Державний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЄКТ

на тему:
«Клієнт-серверний додаток буккросингу»

Студента 4 курсу, 7 групи, спеціальності 121 «Інженерія програмного забезпечення» освітньої програми «Інженерія програмного забезпечення» _____ підпис студента — Жихора Романа Юрійович

Науковий керівник, асистент кафедри інженерії програмного забезпечення _____ підпис керівника — Хорольська Карина Вікторівна

Гарант освітньої програми кандидат технічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис гаранта — Рзаєва Світлана Леонідівна

КИЇВ – 2023

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь бакалавр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«14» листопада 2022 р.

Завдання
на випускний кваліфікаційний проєкт студентові
Жихору Роману Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проєкту «Клієнт-серверний додаток буккросингу»

Затверджена наказом ректора від «6» грудня 2022 р. № 3288

2. Строк здачі студентом закінченого проєкту 5 червня 2023

3. Цільова установка та вихідні дані до проєкту

Мета проєкту розробка клієнт орієнтований додаток для буккросингу та створення бази даних для його коректної роботи.

Об'єкт дослідження: клієнт-серверний додаток для буккросингу.

Предмет дослідження: розробка дизайну, вибір технологій для створення вебсайту для буккросингу та розробка вебсайту.

4. Консультанти проєкту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

КИЇВ – 2023

5. Зміст випускного кваліфікаційного проєкту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАСТОСУВАННЯ КЛІЄНТ-СЕРВЕРНОГО ДОДАТКА

КЛІЄНТ-СЕРВЕРНОГО ДОДАТКА

1.1. Предметна область клієнт-серверного додатка

1.2. Обґрунтування вибору веб платформи

1.3. Обґрунтування вибору мови програмування C#

1.4. Обґрунтування вибору сервера MS SQL Express

1.5. Технічне завдання

1.6. Висновок до розділу 1

РОЗДІЛ 2. ВИБІР СЕРЕДОВИЩА РОЗРОБКИ ТА ПРОЄКТУВАННЯ МОБІЛЬНОГО ДОДАТКА

2.1. Вибір середовища розробки

2.2. Архітектура додатка

2.3. Проєктування бази даних

2.4. Макет програмного вебдодатка

2.5. Висновок до розділу 2

РОЗДІЛ 3. РОЗРОБКА КЛІЄНТ-СЕРВЕРНОГО ДОДАТКА

3.1. Встановлення необхідного програмного забезпечення

3.2. Створення проєкту та налаштування сервісів розробки

3.3. Створення моделей, бази даних і табличок

3.4. Створення контролерів та логіки клієнт-серверного додатка

3.5. Верстка сторінки Razor pages та додавання стилів

3.6. Результат роботи

3.7. Висновок до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

КИЇВ – 2023

6. Календарний план виконання проєкту

№ пор.	Назва етапів випускного кваліфікаційного проєкту	Строк виконання етапів проєкту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускного кваліфікаційного проєкту*	*21.09.2022*	*21.09.2022*
2.	*Розробка та затвердження завдання на проєкт*	*14.11.2022*	*14.11.2022*
3.	*Вступ та перелік літературних джерел*	*23.12.2022*	*23.12.2022*
4.	*Розділ 1. Аналіз предметної області застосування клієнт-серверного додатка*	*27.01.2023*	*27.01.2023*
5.	*Розділ 2. Вибір середовища розробки та проєктування мобільного додатка*	*03.03.2023*	*03.03.2023*
6.	*Розділ 3. Розробка клієнт-серверного додатка*	*14.04.2023*	*14.04.2023*
7.	*Висновки*	*28.04.2023*	*28.04.2023*
8.	*Здача випускного кваліфікаційного проєкту на кафедру (перша перевірка)*	*17.05.2023*	*17.05.2023*
9.	*Підготовка автореферату та презентації доповіді*	*26.05.2023*	*26.05.2023*
10.	*Попередній захист випускного кваліфікаційного проєкту*		
11.	*Зовнішнє рецензування випускного кваліфікаційного проєкту*		
12.	*Здача прошитого випускного кваліфікаційного проєкту на кафедру*		
13.	*Публічний захист випускного кваліфікаційного проєкту*		

7. Дата видачі завдання «14» листопада 2022 р.

8. Науковий керівник випускного кваліфікаційного проєкту Хорольська К.В.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Рзаєва С.Л.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Жихор Р.Ю.
(прізвище, ініціали, підпис)

КИЇВ – 2023

11. Відгук керівника випускного кваліфікаційного проєкту

Науковий керівник випускного кваліфікаційного проєкту Хорольська К. В.

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проєкт

Випускний кваліфікаційний проєкт студента Жихор Р.Ю.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми Рзаєва С.Л.

(прізвище, ініціали, підпис)

Завідувач кафедри Криворучко О. В.

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

КИЇВ – 2023

АНОТАЦІЯ

Відповідно до мети дослідження робота присвячена дослідженню процесу обміну книгами серед прихильників читання друкованої літератури та розробці клієнт-серверного вебдодатка для буккросингу, який сприятиме полегшенню та доступності обміну книгами.

Розробка серверної частини виконана у середовищі розробки Visual Studio, та використаний SQL Server Express.

Ключові слова: буккросинг, вебдодаток, клієнт-серверний додаток, Visual Studio 2022, SQL server express, c#, entity framework, bootstrap, CSS, razor pages, ASP.NET MVC.

ABSTRACT

According to the purpose of the study, the work is devoted to the research of the book exchange process among the fans of reading printed literature and the development of a client-server web application for book crossing, which will facilitate and make book exchange easier and more accessible.

The server side was developed in the Visual Studio development environment and SQL Server Express was used.

Keywords: book crossing, web application, client-server application, Visual Studio 2022, SQL server express, c#, entity framework, bootstrap, CSS, razor pages, ASP.NET MVC.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення

EF – Entity Framework

SQL – Sequence Query Language

MVC – Model-View-Controller

CSS – Cascading Style Sheets

SSMS – SQL Server Management Studio

СКБД – система керування бази даних

БД – база даних

HTTPS - HyperText Transfer Protocol Secure

SSL - Secure Sockets Layer

ПК – персональний комп’ютер

ЗМІСТ

ВСТУП **3**

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАСТОСУВАННЯ КЛІЄНТ-СЕРВЕРНОГО ДОДАТКА **8**

1.1. Предметна область клієнт-серверного додатка 8

1.2. Обгрунтування вибору веб платформи 8

1.3. Обгрунтування вибору мови програмування C# 9

1.4. Обгрунтування вибору сервера MS SQL Express 10

1.5. Технічне завдання 10

1.5.1.Загальні відомості 10

1.5.2. Мета та призначення створення системи 11

1.5.3. Вимоги до системи 11

1.5.4. Вимоги до надійності та продуктивності системи 13

1.5.5. Вимоги до безпеки системи 13

1.5.6. Вимоги до інтерфейсу користувача 13

1.5.7. Вимоги до програмного забезпечення 14

1.5.8. Вимоги до технічного забезпечення 14

1.6. Висновки до розділу 1 14

РОЗДІЛ 2 ВИБІР ПРОГРАМНИХ ЗАСОБІВ ТА ПРОЄКТУВАННЯ ВЕБДОДАТКА **16**

2.1. Вибір середовища розробки 16

2.2. Архітектура додатка 17

2.3. Проєктування бази даних 24

2.4. Макет вебдодатка 29

2.5. Висновки до розділу 2 30

РОЗДІЛ 3 РОЗРОБКА КЛІЄНТ-СЕРВЕРНОГО ДОДАТКА **31**

3.1. Встановлення необхідного програмного забезпечення 31

3.2. Створення проєкту та налаштування сервісів розробки 35

3.3. Створення моделей, бази даних і табличок 39

3.4. Створення контролерів та логіки клієнт-серверного додатка 41

3.5. Верстка сторінки Razor pages та додавання стилів 43

					ДТЕУ 121 07-12.БР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		23.12.22	Клієнт-серверний додаток буккросингу	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Хорольська К.В		23.12.22		*З*	*3*	*53*
Гарант		Рзаєва С.Л.		23.12.22		*Факультет інформаційних технологій 4 курс, 7 група*		
Розробив		Жихор Р. Ю.		23.12.22	*Зміст*			

3.6. Результат роботи.....45
3.7. Висновок до розділу 3.....50
ВИСНОВКИ ТА ПРОПОЗИЦІЇ.....51
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....52
ДОДАТКИ.....54

ВСТУП

Актуальність. Кількість людей, які захоплюються літературою, є надзвичайно великою. Це не просто ті, хто люблять почитати, а ті, хто надає особливу перевагу друкованим книгам. В цьому є свій особливий сенс та шарм. Завдяки цифровим технологіям люди можуть ділитися своїми емоціями в соціальних мережах після прочитання різноманітних книг, які їх надихнули. Таким чином, вони стимулюють інших читати іноді навіть ті книги, які не є дуже популярними.

Збільшення кількості читачів своєю чергою призводить до збільшення ринку книг в Україні. Інтернет-тренди, флешмоби та знаменитості хвиля за хвилею залучають людей до читання. Читання книг стає модним.

Буккросинг зародився у далекому 2001 році, і з кожним роком набуває все більшої популярності. Обмін книгами для деяких став буденністю, а для інших – відкриттям. Немає сумнівів, що ренесанс читання серед населення призводить до збільшення популярності вищеописаного процесу як офлайн, так і онлайн, за допомогою клієнт-серверного додатка буккросингу «SwitchBook».

Задоволення від читання книг – це одне з найприємніших відчуттів для людей, які люблять літературу. Для багатьох людей читання є засобом знайти затишок та покій у сучасному світі, що повністю насичений електронікою та різноманітними віртуальними розвагами.

Цифрові технології відкрили нові можливості для любителів книг, дозволяючи ділитися своїми враженнями від прочитання книг зі світом через соціальні мережі та інші вебресурси. Завдяки цьому стало можливим швидко

та просто знайти нових читачів та дізнатися про нові книги, які можна почитати.

Однак, не зважаючи на розширені можливості цифрової епохи, багато людей продовжують віддавати перевагу друкованим книгам. Вони підкреслюють, що запах та текстура паперу, а також відчуття перегортання сторінок – це нічого не замінить. Тому, друкування та видавництва книг залишаються дуже важливими складовими культурного життя в нашому суспільстві.

Збільшення кількості читачів літератури має дуже важливий ефект на ринок книг в Україні та інших країнах. Набір нових читачів спонукає видавництва видавати нові книги та перевидавати вже популярніші. Крім того, це дозволяє розвивати нові технології та підходи до видавничої справи, що своєю чергою приводить до появи нових ініціатив та проєктів у галузі культури та мистецтва.

Буккросинг став однією з таких ініціатив, що набула широкого розповсюдження серед любителів книг. Це спеціальна практика обміну книгами між людьми, яка дозволяє знайти нових читачів для старих книг, а також отримати нові книги для себе. Буккросинг зазвичай відбувається за допомогою спеціальних сайтів та програм, де можна зареєструвати свою книгу та вказати місце, де її можна забрати.

Ця ініціатива не тільки допомагає зберегти середовище, зменшуючи кількість відходів від друкованих книг, але й сприяє популяризації книг та залученню нових читачів. Буккросинг може бути особливо корисним для людей, які мешкають у віддалених районах, де немає доступу до бібліотек та книжкових магазинів.

Отже, любов до книг залишається важливим елементом культурного життя в сучасному світі. Незалежно від того, чи віддається перевага друкованим книгам або електронним, читання книг є одним з найкращих

способів відпочинку та розвитку для нас усіх, а клієнт-серверний додаток «SwitchBook» допоможе читачам у цьому.

Клієнт-серверний додаток буккросингу є актуальним через відносно сталу кількість прихильників читання друкованої літератури, зростання популярності буккросингу, а розробка вебсайту зможе значно полегшити процес обміну книгами та зробити його більш доступним, а також вебсайт буккросингу може мати значний потенціал для розширення.

Метою випускної кваліфікаційної роботи розробка клієнт орієнтований додаток для буккросингу та створення бази даних для його коректної роботи.

Метою дослідження є дослідити процес обміну книгами любителями читання друкованої літератури, розробити алгоритм впровадження схожої схеми в цифрових технологіях та реалізувати його у вигляді вебсайту.

Об'єктом дослідження є клієнт-серверний додаток для буккросингу.

Предметом дослідження є розробка дизайну, вибір технологій для створення вебсайту для буккросингу та розробка вебсайту.

Відповідно до мети дослідження поставлені наступні *завдання*:

- Вибір та встановлення інструментарію розробки;
- Вибір та встановлення SQL сервера;
- Створення архітектури додатка;
- Створення фізичної та логічної моделі бази даних;
- Створення макета вебдодатка;
- Розробка вебдодатка;

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАСТОСУВАННЯ КЛІЄНТ-СЕРВЕРНОГО ДОДАТКА

1.1. Предметна область клієнт-серверного додатка

Предметною областю є буккросинг - це система обміну книгами між читачами. Буккросинг дає можливість знайти нових читачів для книг, відкрити для себе нову літературу, розширити свої знання і познайомитися з новими людьми.

Основна мета буккросингу полягає в тому, щоб кожен читач зміг знайти книгу, яку він хоче прочитати, а також поділитися своїми книгами з іншими читачами. Буккросинг дозволяє ефективно використовувати ресурси, оскільки тим, хто хоче прочитати книгу, не доводиться купувати її, а тим, хто хоче поділитися книгою, не доводиться зберігати її без використання.

Сфери застосування буккросингу можуть бути різноманітними: від домашніх бібліотек до шкіл, університетів, бібліотек та книжкових магазинів. Буккросинг може бути використаний як в малих спільнотах, так і великих містах з багатою культурною та освітньою інфраструктурою. Більш того, буккросинг може бути корисним не тільки для дорослих, але і для дітей, щоб розвивати їх інтерес до читання та розширювати свої знання.

1.2. Обґрунтування вибору веб платформи

Було вирішено розробляти саме веб застосунок, а не програму під конкретну операційну систему. Цьому слугували такі причини як кросплатформеність вебдодатків та той факт що веб додаток не обов’язково

встановлювати на гаджет, адже у сучасних смартфонах та комп'ютерах встановлений браузер.

