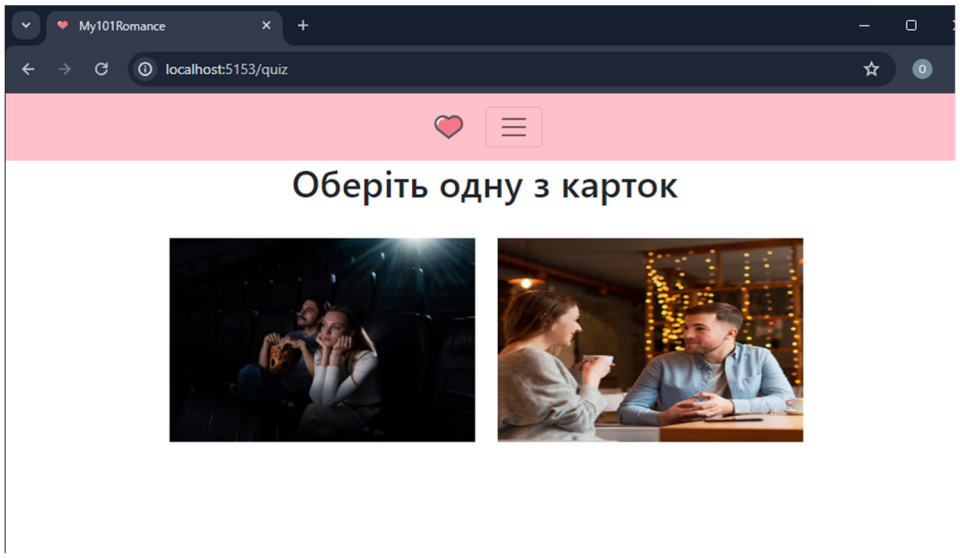

Рисунок 3.24 – Вікторина

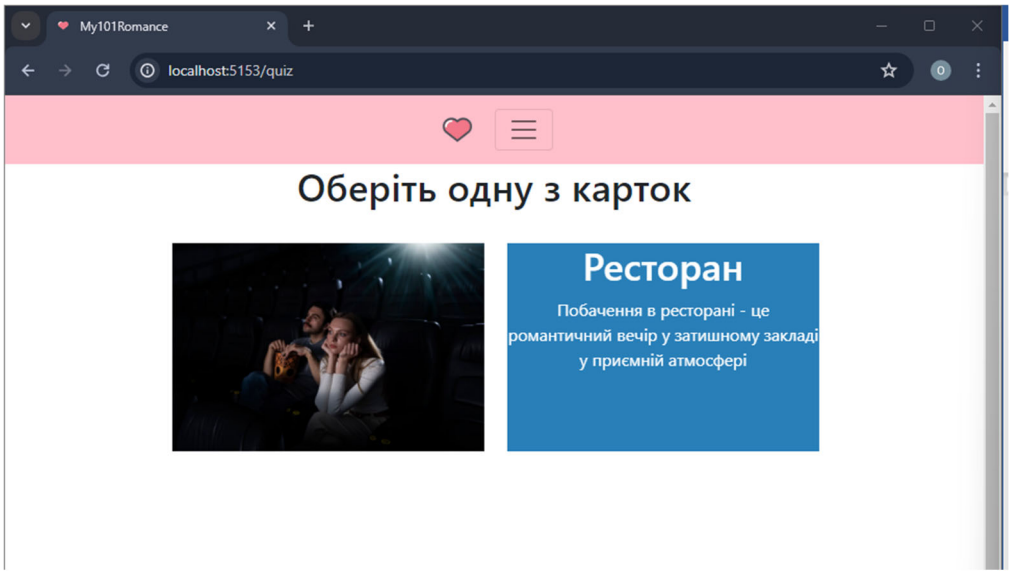

Рисунок 3.25 – Перегляд заголовку та опису

На сторінці вікторини у файлі Quiz.cshtml розташовано скрипт, який відповідає за збільшення рейтингу для карток.

```
<script>
    function selectCard(cardId) {
        fetch('/Quiz/SelectCard', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(cardId)
        })
            .then(response => {
                if (!response.ok) {
                    return response.json().then(data => {
                        throw new Error(`Network response was not ok: ${data.message}`);
                    });
                }
                return response.json();
            })
            .then(data => {
                console.log(data);
                window.location.reload();
            })
            .catch(error => {
                console.error('Ошибка:', error);
            });
    }
</script>
```

В момент, коли користувач вибирає картку, з клієнтської частини на серверну надходить запит у контролер Quiz до методу SelectCard методом Post, в якому передається Id картки, яку вибрали.

