Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету(відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної циклової комісії))

Пояснювальна записка

до дипломної роботи
перший (бакалаврський)
(рівень вищої освіти)

на тему: «Розроблення вебзастосунку "МобіТренд" засобами ASP.NET»

Виконав: студент II курсу групи КНС-21

спеціальності: 122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Гудзеляк Р. Б.
(прізвище та ініціали)

Керівник Борецька І.Б.
(прізвище та ініціали)

Рецензент Процик Ю.С.
(прізвище та ініціали)

Львів-2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти перший(бакалаврський)
Спеціальність 122 "Комп'ютерні науки"

ЗАТВЕРДЖУЮ:
Завідувачка кафедри КН
Борецька І.Б.
„10" червня 2025 року

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Гудзеляк Ростислав Богданович
(прізвище, ім'я, по батькові)

1.Тема бакалаврської роботи: «*Розроблення вебзастосунку "МобіТренд" засобами ASP.NET*».

керівник роботи: Борецька Ірина Богданівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "15" листопада 2024 року, № С-882

2.Термін подання студентом проекту(роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Розробити вебсайт для покупок смартфонів та аксесуарів. Сайт з можливістю шукати смартфони у власній базі даних і також добавляти тренди. Користувач має змогу використовувати вебсайт в любий момент.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Стан проблемної області
Інформаційне та математичне забезпечення
Програмне та технічне забезпечення
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалу до доповіді.

6. Дата видачі завдання 18 листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№, з/п	Етапи бакалаврської роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів.	19.11.24 р. – 31.01.25 р.	Виконано
2.	Постановка задачі і її формалізація	01.02.25 р. – 15.02.25 р.	Виконано
3.	Виконання вхідного етапу технології	15.02.25 р. – 18.03.25 р.	Виконано
4.	Реалізація головних класів проекту	18.03.25 р. – 25.03.25 р.	Виконано
5.	Виконання етапу відлагодження проєкту	25.03.25 р. – 30.03.25 р.	Виконано
6.	Виконання етапу впровадження та випуску бета-версії.	30.03.25 р. – 01.04.25 р.	Виконано
7.	Оформлення записки до дипломного проекту.	01.06.25 р. – 10.06.25 р.	Виконано

Студент _____ Гудзеляк Р. Б.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Борецька І. Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота присвячена розробленню вебсайту інтернет-магазину мобільних телефонів. Основна мета полягає у створенні зручного, функціонального і сучасного ресурсу для перегляду, вибору та купівлі мобільних пристроїв. Сайт розроблено з використанням мови програмування C# та фреймворку ASP.NET MVC, що забезпечує швидкість розробки, масштабованість та гнучкість у реалізації логіки.

Під час розробки використано шаблон MVC, шаблонні HTML-сторінки Razor, механізм маршрутизації, базу даних Azure sql та бібліотеки для інтеграції онлайн-оплати й подальшої реалізації авторизації через Google (OAuth 2.0). Передбачено реалізацію панелі адміністратора для керування товарами та категоріями, функціональність знижок, а також автоматичне обрахування цін із урахуванням знижки та кількості.

Ключові слова: ASP.NET MVC, Entity Framework, Azure, Razor.

ABSTRACT

This thesis is about developing a website for an online mobile phone store. The main goal is to create a convenient, functional and modern resource for viewing, selecting and purchasing mobile devices. The site was developed using the C# programming language and the ASP.NET MVC framework, which ensures fast development, scalability, and flexibility in logic implementation.

During the development, we used the MVC template, Razor template HTML pages, a routing mechanism, a Azure sql database, and libraries for integrating online payment and further implementing authorization via Google (OAuth 2.0). The solution includes an admin panel for managing products and categories, discount functionality, and automatic price calculation based on discounts and quantities.

Keywords: ASP.NET MVC, Entity Framework, Azure, Razor.

ТЕХНІЧНЕ ЗАВДАННЯ

Завдання передбачає створення високоякісного інтернет-сайту для онлайнової торгівлі смартфонами, забезпечуючи високий рівень безпеки, стабільності і швидкодії. Для реалізації цього завдання передбачено використання актуальних технологій.

Зберігання інформації здійснюватиметься у реляційній базі даних Azure SQL Server, яка забезпечує надійність та можливість масштабування. Серверна частина буде побудована на основі ASP.NET, що відповідатиме за обробку запитів та реалізацію бізнес-логіки. Для створення структури бази даних буде застосовано підхід Code First з використанням Entity Framework, що дозволить зручно керувати моделями.

Користувацький інтерфейс буде реалізований з використанням Bootstrap, що забезпечить привабливий дизайн та зручну навігацію. Аутентифікація користувачів здійснюватиметься через протокол OAuth 2.0, що гарантує безпечний і простий спосіб входу на сайт.

Для обробки замовлень та надсилання повідомлень користувачам використовуватиметься сервіс SendGrid API. Оплата товарів буде інтегрована через систему LiqPay, що забезпечить безпечні та зручні платіжні операції.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

MVC – Model View Controller (патерн архітектури для поділу логіки, інтерфейсу та даних).

AI – Artificial intelligence - Штучний інтелект.

ORM – Object-Relational Mapping – об'єктно-реляційне відображення.

SQL – Structured Query Language – мова структурованих запитів.

EF Core – Entity Framework Core – ORM для .NET.

UI – User Interface – інтерфейс користувача.

UX – User Experience – користувацький досвід.

UI/UX – це принципи та відповідний процес розробки інтерфейсу користувача для машин та програмного забезпечення при якому досягають максимально зручний та інтуїтивний спосіб їх використання користувачами.

CRUD – Create, Read, Update, Delete – базові операції з даними.

API – Application Programming Interface – інтерфейс прикладного програмування.

OAuth 2.0 – стандарт авторизації з використанням зовнішніх провайдерів (Google, Facebook).

ВСТУП

Сучасний ринок електронної комерції активно розвивається, і все більше компаній переходять до онлайн-продажів, що потребує створення ефективних вебрішень.

У межах дипломного проєкту розроблено вебсайт інтернет-магазину мобільних телефонів, що дозволяє користувачам здійснювати зручний перегляд товарів, додавати їх у кошик, оформлювати замовлення, а також адмініструвати дані про продукцію через зручну панель керування.

Рішення побудовано на базі ASP.NET MVC та Azure SQL Server, що забезпечує масштабованість, гнучкість і підтримку сучасних практик розробки. У роботі реалізовано логіку відображення знижок, підтримку фільтрації, сортування, а також інтеграцію з сервісами електронної пошти (SendGrid) та платіжними системами (LiqPay).

Об’єкт дослідження – процес розробки інтернет-магазину з підтримкою адміністративної частини та функцій користувача.

Предмет дослідження – реалізація логіки керування товарами, кошиком, знижками та оплати онлайн.

Мета роботи – створення зручної, безпечної та функціональної системи продажу мобільних телефонів через вебінтерфейс.

Практичне значення – система може бути використана як основа для повноцінного онлайн-магазину з можливістю масштабування та подальшого розвитку, зокрема — впровадження авторизації через Google (OAuth 2.0) та автоматизованої аналітики продажів.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд проблемної області.

У сучасному цифровому середовищі онлайн-покупки стали невід'ємною частиною повсякденного життя мільйонів людей. Особливо популярними є інтернет-магазини електроніки, зокрема мобільних телефонів, які займають лідируючі позиції серед товарів, що купуються в інтернеті. Користувачі прагнуть швидко, зручно та безпечно знаходити й замовляти актуальні моделі смартфонів, орієнтуючись на технічні характеристики, рейтинг, відгуки та цінову політику продавців.

Разом із цим зростає потреба у розробці сучасних вебсервісів, що можуть забезпечити комфортний досвід взаємодії для кінцевого споживача: зручний інтерфейс, швидку навігацію, надійний механізм пошуку, фільтрації, а також можливість безпечної оплати. Традиційні інтернет-магазини часто не охоплюють усього спектру таких потреб або реалізовані застарілими технологіями, що ускладнює масштабування та розвиток проєкту.

