ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра інженерії програмного забезпечення

Пояснювальна записка
до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему**: «Розробка сервісу White Elephant для обміну непотрібними подарунками мовою C#»**

Виконав: студент 4 курсу, групи ПД–41
спеціальності
121 Інженерії програмного забезпечення
(шифр і назва спеціальності)
Конох В.В.
(прізвище та ініціали)
Керівник Золотухіна О.А.
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти - «Бакалавр»

Спеціальність - 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

Негоденко О.В.

« » 2023 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Конох Владислав Віталійович

(прізвище, ім‘я, по батькові)

1. Тема роботи: «Розробка сервісу White Elephant для обміну непотрібними подарунками мовою C#»

Керівник роботи Золотухіна О.А., к.т.н., доц., доцент кафедри ІПЗ

(прізвище, ім‘я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «24»лютого 2023 року № 26.

2. Строк подання студентом роботи «1» червня 2023 року

3. Вхідні дані до роботи:
   3.1 Офіційна документація Microsoft
   3.2 Науково-технічна література
   3.3 Норми та постанови у сфері благодійності та обміну подарунками

4. Зміст розрахунково-пояснювальної записки
   4.1 Огляд існуючих веб-сервісів для обміну непотрібними подарунками та їх аналіз
   4.2 Проектування та розробка
   4.3 Реалізація функціональності
   4.4 Тестування додатку
   4.5 Висновки
5. Перелік демонстраційного матеріалу
   5.1 Мета, об’єкт та предмет дослідження
   5.2 Задачі дипломної роботи
   5.3 Аналіз аналогів

5.4 Вимоги до програмного забезпечення
5.5 Програмні засоби реалізації
5.6 Діаграма варіантів використання
5.7 Діаграма класів
5.8 Схема бази данних
5.9 Єкранні форми

Дата видачі завдання «25» лютого 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	25.02.2023-28.02.2023	Виконано
2	Аналіз та дослідження існуючих аналогів		Виконано
3	Дослідження програмних засобів		Виконано
4	Моделювання об’єкту проєктування		Виконано
5	Розробка функціоналу додатка		Виконано
6	Вступ, висновки, реферат		Виконано
7	Розробка обов’язкових демонстраційних матеріалів		Виконано
8	Попередній захист роботи		Виконано
9	Здача роботи	01.06.2023	Виконано

Студент_____ Конох В.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Золотухіна О.А.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи: <> с., <> табл., <> рис., <> джерела

C#, ASP.NET, ENTITY FRAMEWORK, MSSQL, MVC, RAZOR PAGES, ASP.NET IDENTITY.

Об`єкт дослідження – процес обміну непотрібними речами між користувачами.

Предмет дослідження – програмне забезпечення для обміну непотрібними речами.

Мета роботи – спрощення обміну непотрібними речами за рахунок застосування веб-додатку, створеного мовою C#.

Методи дослідження – методи розробки та проєктування програмного забезпечення, методи обробки вхідних даних, методи тестування програмного забезпечення.

В роботі розглянуті та проаналізовані існуючі аналоги програмного забезпечення для обміну непотрібними речами. Досліджено методи створення сервісу для обміну непотрібними подарунками. Розроблено програмне рішення для обміну непотрібними подарунками.

Для розробки було обрано архітектуру MVC, мову програмування C#,фреймворк ASP.NET MVC який є нативним та простим інструментом для розробки веб-додатків, ORM Entity Framework для роботи з базою даних MSSQL.

Ключовими перевагами веб-сервісу для обміну непотрібними подарунками “White Elephant” перед існуючими конкурентами є можливість додати фото подарунка в оголошення та створення списку бажань

Галузь використання – благодійність та обмін унікальними речами.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AJAX – Asynchronous JavaScript and XML

API – Application Programming Interface

MVC – model-view-controller architecture

XML – Extensible Markup Language

ПЗ – програмне забезпечення

ВСТУП

Сучасне суспільство виявляє все більше зацікавленості у використанні онлайн-сервісів для забезпечення комунікації, розваг та спільного виконання завдань. Разом із зростанням популярності таких сервісів з'являється потреба в постійному розвитку нових та оригінальних інструментів для задоволення потреб користувачів.

Одною зі сфер, де виникає потреба у нових сервісах, є організація обміну непотрібними подарунками. Традиційний метод обміну подарунками зазвичай вимагає фізичної присутності учасників, організації складного розподілу та ведення обліку. Це може бути незручним та часоємним процесом, особливо коли учасники знаходяться на великій відстані або мають обмежений доступ до фізичних зустрічей.

У зв'язку з цим, розробка онлайн-сервісу для обміну непотрібними подарунками стає актуальною проблемою. Такий сервіс має потенціал спростити процес організації обміну подарунками, забезпечити зручний інтерфейс для учасників та автоматизувати ряд операцій, пов'язаних з розподілом та обліком подарунків.

Об`єкт дослідження – процес обміну непотрібними речами між користувачами.

Предмет дослідження – програмне забезпечення для обміну непотрібними речами.

Мета роботи – спрощення обміну непотрібними речами за рахунок застосування веб-додатку, створеного мовою C#.

Методи дослідження – методи розробки та проєктування програмного забезпечення, методи обробки вхідних даних, методи тестування програмного забезпечення.

Опираючись на поставлену мету, були визначені наступні задачі:

- аналіз обраної предметної області в сфері обміну непотрібними подарунками;
- порівняння існуючих аналогів програмного забезпечення, яке можна використовувати для обміну речами чи подарунками на комерційній та безоплатній основі;

- визначення та моделювання вимог до програмного забезпечення;
- розробити архітектуру веб-сервісу для обміну непотрібними подарунками;
- проєктування та розробка програмного забезпечення для обміну непотрібними подарунками;
- провести аналіз та роробити засоби безпеки та захисту інформації користувачів;
- тестування розробленого додатку.

1 ОГЛЯД ІСНУЮЧИХ ВЕБ-СЕРВІСІВ ДЛЯ ОБМІНУ НЕПОТРІБНИМИ ПОДАРУНКАМИ ТА ЇХ АНАЛІЗ

1.1 Поняття веб-сервісу для обміну непотрібними подарунками

Веб-сервіс для обміну подарунками – це онлайн-платформа, що дозволяє користувачам обмінюватися між собою різними речами, які вони більше не потребують. Це можуть бути різноманітні предмети, такі як одяг, електроніка, меблі, книги тощо. Веб-сервіс для обміну непотрібними подарунками зазвичай дозволяє користувачам створювати облікові записи, додавати предмети, які вони хотіли б обміняти, переглядати та обирати речі, які їм цікаві, а також здійснювати контакт з іншими користувачами щодо угоди з обміну. Веб-сервіс для обміну непотрібними подарунками може також використовуватись для досягнення цілей зменшення кількості відходів за рахунок повторного використання речей.

1.2 Аналіз існуючих веб-сервісів для обміну непотрібними подарунками

Аналіз існуючих веб-сервісів для обміну непотрібними подарунками є важливим етапом у розробці нового веб-сервісу. Це дає змогу зрозуміти, які функції і можливості вже реалізовані у інших сервісах, а також виявити їхні недоліки та проблеми, щоб уникнути їх у майбутньому. Найбільш відомими сервісами для обміну непотрібними подарунками є Giftster, Secret Santa Orginizer та Freecycle.

1.2.1 Сервіс Giftster

"Giftster" є безкоштовним веб-сервісом для обміну подарунками, який дозволяє користувачам створювати список бажань і ділитися ним зі своїми друзями та родиною. Крім того, він надає можливість створювати групові подарунки, де кожен учасник може додати до списку бажань свої побажання. Після того, як усі учасники

зробили свій вибір, "Giftster" випадковим чином присвоює кожному учаснику ім'я людини, якій вони мають зробити подарунок.

Переваги:

–безкоштовний сервіс з великою кількістю функцій;

–можливість створювати групові подарунки і ділитися списками бажань зі своїми друзями та родиною;

–простий та зрозумілий інтерфейс користувача.

Недоліки:

–інтерфейс може бути трохи застарілим порівняно з іншими сервісами;

–є деякі обмеження в налаштуваннях обміну подарунками, наприклад, неможливість створити більше одного списку бажань на користувача.

Приклади графічного інтерфейсу головної сторінки, сторінки користувача та вкладка для створення списку побажань Giftster зображені на рисунках 1.1, 1.2, 1.3.

Рисунок 1.1 – Головна сторінка Giftster

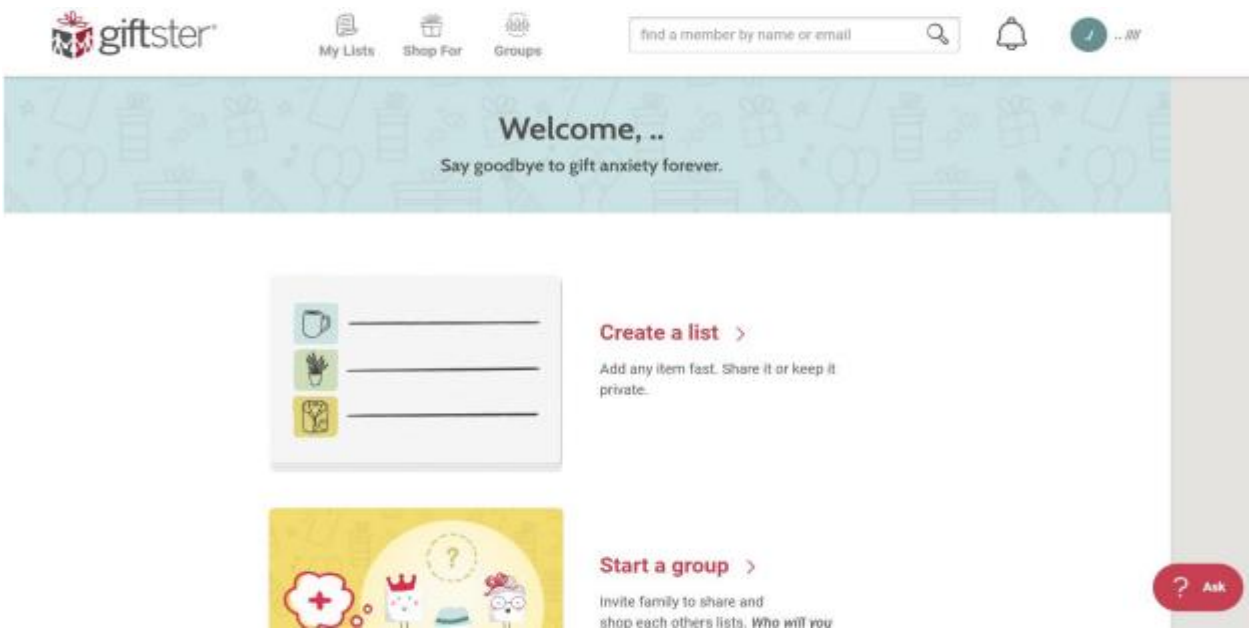


Рисунок 1.2 – Сторінка користувача Giftster

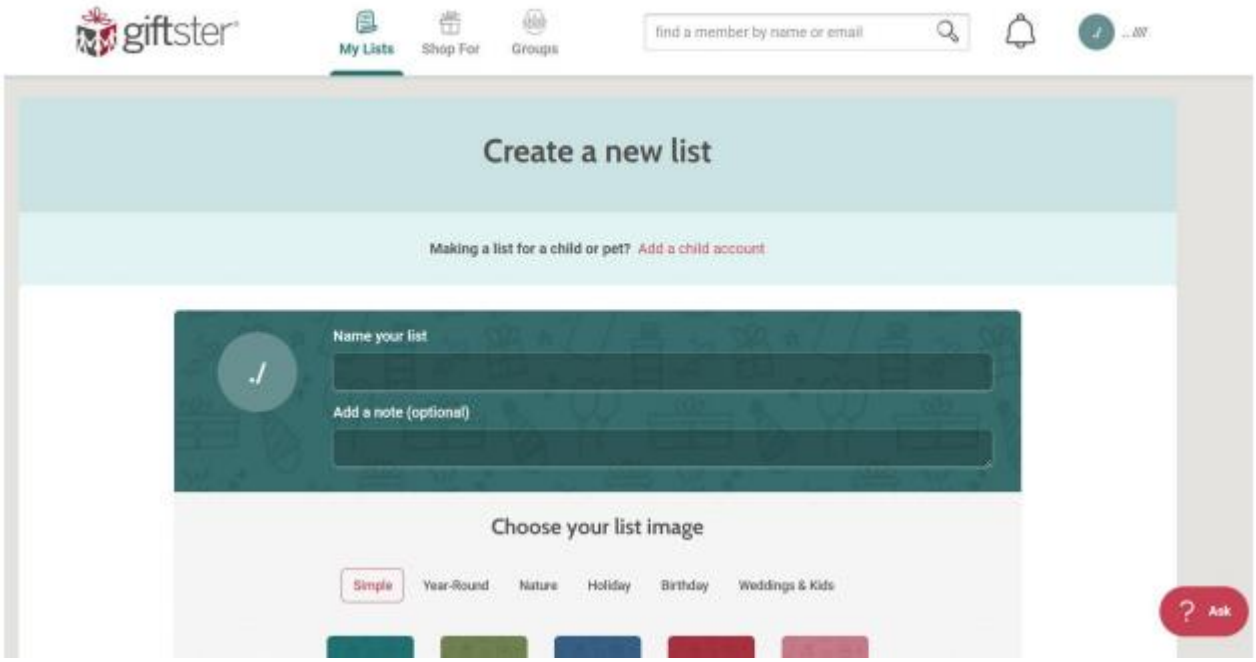


Рисунок 1.3 – Приклад створення списку подарунків Giftster

1.2.2 Сервіс Secret Santa Organizer

“Secret Santa Organizer” розроблений для організації секретного СантаКлауса серед різних груп, таких як родина, друзі, колеги тощо. Він надає можливість

користувачам зареєструватися і заповнити свій профіль з вказанням бажань, а також вибрати, з ким вони б хотіли обмінятися подарунками. Організатор групи може згенерувати звіти для кожного учасника з інформацією про те, кого обрати для подарунка.

Переваги:

–простий та зрозумілий інтерфейс;

–можливість організації груп серед користувачів з різних місць;

–вмонтований інструмент для генерації пар для обміну подарунками.

Недоліки:

–обмежені можливості налаштування прав доступу користувачів до груп;

–відсутність інструментів для створення списку бажань та нагадувань про терміни обміну подарунками.

Приклад графічного інтерфейсу для створення події Secret Santa Organizer зображені на рисунку 1.4.

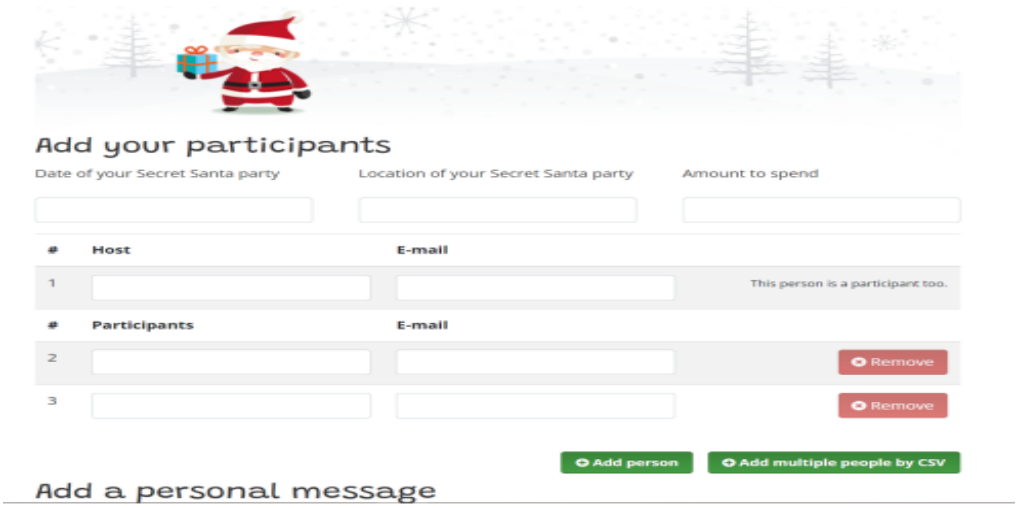


Рисунок 1.4 – Приклад створення події Secret Santa

1.2.3 Мережа Freecycle

Freecycle - це мережа локальних груп, які допомагають місцевим жителям обмінюватися непотрібними речами, замість того, щоб викидати їх у смітник. Сервіс

включає онлайн-форум, де користувачі можуть виставляти оголошення про предмети, які вони хочуть віддати або прийняти від інших людей.

Переваги:

–безкоштовний;

–захищає навколишнє середовище, допомагаючи уникнути викидування непотрібних речей;

–дозволяє знайти новий власник для предметів, які можуть бути корисні іншим людям.

Недоліки:

–обмін відбувається тільки в межах локальної групи;

–немає функції оплати за предмети, які ви хочете придбати;

–система безпеки та перевірки користувачів не така сувора, як у деяких інших сервісах.

Приклади графічного інтерфейсу списку речей на обмін Freecycle зображені на рисунку 1.5.

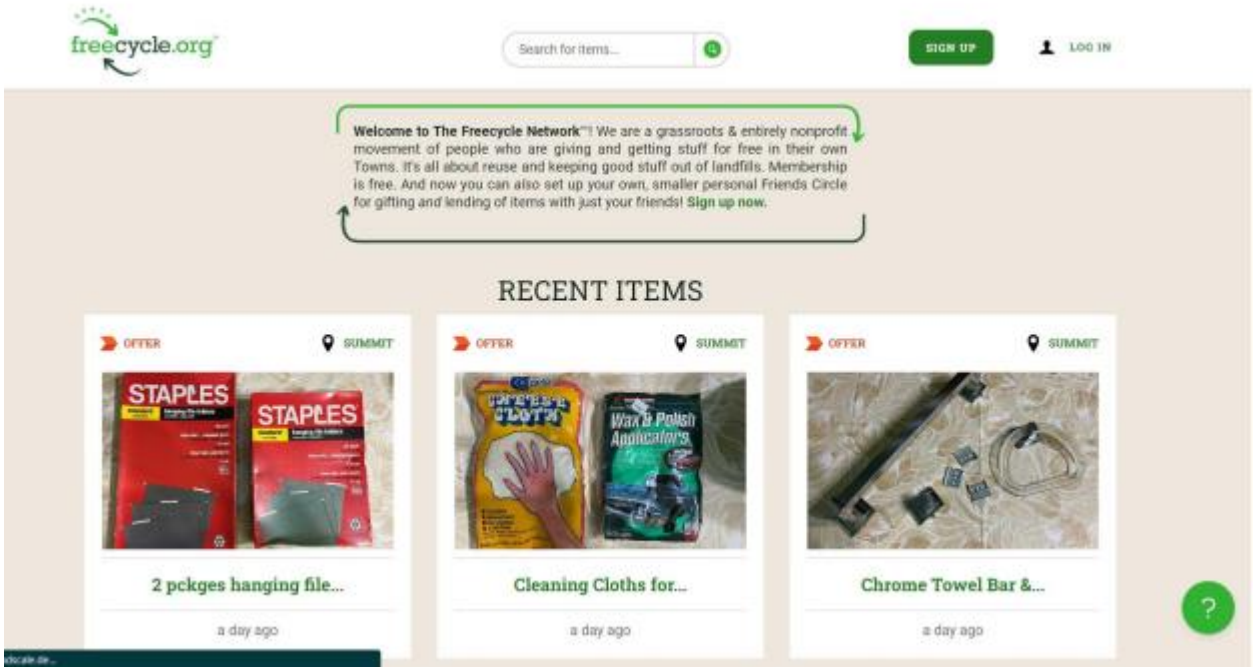


Рисунок 1.5 – Приклад головної сторінки FreeCycle

Зведені результати аналізу характеристик розглянутих додатків наведено у таблиці 1.1.

Таблиця 1.1 – Зведені результати аналізу характеристик додатків для обміну непотрібними подарунками.

Показник	**Giftster**	**Secret Santa**	**FreeCycle**
Платформи	Android, Web	IOS, Web	Web
Створення списку бажань	+	-	-
Обмін з сторонніми користувачами	+	+	+
Налаштування конфіденційності	+	-	-
Інтеграція з соц мережами	-	-	-
Групові обміни	+	+	-
Система відгуків та оцінок користувачів	-	+	-
Додавання фото товарів	+	-	-

Під час аналізу веб-сервісів також важливо враховувати вимоги до користувачів, такі як легкість використання, швидкість реакції та зручність навігації. Дизайн та інші фактори також можуть впливати на привабливість веб-сервісу та його успішність. Наприклад, простий та зрозумілий інтерфейс може забезпечити кращу взаємодію користувачів з веб-сервісом та збільшити його популярність.

2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ВЕБ-СЕРВІСУ

2.1 Моделювання вимог до програмного забезпечення

Діаграма варіантів використання, демонструє ключові взаємодії між акторами та системою та висвітлює основні функціональні можливості системи. Ця діаграма служить орієнтиром для розуміння загального потоку дій. Загальний вид діаграми варіантів використання сервісу для обміну непотрібними подарунками наведено на рисунку 2.1.

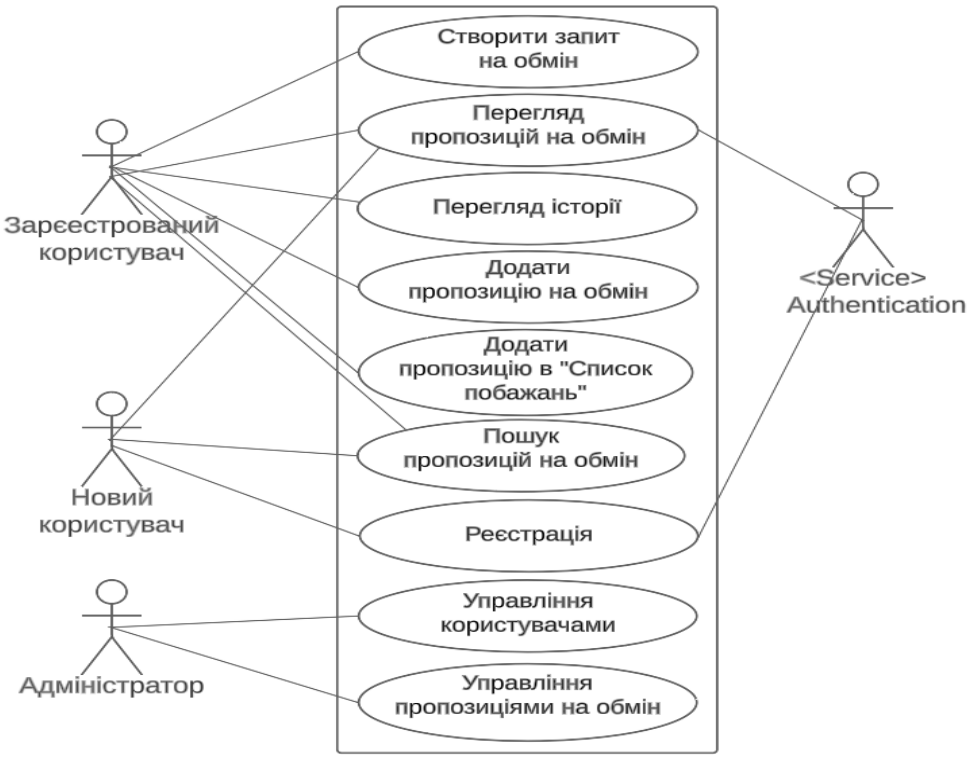


Рисунок 2.1 - Діаграма варіантів використання

Функціонал сервісу для обміну непотрібними подарунками відрізняється залежно того, зарєеструвався користувач чи ні. Новому користувачу доступно лише переглядати доступні для обміну подарунки та строка пошуку. Відповідно, основний функціонал, а саме - створення пропозиції/запиту на обмін, перегляд історії обмінів

та можливість додати пропозицію у список побажань, буде доступний користувачу тільки після рєестрації.