Отже, веб застосунок не займає додаткове місце у пам'яті смартфона чи то ноутбука, що може бути критичним для старих моделей гаджетів, чи може спричинити додаткові складнощі на шляху до обміну книг, тому що встановлення додатків займає час.

1.3. Обґрунтування вибору мови програмування C#

Отже, як відомо з минулого пункту створення вебсайту дозволяє запускати програмний продукт на будь-якій операційній системі, де можна встановити веббраузер. Тож незалежно від того, який гаджет є у користувача: мобільний чи ПК, він може користуватися сайтом, який буде оптимізований під пристрої з різною шириною екрана.

Існує безліч мов програмування які можуть бути використані для розробки вебсайту буккросингу. Через ряд причин було обрано мову програмування C# у якості мови, якою писатиметься Backend частина застосунку. Переваги використання C# включають:

- Строга типізація: C# є строго типізованою мовою програмування, що дозволяє зменшити кількість помилок в програмі та забезпечує безпечну роботу з даними.
- Об'єктноорієнтований підхід: C# базується на об'єктноорієнтованому підході, що дозволяє забезпечити легкість управління кодом, розширення та модифікацію.
- Платформно-незалежний код: Код, написаний на C#, може бути виконаний на будь-якій платформі, що підтримує відповідну віртуальну машину.

Фреймворк Asp.Net дозволяє забезпечити високу продуктивність вебдодатків та простоту управління. Деякі переваги використання Asp.Net включають:

- Безпека: Asp.Net має вбудовану систему безпеки, яка дозволяє забезпечити захист вебдодатків від різноманітних загроз.
- Масштабованість: Asp.Net дозволяє легко масштабувати вебдодатки та забезпечує швидкий доступ до даних.
- Підтримка баз даних: Asp.Net дозволяє взаємодіяти з різними базами даних, зокрема з базами даних SQL, що дозволяє забезпечити ефективне зберігання та управління даними.

Таким чином, використання мови програмування C# та фреймворку Asp.Net дозволить забезпечити високу продуктивність, безпеку та масштабованість клієнт-серверного додатка для буккросингу.

1.4. Обґрунтування вибору сервера MS SQL Express

Вибір сервера бази даних надзвичайно важливий, адже на сервері БД зберігатимуться всі дані з вебсайту: книжки, обміни чи то користувачі. Усе наповнення сайту тягнеться та налаштовується саме з бази даних.

Для кращої сумісності було обрано безплатний сервер реляційної бази даних MS SQL Express 2019.

Його легко встановлювати, можна керувати сервером через SQL Server Management Studio, а також широкий функціонал.

1.5. Технічне завдання

1.5.1.Загальні відомості

1.1. Найменування системи

Повне найменування системи: клієнт-серверний додаток буккросингу «SwitchBook».

Скорочене найменування системи: вебсайт «SwitchBook».

1.2. Планові терміни початку та закінчення робіт

Початок розробки запланований на 30.01.2023р, закінчення робіт до 31.06.2023р.

1.3. Порядок оформлення і пред'явлення результатів робіт

Результати робіт представляються у вигляді відповідних документів та відповідному програмному продукті.

1.4. Головний бенефіціар та потенційні користувачі системи

Головним бенефіціаром є власник системи. Потенційними користувачами є всі охочі здійснювати обмін книгами.

1.5.2. Мета та призначення створення системи

- Призначення системи

Створення онлайн-сервісу з обміну книгами (буккросинг).

- Мета створення системи

Надати можливість користувачам здійснювати обмін фізичними книгами онлайн.

1.5.3. Вимоги до системи

3.1. Вимоги до структури та функціонування системи

Структура системи має бути логічною та зрозумілою для користувача.

Система повинна містити такі підсистеми:

- авторизація користувачів,
- каталог книг,
- історія прийнятих пропозицій обміну.

Відображення списку книг повинно бути зрозумілим та зручним для користувача.

3.2 Вимоги до взаємодії з користувачем

Користувач повинен мати можливість зареєструватися та авторизуватися в системі.

Зареєстрований користувач повинен мати можливість внести книги в

каталог всіх книг, а також шукати та надсилати пропозиції обміну іншим користувачам. Після створення пропозиції обміну користувач якому ця пропозиція надійшла має право прийняти її або ж відмовитись.

3.3. Вимоги до безпеки та захисту даних

Система повинна забезпечувати безпеку та захист даних користувачів.

Користувач повинен мати можливість змінювати свій пароль та персональні дані.

Шифрування пароля повинно здійснюватися за допомогою алгоритму хешування SHA-256.

3.4. Вимоги до мов та технологій

Система має бути розроблена на фреймворку Asp.Net мовою програмування C# з використанням бази даних SQL та Entity Framework.

Для авторизації та аутентифікації користувачів має бути використано Identity Framework.

Всі елементи інтерфейсу користувача повинні бути доступні українською мовою.

3.5. Вимоги до функціональності системи

3.5.1. Реєстрація та авторизація користувачів

- Користувач повинен мати можливість зареєструватися в системі, вказавши свої персональні дані;
- Користувач повинен мати можливість авторизуватися в системі, використовуючи свій email та пароль.

3.5.2. Каталог книг

- Система повинна містити каталог книг, що складається з наступних елементів: назва книги, автор, опис, обкладинка;
- Користувач повинен мати можливість додавати нові книги до каталогу, вказуючи всі необхідні дані;
- Користувач повинен мати можливість переглядати каталог книг

та шукати книги за ключовими словами.

3.5.3. Обмін книг

Користувач повинен мати можливість відправити пропозицію обміну на книгу, яку він бажає прочитати, вказавши свої контактні дані у своєму профілі. Після цього інший користувач повинен мати можливість прийняти пропозиції обміну або їх відхилити. Після підтвердження повинна з’явитися контактна інформація про обох користувачів.

3.5.4. Робота з акаунтом користувача

Користувач повинен мати можливість змінювати свій пароль та адресу.

1.5.4. Вимоги до надійності та продуктивності системи

- Надійність

Система повинна працювати без збоїв і помилок.

- Продуктивність

Система повинна забезпечувати швидкий доступ до даних та підтримувати обробку багатьох операцій з базою даних одночасно.

1.5.5. Вимоги до безпеки системи

- Аутентифікація та авторизація

Система повинна вимагати аутентифікації та авторизації користувачів для доступу до приватної інформації та функцій системи.

- Захист даних

Система повинна забезпечувати захист конфіденційної та особистої інформації користувачів від несанкціонованого доступу та втрати даних і забезпечувати захист від шкідливого програмного забезпечення й атак на безпеку даних.

1.5.6. Вимоги до інтерфейсу користувача

- Зручність використання

Інтерфейс повинен бути зрозумілим та інтуїтивно зрозумілим для користувачів будь-якого рівня технічної грамотності. Також інтерфейс

повинен бути зручним для використання на гаджетах з різною шириною екрана такі як комп'ютери, планшети та мобільні пристрої.

- Надійність інтерфейсу

Інтерфейс повинен працювати без збоїв та помилок. Інтерфейс повинен бути простим у використанні та надійним.

1.5.7. Вимоги до програмного забезпечення

Вимоги до програмного забезпечення:

- Веб додаток повинен бути розроблений з використанням сучасних технологій (MS SQL Server, Visual Studio…) та мови програмування C#.
- Врахування масштабованості для врахування майбутніх потреб користувачів.
- Система повинна бути сумісною з різними браузерами.
- Вебсайт повинен бути захищеним від вторгнень та зловживань.
- Інтерфейс вебдодатка повинен бути простим та зрозумілим для користувачів різного рівня технічної грамотності.

1.5.8. Вимоги до технічного забезпечення

Нижче перечислені вимоги до технічного забезпечення:

- Система повинна мати достатній обсяг пам'яті та містити достатню кількість процесорів для забезпечення швидкої роботи системи.
- Система повинна мати захист від хакерських атак та інших вторгнень.

1.6. Висновки до розділу 1

Веб додаток SwitchBook повинен допомогти читачам книг знайти людину для обміну книгами (буккросингу). Додаток працюватиме на фреймворку ASP.Net та написаний мовою програмування C# задля кращого

перформансу. Для кращої сумісності між C# та реляційною базою даних використовуватиметься MS SQL Server 2019.

Чітко прописані вимоги дозволяють розпочати етап розробки архітектури, логічної та фізичної моделей бази даних та макету вебдодатка.

РОЗДІЛ 2
ВИБІР ПРОГРАМНИХ ЗАСОБІВ ТА ПРОЄКТУВАННЯ ВЕБДОДАТКА

2.1. Вибір середовища розробки

Visual Studio є інтегрованою середовищем розробки (IDE) для мови програмування C# та фреймворку Asp.Net від Microsoft, та надає повну інтеграцію між сервісами компанії що створила її.

Visual Studio має вбудовану функцію автодоповнення, яка допомагає розробникам швидше писати код завдяки зменшенню кількості рутинної роботи та запобігають помилкам тим самим спрощуючи розробку.

Функція дебагу в Visual Studio допомагає легко та швидко знаходити проблемні ділянки коду, що своєю чергою допомагає прискорити процес розробки вебдодатка. Беззаперечно ця функція є дуже корисною, адже розробникам не приходиться самостійно проходити весь шлях відпрацювання коду, замість цього середовище розробки проходиться по пайплайну та вказує ділянку коду з помилкою.

У Visual Studio вбудований Git. У середовищі розробки налагоджено клонування, пуші, пули, створення гілок, та зливання останніх.

Також можна використовувати NuGet пакети, які допомагають прискорити розробку програми таким чином, що не потрібно писати код, який вже коли-небудь був написаним та є у вільному доступі. Розробник концентрується на ключових зв'язках між додатковими бібліотеками та своїм програмним кодом, що у тандемі дає високий перформанс та якість виконання роботи.

2.2. Архітектура додатка

2.2.1. Use-case діаграма

Діаграма варіантів використання (use case diagram, діаграма сценаріїв) - це модель системи, що визначає відношення між користувачами (агентами) та варіантами використання. Use case діаграма є описом загальних функцій системи, що не визначає алгоритми та внутрішні компоненти програми. Тобто, по суті, створюємо варіації використання програми. На рисунку 2.1. зображена Use-case діаграма.

Опис діаграми:

- Користувачі можуть здійснювати пошук книг на сайті. Вони можуть шукати книги за назвою, автором або описом книги.
- Зареєстровані користувачі мають можливість створювати обміну, приймати, відмовлятися та скасовувати пропозиції обміну.
- Зареєстрований користувач має доступ до менеджменту своєї бібліотеки книг.
- Система здійснює авторизацію та аутентифікацію користувачів на сайті, а також зберігає інформацію про обміни користувачів.

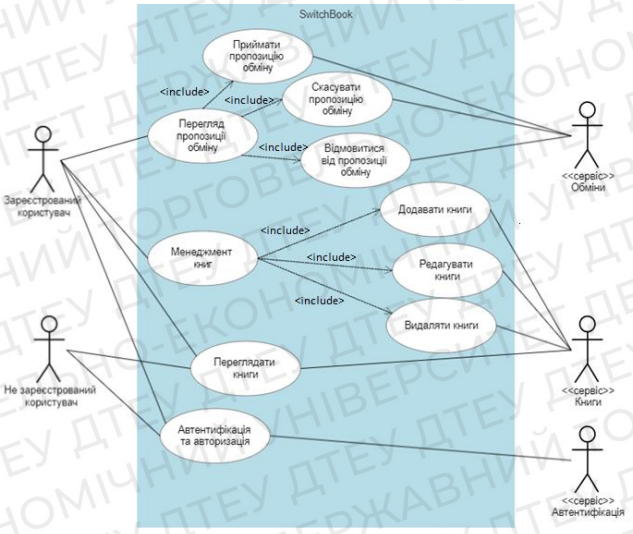

Рисунок 2.1. Use case діаграма

Джерело: побудовано автором

2.2.2. Діаграма класів

На рисунку Рис. 2.2. зображено діаграму класів. У діаграмі класів розміщено елементи, що визначають класи програми: моделі, контроллери, а всередині них визначаються методи та поля з типами даних.

Діаграма класів існує для кращого розуміння того як програма виглядає зсередини. Таким чином розробник може зрозуміти основну логіку відношення класів.

У цій діаграмі присутні такі класи як: Address, User, Book, Order, а також клас контексту даних – ApplicationDbContext, і контролери: BooksController, HomeController, MyBooksController, MyOrdersController, TradeController. У класах також прописані їхні поля та методи.

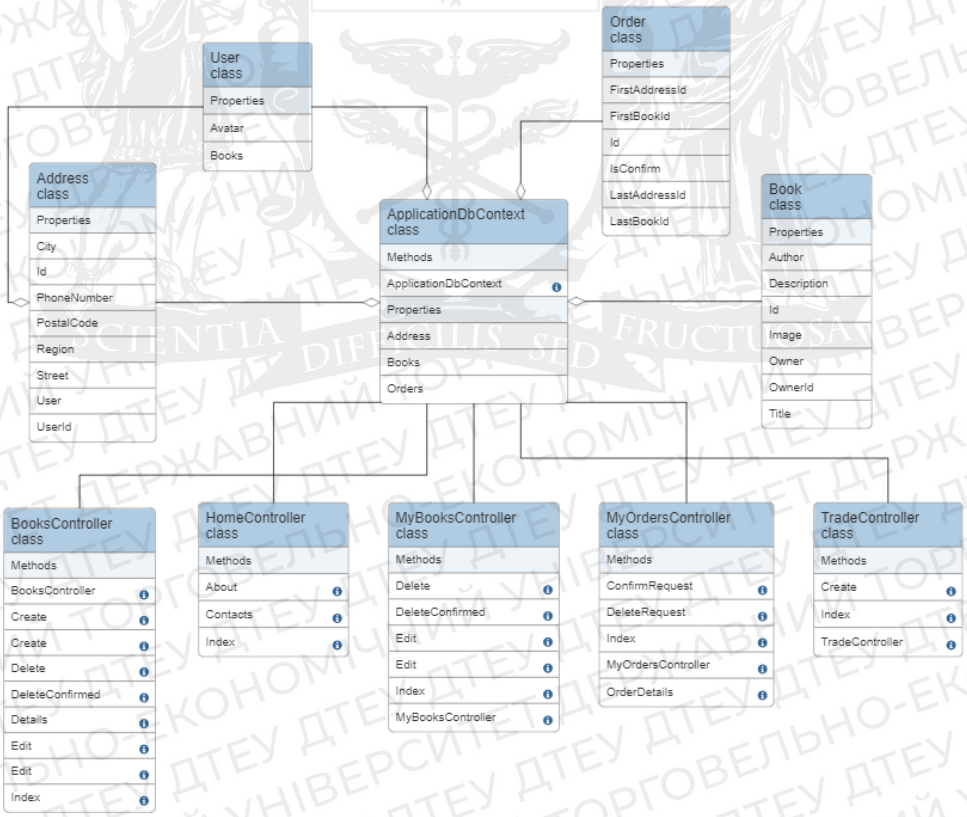

Рисунок 2.2. Діаграма класів

Джерело: побудовано автором

2.2.3. Еталонна модель

На рисунку Рис. 2.3. зображено еталонну модель вебсайту. З цієї моделі стає зрозумілим архітектурні особливості проєктування та розробки вебдодатка. Користувач взаємодіє з інтерфейсом та логікою, та, чи не найважливіше захищений такими протоколами передачі даних як HTTPS, сайт отримав SSL сертифікат. Також, повинен бути сервер зі сконфігурованою базою даних.

