МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

**КУРСОВА РОБОТА**

з курсу: «Об’єктно-орієнтоване програмування»

на тему: «Розроблення програмного забезпечення з використанням

об’єктно-орієнтованої парадигми»

Виконав: студент (ка) 2 курсу групи 621п

Спеціальність 121 – Інженерія програмного забезпечення

(код та найменування)

Бажанов В.О.

(прізвище й ініціали студента(ки))

Керівник: Лучшев П.О.

(прізвище й ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_ Лучшев П.О.

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_ Дем’яненко В.А.

(підпис) (прізвище та ініціали)

Харків – 2024

**ТИПОВЕ ЗАВДАННЯ НА КУРСОВУ РОБОТУ**

Варіант 11. Інтернет-магазин

|  |  |  |
| --- | --- | --- |
| **Критерій оцінювання** | **Бали** | **Штрафи** |
| **Покрокове виконання курсової роботи:** | | |
| Розділ 1(крайній термін – **3 тиждень**) | 0..15 | -3 |
| Розділ 2(крайній термін – **6 тиждень**) | 0..15 | -3 |
| Розділ 3(крайній термін – **9 тиждень**) | 0..15 | -3 |
| Розділ 4(крайній термін – **12 тиждень**) | 0..15 | -3 |
| Розділ 5(крайній термін – **13 тиждень**) | 0..5 | -1 |
| Оформлення пояснювальної записки (термін – **14 тиждень**) | 0..3 | -1 |
| **Підготовка до захисту курсової роботи:** | | |
| доповідь з презентацією (термін – **15 тиждень**) | 0..1 |  |
| відеоролик з демонстрацією роботи програми  (термін – **15 тиждень**) | 0..1 |  |
| **Захист курсової роботи:** | | |
| доповідь з презентацією | 0..5 |  |
| демонстрація роботи програми і github-репозиторію | 0..5 |  |
| відповіді на питання | 0..5 |  |
| використання\*:   1. інкапсуляції 2. статичних членів класів 3. інтерфейсів 4. абстрактних класів 5. спадкування 6. поліморфізму 7. .NET-delegates і events 8. сереалізації/десереалізації об’єктів 9. Collections.Generic 10. LINQ   \* мають бути представленні на захисті курсової роботи у презентації | 0..15 |  |
| **Всього за курсову роботу:** | **0..100** | **0..14** |

**Зміст**

[СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ 2](#_Toc166323718)

[ВСТУП 2](#_Toc166323719)

[1 ФУНКЦІОНАЛЬНІ ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 2](#_Toc166323720)

[1.1 Аналіз функціональності програмних аналогів 2](#_Toc166323721)

[1.2 Побудова Mind Map для заданої предметної області 2](#_Toc166323722)

[1.3 Виділення ролей користувачів і формулювання функціональних вимог 2](#_Toc166323723)

[1.4 Проєктування інтерфейсу користувача 2](#_Toc166323724)

[1.5 Розроблення функціональних тестів 2](#_Toc166323725)

[1.5.1 Функціональні тести для ролі користувача «Адміністратор» 2](#_Toc166323726)

[1.5.2 Функціональні тести для ролі користувача «Гість» 2](#_Toc166323727)

[2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ 2](#_Toc166323728)

[2.1 Виділення і опис класів предметної області 2](#_Toc166323729)

[2.2 Встановлення зв’язків між класами 2](#_Toc166323730)

[3 ПРОГРАМНА РЕАЛІЗАЦІЯ КЛАСІВ ПРЕДМЕТНОЇ ОБЛАСТІ І ЇХ ТЕСТУВАННЯ 2](#_Toc166323731)

[3.1 Структура проєкту з реалізацією класів предметної області 2](#_Toc166323732)

[3.2 Реалізація інтерфейсів і каркасів класів предметної області 2](#_Toc166323733)

[3.3 Розроблення unit-тестів для класів предметної області 2](#_Toc166323734)

[3.4 Повна реалізація класів предметної області 2](#_Toc166323735)

[3.5 Результати unit-тестування класів предметної області 2](#_Toc166323736)

[4 ПРОЄКТУВАННЯ І ПРОГРАМНА РЕАЛІЗАЦІЯ КЛАСІВ ІНТЕРФЕЙСУ КОРИСТУВАЧА 2](#_Toc166323737)

[4.1 Структура проєкту з реалізацією класів інтерфейсу користувача 2](#_Toc166323738)

[4.2 Виділення класів для реалізації інтерфейсу користувача 2](#_Toc166323739)

[4.3 Програмна реалізація класів інтерфейсу користувача 2](#_Toc166323740)

[5 ФУНКЦІОНАЛЬНЕ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 2](#_Toc166323741)

[5.1 Функціональне тестування для ролі користувача «Адміністратор» 2](#_Toc166323742)

[5.2 Функціональне тестування для ролі користувача «Зареєстрований користувач» 2](#_Toc166323743)

[5.3 Функціональне тестування для ролі користувача «Гість» 2](#_Toc166323744)

[ВИСНОВКИ 2](#_Toc166323745)

[ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАНЬ 2](#_Toc166323746)

[ДОДАТОК А. Лістинг класів предметної області 2](#_Toc166323747)

[ДОДАТОК Б. Лістинг класів інтерфейсу користувача 2](#_Toc166323748)

[ДОДАТОК В. Лістинг класів unit-тестів 2](#_Toc166323749)

[ДОДАТОК Г. Назва додатку 2](#_Toc166323750)

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

|  |  |  |
| --- | --- | --- |
| CRUD | – | (*англ*. Create Read Update Delete) основні функції управління даними «створення, читання, оновлення і вилучення». |
| ООП | – | об’єктно-орієнтоване програмування |
| ORM | – | (*англ*. Object-Relational Mapping) підхід до програмування, який дозволяє взаємодіяти з базою даних у формі об'єктів програмної мови програмування, замість написання запитів мовою структурованих запитів бази даних |
| MVC | – | (*англ*. Model View Controller) це архітектурний шаблон, що розділяє програму на три взаємодіючі компоненти. |

# ВСТУП

У сучасному світі інтернет-магазини стали невід'ємною частиною повсякденного життя, надаючи споживачам можливість купувати товари та послуги з будь-якої точки світу. З ростом популярності електронної комерції, інтернет-магазини стали зручним і ефективним способом здійснення покупок для різних груп людей.

Інтернет-магазини пропонують безліч переваг, таких як відсутність черг, зручність здійснення покупок у будь-який час доби, широкий асортимент товарів і можливість порівняння цін. Це забезпечує покупцям максимально комфортний та швидкий процес вибору і покупки товарів.

Потенційні покупці інтернет-магазинів - це люди, які цінують свій час та комфорт, бажаючи отримати необхідні товари без необхідності відвідувати фізичні магазини. Це можуть бути молоді професіонали, які мають насичений графік, або сім'ї, які шукають найкращі пропозиції та знижки.

Серед інтернет-магазинів продуктової діяльності в Україні можна виділити такі популярні платформи як "Сільпо", "АТБ" та "Metro", які пропонують широкий вибір товарів.

Під час роботи над курсовою роботою, було використано такий стек-технологій, як: ASP.NET Core MVC - фреймворк для побудови веб-додатків на платформі .NET Core; Entity Framework - ORM, який надає доступ до бази даних у формі об'єктів, дозволяючи взаємодіяти з базою даних через об'єктно-орієнтований підхід; База даних - я використав SQL Server для зберігання даних мого проекту.

# ФУНКЦІОНАЛЬНІ ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Аналіз функціональності програмних аналогів

1. Silpo [[2](#_ПЕРЕЛІК_ДЖЕРЕЛ_ТА)]

Сервіс дозволяє передивитись товари, додати які цікавлять до кошика, після чого оформити оплату. Присутня можливість авторизації на сайту.

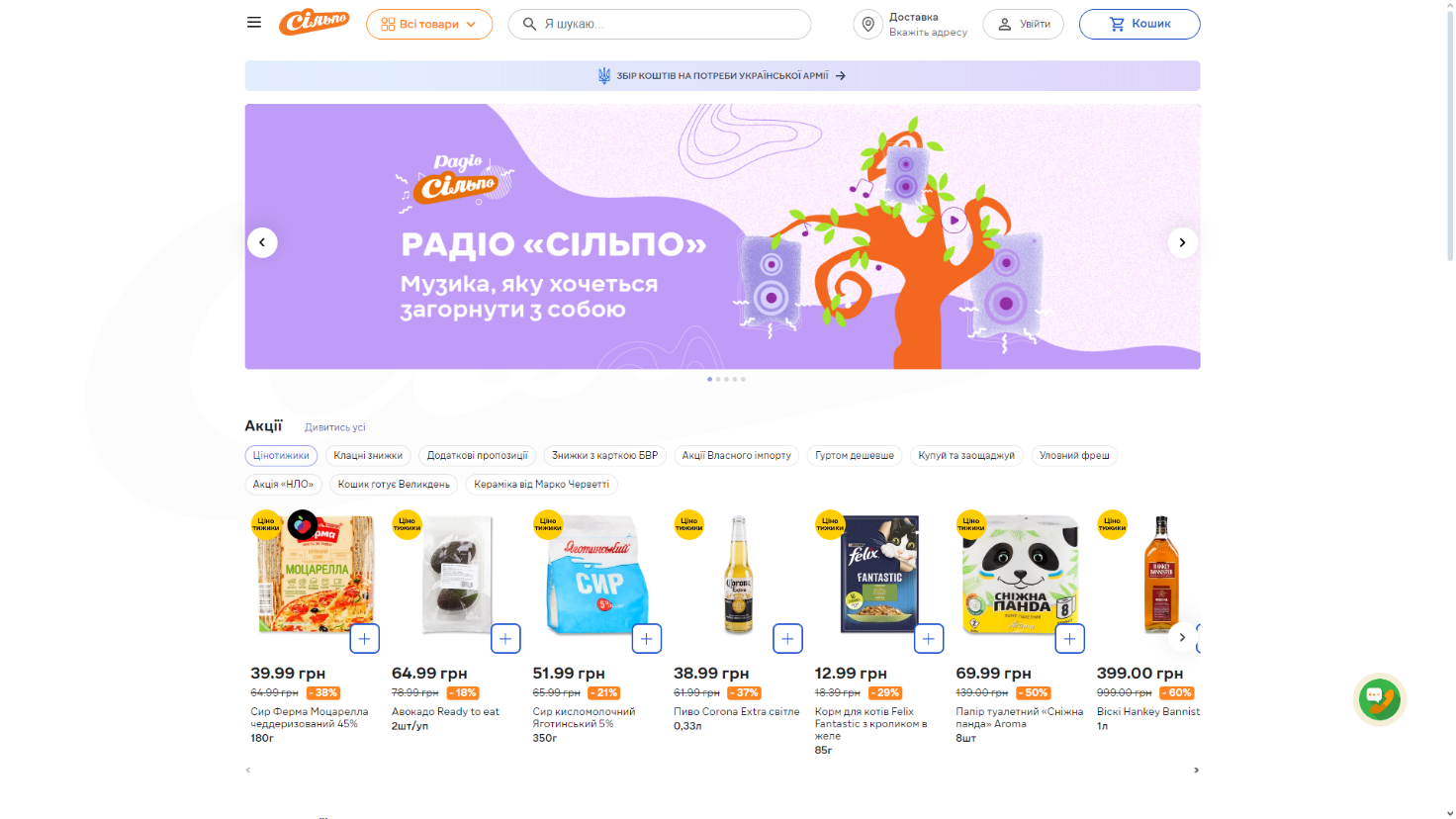


Рисунок 1.1 – Головна сторінка сайту «Silpo»