Створення онлайн-магазину мобільних телефонів на основі технологій ASP.NET Core MVC та Azure SQL Server, з використанням Entity Framework Core, дозволяє побудувати гнучке рішення, яке відповідає сучасним вимогам. Хостинг вебдодатку та бази даних у хмарному середовищі Microsoft Azure забезпечує постійну доступність системи, високу продуктивність і безпеку.

Окрім основного функціоналу (перегляд товарів, категоризація, додавання до кошика, замовлення), проєкт передбачає підтримку системи знижок, відображення трендових товарів, фільтрацію та сортування за ціною, категорією й назвою, а також інтеграцію з платіжною системою LiqPay. Для підвищення довіри та зручності користувачів планується впровадження авторизації через Google (OAuth 2.0).

Таким чином, розробка даного вебзастосунку є актуальним завданням, яке поєднує функціональність, масштабованість, безпеку та сучасні інструменти веброзробки, і має значну практичну цінність для кінцевого користувача та бізнесу.

1.2. Сучасні тенденції та виклики

У період стрімкого розвитку цифрових технологій та глобалізації електронної торгівлі ринок мобільних телефонів зазнає постійних змін і вдосконалення. Виробники регулярно випускають нові моделі смартфонів з покращеними характеристиками, що спричиняє високу динаміку попиту й пропозиції. Сучасні покупці прагнуть не лише придбати актуальний гаджет, а й зробити це зручно, вигідно та безпечно.

Одним із ключових викликів є надлишок інформації та моделей, що часто ускладнює прийняття рішення. Користувачам важко зорієнтуватися серед численних технічних характеристик, брендів, версій, знижок і відгуків. Це створює потребу у зручних інструментах фільтрації, сортування та рекомендацій, які допомагають швидко знайти оптимальний варіант.

Ще одним важливим аспектом є конкуренція між онлайн-магазинами. Відвідувачі очікують на сучасний дизайн, швидку роботу сайту, адаптивність до мобільних пристроїв, надійні платіжні системи та автоматизовану підтримку. Недостатня увага до цих деталей може знизити довіру та призвести до втрати клієнтів.

Також зростає уважність до безпеки: користувачі очікують надійного захисту своїх персональних даних і фінансових операцій. Тому інтеграція захищених протоколів автентифікації (наприклад, через Google за допомогою OAuth 2.0), використання перевірених платіжних шлюзів (LiqPay) та розміщення сервісу у хмарному середовищі Azure для високої доступності стають критично важливими елементами проєкту.

Таким чином, тенденції ринку вимагають від розробників створення інтуїтивно зрозумілих, гнучких і безпечних рішень, здатних швидко адаптуватися до змін попиту, забезпечуючи при цьому якісний досвід користувача.

1.3. Роль інетелектуальних систем

У сучасних інтернет-магазинах важливу роль відіграють механізми, що допомагають користувачеві швидко знайти потрібний товар серед великого асортименту. Хоча у даному проєкті не використовується штучний інтелект у повному розумінні цього терміна, реалізовані функціональні можливості забезпечують інтелектуальну взаємодію користувача з системою шляхом застосування фільтрації, сортування, категоризації та пошуку за ключовими словами.

Система надає користувачеві можливість фільтрувати товари за основними характеристиками: ціною, категорією, наявністю знижки, кількістю товару тощо. Це дозволяє значно звузити коло вибору та зосередитися лише на тих товарах, які відповідають критеріям користувача. Наприклад, покупець може обрати лише смартфони в діапазоні до 15 000 грн або переглядати лише моделі, що є в наявності.

Також реалізовано зручне сортування товарів за ціною, назвою або наявністю знижки, що допомагає користувачеві порівнювати пропозиції та приймати обґрунтовані рішення щодо купівлі. Пошукова система сайту дозволяє знаходити телефони за назвою або частиною ключових слів, що значно скорочує час на перегляд усієї вітрини.

1.4. Переваги використання ASP.NET

ASP.NET — це сучасна платформа для створення вебдодатків, розроблена корпорацією Microsoft. Вона широко використовується для розробки вебсайтів різного рівня складності, включаючи комерційні системи, що потребують високої продуктивності, безпеки та масштабованості. У межах проєкту створення інтернет-магазину мобільних телефонів ASP.NET виступає основною технологією для реалізації бекенд-частини системи, обробки запитів, керування логікою бізнес-процесів і взаємодії з базою даних.

Однією з головних переваг ASP.NET є висока продуктивність та масштабованість, що є надзвичайно важливим для електронної комерції. Магазин

повинен обробляти численні запити користувачів, зберігати інформацію про товари, замовлення, оплати та активність клієнтів. ASP.NET дозволяє ефективно обробляти ці дані, забезпечуючи стабільну роботу навіть при високому навантаженні, що гарантує швидку та безперебійну роботу сайту.

ASP.NET також надає потужні засоби безпеки, які критично важливі для онлайн-продажів. Зокрема, платформа має вбудовані механізми захисту від типових загроз, таких як SQL-ін’єкції, XSS (міжсайтове скриптування), CSRF (міжсайтові запити), та інші. Це дозволяє розробникам забезпечити надійний захист персональних і платіжних даних користувачів, що є необхідною умовою для функціонування будь-якого інтернет-магазину.

Ще однією суттєвою перевагою ASP.NET є гнучкість у підключенні зовнішніх сервісів та API, включаючи сервіси оплати (наприклад, Portmone), хмарні сховища (Microsoft Azure), сервіси електронної пошти (SendGrid) та інші. У межах даного проєкту база даних та сам застосунок розміщені на Microsoft Azure, що забезпечує цілодобову доступність, високу надійність і масштабованість інфраструктури.

Також ASP.NET має зручну екосистему розробки, включаючи Entity Framework, що використовується для реалізації підходу Code First, де база даних створюється на основі моделей. Це спрощує оновлення структури БД під час розвитку проєкту. Інструменти Visual Studio, бібліотеки NuGet, шаблони проєктів і велика кількість навчальних ресурсів дозволяють швидко реалізовувати функціональність і легко підтримувати код у майбутньому.

Отже, використання ASP.NET у проєкті дозволяє створити безпечний, продуктивний і масштабований інтернет-магазин мобільних телефонів, готовий до роботи з реальними користувачами в умовах постійної доступності.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Visual Studio 2022

Visual Studio 2022 — це сучасне інтегроване середовище розробки (IDE), створене компанією Microsoft. Це одна з найпотужніших платформ для створення програмного забезпечення, яка підтримує широкий спектр мов програмування і технологій, що дозволяє розробникам створювати різноманітні додатки — від вебсайтів і мобільних застосунків до складних ігор і хмарних сервісів. Завдяки багатому набору інструментів та функцій, Visual Studio 2022 забезпечує високий рівень продуктивності, зручності та ефективності при розробці програмного забезпечення.

Два розширення Visual Studio є важливими для роботи над проектами ASP.NET Core MVC. Перше називається Razor Language Service і забезпечує підтримку IntelliSense для помічників тегів під час редагування подань Razor. Друге називається Project File Tools і забезпечує автоматичне завершення редагування файлів .csproj, що спрощує процес додавання пакетів NuGet до проектів[4].

Основні можливості та цілі використання

Visual Studio 2022 призначене для роботи з різноманітними платформами і технологіями, що робить його універсальним інструментом для розробників у різних галузях. Основні сфери застосування включають:

Розроблення вебзастосунків: За допомогою таких технологій, як ASP.NET, Blazor, та інших вебфреймворків, розробники можуть створювати масштабовані і високопродуктивні вебсервіси та сайти. Це дозволяє створювати сучасні інтерфейси, що легко адаптуються під різні пристрої та браузери. Розробка веб-застосунків часто може бути ітеративним процесом, під час якого ви вносите невеликі зміни до представлень або класів і запускаєте застосунок, щоб перевірити їхній ефект. MVC та Visual Studio працюють разом для підтримки цього ітеративного підходу, щоб зробити перегляд впливу змін швидким і легким[1].