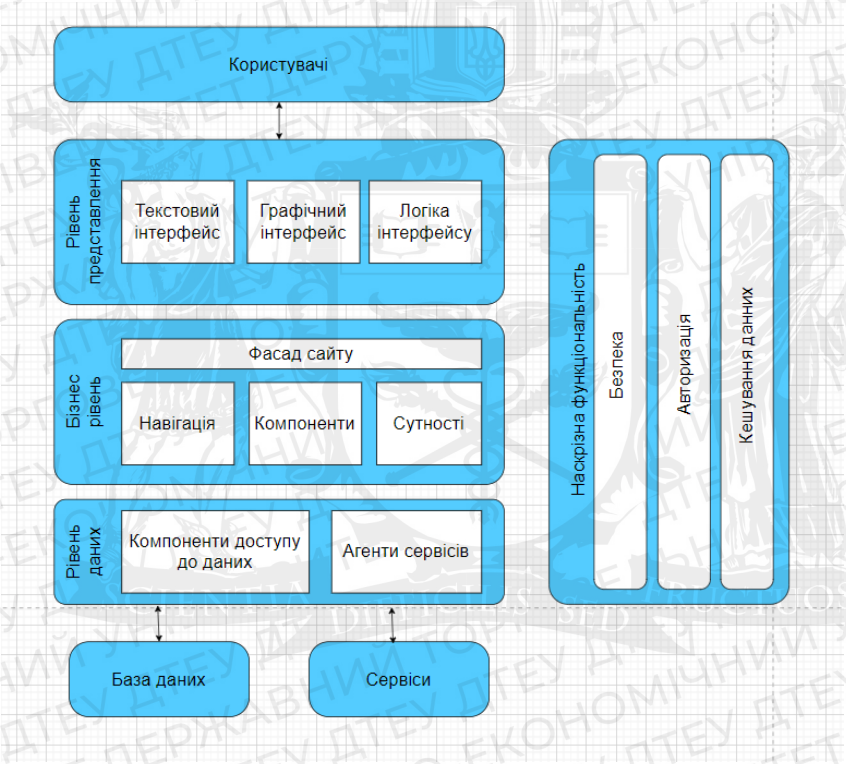

Рисунок 2.3. Еталонна модель вебсайту

Джерело: побудовано автором

2.2.4. Блок схема

На рисунку Рис. 2.4. зображено блок схему веб-проєкту. Це графічне, візуальне проєктування, що описує структурні зв'язки всіх розділів проєкту, навігацію, функціональні частини. Блок схема клієнт-серверного додатка

складається із початку та кінця, а також алгоритму за яким користувачі будуть робити обмін книгами.

Блок схеми дозволяють змоделювати майбутній веб-проєкт, що дозволяє до розробки самого проєкту продумати всі складові усунути недоліки та виділити оптимальні та успішні рішення.

Також за допомогою блок-схеми можна описати алгоритм роботи веб-проєкту, що згодом суттєво полегшить написання ТЗ та саму розробку.

Рисунок 2.4. Блок схема вебсайту

Джерело: побудовано автором

2.2.5. Діаграма послідовності

Особливості взаємодії елементів модельованої системи можуть бути зображені на діаграмах кооперації й послідовності. Діаграми кооперації

використовуються для опису функціонування систем, хоча час в явному вигляді в них відсутній. Проте часовий аспект поведінки буде мати значення коли розробляють саме синхронні процеси, які описують взаємодію об’єктів.

На діаграмі послідовності (Див. Рис. 2.5) неявно присутня вісь часу, що дозволяє візуалізувати часові відношення між повідомленнями, які передаються. За допомогою діаграми послідовності можна подати взаємодію елементів моделі як своєрідний часовий графік "життя" всієї сукупності об'єктів, зв'язаних між собою для реалізації варіанту використання системи для досягнення мети (обміну книгами).

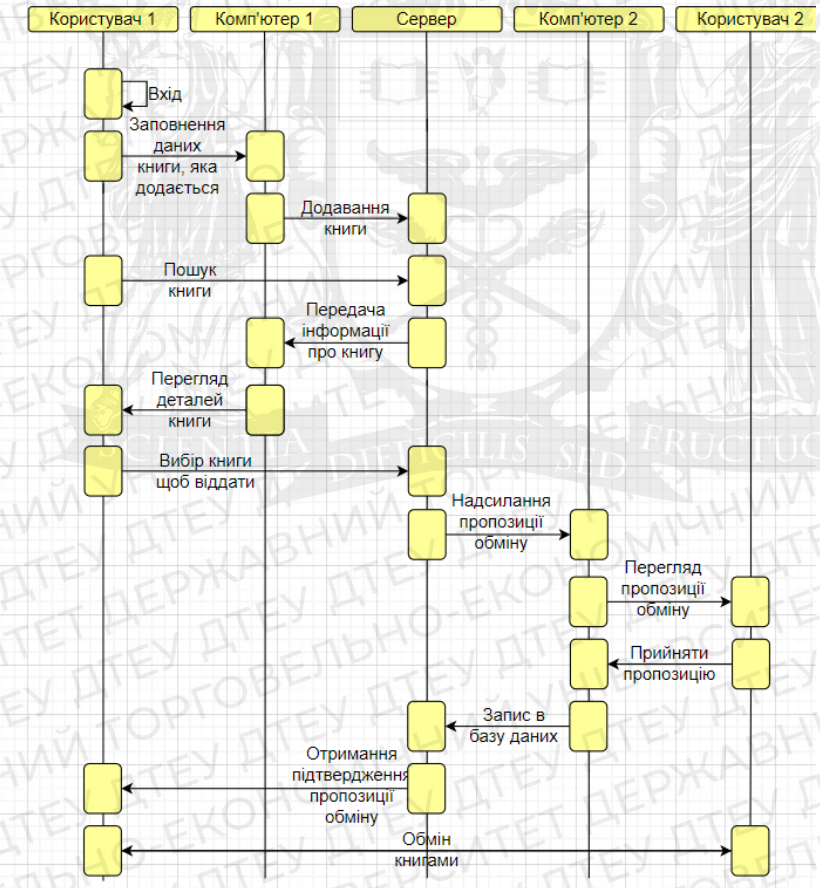

Рисунок 2.5. Діаграма послідовності

Джерело: побудовано автором

2.2.6. Діаграма стану

Діаграма станів — це діаграма, на якій зображено зміну стану виконуваної особи (об’єкта) в часі. Тобто у нас є початок, є кінець і проміжні стани, які і є елементами діаграми. Діаграма станів будується стандартизовано відповідно до правил побудови такої діаграми.Діаграму розроблюваного вебдодатка зображено на Рис. 2.6.

Елементами діаграми є:

1. Чорне коло яке позначає початковий стан.
2. Коло з кільцем навколо, яке познчає кінцевий стан об’єкта.
3. Округлений прямокутник, який означає один зі станів. Усередині прямокутника записується назва стану.
4. Стрілка, що позначає перехід.

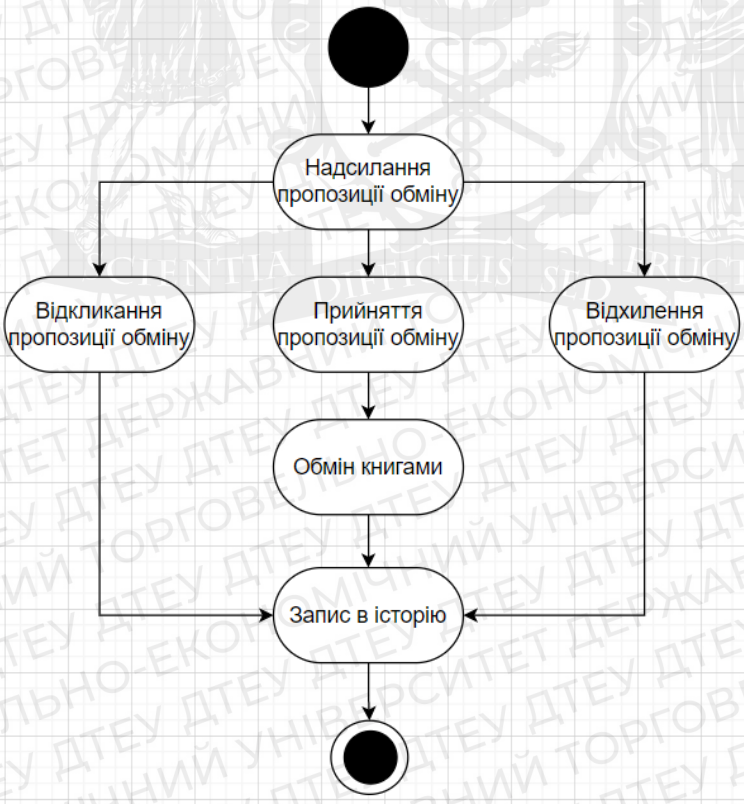

Рисунок 2.6. Діаграма стану

Джерело: побудовано автором

2.2.7. Діаграма розгортання

Діаграма розгортання — це діаграма, на якій зображено пристрої та компоненти, які необхідні для роботи програми. На цій діаграмі представлені екземпляри компонентів.

На діаграмі розгортання, що зображена на Рис. 2.7. Зображено користувача, та машину, якою він взаємодіє за допомогою пристроїв вводу-виводу інформації, а також з'єднання машини з вебсервером, у якому містяться такі компоненти як інтерфейс бази даних та вебінтерфейс для обробки алгоритмів. Своєю чергою, база даних зберігається на окремому сервері, з'єднання з якою здійснюється за допомогою рядку з'єднання у ще не скомпільованому додатку в інтегрованому середовищі розробки.

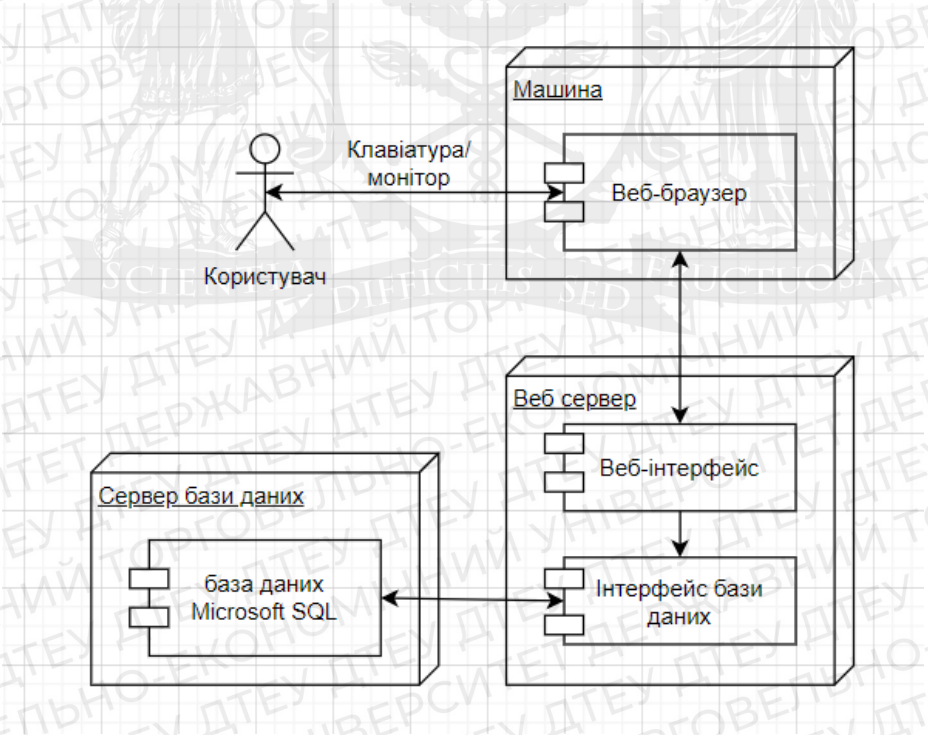

Рисунок 2.7. Діаграма розгортання і компонентів

Джерело: побудовано автором (знімок екрана)

2.3. Проєктування бази даних

База даних (БД) - це організована колекція даних, які зберігаються та використовуються для ефективного доступу до інформації. База даних містить структури даних, які описують відносини між об'єктами та дозволяють виконувати операції над цими об'єктами. Бази даних використовуються в різних галузях, таких як банківська справа, бізнес, медицина, освіта та інші.

2.3.1. Логічна модель бази даних

Логічна модель БД описує дані та взаємозв'язки детально на абстрактному рівні. В основному ця модель включає таблиці та відносини між ними (Див. Рис. 2.8).

Логічна модель даних включає первинні ключі. При створенні логічної моделі необхідно створити зв’язки між таблицями з первинним ключем та таблицями з іноземними ключами. На етапі проєктування логічної моделі також відбувається нормалізація бази даних.