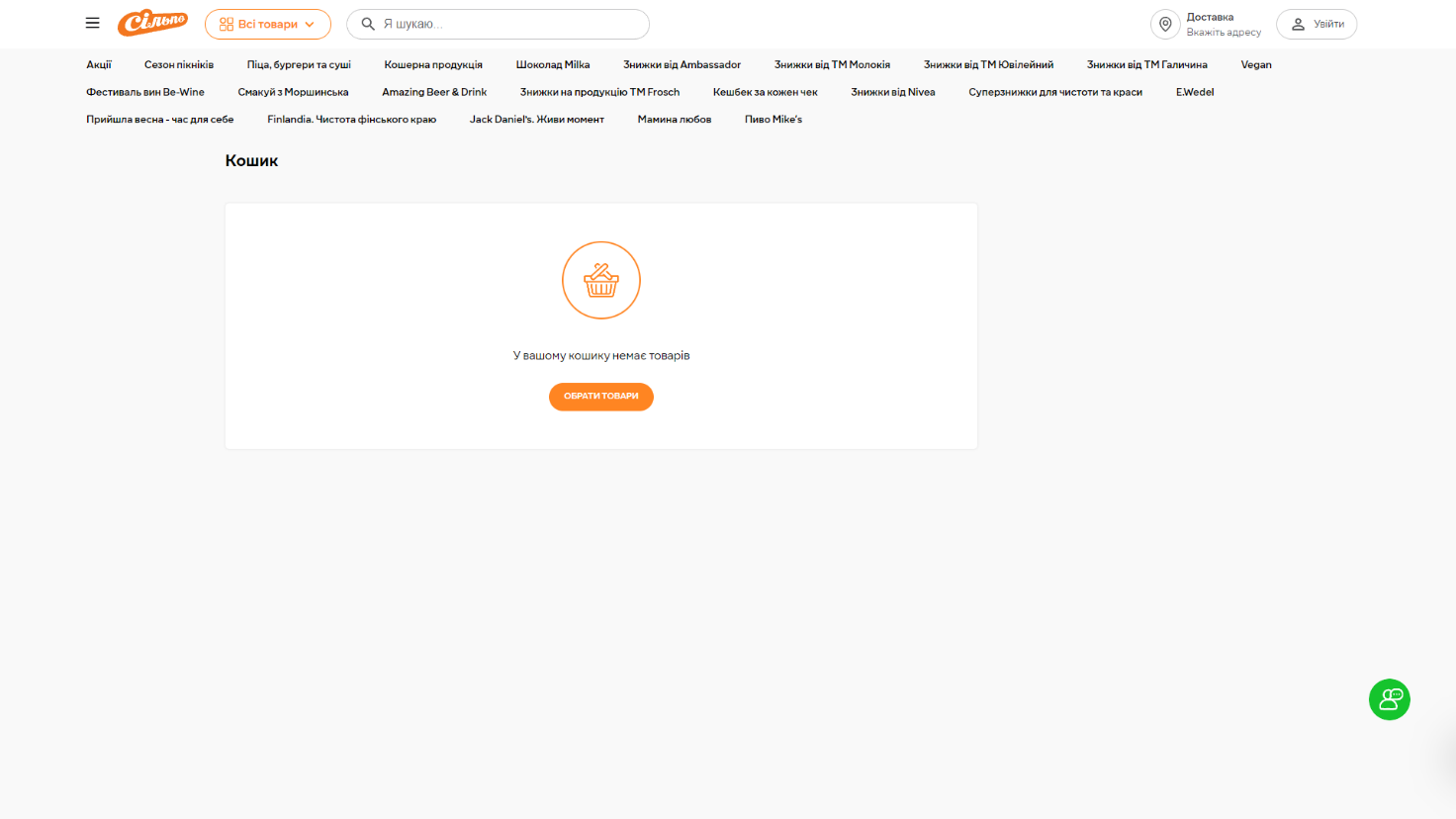


Рисунок 1.2 – Кошик

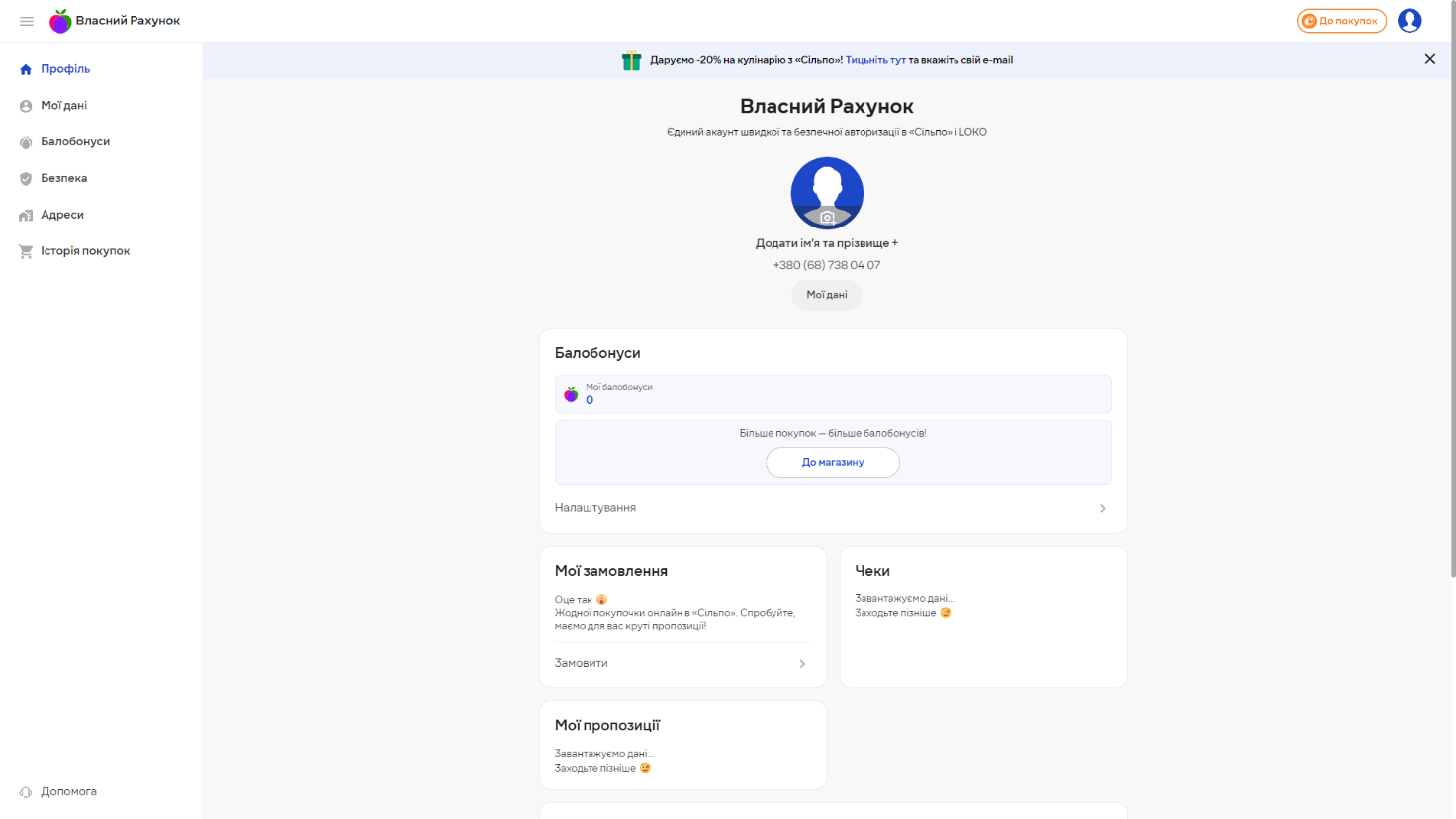


Рисунок 1.3 – Особистий профіль

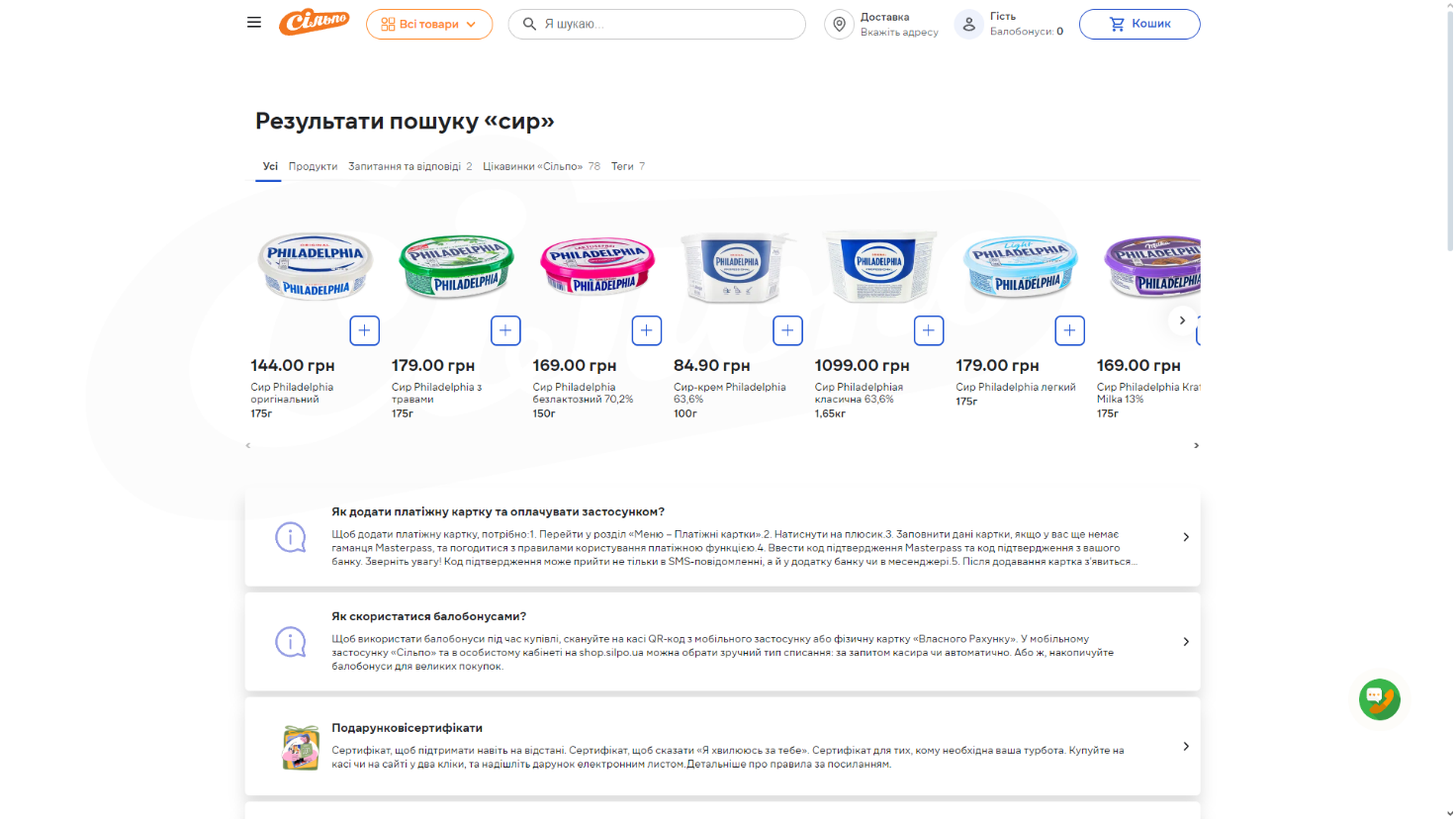


Рисунок 1.4 – Пошук

1. АТБ [[2](#_ПЕРЕЛІК_ДЖЕРЕЛ_ТА)]

Сервіс дозволяє передивитись товари, додати які цікавлять до кошика, після чого оформити оплату. Також доступна можливість авторизуватись/зареєструватись.