Мобільна розробка: За допомогою платформ Xamarin і .NET Multi-platform App UI (.NET MAUI) можливе створення кросплатформних мобільних додатків для iOS і Android, що дозволяє зменшити час і ресурси на розробку та підтримку окремих версій додатків під різні операційні системи.

Розробка настільних додатків: Підтримка таких технологій, як Windows Presentation Foundation (WPF), Windows Forms і Universal Windows Platform (UWP), дає змогу створювати сучасні, функціональні та зручні у використанні програми для операційної системи Windows.

Розробка ігор: Visual Studio підтримує популярні ігрові движки, такі як Unity і Unreal Engine, що дозволяє створювати високоякісні ігри з використанням потужних інструментів для редактування, тестування і відлагодження.

Хмарні сервіси: Глибока інтеграція з платформою Azure забезпечує можливість створення, розгортання та управління хмарними додатками та сервісами, що особливо актуально для сучасних enterprise-проектів.

Машинне навчання та штучний інтелект: Підтримка мов і інструментів для розробки моделей машинного навчання — Python, R, та інші — дозволяє створювати інтелектуальні системи і рішення на базі штучного інтелекту.

Основні напрямки розробки

Visual Studio 2022 дозволяє створювати широкий спектр додатків та сервісів:

Вебдодатки та сервіси: За допомогою ASP.NET Core, Blazor та інших сучасних вебтехнологій можна швидко створювати високонавантажені сервісні рішення.

Мобільні додатки: Кросплатформна розробка за допомогою Xamarin та .NET MAUI дозволяє створювати мобільні застосунки для Android і iOS з єдиним кодовим базисом.

Настільні додатки: Підтримка WPF, Windows Forms і UWP дозволяє створювати багатофункціональні програми для Windows із сучасним інтерфейсом.

Ігри: Інтеграція з Unity і Unreal Engine дозволяє розробляти високоякісні ігрові проекти.

Хмарні рішення: Глибока інтеграція з Azure відкриває можливості створення, автоматичного розгортання та управління хмарними сервісами.

Мікросервісна архітектура: Підтримка Docker і Kubernetes дозволяє створювати розподілені системи, що є сучасним підходом у розробці великих і масштабованих додатків.

Моделі машинного навчання: За допомогою відповідних інструментів можна проектувати, тренувати та тестувати моделі штучного інтелекту.

Мови програмування та технології

Visual Studio 2022 підтримує широкий набір мов програмування і технологій, що дозволяє розробникам працювати із знайомими інструментами:

C# — основна мова для розробки на платформі .NET, яка широко використовується для створення будь-яких додатків.

Visual Basic і F# — також підтримуються для специфічних цілей та проектів.

C++ — використовується для високопродуктивних системних застосунків, ігор, драйверів та інших ресурсомістких програм.

JavaScript і TypeScript — для фронтенд-розробки і створення інтерактивних вебінтерфейсів.

Python — популярна мова у сферах машинного навчання, наукових обчислень, веброзробки.

Java — підтримка для кросплатформних додатків.

HTML і CSS — для створення інтерфейсів вебдодатків.

SQL — для роботи з базами даних.

R — для статистичних аналізів і обробки даних.

Також підтримуються мови Go, Ruby, PHP через розширення.

Нові та покращені функції Visual Studio 2022

З виходом нової версії IDE було реалізовано ряд покращень, спрямованих на підвищення швидкості роботи, стабільності та зручності використання:

Швидкість і продуктивність: Значні оптимізації внутрішніх процесів забезпечують швидке відкриття проєктів, швидше компілювання та відлагодження.

Підтримка 64-бітної архітектури: Це дозволяє працювати з великими проектами без ризику зниження продуктивності через обмеження пам’яті.

Інтелектуальний редактор коду: Покращений IntelliSense, автоматичне завершення коду, підсвітка синтаксису та можливості рефакторингу спрощують процес написання коду.

Інтеграція з Git: Вбудована підтримка систем контролю версій дозволяє зручно працювати з репозиторіями, управляти гілками та історією змін.

Покращена відладка і діагностика: Потужні інструменти для пошуку і виправлення багів, профілювання продуктивності та аналізу роботи додатків.

Інструменти для автоматизованого тестування: Підтримка різних фреймворків для юніт-тестів і автоматичних сценаріїв тестування.

Живе тестування і відлагодження: Можливість бачити результати змін у реальному часі без повторної компіляції.

Підтримка контейнерів: Інтеграція з Docker забезпечує створення та тестування контейнеризованих застосунків.

Інтуїтивний інтерфейс: Новий дизайн і налаштування зробили роботу з IDE більш зручною і приємною.

2.2. Visual Studio Code

Visual Studio Code (скорочено VS Code) — це безкоштовний, легкий і високопродуктивний редактор коду з відкритим вихідним кодом, розроблений компанією Microsoft. Відмінною рисою даного інструменту є його універсальність: він підтримує роботу з різними мовами програмування і платформами, забезпечуючи

широкий спектр можливостей для розробників будь-якого рівня. На відміну від повнофункціональних IDE, VS Code поєднує у собі функціонал текстового редактора та інструментів, характерних для інтегрованого середовища розробки, що робить його дуже гнучким і зручним у використанні.

Мета та основний напрямок застосування

Основне призначення VS Code — це редагування, налагодження та тестування коду. Його застосовують у різних сферах програмної розробки, що дозволяє створювати широкий спектр додатків та сервісів:

Веброзробка: За допомогою підтримки HTML, CSS, JavaScript, TypeScript та фреймворків, таких як Angular, React, Vue.js, VS Code є ідеальним інструментом для створення сучасних вебдодатків і сайтів. Вбудовані функції автозавершення, підсвітки синтаксису та інтеграція з системами контролю версій суттєво прискорюють процес розробки.

Мобільна розробка: Завдяки підтримці платформ Flutter і React Native, розробники можуть створювати мобільні застосунки для iOS та Android, користуючись однією середовищем і спрощуючи процес підтримки.

Розробка серверних додатків: Підтримка мов програмування, таких як Node.js, Python, Java, C# і багато інших, дозволяє створювати високопродуктивні серверні рішення, API та бекенд-системи.

Хмарні сервіси: Інтеграція з хмарними платформами, зокрема Azure, AWS і Google Cloud, дозволяє розробляти, розгортати та керувати хмарними додатками безпосередньо із редактора.

Машинне навчання і штучний інтелект: Підтримка мов, таких як Python і R, а також інтеграція з Jupyter Notebook, дає можливість працювати з моделями машинного навчання, аналізувати дані і створювати AI-рішення.

Розробка ігор: Завдяки розширенням для Unity, Godot та інших ігрових движків, VS Code стає корисним інструментом для створення ігор із сучасною графікою та складною логікою.

Текстове редагування і обробка файлів: Окрім програмування, він зручний для роботи з різними форматами файлів, скриптами та конфігураційними файлами.

Підтримувані технології та мови

VS Code підтримує широкий набір мов програмування та технологій, що робить його універсальним інструментом:

JavaScript і TypeScript — основні мови для фронтенд-розробки і серверних додатків.

Python — для наукових обчислень, автоматизованих сценаріїв, машинного навчання і веброзробки.

Java — універсальна мова для кросплатформних і серверних додатків.

C# — для розробки під платформу .NET, створення додатків та ігор.

C++ — для системного програмування, високопродуктивних додатків і ігор.

Go — для створення високопродуктивних і масштабованих серверних систем.

PHP, Ruby — для веброзробки та автоматизації.

HTML, CSS — для створення інтерфейсів вебдодатків.

SQL — для роботи з базами даних.

R — для статистичних обчислень і аналізу даних.