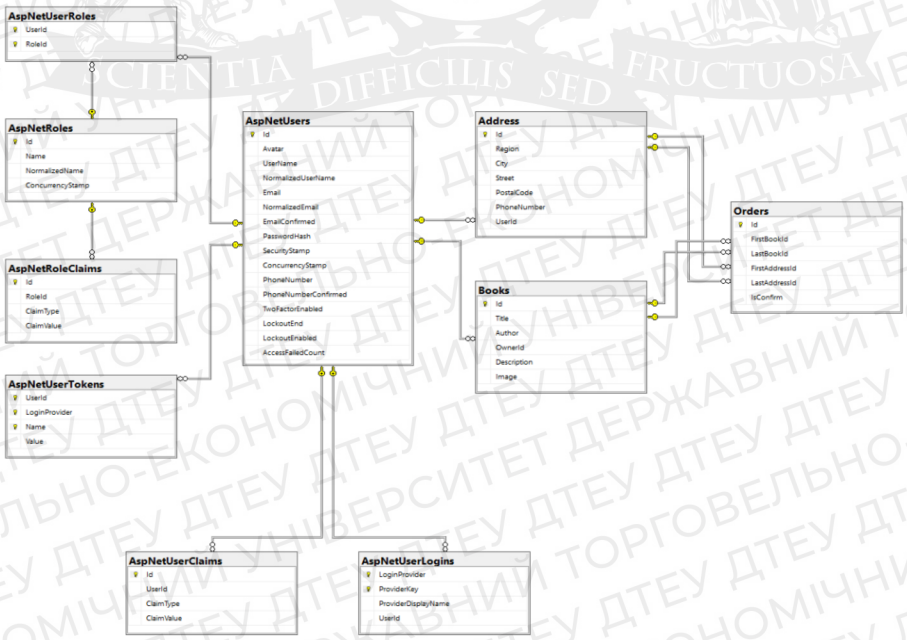

Рисунок 2.8. Логічна модель бази даних

Джерело: побудовано автором

Логічна модель бази даних - це модель, яка описує дані, таблиці та зв'язки між ними. Логічна модель є незалежною від конкретної СКБД, але враховує обмеження та можливості, які надає конкретна СКБД.

2.3.2. Фізична модель бази даних

У фізичній моделі БД розміщуються всі деталі логічної бази даних та дані, які неообхідні для її створення.

Фізична модель даних складається з назви таблиці, ключів, індексованого поля, типу даних та чи може приймати поле нульове значення.

Таблиця 1

AspNetUsers

Назва поля	**Ключ**	**Індексоване поле**	**Тип даних**	**Null-значення**
Id	PK	Так	nvarchar(450)	Ні
Avatar			varbinary(MAX)	
UserName			nvarchar(256)	
NormalizedUserName			nvarchar(256)	
Email			nvarchar(256)	
NormalizedEmail			nvarchar(256)	
EmailConfirmed			bit	Ні
PasswordHash			nvarchar(MAX)	
SecurityStamp			nvarchar(MAX)	
ConcurrencyStamp			nvarchar(MAX)	
PhoneNumber			nvarchar(MAX)	
PhoneNumberConfirmed			bit	Ні
TwoFactorEnabled			bit	Ні
LockoutEnd			datetimeoffset(7)	
LockoutEnabled			bit	Ні
AccessFailedCount			bit	Ні

Джерело: побудовано автором

Таблиця 2

Address

Назва поля	Ключ	Індексоване поле	Тип даних	Null-значення
Id	PK	Так	int	Ні
UserId	FK		nvarchar(450)	
Region			nvarchar(MAX)	
City			nvarchar(MAX)	
Street			nvarchar(MAX)	
PhoneNumber			nvarchar(MAX)	
PostalCode			nvarchar(MAX)	

Джерело: побудовано автором

Таблиця 3

Orders

Назва поля	Ключ	Індексоване поле	Тип даних	Null-значення
Id	PK	Так	int	Ні
FirstBookId	FK		int	Ні
LastBookId	FK		int	Ні
FirstAddressId	FK		int	Ні
LastAddressId	FK		int	Ні
IsConfirm			bit	Ні

Джерело: побудовано автором

Таблиця 4

AspNetUserRoles

Назва поля	Ключ	Індексоване поле	Тип даних	Null-значення
UserId	FK		nvarchar(450)	Ні
RoleId	FK		nvarchar(450)	Ні

Джерело: побудовано автором

Таблиця 5

Books

Назва поля	Ключ	Індексоване поле	Тип даних	Null-значення
Id	PK	Так	int	Ні
Title			nvarchar(MAX)	
Author			nvarchar(MAX)	
OwnerId	FK	Так	nvarchar(450)	
Description			nvarchar(MAX)	
Image			varbinary(MAX)	

Джерело: побудовано автором

Таблиця 6

AspNetUserTokens

Назва поля	Ключ	Індексоване поле	Тип даних	Null-значення
UserId	FK		nvarchar(450)	Ні
LoginProvider			nvarchar(128)	
Name			nvarchar(128)	
Value			nvarchar(MAX)	

Джерело: побудовано автором

Таблиця 7

AspNetUserLogins

Назва поля	Ключ	Індексоване поле	Тип даних	Null-значення
LoginProvider			nvarchar(128)	Ні
ProviderKey			nvarchar(128)	Ні
ProviderDisplayName			nvarchar(MAX)	
UserId	FK	Так	nvarchar(450)	Ні

Джерело: побудовано автором

Таблиця 8

AspNetUserClaims

Назва поля	Ключ	Індексоване поле	Тип даних	Null-значення
Id	PK	Так	int	Ні
UserId	FK	Так	nvarchar(450)	Ні
ClaimType			nvarchar(MAX)	
ClaimValue			nvarchar(MAX)	

Джерело: побудовано автором

Таблиця 9

AspNetRoles

Назва поля	Ключ	Індексоване поле	Тип даних	Null-значення
Id	PK	Так	nvarchar(450)	Ні
Name			nvarchar(256)	
NormalizedName			nvarchar(256)	
ConcurrencyStamp			nvarchar(MAX)	

Джерело: побудовано автором

Таблиця 10

AspNetRoleClaims

Назва поля	Ключ	Індексоване поле	Тип даних	Null-значення
Id	PK	Так	int	Ні
RoleId	FK	Так	nvarchar(450)	Ні
ClaimType			nvarchar(MAX)	
ClaimValue			nvarchar(MAX)	

Джерело: побудовано автором

Отже, фізична модель бази даних детально описує кожен стовпець таблиці бази даних. За допомогою цієї моделі можна приступати безпосередньо до створення таблиць у базі даних сервера.

2.4. Макет вебдодатка

Для створення дизайну, використовувався сервіс розробки інтерфейсів Figma від однойменної компанії.

Figma — це цифровий графічний редактор, який полегшує співпрацю [12]. У ньому можна створити прототип сайту, інтерфейс для редагування та обговорити створення сайту в реальному часі. У Figma є можливість демонструвати компоненти інтерфейсу, створювати інтерактивні веб-сайти та програми, ілюстрації та векторну графіку.

Проєкт у фігма (див. Рис. 2.9) можна переглянути за посиланням: https://www.figma.com/file/jCJ5k0pWLAVPaUZLIxg3PK/%D0%92%D0%9A%D0%A0-%D0%A1%D0%B2%D1%96%D1%87%D0%91%D1%83%D0%BA?node-id=0%3A1&t=53QtHSkmitTNWJV4-0

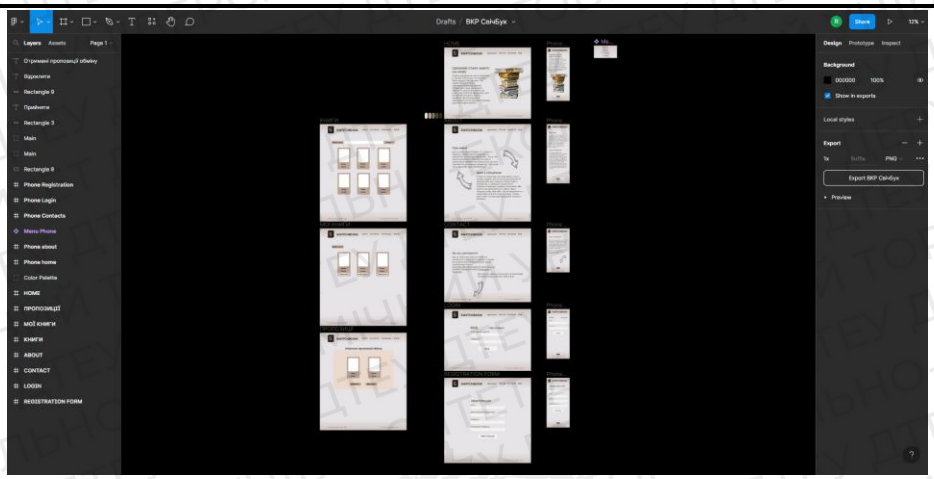

Рисунок 2.9. Проєкт Figma

Джерело: побудовано автором

Після створення дизайну можна розпочинати верстку сторінок та подальшу розробку вебсайту.

2.5. Висновки до розділу 2

Важливо пройти кожен етап та правильно сформувати усі деталі, щоб під час розробки ПЗ не виникло непередбачуваних обставин таких як зміна таблиць в БД чи зміна взаємодії користувача з сайтом, адже після розробки фінального продукту вносити зміни у вже функціонал що вже існує може потребувати більше ресурсів аніж розробка нового функціоналу.

Звісно, це все залежить від конкретної функції, та підготування до розробки це дуже важливий етап, який не можна пропускати.

У розділі 2 було обрано середовище розробки, обґрунтовано доцільність його використання, створено архітектурні діаграми клієнт-серверного додатка «SwitchBook», спроєктована логічна та фізична модель бази даних та розроблено дизайн відповідно до вимог.

РОЗДІЛ 3

РОЗРОБКА КЛІЄНТ-СЕРВЕРНОГО ДОДАТКА

3.1. Встановлення необхідного програмного забезпечення

3.1.1. Встановлення SQL Server Express

SQL Server Express є безплатною версією бази даних SQL Server, яка буде використана для розробки вебдодатка. Щоб встановити SQL Server Express були зроблені наступні кроки:

1. Було завантажено виконуваний файл SQL Server Express з офіційного сайту Microsoft (Див. Рис. 3.1).

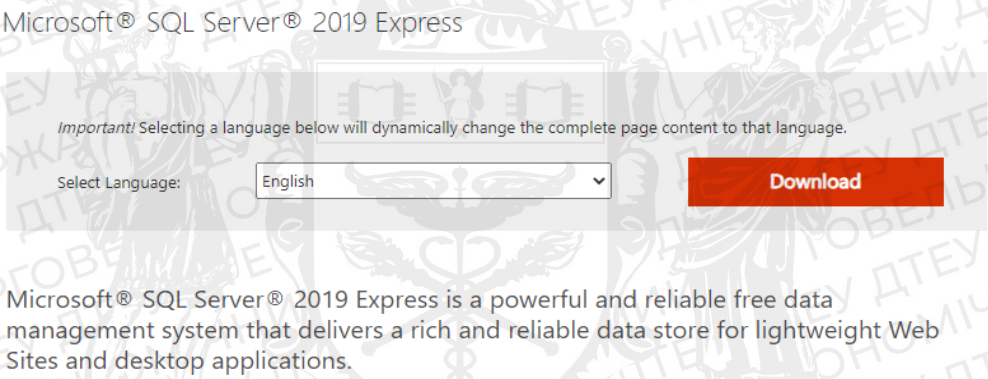

Рисунок 3.1. Частина сторінки з вибором мови та кнопкою «Завантажити»

Джерело: побудовано автором (знімок екрана)

2. Запущено виконуваний файл та встановлено новий екземпляр SQL Server Express (Див. Рис. 3.2).

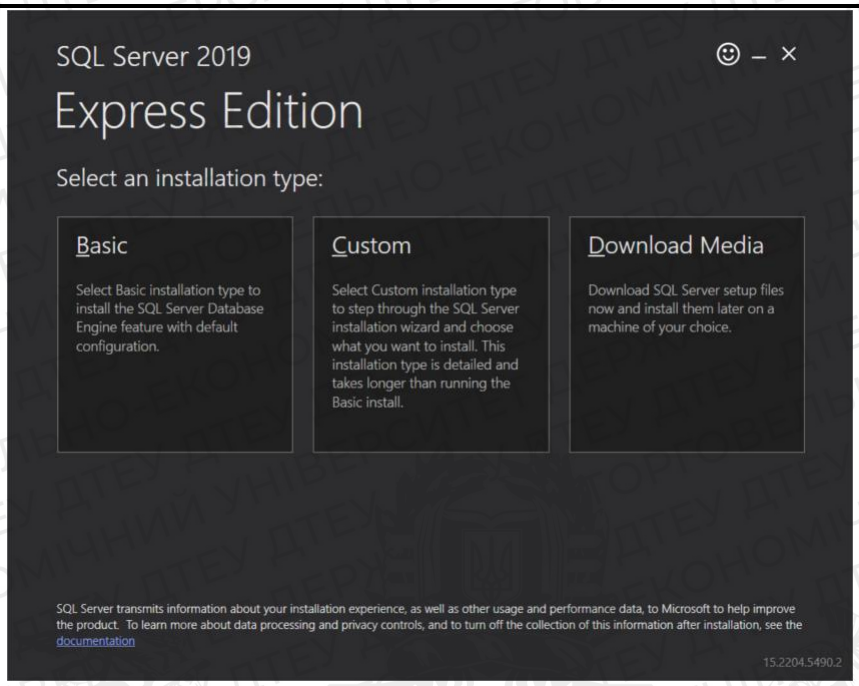

Рисунок 3.2. Початкова сторінка SQL Server 2019

Джерело: побудовано автором (знімок екрана)

3. Для розробки вебдодатків було обрано "Базову версію сервера". Прийнято ліцензійну угоду (Див. Рис. 3.3).