Також сервіс користувача з правами адміністратора якому доступно управління користувачами та управління пропозиціями на обмін.

2.2 Опис архітектури MVC

Для розробки сервісу для обміну непотрібними подарунками була обрана архітектура MVC. Архітектура MVC (Model-View-Controller) - це підхід до розробки програмного забезпечення, який розділяє програму на три складові: Модель, Вид та Контролер. Кожна складова виконує свої функції та має свою відповідальність, що дозволяє розділити логіку програми на окремі компоненти та робити її більш масштабованою, зручною для розробки та підтримки. Графічний приклад архітектури MVC зображено на рисунку 2.2.

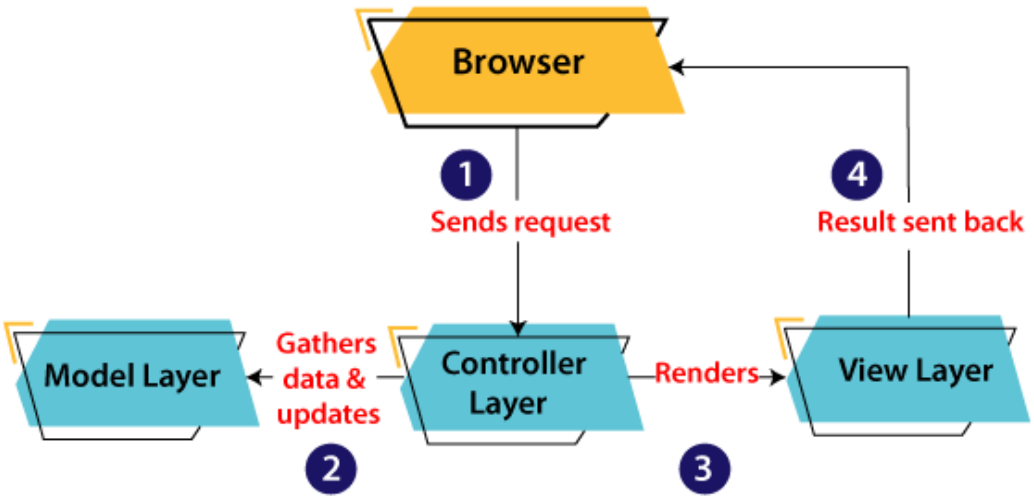


Рисунок 2.2 – Графічний вигляд архітектури MVC

2.2.1 Модель в архітектурі MVC

У патерні архітектури MVC Модель (Model) відповідає за представлення даних та логіку додатку, яка пов'язана з цими даними. Модель представляє собою окремий компонент, який може бути використаний в багатьох частинах додатку та має наступні основні функції.

Забезпечення доступу до даних: Модель відповідає за зберігання та доступ до даних додатку. Дані можуть зберігатися в базі даних, файловій системі, веб-службах або будь-якому іншому джерелі даних. Модель забезпечує доступ до цих даних та забезпечує, що вони відображають поточний стан додатку.

Логіка додатку: Модель містить логіку додатку, яка відповідає за обробку даних та роботу з ними. Наприклад, Модель може містити логіку перевірки валідності даних, розрахунку показників, генерації звітів тощо.

Взаємодія з Контролером: Модель взаємодіє з Контролером, щоб обмінюватися даними та виконувати логіку додатку. Контролер може викликати функції Моделі, щоб отримати доступ до даних та виконати розрахунки.

Повідомлення про зміни даних: Модель повідомляє Контролер про будь-які зміни даних, які відбуваються у додатку. Це дозволяє Контролеру оновлювати відображення даних на Виді.

Незалежність від інтерфейсу користувача: Модель повинна бути незалежною від інтерфейсу користувача. Це означає, що Модель повинна працювати з даними незалежно від того, яким чином вони будуть відображені на Виді.

Тести: Модель є легко тестованою. Оскільки Модель містить логіку додатку та доступ до даних, вона може бути протестована незалежно від інтерфейсу користувача та Виду. Це дозволяє підтримувати якість коду та запобігати виникненню помилок у додатку.

Незалежність від технологій: Модель повинна бути незалежною від технологій. Це означає, що Модель не повинна залежати від конкретної бази даних, веб-служби

або будь-якої іншої технології. Це дозволяє змінювати технології, які використовуються в додатку, не змінюючи Модель.

Незалежність від інших компонентів: Модель повинна бути незалежною від інших компонентів додатку. Це означає, що Модель повинна бути незалежною від Виду та Контролера, і не повинна залежати від їх реалізації. Це дозволяє збільшувати масштабованість та забезпечувати більшу гнучкість додатку.

Узагальнюючи, Модель у патерні архітектури MVC представляє собою складову, яка забезпечує доступ до даних та містить логіку додатку. Це дозволяє забезпечити незалежність від інших компонентів додатку та забезпечує легке тестування.

2.2.2 Вид в архітектурі MVC

У патерні архітектури MVC Вид (View) є однією з основних складових. Вид відповідає за представлення даних користувачу та забезпечує інтерфейс взаємодії з додатком. Дані для Виду надходять з Моделі, а дії користувача передаються до Контролера для подальшої обробки, тобто основними функціями виду є наступні.

Представлення даних: Вид відповідає за представлення даних користувачу у вигляді веб-сторінки, форми або іншого інтерфейсу взаємодії. Вид повинен бути зручним та привабливим для користувача, тому його дизайн має бути ретельно продуманим.

Реагування на дії користувача: Вид повинен бути здатний реагувати на дії користувача, такі як натискання кнопок, заповнення форм тощо. Дії користувача передаються до Контролера для подальшої обробки.

Незалежність від Моделі: Вид повинен бути незалежним від Моделі та містити мінімум логіки. Він отримує дані з Моделі та відображає їх, але не займається їх обробкою. Це дозволяє зберігати простоту та гнучкість Виду, що дозволяє легко змінювати його без впливу на Модель та Контролер.

Незалежність від технологій: Вид повинен бути незалежним від технологій та може використовувати будь-яку технологію, яка дозволяє створювати веб-інтерфейси.

Повторне використання: Вид повинен бути повторно використовуваним. Якщо у додатку є кілька екранів зі схожим виглядом, то можна використовувати один і той самий Вид для їх відображення.

Шаблонізація: Шаблонізація дозволяє використовувати один і той самий Вид для відображення різних даних. Шаблони Виду містять заготовки коду, які можна заповнювати даними з Моделі. Це дозволяє швидко створювати нові Види та забезпечує їх гнучкість.

Розділення на рівні: Вид повинен бути розділений на рівні, щоб забезпечити його гнучкість та простоту розробки. Наприклад, великий Вид може бути розділений на менші підвиди, кожен з яких відповідає за свою частину інтерфейсу.

Тестованість: Вид повинен бути легко тестованим. Для цього можна використовувати автоматизовані тести, які перевіряють правильність відображення даних та взаємодії з користувачем.

Узагальнюючи, у архітектури MVC Вид є одним із ключових компонентів, який відповідає за представлення даних користувачу та забезпечує інтерфейс взаємодії з додатком. Забезпечуючи його гнучкість та простоту розробки, можна створювати швидкі та ефективні веб-додатки.

2.2.3 Контроллер в архітектурі MVC

У патерні архітектури MVC, Контролер відповідає за обробку запитів користувача та взаємодію з Моделлю та Видом для забезпечення функціональності додатку, його основними функціями є:

Обробка запитів: Контролер приймає запити від користувача та виконує необхідні дії для обробки цих запитів.

Розподіл завдань: Контролер може розподіляти завдання між Моделлю та Видом, щоб забезпечити ефективну взаємодію між ними та підтримку розширюваності та гнучкості.

Обробка помилок: Контролер може обробляти помилки та повідомлення про помилки, що повертаються від Моделі та Виду, та надавати відповідні повідомлення для користувача.

Маршрутизація: Контролер може здійснювати маршрутизацію запитів на основі шляху запиту та інших параметрів. Наприклад, запити на адресу "/users" можуть бути оброблені контролером, що відповідає за користувачів.

Тестування: Контролер повинен бути легко тестованим для підтримки безперервної інтеграції та розробки.

Розділення на рівні: Контролер повинен бути розділений на рівні, щоб забезпечити його гнучкість та простоту розробки. Наприклад, великий контролер може бути розділений на менші підконтролери, кожен з яких відповідає за свою частину функціональності додатку.

Зберігання стану: Контролер може зберігати стан додатку та переключати його між станами, наприклад, для обробки різних типів запитів.

Розробка: Контролер повинен бути легко розширюваним та змінюваним для підтримки розробки нових функцій та внесення змін до існуючої функціональності.

Валідація: Контролер може виконувати валідацію вхідних даних та виводити повідомлення про помилки користувачу.

Сесії: Контролер може зберігати дані сесій користувача та виконувати необхідні дії для забезпечення безпеки та конфіденційності даних.

У більш складних додатках Контролер може містити декілька методів дії, кожен з яких відповідає за конкретну функціональність додатку. Це може допомогти зберегти код чистим та організованим, зменшити кількість дублюючого коду та зробити додаток більш підтримуваним та розширюваним.

Узагальнюючи, Контролер у патерні архітектури MVC відповідає за обробку запитів користувача, маршрутизацію запитів, взаємодію з Моделлю та Видом, обробку помилок, збереження стану додатку та інші функції, необхідні для забезпечення функціональності та ефективності додатку.

2.2.4 Переваги архітектури MVC

Переваги архітектури Model-View-Controller (MVC) перш за все обумовлюються за разунок розділення обов'язків між моделю, видом та контролером,які відповідають за свої частини функціональності, що зменшує кількість залежностей між компонентами додатку. Це дозволяє зберігати код чистим та організованим, сприяє підтримці та розширенню додатку та мати наступні плюси.

Висока повторюваність. Модель та Вид можуть бути повторно використані у різних контекстах, що зменшує кількість дублюючого коду та сприяє швидкому розробці.

Легкість тестування. Компоненти MVC можна тестувати окремо один від одного, що зменшує час, необхідний для виконання тестів та сприяє збільшенню якості коду.

Розширюваність. Кожен компонент MVC може бути легко розширений та змінений без впливу на решту додатку. Це дозволяє швидко додавати нові функції та внесення змін до існуючої функціональності.

Підтримка багатомовності. У MVC можна легко розділити логіку додатку та його інтерфейс користувача. Це сприяє підтримці багатомовних додатків.

Гнучкість. Архітектура MVC дозволяє використовувати різні технології для реалізації кожного з компонентів. Наприклад, Модель може використовувати базу даних, або може бути збережена у файловій системі. Вид може бути реалізований за допомогою різних інтерфейсів користувача, таких як веб-сторінки, мобільні додатки або десктопні програми.

Команда розробників може бути розділена на групи, які відповідають за різні компоненти архітектури. Це сприяє більш ефективній роботі, якщо команда складається з багатьох розробників.

Архітектура MVC дозволяє використовувати шаблони проектування та добрі практики програмування, що забезпечує високу якість коду та зменшує ризик виникнення помилок.

Код може бути більш зрозумілим та читабельним, оскільки він організований за логічними блоками. Це дозволяє розробникам швидко зорієнтуватися у коді та знайти необхідну функціональність.

Існує багато фреймворків, які реалізують архітектуру MVC та надають додаткові можливості для розробки додатків, наприклад, ASP.NET MVC, Ruby on Rails, Laravel та багато інших. Використання таких фреймворків дозволяє збільшити швидкість розробки та зменшити кількість дублюючого коду.

Загалом, архітектура MVC дозволяє розробникам створювати додатки, які є легкими для розуміння та підтримки. Вона сприяє розділенню обов'язків та зменшує залежності між компонентами додатку, що забезпечує більш ефективну роботу розробників та збільшує швидкість розробки.

2.2.5 Недоліки архітектури MVC

Хоча архітектура MVC має багато переваг, вона також має деякі недоліки, які варто враховувати при виборі архітектури для проекту.

Під час розробки додатків за архітектурою MVC може виникнути багато коду, оскільки кожен компонент має свою власну логіку та функціональність. Це може збільшити складність проекту та зробити його важким для підтримки.

Перекладання даних між компонентами додатку може бути витратним та викликати проблеми з продуктивністю. Крім того, якщо дані в одному з компонентів додатку змінюються, то потрібно оновлювати інші компоненти, що може викликати складність та підвищення ризику помилок.

Розробка додатків за архітектурою MVC може вимагати від розробника знань та досвіду використання шаблонів проектування та інших добрих практик програмування. Це може збільшити складність вивчення та розробки проекту для початківців.

Незважаючи на те, що архітектура MVC зменшує залежності між компонентами додатку, вона не розглядає взаємодію між компонентами на вищому рівні, що може створювати складнощі при розробці складних додатків.

Іноді може виникати проблема з взаємодією між різними моделями, які використовуються в різних компонентах додатку. Це може призвести до дублювання даних та погіршення продуктивності.

Як висновок, архітектура MVC є ефективним інструментом для розробки програмного забезпечення, особливо для великих та складних проектів, але вона має свої недоліки та не підходить для всіх типів додатків. Вибір архітектури повинен залежати від специфіки проекту та вимог до додатку, а також від знань та досвіду розробника.Тобто, MVC не є панацеєю для всіх видів додатків та не вирішує всі проблеми, пов'язані з проектуванням та розробкою програмного забезпечення. Важливо також враховувати інші аспекти розробки додатків, такі як безпека, тестування, взаємодія з користувачами та інші.

2.3 Розробка архітектури сервісу

Сервіс White Elephant для обміну непотрібними подарунками створено на основі архітектури MVC, яка використовується для створення веб-додатків на мові C# з використанням фреймворка ASP.NET. Основні компоненти цієї архітектури це модель, представлення та контролер.

Модель (Model) -в цьому додатку представлена класами, розташованими у каталозі Models. Вони відповідають за обробку даних та логіку програми. У даному

випадку, класи моделі взаємодіють з базою даних для отримання та збереження інформації.

Представлення (View) в цьому додатку розташовані у каталозі Views. Вони відображають дані, які були отримані з моделі. Представлення дозволяють користувачеві взаємодіяти з додатком, надаючи йому можливість редагувати, видаляти та додавати дані.

Контролери (Controller) в цьому додатку розташовані у каталозі Controllers. Вони відповідають за обробку запитів користувачів та взаємодію з моделлю та представленнями. Контролери отримують запити від користувача та виконують певну логіку, зазвичай зв'язану з роботою з моделлю та передачі відповідей в представлення.

У даному додатку архітектура MVC є відповідальністю фреймворка ASP.NET та його компонентів. Для пояснення більш детальної архітектури додатку кожен компонент кожен компонент розглянуто детальніше.

Модель в додатку відповідає за обробку даних та бізнес-логіку програми. У даному випадку, модель розташована у каталозі Models та складається з таких класів, як Product.cs, Category.cs, Order.cs, User.cs тощо. У класах моделі описані властивості, які визначають структуру та типи даних, які будуть використовуватись в додатку. Крім того, вони також містять методи, що забезпечують доступ до бази даних, отримання та збереження інформації.

Представлення відповідають за відображення даних, які були отримані з моделі. У додатку "W-E" вони розташовані у каталозі Views та складаються з різних файлів з розширенням .cshtml, наприклад, Index.cshtml, Edit.cshtml, Details.cshtml тощо. У файлі представлення використовуються технології Razor та HTML, які дозволяють відображати дані з моделі та створювати веб-сторінки. У додатку "W-E" також використовується фреймворк Bootstrap для стилізації веб-сторінок та покращення їх візуального вигляду.

Контролери відповідають за обробку запитів користувачів та взаємодію з моделлю та представленнями. У додатку "W-E" вони розташовані у каталозі

Controllers та складаються з таких класів, як AccountController.cs, LotsController.cs, OrdersController.cs. Контролери отримують запити від користувачів та викликають відповідні методи моделі для отримання та збереження інформації. Після цього, вони викликають відповідні представлення для відображення даних. Крім того, контролери також можуть виконувати додаткову обробку запитів, наприклад, валідацію введених даних, обробку помилок, перенаправлення користувача на іншу веб-сторінку тощо.

2.4 Опис інструментів розробки

2.4.1 Мова програмування C#

Мова C# є об'єктно-орієнтованою мовою програмування, розробленою компанією Microsoft. Її розробка була оголошена в 1999 році і вона була випущена в 2002 році як частина платформи .NET Framework, вона використовується для розробки різноманітних додатків, включаючи десктопні програми, веб-додатки, ігри, мобільні додатки і багато іншого. Вона має багато спільного з мовами програмування Java і C++, що дозволяє програмістам, які вже працювали з цими мовами, швидко вивчити C#. Також, C# має багато функцій, таких як автоматичне управління пам'яттю, механізм делегування, лямбда-вирази, атрибути і т.д. Вона також підтримує асинхронне програмування, що дозволяє писати ефективні коди, які не блокують головний потік виконання. C# є платформонезалежною мовою програмування, що дозволяє її виконувати на різних операційних системах, таких як Windows, Linux і macOS. Завдяки тому, що вона є частиною платформи .NET, вона підтримує ряд інших мов програмування, таких як F#, VB.NET і інші.

Однією з головних переваг C# є те, що вона є частинами платформи .NET Framework, що дозволяє програмістам легко взаємодіяти з іншими компонентами платформи .NET, такими як Windows Forms, ASP.NET і ADO.NET.C# підтримує різні типи даних, такі як цілі числа, дійсні числа, рядки, булеві значення і т.д. Вона також підтримує масиви, колекції, структури, переліки та інші типи даних. C# має вбудовану

підтримку LINQ (Language-Integrated Query), що дозволяє програмістам виконувати запити до даних з використанням зручної синтаксису. Вона також підтримує взаємодію з базами даних через ADO.NET і Entity Framework.Ще однією перевагою C# є те, що вона підтримує безпечну роботу з пам'яттю, що дозволяє уникнути багатьох проблем, пов'язаних з використанням неконтрольованої пам'яті.

Узагальнюючи, C# є потужною мовою програмування з багатьма перевагами, що дозволяють розробляти різноманітні додатки в різних сферах, включаючи веб-розробку, мобільні додатки, десктопні додатки і багато іншого.

2.4.2 ASP.NET MVC Framework

ASP.NET MVC Framework є одним з найпопулярніших фреймворків для розробки веб-додатків на мові програмування C#. Цей фреймворк заснований на підході Model-View-Controller (MVC), який дозволяє розділити додаток на три складові: модель (Model), представлення (View) та контролер (Controller).

Модель відповідає за обробку даних додатку. Це можуть бути дані з бази даних, веб-сервісів або будь-які інші дані, з якими повинен працювати додаток. Зазвичай модель містить класи, які описують сутності, з якими працює додаток. Представлення відповідає за відображення даних додатку. Це можуть бути HTML-сторінки, зображення, PDF-файли та інші типи вмісту, який відображається користувачу. У ASP.NET MVC представлення зазвичай створюються з використанням Razor-шаблонів, які дозволяють поєднувати HTML-код зі змінними та кодом на C#.

Контролер відповідає за обробку запитів користувача та керування взаємодією між моделлю та представленням. Контролер отримує запит від користувача, обробляє його та виконує потрібні дії, щоб повернути відповідь користувачу. У контролері можуть бути методи для обробки різних типів запитів, таких як GET, POST, PUT та DELETE.

Окрім цих основних складових, ASP.NET MVC має такі функції, як маршрутизація, валідація введених даних, авторизація та аутентифікація

користувачів. Маршрутизація дозволяє визначити, який метод контролера має бути викликаний для обробки певного запиту. Валідація введених даних дозволяє перевірити, чи відповідає введені користувачем дані заданим вимогам, наприклад, чи містить вони потрібні символи або не перевищують максимальну довжину.

Авторизація та аутентифікація дозволяють контролювати доступ користувачів до різних частин додатку та забезпечувати безпеку даних.ASP.NET MVC підтримує використання технологій, таких як AJAX та jQuery, що дозволяє створювати більш інтерактивні та динамічні веб-додатки. Крім того, ASP.NET MVC дозволяє розробникам використовувати різні шаблони проектів, такі як Empty, Web Application або Web API, що дозволяє створювати додатки з різними цілями та функціональністю.

Однією з переваг ASP.NET MVC є його висока продуктивність, яка забезпечується використанням різних оптимізаційних технік та кешуванням даних. Крім того, фреймворк має велику спільноту розробників, яка допомагає вирішувати проблеми та підтримувати фреймворк.

У загальному, ASP.NET MVC є потужним та гнучким фреймворком для розробки веб-додатків на мові програмування C#. Він дозволяє розробникам легко розділити логіку додатку на модулі та дозволяє створювати високоякісні веб-додатки з використанням найсучасніших технологій.

2.4.3 Razor Pages

Razor Pages - це фреймворк, що входить до складу ASP.NET Core, який дозволяє легко створювати веб-додатки з використанням моделі сторінок. Він базується на синтаксисі Razor, що дозволяє швидко та ефективно розробляти сторінки веб-додатків.Основна ідея застосування Razor Pages полягає в тому, щоб кожна сторінка веб-додатку мала свій власний код та модель, що забезпечує зручність розробки та підтримки коду. Розробка з використанням Razor Pages дозволяє створювати додатки з високою продуктивністю та швидкістю завантаження сторінок.Фреймворк Razor Pages має декілька ключових особливостей.

Модель сторінки. Razor Pages дозволяє створювати окремі моделі для кожної сторінки веб-додатку. Це дозволяє зберігати логіку сторінки окремо від інших частин додатку та робить розробку та підтримку коду більш зручною.

Розділення коду та представлення. Razor Pages дозволяє розділяти код та представлення, що дозволяє зробити код більш чистим та організованим.

Вбудований підтримка AJAX. Razor Pages має вбудовану підтримку AJAX, що дозволяє створювати динамічні сторінки з використанням AJAX-запитів.

Вбудований підтримка валідації даних. Razor Pages дозволяє використовувати вбудовану підтримку валідації даних для перевірки правильності введення даних користувачем.

Розширюваність та гнучкість. Razor Pages дозволяє використовувати бібліотеки та розширення, що забезпечують більшу гнучкість та функціональність фреймворку.

Інтеграція з ASP.NET Core. Razor Pages побудований на базі фреймворка ASP.NET Core, що дозволяє легко інтегрувати Razor Pages з іншими компонентами та функціоналом ASP.NET Core.

Легкість використання. Razor Pages є легким та зрозумілим для використання фреймворком. Розробка з використанням Razor Pages не вимагає великої кількості коду, що дозволяє зосередитися на розробці функціоналу та дизайну сторінок.

SEO-оптимізація. Razor Pages дозволяє створювати SEO-оптимізовані сторінки, що забезпечує підвищення рейтингу додатку в пошукових системах.

Шаблони. Razor Pages має вбудовану підтримку шаблонів, що дозволяє створювати повторювані компоненти та використовувати їх на різних сторінках додатку.

Вбудована підтримка автентифікації та авторизації. Razor Pages має вбудовану підтримку автентифікації та авторизації, що дозволяє легко налаштувати ці функції для додатку.

В цілому, фреймворк Razor Pages є потужним інструментом для розробки веб-додатків з високою продуктивністю та зручним інтерфейсом. Його можна

використовувати для створення різноманітних додатків, від простих блогів до складних корпоративних систем.

2.4.4 MS SQL Server

Microsoft SQL Server (MS SQL Server) - це система керування базами даних (СКБД), яка розробляється компанією Microsoft. Вона використовується для зберігання та обробки великих обсягів даних, які можуть бути доступні з різних додатків. MS SQL Server має низку переваг порівняно з іншими СКБД, що дозволяє йому бути популярним серед підприємств та розробників. Деякі з цих переваг включають високу продуктивність - MS SQL Server може опрацьовувати великі обсяги даних швидко та ефективно, надійність та безпеку - MS SQL Server має вбудовані функції, що дозволяють забезпечити безпеку та надійність бази даних, наприклад, автоматичне резервне копіювання даних та захист від несанкціонованого доступу, підтримку масштабування до великих обсягів даних та багатокористувацького доступу, достатньо зручний та легкий у використанні інтерфейс, що дозволяє швидко створювати та керувати базами даних.