В свою чергу, метод SelectCard на серверній частині отримає Post запит, як аргумент у методі, і за допомогою анотації даних [FromBody] забере із запиту параметр cardId. У методі проходить перевірка на наявність такої картки у базі даних. У разі, якщо такий cardId (Id) не порожній, то надходить запит до конкретної картки та збільшується параметр Rating на +1. Далі зберігаються зміни у базі даних. Без збереження не буде підвищення рейтингу картки. Після збереження проходить повернення результату у вигляді Http статусу Ok(), тобто статус запиту 200, що означає вдалий запит.

```
[HttpPost("/Quiz/SelectCard")] // Specify the URL for the SelectCard method
public async Task<IActionResult> SelectCard([FromBody] int cardId)
{
    var card = await _cardService.GetCardById(cardId);
```

```
    if (card != null)
    {
        card.Rating++;
        await _cardService.UpdateCard(card);
        // Return a JSON response with a success message
        return Ok(new { message = "Success" });
    }
    else
    {
        return NotFound(new { message = "Card not found" });
        // Return a JSON response with a not found message
    }
}
```

Перегляд рейтингу до використання вікторини для звичайного користувача наведено рис. 3.26, а для користувачів у ролі admin чи root – на рис. 3.27.

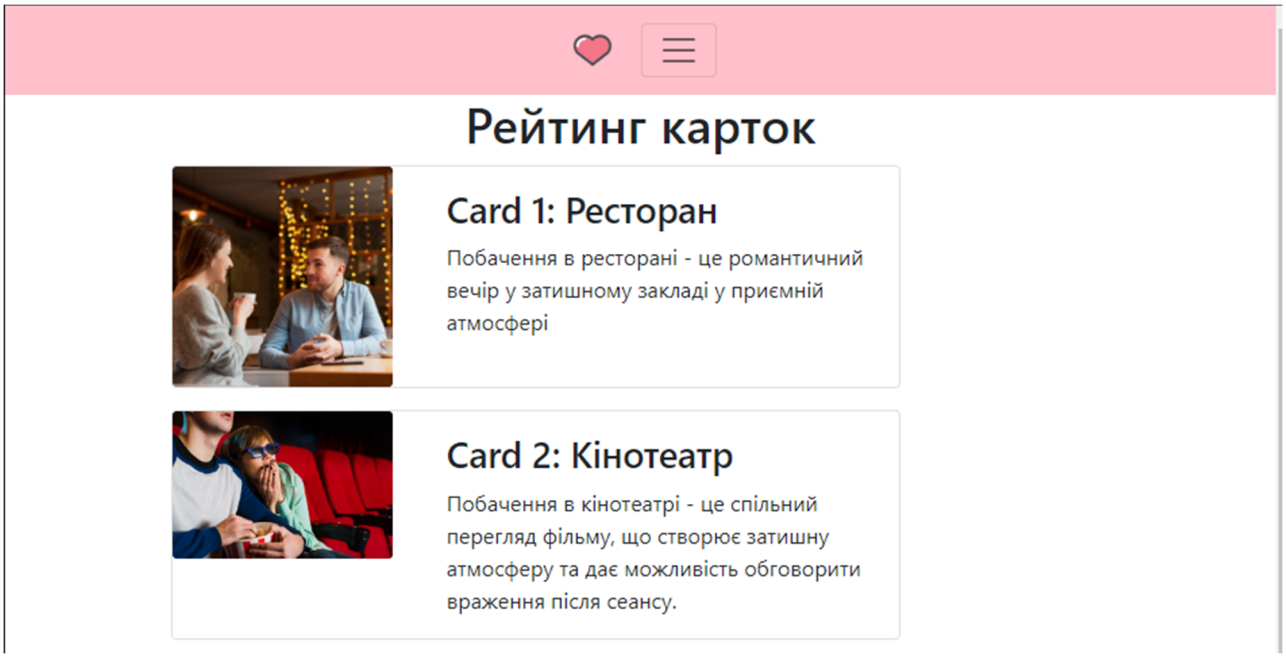

Рисунок 3.26 – Подання рейтингу для звичайних користувачів

На цих рисунках зображена додаткова інформація для адміністраторів про унікальний ID картки та їх рейтинг.