Крім того, підтримуються і менш популярні мови через розширення, що робить VS Code гнучким і розширюваним.

Головні функції і переваги

Visual Studio Code — це легкий редактор, який не має всіх функцій повного продукту Visual Studio, але працює на різних платформах і чудово здатний обробляти проекти ASP.NET Core MVC та Entity Framework Core[4].

Використання VS Code забезпечує низку корисних функцій, які підвищують швидкість і зручність процесу розробки:

Легка вага і швидкодія: VS Code швидко запускається і працює навіть на слабких системах, забезпечуючи миттєвий доступ до інструментів і файлів.

Інтелектуальний редактор коду: Завдяки IntelliSense користувач отримує автозавершення коду, підсвітку синтаксису, підказки та автоматичне виправлення помилок, що прискорює написання і знижує кількість помилок.

Інтеграція з системами контролю версій: Вбудована підтримка Git дозволяє керувати гілками, переглядати історію змін і працювати з репозиторіями у редакторі.

Підтримка розширень: Marketplace пропонує тисячі розширень, що додають функціональність — від підтримки нових мов і фреймворків до інтеграції з сторонніми сервісами.

Вбудований термінал: Відкрите і налаштовуване вікно командного рядка дозволяє запускати скрипти і команди без необхідності перемикатися між програмами.

Налаштування середовища: Можливість детально налаштовувати конфігурацію, створювати власні шаблони та сценарії.

Підтримка віддаленої роботи: Працювати з віддаленими серверами, контейнерами або через Windows Subsystem for Linux (WSL) дуже просто.

Інтеграція з хмарними платформами: Інструменти для роботи з Azure, AWS і Google Cloud дозволяють керувати хмарними ресурсами безпосередньо з редактора.

Спільна робота: За допомогою функції Live Share розробники можуть одночасно працювати над одним проектом, обмінюватися кодом і коментувати його у реальному часі.

2.3. ASP.NET MVC

ASP.NET Core MVC — це фреймворк для розробки веб-застосунків від Microsoft, який поєднує в собі ефективність та акуратність архітектури model-view-controller (MVC), ідеї та методи гнучкої розробки та найкращі риси платформи .NET[1]. Цей підхід дозволяє створювати масштабовані, модульні та легко підтримувані вебзастосунки, забезпечуючи розподіл логіки, інтерфейсу та даних.

Розробка на ASP.NET MVC стала логічним продовженням традицій ASP.NET Web Forms, але з більшою гнучкістю, контролем та можливостями для сучасної розробки вебсайтів і сервісів. Завдяки архітектурі MVC, розробники отримують інструменти для чіткої організації коду, зменшення його дублювання і підвищення швидкості розробки.

Архітектура ASP.NET MVC

Основною ідеєю ASP.NET MVC є розподіл функціональності на три основні компоненти:

Model (Модель) — це частина, яка відповідає за дані та бізнес-логіку. Вона зчитує, зберігає і обробля інформацію, що використовуються у додатку. Наприклад, це можуть бути класи, що працюють з базою даних або логіка обчислень.

View (Представлення) — це інтерфейс користувача, який відображає дані, отримані з моделі. View відповідає за візуалізацію і може бути реалізована у вигляді HTML-сторінок, що використовують Razor-синтаксис для вставки даних.

Controller (Контролер) — це посередник між моделлю і видом. Він отримує запити від користувача, обробля їх, викликає відповідні моделі для роботи з даними і передає результати у вигляд для візуалізації.

Ця архітектура забезпечує високий рівень розділення відповідальності, що полегшує підтримку і масштабування проекту.

Основні переваги ASP.NET MVC

Розробка з використанням ASP.NET MVC має низку вагомих переваг:

Гнучкість і контроль — розробник має повний контроль над процесом генерації вебсторінок, їх структурою та логікою. Це особливо важливо для створення складних і високонавантажених систем.

Чітка структура проекту — розподіл на моделі, представлення і контролери дозволяє легко орієнтуватися у коді, швидко знаходити і змінювати потрібні компоненти.

Підтримка тестування — архітектура MVC сприяє створенню юніт-тестів для окремих компонентів, що підвищує якість і надійність додатків.

Розширюваність — легко додавати нові функціональні можливості або змінювати існуючі, не порушуючи роботу всього проекту.

Можливості для SEO — гнучка структура дозволяє створювати чисті URL та зручно налаштовувати пошукову оптимізацію.

Інтеграція з сучасними технологіями — легко поєднується з JavaScript-фреймворками (Angular, React, Vue.js), CSS-стилями, API та іншими компонентами.

Практичне застосування ASP.NET MVC

Отже, ASP.NET Core — це сучасний, гнучкий і легкий фреймворк, ідеально підходящий для створення різноманітних веб-додатків, API та мікросервісів. Він підтримує популярні архітектурні шаблони, такі як MVC і Blazor, що дозволяє розробникам використовувати C# як на стороні сервера, так і на клієнті, забезпечуючи зручність та ефективність у розробці сучасних веб-інтерфейсів[5].

ASP.NET MVC активно використовується у створенні складних корпоративних вебзастосунків, порталів, інтернет-магазинів, систем управління контентом та будь-яких додатків, що вимагають високої гнучкості і масштабованості.

Розробники застосовують цей фреймворк для створення:

Бізнес-систем з багатим функціоналом і високим рівнем безпеки.

Порталів та порталів з інтерактивними сервісами та особистими кабінетами.

Інтернет-магазинів, що мають зручний інтерфейс і складну логіку роботи.

Корпоративних порталів для внутрішнього використання, де важлива інтеграція з іншими системами.

Вебсервісів і REST API, що підтримують мобільні застосунки і сторонні інтеграції.

Інтеграція з іншими технологіями

ASP.NET MVC легко поєднується з сучасними технологіями:

Entity Framework — для роботи з базами даних у режимі ORM.

JavaScript-фреймворки — для створення динамічних і сучасних інтерфейсів.

Bootstrap — для швидкого створення адаптивних і красивих інтерфейсів.

Web API — для створення RESTful сервісів.

SignalR — для реалізації реального часу у вебзастосунках.

Це дозволяє створювати сучасні, високонавантажені системи з високою продуктивністю і високим рівнем безпеки.

Razor-синтаксис

Однією з ключових особливостей у побудові інтерфейсу є використання Razor-синтаксису — це зручна та потужна серверна мова розмітки, яка дозволяє ефективно поєднувати HTML-код із C#-логікою. Razor використовується у представленнях (View) і дає змогу виводити динамічні дані без потреби у складних конструкціях.

Наприклад, через Razor легко реалізується виведення ціни товару зі знижкою, використовуючи формулу:

$$P_{sale} = P \times (1 - \frac{D}{100}) \quad (2.1)$$

P — початкова ціна,

D — відсоток знижки,

P_{sale} — підсумкова ціна зі знижкою.

Приклад формули для обрахунку сумарної кількості товарів в кошику з урахуванням скидки:

$$Total = \sum_{i=1}^{n} \lfloor P_i * (1 - \frac{D_i}{100}) \rceil * Q_i) \quad (2.2)$$

P_i — базова ціна товару,

D_i — відсоток знижки,

Qi — кількість одиниць,

$\lfloor \cdot \rceil$ — округлення (як у Math.Round()),

n — кількість товарів у кошику.

Це основна формула для пагінації (посторінкової навігації) в базі даних або списку елементів:

$$Offset = (p - 1) \times s \quad (2.3)$$

- p — номер поточної сторінки,
- s — кількість елементів на сторінці.

2.4. Entity Framework Core

Entity Framework Core (EF Core) — це сучасна, легка і розширювана ORM (Object-Relational Mapper) бібліотека для платформи .NET. Вона є еволюцією традиційного Entity Framework і призначена для роботи з реляційними базами даних у режимі код-орієнтованого підходу[3]. EF Core дозволяє розробникам працювати з базою даних через об'єктний модельний рівень, абстрагуючись від складностей SQL-запитів і управління схемою бази.