Також MS SQL Server має декілька редакцій, включаючи Enterprise, Standard та Express. Кожна з них має свої особливості та можливості. MS SQL Server може бути використаний для різних типів додатків, включаючи веб-сайти, додатки підприємств та аналітичні звіти, має також широку підтримку мов програмування, включаючи C#, VB.NET, Java та Python, що дозволяє розробникам використовувати свої улюблені мови програмування для роботи з базою даних.

У MS SQL Server є також можливість створювати процедури зберігання, функції та тригери, що дозволяє розробникам забезпечувати більш високу продуктивність та ефективність при обробці даних.

MS SQL Server також має підтримку для розподіленої обробки даних та кластеризації серверів, що дозволяє розподіляти навантаження на декілька серверів та забезпечувати високу доступність бази даних.

Одним з найбільш важливих аспектів MS SQL Server є його підтримка для бізнес-аналітики та звітності. MS SQL Server має вбудовані засоби для створення складних запитів та звітів, що дозволяє користувачам аналізувати дані та виконувати звіти з великої кількості джерел даних.

Узагальнюючи, MS SQL Server є потужною та надійною системою керування базами даних, що має велику кількість можливостей та функцій. Вона використовується для зберігання та обробки великих обсягів даних в різних додатках та має широку підтримку мов програмування. MS SQL Server також має засоби для бізнес-аналітики та звітності, що дозволяє користувачам аналізувати дані та виконувати звіти з великої кількості джерел даних.

2.4.5 ORM Entity Framework

ORM (Object-Relational Mapping) - це технологія, що дозволяє програмістам працювати з базою даних, використовуючи об'єктно-орієнтований підхід до розробки програмного забезпечення. ORM дозволяє використовувати об'єкти як рівень доступу до даних, замість роботи з запитами SQL безпосередньо. Entity Framework - це ORM-фреймворк для платформи .NET, який дозволяє програмістам працювати з базою даних за допомогою об'єктів. Entity Framework автоматично генерує SQL-запити на основі запитів до об'єктів та підтримує багато різних баз даних, включаючи MS SQL Server, Oracle та MySQL. Entity Framework має кілька ключових функцій та можливостей.

Маппінг об'єктів на таблиці бази даних. Entity Framework дозволяє програмістам визначати, які об'єкти відповідають таблицям бази даних, та які властивості об'єктів пов'язані з колонками таблиць.

Генерація SQL-запитів. Entity Framework автоматично генерує SQL-запити на основі запитів до об'єктів. Це означає, що програмістам не потрібно писати запити SQL безпосередньо.

Використання LINQ. Entity Framework підтримує мову запитів LINQ (Language Integrated Query), що дозволяє програмістам писати запити до бази даних з використанням звичайних мов програмування, таких як C# або VB.NET.

Модель Code First. Entity Framework має можливість визначати структуру бази даних на основі класів об'єктів. Це означає, що програмісти можуть спочатку визначити класи об'єктів, а потім Entity Framework автоматично створить необхідні таблиці бази даних.

Lazy Loading. Entity Framework підтримує lazy loading (ліниву завантаження), що дозволяє завантажувати дані з бази даних, коли вони потрібні, а не завантажувати всі дані одразу. Це може допомогти зменшити час завантаження та пам'ять, що використовується програмою.

Відслідковування змін. Entity Framework автоматично відслідковує зміни об'єктів та дозволяє програмістам легко зберігати ці зміни в базі даних. Це може допомогти забезпечити консистентність даних та уникнути конфліктів збереження даних.

Підтримка транзакцій. Entity Framework підтримує транзакції бази даних, що дозволяє забезпечити консистентність даних та уникнути некоректних змін.

Entity Framework є потужним інструментом для роботи з базами даних в платформі .NET. Він дозволяє програмістам працювати з базою даних за допомогою об'єктно-орієнтованого підходу та зменшити кількість коду, що потрібен для роботи з базою даних. З використанням Entity Framework програмісти можуть зосередитись на розробці функціональності програми, а не на деталях роботи з базою даних.

2.4.6 ASP.NET Core Identity

ASP.NET Core Identity є розширенням ASP.NET Core, яке забезпечує функціональність для аутентифікації та авторизації користувачів у веб-додатках. Цей інструмент надає зручні API для керування користувачами, ролями та авторизацією, також дозволяє створювати, редагувати та видаляти користувачів. Він забезпечує

можливість реєстрації нових користувачів, підтвердження їх електронної пошти, скидання та зміни паролів. Крім того, він надає засоби для роботи з ролями, дозволяючи створювати, редагувати та видаляти ролі, а також назначати користувачів на ролі.

Одним з ключових компонентів ASP.NET Core Identity є система авторизації. Вона дає можливість визначати політики авторизації, які визначають, які користувачі мають доступ до певних ресурсів або виконання певних дій. Політики авторизації можуть бути засновані на ролях, клеймах або вимогах до властивостей користувача. Завдяки цій системі, можна легко контролювати доступ користувачів до різних частин додатка. ASP.NET Core Identity забезпечує зручні інструменти для збереження даних користувачів. Він може працювати з різними джерелами збереження даних, включаючи SQL бази даних (за допомогою Entity Framework), NoSQL бази даних, а також зовнішні постачальники автентифікації, такі як Azure AD або Facebook.

Крім вбудованої функціональності, ASP.NET Core Identity дозволяє розширювати та настроювати його для власних потреб. Можна створювати власні класи моделей, реалізовувати власні

3. РЕАЛІЗАЦІЯ ФУНКЦІОНАЛЬНОСТІ ВЕБ-СЕРВІСУ

3.1 Розробка ключових функцій веб-сервісу

Аналіз особливостей процесу обміну непотрібними подарунками та існуючих програмних рішень дозволяє сформулювати вимоги до майбутнього програмного продукту. Програма для обміну непотрібними подарунками повинна бути реалізована для веб-середовища та забезпечувати наступні функції:

– реєстрація та авторизація користувача: система має можливість реєстрації нових користувачів та авторизації вже існуючих, користувачі можуть увійти в систему, щоб переглядати свої замовлення та редагувати свій профіль;

– відображення каталогу пропозицій на обмін: користувачі можуть переглядати список доступних подарунків;

– пошук пропозицій на обмін: користувачі можуть шукати конкретні типи подарунків в базі даних за допомогою форми пошуку на головній сторінці;

– додавання подарунків в “список побажань”: користувачі можуть додавати подарунків у свій кошик для їх відстеження;

– додавання та редагування товарів: адміністратори сервісу можуть додавати нові товари в базу даних та редагувати існуючі;

– створення договору про обмін: авторизованим користувачам надається можливість використовувати цей функціонал для створення договорів про обмін між ними та встановлення умов обміну товарів;

– історія створених договорів про обмін: після успішного обміну речами кожен авторизований користувач може знайти історію своїх обмінів у розділі "orders", який являє собою список з лотами та їхніми загальними даними;

– управління користувачами: даний розділ відображається виключно для адміністратора та містить список усіх зареєстрованих користувачів, а також загальну інформацію таку як повне ім'я, скорочене ім'я або нікнейм, та електронна пошта.

3.1.1 Реєстрація та авторизація користувача

У веб-сервісі наявна функціональність реєстрації користувача, що дозволяє створювати нові облікові записи для користувачів. Для реєстрації необхідно натиснути кнопку "Register" у верхньому правому куті сторінки, яка зображена на рисунку 3.1.

Рисунок 3.1 – Реєстрація та авторизація користувача

Після цього відбувається перехід на сторінку реєстрації, що зображена на рисунку 3.2, де користувач повинен заповнити форму з наступними полями:

–Full name;

–Email;

–Password;

–Confirm Password.

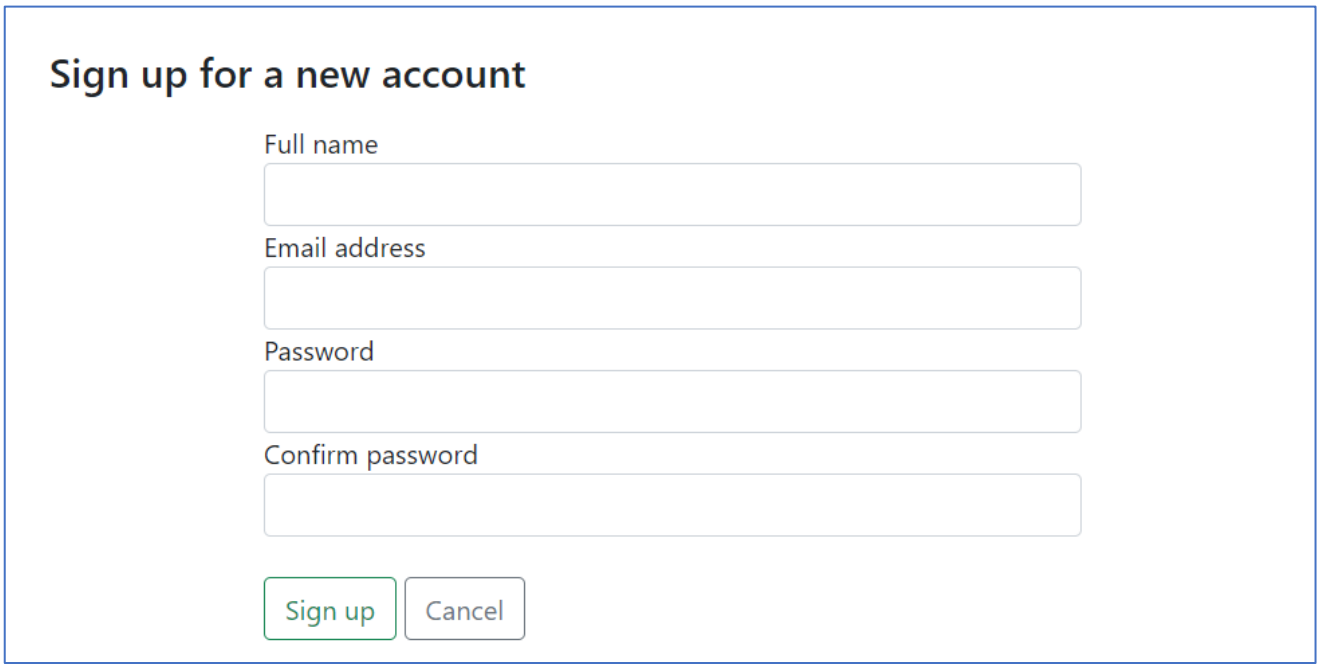


Рисунок 3.2 – Реєстрація нового користувача

Поля “Full name”, "Email", "Password" і "Confirm Password" є обов'язковими для заповнення. У разі, якщо введені паролі не співпадають, з'являється повідомлення про помилку та користувач повинен ввести паролі ще раз. Для заповнення інших полів користувач може використовувати свої особисті дані, які стосуються профілю користувача. Після заповнення всіх обов'язкових полів та натиснення кнопки "Sign up", інформація про нового користувача зберігається в базі даних. Крім того, користувач може побачити інформацію про свій обліковий запис, а також редагувати його, натиснувши на відповідні кнопки у верхньому правому куті сторінки.

Для того, щоб авторизуватися зареєстрованому користувачу потрібно натиснути на кнопку “Log in” у правому кутку сторінки, після чого для входу на сторінці заповнити поля “Email address” та “Password”, які зображені на рисунку 3.3.

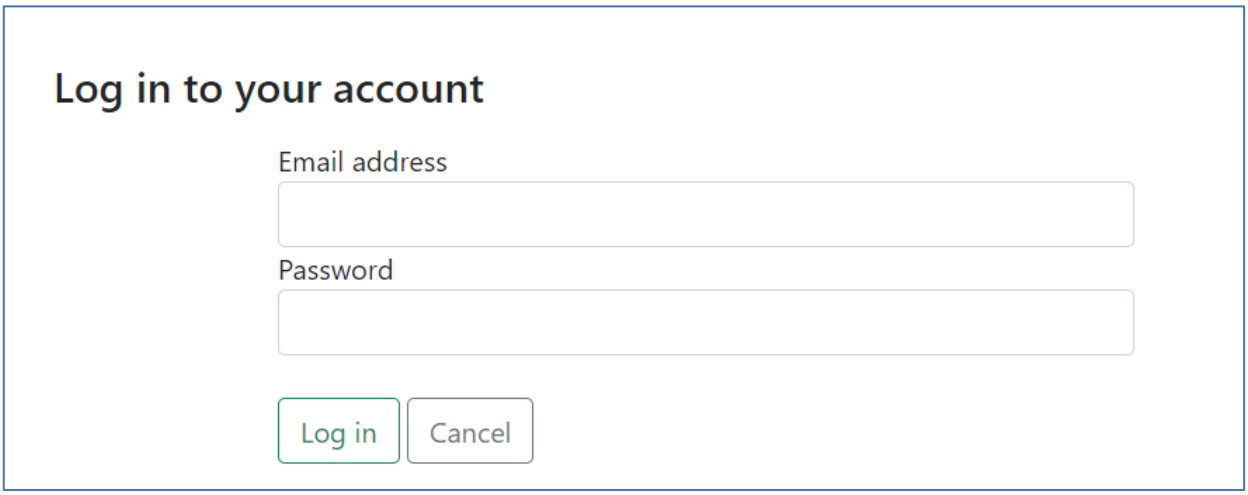


Рисунок 3.3 – Сторінка для входу зареєстрованого користувача

Для авторизації використовується механізм аутентифікації на основі токенів, який забезпечує безпеку передачі даних між клієнтом та сервером. Користувач може ввійти до системи, використовуючи електронну пошту та пароль. Авторизація користувача реалізована за допомогою ASP.NET Core Identity Framework, який надає стандартний набір функціоналу для авторизації і аутентифікації користувачів. Всі дані про користувачів (ім'я, електронна пошта, пароль) зберігаються в базі даних MS SQL

Server. Крім того, користувачі можуть бути розділені на різні ролі, такі як адміністратор, менеджер, користувач тощо, для керування доступом до різних частин веб-сайту.У проекті знаходиться контролер AccountController, який містить методи для реєстрації нового користувача та для отримання даних про поточного користувача.

3.1.2 Відображення каталогу пропозицій на обмін

Каталог містить список доступних подарунків, які можуть бути обмінені між користувачами системи. На цій сторінці можуть бути відображені наступні дані про кожен подарунок:

–зображення: може бути відображене зображення або фотографія подарунку, щоб користувач міг бачити його зовнішній вигляд;

–назва: під назвою подарунку може бути вказана його назва або заголовок, щоб користувач мав загальне уявлення про те, про що саме йде мова;

–опис: короткий опис або детальніша інформація про подарунок можуть бути надані, щоб дати користувачам більше інформації про його особливості або властивості;

–умови обміну: може бути вказаний тип обміну, тобто обміняти на щось або віддати безкоштовно, який допомагає користувачам зрозуміти його цінність у контексті обміну;

–додаткові атрибути: залежно від конкретної системи, можуть бути надані додаткові атрибути про подарунок, такі як категорія (наприклад, одяг, книги, електроніка), стан (новий, вживаний), розмір, колір тощо.

Описана функціональність дозволяє користувачам оглядати та вибирати подарунки, які вони бажають отримати в обмін на свій власний подарунок (рис. 3.4.)

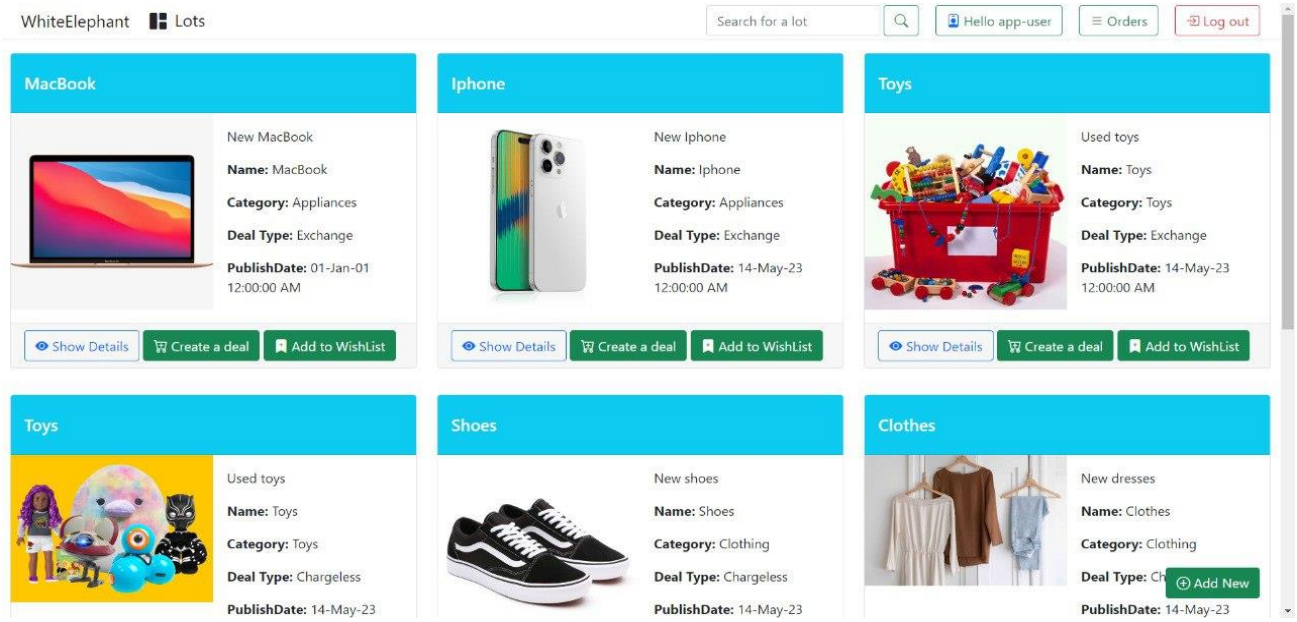


Рисунок 3.4 – Каталог пропозицій для обміну

3.1.3 Пошук пропозицій на обмін

Функціонал пошуку пропозицій, який зображено на рисунку 3.5, дозволяє користувачам здійснювати пошук та отримувати інформацію про доступні лоти.

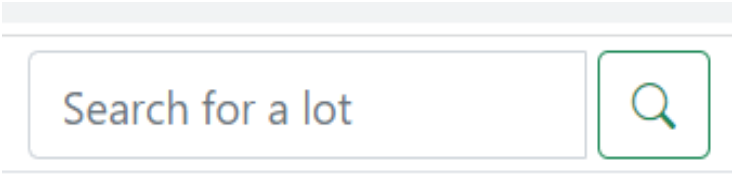


Рисунок 3.5 – Пошуковий рядок

Основні можливості включають нижченаведені.

Пошук за ключовими словами: користувач може ввести ключові слова або фрази, пов'язані з лотами, які його цікавлять. Наприклад, він може ввести назву товару, категорію, опис, або будь-яку іншу релевантну інформацію. Система шукає лоти, що містять введені ключові слова, і повертає відповідні результати. Результат зображено на рисунку 3.6.

Детальна інформація про лот: при виборі конкретного лота з результатів пошуку користувач отримує детальну інформацію про нього. Вона може включати заголовок лота, опис, фотографії, умови обміну, дату публікації лоту.

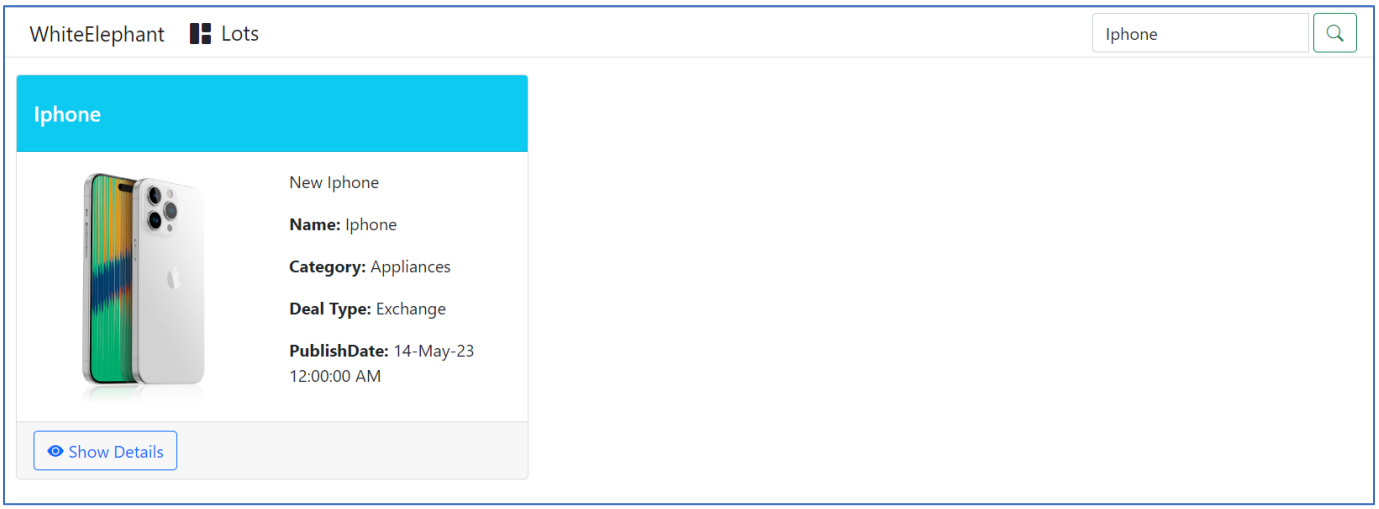


Рисунок 3.6 – Результати пошуку лоту

3.1.4 Додавання подарунків в “Список побажань”

"Додавання подарунків в список побажань" на веб-сервісі дозволяє авторизований користувачам відзначати та зберігати певні лоти, які вони бажають отримати у власному списку бажань. Основні можливості функціоналу включають нижченаведені.

Додавання до списку побажань. Користувач може додавати певні лоти або товари в свій список побажань шляхом натискання на відповідну кнопку “Add to Wishlist”, яка зображена на малюнку 3.7. Коли користувач знаходить цікавий йому лот, він може додати його до списку побажань, щоб зберегти його для подальшого використання.

Перегляд списку побажань. Користувач може переглядати свій список побажань, де відображаються всі лоти, які він додавав. Зайти до списку можна натиснувши на відповідний значок,який з'являється у верхньому правому кутку в момент додавання лоту в список. Користувач може переглянути інформацію про кожен пункт списку, таку як назва, зображення, умови обміну та інші деталі.