Після проходження вікторини рейтинг карток автоматично буде змінено як в базі даних, так і для рейтингу в реальному часі, рис. 3.28.

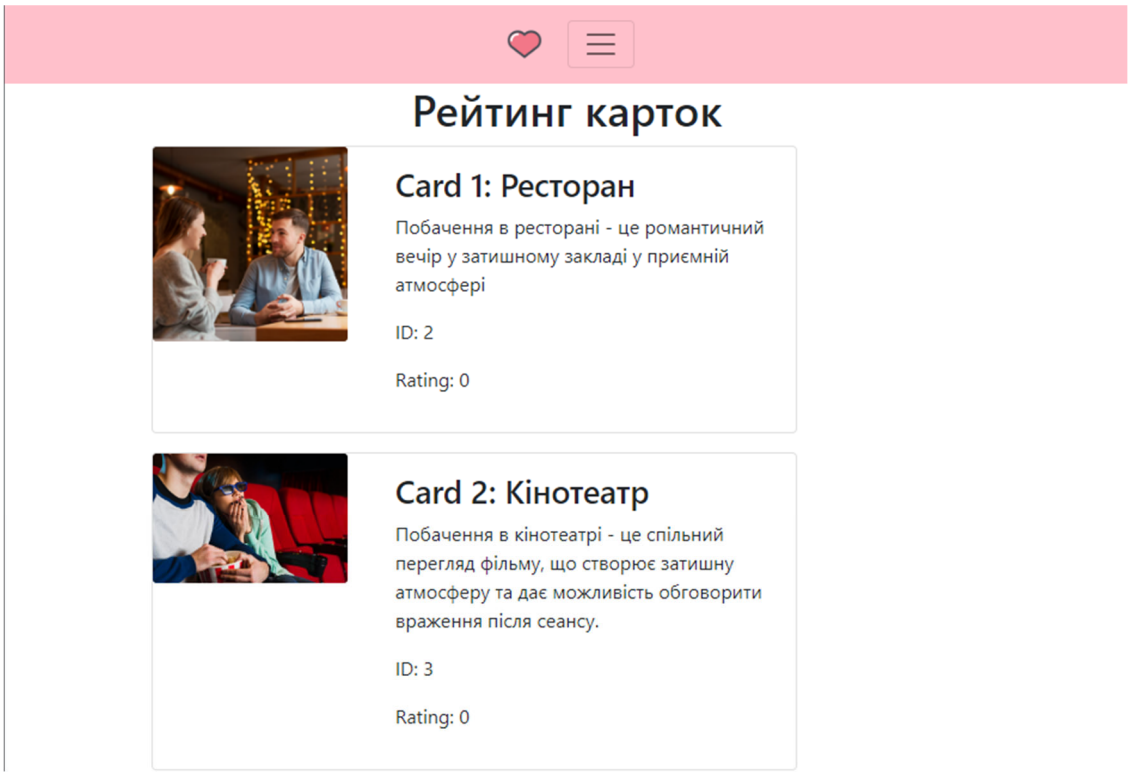

Рисунок 3.27 – Подання рейтингу для адміністраторів

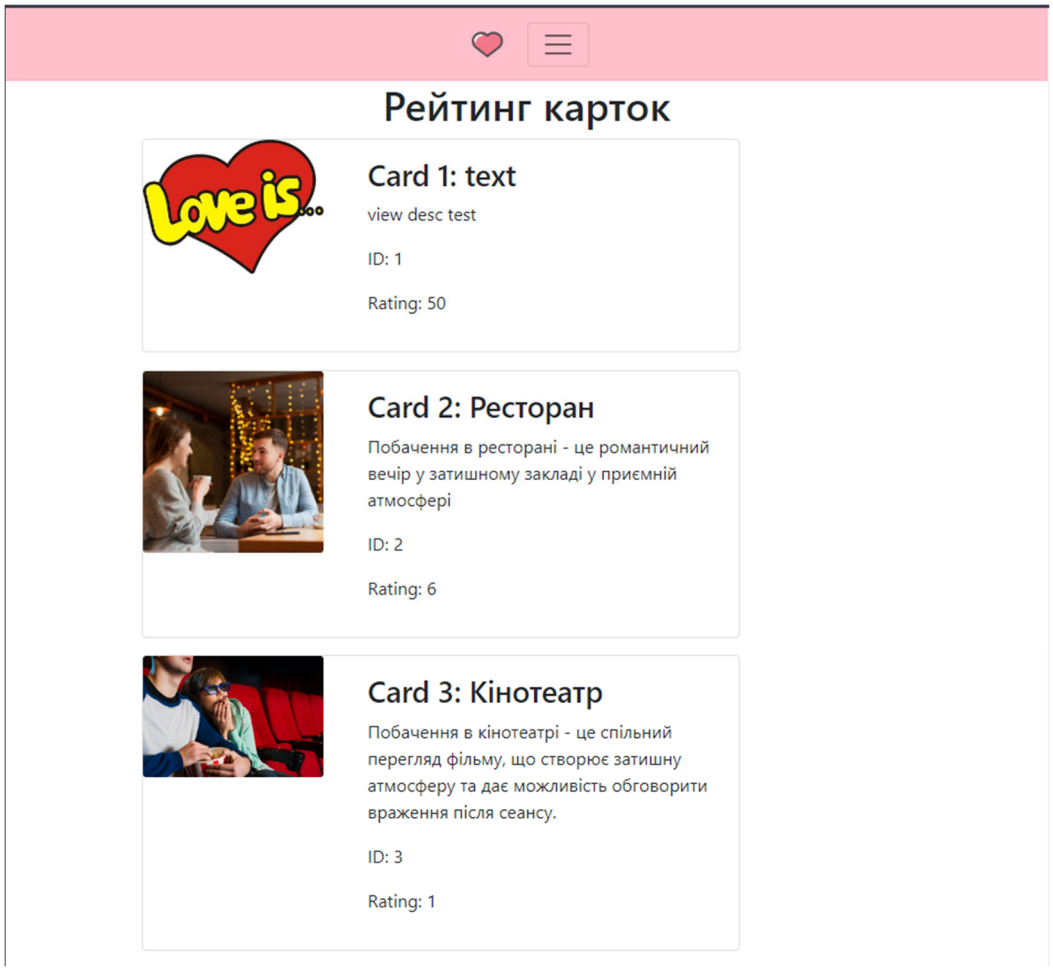

Рисунок 3.28 – Подання рейтингу після вікторини

Отже, для вебзастосунку було розроблено та продемонстровано функціонал, який відповідає базовим вимогам проєкту.

Елементи функціоналу мають реєстрацію та авторизацію користувачів, забезпечення ролей та керування ними, а також інтерактивну вікторину з можливістю формування рейтингу карток. Реєстрація нових користувачів здійснюється за допомогою форми, де вводяться ім'я користувача, електронна пошта, пароль та підтвердження пароля. Після успішної реєстрації дані нового користувача зберігаються у базі даних. Авторизація дозволяє зареєстрованим користувачам входити до системи, використовуючи свої облікові дані, і забезпечує доступ до персоналізованого контенту, включаючи вікторину.

Керування ролями користувачів реалізовано через адміністративний інтерфейс, де можна переглядати доступні ролі, редагувати їх назви та додавати користувачів до різних ролей. Це забезпечує гнучке управління доступом до різних функцій застосунку.

Функціонал вікторини дозволяє користувачам вибирати картки, кожна з яких має заголовок і опис. Під час вибору картки відбувається передача даних на сервер, де оновлюється рейтинг картки у базі даних. Це забезпечує динамічне оновлення рейтингу в реальному часі.

3.4 Висновки до розділу 3

У розділі розглянуто особливості реалізації проєкту засобами IDE JetBrains Rider, ASP.NET Core та Bootstrap. Описано налаштування проєкту, створення сутностей, підключення до бази даних через Microsoft SQL Server та Microsoft SQL Server Management Studio. Реалізація вікторини формує рейтинг ідей для побачень.