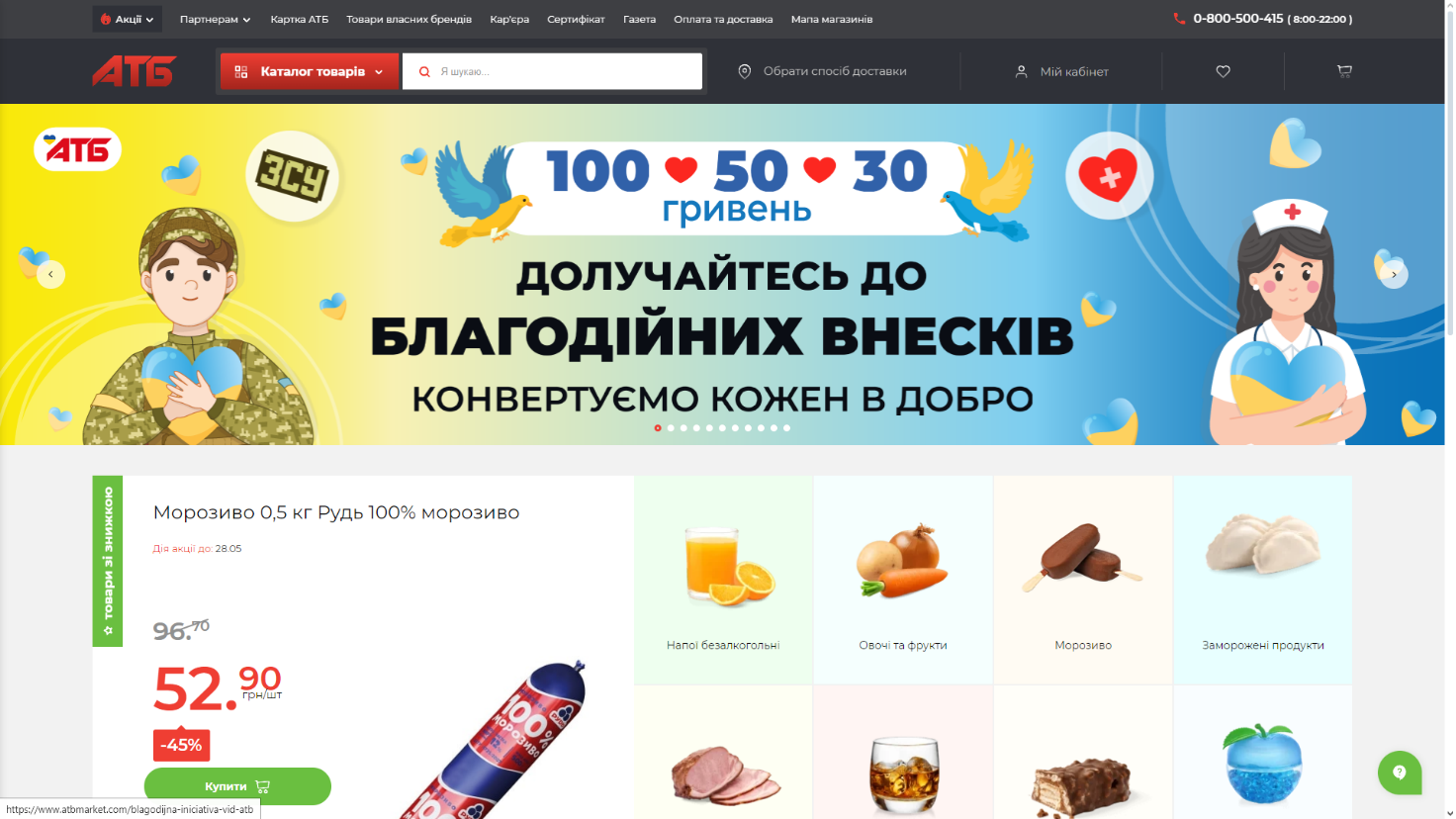


Рисунок 1.5 – Головна сторінка сайту «АТБ»