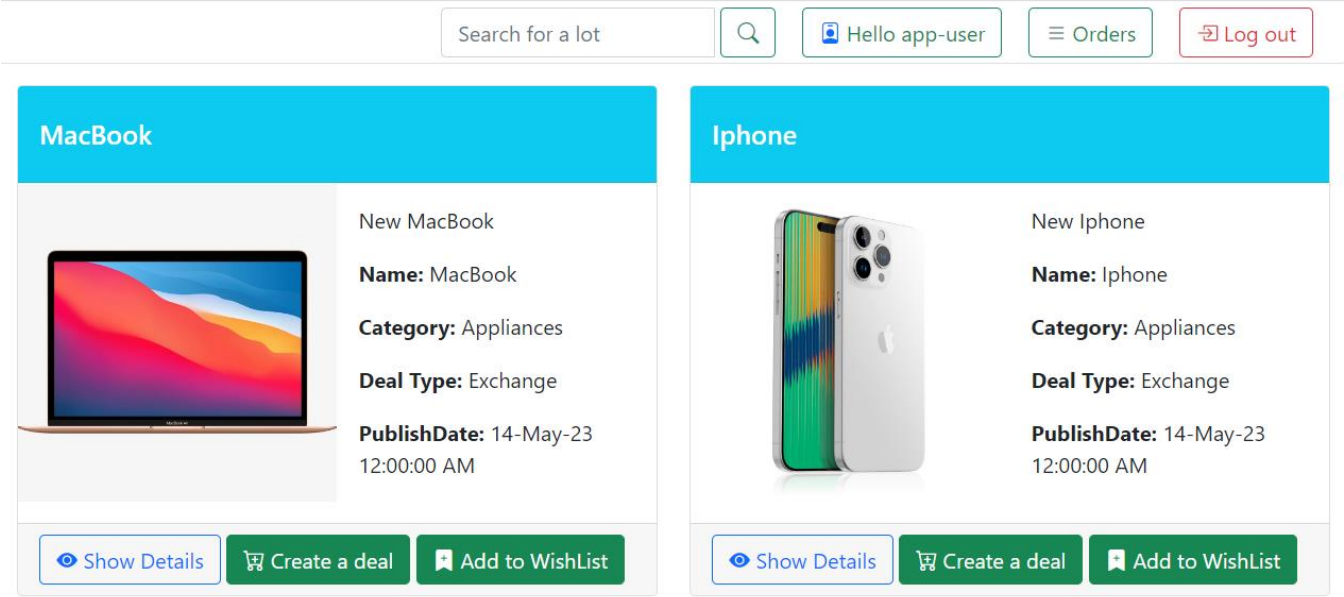


Рисунок 3.7 – Додавання лоту в список побажань

Видалення зі списку побажань. Користувач може видалити певний лот або товар зі свого списку побажань, якщо він більше не цікавиться отриманням цього предмета. Це дає користувачеві можливість керувати своїм списком побажань і видаляти елементи за потреби, що зображено на рисунку 3.8.

Також на сторінці присутній й інший функціонал, який взаємодіє з іншими сторінками веб-сервісу, як наприклад перехід на головну сторінку з лотами “Add more items” або створення договору про обмін “Create a deal”. Приклад зображено на рисунку 3.8.

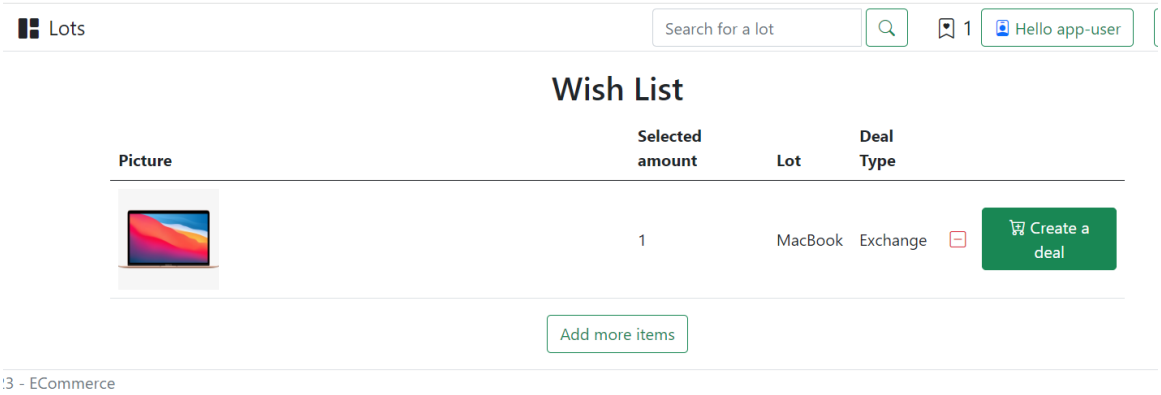


Рисунок 3.8 – Список бажань

3.1.5 Додавання та редагування товарів

Даний функціонал надає користувачам можливість додавати нові товари до системи та редагувати існуючі дані товарів. Це забезпечує оновлення та розширення каталогу товарів на сайті. Авторизований користувач має можливість додати лот натиснувши на кнопку “Add new” в лівому нижньому кутку, яка зображена на рисунку 3.9.

Рисунок 3.9 – Кнопка для додавання нового лоту

Основні можливості функціоналу "Додавання та редагування товарів" описано нижче.

Додавання нових товарів: користувач може заповнити форму для додавання нового товару до системи. Він може вказати такі дані як назва товару, опис, зображення, умови обміну, категорія, тощо. Після заповнення відповідних полів користувач може зберегти новий товар в системі, що дозволяє йому стати доступним для пошуку та перегляду іншими користувачами, що зображено на рисунку 3.10.

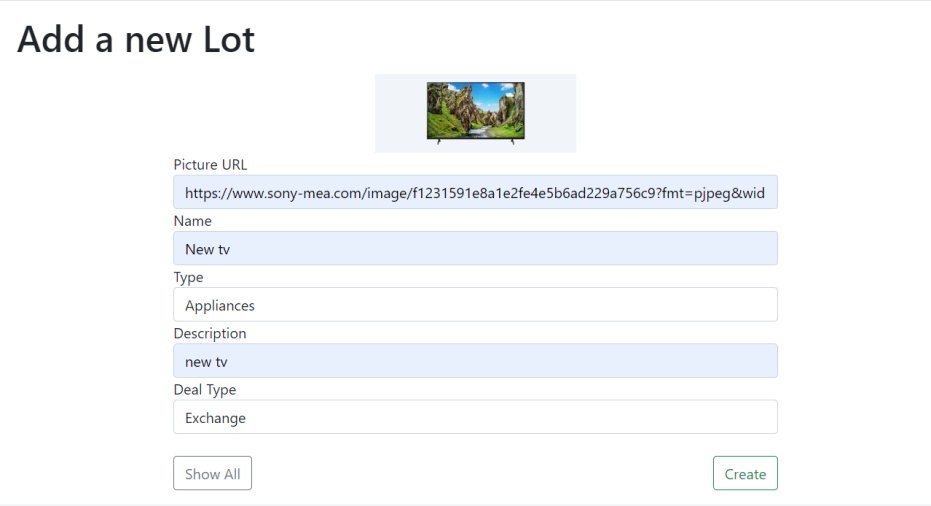


Рисунок 3.10 – Сторінка з формою для заповнення інформації про лот

Редагування існуючих товарів: тільки адміністратор може вибрати конкретний товар з каталогу та відредагувати його дані, натиснувши на значок поряд з назвою товару, який зображений на рисунку 3.11.

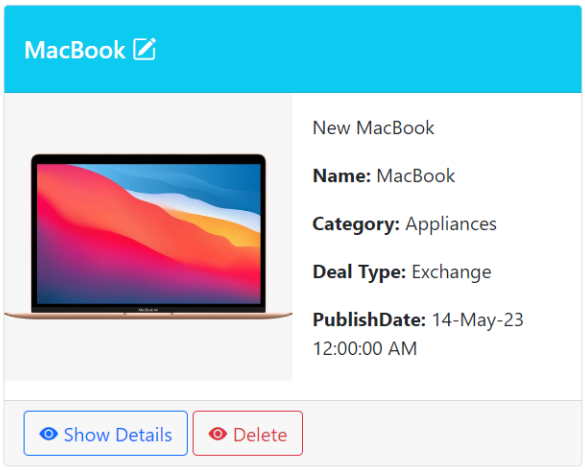


Рисунок 3.11 – Кнопка для редагування лоту

Він може змінити будь-яку інформацію, пов'язану з товаром, таку як назва, опис, зображення, умови обміну, категорія, тощо. Після внесення змін адміністратор може зберегти оновлені дані товару в системі, що зображено на рисунку 3.12.

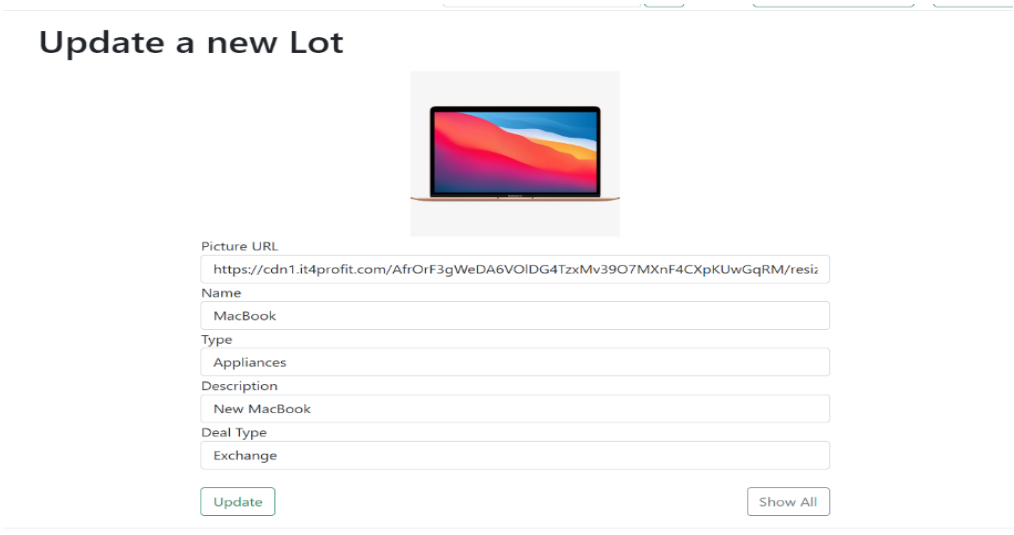


Рисунок 3.12 – Редагування інформації про лот

Перевірка та підтвердження змін. Перед збереженням нового товару або оновленням існуючого, система може здійснювати перевірку та підтвердження введених даних. Вона включає перевірку на наявність обов'язкових полів, правильність формату введених даних, щоб гарантувати коректність та цілісність даних товарів. Приклад наведено на рисунку 3.13.

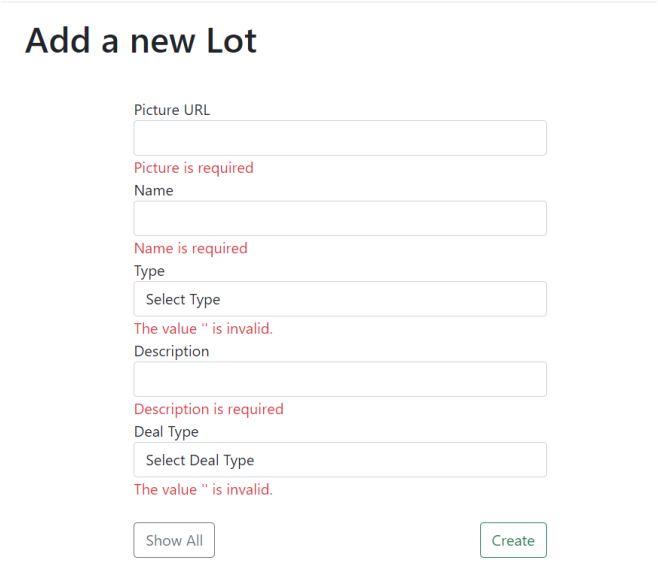


Рисунок 3.13 – Перевірка полів

3.1.6 Створення договору про обмін

Даний функціонал надає авторизованим користувачам можливість створювати договори про обмін між собою та установлення умов обміну товарів. Цей функціонал допомагає забезпечити чіткість, валідність та взаємну згоду сторін щодо обміну товарів. Натиснувши на кнопку “Create a deal”,яка зображена на рисунку 3.14., на сторінці зі створення договору.

Зайти на сторінку з обмінами можна натиснувши на значок корзини, який з'являється після додавання товару до неї. Надалі функціонал сторінки залежить від типу обміну, зазначеного автором лоту. Приклад зображено на рисунку 3.15.

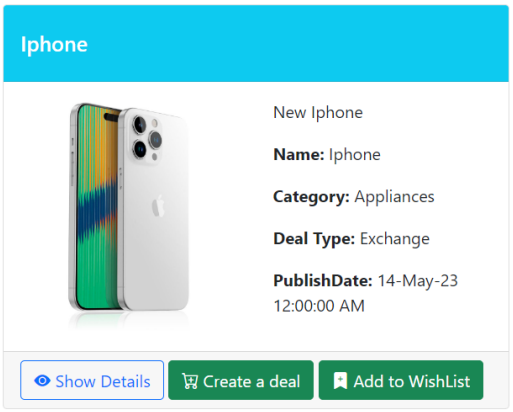


Рисунок 3.14 – Кнопка для створення договору про обмін

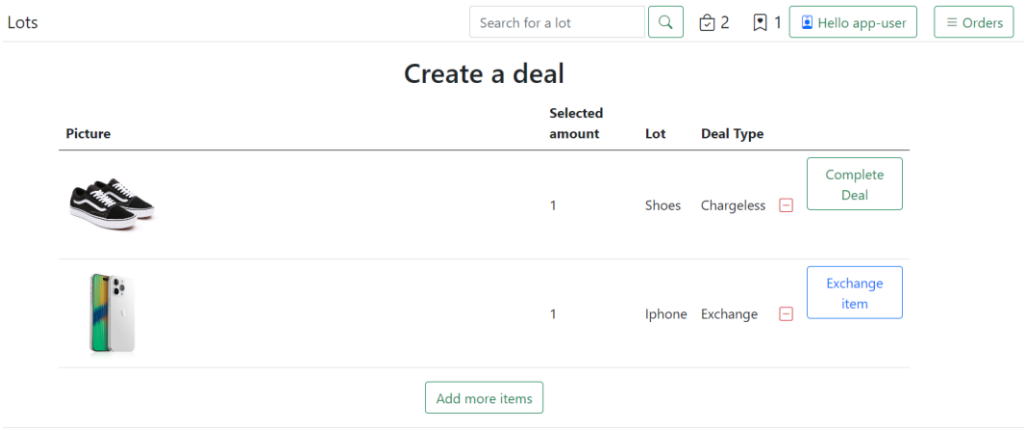


Рисунок 3.15 – Функціонал сторінки для створення договору про обмін

Різниця полягає в тому, що подарунок з типом “Chargless” віддається на безкоштовній основі та не потребує обміну з іншої сторони, тому достатньо натиснути на кнопку “Complete Deal”, щоб його отримати. Якщо ж користувач обрав лот з поміткою “Exchange”, він має запропонувати власну річ на обмін та заповнити форму, яка дасть головну інформацію про товар, а саме фотографію, опис та стан речі(нова або використанна).Приклад зображено на рисунку 3.16.

Дані поля також містять перевірку на наявність полів. Якщо ж користувач не заповнив дану форму, він отримає сповіщення, яке зображене на рисунку 3.17.

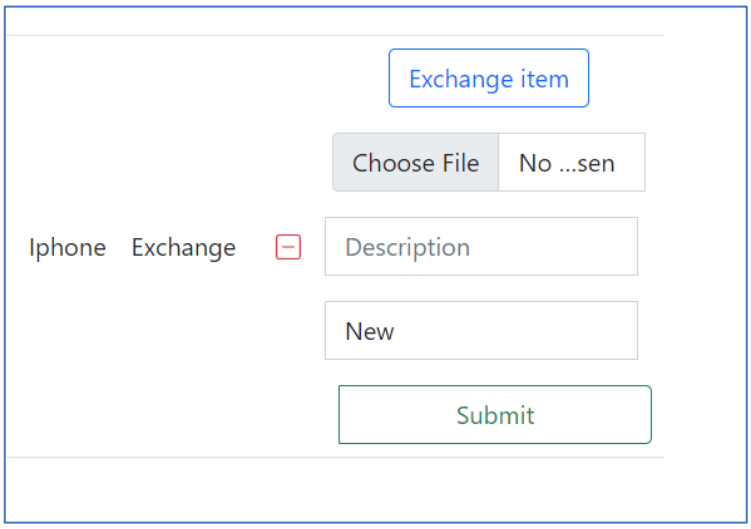


Рисунок 3.16 – Додавання речі на обмін до договору

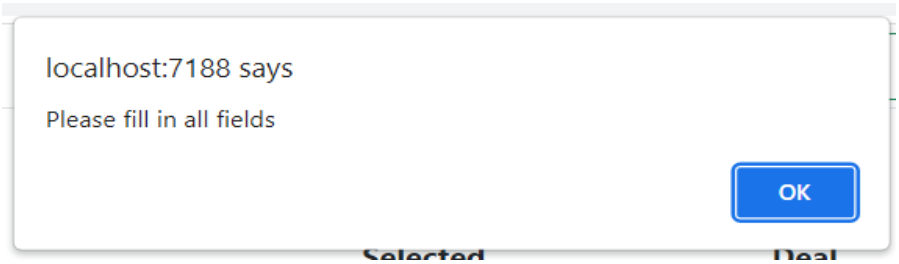


Рисунок 3.17 – Сповіщення про незаповнені поля

Після успішного заповнення форми користувач отримує сповіщення, про успішне відправлення запиту та може перевірити свій список замовлень у розділі “Orders”, що зображено на рисунку 3.18.

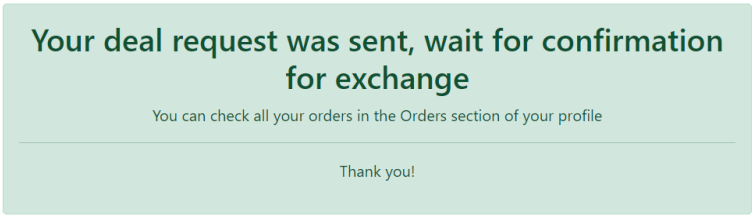


Рисунок 3.18 – Підтвердження про надсилання запиту для обміну

3.1.7 Історія створених договорів про обмін

Після успішного створення договору про обмін, речі для обміну з'являються на сторінці “Orders”. Приклад зображено на рисунку 3.19. Дана сторінка містить загальную інформацію про обмін таку як умови обміну, назва речі, кількість та зображення.

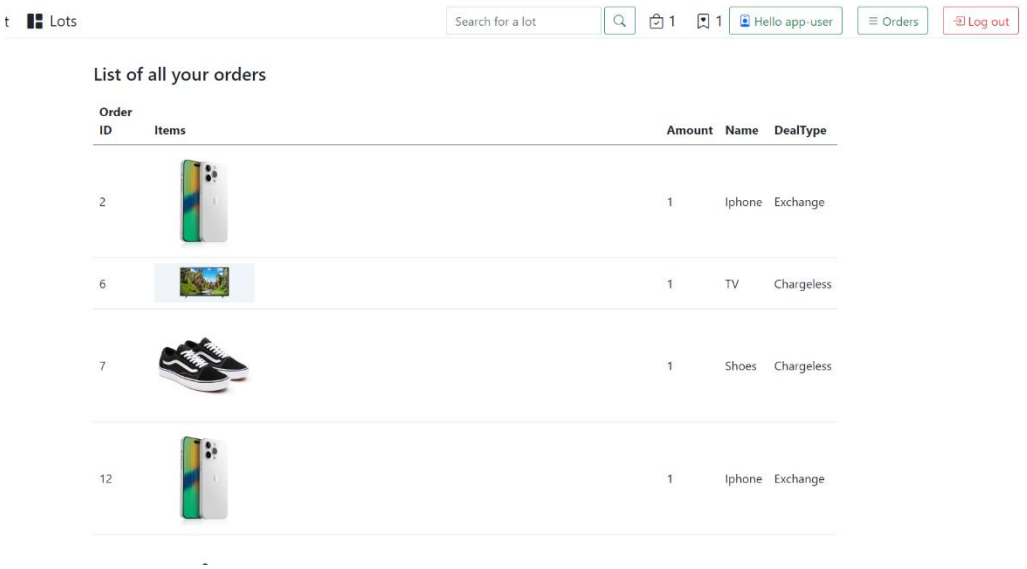


Рисунок 3.19 – Список замовлень

3.1.8 Управління користувачами

Управління користувачами доступне виключно для адміністратора та являє собою загальний список користувачів з наступною інформацією: повне ім'я, скорочене ім'я або нікнейм, та електронна пошта.

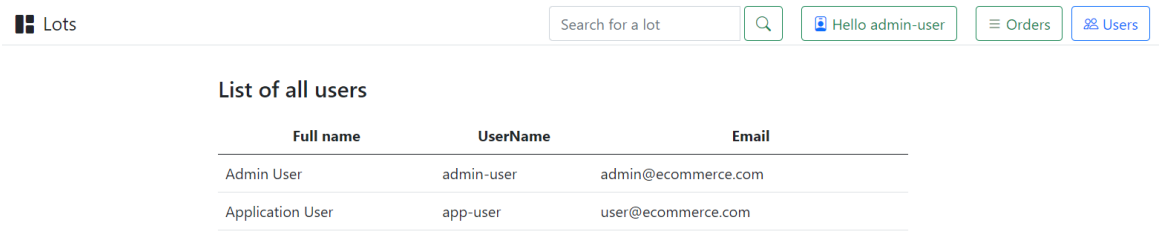


Рисунок 3.20 – Список користувачів

3.2 Розробка структури бази даних та моделей

Структура бази даних для веб-сервісу зазвичай залежить від конкретної функціональності сервісу. Зазвичай для веб-сервісів для обміну речами потрібні збереження даних про користувачів, лоти, замовлення та інші пов'язані дані. Основними таблицями сервісу є**:**

– таблиця користувачів (Users) - зберігає дані про зареєстрованих користувачів, такі як ім'я, прізвище, адреса електронної пошти, хеш пароля та інші додаткові дані;

– таблиця лотів (Lots) - зберігає дані про лоти, такі як назва, опис, фотографії, ціна, категорія, стан тощо;

– таблиця замовлень (Orders) - зберігає дані про замовлення, такі як ID користувача, ID лоту, кількість, загальна вартість та інші додаткові дані.

Ці таблиці пов'язані між собою за допомогою зовнішніх ключів, щоб забезпечити цілісність даних та зручний доступ до інформації. Наприклад, таблиця замовлень може містити зовнішній ключ, що посилається на таблицю користувачів, тим самим забезпечуючи зв'язок між замовленнями та користувачами.

В даному додатку використовується база даних SQL Server, зі структурою, що відповідає шаблону Code First з Entity Framework. В моделі бази даних присутні нижченаведені таблиці.

Модель WishListItem:

–Id: це унікальний ідентифікатор і є первинним ключем таблиці WishListItem;

–Amount: кількість одиниць товару;

–DealType: зберігає умови обміну;

–LotId: ідентифікатор лота, який був доданий до списку бажань;

–Lot: зв'язок з подарунком, який потрапляє до списку бажань;

–WishListId: зв'язок з списком бажань, котрому належить обраний подарунок;

–WishListModel: зв'язок з списком бажань.

Модель WishListModel:

–Id: це унікальний ідентифікатор і є первинним ключем таблиці WishListModel;

–Email: електронна пошта користувача;

–UserId: унікальний ідентифікатор користувача;

–User:зв'язок з користувачем, котрому належить сформований список бажань;

–WishListItems:речі у списку бажань.

Модель Lots:

–Id: це унікальний ідентифікатор лота і є первинним ключем таблиці Lots;

–Picture: зберігає шлях до зображення лота;

–Name: зберігає назву лота, це обов'язкове поле і його довжина має бути від 4 до 50 символів;

–Type: зберігає тип лота, це обов'язкове поле і може мати одне зі значень з перерахування LotType (Painting, Sculpture, Photography, Other);

–Description: зберігає опис лота, обов'язкове поле;

–DealType: зберігає умови обмін, обов'язкове поле;

–PublishDate: зберігає дату публікації лота на аукціоні.

Ця модель відображає структуру таблиці Lots і відповідає вимогам до зберігання даних про лоти. Крім того, у моделі використовується анотування Display для задання назв полів у вигляді тексту, що відображається на сторінках адміністратора.

Модель Order:

–Id: унікальний ідентифікатор замовлення, який є первинним ключем;

–Email: email користувача, який здійснив замовлення;

–UserId: унікальний ідентифікатор користувача, який здійснив замовлення, це зв'язується з таблицею користувачів ApplicationUser;

–User: зв'язок з користувачем, який здійснив замовлення;

–OrderItems: список елементів замовлення (клас OrderItem), який зберігає всі деталі замовлення.