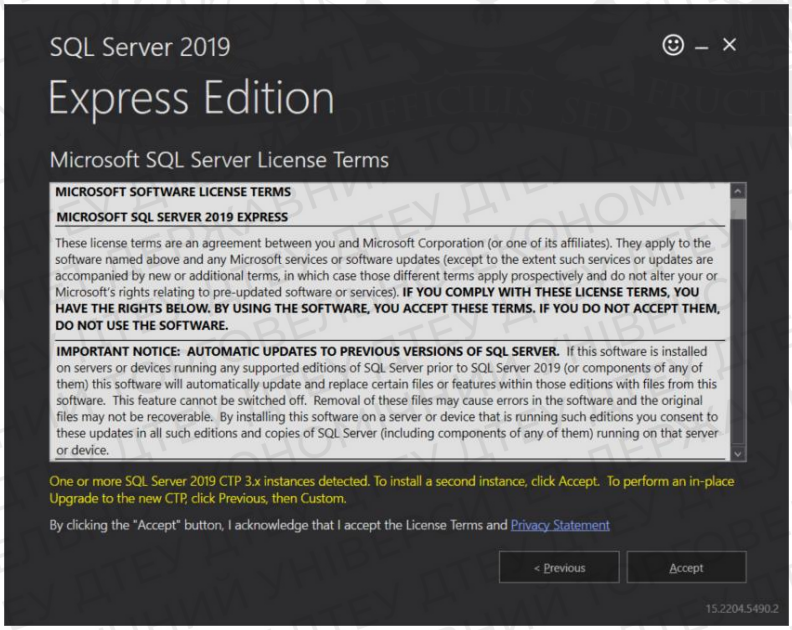

Рисунок 3.3. Прийняття ліцензійних угод SQL Server 2019 Express

Джерело: побудовано автором (знімок екрана)

4. Після завершення встановлення запущено SQL Server Management Studio та під'єднано до встановленого екземпляра SQL Server Express за допомогою вказаного імені сервера та обраних налаштувань автентифікації (Див. Рис. 3.4).

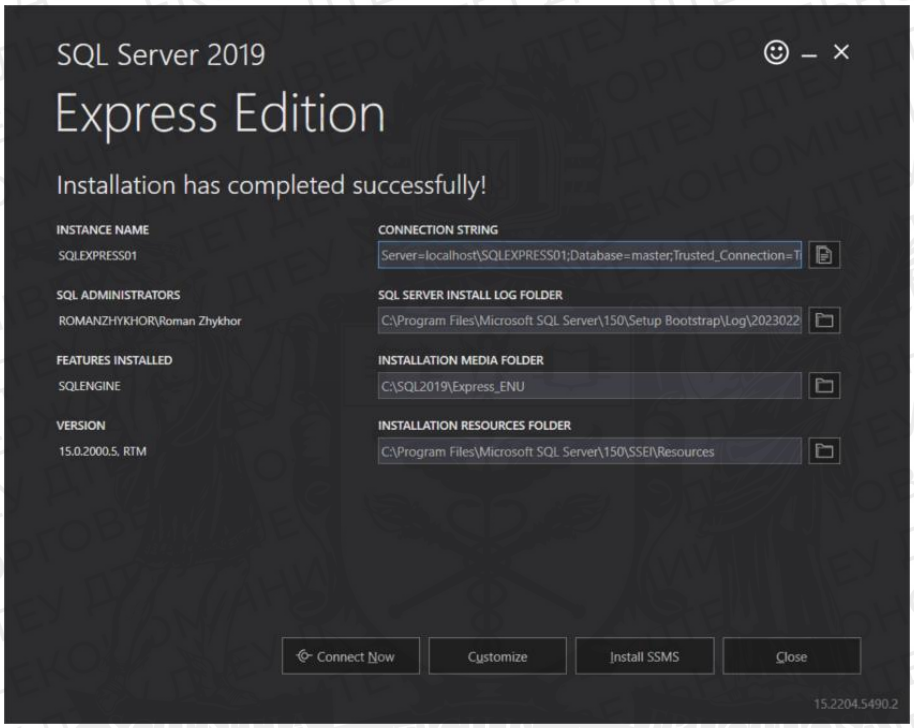

Рисунок 3.4. Вікно завершення встановлення SQL Server 2019 Express

Джерело: побудовано автором (знімок екрана)

Після успішного встановлення SQL Server Express його можна використовувати для створення баз даних, таблиць та розміщення, редагування інформації (Див. Рис. 3.5).

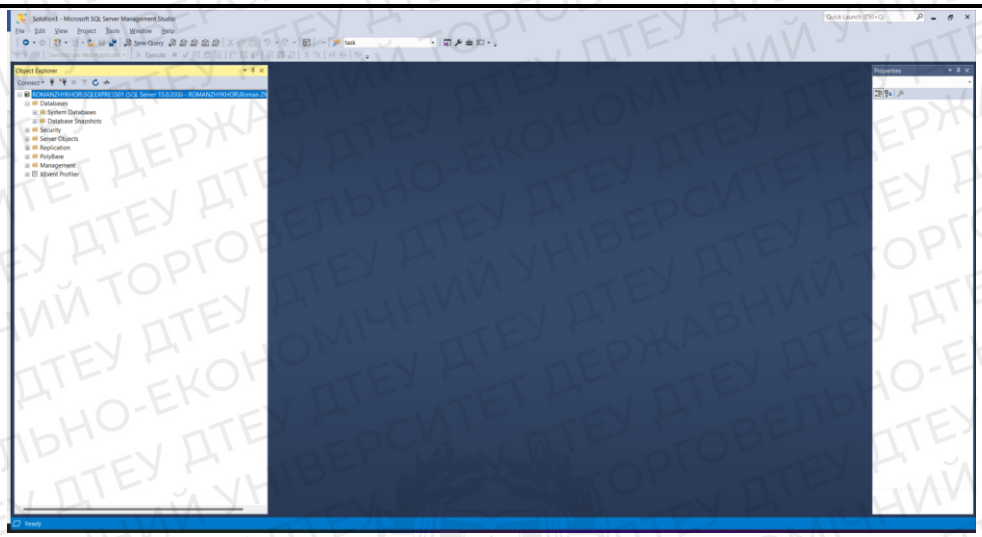

Рисунок 3.5. Вигляд встановленого сервера через SQL Server Management Studio

Джерело: побудовано автором (знімок екрана)

3.1.2. Встановлення Visual Studio 2022

Visual Studio - це головний інструмент написання алгоритмів клієнт-серверного додатка «SwitchBook».

Процес встановлення Visual Studio 2022 на Windows:

1. Спочатку потрібно завантажити встановлювач Visual Studio 2022 з офіційного вебсайту Microsoft (Див. Рис. 3.6).

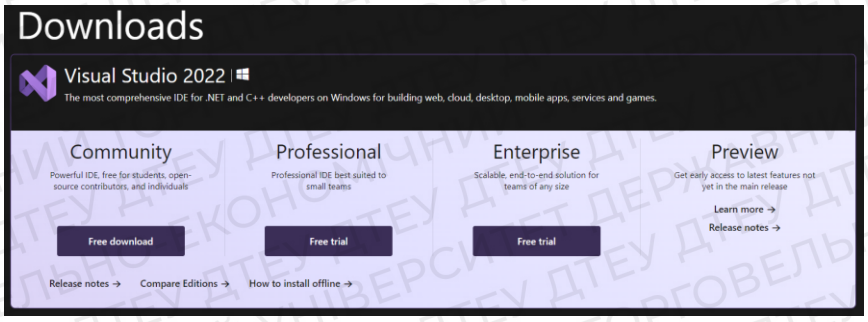

Рисунок 3.6. Частина вебсайту з кнопками завантаження встановлювача Visual Studio Installer

Джерело: побудовано автором (знімок екрана)

2. Встановлено Visual Studio Installer.

3. Обрано та встановлено компоненти які необхідні для розробки вебсайту буккросингу (Див. Рис. 3.7).

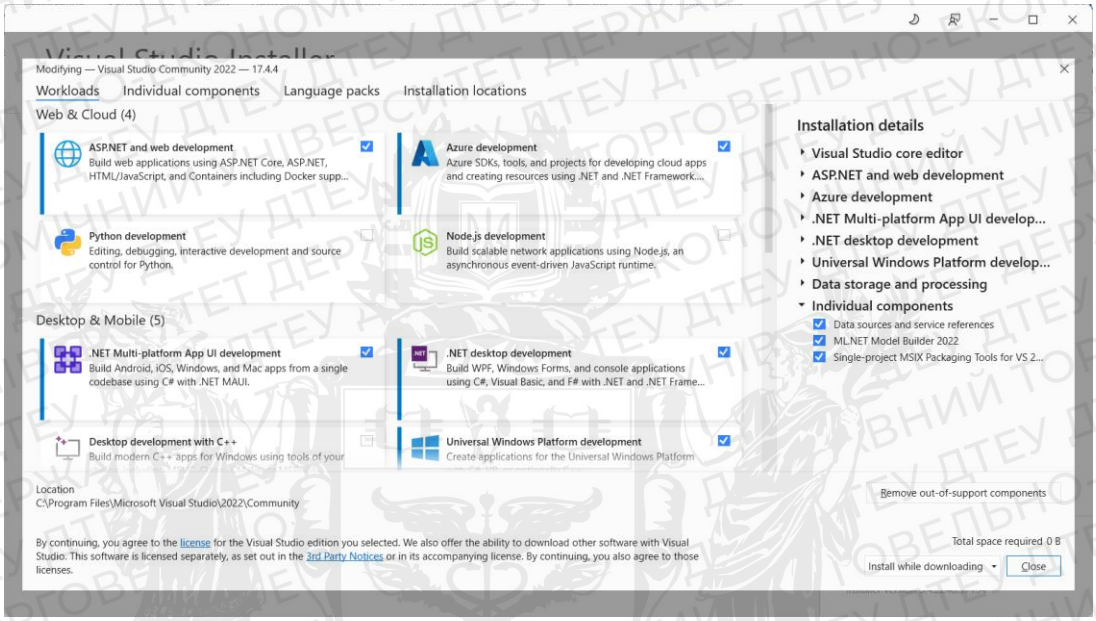

Рисунок 3.7. Вікно з вибором компонентів.

Джерело: побудовано автором (знімок екрана)

4. Після завершення встановлення було запущено Visual Studio 2022 та розпочате налаштування проєкту вебдодатка мовою програмування C# на фреймворку Asp.Net.

3.2. Створення проєкту та налаштування сервісів розробки

3.2.1. Створення проєкту

Visual Studio дозволяє обрати шаблон із заготовленим кодом для швидкого початку створення логіки програмних продуктів.

Для створення проєкту було зроблено наступні пункти:

1. Відкрито Visual Studio та натиснуто кнопку "Створити новий проєкт" на головному екрані.

2. У вікні "Створення проєкту" вибрано шаблон "ASP.NET Core Web App (MVC)".

3. Введено назву проєкту, усі параметри, та визначено місце збереження проєкту та натиснуто кнопку «Створити» (Див. Рис. 3.8).

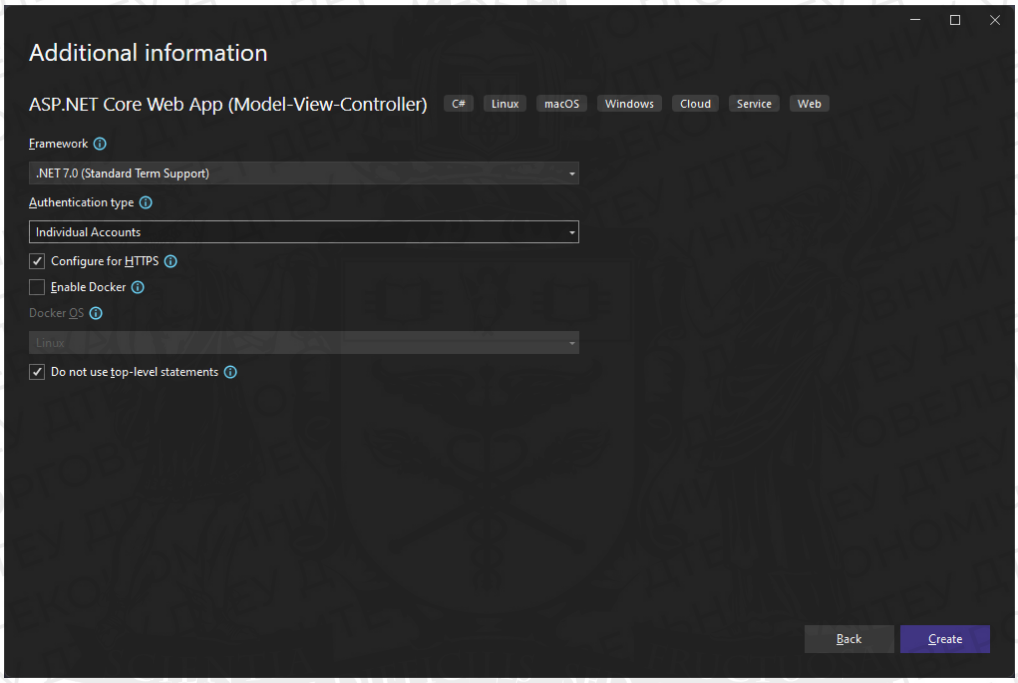

Рисунок 3.8. Створення проєкту «SwitchBook»

Джерело: побудовано автором (знімок екрана)

Таким чином було створено проєкт «SwitchBook».

3.2.2. Встановлення необхідних NuGet пакетів

Додавання NuGet пакетів до проєкту в Visual Studio зроблено наступним чином:

1. Натиснено правою кнопкою миші на проєкті та вибрано пункт меню «Manage NuGet Packages» (Див. Рис. 3.9).

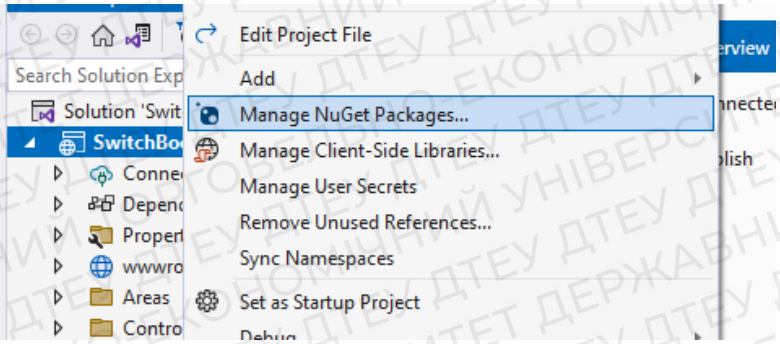

Рисунок 3.9. Пункт меню «Manage NuGet Packages»

Джерело: побудовано автором (знімок екрана)

2. У вкладці "Browse" знайдено та встановлено необхідні NuGet пакети (зображено на Рис. 3.10).