Користувачі можуть зареєструватися, пройти авторизацію і взяти участь у вікторині, підвищуючи рейтинг карток. Реалізація різних ролей для користувачів забезпечує гнучке керування доступом до функціональності системи. Впровадження цих функцій забезпечило не тільки технічну, а й бізнесову логіку проєкту, що зробило його привабливим і корисним для кінцевих користувачів. Всі елементи реалізації описані та ілюстровані відповідними рисунками.

ВИСНОВКИ

Кваліфікаційна робота присвячена розробці системи формування інтерактивних рейтингів засобами технологій ASP.NET Core. В процесі виконання роботи проаналізовано вимоги до проєкту, специфіку розробки вебзастосунків та вибрано відповідний стек технологій розробки. Використання архітектурних підходів Domain-Driven Design (DDD) та шарової архітектури забезпечило створення логічно структурованої, гнучкої, масштабованої системи.

У першому розділі проведено аналіз аналогів, який виявив переваги і недоліки статичних рейтингів на вибрану тематику. Це надало можливості для формування функціональних вимог до створюваного у роботі інтерактивного рейтингу. Ключовими вимогами до проєкту є: актуальність ідей, великий вибір та адаптивний інтерфейс на різних пристроях.

Другий розділ роботи присвячений аналізу сучасних технологій та інструментів для розробки вебзастосунків. Використання HTML, CSS та Bootstrap для фронтенду забезпечило створення привабливого, зручного інтерфейсу користувача. Для бекенду вибрано ASP.NET Core, що забезпечило високу продуктивність та надійність системи. Архітектура проєкту, що поєднує в собі DDD, шарову архітектуру та паттерн Model-View-Controller (MVC), сприяла модульності, легкості в підтримці та розвитку системи. Діаграма прецендентів (Use Case) допомогла визначити функціональні вимоги та можливості для користувачів, забезпечуючи інтуїтивно зрозумілий та ефективний інтерфейс.

Третій розділ описує процес реалізації проєкту за допомогою IDE JetBrains Rider, ASP.NET Core та Bootstrap. Створено сутності та налаштовано підключення до бази даних через Microsoft SQL Server і Microsoft SQL Server Management Studio. Реалізація вікторини, що формує рейтинг ідей для побачень, підвищила інтерактивність системи. Користувачі можуть реєструватися, проходити авторизацію і брати участь у вікторині, що робить систему більш привабливою та корисною. Впровадження різних ролей для користувачів забезпечує гнучке управління доступом до функціональності системи.

Загалом виконана кваліфікаційна робота продемонструвала сучасний підхід до розробки системи формування інтерактивних рейтингів на прикладі вебзастосунків, використовуючи новітні технології та архітектурні підходи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Грабіна К. В., Шендрик В. В., Данченко О. Б. Застосування SWOT-аналізу для ідентифікації ризиків проєкту. С. 133-136.
2. Tezzt.com. URL: https://tezzt.com/
3. Wordwall.net. URL: https://wordwall.net/
4. Ukrerudyt. URL: https://ukrerudyt.com/
5. Трубачова В. Конструктор сайтів: аналіз роботи. *Математичні методи, моделі та інформаційні технології у науці, освіті, економіці, виробництві*: зб. тез II Всеукр. наук.-практ. Інтернет-конф. з проблем вищої освіти і науки, м. Маріуполь, 29 квітня 2020 р. / Маріупольський державний університет; уклад. Шабельник Т. В., Дяченко О. Ф., Морозова А. О., Лазаревська Ю.А. Маріуполь : МДУ, 2020. С. 282-285.
6. Wix.com. URL: https://uk.wix.com/
7. Дворковий І. В., Варварич Б. В. Сучасні технології в галузі створення frontend and backend. *Інноваційна Україна: креативні ідеї та проєкти*: зб. доп. 87-ї наук. студ. конф., м. Київ, 4–13 травня 2020 р. Київ, КНЕУ, 2020 С. 342-343.
8. Що таке back-end розробка і чим займається back-end розробник? URL: https://icstudio.online/post/shcho-take-back-end-rozrobka-i-chim-zajmayetsya-back-end-rozrobnik
9. ASP.NET documentation. URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0
10. Солом'яний О. М. Веброзробка засобами ASP.NET Core. *Кіберпростір в умовах війни та глобальних викликів XXI століття: теорія та практика*: матер. всеукр. наук.-практ. конф. (м. Одеса, 24 листопада 2023 р.). Одеса, 2023. С. 185-187. DOI: 10.32837/11300.27179. URL: https://hdl.handle.net/11300/27179.
11. Bootstrap. URL: https://getbootstrap.com/
12. Підхід до проєктування ПЗ DDD. URL: https://foxminded.ua/ddd-shcho-tse/ .