Атрибут [Key] встановлює поле Id як первинний ключ таблиці. Атрибут [ForeignKey] встановлює зв'язок з таблицею ApplicationUser, тобто зберігає зовнішній ключ UserId для таблиці Order.

Модель OrderItem:

–Id: унікальний ідентифікатор товару у замовленні;

–Amount: кількість одиниць товару;

–LotId: ідентифікатор лота, який був доданий до замовлення;

–Lot: лот, який був доданий до замовлення;

–OrderId: ідентифікатор замовлення, до якого було додано цей товар;

–Order: замовлення, до якого був доданий цей товар.

Атрибут [Key] вказує, що поле Id є первинним ключем таблиці в базі даних. Атрибути [ForeignKey("LotId")] та [ForeignKey("OrderId")] вказують на зв'язок з відповідними таблицями бази даних за допомогою зовнішніх ключів.

Ця модель використовується для зберігання інформації про товари у замовленні, тобто кожен об'єкт цієї моделі відповідає одному товару у замовленні.

Модель ShoppingCartItem:

–Id - ключове поле таблиці, що відповідає за унікальний ідентифікатор запису в таблиці ShoppingCartItem;

–Lot - зв'язок з моделлю Lot, що вказує на конкретний лот, який додається до кошика покупця;

–Amount - кількість лотів, які додаються до кошика покупця;

–ShoppingCartId - ідентифікатор кошика, до якого додається лот.

Таким чином, модель ShoppingCartItem відображає зв'язок між лотами та кошиками покупців та дозволяє зберігати кількість кожного лота в кошику.

Загальна модель бази данних зображена на рисунку 3.21.

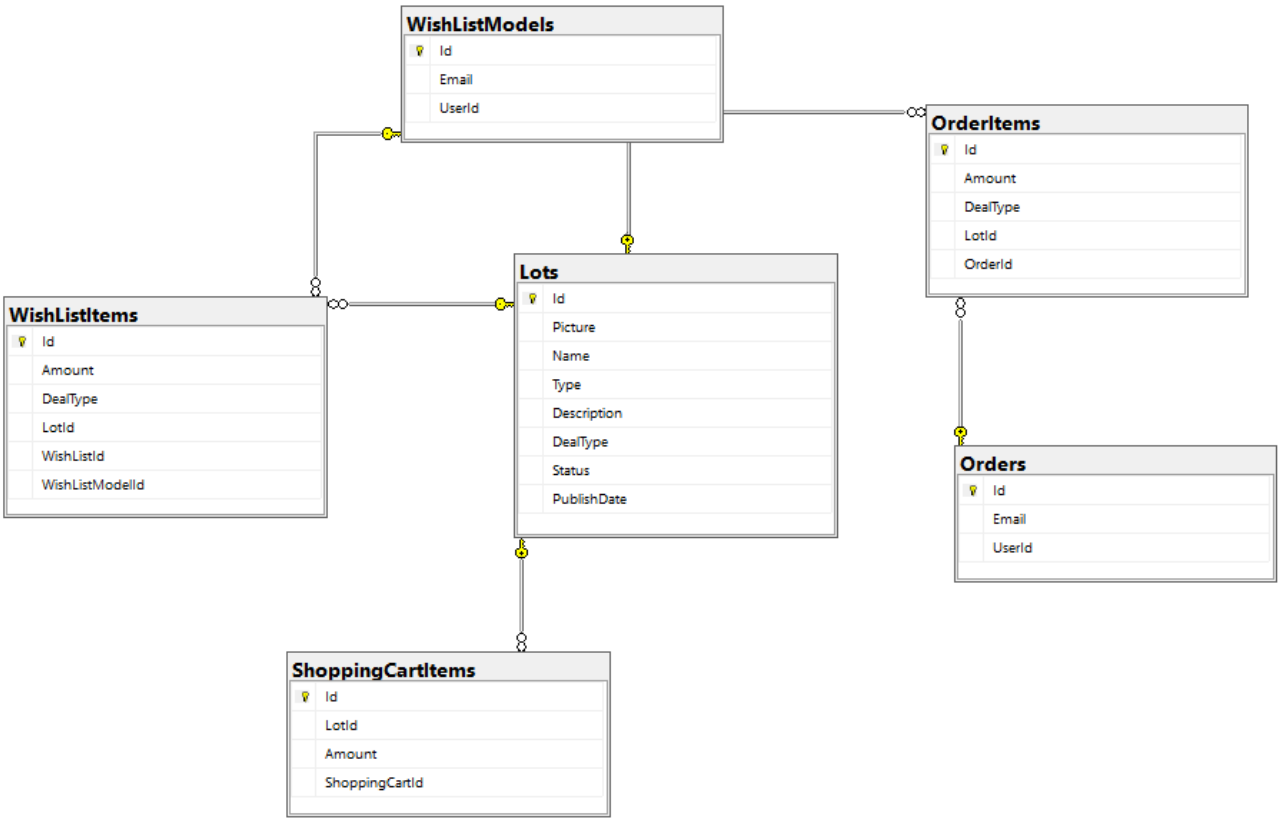


Рисунок 3.21 – Загальна модель бази даних

3.3 Структура моделей та їх функції

У сервісі існують наступні моделі та їх функції:

– клас WishListItem являє собою елемент списку бажань, який несе в собі інформацію про збережений лот, а також відповідає за відображення даних по кожному елементу списку;

– клас WishListModel використовується для відображення списку бажань користувача, тут відбувається підрахунок загальної кількості об’єктів списку та стислого опису кожного вибраного лоту;

– клас Lot - цей клас використовується для створення нового лоту, редагування існуючого лоту, відображення інформації про лот на сторінці деталей, а також для відображення списку лотів на головній сторінці та результатів пошуку;

– клас ShoppingCartItem - цей клас використовується для зберігання товарів в кошику, додавання товару до кошика, видалення товару з кошика та підрахунку загальної кількості предметів у кошику;

– клас Order - цей клас використовується для збереження інформації про замовлення користувача, відображення списку замовлень на сторінці профілю користувач;

– клас OrderItem представляє модель елементу замовлення, який відповідає за зберігання даних про кількість, умови обміну та зв'язки з лотом і замовленням.

4. УДОСКОНАЛЕННЯ ТА ПЕРСПЕКТИВИ РОЗВИТКУ ВЕБ-СЕРВІСУ

4.1 Пропозиції щодо покращення функціональності

Веб-аналітика - це дослідження даних веб-сайту, яке дозволяє зрозуміти, як користувачі взаємодіють з веб-сайтом та як можна покращити його функціональність для поліпшення користувацького досвіду. Веб-аналітика включає в себе збір даних, їх аналіз та висновки на основі цих даних, щоб покращити ефективність веб-сайту. У цьому розділі ми розглянемо основні інструменти та методи веб-аналітики, які можуть бути використані для вдосконалення веб-сервісу для обміну непотрібними подарунками.

Додати можливість видалення оголошень користувачами. Зараз немає можливості видалити оголошення після його створення. Якщо користувач помилився або вирішив забрати річ, йому необхідно звернутися до адміністратора, щоб видалити оголошення. Додавання кнопки "Видалити" на сторінку оголошення дозволить користувачам видаляти свої оголошення без додаткової допомоги адміністратора.

Додати функціонал зворотнього зв'язку між користувачами. Зараз немає можливості надсилати повідомлення користувачам, які розміщують оголошення. Додавання системи повідомлень дозволить користувачам зв'язуватися між собою і узгоджувати деталі обміну.

Додати можливість зберігати оголошення у вибраному списку. Користувачі могли б зберігати оголошення, які їм подобаються, у вибраному списку для подальшої перевірки. Це зручно, якщо користувач не має можливості зв'язатися з власником оголошення в той же час, але хоче зберегти цікаву йому річ.

Додати можливість додавати додаткові фотографії до оголошення. Зараз користувач може додати лише одну фотографію до оголошення. Додавання можливості додавати додаткові фотографії дозволить більш детально продемонструвати річ, яку користувач хоче обміняти.

Можливість відстежувати статус замін (прийнято, відхилено, в обробці тощо) та історію обміну.

Рейтинг користувачів, який відображає кількість успішних обміну і рівень довіри.

Використання мапи для показу місцезнаходження обмінюваних речей.

Можливість зберігати список обмінюваних речей, щоб користувачі могли згадати, що вже обмінювали та з ким.

Ці додаткові функції можуть значно покращити користувальницький досвід та зробити веб-сервіс більш привабливим для користувачів. Однак, їхнє втілення може зайняти багато часу та ресурсів, тому варто обдумати, які функції є найбільш важливими та потрібними для користувачів та розпочати з їх реалізації.

4.2 Аналіз можливостей розширення веб-сервісу

Для розширення функціональності веб-сервісу по обміну непотрібними подарунками можна розглянути декілька можливих напрямків:

– Додавання функціоналу соціальних мереж: забезпечення можливості зареєструватися та увійти до веб-сервісу за допомогою соціальних мереж (наприклад, Facebook, Twitter, Instagram), обмін повідомленнями між користувачами та можливість додавати публікації з інших соціальних мереж.

– Розширення можливостей пошуку: можна розширити функціонал пошуку за різними параметрами, такими як категорії, місцезнаходження, вартість товарів та інші параметри.

– Додавання системи рейтингів та відгуків: це може сприяти підвищенню довіри до користувачів веб-сервісу, оскільки вони зможуть знаходити інформацію про інших користувачів, що мають відповідні відгуки.

– Розширення системи оплати: додавання нових методів оплати (наприклад, Apple Pay, Google Pay), а також можливість перерахування коштів між користувачами.

Додавання механізмів забезпечення безпеки: забезпечення захисту даних користувачів від зловмисників, наприклад, шифрування даних та захист від SQL-ін’єкцій.

4.3 Аналіз питань безпеки та захисту даних

Питання безпеки та захисту даних є надзвичайно важливими для будь-якого веб-сервісу, зокрема і для сервісу по обміну непотрібними подарунками. Користувачі повинні бути впевнені, що їх особисті дані та інформація про обмін речами захищені від несанкціонованого доступу та використання. Однією з основних проблем є ризик хакерських атак та витоку даних. Тому важливо дотримуватись вимог стандартів безпеки, які забезпечують захист від таких загроз. Наприклад, сервіс повинен використовувати протокол HTTPS для шифрування всієї передачі даних між клієнтом та сервером. Також, потрібно забезпечити захист бази даних та її доступність тільки авторизованим користувачам. Іншою проблемою є використання слабких паролів користувачами, що робить їх облікові записи вразливими для злочинців. Тому потрібно забезпечити вимоги до паролів, наприклад, вимагати складних паролів та періодичну зміну паролів. Також важливо регулярно проводити аудит безпеки веб-сервісу, щоб виявити потенційні проблеми та вразливості та вжити необхідних заходів для їх усунення. Загалом, безпека та захист даних є надзвичайно важливими аспектами для будь-якого веб-сервісу, тому необхідно дбайливо стежити за їхньою ефективністю та проводити регулярні оновлення, щоб забезпечити безпеку користувачів та захист їхніх даних.

5 ТЕСТУВАННЯ ДОДАТКУ

Основна мета тестування програмного забезпечення полягає в перевірці, чи відповідає реальна поведінка програми очікуваній. Це досягається шляхом виконання певних тестів заздалегідь визначеним набором даних та параметрів. Тестування є одним із методів контролю якості програмного забезпечення. В цьому розділі розглядаються основні аспекти тестування веб-сервісів, розробляються тест-кейси для перевірки функціональності додатку і проводиться тестування відповідно до плану тестування.

5.1 Визначення об’єктів тестування веб-сервісу

Основним завданням тестування веб-сервісу є перевірка його коректної роботи в нормальних умовах використання, а також правильної реакції на випадки, що відхиляються від звичайних сценаріїв користувача. Тестування веб-сервісів має свої особливості, які відрізняють його від тестування звичайних десктопних та мобільних додатків. В таблиці 5.1 наведені ключові характеристики, які потребують тестування в веб-сервісі, з урахуванням його функціональних можливостей.

Таблиця 5.1 – Ключові характеристики для тестування в веб-сервісі

Назва характеристики	**Вимоги до реалізації у веб-сервісі**
Перевірка основних функцій	У цьому етапі тестування перевіряється, чи працюють основні функції веб-сервісу згідно з очікуваннями. Наприклад, якщо це електронний магазин, тестується можливість перегляду товарів, додавання їх у кошик, оформлення замовлення і т.д.

Продовження таблиці 5.1 – Ключові характеристики для тестування в веб-сервісі

Назва характеристики	**Вимоги до реалізації у веб-сервісі**
Обробка різних типів запитів та відповідей	Веб-сервіс повинен коректно обробляти різні типи запитів, такі як GET, POST, PUT, DELETE і т.д., а також надавати відповідні відповіді згідно зі специфікаціями. Тестування включає перевірку правильності обробки запитів та повернення правильних відповідей.
Обробка помилок	Веб-сервіс повинен коректно обробляти помилки та відображати відповідні повідомлення або статуси помилок. Тестування включає перевірку поведінки веб-сервісу при некоректних або неповних запитах, перевірка повернених статусів помилок і відповідних повідомлень.
Автентифікація та авторизація	Якщо веб-сервіс вимагає автентифікації користувачів, тестування включає перевірку правильності процесу автентифікації, перевірку ролей та прав доступу користувачів і обмежень на доступ до функцій та даних.
Робота з базами даних	Якщо веб-сервіс використовує базу даних для зберігання даних, тестування включає перевірку правильності зберігання, оновлення та видалення даних в базі даних. Тестування також може включати перевірку відповідності даних у базі даних очікуваним результатам.
Співвідношення розміру екрану і елементів інтерфейсу	Кнопки повинні мати достатні розміри, щоб їх можна було легко натиснути, при цьому уникнувши активації сусідніх елементів. Розмір елементів має бути настільки великим, щоб їх можна було знайти та взаємодіяти з ними без зайвих зусиль.

Продовження таблиці 5.1 – Ключові характеристики для тестування в веб-сервісі

Назва характеристики	**Вимоги до реалізації у веб-сервісі**
Локалізація	Мова веб-сервісу англійська. Формат дат повинен відповідати налаштуванням формату дати, встановленим на пристрої.
Безпека	Тестування веб-сервісу повинно оцінювати рівень безпеки, перевіряти захист від вразливостей, таких як введення некоректних даних, вразливості в автентифікації та авторизації, захист від атак, SQL-ін'єкцій тощо.

Зважаючи на визначені характеристики, тест-план для перевірки додатку може включати наступні види тестування:

– функціональне тестування – це процес перевірки працездатності додатку в звичайних умовах використання. Це означає перевірку правильності виконання функцій додатку, незалежно від орієнтації екрану, використання вбудованої чи спеціальної клавіатури та інших факторів.;

– тестування локалізації – це процес перевірки коректного відображення елементів інтерфейсу в залежності від мови та налаштувань мобільного пристрою. Це включає перевірку правильного відображення дати, часу та інших текстових елементів з урахуванням мовних налаштувань користувача;

– юзабіліті-тестування – це процес перевірки, наскільки візуальний інтерфейс відповідає потребам користувача з точки зору UX (користувацького досвіду). Це включає оцінку вигляду, розміру, розташування та інших параметрів елементів графічного інтерфейсу, обсягу даних на одній екранній сторінці, швидкості взаємодії з елементами, колірної гами та інших аспектів, які впливають на зручність використання додатку користувачем;

– тестування безпеки – це процес перевірки веб-сервісу на вразливості та захист від можливих атак. Це включає перевірку вразливостей, таких як SQL-ін'єкції, перехоплення сесій, недостатню автентифікацію та авторизацію, а також застосування заходів безпеки, таких як шифрування даних та захист від несанкціонованого доступу;

5.2 Розробка тест-кейсів

Тестування функціональних можливостей програми здійснюється шляхом використання методу тестування "чорного ящика". Цей підхід базується на перевірці функціональної специфікації та вимог, без доступу до внутрішньої структури коду або бази даних. На рівні звичайного користувача, на вхід подаються конкретні набори даних, а результат порівнюється з очікуваними даними. Кожен тест-кейс містить опис послідовності дій, необхідних для досягнення певної мети, а також набори тестових даних, що задовольняють вибраним критеріям покриття.

Тестові випадки (тест-кейси) були розроблені для перевірки відповідності функцій системи встановленим вимогам. Залежно від очікуваного результату, тест-кейси можуть бути позитивними або негативними:

– позитивний тест-кейс використовує лише коректні дані і перевіряє, чи виконує додаток функцію, для якої він призначений;

– негативний тест-кейс оперує як коректними, так і некоректними даними (принаймні один некоректний параметр) з метою перевірки виняткових ситуацій (спрацьовування валідаторів) та переконання в тому, що функція не виконується при спрацьовуванні валідатора.

Структура тест-кейсу складається з розділів "Action" (дія або послідовність дій, що виконуються під час тестування), "Expected Result" (очікуваний результат) та "Test Result" (реальні результати роботи функції, часто позначені як "passed/failed/blocked").

Згідно з розробленим інтерфейсом системи, користувач має можливість вводити дані, пов'язані з ключовими об'єктами програми, такими як гаманці, картки, категорії

та операції з ними. Тест-кейси для таких ситуацій включають позитивні тести (коли користувач вводить правильні дані з точки зору програми) та негативні тести (наприклад, коли користувач вводить некоректний тип даних). Для формування тестових ситуацій були використані методи функціонального тестування, зокрема розбиття на класи еквівалентності та метод кордонних умов. Позитивні тести позначені як "ОК", негативні тести - як "NOK".

В таблиці 5.2 наведено розроблені тест-кейси для функціонального тестування. При тестуванні системи усі наведені тести було пройдено із результатом passed.

Таблиця 5.2 – Тест-кейси для функціонального тестування

№ тест-кейсу	ОК/ NOK	Дії	Очікуваний результат
1	ОК	1. Пройти етап авторизації для користувача. 2. Натиснути на кнопку у правому нижньому кутку “Add New”. 3. Заповнити форму для створення лоту. 4. Створити лот натиснувши на кнопку “Create”	1. Після натискання на кнопку “Add New” відкривається форма для заповнення інформації про лот. 2. Усі поля для вводу доступні. 3. Випадаючі списки для вибору доступні. 4. Зображення відображається на екрані після його додавання. 5. Новий лот успішно додається до системи і відображається у списку лотів на головній сторінці.

Продовження таблиці 5.2 – Тест-кейси для функціонального тестування

№ тест-кейсу	ОК/ NOK	Дії	Очікуваний результат
2	ОК	1. Ввести ключове слово в поле “Search for a lot” і натиснути на значок пошуку.	1.Відображаються результати пошуку, що відповідають введеному ключовому слову.
3	ОК	1. Пройти етап авторизації для користувача. 2. Зі списку лотів обрати один та додати його до списку бажань натиснувши на кнопку “Add to WishList”.	1. Після натискання на кнопку “Add to WishList” відкривається сторінка зі списком бажань користувача. 2. З’являється значок списку бажань у правому верхньому кутку екрана та кількість речей у списку.
4	ОК	1.Пройти етап авторизації для користувача. 2.Обрати лот з умовою для обміну “Chargeless” та створити договір про обмін натиснувши на кнопку “Create a deal”. 3.Натиснути на кнопку “Complete a deal”	1.Сторінка для створення договору обміну відображається коректно з потрібною інформацією про лоти. 2.Після натискання на кнопку “Complete a deal” з’являється повідомлення про успішне створення замовлення.
5	ОК	1. Настинути на кнопку “Show Details” на одному з лотів.	1.Після натискання на кнопку відкривається сторінка з детальною інформацією про лот.

Юзабіліті тестування визначає, наскільки інтерфейс програми відповідає потребам та вимогам користувача. Цей вид тестування оцінює зручність використання

програмного продукту, включаючи його навігацію, читабельність тексту, розміщення елементів інтерфейсу, а також відповідність кольорів та стилів.

Тест-кейси для юзабіліті-тестування відображено у таблиці 5.3. При тестуванні системи усі наведені тести було пройдено із результатом passed.

Таблиця 5.3 – Юзабіліті-тестування

№ тест-кейсу	OK/ NOK	Дії	Очікуваний результат
1	OK	1. Оглянути текстові елементи інтерфейсу, такі як заголовки, кнопки, посилання	1. Текст повинен легко читатися, бути зрозумілим та відповідати діям елементів, за якими він закріплений.
2	OK	1. Оцінити розміщення елементів, таких як кнопки, поля введення, списки тощо	1. Елементи повинні бути розміщені логічно та зручно для користувача, без перенасичення або зайвих відступів
3	OK	1. Спробувати перейти до різних розділів інтерфейсу за допомогою навігаційних елементів (меню, посилання)	1. Навігація повинна бути зручною та інтуїтивно зрозумілою, користувач повинен легко знайти потрібний розділ

Тест-кейси для тестування безпеки наведені у таблиці 5.4.

Таблиця 5.4 – Тестування безпеки веб-сервісу

№ тест-кейсу	OK/ NOK	Дії	Очікуваний результат
1	NOK	1. Пройти етап авторизації для користувача. 2. Натиснути на кнопку у правому нижньому кутку “Add New”. 3. Залишити форму для заповнення пустою. 4. Створити лот натиснувши на кнопку “Create”.	1. Під полями, які мають бути обов’язково заповнені і залишились пустими, з’являється повідомлення червоного кольору про необхідність їх заповнення при настанні на кнопку “Create”.
2	NOK	1. Пройти етап авторизації для користувача. 2. Обрати лот з умовою для обміну “Exchange” та створити договір про обмін натиснувши на кнопку “Create a deal”. 3. Натиснути на кнопку “Exchange item”. 4. Залишити форму для заповнення пустою. 5. Настинути на кнопку “Submit”.	1. З’являється повідомлення про заповнення полів “Please fill in all fields”.

Продовження таблиці 5.4 – Тестування безпеки веб-сервісу

№ тест-кейсу	ОК/ NOK	Дії	Очікуваний результат
3	NOK	1.У правому кутку екрана натиснути на кнопку “Register” 2.Залишити форму для заповнення пустою. 3.Настинути на кнопку “Sign up”.	1.Під полями, які мають бути обов’язково заповнені і залишились пустими, з’являється повідомлення червоного кольору про необхідність їх заповнення при настанні на кнопку “Sign up”.

ВИСНОВКИ

1. Проведено аналіз обраної предметної області в сфері обміну непотрібними подарунками. Визначено особливості процесу та ключові потреби користувачів.

2. Проведено аналіз існуючих аналогів програмного забезпечення, яке можна використовувати для обміну речами чи подарунками на комерційній та безоплатній основі. Визначено ключові функції сервісів, їх переваги та недоліки, які враховано при розробці сервісу White Elephant.

3. Визначено основні вимоги до функцональних можливостей системи та проведено моелювання вимог з використанням діаграми прецедентів.

4. Розроблено архітектуру веб-сервісу для обміну непотрібними подарунками. Вона включає в себе моделі даних, контролери та репозиторії. Архітектура дозволяє ефективно взаємодіяти з базою даних та забезпечує швидкий доступ до необхідної інформації.

5. Розроблено програмне забезпечення для обміну непотрібними подарунками, який дозволяє користувачам додавати та переглядати оголошення про обмін непотрібними подарунками.

6. Забезпечено безпеку та захист інформації користувачів. Для забезпечення безпеки передачі даних між клієнтом і сервером використано протокол HTTPS. Цей протокол шифрує всі дані, що передаються між сервером і клієнтом, тим самим запобігаючи можливість перехоплення цих даних третьою стороною. Для аутентифікації та авторизації користувачів. використано бібліотеку Microsoft.AspNetCore.Identity, яка дозволяє реалізувати механізми аутентифікації та авторизації користувачів на основі ролей. Таким чином, доступ до різних функцій сервісу можуть мати лише користувачі з відповідними ролями. Для застосування різних рівнів доступу до даних користувачів розроблено відповідну структуру моделей, яка дозволяє відрізняти різні типи користувачів за рівнем доступу до даних. Таким чином, можна забезпечити доступ до конфіденційних даних лише

користувачам з відповідними правами. Реалізовано резервне копіювання даних. Це дозволяє запобігти втраті даних у разі виникнення непередбачуваних ситуацій, таких як крах системи або кібератака. У результаті виконання цих заходів було досягнуто високого рівня безпеки та захисту інформації користувачів веб-сервісу.