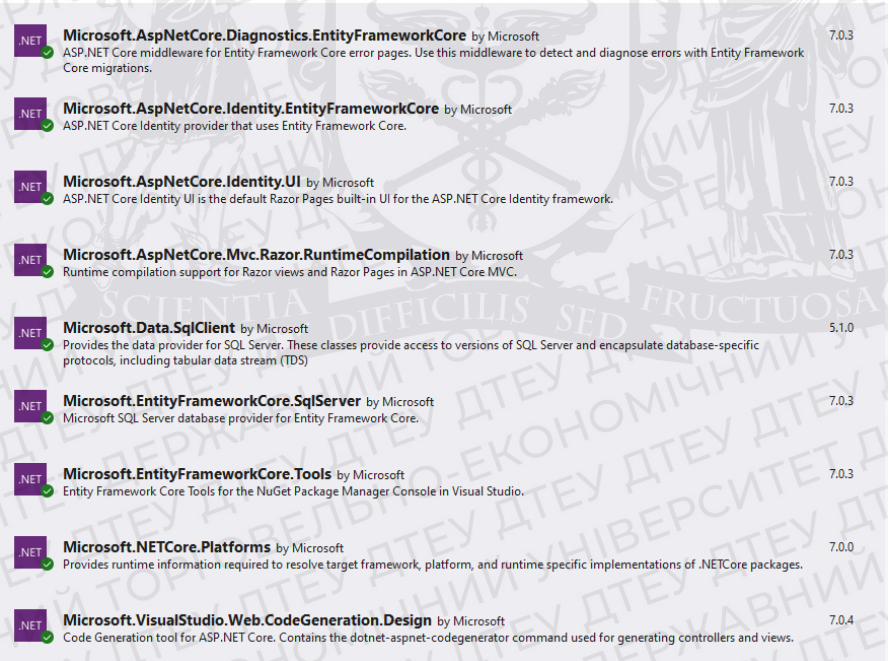

Рисунок 3.10. Необхідні NuGet пакети для розробки вебсайту для обміну книг.

Джерело: побудовано автором (знімок екрана)

Після встановлення пакета він буде доданий до проєкту, після цього є змога його використовувати.

3.2.3. Підключення бази даних та налаштування Identity Framework

Щоб під'єднати базу даних SQL Server Express до проєкту з використанням Entity Framework в Visual Studio, були зроблені наступні кроки:

1. Доданий рядок з'єднання до конфігураційного файлу, що знаходиться в корені проєкту. Рядку з'єднання була присвоєна назва «DefaultConnection» (Див. Рис. 3.11).

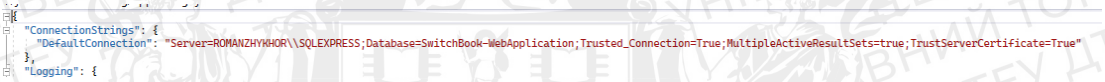

Рисунок 3.11. Рядок з'єднання з базою даних.

Джерело: побудовано автором (знімок екрана)

2. Створений файл контексту даних у теці «Data» (Див. Рис. 3.12).

```
public class ApplicationDbContext : IdentityDbContext
{
    0 references
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
}
```

Рисунок 3.12. Код контексту даних.

Джерело: побудовано автором (знімок екрана)

3. Були додані відповідні сервіси до проєкту: сервіс Entity Framework та Identity Framework (у файл Program.cs) (Див. Рис. 3.13).

```
// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString( name: "DefaultConnection");
builder.Services.AddDbContext<ApplicationDbContext>( optionsAction: options =>
    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddDefaultIdentity<IdentityUser>( configureOptions: options => options.SignIn.RequireConfirmedAccount = true)
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

Рисунок 3.13. Налаштування сервісів Entity та Identity.

Джерело: побудовано автором (знімок екрана)

3.3. Створення моделей, бази даних і табличок

Для створення моделей даних в C# було використано підхід Code First під час розробки з використанням Entity Framework.

Щоб створити моделі даних за допомогою Code First було зроблено наступні кроки:

1. Описано класи в програмі, які відповідають таблицям у базі даних. Кожен клас відображає структуру таблиці, тобто містить властивості, які відповідають стовпцям таблиці (Див. Рис. 3.14).

```
2 references
public class Address
{
    4 references
    public int Id { get; set; }
    3 references
    public string Region { get; set; }
    3 references
    public string City { get; set; }
    3 references
    public string Street { get; set; }
    3 references
    public string PostalCode { get; set; }
    3 references
    public string PhoneNumber { get; set; }

    5 references
    public string UserId { get; set; }
    0 references
    public User User { get; set; }
}
```

Рисунок 3.14. Клас Address. Приклад класу, в якому встановлено Foreign Key.

Джерело: побудовано автором (знімок екрана)

2. Також на цьому етапі було створено клас User, що наслідує клас IdentityUser, таким чином були визначені додаткові поля у базі даних користувача (Див. Рис. 3.15).

```
44 references
public class User : IdentityUser
{
    1 reference
    public byte[] Avatar { get; set; }
    0 references
    public ICollection<Book> Books { get; set; }
}
```

Рисунок 3.15. Клас User

Джерело: побудовано автором (знімок екрана)

3. Після того, як всі класи-моделі були створені в контексті бази даних визначено DbSet для кожного класу-моделі, щоб відобразити таблиці бази даних.

4. У контексті бази даних було змінено клас, який наслідує клас контексту даних (IdentityDbContext) на IdentityDbContext<User>, щоб саме цей клас використовувався для контексту користувача в базі даних (Див. Рис. 3.16).

```
20 references
public class ApplicationDbContext : IdentityDbContext<User>
{
    0 references
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    31 references
    public DbSet<Book> Books { get; set; }
    7 references
    public DbSet<Address> Address { get; set; }
    11 references
    public DbSet<Order> Orders { get; set; }
}
```

Рисунок 3.16. Клас контексту даних після зроблених кроків.

Джерело: побудовано автором (знімок екрана)

3.4. Створення контролерів та логіки клієнт-серверного додатка

3.4.1. Створення контролерів

Щоб створити контролер та під'єднати контекст бази даних до нього, необхідно виконати наступні кроки:

1. У теці "Controllers" натисніть правою кнопкою миші та виберіть опцію "Add" → "Controller" (Див. Рис. 3.17).

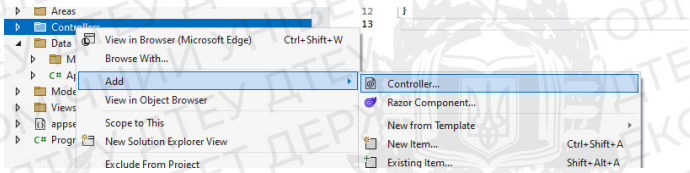

Рисунок 3.17. Додавання контролера через контекстне меню

Джерело: побудовано автором (знімок екрана)

2. У вікні що відкриється вибрано опцію "MVC Controller - Empty", а потім натиснуто кнопку "Add".

3. Вибрано ім'я контролера та натиснуто кнопку "Add" (Див. Рис. 3.18).

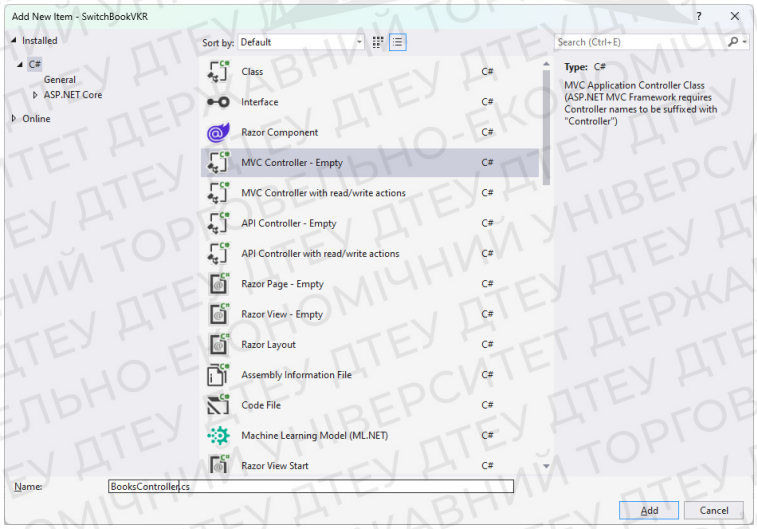

Рисунок 3.18. Клас контексту даних після зроблених кроків.

Джерело: побудовано автором (знімок екрана)

4. Після цього було створено контролер.

3.4.2. Додавання контексту даних та процес розробки логіки

Щоб додати контекст даних для проведення операцій з даними у таблицях бази даних було створено конструктор, та ініційовано з параметрів що передаються в нього клас контексту даних, а також приватну змінну, що ініціалізується в конструкторі (Див. Рис. 3.19).

```
1 reference
public class BooksController : Controller
{
    private readonly ApplicationDbContext _context;

    0 references
    public BooksController(ApplicationDbContext context)
    {
        _context = context;
    }
```

Рисунок 3.19. Конструктор та змінна у конструкторі контролера «BooksController».

Джерело: побудовано автором (знімок екрана)

Після цього було створено методи, що обробляють Get, Post, Put або Delete HTTP запити. Приклад одного з таких методів зображено на рисунку 3.19. Метод передає певну модель даних, що потім використовується для маніпуляцій з даними у таблицях бази даних (Див. Рис. 3.20).

```
[AllowAnonymous]
3 references
public async Task<IActionResult> Index(string query)
{
    var books = await _context.Books.Where(x => 
        EF.Functions.Like(matchExpression: x.Title, pattern: $"%{query}%") || EF.Functions.Like(matchExpression: x.Author, pattern: $"%{query}%") ||
        EF.Functions.Like(matchExpression: x.Description, pattern: $"%{query}%")).ToListAsync();

    var orders = await _context.Orders.ToListAsync();
    foreach (var order in orders)
    {
        var mybook = await _context.Books.FirstOrDefaultAsync(x => x.Id == order.FirstBookId);
            books.Remove(mybook);
            mybook = await _context.Books.FirstOrDefaultAsync(x => x.Id == order.LastBookId);
            books.Remove(mybook);
    }

    return View(books);
}
```

Рисунок 3.20. Метод що опрацьовує Get запит (завантажує сторінку).

Джерело: побудовано автором (знімок екрана)

3.5. Верстка сторінки Razor pages та додавання стилів

3.5.1. Верстка сторінки

Razor Pages - це шаблон для створення вебсторінок в Asp.Net Core, який дозволяє розподілити логіку, яка обробляє HTTP-запити, та розмітку HTML на окремі файли. За допомогою Razor Pages можна створити вебсторінки, які динамічно змінюються залежно від запитів користувачів, використовуючи вбудовану модель взаємодії з базою даних, та інші можливості Asp.Net Core.

В Asp.Net можна розробляти Single Page Applications (SPA), що дозволяє створювати вебдодатки з більшою плавністю та інтерактивністю, відповідно зменшити час на завантаження сторінки, покращити користувацький досвід і зробити сайт більш зручним для використання. Крім того, SPA дозволяє зменшити навантаження на сервер, оскільки не потрібно кожен раз завантажувати всі елементи сторінки з сервера.

Для створення представлення були зроблені наступні кроки:

1. Створено теку в теці «Views», що називається так, як і контролер (контролер має назву «BooksController», тому тека названа «Books»).

2. Усередину додано файли представлення, що називаються відповідно до методів у контролері (метод «Index», назва файлу «Index.cshtml») (Див. Рис. 3.21).

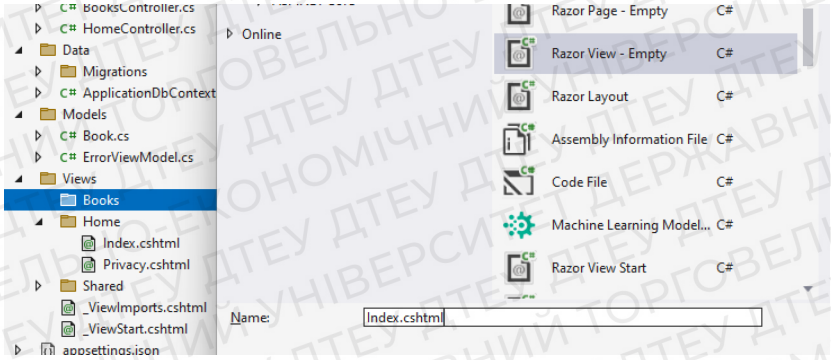

Рисунок 3.21. Створення файлу представлення.

Джерело: побудовано автором (знімок екрана)

3. Усередину був записаний код, необхідний для коректного відображення даних (Див. Рис. 3.22).

```
@model IEnumerable<SwitchBook.Models.Book>

@{
    ViewData["Title"] = "Пошук книги";
}

<h1>Доступні книги</h1>

<p>
    <a asp-action="Create" class="btn btn_book">Додати книгу</a>
</p>
<form asp-action="Index" class="book_search">
    <input type="text" name="query"/>
    <input type="submit" value="Пошук"/>
</form>

<div class="avaliable_books">
    @foreach (var item in Model)
    {
        <a class="avaliable_book" asp-action="Details" asp-route-id="@item.Id">
            @if (item.Image != null)
            {
                <img class="avaliable_book_image" src="@(Convert.ToBase64String(item.Image))"/>
            }
            else
            {
                <div class="avaliable_book_image"></div>
            }
            <div class="avaliable_book_text">
                <h4>@Html.DisplayFor(expression: modelItem => item.Title)</h4>
                <hr/>
                <h5>@Html.DisplayFor(expression: modelItem => item.Author)</h5>
                <h6 maxlength="1">@Html.DisplayFor(expression: modelItem => item.Description)</h6>
            </div>
        </a>
    }
</div>
```

Рисунок 3.22. Приклад коду (файл представлення Index контролера Books).

Джерело: побудовано автором (знімок екрана)

3.5.2. Створення та застосування стилів

CSS (Cascading Style Sheets) - це мова стилів, що використовується для опису зовнішнього вигляду вебсторінок, наприклад, кольорів, шрифтів, розмірів, міжрядкових інтервалів тощо. Використання CSS дозволяє розділити оформлення вебсторінки від її структури, що полегшує редагування та підтримку коду. Крім того, CSS дозволяє зменшити розмір HTML-коду вебсторінки, що поліпшує її швидкодію та зручність використання.

CSS є важливою складовою будь-якого сучасного вебсайту. CSS використовується для візуального оформлення і форматування сторінок вебсайту, включаючи кольори, шрифти, розміри, макети, анімації та інші

аспекти. Без CSS вебсайти будуть виглядати дуже просто і нецікаво для користувачів.

Стилі CSS записуються у файли стилів, що знаходиться відносно проєкту у теці wwwroot/CSS.

Для створення файлу стилю використовується контекстне меню.

Вебсайт SwitchBook було стилізовано за допомогою написаних файлів стилю, та фреймворку Bootstrap 5 (Див. Рис. 3.23).