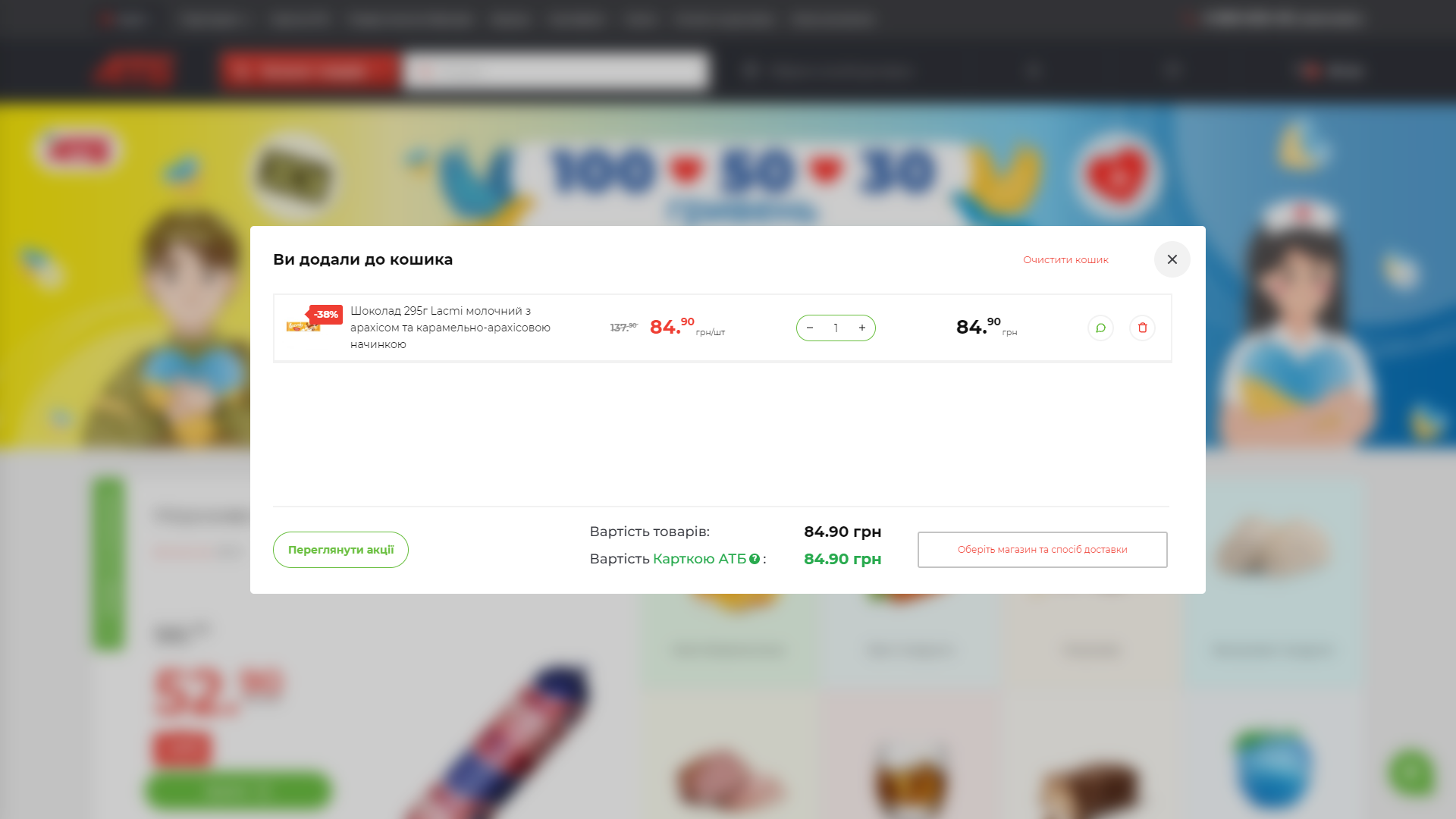


Рисунок 1.6 – Кошик

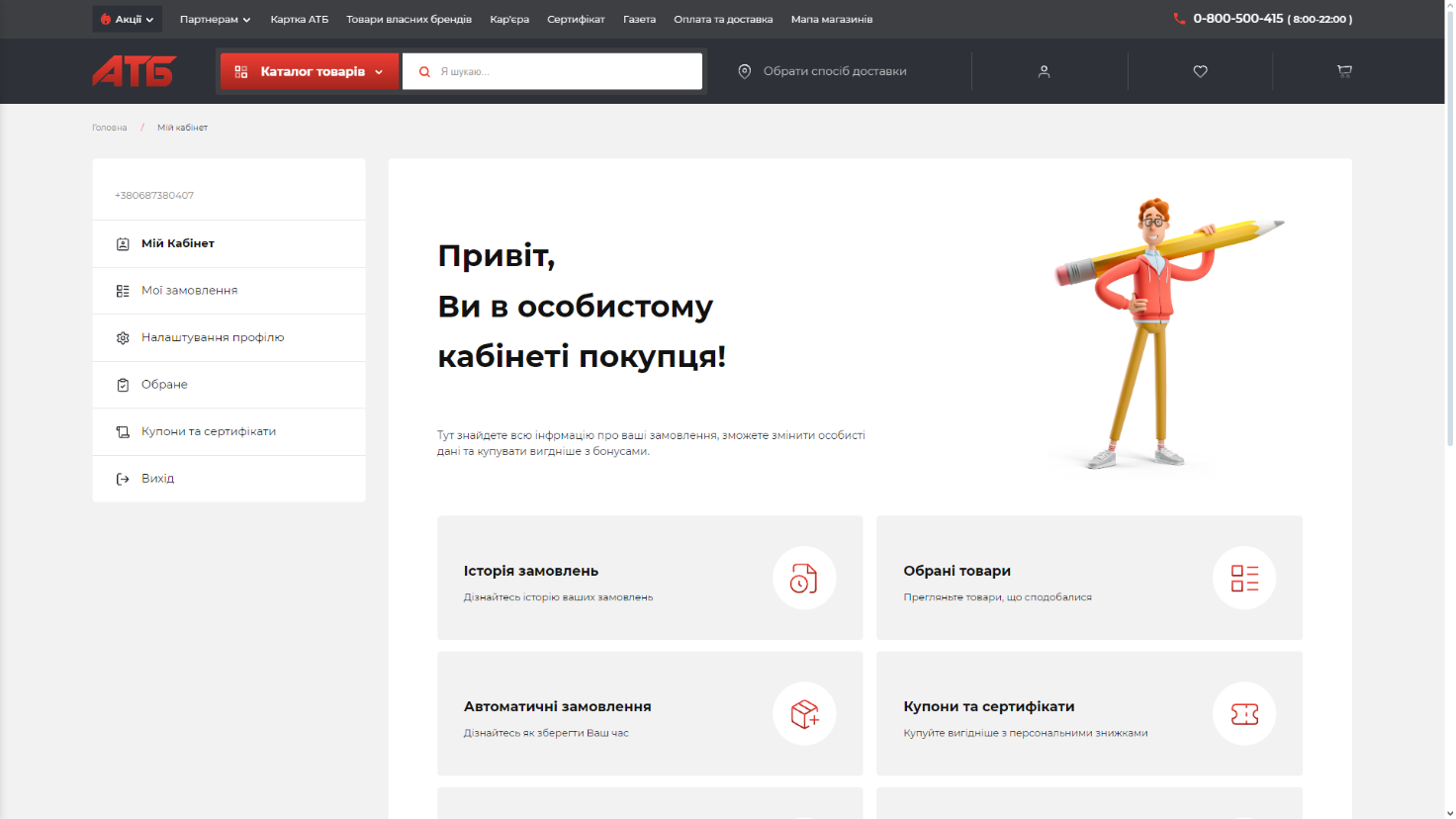


Рисунок 1.7 – Особистий профіль

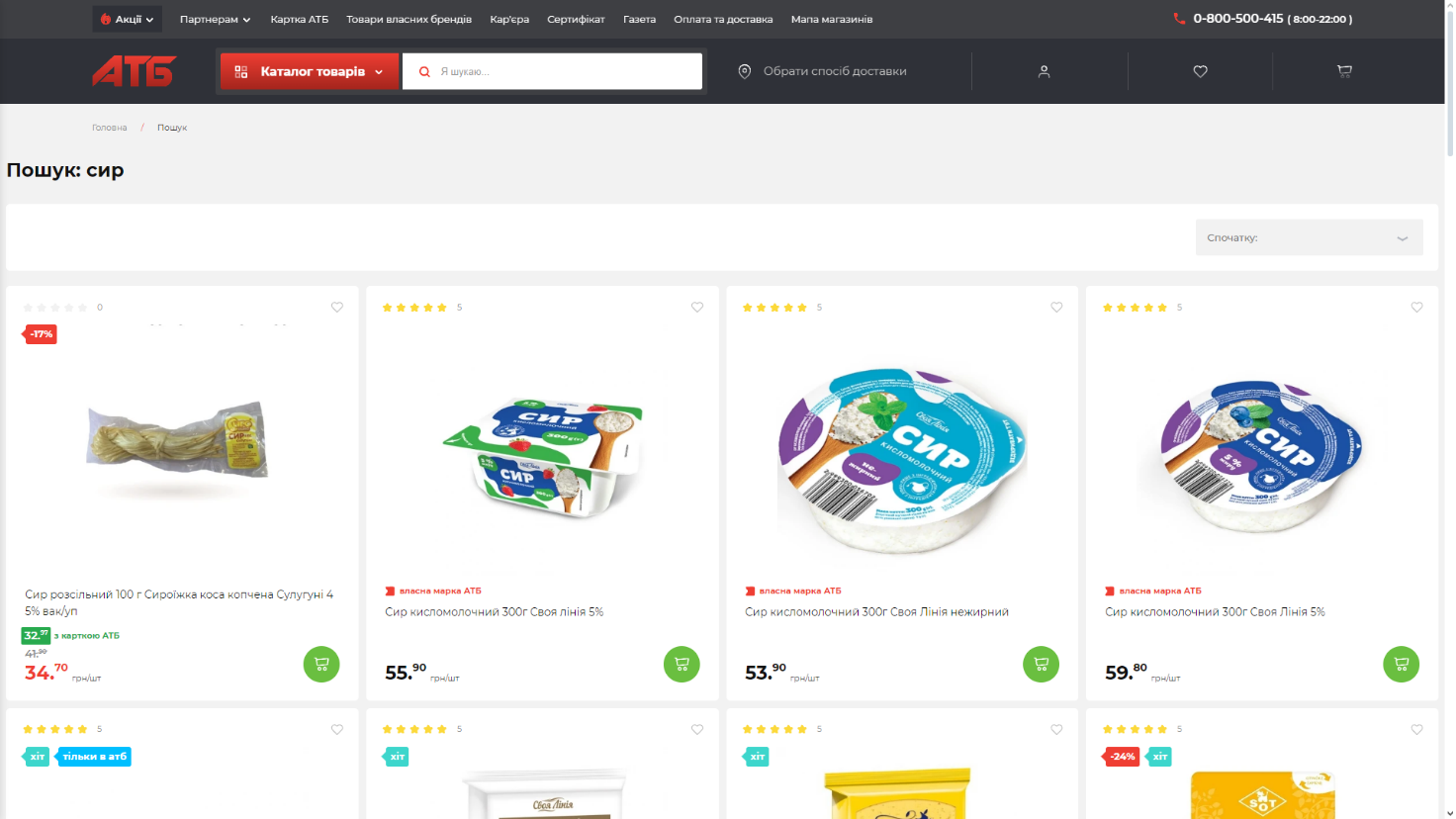


Рисунок 1.8 – Особистий кабінет

## Побудова Mind Map для заданої предметної області

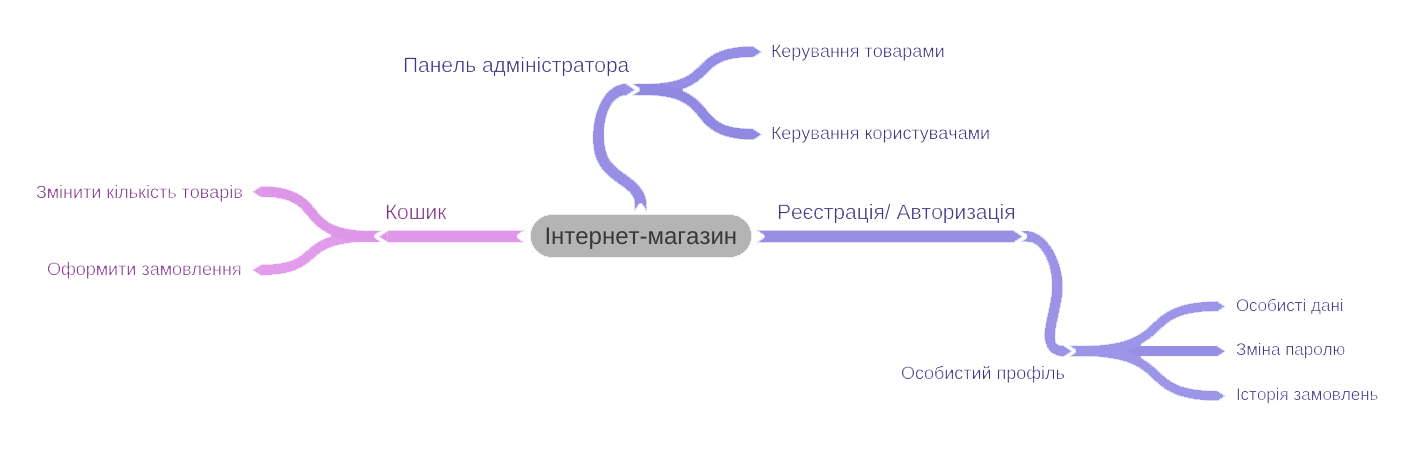


Рисунок 1.9 – Mind map

## Виділення ролей користувачів і формулювання функціональних вимог

Були виділені три ролі користувачів: «Адміністратор», «Зареєстрований користувач», «Гість».

Опишемо функціональні вимоги кожного типу користувачів.

Таблиця 1.1 – Функціональні вимоги користувача з роллю «Адміністратор»

|  |  |
| --- | --- |
| **Іденти-фікатор** | **Функціональні вимоги** |
| А.1. | Адміністратор може додавати, редагувати та видаляти товари. |
| А.2. | Адміністратор може додавати, редагувати та видаляти користувачів. |

Таблиця 1.2 – Функціональні вимоги користувача з роллю «Зареєстрований користувач»

|  |  |
| --- | --- |
| **Іденти-фікатор** | **Функціональні вимоги** |
| ЗК.1. | Зареєстрований користувач може купити товари. |
| ЗК.2. | Зареєстрований користувач може редагувати особистий профіль.  ***Примітка:***   1. можлива зміна пароля; 2. можлива зміна ім’я; 3. можлива зміна номеру телефону. |

Таблиця 1.3 – Функціональні вимоги користувача з роллю «Гість»

|  |  |
| --- | --- |
| **Іденти-фікатор** | **Функціональні вимоги** |
| Г.1. | Гість може передивитись наявні товари. |
| Г.2. | Гість може зареєструватись. |

## Проектування інтерфейсу користувача

Для реалізації замовником було обрано web-застосунок.

На основі сформульованих функціональних вимог були розроблені екранні форми, які наведені на рис. 1.10 – 1.12).

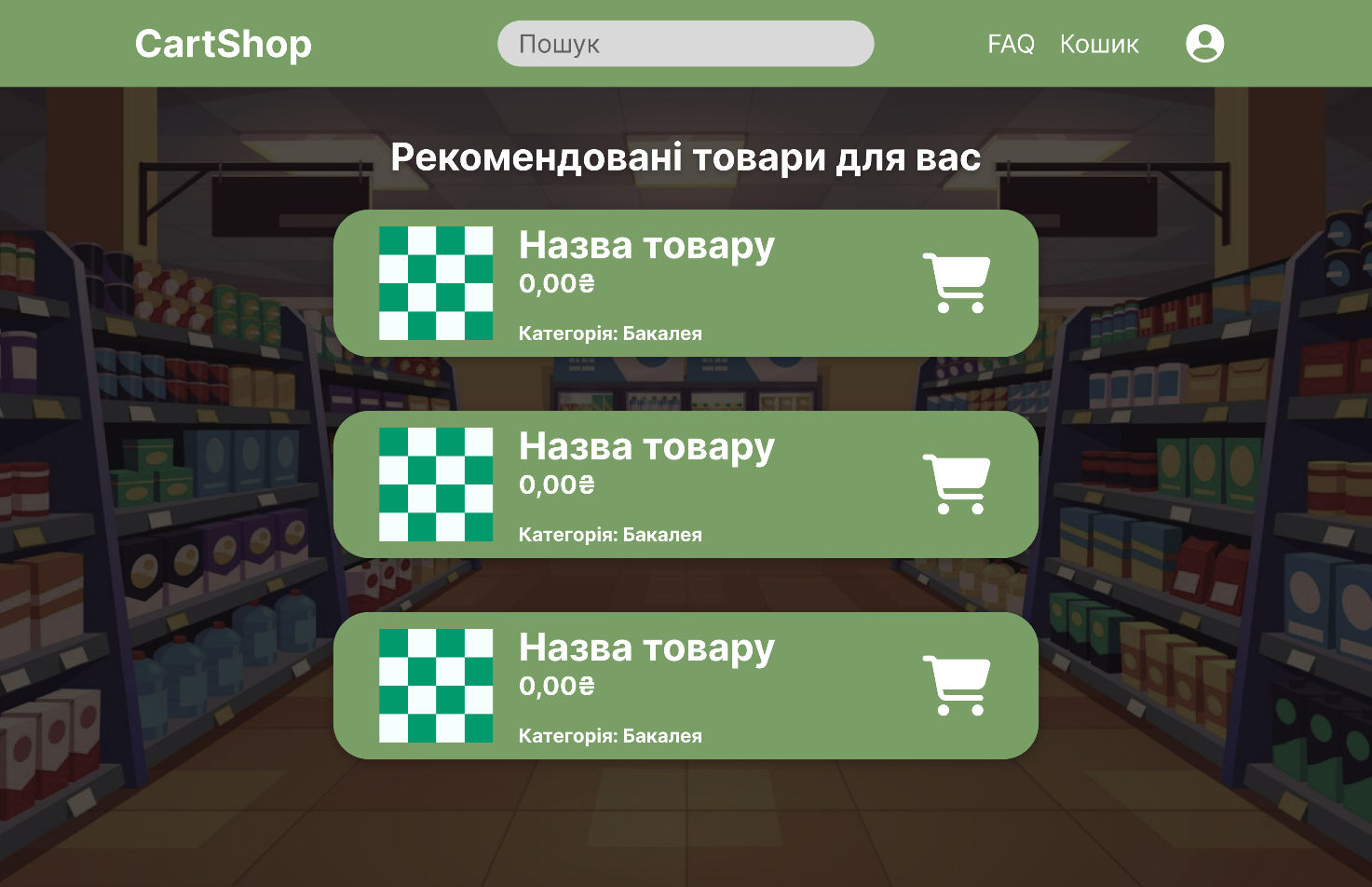


Рисунок 1.10 – Макет екранної форми «Головна сторінка»