7. Розроблено тест-кейси, що покривають ключові функціональні вимоги додатку та проведено тестування програмного забезпечення.

ПЕРЕЛІК ПОСИЛАНЬ

1. Giftster. "Giftster - Wish list registry for Christmas, birthday, baby, and weddings" [Електронний ресурс] / Giftster. – 2023. – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.giftster.android.giftster&hl=en_US.

2. "Secret Santa Organizer" [Електронний ресурс] // Secret Santa Organizer. – 2023. – Режим доступу до ресурсу: https://apps.apple.com/us/app/secret-santa-organizer/id582181756.

3. "The Freecycle Network" [Електронний ресурс] // Freecycle. – 2023. – Режим доступу до ресурсу: https://www.freecycle.org/.

4. Taylor R. "Principled Design of the Modern Web Architecture" / R. Taylor, R. Fielding. // ACM. – 2002. – С. 115–150.

5. Shepeleva V. "Designing a Web Service for Organizing the Exchange of Unnecessary Gifts," / V. Shepeleva, T. Kovalenko, A. Nikiforova. // International Scientific Conference. – 2021. – С. 131–138.

6. C# Docs [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/ru-ru/dotnet/csharp/.

7. Overview of ASP.NET Core MVC [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-7.0.

8. Overview of Entity Framework Core - EF Core [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/ef/core/.

9. Razor Pages in ASP.NET Core [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-7.0&tabs=visual-studio.

10. Introducing SQL Server 2022 [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://www.microsoft.com/en-us/sql-server.

11. Identity on ASP.NET Core [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-7.0&tabs=visual-studio.

12. Richter J. CLR via C# / Jeffrey Richter.. – 896 с. – (Microsoft Press).

13. Smith J. Entity Framework Core in Action / Jon Smith., 2018. – 520 с. – (First Edition).

14. Pattankar M. Mastering ASP.NET Web API: Build powerful HTTP services and make the most of the ASP.NET Core Web API platform / M. Pattankar, M. Hurbuns., 2017. – 330 с.

ДОДАТОК А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка сервісу White Elephant для обміну непотрібними подарунками мовою C#

Виконав студент 4 курсу
групи ПД-41
Конох Владислав Віталійович
Керівник роботи
К.т.н, доц, доцент кафедри ІПЗ Золотухіна Оксана Анатоліївна

Київ – 2023

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – спрощення обміну непотрібними речами за рахунок застосування веб-додатку, створеного мовою C#
- **Об’єкт дослідження** – процес обміну непотрібними речами між користувачами
- **Предмет дослідження** – програмне забезпечення для обміну непотрібними речами

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1.Аналіз обраної предметної області в сфері обміну непотрібними подарунками;
2.Порівняння існуючих аналогів програмного забезпечення, яке можна використовувати для обміну речами чи подарунками на комерційній та безоплатній основі;
3.Визначення та моделювання вимог до програмного забезпечення;
4.Розробити архітектуру веб-сервісу для обміну непотрібними подарунками;
5.Проєктування та розробка програмного забезпечення для обміну непотрібними подарунками;
6.Провести аналіз та роробити засоби безпеки та захисту інформації користувачів;
7.Тестування розробленого додатку.

3

АНАЛІЗ АНАЛОГІВ


Показник	Giftster	Secret Santa	FreeCycle	WhiteElephant
Платформи	Android, Web	IOS, Web	Web	Web
Створення списку бажань	+	-	-	+
Обмін з сторонніми користувачами	+	+	+	+
Реєстрація та авторизація	+	+	+	+
Інтеграція з соц мережами	-	-	-	-
Система відгуків та оцінок користувачів	-	+	-	-
Додавання фото товарів	+	-	-	+

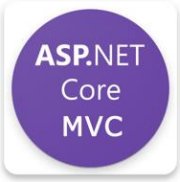
4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1. Реєстрація та авторизація користувача
2. Відображення каталогу пропозицій на обмін
3. Пошук пропозицій на обмін
4. Додавання подарунків в “Список побажань”
5. Додавання та редагування товарів
6. Створення договору про обмін
7. Перегляд історії створених договорів про обмін
8. Управління даними користувача

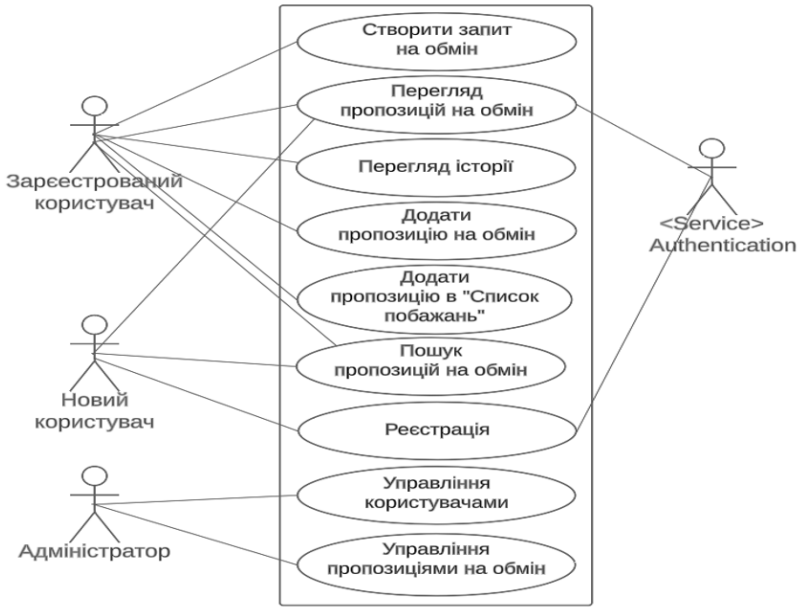
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



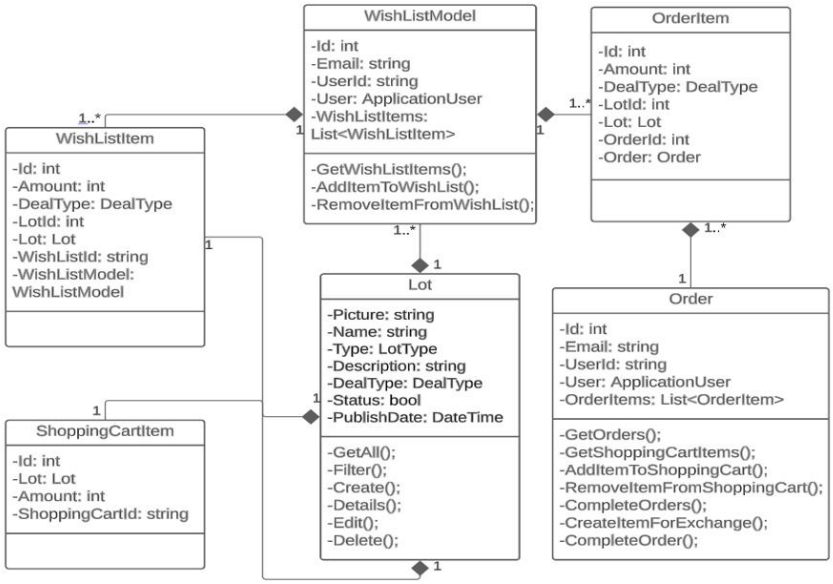
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



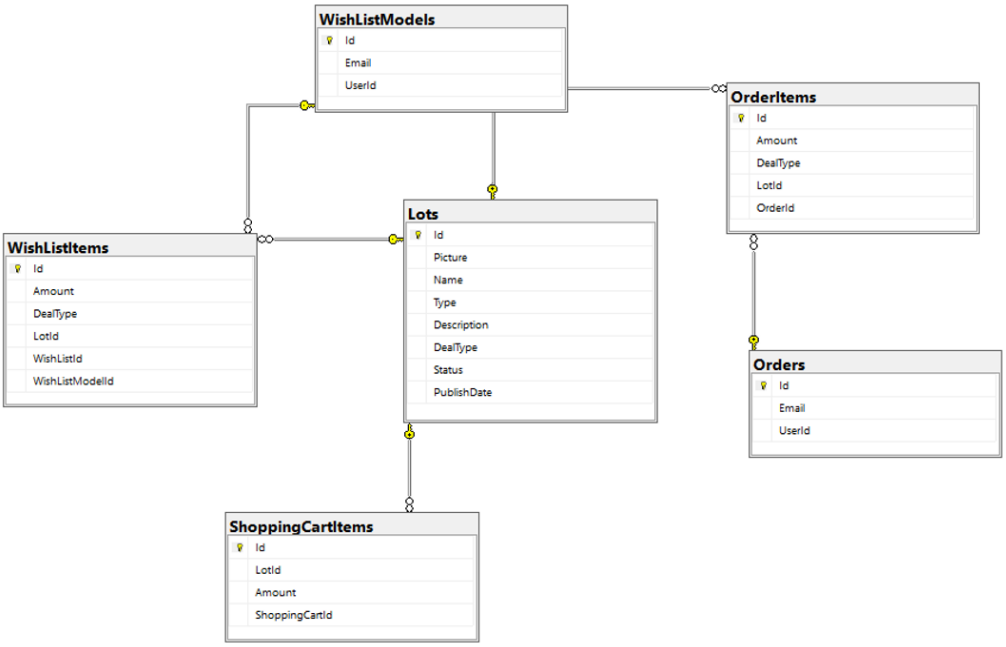
7

ДІАГРАММА КЛАСІВ



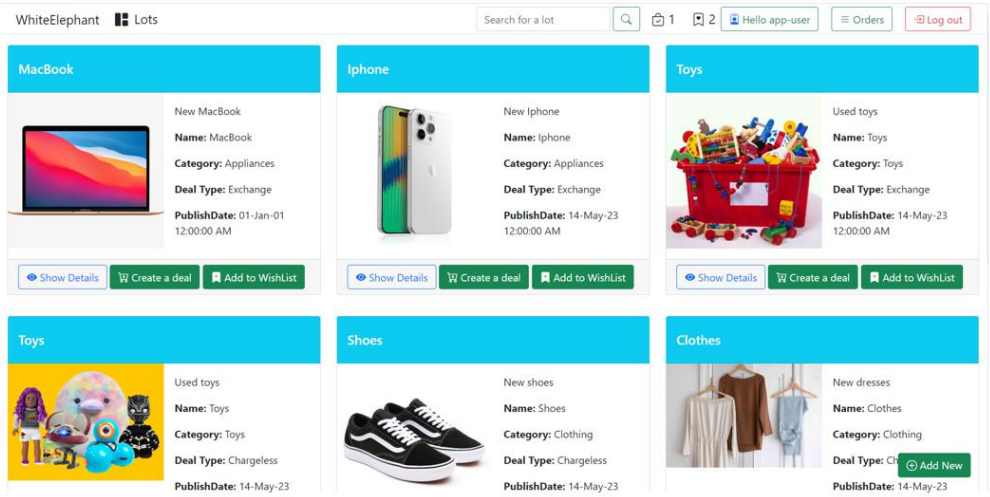
8

СХЕМА БАЗИ ДАНИХ



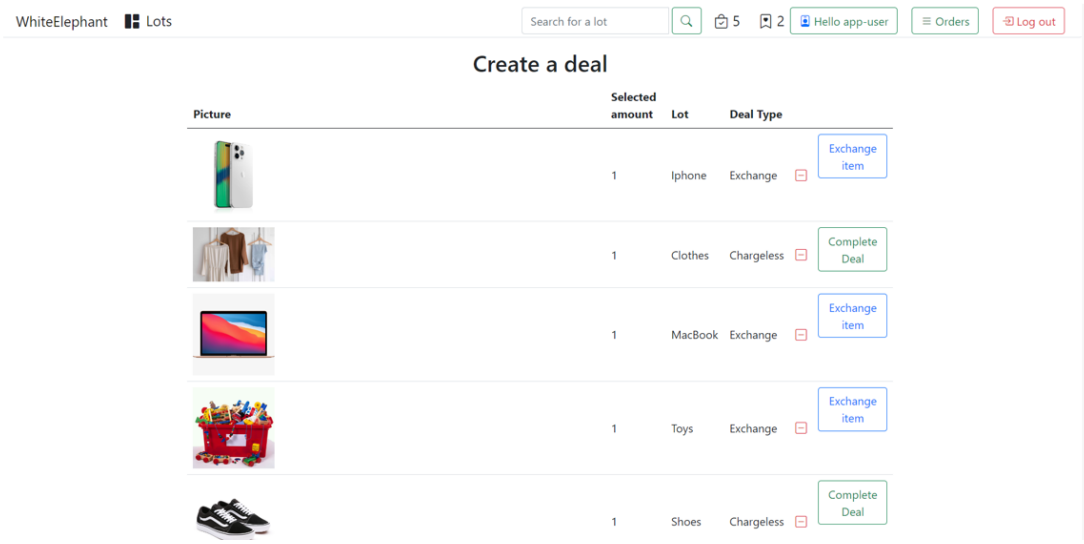
9

ГОЛОВНИЙ ЕКРАН ВЕБ-СЕРВІСУ



10

СТВОРЕННЯ ДОГОВОРУ ПРО ОБМІН



11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Конох В.В. Аналіз програмного забезпечення для обміну непотрібними подарунками / Матеріали всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інфокомунікаційних технологіях». Збірник тез. 20.04.2023, ДУТ, м. Київ — К.: ДУТ, 2023. — С. 31.
2. Конох В.В. Соціальні аспекти діджиталізації взаємодія техніки та суспільства / Матеріали міжнародної науково-практичної конференції «Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії». Збірник тез подано до друку

12

ВИСНОВКИ

1. Проведено аналіз обраної предметної області в сфері обміну непотрібними подарунками. Визначено особливості процесу та ключові потреби користувачів.
2.Проведено аналіз обраної предметної області в сфері обміну непотрібними подарунками. Визначено особливості процесу та ключові потреби користувачів.
3.Визначено основні вимоги до функцональних можливостей системи та проведено моелювання вимог з використанням діаграми прецедентів.
4. Розроблено архітектуру веб-сервісу для обміну непотрібними подарунками.
5. Розроблено програмне забезпечення для обміну непотрібними подарунками, який дозволяє користувачам додавати та переглядати оголошення про обмін непотрібними подарунками.
6.Забезпечено безпеку та захист інформації користувачів.
7.Розроблено тест-кейси, що покривають ключові функціональні вимоги додатку та проведено тестування програмного забезпечення.

13

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б
ЛІСТИНГИ ПРОГРАМИ

Program.cs

```
using ECommerce.Data;
using ECommerce.Data.Cart;
using ECommerce.Data.Services;
using ECommerce.Models;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using WishList = ECommerce.Data.WishList.WishList;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();
builder.Services.AddScoped(sc => ShoppingCart.GetShoppingCart(sc));
builder.Services.AddScoped(wl => WishList.GetWishList(wl));
builder.Services.AddSession();

builder.Services.AddControllersWithViews();

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

builder.Services.AddScoped<ILotsService,LotService>();
builder.Services.AddScoped<IOrdersService, OrdersService>();
builder.Services.AddScoped<IWishListService, WishListService>();
//Authentication and authorization
builder.Services.AddIdentity<ApplicationUser, IdentityRole>().AddEntityFrameworkStores<AppDbContext>();
builder.Services.AddMemoryCache();
builder.Services.AddSession();
builder.Services.AddAuthentication(options =>
{
    options.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;
});


var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}



app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();
app.UseSession();

//Authentication & Authorization
app.UseAuthentication();
app.UseAuthorization();

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

AppDbInitializer.Seed(app);
```

```
AppDbInitializer.SeedUsersAndRolesAsync(app).Wait();

app.Run();
```

AccountController.cs

```
using ECommerce.Data.Static;
using ECommerce.Data;
using ECommerce.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ECommerce.Data.ViewModels;

namespace ECommerce.Controllers
{
    public class AccountController : Controller
    {
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly SignInManager<ApplicationUser> _signInManager;
        private readonly AppDbContext _context;

        public AccountController(UserManager<ApplicationUser> userManager, SignInManager<ApplicationUser> signInManager,
AppDbContext context)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _context = context;
        }


        public async Task<IActionResult> Users()
        {
            var users = await _context.Users.ToListAsync();
            return View(users);
        }


        public IActionResult Login() => View(new LoginViewModel());

        [HttpPost]
        public async Task<IActionResult> Login(LoginViewModel loginViewModel)
        {
            if (!ModelState.IsValid) return View(loginViewModel);

            var user = await _userManager.FindByEmailAsync(loginViewModel.EmailAddress);
            if (user != null)
            {
                var passwordCheck = await _userManager.CheckPasswordAsync(user, loginViewModel.Password);
                if (passwordCheck)
                {
                    var result = await _signInManager.PasswordSignInAsync(user, loginViewModel.Password, false, false);
                    if (result.Succeeded)
                    {
                        return RedirectToAction("Index", "Lots");
                    }
                }
                TempData["Error"] = "Wrong credentials. Please, try again!";
                return View(loginViewModel);
            }

            TempData["Error"] = "Wrong credentials. Please, try again!";
            return View(loginViewModel);
        }


        public IActionResult Register() => View(new RegisterViewModel());

        [HttpPost]
```

```
        public async Task<IActionResult> Register(RegisterViewModel registerViewModel)
        {
            if (!ModelState.IsValid) return View(registerViewModel);

            var user = await _userManager.FindByEmailAsync(registerViewModel.EmailAddress);
            if (user != null)
            {
                TempData["Error"] = "This email address is already in use";
                return View(registerViewModel);
            }

            var newUser = new ApplicationUser()
            {
                FullName = registerViewModel.FullName,
                Email = registerViewModel.EmailAddress,
                UserName = registerViewModel.EmailAddress
            };
            var newUserResponse = await _userManager.CreateAsync(newUser, registerViewModel.Password);

            if (newUserResponse.Succeeded)
                await _userManager.AddToRoleAsync(newUser, UserRoles.User);

            return View("RegisterCompleted");
        }

        [HttpPost]
        public async Task<IActionResult> Logout()
        {
            await _signInManager.SignOutAsync();
            return RedirectToAction("Index", "Lots");
        }

        public IActionResult AccessDenied(string ReturnUrl)
        {
            return View();
        }

    }
}
```

LotsController.cs

```
using ECommerce.Data.Cart;
using ECommerce.Data.Services;
using ECommerce.Models;
using Microsoft.AspNetCore.Mvc;

namespace ECommerce.Controllers
{
    public class LotsController : Controller
    {
        private readonly ILotsService _service;

        public LotsController(ILotsService service)
        {
            _service= service;
        }

        public async Task<IActionResult> Index()
        {
            return View(await _service.GetAll());
        }

        public async Task<IActionResult> Filter(string searchString)
        {
            var allLots = await _service.GetAll();

            if (!string.IsNullOrEmpty(searchString))
```

```
        {
            var filteredResult = allLots.Where(n => n.Name.Contains(searchString) || n.Description.Contains(searchString)).ToList();
            return View("Index", filteredResult);
        }
        return View("Index", allLots);
    }

    public async Task<IActionResult> Create()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Create([Bind("Picture,Name,Type,Description,DealType")]Lot lot)
    {
        if (!ModelState.IsValid)
        {
            return View(lot);
        }
        await _service.Add(lot);
        return RedirectToAction(nameof(Index));
    }

    public async Task<IActionResult> Details(int id)
    {
        var lotDetails = await _service.GetById(id);

        if (lotDetails == null) return View("NotFound");
        return View(lotDetails);
    }

    public async Task<IActionResult> Edit(int id)
    {
        var lotDetails = await _service.GetById(id);
        if (lotDetails == null) return View("NotFound");
        return View(lotDetails);
    }

    [HttpPost]
    public async Task<IActionResult> Edit(int id, [Bind("Id,Picture,Name,Type,Description,DealType")] Lot lot)
    {
        if (!ModelState.IsValid)
        {
            return View(lot);
        }
        await _service.Update(lot, id);
        return RedirectToAction(nameof(Index));
    }

    public async Task<IActionResult> Delete(int id)
    {
        var lotDetails = await _service.GetById(id);
        if (lotDetails == null) return View("NotFound");
        return View(lotDetails);
    }

    [HttpPost, ActionName("Delete")]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var lotDetails = await _service.GetById(id);
        if (lotDetails == null) return View("NotFound");

        await _service.Delete(id);
        return RedirectToAction(nameof(Index));
    }
  }
}
```

OrdersController.cs

```
using ECommerce.Data.Cart;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;
using ECommerce.Data.Services;
using ECommerce.Data.ViewModels;

namespace ECommerce.Controllers
{
    //[Authorize]

    public class OrdersController : Controller
    {
        private readonly ILotsService _lotsService;
        private readonly ShoppingCart _shoppingCart;
        private readonly IOrdersService _ordersService;

        public OrdersController(ILotsService lotsService, ShoppingCart shoppingCart, IOrdersService ordersService)
        {
            _lotsService = lotsService;
            _shoppingCart = shoppingCart;
            _ordersService = ordersService;
        }

        public async Task<IActionResult> Index()
        {
            string userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
            string userRole = User.FindFirstValue(ClaimTypes.Role);

            var orders = await _ordersService.GetOrdersByUserIdAndRoleAsync(userId, userRole);
            return View(orders);
        }

        public IActionResult ShoppingCart()
        {
            var items = _shoppingCart.GetShoppingCartItems();
            _shoppingCart.ShoppingCartItems = items;

            var response = new ShoppingCartViewModel()
            {
                ShoppingCart = _shoppingCart
            };

            return View(response);
        }

        public async Task<IActionResult> AddItemToShoppingCart(int id)
        {
            var item = await _lotsService.GetById(id);

            if (item != null)
            {
                //error
                _shoppingCart.AddItemToCart(item);
            }

            return RedirectToAction(nameof(ShoppingCart));
        }

        public async Task<IActionResult> RemoveItemFromShoppingCart(int id)
        {
            var item = await _lotsService.GetById(id);

            if (item != null)
            {
                _shoppingCart.RemoveItemFromCart(item);
```

```
            }

            return RedirectToAction(nameof(ShoppingCart));
        }

        public async Task<IActionResult> CompleteOrders()
        {
            var items = _shoppingCart.GetShoppingCartItems();
            string userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
            string userEmailAddress = User.FindFirstValue(ClaimTypes.Email);

            await _ordersService.StoreOrdersAsync(items, userId, userEmailAddress);
            await _shoppingCart.ClearShoppingCartAsync();

            return View("OrderCompleted");
        }

        public async Task<IActionResult> CreateItemForExchange(int id)
        {
            var item = await _lotsService.GetById(id);

            var cartItem = _shoppingCart.GetShoppingCartItem(id);
            string userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
            string userEmailAddress = User.FindFirstValue(ClaimTypes.Email);

            await _ordersService.StoreOrderAsync(cartItem, userId, userEmailAddress);

            if (item != null)
            {
                _shoppingCart.RemoveItemFromCart(item);
            }
            return View("OrderCompletedWithExchangeItem");
        }

        public async Task<IActionResult> CompleteOrder(int id)
        {
            var item = await _lotsService.GetById(id);

            var cartItem = _shoppingCart.GetShoppingCartItem(id);
            string userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
            string userEmailAddress = User.FindFirstValue(ClaimTypes.Email);

            await _ordersService.StoreOrderAsync(cartItem, userId, userEmailAddress);

            if (item != null)
            {
                _shoppingCart.RemoveItemFromCart(item);
            }
            return View("OrderCompleted");
        }
    }
}
```