13. Layered Architecture Used in Software Development. URL: https://dev.to/sardarmudassaralikhan/layered-architecture-used-in-software-development-8jd.
14. Model-View-Controller Patter. URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-8.0
15. Purnasari M., Hartiwi Y., Nurhayati N. Perancangan Sistem Informasi Pengelolaan Dana Masjid Berbasis Web Menggunakan Unified Modeling Language (UML), 2020. Vol. 2 No. 6. URL: https://djournals.com/resolusi/article/view/416
16. What is use case? URL: https://www.techtarget.com/searchsoftwarequality/definition/use-case
17. Що таке ментальна карта, і як її створювати? URL: https://mc.today/uk/shho-take-mentalna-karta-i-yak-yiyi-stvoryuvati/
18. JetBrains Rider. URL: https://www.jetbrains.com/rider/
19. Troelsen, A., Japikse, P. Build a Data Access Layer with Entity Framework Core. In: Pro C# 9 with .NET 5. Apress, Berkeley, CA, 2021. URL: https://link.springer.com/chapter/10.1007/978-1-4842-6939-8_23
20. Що таке коренева папка або кореневий каталог? URL: https://salo.li/Cd245Cd
21. Must-Know Data Annotations for ASP.NET Core: Simplifying Data Management. URL: https://levelup.gitconnected.com/20-important-data-annotations-in-asp-net-core-mvc-f0935dd91661
22. DatabaseFacade.EnsureCreated Method. URL: https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.infrastructure.databasefacade.ensurecreated?view=efcore-8.0
23. Gatev, R. Using Dapr in ASP.NET Core. In: Introducing Distributed Application Runtime (Dapr). Apress, Berkeley, CA, 2021. URL: https://link.springer.com/chapter/10.1007/978-1-4842-6998-5_14

ДОДАТКИ

Додаток А. Програмний код Program.cs

```
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using My101Romance.CustomAuthRoute;
using My101Romance.DAL;
using My101Romance.DAL.Interfaces;
using My101Romance.DAL.Repositories;
using My101Romance.Domain.Entity;
using My101Romance.Services;
using My101Romance.Services.Implementations;
using My101Romance.Services.Interfaces;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllersWithViews();
builder.Services.AddServices();
builder.Services.AddRepository();
builder.Services.AddSession();

builder.Services.AddDbContext<AppDbContext>(options =>
{
 options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"));
});

builder.Services.AddIdentity<AppUser, IdentityRole>(options =>
{
    options.Password.RequiredLength = 4;
    options.Password.RequireLowercase = true;
    options.Password.RequireUppercase = true;
    options.Password.RequireNonAlphanumeric = true;
})
    .AddEntityFrameworkStores<AppDbContext>();
builder.Services.AddScoped<RoleManager<IdentityRole>>();
builder.Services.ConfigureApplicationCookie(options =>
{
    options.LoginPath = "/Account/Login";
    options.LogoutPath = "/Account/Logout";
});

builder.Services.AddMemoryCache();
builder.Services.AddSession();
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie();

builder.Configuration.Bind("Project", new Config());
builder.Configuration.Bind("SocialLinks", new Config());

var app = builder.Build();
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
```

```
    app.UseHsts();
}
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();
app.UseSession();
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
app.MapControllerRoute(
    name: "quiz",
    pattern: "Quiz/SelectCard",
    defaults: new { controller = "Quiz", action = "SelectCard" });
app.MapControllerRoute(
    name: "login",
    pattern: "login",
    defaults: new { controller = "Account", action = "Login" });
app.MapControllerRoute(
    name: "register",
    pattern: "register",
    defaults: new { controller = "Account", action = "Register" });
app.MapControllerRoute(
    name: "logout",
    pattern: "Account/Logout",
    defaults: new { controller = "Account", action = "Logout" });
app.MapControllerRoute(
    name: "quiz",
    pattern: "quiz",
    defaults: new { controller = "Quiz", action = "Play" });
app.MapControllerRoute(
    name: "top",
    pattern: "top",
    defaults: new { controller = "Card", action = "Top" },
    constraints: new { isAuthenticated = new IsNotAuth() });
app.MapControllerRoute(
    name: "top18plus",
    pattern: "top",
    defaults: new { controller = "Card", action = "Top18Plus" },
    constraints: new { isAuthenticated = new IsAuth() });
app.MapControllerRoute(
    name: "random",
    pattern: "random",
    defaults: new { controller = "Card", action = "ShowRandomCards" });
app.MapControllerRoute(
    name: "new role",
    pattern: "admin/addrole",
    defaults: new { controller = "Admin", action = "CreateRole" });
app.Run();
```