За допомогою об'єктно-реляційного відображальника (ORM) ви можете маніпулювати базою даних, використовуючи об'єктно-орієнтовані концепції, такі як класи та об'єкти, зіставляючи їх з концепціями бази даних, такими як таблиці та стовпці. EF Core базується на існуючих бібліотеках Entity Framework, але відрізняється від них. Вона була створена як частина кросплатформної розробки .NET Core, але з додатковими цілями. Існує багато різних типів баз даних, але, ймовірно, найпоширенішим сімейством є реляційні бази даних, доступ до яких здійснюється за допомогою структурованої мови запитів (SQL)[6].

Основні поняття і архітектура EF Core

EF Core базується на концепції "модель-контекст" (Model-DbContext):

Модель — це набір класів, що відображають структуру таблиць бази даних. Це можуть бути POCO-класи (Plain Old CLR Objects), що описують сутності, їх властивості та стосунки.

DbContext — це основний клас, який зв'язує модель з базою даних і забезпечує доступ до таблиць через властивості типу DbSet<T>. Він відповідає за виконання CRUD-операцій, управління транзакціями і відслідковування змін.

Переваги використання EF Core

EF Core має багато переваг, що робить його популярним серед розробників:

Простота і швидкість: не потрібно писати SQL-запити вручну — достатньо працювати з об'єктами та LINQ.

Підтримка різних баз даних: працює з SQL Server, PostgreSQL, MySQL, SQLite, Oracle і іншими системами.

Міграції бази даних: автоматичне управління схемою бази даних через міграції, що спрощує оновлення та підтримку.

Підтримка LINQ: потужна система запитів у стилі LINQ дозволяє писати вирази, що зручно читаються і зручно виконуються.

Транзакції і управління даними: автоматичне та ручне управління транзакціями для забезпечення цілісності даних.

Підтримка асинхронних операцій: покращена продуктивність за рахунок асинхронних методів.

Практичне застосування EF Core

EF Core застосовується у створенні широкого спектру додатків:

Бізнес-системи — системи управління ресурсами, даними, обліком.

Вебзастосунки — для роботи з даними у бекенд-частині.

Мобільні і десктопні додатки — для локального зберігання та обробки даних.

API-сервіси — для зручної роботи з базою даних через REST API.

Наприклад, у вебдодатку для управління клієнтською базою, EF Core дозволяє легко додавати, оновлювати і видаляти записи, виконуючи запити у стилі LINQ.

Міграції та управління схемою

Одна з ключових функцій EF Core — система міграцій, що дозволяє оновлювати структуру бази даних у процесі розробки без втрати даних або необхідності ручного редагування схеми. Команди CLI або інтеграція з Visual Studio дозволяють створювати, застосовувати і відкатувати міграції.

Безпека і оптимізація

EF Core підтримує такі механізми безпеки і оптимізації:

Параметризовані запити — захист від SQL-ін’єкцій.

Кешування — зменшення кількості запитів до бази.

Оптимізація запитів — автоматичне формування ефективних SQL-запитів.

Логування і налагодження — для аналізу і покращення продуктивності.

Entity Framework Core — що є платформою об'єктно-реляційного відображення (ORM) Microsoft .NET. Платформа ORM представляє таблиці, стовпці та рядки реляційної бази даних через звичайні об'єкти C#[2]. Потужний інструмент для роботи з базами даних у сучасних .NET-застосунках. Його можливості значно спрощують розробку, підвищують швидкість і якість створення систем, знижують ймовірність помилок і дозволяють швидко масштабувати додатки. Завдяки підтримці різних баз даних, системі міграцій і простоті використання EF Core став стандартним рішенням для роботи з даними в екосистемі .NET.

2.5. Bootstrap

Bootstrap — це популярний фреймворк для швидкої розробки адаптивних і сучасних вебінтерфейсів. Його створила команда Twitter у 2011 році, і з того часу він став однією з найпопулярніших бібліотек для фронтенд-розробки. Основною метою Bootstrap є спрощення процесу створення красивих, зручних і мобільних вебсайтів без необхідності глибоких знань CSS та JavaScript.

Bootstrap використовує 12-колонкову систему сітки. Відповідно, ширина кожної колонки визначається так:

$$\text{Ширина колонки } (\%) = \frac{col}{12} \times 100 \qquad (2.3)$$

col — кількість колонок.

Основні компоненти Bootstrap

CSS-компоненти — набір стильових класів для швидкого формування макетів, форм, кнопок, таблиць, карток і інших елементів інтерфейсу.

JavaScript-плагіни — для додавання інтерактивних елементів, таких як модальні вікна, каруселі, випадаючі меню, таби і т.д.

Гнучка сіткова система (Grid System) — дозволяє створювати адаптивний макет сторінки, що автоматично підлаштовується під будь-який розмір екрана.

Шрифти та іконки — інтеграція з наборами іконок (Font Awesome, Bootstrap Icons) для зручного додавання візуальних елементів.

Адаптивність — всі компоненти автоматично підлаштовуються під мобільні, планшетні і десктопні пристрої.

Переваги використання Bootstrap

Швидкість і простота — швидкий старт проекту без необхідності писати з нуля стилі та скрипти.

Адаптивність — автоматичне налаштування макету під будь-який пристрій.

Модульність і гнучкість — можна легко додавати або видаляти компоненти залежно від потреб.

Велика спільнота і документація — широкий вибір прикладів, уроків і підтримка.

Сумісність із сучасними браузерами — Bootstrap підтримує всі популярні браузери.

Практичне застосування

Bootstrap широко використовується у створенні корпоративних сайтів, лендингів, інтернет-магазинів і вебдодатків. Завдяки своїй гнучкості та простоті використання, він дозволяє швидко створювати привабливі, зручні та адаптивні інтерфейси навіть без глибоких знань у фронтенд-розробці.

Є кілька способів налаштувати Bootstrap. Найзручніший варіант залежить від особливостей вашого проекту, складності інструментів збірки, версії Bootstrap, яку ви використовуєте, і підтримки браузерами[14].

Bootstrap — це незамінний інструмент для сучасної веброзробки, що дозволяє швидко і ефективно створювати красиві, функціональні і мобільні сайти. Його

популярність пояснюється простотою, гнучкістю і широкою підтримкою, що робить його однією з головних технологій у арсеналі фронтенд-розробника.

2.6. Microsoft Azure

Azure — це хмарна платформа від компанії Microsoft, розроблена для спрощення процесу створення сучасних додатків та пропонує широкий спектр сервісів для розробки, розгортання та управління додатками і інфраструктурою через глобальну мережу дата-центрів. Azure дозволяє компаніям і розробникам створювати масштабовані і надійні рішення без необхідності власного фізичного обладнання, що особливо актуально в епоху цифрової трансформації[9].

Завдяки широкій підтримці інструментів, які ви вже використовуєте, таких як Visual Studio та Visual Studio Code, а також комплексній бібліотеці SDK, Azure розроблена для того, щоб зробити вас, .NET-розробника, продуктивним з самого початку.

Azure автоматично масштабує застосунок —часто використовується логіка масштабування, це оцінка потрібної кількості серверів (інстансів):

$$\text{Кількість інстансів} = \left\lceil \frac{\text{Потужність 1 інстансу}}{\text{Навантаження}} \right\rceil \quad (2.5)$$

Потужність 1 інстансу - скільки запитів, операцій або користувачів може обробити один сервер за одиницю часу,

Навантаження: загальне очікуване навантаження (наприклад, 1000 запитів за хвилину),

⌈ ... ⌉: математична функція округлення вгору.

Основні компоненти і сервіси Azure

Обчислювальні ресурси: віртуальні машини (Azure Virtual Machines), контейнери (Azure Kubernetes Service), функції безсерверного обчислення (Azure Functions).

Зберігання даних: Blob Storage, File Storage, Data Lake.

Бази даних: Azure SQL Database, Cosmos DB, Database for MySQL і PostgreSQL.