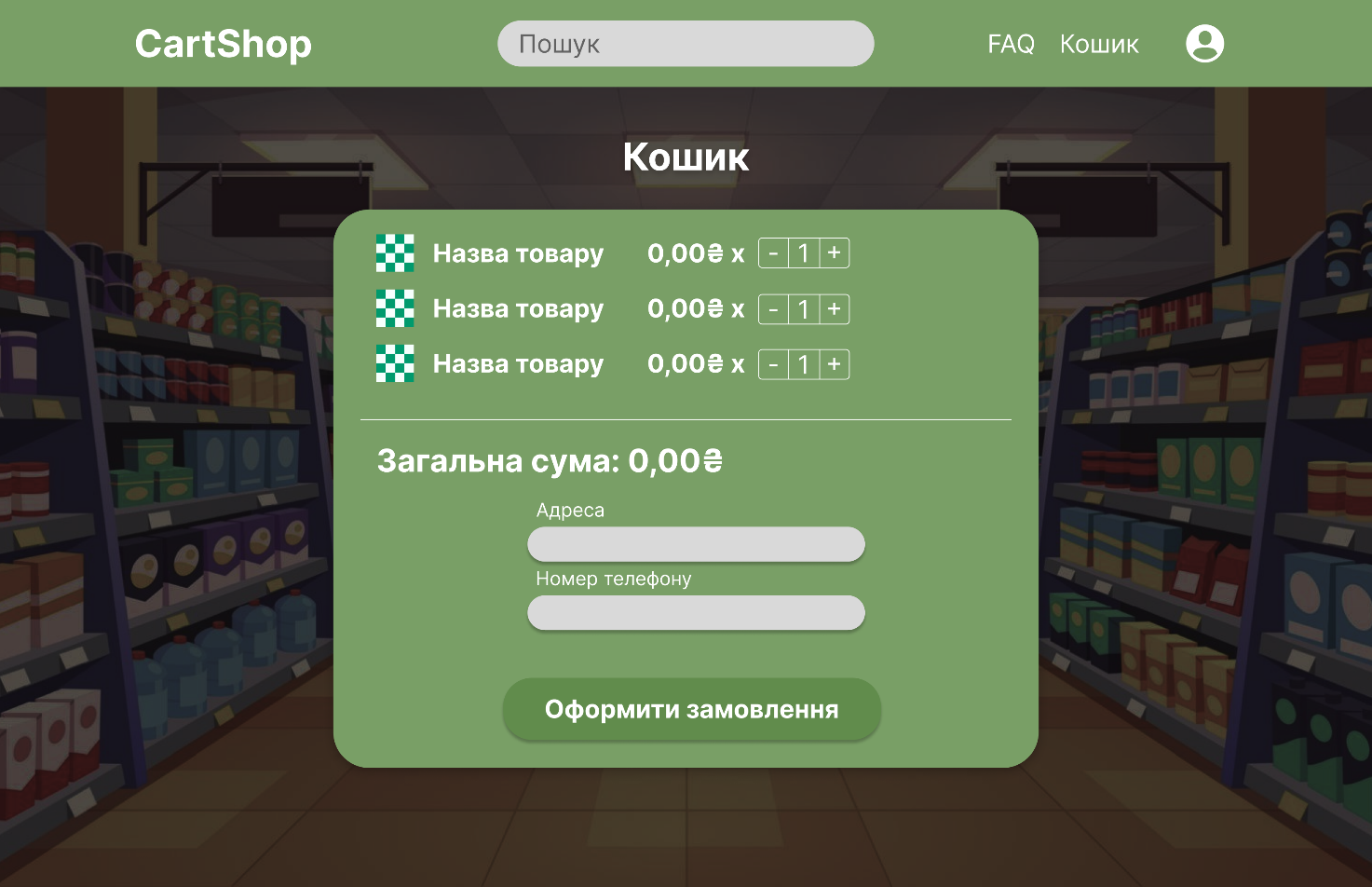


Рисунок 1.11 – Макет екранної форми «Кошик»

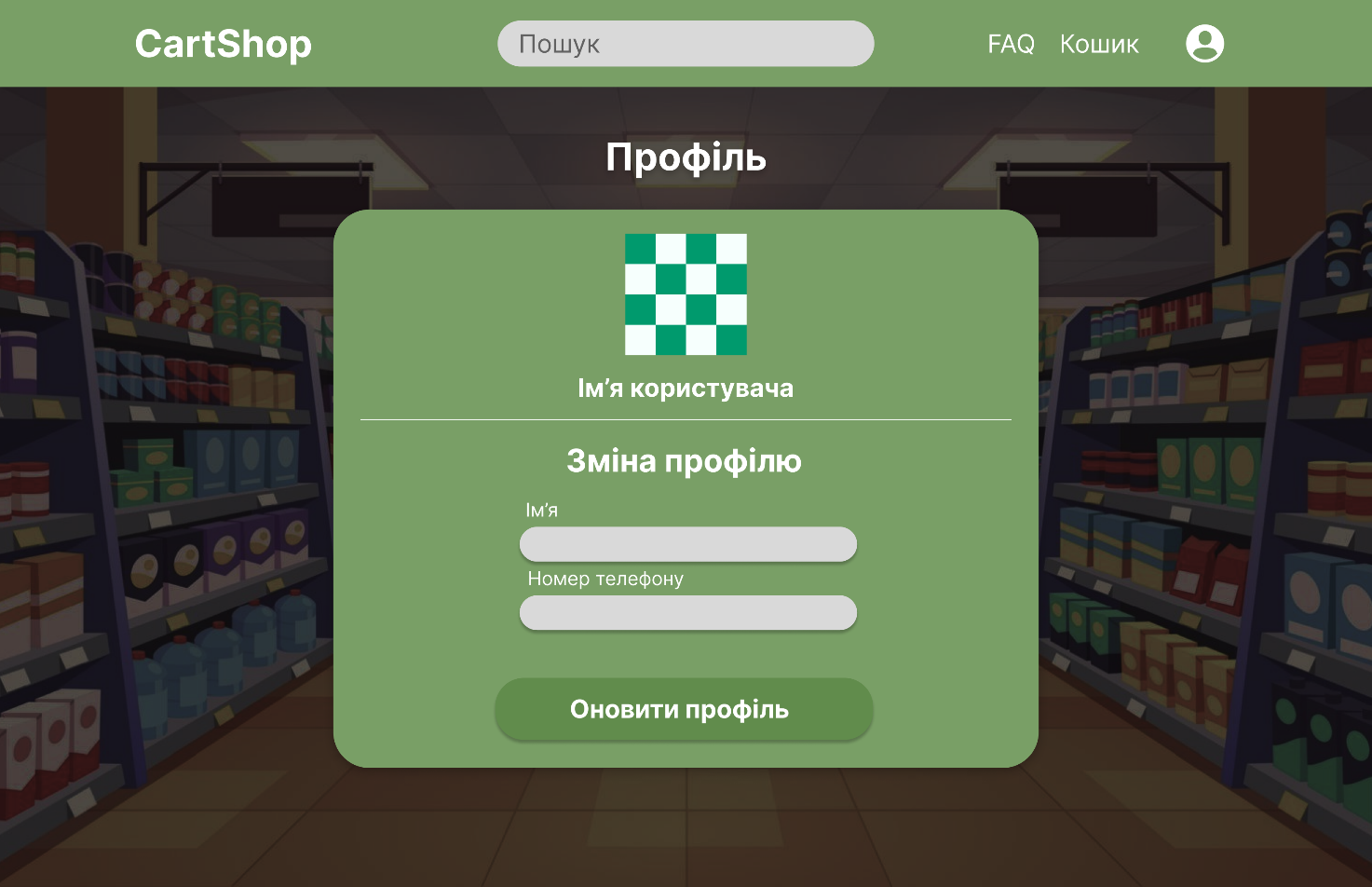


Рисунок 1.12 – Макет екранної форми «Особистий профіль»

## Розроблення функціональних тестів

### Функціональні тести для ролі користувача «Адміністратор»

На основі опису предметної області, а також функціональних вимог і екранних форм були розроблені функціональні тести для ролі користувача «Адміністратор» (див. табл. 1.4).

Таблиця 1.4 – Функціональні тести для ролі користувача «Адміністратор»

|  |  |  |
| --- | --- | --- |
| **Іденти-фікатор тесту** | **Послідовність дій користувача** | **Очікуваний результат** |
| *А.1. Адміністратор може додавати/редагувати/видаляти товари.* | | |
| А.1.1 | 1. Ввести назву. 2. Ввести ціну. 3. Вставити зображення. 4. Натиснути кнопку «Створити». | Успішне додавання товару |
| А.1.2 | 1. Натиснути кнопку «Редагувати». 2. Змінити назву. 3. Змінити ціну. 4. Натиснути кнопку «Зберегти». | Успішне редагування товару. |
| А.1.3 | Натиснути кнопку «Видалити». | Успішне видалення товару |
| А.1.4 | 1. Не ввести назву. 2. Ввести ціну. 3. Вставити зображення. 4. Натиснути кнопку «Створити». | Товар не був доданий. Користувач отримав повідомлення «Не були введені всі данні!» |
| А.1.5 | 1. Натиснути кнопку «Редагувати». 2. Стерти назву. 3. Змінити ціну. 4. Натиснути кнопку «Зберегти». | Товар не був відредагований. Користувач отримав повідомлення «Не були введені всі данні!» |
| *А.2. Адміністратор може додавати/редагувати або видаляти користувачів.* | | |
| А.2.1 | 1. Ввести ім’я. 2. Ввести номер телефона. 3. Ввести пошту. 4. Обрати роль користувача. 5. Ввести пароль. 6. Натиснути кнопку «Створити». | Успішне додавання користувача |
| А.2.2 | 1. Натиснути кнопку «Редагувати». 2. Змінити ім’я. 3. Змінити номер телефона. 4. Змінити роль користувача 5. Змінити пошту. 6. Натиснути кнопку «Зберегти». | Успішне редагування користувача |
| А.2.3 | Натиснути кнопку «Видалити». | Успішне видалення користувача |
| А.2.4 | 1. Не ввести ім’я. 2. Ввести номер телефона. 3. Ввести пошту. 4. Обрати роль користувача. 5. Ввести пароль. 6. Натиснути кнопку «Створити». | Користувач не був доданий. Користувач отримав повідомлення «Не були введені всі данні!» |
| А.2.5 | 1. Натиснути кнопку «Редагувати» 2. Стерти ім’я. 3. Змінити номер телефона. 4. Змінити пошту. 5. Змінити роль користувача. 6. Натиснути кнопку «Зберегти». | Користувач не був відредагований. Користувач отримав повідомлення «Не були введені всі данні!» |

### Функціональні тести для ролі користувача «Зареєстрований користувач»

На основі опису предметної області, а також функціональних вимог і екранних форм були розроблені функціональні тести для ролі користувача «Зареєстрований користувач» (див. табл. 1.5).

Таблиця 1.5 – Функціональні тести для ролі користувача «Зареєстрований користувач»

|  |  |  |
| --- | --- | --- |
| **Іденти-фікатор**  **тесту** | **Послідовність дій користувача** | **Очікуваний результат** |
| *ЗК.1. Зареєстрований користувач може змінити особисті дані.* | | |
| ЗК.1.1 | 1. Змінити ім’я. 2. Змінити номер телефону. 3. Натиснути кнопку «Зберегти». | Успішна зміна особистих даних |
| ЗК.1.2 | 1. Стерти ім’я. 2. Змінити номер телефону. 3. Натиснути кнопку «Зберегти». | Дані не були зміненні. Користувач отримав повідомлення «Не були введені всі данні!» |
| *ЗК.2. Зареєстрований користувач може купити товари.* | | |
| ЗК.2.1 | 1. Додати товари до кошику. 2. Обрати кількість. 3. Натиснути кнопку «Купити» | Успішна покупка товара/товарів |

### Функціональні тести для ролі користувача «Гість»

На основі опису предметної області, а також функціональних вимог і екранних форм були розроблені функціональні тести для ролі користувача «Гість» (див. табл. 1.6).