```
1  .home_page...
9
10 .text...
15
16 .home_text_title {
17     font-family: Montserrat;
18     font-style: normal;
19     font-weight: 600;
20     font-size: 40px;
21     line-height: 59px;
22     color: #4C4333;
23 }
24
25 .home_text_filling...
34
35
36 @media only screen and (max-width: 1248px) {
37     .text {
38         width: 100%;
39     }
40 }
41
42 @media only screen and (max-width: 725px)...
```

Рисунок 3.23. Приклад CSS коду з використанням медіазапитів.

Джерело: побудовано автором (знімок екрана)

3.6. Результат роботи

В цьому проєкті було розроблено вебдодаток, який дозволяє користувачам обмінюватися книгами. Додаток був розроблений на платформі ASP.NET з використанням мови програмування C# та бази даних SQL Server.

Основний функціонал, що було розроблено для додатка, включає наступне:

1. Реєстрація та авторизація користувачів.

Користувачі можуть створити обліковий запис та авторизуватися в системі. Після авторизації користувачі можуть здійснювати дії, які є

доступними тільки зареєстрованим користувачам. На рисунку Рис. 3.24. зображена сторінка реєстрації користувача.

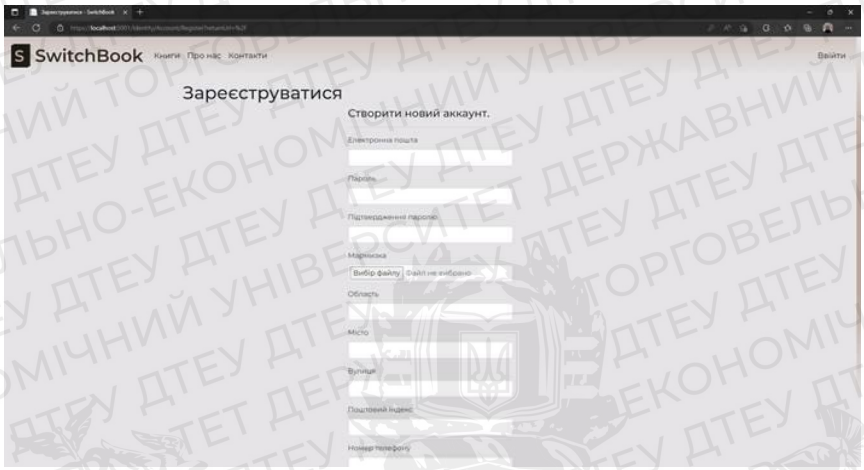

Рисунок 3.24. Сторінка реєстрації користувача.

Джерело: побудовано автором (знімок екрана)

2. Додавання книг до бібліотеки.

Користувачі можуть додавати нові книги до бібліотеки, вказуючи назву, автора, відгук та картинку. Книги зберігаються в базі даних (Див. Рис. 3.25)..

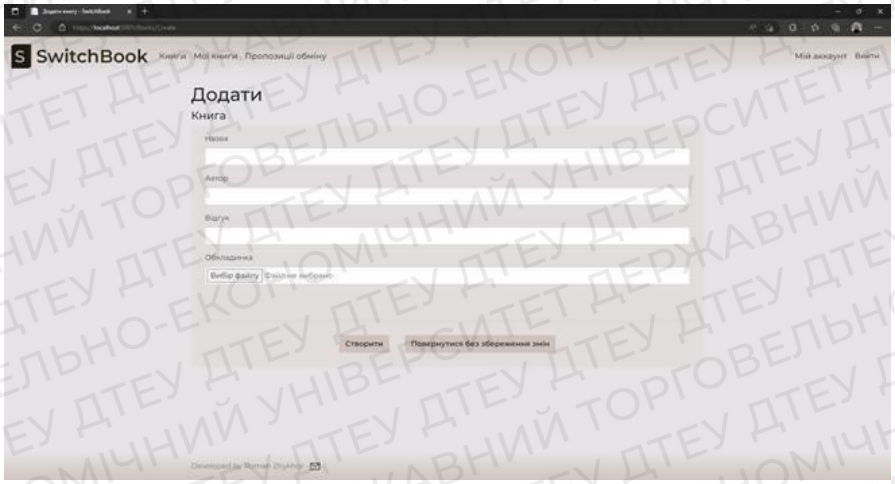

Рисунок 3.25. Сторінка додавання книги до своєї бібліотеки.

Джерело: побудовано автором (знімок екрана)

3. Пошук книг.

Користувачі можуть шукати книги в бібліотеці за різними критеріями, такими як назва, автор та відгук. На рисунку Рис. 3.26. зображена сторінка всіх доступних до обміну книг.

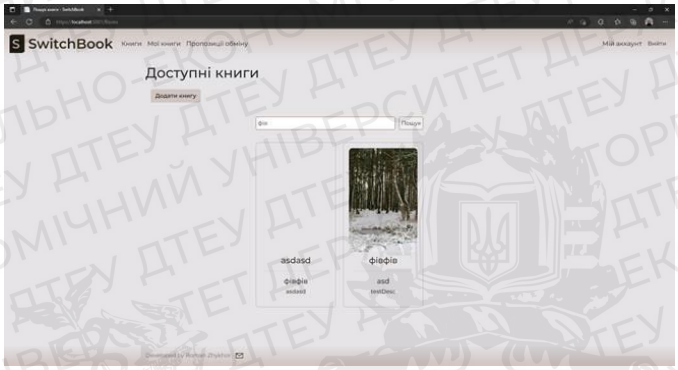

Рисунок 3.26. Сторінка з доступними книгами за пошуковим запитом.

Джерело: побудовано автором (знімок екрана)

4. Запит на обмін книгами.

Користувачі можуть робити запити на обмін книгами з іншими користувачами, вибираючи книгу зі своєї бібліотеки та пропонуючи обміняти її на книгу з бібліотеки іншого користувача. Процес обміну книгами зображено на рисунку Рис. 3.27.

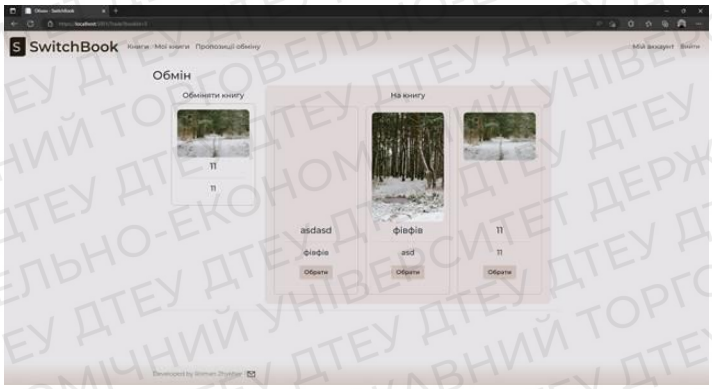

Рисунок 3.27. Створення запиту на обмін книгами.

Джерело: побудовано автором (знімок екрана)

5. Менеджмент обмінів.

Користувачі можуть переглядати запити на обмін книгами, які були створені ними або адресовані до них. Вони можуть приймати або відхиляти запити, та переглядати деталі обміну з іншими користувачами (Див. Рис. 3.28).

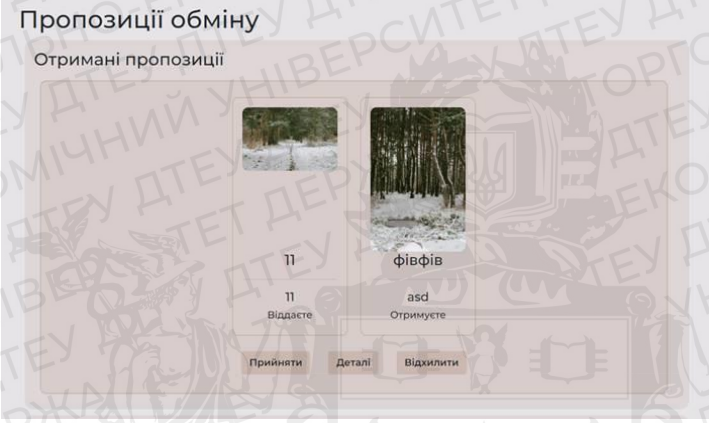

Рисунок 3.28. Сторінка менеджменту пропозицій обміну.

Джерело: побудовано автором (знімок екрана)

6. Профіль користувача.

Користувачі можуть переглядати та змінювати свій профіль, додавати фотографії, опис про себе та інші деталі (Див. Рис. 3.29).

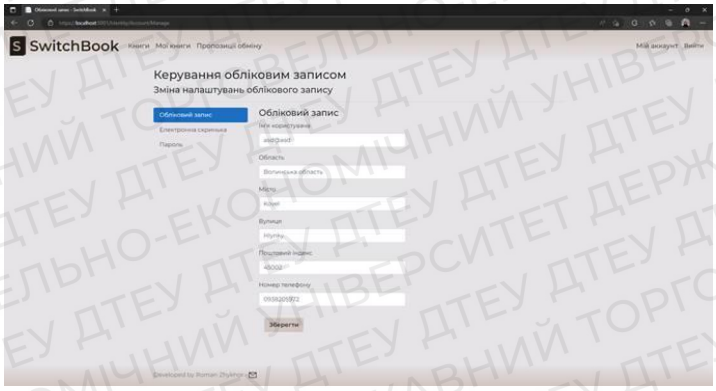

Рисунок 3.29. Сторінка редагування особистої сторінки.

Джерело: побудовано автором (знімок екрана)

7. Під час розробки вебсайту було використано підхід Responsive Web Design, який дозволяє сайту адаптуватись до різних розмірів екранів пристроїв, включаючи настільні комп'ютери, планшети та мобільні пристрої. Для досягнення цієї мети було використано медіазапити та різні техніки CSS, такі як пропорційність, відносні одиниці, флексбокси та сітки (Див. Рис. 3.30).

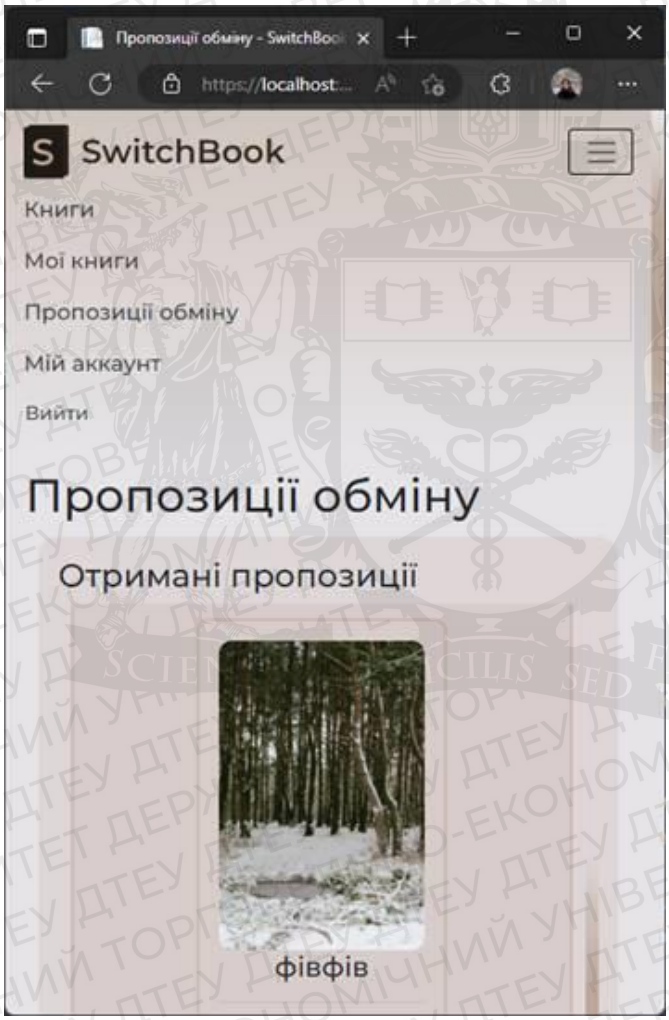

Рисунок 3.30. Сторінка пропозицій обміну оптимізована під мобільні пристрої.

Джерело: побудовано автором (знімок екрана)

3.7. Висновок до розділу 3

Вебсайт буккросингу було створено завдяки наявній архітектурі, що була попередньо створена. Описаний алгоритм дії веб застосунку дозволив відносно швидко та без додаткових поправок розробити клієнт серверний додаток, налаштувати та створити базу даних, створити у ній таблички за допомогою Entity Framework. Розробити швидкий алгоритм обробки даних, адже дані в Asp.Net оброблюються швидше ніж в інших фреймворків, в основі яких мова програмування JavaScript.

Користувачі мають змогу зареєструватися, авторизуватися, додавати та видаляти книги, змінювати їх опис, обкладинку, надсилати та приймати запити на обмін книгами. Також користувачі можуть редагувати інформацію про себе, змінювати пароль.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

З проведених досліджень можна зробити наступні висновки:

1. Проведено аналіз предметної області клієнт-серверного додатка. В результаті аналізу визначено що читання книг популярний спосіб проведення відпочинку.

Також визначено мову програмування та сервер бази даних. Для кращої реалізації та швидкодії майбутнього клієнт-серверного застосунку.

2. В якості платформи для роботи з клієнт-серверним додатком було обрано вебсайт, а значить користувачі сервісу можуть використовувати його на будь-якій операційній системі де встановлено браузер.

3. В процесі виконання проєкту створено архітектуру (use case, класів, еталонну модель, блок схему, діаграму послідовності, діаграму стану, діаграму розгортання та компонентів), логічну та фізичну модель бази даних, дизайн вебсайту та розроблено веб додаток для буккросингу «SwitchBook» з оптимізацією сторінок для різної ширини екрана, таким чином користуватись сайтом зручно з різних гаджетів.

4. Розроблена система повністю задовольняє вимоги, що були поставлені перед розробкою всієї системи.

Таким чином була створена система для обміну книгами.

Пропозиції для програми: розробити особисті чати для обміну повідомлень всередині системи, розробити систему друзів, та винагород за кількість обмінених книг. Розробка системи верифікації користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вебсайт Bootstrap [Електронний ресурс]. – Режим доступу : https://getbootstrap.com/. Дата звернення: 21.02.2023 р.

2. Вебсайт Microsoft «ASP.NET» [Електронний ресурс]. – Режим доступу: https://dotnet.microsoft.com/en-us/apps/aspnet. Дата звернення: 21.02.2023 р.

3. Вебсайт Codeproject «An Introduction to Entity Framework for Absolute Beginners» [Електронний ресурс]. Режим доступу: https://www.codeproject.com/Articles/363040/An-Introduction-to-Entity-Framework-for-Absolute-B Дата звернення: 21.02.2023 р.

4. Вебсайт JetBrains «Basics of Razor Pages. A complete Razor Pages pipeline.» [Електронний ресурс]. – Режим доступу: https://www.jetbrains.com/dotnet/guide/tutorials/basics/razor-pages/. Дата звернення: 21.02.2023 р.

5. Вебсайт thgmwriters «Global book reading statistics for 2022 and 2023 (complete survey data)» [Електронний ресурс]. – Режим доступу : https://thgmwriters.com/blog/global-book-reading-statistics-2022-2023-complete-survey-data/. Дата звернення: 21.02.2023 р.

6. Вебсайт ClickIt «Web Application Architecture: The Latest Guide 2022» [Електронний ресурс]. – Режим доступу : https://www.clickittech.com/devops/web-application-architecture/. Дата звернення: 21.02.2023 р.

7. Вебсайт TechEmpower «Web Framework Benchmarks» [Електронний ресурс]. – Режим доступу : https://www.techempower.com/benchmarks/#section=data-r21&hw=ph&test=plaintext. Дата звернення: 21.02.2023 р.

8. Вебсайт MDN Web Docs [Електронний ресурс]. – Режим доступу : https://developer.mozilla.org/. Дата звернення: 21.02.2023 р.

9. Вебсайт Microsoft «C# programming guide» [Електронний ресурс]. – Режим доступу : https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/. Дата звернення: 21.02.2023 р.

10. Вебсайт Microsoft SQL Server Documentation [Електронний ресурс]. – Режим доступу : https://docs.microsoft.com/en-us/SQL/SQL-server/. Дата звернення: 21.02.2023 р.

11. Вебсайт Diagrams [Електронний ресурс]. – Режим доступу : https://app.diagrams.net/. Дата звернення: 21.02.2023 р.

12. Вебсайт Figma [Електронний ресурс]. – Режим доступу : https://www.figma.com/ .Дата звернення 22.02.2023р.

ДОДАТКИ

ДОДАТОК А

Код класу BooksController