Мережеві рішення: Virtual Network, Load Balancer, VPN Gateway.

Інструменти для розробників: Visual Studio, Azure DevOps, CI/CD pipelines.

Аналітика і машинне навчання: Azure Machine Learning, Power BI.

Безпека і управління: Azure Security Center, Identity and Access Management (IAM).

Переваги використання Azure

Масштабованість: автоматичне або ручне масштабування ресурсів залежно від навантаження.

Глобальна доступність: дата-центри в усьому світі дозволяють створювати глобальні рішення.

Гнучкість: підтримка різних мов програмування, платформ і інструментів.

Безпека і відповідність стандартам: відповідність ISO, GDPR, SOC і іншим стандартам.

Інтеграція з іншими сервісами Microsoft: Office 365, Dynamics 365, Power Platform.

Практичне застосування

Azure широко використовується для хмарної хостінгу додатків, створення масштабованих сервісів, зберігання даних, аналітики та машинного навчання. Наприклад, компанія може розгортати вебсервіси на Azure, використовувати бази даних Cosmos DB для обробки великих обсягів даних, або автоматизувати процеси за допомогою Azure DevOps.

Microsoft Azure — це потужна, гнучка і безпечна платформа для реалізації будь-яких хмарних рішень. Вона дозволяє підприємствам скоротити витрати на інфраструктуру, підвищити гнучкість і швидкість доставки продуктів, а розробникам — швидко створювати і масштабувати сучасні додатки.

2.7. SendGrid API

SendGrid — це хмарна платформа для надсилання електронної пошти, яку використовують для розсилки transactional та маркетингових листів. Вона належить компанії Twilio і широко застосовується для забезпечення високої доставлюваності повідомлень у різних додатках — від вебсайтів і мобільних додатків до

корпоративних систем. API SendGrid дозволяє автоматизувати процес відправки пошти, інтегрувати його у будь-який додаток і керувати розсилками через програмний інтерфейс.

Основні можливості SendGrid API

Відправка транзакційних листів: підтвердження реєстрації, паролі, підтвердження замовлень.

Маркетингові кампанії: масові розсилки, персоналізовані повідомлення, автоматизація.

Шаблони листів: створення і використання динамічних шаблонів для зручності.

Статистика і аналітика: відкриття листів, кліки, відмови.

Обробка подій: webhooks для отримання інформації про статус повідомлень у реальному часі.

Безпека: підтримка DKIM, SPF, DMARC для підвищення доставлюваності.

Переваги використання SendGrid API

Висока доставлюваність: платформа забезпечує доставку листів у поштові скриньки користувачів.

Масштабованість: здатність обробляти великі обсяги листів без збоїв.

Легка інтеграція: підтримка REST API, SDK для різних мов програмування (C#, Python, Java, Node.js тощо).

Автоматизація: можливість налаштовувати тригери і автоматичні розсилки.

Безпека і відповідність: підтримка стандартів безпеки і захисту даних.

Практичне застосування

SendGrid — це веб-API, схоже на REST, яке допомагає вам відправляти багато електронних листів швидко і надійно. Воно дозволяє легко працювати з великими кількостями пошти, будь то транзакційні листи, маркетингові кампанії, управління контактами або відстеження статистики. API має зручні кінцеві точки, які легко

інтегрувати у вашу систему, щоб автоматизувати процеси і працювати ефективніше[12].

SendGrid API — це надійне рішення для автоматизації email-комунікацій у сучасних додатках. Його інтеграція забезпечує підвищену доставлюваність, зручність управління і аналітику, що робить його незамінним інструментом для будь-якого бізнесу, що залежить від електронної пошти.

2.8. OAuth 2.0

OAuth 2.0 — це сучасний протокол авторизації, який дозволяє безпечно надавати третім сторонам доступ до ресурсів користувача без необхідності розкривати паролі. Це широко застосовується для реалізації систем входу через популярні платформи, такі як Google, Facebook, Microsoft та інші, забезпечуючи зручну і безпечну автентифікацію користувачів.

Основні можливості Oauth 2.0:

Авторизація без передачі пароля: користувач може увійти у систему через стороннього провайдера без розкриття своїх облікових даних на сторонніх сервісах.

Токени доступу: для обмеженого часу і з певними правами видаються токени, які використовуються для доступу до захищених ресурсів.

Ролі і права доступу: дозволяє налаштовувати різні рівні доступу для користувачів та систем.

Підтримка різних сценаріїв: Authorization Code, Implicit, Client Credentials та Resource Owner Password Credentials — кожен підходить для специфічних випадків.

Безпечна передача даних: протокол використовує HTTPS для шифрування всіх обмінів даними між клієнтом, сервером авторизації і ресурсами.

Переваги використання OAuth 2.0:

Забезпечує високий рівень безпеки: користувачі не передають свої паролі стороннім сервісам.

Підвищує зручність: можливість входу через популярні облікові записи з мінімальними зусиллями.

Масштабованість і гнучкість: легко інтегрується з різними системами та додатками.

Підтримка стандартів: широко прийнятий і підтримується багатьма платформами і мовами програмування.

Практичне застосування:

OAuth — це спосіб дозволити стороннім додаткам отримати доступ до ваших даних без того, щоб ділитися паролем. Він розділяє роль власника ресурсу і клієнта, тобто того, хто хоче отримати доступ. Коли клієнт (наприклад, додаток або сайт) хоче отримати доступ до захищених даних, він запитує у сервера авторизації дозвіл. Замість того, щоб просити у власника ресурсу пароль, клієнт отримує спеціальний токен — короткий рядок, який дає право на доступ до певних даних на обмежений час. Власник ресурсу спершу дає згоду, і тоді сервер видає цей токен. Потім клієнт використовує його, щоб безпечно отримати доступ до потрібних даних на сервері — без необхідності вводити пароль[10].

2.9. LiqPay API

LiqPay — це популярна платіжна система, яка дозволяє швидко і безпечно здійснювати електронні платежі через інтернет. Вона належить компанії ПриватБанк і широко використовується для інтеграції онлайн-оплати у різних вебдодатках і сервісах.

Формула цифрового підпису, яка використовується для перевірки справжності запиту до платіжної системи LiqPay.

$$Sign = base64\_encode(sha1(private\_key + data + private\_key)) \quad (2.6)$$

private_key - секретний ключ, виданий LiqPay при реєстрації,

data - JSON-рядок з даними платежу, який попередньо закодований у Base6,

+ - конкатенація рядків (з’єднання),

sha1(...): криптографічна хеш-функція SHA-1, яка повертає 160-бітне (20 байт) значення,

base64_encode(...): перетворює байти SHA1 у читабельний формат (рядок).

Основні можливості LiqPay API[11]:

Обробка платежів: можливість приймати платежі за товари і послуги через різні платіжні картки, мобільних операторів і електронні гаманці.

Створення платіжних форм: генерація безпечних форм для внесення платежів, що можна вбудовувати у вебсайти.

Підтримка кількох валют: можливість працювати з різними валютами відповідно до потреб бізнесу.

Автоматичне підтвердження платежів: система надсилає вебхуки або відповіді сервера для підтвердження успішності транзакції.

Захист і безпека: підтримка стандартів PCI DSS, шифрування даних і цифрових підписів для запобігання шахрайству.

Модулі для популярних мов програмування: SDK для PHP, Python, Java, C# та інших, що полегшують інтеграцію.

Переваги використання LiqPay API:

Швидка інтеграція: простий процес налаштування і запуск платіжних форм.

Висока безпека: застосування сучасних протоколів шифрування і підтверджень для захисту даних.

Гнучкість налаштувань: можливість налаштовувати зовнішній вигляд платіжних форм і процесів.

Широкий спектр опцій оплати: підтримка мобільних платежів, банківських карт і електронних гаманців.

Масштабованість: здатність обробляти велику кількість транзакцій без збоїв.