Таблиця 1.6 – Функціональні тести для ролі користувача «Гість»

|  |  |  |
| --- | --- | --- |
| **Іденти-фікатор**  **тесту** | **Послідовність дій користувача** | **Очікуваний результат** |
| *Г.1. Гість передивитись наявні товари.* | | |
| Г.1.1 | 1. Перейти в розділ «Найчастіші запитання» 2. Перейти в розділ «Сеанси» | Користувач успішно зміг передивитись наявну інформацію |
| Г.1.2 | 1. Перейти в розділ з сеансами. 2. Вибрати сеанс. 3. Вибрати час сеансу 4. Ввести номер телефона. 5. Вибрати місця. 6. Натиснути кнопку «Купити». | Успішна покупка квитка/квитків |
| *Г.2. Гість може зареєструватися* | | |
| Г.2.1 | 1. Ввести ім’я. 2. Ввести номер телефону. 3. Ввести пошту. 4. Ввести пароль. 5. Натиснути кнопку «Реєстрація». | Користувач успішно зареєструвався |
| Г.2.2 | 1. Не ввести ім’я. 2. Ввести номер телефону. 3. Ввести пошту. 4. Ввести пароль. 5. Натиснути кнопку «Реєстрація» | Реєстрація не успішна. Користувач отримав повідомлення «Не були введені всі дані!» |

# МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

## Виділення і опис класів предметної області

**Клас «User» – користувач**

Таблиця 2.1 – Виділення характеристик класу «User»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| Ім’я | Рядок | Не пустий рядок, хоча б 1 символ |
| Електронна пошта | Рядок | Не пустий рядок, наявність символу ‘@’ |
| Номер телефону | Рядок | Не пустий рядок, відповідність формату:  +38(0XX)-XXXXXXX |
| Роль | Enum | User, Admin |
| Пароль | Рядок | Не пустий рядок, хоча б 6 символів |

Таблиця 2.2 – Виділення поведінки класу «User»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Бізнес-правила** |
| Зміна особистих даних | Номер телефону, пароль, ім’я | IActionResult | 1. Перевірити чи дійсні вхідні дані. 2. Якщо пройдені перевірки, то змінити дані користувача |
| Додавання нового користувача | Електронна пошта, Ім’я, Телефон, Роль, Пароль | IActionResult | 1. Перевірити чи дійсні вхідні дані. 2. Якщо пройдені перевірки, то додати користувача. |
| Редагування користувача | Електронна пошта, Ім’я, Телефон, Роль | IActionResult | 1. Перевірити чи дійсні вхідні дані. 2. Якщо пройдені перевірки, то змінити користувача. |
| Видалення користувача | Ідентифікаційний номер | IActionResult | Якщо існує такий користувач, то він буде видалений |

**Клас «Product» – фільм**

Таблиця 2.3 – Виділення характеристик класу «Product»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| Назва | Рядок | Не пустий рядок, довжина: хоча б 1 символ |
| Ціна | Дробове число | >0 |
| Картинка | Рядок (посилання) | Не пустий рядок, довжина: хоча б 1 символ |

Таблиця 2.4 – Виділення поведінки класу «Product»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Примітка** |
| Додавання в кошик | Назва, ціна, кількість | IActionResult | Якщо такий продукт існує, то він додається в кошик |

## Встановлення зв’язків між класами

На рисунку 2.1 наведена діаграма класів, яка була побудова з використанням сервісу <https://www.lucidchart.com>.

|  |  |
| --- | --- |
|  | Рисунок 2.1 – Діаграма класів |

# ПРОГРАМНА РЕАЛІЗАЦІЯ КЛАСІВ ПРЕДМЕТНОЇ ОБЛАСТІ І ЇХ ТЕСТУВАННЯ

## Структура проекту з реалізацією класів предметної області

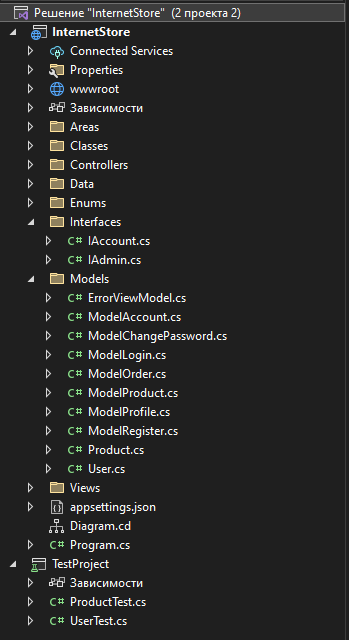


Рисунок 3.1 – Структура проекту з реалізацією предметної області

## Реалізація інтерфейсів і каркасів класів предметної області

**Інтерфейс «IAccount» – користувач**

using Microsoft.AspNetCore.Mvc;

using System.Threading.Tasks;

namespace InternetStore.Interfaces

{

public interface IAccount

{

Task<IActionResult> Login();

Task<IActionResult> Register();

Task<IActionResult> Logout();

Task<IActionResult> Profile();

Task<IActionResult> ChangePassword();

Task<IActionResult> History();

}

}

**Інтерфейс «IAdmin» – адміністратор**

using Microsoft.AspNetCore.Mvc;

namespace InternetStore.Interfaces

{

public interface IAdmin

{

Task<IActionResult> ManageAccounts();

Task<IActionResult> ManageProducts();

Task<IActionResult> AddAccount();

Task<IActionResult> AddProduct();

Task<IActionResult> EditAccount(string Id);

Task<IActionResult> EditProduct(int Id);

Task<IActionResult> DeleteAccount(string id);

Task<IActionResult> DeleteProduct(int id);

}

}

**Клас «User» – зареєстрований користувач**

using InternetStore.Enums;

using Microsoft.AspNetCore.Identity;

namespace InternetStore.Models

{

public class User: IdentityUser

{

}

}

**Клас «Product» – адміністратор**

namespace InternetStore.Models

{

public class Product

{

}

}

## Розроблення unit-тестів для класів предметної області

Розроблені unit-тести наведені у [Додатку В](#_ДОДАТОК_В._Лістинг).

Таблиця 3.1 – Покриття unit-тестами класу «User»

|  |  |
| --- | --- |
| **Назва методу** | **Кількість розроблених unit-тестів** |
| TestUser | 1 |

Таблиця 3.2 – Покриття unit-тестами класу «Product»

|  |  |
| --- | --- |
| **Назва методу** | **Кількість розроблених unit-тестів** |
| TestProduct | 1 |

## Повна реалізація класів предметної області

У [Додатку А](#_ДОДАТОК_А._Лістинг) наведено повну реалізацію програмного коду розроблених класів предметної області.

## Результати unit-тестування класів предметної області

На рисунках 3.2 – 3.3 наведено скріншоти проходження розроблених unit-тестів для класів предметної області.



Рисунок 3.2 – Скріншот проходження unit-тестів для класу User



Рисунок 3.3 – Скріншот проходження unit-тестів для класу Product

# ПРОЄКТУВАННЯ І ПРОГРАМНА РЕАЛІЗАЦІЯ КЛАСІВ ІНТЕРФЕЙСУ КОРИСТУВАЧА

## Структура проекту з реалізацією класів інтерфейсу користувача

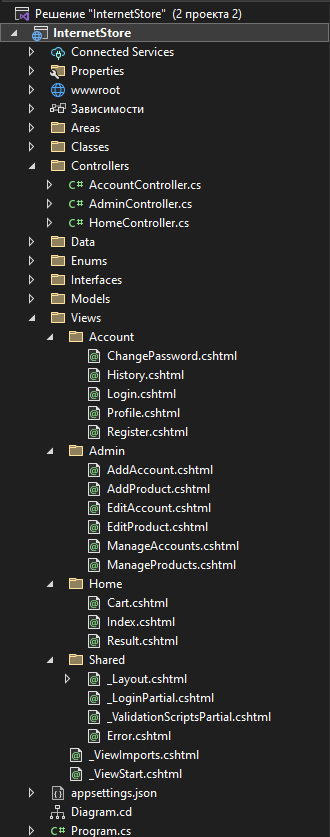


Рисунок 4.1 – Структура проекту з інтерфейсом користувача

## Виділення класів для реалізації інтерфейсу користувача

Класи HomeController, AccountController, AdminController успадковують базовий клас Controller. Це дає можливість використовувати всі функціональні можливості, які надає Controller.

Методи класу HomeController:

1. Index():

* Відповідає за обробку запитів до головної сторінки.
* Отримує список продуктів із бази даних та повертає представлення з цими продуктами.

1. Result(string query):

* Відповідає за обробку запитів до сторінки результатів пошуку.
* Якщо запит пустий або містить тільки пробіли, перенаправляє на головну сторінку.
* Отримує список продуктів, які відповідають пошуковому запиту, та повертає представлення з цими продуктами.

1. AddToCart(int id) (HttpPost, Authorize):

* Відповідає за обробку POST-запитів до додавання продукту до кошика.
* Отримує поточного користувача та продукт за його ідентифікатором.
* Якщо продукт знайдено, додає або оновлює запис у таблиці замовлень.
* Зберігає зміни у базі даних та повертає відповідь Ok.

1. Cart() (Authorize):

* Відповідає за обробку запитів до сторінки кошика.
* Отримує поточного користувача та список продуктів у його кошику.
* Повертає представлення з цими продуктами.

1. Cart(List<ModelOrder> orders) (HttpPost, Authorize):

* Відповідає за обробку POST-запитів до оформлення замовлення.
* Створює нові записи у таблиці продуктів на основі записів у кошику користувача.
* Видаляє всі записи з кошика після оформлення замовлення.
* Зберігає зміни у базі даних та перенаправляє на сторінку історії замовлень.

Методи класу AccountController:

1. Login() (HttpGet, AllowAnonymous):

* Відповідає за відображення форми авторизації.
* Повертає представлення форми авторизації.

1. Login(ModelLogin model) (HttpPost, AllowAnonymous, ValidateAntiForgeryToken):

* Відповідає за обробку POST-запитів до форми авторизації.
* Перевіряє, чи є введені дані валідними, та виконує авторизацію користувача.
* У разі успішної авторизації виконує подію UserEventOccurred з повідомленням про успішну авторизацію.

1. Register() (HttpGet, AllowAnonymous):

* Відповідає за відображення форми реєстрації.
* Повертає представлення форми реєстрації.

1. Register(ModelRegister model) (HttpPost, AllowAnonymous, ValidateAntiForgeryToken):

* Відповідає за обробку POST-запитів до форми реєстрації.
* Перевіряє, чи є введені дані валідними, та виконує реєстрацію нового користувача.
* У разі успішної реєстрації виконує подію UserEventOccurred з повідомленням про успішну реєстрацію.

1. Logout() (HttpPost, ValidateAntiForgeryToken):

* Відповідає за вихід користувача з системи.
* Виконує вихід користувача та перенаправляє на головну сторінку.

1. Profile() (Authorize):

* Відповідає за відображення профілю користувача.
* Отримує поточного користувача та повертає представлення з даними користувача.

1. Profile(ModelProfile model) (HttpPost, Authorize):

* Відповідає за обробку POST-запитів до форми редагування профілю.
* Перевіряє, чи є введені дані валідними, та оновлює дані користувача.
* У разі успішного оновлення виконує подію UserEventOccurred з повідомленням про успішну зміну даних.

1. ChangePassword() (Authorize):

* Відповідає за відображення форми зміни пароля.
* Повертає представлення форми зміни пароля.