Додаток Б. Програмний код майстер-сторінки

```
<!DOCTYPE html>
<html lang="en">
<head>


    @await Html.PartialAsync("Partial/MetatagsPartial")
    @await Html.PartialAsync("Partial/CssPartial")
</head>
<body style="height: 100%">
<div class="wrap" style="min-height:100%; display: flex; flex-direction: column;
height: 100vh;">
    @await Html.PartialAsync("Partial/HeaderPartial")
        <main style="flex: 1 1 auto;">
            @RenderBody()
        </main>
    @await Html.PartialAsync("Partial/FooterPartial")

    @await Html.PartialAsync("Partial/ScriptsPartial")
</div>


@await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

Додаток В. Копії документів про апробацію результатів роботи

Список використаних джерел

1. Tidyverse – Wikipedia. URL: https://en.wikipedia.org/wiki/Tidyverse .
2. Маніпуляція з даними за допомогою dplyr і не тільки r_econometricks.knit. URL: https://aranaur.github.io/r_econometrics/dplyr.html .
3. Create Elegant Data Visualisations Using the Grammar of Graphics ggplot2. URL: https://ggplot2.tidyverse.org/ .
4. Plot function – R Documentation. URL: https://www.rdocumentation.org/packages/graphics/versions/3.6.2/topics/plot .

Ключові слова: пакет, tidyverse, функція, датасет, tibbl.
Keywords: tidy data, package, tidyverse, function, dataset, tibble.

Науковий керівник: к.пед.н., доцент Лобода Ю.Г.

ВЕБРОЗРОБКА ЗАСОБАМИ ASP.NET CORE

Солом'яний Олексій

студент 4-го курсу факультету кібербезпеки та інформаційних технологій Національного університету «Одеська юридична академія»

Наразі веброзробка засобами стека технологій .NET набуває популярності у секторі високонавантажених застосунків для бізнес-сектору. З кожним роком вимоги користувачів до сайтів зростають, тим самим збільшується навантаження на сайти за кількістю запитів. Чим більше бізнес, тим швидше компанія стикається з проблемою великого навантаження на її вебресурси. Для розв'язування такого роду проблем варто або збільшувати ресурси сайту, або при розробці сайту використовувати технології, які дозволяють ефективно та гнучко проєктувати сервіси з великим навантаженням.

ASP.NET Core – технологія для розробки вебсайтів та вебзастосунків з великим навантаженням. З перших версій ця технологія проєктувалася під великі навантаження, тому весь код пишеться засобами асинхронного програмування [1].

Більшість застосунків на ASP.NET Core працюють за шаблоном MVC (Model-View-Controller) [2]. За цією моделлю кожен запит користувача проходить через 3 етапи: 1) етап Model формує дані, які користувач відправляє на бізнес-логіку сайту, а контролер передає їх далі на відображення; 2) View – це відображення інформації на сторінці, яку сайт обробив; 3) Controller – це етап оброблення даних, які надав користувач для взаємодії, далі вони переходять до моделі. Після цих етапів користувач отримує результат запиту, будь-то перехід за посиланням чи то заповнення форми замовлення або керування панеллю адміністратора на сайті. Поділ на три основні групи компо-

нентів (моделі, подання та контролери) дозволяє реалізувати принципи поділу завдань.

Перевагами ASP.NET Core є:

- кросплатформність: безперешкодне розгортання застосунків на різних операційних системах (Windows, Linux, MacOS) з оптимізацією та прямою підтримкою;