Практичне застосування:

LiqPay API ідеально підходить для автоматичного оброблення онлайн-платежів у магазинах, сервісах бронювання, підписках і будь-яких інших додатках, що вимагають швидкої і безпечної системи оплати. Наприклад, при оформленні замовлення користувачі можуть обрати оплату через LiqPay, і система автоматично генерує платіжну форму або посилання для завершення транзакції. Після успішної оплати API LiqPay надсилає підтвердження, що дозволяє автоматизувати обробку замовлень і оновлення статусів.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Проєктування бази даних

Під час проєктування бази даних для вебзастосунку інтернет-магазину було визначено ключові області, необхідні для забезпечення основної функціональності сайту. Основними вимогами стали: можливість реєстрації та входу користувачів, зокрема через Google-акаунт; перегляд та купівля смартфонів і аксесуарів; пошук і фільтрація товарів за назвою та категорією; а також адміністративне керування товарами.

На основі цих вимог була побудована структура бази даних, яка включає такі основні сутності:

Користувачі – реалізовано через вбудовану систему автентифікації ASP.NET Identity, яка створює відповідні таблиці для зберігання облікових записів, ролей, паролів, зовнішніх логінів тощо.

Продукти (Products) – таблиця зберігає інформацію про товари: назва, опис, ціна, кількість, знижка, посилання на зображення тощо.

Категорії (Categories) – таблиця, що класифікує товари за категоріями, такими як “Смартфони”, “Аксесуари”, “Навушники” тощо.

Певна увага приділялася ролі адміністратора, який має змогу додавати, редагувати та видаляти товари. Це забезпечує актуальність асортименту та зручне керування(Рис 3.1).

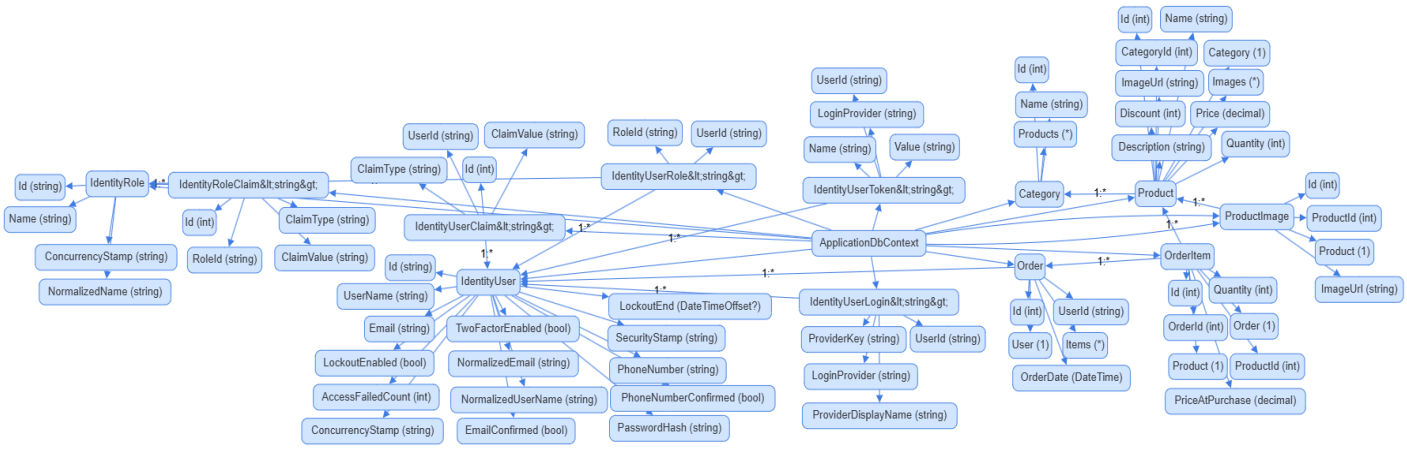


Рисунок 3.1 – Структура бази даних інтернет-магазину.

У процесі реалізації було враховано, що таблиця продуктів є центральною в застосунку, адже саме з нею найчастіше взаємодіють користувачі: переглядають список товарів, фільтрують за категоріями, шукають за ключовими словами. З огляду на це, для підвищення продуктивності було здійснено індексацію ключових полів, зокрема тих, які беруть участь у пошуку та фільтрації. Наприклад, поля назви продукту та ідентифікатора категорії отримали індекси для пришвидшення вибірок. Це дозволило значно скоротити час відповіді сервера під час завантаження списку товарів або застосування фільтрів.

Такий підхід забезпечив не лише швидку та стабільну роботу сайту для кінцевих користувачів, але й ефективне адміністрування товарного каталогу.

3.2. Backend частина проекту

Засобами ASP.NET Core проектувалась backend частина проекту, для кращого розуміння та чистоти коду, проект було зроблено патерном MVC.

Детальніше про наступні розділи дальше. (Рис 3.2)

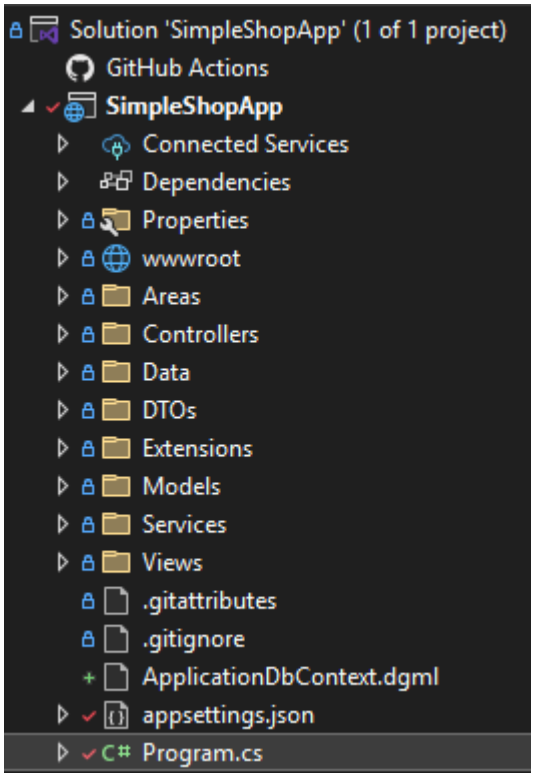


Рисунок 3.2 – Структура проекту.

Цей проєкт виконує ключову роль у забезпеченні взаємодії між вебзастосунком і базою даних. Для ефективного управління даними було використано сучасний ORM-інструмент — Entity Framework Core, який значно спростив реалізацію всіх необхідних функціональних можливостей.

В основі підсистеми доступу до даних лежить підхід **"Code First"**, згідно з яким спочатку створюються класи-моделі у коді, що описують структуру майбутніх таблиць, а сама база даних автоматично генерується на основі цих моделей. Такий підхід забезпечує високу гнучкість під час подальших змін і дає можливість швидко адаптувати структуру даних відповідно до нових вимог.