1. ChangePassword(ModelChangePassword model) (HttpPost, Authorize, ValidateAntiForgeryToken):

* Відповідає за обробку POST-запитів до форми зміни пароля.
* Перевіряє, чи є введені дані валідними, та змінює пароль користувача.
* У разі успішної зміни пароля виконує подію UserEventOccurred з повідомленням про успішну зміну пароля.

1. History() (Authorize):

* Відповідає за відображення історії замовлень користувача.
* Отримує поточного користувача та список продуктів, які він замовив.
* Повертає представлення з історією замовлень.

Методи класу AdminController:

1. ManageAccounts() (Authorize):

* Відображає список усіх облікових записів користувачів, крім поточного адміністратора.
* Доступний лише для адміністратора.

1. ManageProducts() (Authorize):

* Відображає список усіх продуктів.
* Доступний лише для адміністратора.

1. AddAccount() (Authorize):

* Відображає форму для додавання нового облікового запису користувача.
* Доступний лише для адміністратора.

1. AddAccount(ModelAccount account) (HttpPost, Authorize):

* Обробляє POST-запити для додавання нового облікового запису користувача.
* Додає нового користувача в систему та перенаправляє на сторінку управління обліковими записами у разі успіху.
* Доступний лише для адміністратора.

1. AddProduct() (Authorize):

* Відображає форму для додавання нового продукту.
* Доступний лише для адміністратора.

1. AddProduct(Product product, IFormFile Image) (HttpPost, Authorize):

* Обробляє POST-запити для додавання нового продукту.
* Зберігає зображення продукту на сервері та додає продукт в базу даних.
* Доступний лише для адміністратора.

1. EditAccount(string Id) (HttpGet, Authorize):

* Відображає форму для редагування облікового запису користувача за вказаним ID.
* Доступний лише для адміністратора.

1. EditAccount(ModelAccount account) (HttpPost, Authorize):

* Обробляє POST-запити для редагування облікового запису користувача.
* Оновлює дані користувача в системі та перенаправляє на сторінку управління обліковими записами у разі успіху.
* Доступний лише для адміністратора.

1. EditProduct(int Id) (Authorize):

* Відображає форму для редагування продукту за вказаним ID.
* Доступний лише для адміністратора.

1. EditProduct(Product currproduct) (HttpPost, Authorize):

* Обробляє POST-запити для редагування продукту.
* Оновлює дані продукту в базі даних та перенаправляє на сторінку управління продуктами у разі успіху.
* Доступний лише для адміністратора.

1. DeleteAccount(string id) (HttpPost, Authorize, ValidateAntiForgeryToken):

* Обробляє POST-запити для видалення облікового запису користувача за вказаним ID.
* Видаляє користувача з системи та перенаправляє на сторінку управління обліковими записами у разі успіху.
* Доступний лише для адміністратора.

1. DeleteProduct(int id) (HttpPost, Authorize, ValidateAntiForgeryToken):

* Обробляє POST-запити для видалення продукту за вказаним ID.
* Видаляє продукт з бази даних та зображення продукту з серверу.
* Перенаправляє на сторінку управління продуктами у разі успіху.
* Доступний лише для адміністратора.

## Програмна реалізація класів інтерфейсу користувача

У [Додатку Б](#_ДОДАТОК_Б._Лістинг) наведено повну реалізацію програмного коду розроблених класів інтерфейсу користувача.

# ФУНКЦІОНАЛЬНЕ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Функціональне тестування для ролі користувача «Адміністратор»

Для проведення функціонального тестування для ролі «Адміністратор» були використані розроблені у п. 1.4.1. функціональні тести. Результати проведення функціонального тестування наведені нижче.

**Ідентифікатор тесту:** А.1.1.

**Вхідні дані:**

1. Ввести дату сеансу: «20.01.2023».

2. Ввести час сеансу: «10:00».

3. Обрати зал у кінотеатрі: «2».

4. Обрати фільм: «Аватар: шлях води».

5. Натиснути кнопку «Додати».

**Очікуваний результат:** Успішне додавання сеансу

**Отриманий результат:**

Рисунок 5.1 – Скриншот виконання тесту А.1.1

**Ідентифікатор тесту:**

**Вхідні дані:**

**Очікуваний результат:**

**Отриманий результат:**

Рисунок 5.2 – Скриншот виконання тесту \_\_\_\_\_

## Функціональне тестування для ролі користувача «Зареєстрований користувач»

Для проведення функціонального тестування для ролі «Зареєстрований користувач» були використані розроблені у п. 1.4.2. функціональні тести. Результати проведення функціонального тестування наведені нижче.

**Ідентифікатор тесту:**

**Вхідні дані:**

**Очікуваний результат:**

**Отриманий результат:**

Рисунок 5.3 – Скриншот виконання тесту \_\_\_\_\_

## Функціональне тестування для ролі користувача «Гість»

Для проведення функціонального тестування для ролі «Гість» були використані розроблені у п. 1.4.3. функціональні тести. Результати проведення функціонального тестування наведені нижче.

**Ідентифікатор тесту:**

**Вхідні дані:**

**Очікуваний результат:**

**Отриманий результат:**

Рисунок 5.4 – Скриншот виконання тесту \_\_\_\_\_

# ВИСНОВКИ

У результаті виконання курсової роботи було розроблено програмне забезпечення з використанням об’єктно-орієнтованої парадигми для вказати предметну область.

В ході роботи було…. (коротко описати що саме було зроблено під час виконання курсової роботи).

…

….

**Примітка:**

Посилання на github-репозиторій: <https://github.com/zhytnykOM/Course-Project.git>

Посилання на відеоролик: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАНЬ

1. Воробйов, Ю. А. Правила оформлення навчальних і науково-дослідних документів [Текст] : навч. посіб. / Ю. А. Воробйов, Ю. О. Сисоєв. – 4-те вид., випр. і доп. – Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2019. – 88 с.
2. Автоматичне оформлення джерел по ВАК України [Електронний ресурс] – Режим доступу до ресурсу: <http://vak.in.ua/do.php>.
3. Сільпо [Електронний ресурс] – Режим доступу до ресурсу: <https://silpo.ua>.
4. АТБ-Маркет – доставка продуктів додому в Києві та по всій Україні [Електронний ресурс] – Режим доступу до ресурсу: <https://www.atbmarket.com>.

# ДОДАТОК А. Лістинг класів предметної області

Клас «ModelAccount»

using InternetStore.Enums;

using System.Data;

namespace InternetStore.Models

{

public class ModelAccount

{

public string Id { get; set; } = "";

public string Name { get; set; }

public string UserName { get; set; }

public string Email { get; set; }

public string PhoneNumber { get; set; }

public UserRole Role { get; set; }

public string Password { get; set; }

}

}

Клас «ModelChangePassword»

using System.ComponentModel.DataAnnotations;

namespace InternetStore.Models

{

public class ModelChangePassword

{

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[DataType(DataType.Password)]

[Display(Name = "Ваш пароль")]

public string CurrentPassword { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[StringLength(100, ErrorMessage = "Довжина пароля менше 8 символів!", MinimumLength = 8)]

[DataType(DataType.Password)]

[Display(Name = "Новий пароль")]

public string NewPassword { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[DataType(DataType.Password)]

[Display(Name = "Підтвердіть новий пароль")]

[Compare("NewPassword", ErrorMessage = "Паролі не співпадають!")]

public string ConfirmPassword { get; set; }

}

}

Клас «ModelLogin»

using System.ComponentModel.DataAnnotations;

namespace InternetStore.Models

{

public class ModelLogin

{

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[Display(Name = "Логін")]

public string UserName { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[DataType(DataType.Password, ErrorMessage = "Неправильний формат пароля!")]

[Display(Name = "Пароль")]

public string Password { get; set; }

[Display(Name = "Запам'ятати мене")]

public bool RememberMe { get; set; } = false;

}

}

Клас «ModelOrder»

namespace InternetStore.Models

{

public class ModelOrder

{

public int Id { get; set; }

public string UserId { get; set; }

public int ProductId { get; set; }

public string Name { get; set; }

public double Price { get; set; }

public int Count { get; set; }

}

}

Клас «ModelProduct»

namespace InternetStore.Models

{

public class ModelProduct

{

public int Id { get; set; }

public string UserId { get; set; }

public string Name { get; set; }

public double Price { get; set; }

public int Count { get; set; }

}

}

Клас «ModelProfile»

namespace InternetStore.Models

{

public class ModelProfile

{

public string Name { get; set; }

public string PhoneNumber { get; set; }

}

}

Клас «ModelRegister»

using System.ComponentModel.DataAnnotations;

namespace InternetStore.Models

{

public class ModelRegister

{

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[Display(Name = "Ім'я користувача")]

public string Name { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[Display(Name = "Логін")]

public string UserName { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[EmailAddress]

[Display(Name = "Пошта")]

public string Email { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[Phone]

[Display(Name = "Мобільний телефон")]

public string PhoneNumber { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string Password { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[DataType(DataType.Password)]

[Display(Name = "Підтвердіть пароль")]

[Compare("Password", ErrorMessage = "Паролі не збігаються.")]

public string ConfirmPassword { get; set; }

}

}

Клас «Product»

namespace InternetStore.Models

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public double Price { get; set; }

public string? Image { get; set; }

}

}

Клас «User»

using InternetStore.Enums;

using Microsoft.AspNetCore.Identity;

namespace InternetStore.Models

{

public class User: IdentityUser

{

public string Name { get; set; }

public UserRole Role { get; set; }

}

}

# ДОДАТОК Б. Лістинг класів інтерфейсу користувача

Клас «HomeController»

using InternetStore.Data;

using InternetStore.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using System.Diagnostics;

namespace InternetStore.Controllers

{

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

private readonly ApplicationDbContext \_context;

private readonly UserManager<User> \_userManager;

public HomeController(ILogger<HomeController> logger, ApplicationDbContext context, UserManager<User> userManager)

{

\_logger = logger;

\_context = context;

\_userManager = userManager;

}

public IActionResult Index()

{

List<Product> products = \_context.Products.ToList();

return View(products);

}

public IActionResult Result(string query)

{

if (string.IsNullOrWhiteSpace(query))

{

return RedirectToAction("Index");

}

List<Product> products = \_context.Products.Where(Product => Product.Name.Contains(query)).ToList();

return View(products);

}

[Authorize]

[HttpPost]

public async Task<IActionResult> AddToCart(int id)

{

var user = await \_userManager.GetUserAsync(User);

var product = await \_context.Products.FindAsync(id);

if (product == null)

{

return NotFound();

}

var orderItem = \_context.ModelOrders.FirstOrDefault(o => o.UserId == user.Id && o.ProductId == id);

if (orderItem != null)

{

orderItem.Count++;

\_context.Update(orderItem);

}

else

{

ModelOrder modelProduct = new ModelOrder()

{

ProductId = product.Id,

Name = product.Name,

Price = product.Price,

Count = 1,

UserId = user.Id

};

\_context.ModelOrders.Add(modelProduct);

}

await \_context.SaveChangesAsync();

return Ok();

}

[Authorize]

public async Task<IActionResult> Cart()

{

var user = await \_userManager.GetUserAsync(User);

var product = \_context.ModelOrders.Where(p => p.UserId == user.Id).ToList();

return View(product);

}

[Authorize]

[HttpPost]

public async Task<IActionResult> Cart(List<ModelOrder> orders)

{

var user = await \_userManager.GetUserAsync(User);

foreach (var orderItem in orders)

{

ModelProduct modelProduct = new ModelProduct()

{

Name = orderItem.Name,

Price = orderItem.Price,

Count = orderItem.Count,

UserId = user.Id

};

\_context.ModelProducts.Add(modelProduct);