```
public class BooksController : Controller
{
    private readonly ApplicationDbContext _context;

    public BooksController(ApplicationDbContext context)
    {
        _context = context;
    }

    [AllowAnonymous]
    public async Task<IActionResult> Index(string query)
    {
        var books = await _context.Books.Where(x =>
            EF.Functions.Like(x.Title, $"%{query}%") || EF.Functions.Like(x.Author,
$"%{query}%") ||
            EF.Functions.Like(x.Description, $"%{query}%")).ToListAsync();

        var orders = await _context.Orders.ToListAsync();
        foreach (var order in orders)
        {
            var mybook = await _context.Books.FirstOrDefaultAsync(x => x.Id ==
order.FirstBookId);
            books.Remove(mybook);
            mybook = await _context.Books.FirstOrDefaultAsync(x => x.Id ==
order.LastBookId);
            books.Remove(mybook);
        }

        return View(books);
    }

    public async Task<IActionResult> Details(int? id)
    {
        if (id == null) return NotFound();

        var book = await _context.Books
            .FirstOrDefaultAsync(m => m.Id == id);
        var owner = await _context.Users.FirstOrDefaultAsync(x => x.Id ==
book.OwnerId);
        ViewBag.Owner = owner.UserName;
        if (book == null) return NotFound();

        return View(book);
    }

    public IActionResult Create()
    {
        return View();
    }

    [HttpPost]
    [Authorize]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create(BookViewModel bookvm)
    {
        if (ModelState.IsValid)
        {
            var book = new Book
```

```
            {
                Title = bookvm.Title,
                Author = bookvm.Author,
                Description = bookvm.Description,
                OwnerId = _context.Users.First(x => x.UserName ==
User.Identity.Name).Id
            };
            if (bookvm.Image != null)
            {
                byte[] imageData = null;
                using (var binaryReader = new
BinaryReader(bookvm.Image.OpenReadStream()))
                {
                    imageData = binaryReader.ReadBytes((int)bookvm.Image.Length);
                }

                book.Image = imageData;
            }

            _context.Add(book);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        return View();
    }

    public async Task<IActionResult> Edit(int? id)
    {
        if (id == null) return NotFound();

        var book = await _context.Books.FindAsync(id);
        if (book == null) return NotFound();
        return View(book);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(int id,
[Bind("Id,Title,Author,OwnerId,Description,Image")] Book book)
    {
        if (id != book.Id) return NotFound();

        if (ModelState.IsValid)
        {
            try
            {
                _context.Update(book);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!BookExists(book.Id))
                    return NotFound();
                throw;
            }

            return RedirectToAction(nameof(Index));
        }

        return View(book);
    }

    public async Task<IActionResult> Delete(int? id)
{
```

```
    if (id == null) return NotFound();

    var book = await _context.Books
        .FirstOrDefaultAsync(m => m.Id == id);
    if (book == null) return NotFound();

    return View(book);
}

[HttpPost]
[ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var book = await _context.Books.FindAsync(id);
    _context.Books.Remove(book);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

private bool BookExists(int id)
{
    return _context.Books.Any(e => e.Id == id);
}
  }
```

Код класу MyBooksController

```
public class MyBooksController : Controller
{
    private readonly ApplicationDbContext _context;

    public MyBooksController(ApplicationDbContext context)
    {
        _context = context;
    }

    public async Task<IActionResult> Index()
    {
        var ownerId = await _context.Users.FirstAsync(x => x.UserName ==
User.Identity.Name);
        var mybooks = await _context.Books.Where(x => x.OwnerId ==
ownerId.Id).ToListAsync();

        var orders = await _context.Orders.ToListAsync();
        foreach (var order in orders)
        {
            var mybook = await _context.Books.FirstOrDefaultAsync(x => x.Id ==
order.FirstBookId);
            mybooks.Remove(mybook);
            mybook = await _context.Books.FirstOrDefaultAsync(x => x.Id ==
order.LastBookId);
            mybooks.Remove(mybook);
        }

        return View(mybooks);
    }

    public async Task<IActionResult> Edit(int? id)
    {
        if (id == null) return NotFound();

        var book = await _context.Books.FindAsync(id);
        if (book == null) return NotFound();
        return View(book);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(int id,
[Bind("Id,Title,Author,OwnerId,Description, Image")] Book book,
        IFormFile ImageEdit)
    {
        if (id != book.Id) return NotFound();

        if (!ModelState.IsValid) return View(book);

        try
        {
            if (ImageEdit != null)
            {
                byte[] imageData = null;
                using (var binaryReader = new
BinaryReader(ImageEdit.OpenReadStream()))
                {
                    imageData = binaryReader.ReadBytes((int)ImageEdit.Length);
                }

                book.Image = imageData;
```

```
            }
            else
            {
                var originalBook = await
_context.Books.AsNoTracking().FirstOrDefaultAsync(x => x.Id == book.Id);

                book.Image = originalBook.Image;
            }

            _context.Update(book);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!BookExists(book.Id))
                return NotFound();
            throw;
        }

        return RedirectToAction(nameof(Index));
    }

    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null) return NotFound();

        var book = await _context.Books
            .FirstOrDefaultAsync(m => m.Id == id);
        if (book == null) return NotFound();

        return View(book);
    }

    [HttpPost]
    [ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var book = await _context.Books.FindAsync(id);
        _context.Books.Remove(book);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }

    private bool BookExists(int id)
    {
        return _context.Books.Any(e => e.Id == id);
    }
  }
```

Код класу MyOrdersController

```
public class MyOrdersController : Controller
{
    private readonly ApplicationDbContext _db;

    public MyOrdersController(ApplicationDbContext context)
    {
        _db = context;
    }

    public async Task<IActionResult> Index()
    {
        var myBooks = _db.Books.Where(x => x.OwnerId == _db.Users.First(p =>
p.UserName == User.Identity.Name).Id);
        var orderRequest = await _db.Orders
            .Where(x => myBooks.Select(b => b.Id).Contains(x.FirstBookId) &&
x.IsConfirm == false).ToListAsync();
        var b1 = new List<Book>();
        var b2 = new List<Book>();
        foreach (var order in orderRequest)
        {
            b1.Add(await _db.Books.FirstOrDefaultAsync(x => x.Id ==
order.FirstBookId));
            b2.Add(await _db.Books.FirstOrDefaultAsync(x => x.Id ==
order.LastBookId));
        }

        var orderMyRequest = await _db.Orders
            .Where(x => myBooks.Select(b => b.Id).Contains(x.LastBookId) &&
x.IsConfirm == false).ToListAsync();
        var b1M = new List<Book>();
        var b2M = new List<Book>();
        foreach (var order in orderMyRequest)
        {
            b1M.Add(await _db.Books.FirstOrDefaultAsync(x => x.Id ==
order.FirstBookId));
            b2M.Add(await _db.Books.FirstOrDefaultAsync(x => x.Id ==
order.LastBookId));
        }

        var orderHistory = await _db.Orders.Where(x =>
            (myBooks.Select(b => b.Id).Contains(x.FirstBookId) || myBooks.Select(b
=> b.Id).Contains(x.LastBookId)) &&
            x.IsConfirm == true).ToListAsync();
        var b1H = new List<Book>();
        var b2H = new List<Book>();

        foreach (var order in orderHistory)
        {
            b1H.Add(await _db.Books.FirstOrDefaultAsync(x => x.Id ==
order.FirstBookId));
            b2H.Add(await _db.Books.FirstOrDefaultAsync(x => x.Id ==
order.LastBookId));
        }

        var viewModel = new MyOrdersViewModel
        {
            Requests = new OrderInfo { Books1 = b1, Books2 = b2, Orders =
orderRequest },
            History = new OrderInfo { Orders = orderHistory, Books1 = b1H, Books2 =
b2H },
```

```
MyRequests = new OrderInfo { Orders = orderMyRequest, Books1 = b1M, Books2 = b2M }
        };


        return View(viewModel);
    }

    [HttpGet]
    public async Task<IActionResult> OrderDetails(int OrderId)
    {
        var order = await _db.Orders.FirstOrDefaultAsync(x => x.Id == OrderId);
        var book1 = await _db.Books.FirstOrDefaultAsync(x => x.Id ==
order.FirstBookId);
        var book2 = await _db.Books.FirstOrDefaultAsync(x => x.Id ==
order.LastBookId);
        if (order == null)
            return NotFound();
        ViewBag.OrderId = order.Id;
        ViewBag.Book1 = book1;
        ViewBag.Book2 = book2;
        ViewBag.Address1 = await _db.Address.FirstOrDefaultAsync(x => x.Id ==
order.FirstAddressId);
        ViewBag.Address2 = await _db.Address.FirstOrDefaultAsync(x => x.Id ==
order.LastAddressId);
        ViewBag.Order = order;
        ViewBag.Owner1 = await _db.Users.FirstOrDefaultAsync(x => x.Id ==
book1.OwnerId);
        ViewBag.Owner2 = await _db.Users.FirstOrDefaultAsync(x => x.Id ==
book2.OwnerId);
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> ConfirmRequest(int OrderId)
    {
        var order = await _db.Orders.FirstOrDefaultAsync(x => x.Id == OrderId);
        if (order == null)
            return NotFound();
        order.IsConfirm = true;
        _db.Orders.Update(order);
        await _db.SaveChangesAsync();
        return RedirectToAction("Index");
    }


    [HttpPost]
    public async Task<IActionResult> DeleteRequest(int OrderId)
    {
        var order = await _db.Orders.FirstOrDefaultAsync(x => x.Id == OrderId);
        if (order == null)
            return NotFound();
        _db.Orders.Remove(order);
        await _db.SaveChangesAsync();
        return RedirectToAction("Index");
    }
  }
```

Код класу TradeController

```
public class TradeController : Controller
{
    public ApplicationDbContext _db;

    public TradeController(ApplicationDbContext context)
    {
        _db = context;
    }

    public async Task<ActionResult> Index(int bookId)
    {
        if (ModelState.IsValid)
        {
            ViewBag.bookId = bookId;

            var owner = await _db.Users.FirstAsync(x => x.UserName ==
User.Identity.Name);

            var myBooks = await _db.Books.Where(x => x.OwnerId ==
owner.Id).ToListAsync();
            var book = await _db.Books.FirstOrDefaultAsync(x => x.Id == bookId);
            var model = new TradeViewModel { Book = book, MyBooks = myBooks, Owner =
owner };
            return View(model);
        }

        return Redirect("/Home");
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<ActionResult> Create(int bookId, int myBookId)
    {
        if (!ModelState.IsValid) return Redirect("/Home");

        if (bookId == myBookId) return RedirectToAction("Index", "Books");

        var firstBookOwnerId = await _db.Books.FirstOrDefaultAsync(x => x.Id ==
bookId);

        var secondBookOwnerId = await _db.Books.FirstOrDefaultAsync(x => x.Id ==
myBookId);

        var firstAddress = await _db.Address.FirstOrDefaultAsync(x => x.UserId ==
firstBookOwnerId.OwnerId);

        var secondAddress = await _db.Address.FirstOrDefaultAsync(x => x.UserId ==
secondBookOwnerId.OwnerId);

        var newOrder = new Order
        {
            FirstBookId = bookId,
            LastBookId = myBookId,

            FirstAddressId = firstAddress.Id,

            LastAddressId = secondAddress.Id,

            IsConfirm = false
        };
        await _db.Orders.AddAsync(newOrder);
```

```
        await _db.SaveChangesAsync();
            return Redirect("/Books");
        }
    }
```