- висока швидкість та продуктивність забезпечується різними технологіями: JIT-компіляція дозволяє прискорити завантаження і виконання програми; кешування даних скорочує час доступу до них та покращує продуктивність програм; механізми оптимізації запитів до бази даних зменшують кількість звернень до бази даних та прискорюють виконання запитів; асинхронні операції обробки запитів ефективно впливають на використання ресурсів сервера і дозволяють обробляти запити одночасно; збирач сміття оптимізує і дозволяє знизити навантаження на систему, тим самим покращити продуктивність [3];
- можливість створення застосунків під різні патерни та архітектури: розробка проєкту створена за модульною системою, що дозволяє використовувати лише необхідні компоненти;
- вбудована система Dependency Injection: ця технологія спрощує роботи з залежностями у різних частинах коду та робить їх більш незалежними;
- сучасна архітектура та підтримка SignalR робить з'єднання клієнтської та серверної частини простішим, SignalR виступає обгорткою, яка забезпечує усі стандарти зв'язку server-client та сама вибирає, за яким стандартом буде здійснено зв'язок за мінімальний час та мінімальною кількістю ресурсів;
- масштабованість: у випадку, коли застосунок не справляється з навантаженням, то в ньому є можливість збільшення ресурсів. ASP.NET Core має інтегровану підтримку розробки API та мікросервісних програм, що дозволяє розробникам створювати масштабовані проєкти;
- тестування: для детальної перевірки коду за допомогою unit-тестів необхідно багато часу та ресурсів. Платформа ASP.NET Core підтримує інтеграційне тестування з використанням платформ модульного тестування та вбудованого вебсервера тестування (TestServer), який можна використовувати для обробки запитів без навантаження на мережу. Це дозволяє виконувати тести швидше, ніж під час роботи з реальними вебсерверами.

З іншого боку, сучасний фреймворк ASP.NET Core із багатьма новітніми концепціями та функціями може видатись складним для розробників, які не знайомі з ним. Розробка вебзастосунків засобами ASP.NET Core вимагає високого рівня кваліфікації програміста. Тому до певних недоліків ASP.NET Core

можна віднести: круту криву навчання, обмежену підтримку бібліотек, накладні витрати на продуктивність, складне розгортання, проблеми сумісності, обмежену підтримку Windows. До того ж хоча ASP.NET Core є безплатним з відкритим кодом, використання його у робочому середовищі може вимагати додаткових витрат на інструменти, інфраструктуру та інші ресурси [4].

Зважаючи на численні переваги та враховуючи певну специфіку розробки, можна сказати що ASP.NET Core має все необхідне для створення сучасних вебзастосунків з великим навантаженням. На сьогодні ASP.NET Core – це універсальна система веброзробки, яка має гарну підтримку спільноти розробників (https://github.com/aspnet, https://www.facebook.com/groups/2128178990740761/, https://github.com/dotnetcore, https://devblogs.microsoft.com/dotnet, https://dev.to/t/dotnet), яка постійно оновлює та покращує систему та підходи до розробки.

Список використаних джерел:

1. Overview of ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0.
2. Overview of ASP.NET Core MVC. URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-7.0.
3. Бондаренко С. Що таке ASP.NET? Принцип функціонування та моделі розробки. URL: https://highload.today/uk/shho-take-asp-net-printsip-funktsionuvannya-ta-modeli-rozrobki/
4. What are the disadvantages of ASP.NET Core?. URL: https://www.quora.com/What-are-the-disadvantages-of-ASP-NET-Core

Ключові слова: веброзробка, кросплатформність, ASP.NET Core, MVC.
Keywords: web development, cross-platform, ASP.NET Core, MVC.

Науковий керівник: к. ю. н., доцент Трофименко О.Г.