}

var ordersToRemove = \_context.ModelOrders.Where(o => o.UserId == user.Id).ToList();

\_context.ModelOrders.RemoveRange(ordersToRemove);

await \_context.SaveChangesAsync();

return RedirectToAction("History","Account");

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

Клас «AccountController»

using InternetStore.Data;

using InternetStore.Interfaces;

using InternetStore.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using System.Net.Sockets;

namespace InternetStore.Controllers

{

public class AccountController : Controller, IAccount

{

private readonly UserManager<User> \_userManager;

private readonly SignInManager<User> \_signInManager;

private readonly ApplicationDbContext \_context;

public delegate Task UserEventHandler(string userId, string message);

public event UserEventHandler UserEventOccurred;

public AccountController(UserManager<User> userManager, SignInManager<User> signInManager, ApplicationDbContext context)

{

\_userManager = userManager;

\_signInManager = signInManager;

\_context = context;

UserEventOccurred += async (userId, message) =>

{

TempData["UserNotification"] = message;

};

}

[HttpGet]

[AllowAnonymous]

public async Task<IActionResult> Login()

{

return View();

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Login(ModelLogin model)

{

if (ModelState.IsValid)

{

var user = await \_userManager.FindByNameAsync(model.UserName);

if (user != null && await \_userManager.CheckPasswordAsync(user, model.Password))

{

await \_signInManager.SignInAsync(user, model.RememberMe);

await UserEventOccurred?.Invoke(user.Id, "Успішна авторизація!");

return RedirectToAction("Index", "Home");

}

ModelState.AddModelError(string.Empty, "Помилка авторизації!");

return View(model);

}

return View(model);

}

[HttpGet]

[AllowAnonymous]

public async Task<IActionResult> Register()

{

return View();

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Register(ModelRegister model)

{

if (ModelState.IsValid)

{

var user = new User { Name = model.Name, UserName = model.UserName, Email = model.Email, PhoneNumber = model.PhoneNumber };

var result = await \_userManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

await \_signInManager.SignInAsync(user, isPersistent: false);

await UserEventOccurred?.Invoke(user.Id, "Користувач успішно зареєстрований!");

return RedirectToAction("Index", "Home");

}

ModelState.AddModelError(string.Empty, "Помилка реєстрації!");

return View(model);

}

return View(model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Logout()

{

await \_signInManager.SignOutAsync();

return RedirectToAction("Index", "Home");

}

[Authorize]

public async Task<IActionResult> Profile()

{

var user = await \_userManager.GetUserAsync(User);

var model = new ModelProfile

{

Name = user.Name,

PhoneNumber = user.PhoneNumber

};

return View(model);

}

[Authorize]

[HttpPost]

public async Task<IActionResult> Profile(ModelProfile model)

{

var user = await \_userManager.GetUserAsync(User);

if (ModelState.IsValid)

{

user.Name = model.Name;

user.PhoneNumber = model.PhoneNumber;

var result = await \_userManager.UpdateAsync(user);

if (result.Succeeded)

{

await UserEventOccurred?.Invoke(user.Id, "Дані успішно зміненні!");

return View(model);

}

}

return View(model);

}

[Authorize]

public async Task<IActionResult> ChangePassword()

{

ModelChangePassword model = new ModelChangePassword();

return View(model);

}

[HttpPost]

[Authorize]

[ValidateAntiForgeryToken]

public async Task<IActionResult> ChangePassword(ModelChangePassword model)

{

if (ModelState.IsValid)

{

var user = await \_userManager.GetUserAsync(User);

if (user == null)

{

return RedirectToAction(nameof(Login));

}

var changePasswordResult = await \_userManager.ChangePasswordAsync(user, model.CurrentPassword, model.NewPassword);

if (!changePasswordResult.Succeeded)

{

foreach (var error in changePasswordResult.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

return View(model);

}

await \_signInManager.RefreshSignInAsync(user);

await UserEventOccurred?.Invoke(user.Id, "Успішна зміна пароля!");

return View(model);

}

return View(model);

}

[Authorize]

public async Task<IActionResult> History()

{

var user = await \_userManager.GetUserAsync(User);

List<ModelProduct> products = \_context.ModelProducts.Where(p => p.UserId == user.Id).ToList();

return View(products);

}

}

}

Клас «AdminController»

using InternetStore.Data;

using InternetStore.Models;

using InternetStore.Enums;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using System.Data;

using InternetStore.Interfaces;

namespace InternetStore.Controllers

{

public class AdminController : Controller, IAdmin

{

private readonly UserManager<User> \_userManager;

private readonly ApplicationDbContext \_context;

private readonly IWebHostEnvironment \_webHostEnvironment;

public AdminController(UserManager<User> userManager, ApplicationDbContext context, IWebHostEnvironment webHostEnvironment)

{

\_userManager = userManager;

\_context = context;

\_webHostEnvironment = webHostEnvironment;

}

[Authorize]

public async Task<IActionResult> ManageAccounts()

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == UserRole.Admin)

{

var users = \_userManager.Users.ToList();

List<User> list = users.Where(u => u.Id != user.Id).ToList();

return View(list);

}

return RedirectToAction("Index", "Home");

}

[Authorize]

public async Task<IActionResult> ManageProducts()

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == UserRole.Admin)

{

List<Product> list = \_context.Products.ToList();

return View(list);

}

return RedirectToAction("Index", "Home");

}

[Authorize]

public async Task<IActionResult> AddAccount()

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == UserRole.Admin)

{

ModelAccount account = new ModelAccount();

return View(account);

}

return RedirectToAction("Index", "Home");

}

[HttpPost]

[Authorize]

public async Task<IActionResult> AddAccount(ModelAccount account)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == UserRole.Admin)

{

if (ModelState.IsValid)

{

var adduser = new User

{

Name = account.Name,

UserName = account.UserName,

Email = account.Email,

PhoneNumber = account.PhoneNumber,

Role = account.Role

};

var result = await \_userManager.CreateAsync(adduser, account.Password);

if (result.Succeeded)

{

return RedirectToAction("ManageAccounts");

}

}

return View(account);

}

return RedirectToAction("Index", "Home");

}

[Authorize]

public async Task<IActionResult> AddProduct()

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == UserRole.Admin)

{

Product product = new Product();

return View(product);

}

return RedirectToAction("Index", "Home");

}

[HttpPost]

[Authorize]

public async Task<IActionResult> AddProduct(Product product, IFormFile Image)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == UserRole.Admin)

{

if (Image != null && Image.Length > 0)

{

var uploadsFolder = Path.Combine(\_webHostEnvironment.WebRootPath, "images");

var uniqueFileName = Guid.NewGuid().ToString() + "\_" + Image.FileName;

var filePath = Path.Combine(uploadsFolder, uniqueFileName);

using (var fileStream = new FileStream(filePath, FileMode.Create))

{

await Image.CopyToAsync(fileStream);

}

product.Image = "/images/" + uniqueFileName;

if (ModelState.IsValid)

{

\_context.Products.Add(product);

await \_context.SaveChangesAsync();

return RedirectToAction("ManageProducts");

}

}

return View(product);

}

return RedirectToAction("Index", "Home");

}

[HttpGet]

[Authorize]

public async Task<IActionResult> EditAccount(string Id)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == UserRole.Admin)

{

var selecteduser = await \_userManager.FindByIdAsync(Id);

if (selecteduser != null)

{

ModelAccount account = new ModelAccount()

{

Id = selecteduser.Id,

Name = selecteduser.Name,

UserName = selecteduser.UserName,

Email = selecteduser.Email,

PhoneNumber = selecteduser.PhoneNumber,

Role = selecteduser.Role,

Password = selecteduser.PasswordHash

};

return View(account);

}

return RedirectToAction("ManageAccounts");

}

return RedirectToAction("Index", "Home");

}

[HttpPost]

[Authorize]

public async Task<IActionResult> EditAccount(ModelAccount account)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == UserRole.Admin)

{

if (ModelState.IsValid)

{

var editedUser = await \_userManager.FindByIdAsync(account.Id);

editedUser.Name = account.Name;

editedUser.UserName = account.UserName;

editedUser.Email = account.Email;

editedUser.PhoneNumber = account.PhoneNumber;

editedUser.Role = account.Role;

var result = await \_userManager.UpdateAsync(editedUser);

if (result.Succeeded)

{

return RedirectToAction("ManageAccounts");

}

}

return View(account);

}

return RedirectToAction("Index", "Home");

}

[Authorize]

public async Task<IActionResult> EditProduct(int Id)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == UserRole.Admin)

{

var product = await \_context.Products.FindAsync(Id);

if (product != null)

return View(product);

return RedirectToAction("ManageProducts");

}

return RedirectToAction("Index", "Home");

}

[HttpPost]

[Authorize]

public async Task<IActionResult> EditProduct(Product currproduct)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == UserRole.Admin)

{

if (ModelState.IsValid)

{

var film = \_context.Products.FirstOrDefault(f => f.Id == currproduct.Id);

if (film != null)

{

film.Name = currproduct.Name;

film.Price = currproduct.Price;

\_context.Products.Update(film);

await \_context.SaveChangesAsync();

return RedirectToAction("ManageProducts");

}

}

return View(currproduct);

}

return RedirectToAction("Index", "Home");

}

[HttpPost]

[Authorize]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteAccount(string id)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == UserRole.Admin)

{

var account = await \_userManager.FindByIdAsync(id);

if (account != null)

{

var result = await \_userManager.DeleteAsync(account);

if (result.Succeeded)

{

return RedirectToAction("ManageAccounts");

}

}

else

{

return RedirectToAction("ManageAccounts");

}

}

return RedirectToAction("Index", "Home");

}

[HttpPost]

[Authorize]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteProduct(int id)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == UserRole.Admin)

{

var film = \_context.Products.FirstOrDefault(f => f.Id == id);

if (film != null)

{

var currentDirectory = Directory.GetCurrentDirectory();

var imagePath = Path.Combine(currentDirectory, "wwwroot", film.Image.TrimStart('/'));

if (System.IO.File.Exists(imagePath))

{

try

{

System.IO.File.Delete(imagePath);

Console.WriteLine("Файл успішно видвалений: " + imagePath);

}

catch (Exception ex)

{

Console.WriteLine("Помилка при видаленні файла: " + ex.Message);

}

}

\_context.Remove(film);

await \_context.SaveChangesAsync();

}

return RedirectToAction("ManageProducts");

}

return RedirectToAction("Index", "Home");

}

}

}

# ДОДАТОК В. Лістинг класів unit-тестів

Тестуючий клас «UserTest»

using InternetStore.Enums;

using InternetStore.Models;

namespace TestProject

{

public class UserTest

{

[Fact]

public void TestUser()

{

User user = new()

{

Id = "8fyhj378jf891kf",

Name = "User",

UserName = "user1",

Email = "user1@gmail.com",

PhoneNumber = "0990000000",

Role = UserRole.User

};

Assert.Equal("8fyhj378jf891kf", user.Id);

Assert.Equal("User", user.Name);

Assert.Equal("user1", user.UserName);

Assert.Equal("user1@gmail.com", user.Email);

Assert.Equal("0990000000", user.PhoneNumber);

Assert.Equal(UserRole.User, user.Role);

}

}

}

Тестуючий клас «ProductTest»

using InternetStore.Models;

namespace TestProject

{

public class ProductTest

{

[Fact]

public void TestProduct()

{

Product product = new()

{

Id = 1,

Name = "Помідор",

Price = 4.50,

Image = "/images/djlhfwrfer9fg"

};

Assert.Equal(1, product.Id);

Assert.Equal("Помідор", product.Name);

Assert.Equal(4.50, product.Price);

Assert.Equal("/images/djlhfwrfer9fg", product.Image);

}

}

}