WishListController.cs

```
using ECommerce.Data.Cart;
using ECommerce.Data.Services;
using ECommerce.Data.ViewModels;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;
using ECommerce.Data.WishList;

namespace ECommerce.Controllers
{
    public class WishListController : Controller
    {
        private readonly ILotsService _lotsService;
        private readonly IWishListService _wishListService;
        private readonly WishList _wishList;
```

```
        public WishListController(ILotsService lotsService, IWishListService wishListService, WishList wishList)
        {
            _lotsService = lotsService;
            _wishListService = wishListService;
            _wishList = wishList;
        }

        public async Task<IActionResult> Index()
        {
            string userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

            var items = await _wishListService.GetWishListByUserIdAndRoleAsync(userId);
            return View(items);
        }

        public IActionResult WishList()
        {
            var items = _wishList.GetWishListItems();
            _wishList.WishListItems = items;

            var response = new WishListViewModel()
            {
                WishList = _wishList
            };

            return View(response);
        }

        public async Task<IActionResult> AddItemToWishList(int id)
        {
            var item = await _lotsService.GetById(id);

            if (item != null)
            {
                _wishList.AddItemToWishList(item);
            }
            return RedirectToAction(nameof(WishList));
        }

        public async Task<IActionResult> RemoveItemFromWishList(int id)
        {
            var item = await _lotsService.GetById(id);

            if (item != null)
            {
                _wishList.RemoveItemFromWishList(item);
            }
            return RedirectToAction(nameof(WishList));
        }
    }
}
```

EntityBaseRepository.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.ChangeTracking;

namespace ECommerce.Data.Base
{
    public class EntityBaseRepository<T> : IEntityBaseRepository<T> where T : class, IEntityBase, new()
    {
        private readonly AppDbContext _context;
        public EntityBaseRepository(AppDbContext context)
        {
            _context = context;
        }
        public async Task Add(T entity)
```

```
        {
            await _context.Set<T>().AddAsync(entity);
            await _context.SaveChangesAsync();
        }

        public async Task Delete(int id)
        {
            var entity = await _context.Set<T>().FirstOrDefaultAsync(l => l.Id == id);
            EntityEntry enetityEntry = _context.Entry<T>(entity);
            enetityEntry.State = EntityState.Deleted;
            await _context.SaveChangesAsync();
        }

        public async Task<IEnumerable<T>> GetAll() => await _context.Set<T>().ToListAsync();

        public async Task<T> GetById(int id) => await _context.Set<T>().FirstOrDefaultAsync(l => l.Id == id);

        public async Task Update(T entity, int id)
        {
            EntityEntry enetityEntry = _context.Entry<T>(entity);
            enetityEntry.State= EntityState.Modified;
            await _context.SaveChangesAsync();
        }
    }
}
```

IEntityBase.cs

```
namespace ECommerce.Data.Base
{
    public interface IEntityBase
    {
        int Id { get; set; }

    }
}
```

IEntityBaseRepository.cs

```
using ECommerce.Models;

namespace ECommerce.Data.Base
{
    public interface IEntityBaseRepository<T> where T : class, IEntityBase, new()
    {
        Task<IEnumerable<T>> GetAll();
        Task<T> GetById(int id);
        Task Add(T entity);
        Task Update(T entity, int id);
        Task Delete(int id);
    }
}
```

ShoppingCart.cs

```
using ECommerce.Models;
using Microsoft.EntityFrameworkCore;

namespace ECommerce.Data.Cart
{
    public class ShoppingCart
    {
        public AppDbContext _context { get; set; }

        public string ShoppingCartId { get; set; }
        public List<ShoppingCartItem> ShoppingCartItems { get; set; }

        public ShoppingCart(AppDbContext context)
```

```
        {
            _context = context;
        }

        public static ShoppingCart GetShoppingCart(IServiceProvider services)
        {
            ISession session = services.GetRequiredService<IHttpContextAccessor>()?.HttpContext.Session;
            var context = services.GetService<AppDbContext>();

            string cartId = session.GetString("CartId") ?? Guid.NewGuid().ToString();
            session.SetString("CartId", cartId);

            return new ShoppingCart(context) { ShoppingCartId = cartId };
        }

        public void AddItemToCart(Lot lot)
        {
            //error
            var shoppingCartItem = _context.ShoppingCartItems.FirstOrDefault(n => n.Lot.Id == lot.Id && n.ShoppingCartId ==
ShoppingCartId);

            if (shoppingCartItem == null)
            {
                shoppingCartItem = new ShoppingCartItem()
                {
                    ShoppingCartId = ShoppingCartId,
                    Lot = lot,
                    Amount = 1
                };

                _context.ShoppingCartItems.Add(shoppingCartItem);
            }
            _context.SaveChanges();
        }

        public void RemoveItemFromCart(Lot lot)
        {
            var shoppingCartItem = _context.ShoppingCartItems.FirstOrDefault(n => n.Lot.Id == lot.Id && n.ShoppingCartId ==
ShoppingCartId);

            if (shoppingCartItem != null)
            {
                _context.ShoppingCartItems.Remove(shoppingCartItem);
            }
            _context.SaveChanges();
        }

        public List<ShoppingCartItem> GetShoppingCartItems()
        {
            return ShoppingCartItems ?? (ShoppingCartItems = _context.ShoppingCartItems.Where(n => n.ShoppingCartId ==
ShoppingCartId).Include(n => n.Lot).ToList());
        }

        public async Task ClearShoppingCartAsync()
        {
            var items = await _context.ShoppingCartItems.Where(n => n.ShoppingCartId == ShoppingCartId).ToListAsync();
            _context.ShoppingCartItems.RemoveRange(items);
            await _context.SaveChangesAsync();
        }

        public ShoppingCartItem GetShoppingCartItem(int lotId)
        {
            return _context.ShoppingCartItems
                .SingleOrDefault(n => n.ShoppingCartId == ShoppingCartId && n.Lot.Id == lotId);
        }
    }
}
```

DealType.cs

```
namespace ECommerce.Data.Enums
{
    public enum DealType
    {
        Exchange,
        Chargeless
    }
}
```

LotType.cs

```
namespace ECommerce.Data.Enums
{
    public enum LotType
    {
        Toys = 1,
        Clothing,
        Rust,
        Appliances

    }
}
```

IOrdersService.cs

```
using ECommerce.Models;

namespace ECommerce.Data.Services
{
    public interface IOrdersService
    {
        Task StoreOrdersAsync(List<ShoppingCartItem> items, string userId, string userEmailAddress);
        Task<List<Order>> GetOrdersByUserIdAndRoleAsync(string userId, string userRole);
        Task StoreOrderAsync(ShoppingCartItem item, string userId, string userEmailAddress);
    }
}
```

IWishListService.cs

```
using ECommerce.Models;

namespace ECommerce.Data.Services;

public interface IWishListService
{
    Task<List<WishListModel>> GetWishListByUserIdAndRoleAsync(string userId);
    Task StoreWishListAsync(List<WishListItem> items, string userId, string userEmailAddress);
}
```

LotService.cs

```
using ECommerce.Data.Base;
using ECommerce.Models;
using Microsoft.EntityFrameworkCore;

namespace ECommerce.Data.Services
{
    public class LotService : EntityBaseRepository<Lot>
    {
        public LotService(AppDbContext context) : base(context) { }
    }
}
```

OrdersService.cs

```
using ECommerce.Models;
```

```
using Microsoft.EntityFrameworkCore;

namespace ECommerce.Data.Services
{
    public class OrdersService : IOrdersService
    {
        private readonly AppDbContext _context;
        public OrdersService(AppDbContext context)
        {
            _context = context;
        }

        public async Task<List<Order>> GetOrdersByUserIdAndRoleAsync(string userId, string userRole)
        {
            var orders = await _context.Orders
                .Include(n => n.OrderItems)
                .ThenInclude(n => n.Lot)
                .Include(n => n.User)
                .ToListAsync();

            if (userRole != "Admin")
            {
                orders = orders.Where(n => n.UserId == userId).ToList();
            }

            return orders;
        }

        public async Task StoreOrdersAsync(List<ShoppingCartItem> items, string userId, string userEmailAddress)
        {
            var order = new Order()
            {
                UserId = userId,
                Email = userEmailAddress
            };
            await _context.Orders.AddAsync(order);
            await _context.SaveChangesAsync();

            foreach (var item in items)
            {
                var orderItem = new OrderItem()
                {
                    Amount = item.Amount,
                    LotId = item.Lot.Id,
                    OrderId = order.Id,
                    DealType = item.Lot.DealType
                };
                await _context.OrderItems.AddAsync(orderItem);
            }
            await _context.SaveChangesAsync();
        }
        public async Task StoreOrderAsync(ShoppingCartItem item, string userId, string userEmailAddress)
        {
            var order = new Order()
            {
                UserId = userId,
                Email = userEmailAddress
            };
            await _context.Orders.AddAsync(order);
            await _context.SaveChangesAsync();

                var orderItem = new OrderItem()
                {
                    Amount = item.Amount,
                    LotId = item.Lot.Id,
                    OrderId = order.Id,
                    DealType = item.Lot.DealType
                };
```

```
            await _context.OrderItems.AddAsync(orderItem);

            await _context.SaveChangesAsync();
        }
    }
}
```

WishListService.cs

```
using ECommerce.Models;
using Microsoft.EntityFrameworkCore;

namespace ECommerce.Data.Services
{
    public class WishListService : IWishListService
    {
        public AppDbContext _context { get; set; }

        public WishListService(AppDbContext context)
        {
            _context = context;
        }

        public async Task<List<WishListModel>> GetWishListByUserIdAndRoleAsync(string userId)
        {
            var wishList = await _context.WishListModels
                .Include(n => n.WishListItems)
                .ThenInclude(n => n.Lot)
                .Include(n => n.User)
                .Where(n => n.UserId == userId)
                .ToListAsync();

            return wishList;
        }

        public async Task StoreWishListAsync(List<WishListItem> items, string userId, string userEmailAddress)
        {
            var wishList = new WishListModel()
            {
                UserId = userId,
                Email = userEmailAddress
            };
            await _context.WishListModels.AddAsync(wishList);
            await _context.SaveChangesAsync();

            foreach (var item in items)
            {
                var wishListItem = new WishListItem()
                {
                    Amount = item.Amount,
                    LotId = item.Lot.Id,
                    WishListId = item.WishListId,
                    DealType = item.Lot.DealType
                };
                await _context.WishListItems.AddAsync(wishListItem);
            }
            await _context.SaveChangesAsync();
        }
    }
}
```

UserRoles.cs

```
namespace ECommerce.Data.Static
{
    public class UserRoles
```

```
    {
        public const string Admin = "Admin";
        public const string User = "User";
    }
}
```

ShoppingCartSummary.cs

```
using ECommerce.Data.Cart;
using Microsoft.AspNetCore.Mvc;

namespace ECommerce.Data.ViewComponents
{
    public class ShoppingCartSummary : ViewComponent
    {
        private readonly ShoppingCart _shoppingCart;

        public ShoppingCartSummary(ShoppingCart shoppingCart)
        {
            _shoppingCart = shoppingCart;
        }
        public IViewComponentResult Invoke()
        {
            var items = _shoppingCart.GetShoppingCartItems();

            return View(items.Count);
        }
    }
}
```

WishListVC.cs

```
using Microsoft.AspNetCore.Mvc;

namespace ECommerce.Data.ViewComponents
{
    public class WishListVC : ViewComponent
    {
        private readonly WishList.WishList _wishList;

        public WishListVC(WishList.WishList wishList)
        {
            _wishList = wishList;
        }
        public IViewComponentResult Invoke()
        {
            var items = _wishList.GetWishListItems();

            return View(items.Count);
        }
    }
}
```

ErrorViewModel.cs

```
namespace ECommerce.Data.ViewModels
{
    public class ErrorViewModel
    {
        public string? RequestId { get; set; }

        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}
```

LoginViewModel.cs

```
using System.ComponentModel.DataAnnotations;

namespace ECommerce.Data.ViewModels
{
    public class LoginViewModel
    {
        [Display(Name = "Email address")]
        [Required(ErrorMessage = "Email address is required")]
        public string EmailAddress { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}
```

RegisterViewModel.cs

```
using System.ComponentModel.DataAnnotations;

namespace ECommerce.Data.ViewModels
{
    public class RegisterViewModel
    {
        [Display(Name = "Full name")]
        [Required(ErrorMessage = "Full name is required")]
        public string FullName { get; set; }

        [Display(Name = "Email address")]
        [Required(ErrorMessage = "Email address is required")]
        public string EmailAddress { get; set; }

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

        [Display(Name = "Confirm password")]
        [Required(ErrorMessage = "Confirm password is required")]
        [DataType(DataType.Password)]
        [Compare("Password", ErrorMessage = "Passwords do not match")]
        public string ConfirmPassword { get; set; }
    }
}
```

ShoppingCartViewModel.cs

```
using ECommerce.Data.Cart;

namespace ECommerce.Data.ViewModels
{
    public class ShoppingCartViewModel
    {
        public ShoppingCart ShoppingCart { get; set; }
    }
}
```

WishListViewModel.cs

```
namespace ECommerce.Data.ViewModels
{
    public class WishListViewModel
    {
        public WishList.WishList WishList { get; set; }
    }
}
```

WishList.cs

```
using ECommerce.Data.Services;
using ECommerce.Models;
using Microsoft.EntityFrameworkCore;

namespace ECommerce.Data.WishList
{
    public class WishList
    {
        public AppDbContext _context { get; set; }

        public string WishListId { get; set; }
        public List<WishListItem> WishListItems { get; set; }

        public WishList(AppDbContext context)
        {
            _context = context;
        }

        public static WishList GetWishList(IServiceProvider services)
        {
            ISession session = services.GetRequiredService<IHttpContextAccessor>()?.HttpContext.Session;
            var context = services.GetService<AppDbContext>();

            string wishListId = session.GetString("WishListId") ?? Guid.NewGuid().ToString();
            session.SetString("WishListId", wishListId);

            return new WishList(context) { WishListId = wishListId };
        }

        public void AddItemToWishList(Lot lot)
        {
            var wishListItem = _context.WishListItems.FirstOrDefault(n => n.Lot.Id == lot.Id && n.WishListId == WishListId);

            if (wishListItem == null)
            {
                wishListItem = new WishListItem()
                {
                    WishListId = WishListId,
                    Lot = lot,
                    Amount = 1
                };

                _context.WishListItems.Add(wishListItem);
            }
            _context.SaveChanges();
        }

        public void RemoveItemFromWishList(Lot lot)
        {
            var wishListItem = _context.WishListItems.FirstOrDefault(n => n.Lot.Id == lot.Id && n.WishListId == WishListId);

            if (wishListItem != null)
            {
                _context.WishListItems.Remove(wishListItem);
            }
            _context.SaveChanges();
        }

        public List<WishListItem> GetWishListItems()
        {
            return WishListItems ?? (WishListItems = _context.WishListItems.Where(n => n.WishListId == WishListId).Include(n =>
n.Lot).ToList());
        }

    }
}
```

AppDbContext.cs

```
using ECommerce.Models;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

namespace ECommerce.Data
{
    public class AppDbContext : IdentityDbContext<ApplicationUser>
    {
        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
        {

        }

        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);
        }

        public DbSet<Lot> Lots { get; set; }
        public DbSet<Order> Orders { get; set; }
        public DbSet<OrderItem> OrderItems { get; set; }
        public DbSet<ShoppingCartItem> ShoppingCartItems { get; set; }
        public DbSet<WishListItem> WishListItems { get; set; }
        public DbSet<WishListModel> WishListModels{ get; set; }
    }
}
```

ApplicationUser.cs

```
using Microsoft.AspNetCore.Identity;
using System.ComponentModel.DataAnnotations;

namespace ECommerce.Models
{
    public class ApplicationUser : IdentityUser
    {
        [Display(Name = "Full name")]
        public string FullName { get; set; }
    }
}
```

Lot.cs

```
using ECommerce.Data.Base;
using ECommerce.Data.Enums;
using System.ComponentModel.DataAnnotations;

namespace ECommerce.Models
{
    public class Lot:IEntityBase
    {
        [Key]
        public int Id { get; set; }

        [Display(Name = "Picture URL")]
        [Required(ErrorMessage ="Picture is required")]
        public string Picture { get; set; }

        [Display(Name = "Name")]
        [Required(ErrorMessage = "Name is required")]
        [StringLength(50,MinimumLength =4, ErrorMessage ="Name must be between 4 and 50 characters")]
        public string Name { get; set; }
```

```
        [Display(Name = "Type")]
        [Required(ErrorMessage = "Type is required")]
        public LotType Type { get; set;}

        [Display(Name = "Description")]
        [Required(ErrorMessage = "Description is required")]
        public string Description { get; set;}

        [Display(Name = "Deal Type")]
        [Required(ErrorMessage = "Deal Type is required")]
        public DealType DealType { get; set; }

        [Display(Name = "Status")]
        public bool Status { get; set; }

        [Display(Name = "PublishDate")]
        public DateTime PublishDate { get; set; }

    }
}
```

Order.cs

```
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;

namespace ECommerce.Models
{
    public class Order
    {
        [Key]
        public int Id { get; set; }

        public string Email { get; set; }

        public string UserId { get; set; }
        [ForeignKey(nameof(UserId))]

        public ApplicationUser User { get; set; }

        public List<OrderItem> OrderItems { get; set; }
    }
}
```

OrderItem.cs

```
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;
using ECommerce.Data.Enums;

namespace ECommerce.Models
{
    public class OrderItem
    {
        [Key]
        public int Id { get; set; }

        public int Amount { get; set; }
        public DealType DealType { get; set; }

        public int LotId { get; set; }
        [ForeignKey("LotId")]

        public Lot Lot { get; set; }

        public int OrderId { get; set; }
        [ForeignKey("OrderId")]
```

```
        public Order Order { get; set; }
    }
}
```

ShoppingCartItem.cs

```
using System.ComponentModel.DataAnnotations;

namespace ECommerce.Models
{
    public class ShoppingCartItem
    {

        [Key]
        public int Id { get; set; }

        public Lot Lot { get; set; }
        public int Amount { get; set; }

        public string ShoppingCartId { get; set; }
    }
}
```

WishListItem.cs

```
using ECommerce.Data.Enums;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ECommerce.Models
{
    public class WishListItem
    {
        [Key]
        public int Id { get; set; }

        public int Amount { get; set; }
        public DealType DealType { get; set; }

        public int LotId { get; set; }
        [ForeignKey("LotId")]

        public Lot Lot { get; set; }

        public string WishListId { get; set; }

        public WishListModel WishListModel { get; set; }
    }
}
```

WishListModel.cs

```
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;

namespace ECommerce.Models
{
    public class WishListModel
    {
        [Key]
        public string Id { get; set; }

        public string Email { get; set; }

        public string UserId { get; set; }
        [ForeignKey(nameof(UserId))]
```

```
        public ApplicationUser User { get; set; }

        public List<WishListItem> WishListItems { get; set; }
    }
}
```

AccessDenied.cshtml

```
<div class="row">
    <div class="col-md-6 offset-3 alert alert-danger text-center">
        <h2>You are not allowed to access this page</h2>
        <p>Please, click the button below to be redirected to the home page.</p>
        <hr />
        <p>
            <a asp-controller="Lots" asp-action="Index" class="btn btn-outline-success" >Home Page</a>
        </p>
    </div>
</div>
```

Login.cshtml

```
@using ECommerce.Data.ViewModels;
@model LoginViewModel;

@{
    ViewData["Title"] = "Log in to your account";
}

<div class="row">
    <div class="col-md-6 offset-3">
        <p>
            <h4>Log in to your account</h4>
        </p>

        @if(TempData["Error"] != null)
        {
            <div class="col-md-12 alert alert-danger">
                <span><b>Sorry!</b> - @TempData["Error"] </span>
            </div>
        }

        <div class="row">
            <div class="col-md-8 offset-2">
                <form asp-action="Login">
                    <div asp-validation-summary="ModelOnly" class="text-danger"></div>

                    <div class="form-group">
                        <label asp-for="EmailAddress" class="control-label"></label>
                        <input asp-for="EmailAddress" class="form-control" />
                        <span asp-validation-for="EmailAddress" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="Password" class="control-label"></label>
                        <input asp-for="Password" class="form-control" />
                        <span asp-validation-for="Password" class="text-danger"></span>
                    </div>
                    <span style="padding: 5px;"></span>
                    <div class="form-group">
                        <input class="btn btn-outline-success float-right" type="submit" value="Log in" />
                        <a class="btn btn-outline-secondary" asp-controller="Lots" asp-action="Index">Cancel</a>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>
```

Register.cshtml

```
@using ECommerce.Data.ViewModels;
@model RegisterViewModel;

@{
    ViewData["Title"] = "Sign up for a new account";
}

<div class="row">
    <div class="col-md-6 offset-3">
        <p>
            <h4>Sign up for a new account</h4>
        </p>

        @if (TempData["Error"] != null)
        {
            <div class="col-md-12 alert alert-danger">
                <span><b>Sorry!</b> - @TempData["Error"] </span>
            </div>
        }

        <div class="row">
            <div class="col-md-8 offset-2">
                <form asp-action="Register">
                    <div asp-validation-summary="ModelOnly" class="text-danger"></div>

                    <div class="form-group">
                        <label asp-for="FullName" class="control-label"></label>
                        <input asp-for="FullName" class="form-control" />
                        <span asp-validation-for="FullName" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="EmailAddress" class="control-label"></label>
                        <input asp-for="EmailAddress" class="form-control" />
                        <span asp-validation-for="EmailAddress" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="Password" class="control-label"></label>
                        <input asp-for="Password" class="form-control" />
                        <span asp-validation-for="Password" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="ConfirmPassword" class="control-label"></label>
                        <input asp-for="ConfirmPassword" class="form-control" />
                        <span asp-validation-for="ConfirmPassword" class="text-danger"></span>
                    </div>
                    <span style="padding: 5px;"></span>
                    <div class="form-group">
                        <input class="btn btn-outline-success float-right" type="submit" value="Sign up" />
                        <a class="btn btn-outline-secondary" asp-controller="Lots" asp-action="Index">Cancel</a>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>
```

RegisterCompleted.cshtml

```
<div class="row">
    <div class="col-md-6 offset-3 alert alert-success text-center">
        <h2>Your account was created successfully</h2>
```

```
        <p>You can now log in to your account using your credentials.</p>
        <hr />
        <p>Thank you!</p>
    </div>
</div>
```

Users.cshtml

```
@model IEnumerable<ApplicationUser>

@{
    ViewData["Title"] = "List of all users";
}

<div class="row">
    <div class="col-md-6 offset-3">
        <p>
            <h4>List of all users</h4>
        </p>

        <table class="table">
            <thead>
            <tr class="text-center">
                <th>@Html.DisplayNameFor(model => model.FullName)</th>
                <th>@Html.DisplayNameFor(model => model.UserName)</th>
                <th>@Html.DisplayNameFor(model => model.Email)</th>
            </tr>
            </thead>
            <tbody>
            @foreach (var item in Model)
            {
                <tr>
                    <td class="align-middle">
                        @Html.DisplayFor(modelItem => item.FullName)
                    </td>
                    <td class="align-middle">
                        @Html.DisplayFor(modelItem => item.UserName)
                    </td>
                    <td class="align-middle">
                        @Html.DisplayFor(modelItem => item.Email)
                    </td>
                </tr>
            }
            </tbody>

        </table>

    </div>
</div>
```

Create.cshtml

```
@model Lot

@{
    ViewData["Title"] = "Add a new Lot";
}

<div class="row-text">
    <div class="col-md-8 offset-2">
        <p>
            <h1>Add a new Lot</h1>
        </p>
        <div class="row">
            <div class="col-md-8 offset-2">
```

```
            <form asp-action="Create">
                <div asp-validation-summary="ModelOnly" class="text-danger"></div>
                <div class="col-md-4 offset-4 text-center">
                    <img width="100%" id="PicturePreview" />
                </div>
            <div class="form-group">
                <label asp-for="Picture" class="control-label"></label>
                <input asp-for="Picture" class="form-control"/>
                <span asp-validation-for="Picture" class ="text-danger"></span>
            </div>
        <div class="form-group">
            <label asp-for="Name" clas s= "control-label"></label>
            <input asp-for="Name" class="form-control" />
            <span asp-validation-for="Name" class="text-danger"></span>
        </div>
        <div class="form-group">
                <label asp-for="Type" class="control-label"></label>
                    <select asp-for="Type" class="form-control">
                        <option value="">Select Type</option>
                        <option value="Toys">Toys</option>
                        <option value="Clothing">Clothing</option>
                        <option value="Rust">Rust</option>
                        <option value="Appliances">Appliances</option>
                    </select>
                <span asp-validation-for="Type" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Description" class="control-label"></label>
                <input asp-for="Description" class="form-control" />
                <span asp-validation-for="Description" class="text-danger"></span>
            </div>
                <div class="form-group">
                    <label asp-for="DealType" class="control-label"></label>
                    <select asp-for="DealType" class="form-control">
                        <option value="">Select Deal Type</option>
                        <option value="Exchange">Exchange</option>
                        <option value="Chargeless">Chargeless</option>
                    </select>
                    <span asp-validation-for="DealType" class="text-danger"></span>
                </div>
                <span style="padding: 5px;"></span>
                <div class="form-group">
                <input type="submit" value="Create" class="btn btn-outline-success float-end" />
                <a class="btn btn-outline-secondary" asp-action="Index">Show All</a>
                </div>
            </form>
        </div>
    </div>
</div>

@section Scripts{
    <script>
        $(document).ready(function (){
            var output = document.getElementById('PicturePreview')
            output.src = $("#Picture").val();
        })

        $("#Picture").on("change", function(){
            var output = document.getElementById('PicturePreview')
            output.src = $(this).val();
        })
    </script>
}
```

Delete.cshtml

```
@model Lot

@{
    ViewData["Title"] = "Delete a Lot";
}

<div class="row-text">
    <div class="col-md-8 offset-2">
        <p>
            <h1>Delete confirmation for @Model.Name</h1>
        </p>
        <div class="row">
            <div class="col-md-8 offset-2">
                    <div class="form-group text-center">
                        <img class="border-info rounded-circle" style="max-width: 150px" id="PicturePreview" />
                    </div>
                    <div class="form-group">
                        <label asp-for="Picture" class="control-label"></label>
                        <input asp-for="Picture" class="form-control" readonly/>
                        <span asp-validation-for="Picture" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="Name" clas s="control-label"></label>
                        <input asp-for="Name" class="form-control" readonly/>
                        <span asp-validation-for="Name" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="Type" class="control-label"></label>
                        <input asp-for="Type" class="form-control" readonly />
                        <span asp-validation-for="Type" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="Description" class="control-label"></label>
                        <input asp-for="Description" class="form-control" readonly />
                        <span asp-validation-for="Description" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="DealType" class="control-label"></label>
                    <input asp-for="DealType" class="form-control" readonly />
                    <span asp-validation-for="DealType" class="text-danger"></span>
                    </div>
                <span style="padding: 5px;"></span>
                <div class="form-group">
                    <div class="row">
                        <div class="col">
                            <form asp-action="Delete" style="padding-right: 25px">
                                <input type="hidden" asp-for="Id"/>
                                <input type="submit" value="Confirm" class="btn btn-danger float-right"/>
                            </form>
                        </div>
                        <div class="col-auto">
                            <a class="btn btn-outline-secondary" asp-action="Index">Show All</a>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>

@section Scripts{
    <script>
        $(document).ready(function () {
            var output = document.getElementById('PicturePreview')
            output.src = $("Picture").val();
        })
    </script>
}
```

Details.cshtml

```
@model Lot

@{
    ViewData["Title"] = "Add a new Lot";
}

<div class="row-text">
    <div class="col-md-8 offset-2">
        <p>
            <h1>Details for @Model.Name</h1>
        </p>
        <div class="row">
            <div class="col-md-8 offset-2">
                <div class="col-md-4 offset-4 text-center">
                    <img width="100%" id="PicturePreview" />
                </div>
                    <div class="form-group">
                        <label asp-for="Picture" class="control-label"></label>
                        <input asp-for="Picture" class="form-control" readonly/>
                        <span asp-validation-for="Picture" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="Name" clas s="control-label"></label>
                        <input asp-for="Name" class="form-control" readonly/>
                        <span asp-validation-for="Name" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="Type" class="control-label"></label>
                        <input asp-for="Type" class="form-control" readonly />
                        <span asp-validation-for="Type" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="Description" class="control-label"></label>
                        <input asp-for="Description" class="form-control" readonly />
                        <span asp-validation-for="Description" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="DealType" class="control-label"></label>
                        <input asp-for="DealType" class="form-control" readonly />
                        <span asp-validation-for="DealType" class="text-danger"></span>
                    </div>
                    <span style="padding: 5px;"></span>
            </div>
        </div>
    </div>
</div>

@section Scripts{
    <script>
        $(document).ready(function () {
            var output = document.getElementById('PicturePreview')
            output.src = $("#Picture").val();
        })
    </script>
}
```

Edit.cshtml

```
@model Lot

@{
    ViewData["Title"] = "Update a new Lot";
}
```

```
<div class="row-text">
   <div class="col-md-8 offset-2">
      <p>
         <h1>Update a new Lot</h1>
      </p>
      <div class="row">
         <div class="col-md-8 offset-2">
            <form asp-action="Edit">
               <div asp-validation-summary="ModelOnly" class="text-danger"></div>
               <div class="col-md-4 offset-4 text-center">
                  <img width="100%" id="PicturePreview" />
               </div>
            <div class="form-group">
               <label asp-for="Picture" class="control-label"></label>
               <input asp-for="Picture" class="form-control"/>
               <span asp-validation-for="Picture" class ="text-danger"></span>
            </div>
         <div class="form-group">
            <label asp-for="Name" clas s= "control-label"></label>
            <input asp-for="Name" class="form-control" />
            <span asp-validation-for="Name" class="text-danger"></span>
         </div>
               <div class="form-group">
                  <label asp-for="Type" class="control-label"></label>
                  <select asp-for="Type" class="form-control">
                     <option value="">Select Type</option>
                     <option value="Toys">Toys</option>
                     <option value="Clothing">Clothing</option>
                     <option value="Rust">Rust</option>
                     <option value="Appliances">Appliances</option>
                  </select>
                  <span asp-validation-for="Type" class="text-danger"></span>
               </div>
            <div class="form-group">
               <label asp-for="Description" class="control-label"></label>
               <input asp-for="Description" class="form-control" />
               <span asp-validation-for="Description" class="text-danger"></span>
            </div>
               <div class="form-group">
                  <label asp-for="DealType" class="control-label"></label>
                  <select asp-for="DealType" class="form-control">
                     <option value="">Select Deal Type</option>
                     <option value="Exchange">Exchange</option>
                     <option value="Chargeless">Chargeless</option>
                  </select>
                  <span asp-validation-for="DealType" class="text-danger"></span>
               </div>
               <span style="padding: 5px;"></span>
               <div class="form-group">
               <input type="submit" value="Update" class="btn btn-outline-success" />
               <a class="btn btn-outline-secondary float-end" asp-action="Index">Show All</a>
               </div>
            </form>
         </div>
      </div>
   </div>
</div>

@section Scripts{
   <script>
      $(document).ready(function (){
         var output = document.getElementById('PicturePreview')
         output.src = $("#Picture").val();
      });
      $("#Picture").on("change", function(){
         var output = document.getElementById('PicturePreview')
         output.src = $(this).val();
```

```
    });
  </script>
}

Index.cshtml

@model IEnumerable<Lot>

@{
  ViewData["Title"] = "List of Lots";
}

<div class="row">
  @foreach (var item in Model)
  {
    <div class="col-md-4 col-xs-6 border-primary mb-3">
      <div class="card mb-3" style="max-width: 540px;">
        <div class="row g-0">
          <div class="col-md-12">
            <div class="card-header text-white bg-info">
              <p class="card-text">
                <h5 class="card-title">
                  @item.Name
                  @if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
                  {
                    <a class="text-white float-right" asp-action="Edit" asp-route-id="@item.Id"><i class="bi bi-pencil-
square"></i></a>
                  }
                </h5>
              </p>
            </div>
          </div>
          <div class="col-md-6">
            <img src="@item.Picture" width="100%" alt="@item.Name">
          </div>
          <div class="col-md-6">
            <div class="card-body">
            <p class="card-text">@item.Description</p>
            <p class="card-text"><b>Name: </b>@item.Name</p>
            <p class="card-text"><b>Category: </b>@item.Type</p>
              <p class="card-text"><b>Deal Type: </b>@item.DealType</p>
              <p class="card-text"><b>PublishDate: </b>@item.PublishDate</p>
              <p class="card-text ">
            </div>
          </div>
          <div class="col-md-12">
            <div class="card-footer ">
              <p class="card-text">
                <a class="btn btn-outline-primary float-right" asp-action="Details" asp-route-id="@item.Id">
                  <i class="bi bi-eye-fill"></i> Show Details
                </a>
                @if (User.Identity.IsAuthenticated && User.IsInRole("User"))
              {
                  <a class="btn btn-success text-white"
                asp-controller="Orders"
                asp-action="AddItemToShoppingCart"
                asp-route-id="@item.Id">
                      <i class="bi bi-cart-plus"></i> Create a deal
                  </a>
                  <a class="btn btn-success text-white"
                asp-controller="WishList"
                asp-action="AddItemToWishList"
                asp-route-id="@item.Id">
                      <i class="bi bi-bookmark-plus-fill"></i> Add to WishList
                  </a>
              }
              @if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
                {
                  <a class="btn btn-outline-danger" asp-controller="Lots" asp-action="Delete" asp-route-id="@item.Id">
```

```
                        <i class="bi bi-eye-fill"></i>
                        Delete
                    </a>
                }
            </p>
        </div>
    </div>
</div>
</div>
</div>
}
</div>

@await Html.PartialAsync("_CreateItem", "Lots")
```

Index.cshtml

```
@using System.Security.Claims
@model List<Order>

@{
    ViewData["Title"] = "All orders";
}

<div class="row">
    <div class="col-md-8 offset-2">
        <p>
            <h4>List of all your orders</h4>
        </p>

        <table class="table">
            <thead>
            <tr>
                <th>Order ID</th>
                <th>Items</th>
                <th>Amount</th>
                <th>Name</th>
                <th>DealType</th>
                @if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
                {
                    <th>User</th>
                }
            </tr>
            </thead>
            <tbody>
            @foreach (var order in Model)
            {
                <tr>
                    @if (User.Identity.IsAuthenticated && (User.IsInRole("Admin") || User.Identity.Name == order.User.UserName))
                    {
                        @foreach (var item in order.OrderItems)
                        {
                            <td class="align-middle">@order.Id</td>
                            <td class="align-middle">
                                <img src="@item.Lot.Picture" width="20%" alt="@item.Lot.Name">
                            </td>
                            <td class="align-middle">@item.Amount</td>
                            <td class="align-middle">@item.Lot.Name</td>
                            <td class="align-middle">@item.Lot.DealType</td>

                            @if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
                            {
                                <td class="align-middle"> @order.User.FullName </td>
                            }
                        }
                    }

                </tr>
            }
```

```
            </tbody>
        </table>

    </div>
</div>
```

OrderCompleted.cshtml

```
<div class="row">
    <div class="col-md-6 offset-3 alert alert-success text-center">
        <h2>Order completed successfully</h2>
        <p>You can check all your orders in the Orders section of your profile</p>
        <hr />
        <p>Thank you!</p>
    </div>
</div>
```

OrderCompletedWithExchangeItem.cshtml

```
<div class="row">
    <div class="col-md-6 offset-3 alert alert-success text-center">
        <h2>Your deal request was sent, wait for confirmation for exchange</h2>
        <p>You can check all your orders in the Orders section of your profile</p>
        <hr />
        <p>Thank you!</p>
    </div>
</div>
```

ShoppingCart.cshtml

```
@using ECommerce.Data.Enums
@model ECommerce.Data.ViewModels.ShoppingCartViewModel

@{
    ViewData["Title"] = "Create a deal";
}

<div class="row">
    <div class="col-md-8 offset-2">
        <div class="text-center">
            <h2>Create a deal</h2>
        </div>

        <table class="table">
            <thead>
            <tr>
                <th>Picture</th>
                <th>Selected amount</th>
                <th>Lot</th>
                <th>Deal Type</th>
                <th></th>
                <th></th>
            </tr>
            </thead>
            <tbody>
            @foreach (var item in Model.ShoppingCart.ShoppingCartItems)
            {
                <tr>
                    <td class="align-middle">
                        <img src="@item.Lot.Picture" width="20%" alt="@item.Lot.Name">
                    </td>
                    <td class="align-middle">@item.Amount</td>
                    <td class="align-middle">@item.Lot.Name</td>
                    <td class="align-middle">@item.Lot.DealType<td)>
                    <td class="align-middle">
```

```
                    <a class="text-danger" asp-controller="Orders" asp-action="RemoveItemFromShoppingCart" asp-route-
id="@item.Lot.Id">
                        <i class="bi bi-dash-square"></i>
                    </a>
                </td>
                @if (@item.Lot.DealType == DealType.Exchange)
                {
                    <td class="text-center">
                        <button class="btn btn-outline-primary" type="button" id="exchange-item-button" aria-haspopup="true" aria-
expanded="false" onclick="toggleExchangeItemForm()">
                            Exchange item
                        </button>
                        <div id="exchange-item-form" class="input-group mt-3" style="display: none;">
                            <input id="picture" type="file" class="form-control mx-auto" placeholder="Picture" aria-label="Picture" style="width:
200px;">
                            <input id="description" type="text" class="form-control mt-3" placeholder="Description" aria-label="Description"
style="width: 200px;">
                            <select id="condition" type="text" class="form-control mt-3" placeholder="Condition" aria-label="Condition"
style="width: 200px;">
                                <option value="New">New</option>
                                <option value="Used">Used</option>
                            </select>
                            <span style="padding: 5px;"></span>
                            <a class="btn btn-outline-success mt-3" onclick="return checkInputs()" asp-controller="Orders" asp-
action="CreateItemForExchange" asp-route-id="@item.Lot.Id" style="width: 200px;">Submit</a>
                        </div>
                    </td>
                }
                @if (@item.Lot.DealType == DealType.Chargeless)
                {
                    <td class="text-center">
                        <a class="btn btn-outline-success" asp-controller="Orders" asp-action="CompleteOrder" asp-route-
id="@item.Lot.Id">Complete Deal</a>
                    </td>
                }
            </tr>
        }
        </tbody>
    </table>
    <div class="text-center">
        <div class="btn-group">
            <a class="btn btn-outline-success" asp-controller="Lots" asp-action="Index">Add more items</a>
        </div>
    </div>
  </div>
</div>
<script>
function toggleExchangeItemForm() {
    var form = document.getElementById("exchange-item-form");
    if (form.style.display === "none") {
        form.style.display = "block";
    } else {
        form.style.display = "none";
    }
}
function checkInputs() {
  var picture = document.getElementById('picture').value;
  var description = document.getElementById('description').value;
  var condition = document.getElementById('condition').value;

  if (!picture || !description || !condition) {
    alert('Please fill in all fields');
    return false;
  } else {
    return true;
  }
}
</script>
```

Default.cshtml

```
@model int;

@if (Model > 0)
{
    <li class="nav-item">
        <a class="nav-link text-dark" asp-controller="Orders" asp-action="ShoppingCart">
            <i class="bi bi-bag-check"></i> @Model
        </a>
    </li>
}
```

Default.cshtml

```
@model int;

@if (Model > 0)
{
        <li class="nav-item">
            <a class="nav-link text-dark" asp-controller="WishList" asp-action="WishList">
            <i class="bi bi-bookmark-heart"></i> @Model
            </a>
        </li>
}
```

_CreateItem.cshtml

```
@model string;

@if (User.Identity.IsAuthenticated)
{
    <div style="position: fixed; right: 25px; bottom: 25px;" class="text-white">
        <a asp-controller="@Model" asp-action="Create" class="btn btn-success">
            <i class="bi bi-plus-circle"></i> Add New
        </a>
    </div>
}
```

_Identity.cshtml

```
@using Microsoft.AspNetCore.Identity
@using Microsoft.AspNetCore.Mvc.TagHelpers
@inject UserManager<ApplicationUser> UserManager;

@if (!User.Identity.IsAuthenticated)
{
    <a class="btn btn-outline-success my-2 my-sm-0" asp-controller="Account" asp-action="Login">
        <i class="bi bi-box-arrow-in-right"></i> Log in
    </a>
    <span style="padding: 5px;"></span>

    <a class="btn btn-outline-primary my-2 my-sm-0" asp-controller="Account" asp-action="Register">
        <i class="bi bi-person-plus"></i> Register
    </a>
}
else
{
    @*User profile section*@
    <div class="label">
        <label class="btn btn-outline-success" id="profile"
               aria-expanded="false">
            <i class="bi bi-person-badge"></i> Hello @UserManager.GetUserName(User)
        </label>
    </div>
    <span style="padding: 5px;"></span>
```

```
    <a class="btn btn-outline-success my-2 my-sm-0" asp-controller="Orders" asp-action="Index">
        <i class="bi bi-list"></i> Orders
    </a>
    <span style="padding: 5px;"></span>
    @if (User.IsInRole("Admin"))
    {
        <a class="btn btn-outline-primary my-2 my-sm-0" asp-controller="Account" asp-action="Users">
            <i class="bi bi-people"></i> Users
        </a>
    }
    <span style="padding: 5px;"></span>
    <form asp-controller="Account" asp-action="Logout">
        <button class="btn btn-outline-danger my-2 my-sm-0">
            <i class="bi bi-box-arrow-in-right"></i> Log out
        </button>
    </form>
}
```

_Layout.cshtml

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - WhiteElephant</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.4.1/font/bootstrap-icons.css">
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggler navbar navbar-light bg-white border-bottom box-shadow mb-3">
            <div class="container-fluid">
                <a class="navbar-brand" asp-controller="Lots" asp-action="Index">WhiteElephant</a>
                <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-
controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Lots" asp-action="Index">
                                <i class="bi bi-grid-1x2-fill"></i>
                                <span style="padding: 2px;"></span>
                                Lots
                            </a>
                        </li>
                    </ul>
                    <form class="form-inline my-2 my-lg-0" asp-controller="Lots" asp-action="Filter">
                        <div class="form-group">
                            <div class="input-group">
                                <input name="searchString" type="text" class="form-control mr-sm-2" placeholder="Search for a lot" aria-
label="Search">
                                <span style="padding: 3px;"></span>
                                <div class="input-group-append">
                                    <button class="btn btn-outline-success my-2 my-lg-0" type="submit"><i class="bi bi-search"></i></button>
                                </div>
                            </div>
                        </div>
                    </form>
                    <span style="padding: 5px;"></span>
                    <ul class="navbar-nav">
                        @await Component.InvokeAsync("ShoppingCartSummary")
                    </ul>

                    <span style="padding: 5px;"></span>
                    <ul class="navbar-nav">
```

```
                    @await Component.InvokeAsync("WishListVC")
                </ul>

                <partial name="_Identity.cshtml" />
            </div>
        </div>
    </nav>
</header>
<div class="container-fluid">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>

</div>

<footer class="border-top footer text-muted">
    <div class="container">
        &copy; 2023 - ECommerce
    </div>
</footer>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.min.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>

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

_Layout.cshtml.css

```
/* Please see documentation at https://docs.microsoft.com/aspnet/core/client-side/bundling-and-minification
for details on configuring this project to bundle and minify static web assets. */

a.navbar-brand {
  white-space: normal;
  text-align: center;
  word-break: break-all;
}

a {
  color: #0077cc;
}

.btn-primary {
  color: #fff;
  background-color: #1b6ec2;
  border-color: #1861ac;
}

.nav-pills .nav-link.active, .nav-pills .show > .nav-link {
  color: #fff;
  background-color: #1b6ec2;
  border-color: #1861ac;
}

.border-top {
  border-top: 1px solid #e5e5e5;
}
.border-bottom {
  border-bottom: 1px solid #e5e5e5;
}

.box-shadow {
  box-shadow: 0 .25rem .75rem rgba(0, 0, 0, .05);
}

button.accept-policy {
  font-size: 1rem;
```

```
  line-height: inherit;
}

.footer {
  position: absolute;
  bottom: 0;
  width: 100%;
  white-space: nowrap;
  line-height: 60px;
}
```

_ValidationSctiptsPartial.cshtml

```
<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
<script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>
```

Error.cshtml

```
@model ErrorViewModel
@{
    ViewData["Title"] = "Error";
}

<h1 class="text-danger">Error.</h1>
<h2 class="text-danger">An error occurred while processing your request.</h2>

@if (Model.ShowRequestId)
{
    <p>
        <strong>Request ID:</strong> <code>@Model.RequestId</code>
    </p>
}

<h3>Development Mode</h3>
<p>
    Swapping to <strong>Development</strong> environment will display more detailed information about the error that occurred.
</p>
<p>
    <strong>The Development environment shouldn't be enabled for deployed applications.</strong>
    It can result in displaying sensitive information from exceptions to end users.
    For local debugging, enable the <strong>Development</strong> environment by setting the
<strong>ASPNETCORE_ENVIRONMENT</strong> environment variable to <strong>Development</strong>
    and restarting the app.
</p>
```

NotFound.cshtml

```
<div class="row">
    <div class="col-md-6 offset-3">
        <h1 class="text-black">404 Error</h1>
        <h1 class="text-info">Sorry, we can’t seem to find what you’re looking for.</h1>
        <hr />
        <a class="btn btn-outline-success" asp-controller="Lots" asp-action="Index">Home Page</a>
    </div>
</div>
```

Index.cshtml

```
@model List<WishListModel>

@{
    ViewData["Title"] = "WishList";
}

<div class="row">
    <div class="col-md-8 offset-2">
        <div class="text-center">
```

```
        </div>
        <div class="text-center">
            <h1>wish list</h1>
        </div>
    </div>
</div>
```

WishList.cshtml

```
@model ECommerce.Data.ViewModels.WishListViewModel

@{
    ViewData["Title"] = "WishList";
}

<div class="row">
    <div class="col-md-8 offset-2">
        <div class="text-center">
            <h2>Wish List</h2>
        </div>

        <table class="table">
            <thead>
                <tr>
                    <th>Picture</th>
                    <th>Selected amount</th>
                    <th>Lot</th>
                    <th>Deal Type</th>
                    <th></th>
                    <th></th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Model.WishList.WishListItems)
                {
                    <tr>
                        <td class="align-middle">
                            <img src="@item.Lot.Picture" width="20%" alt="@item.Lot.Name">
                        </td>
                        <td class="align-middle">@item.Amount</td>
                        <td class="align-middle">@item.Lot.Name</td>
                        <td class="align-middle">@item.Lot.DealType</td>
                        <td class="align-middle">
                            <a class="text-danger" asp-controller="WishList" asp-action="RemoveItemFromWishList" asp-route-
id="@item.Lot.Id">
                                <i class="bi bi-dash-square"></i>
                            </a>
                        </td>
                        <td class="align-middle">
                            <a class="btn btn-success text-white"
                               asp-controller="Orders"
                               asp-action="AddItemToShoppingCart"
                               asp-route-id="@item.Lot.Id">
                                <i class="bi bi-cart-plus"></i> Create a deal
                            </a>
                        </td>
                    </tr>
                }
            </tbody>
        </table>
        <div class="text-center">
            <div class="btn-group">
                <a class="btn btn-outline-success" asp-controller="Lots" asp-action="Index">Add more items</a>
            </div>
        </div>
    </div>
</div>
```

_ViewImports.cshtml

```
sing ECommerce
@using ECommerce.Models
@using ECommerce.Data.ViewModels
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

_ViewStart.cshtml

```
@{
    Layout = "_Layout";
}
```