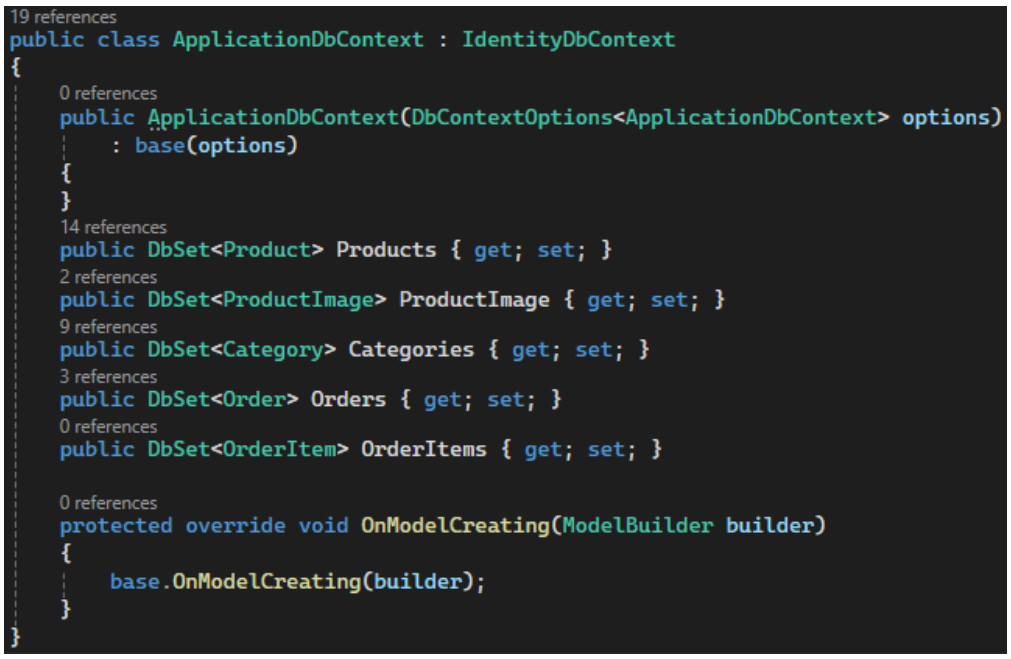
У проєкті реалізовано основні сутності: Product, Category та User. Кожна з них відповідає таблиці у базі даних:

Product містить поля з інформацією про смартфони (назва, опис, ціна, наявність, зображення, тощо).

Category відповідає за класифікацію товарів (смартфони, аксесуари, хіти продажу, тощо).

User є частиною стандартної системи автентифікації ASP.NET Identity і зберігає дані зареєстрованих користувачів.

Уся взаємодія з базою даних здійснюється через спеціальний клас ApplicationDbContext, який успадковується від IdentityDbContext. Він містить колекції DbSet<T> для основних сутностей та конфігурує поведінку моделей за допомогою методу OnModelCreating. Це дозволяє задати всі потрібні зв’язки між таблицями, обмеження, індекси та ініціалізувати початкові дані seed data. (Рис 3.3)

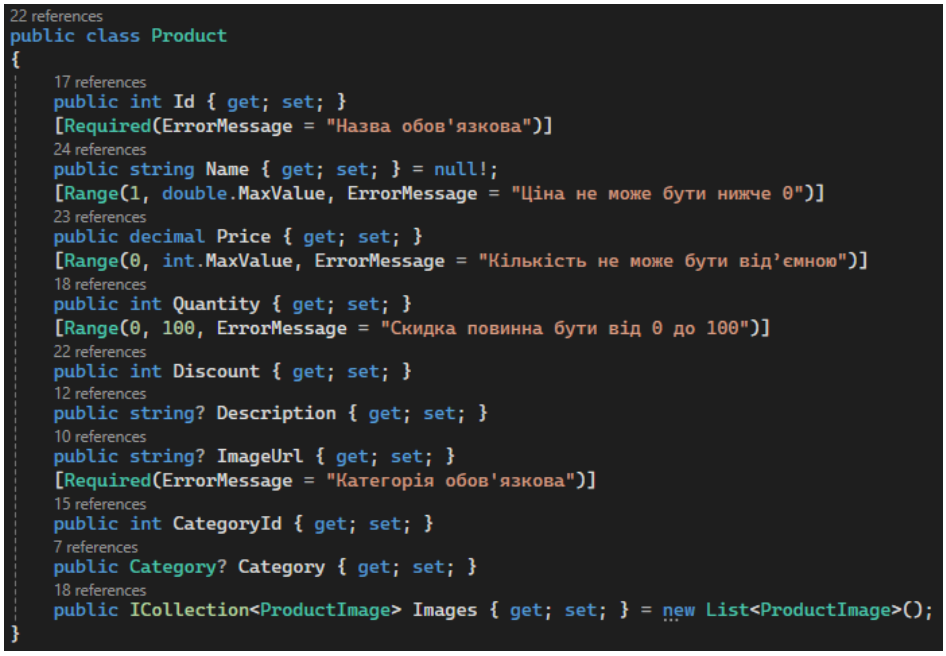


```
19 references
public class ApplicationDbContext : IdentityDbContext
{
    0 references
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
    14 references
    public DbSet<Product> Products { get; set; }
    2 references
    public DbSet<ProductImage> ProductImage { get; set; }
    9 references
    public DbSet<Category> Categories { get; set; }
    3 references
    public DbSet<Order> Orders { get; set; }
    0 references
    public DbSet<OrderItem> OrderItems { get; set; }

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

Рисунок 3.3 – Code First підхід.

Завдяки використанню Entity Framework Core та чіткого розділення відповідальностей, вдалося побудувати гнучку та масштабовану систему для зберігання та обробки даних, що є критично важливою частиною вебзастосунку інтернет-магазину. Такий підхід не лише полегшує супровід проєкту, а й дає можливість легко розширювати функціональність у майбутньому. (Рис 3.4)



```
22 references
public class Product
{
    17 references
    public int Id { get; set; }
    [Required(ErrorMessage = "Назва обов'язкова")]
    24 references
    public string Name { get; set; } = null!;
    [Range(1, double.MaxValue, ErrorMessage = "Ціна не може бути нижче 0")]
    23 references
    public decimal Price { get; set; }
    [Range(0, int.MaxValue, ErrorMessage = "Кількість не може бути від'ємною")]
    18 references
    public int Quantity { get; set; }
    [Range(0, 100, ErrorMessage = "Скидка повинна бути від 0 до 100")]
    22 references
    public int Discount { get; set; }
    12 references
    public string? Description { get; set; }
    10 references
    public string? ImageUrl { get; set; }
    [Required(ErrorMessage = "Категорія обов'язкова")]
    15 references
    public int CategoryId { get; set; }
    7 references
    public Category? Category { get; set; }
    18 references
    public ICollection<ProductImage> Images { get; set; } = new List<ProductImage>();
}
```

Рисунок 3.4 – Product модель.

У проєкту реалізовано всю ключову логіку, що відповідає за обробку даних та виконання бізнес-функцій системи. Саме тут зосереджено найважливіші аспекти роботи із продуктами, категоріями, кошиком та іншими елементами, які безпосередньо впливають на користувацький досвід.

Одним із практичних рішень під час розробки стало оптимальне управління зображеннями товарів. Замість того, щоб зберігати фотографії безпосередньо у базі даних у вигляді byte[], що збільшувало б обсяг і впливало на продуктивність, я реалізував локальне збереження зображень на сервері. У базі даних зберігається лише шлях до зображення, що дозволяє легко й ефективно завантажувати відповідне зображення для відображення у представленнях. Цей підхід спрощує процес завантаження фотографій, економить місце у базі даних і значно покращує швидкодію застосунку.

Виведення зображень реалізовано, зокрема, у представленнях на сторінці Index.cshtml в області вітрини товарів. У цьому файлі для кожного продукту використовується HTML-розмітка з тегом <img>, де значення атрибута src формується динамічно на основі збереженого шляху до зображення.

Однією з функціональних можливостей, реалізованих безпосередньо в Razor, є розрахунок ціни зі знижкою. Це дозволило виводити кінцеву вартість товару прямо в інтерфейсі, не дублюючи логіку у контролерах або бізнес-шарі. Наприклад, у секції з акційними товарами використовується наступна формула. (Рис. 3.5)

```
var discountedPrice = Math.Round(product.Price * (100 - product.Discount) / 100);
```

Рисунок 3.5 – Підрахунок скидки.

Це дозволяє одразу обчислити та відобразити знижку для кожного товару, не перевантажуючи контролер або модель зайвою логікою. Ціна зі знижкою виводиться поруч із перекресленою стандартною ціною, що візуально підкреслює економію. (Рис. 3.6)

Рисунок 3.6 – Код для виводу продуктів зі знижкою.

Для публікації ASP.NET MVC застосунку був створений сервіс Azure App Service (Web App), що забезпечує середовище для розміщення вебсайтів з підтримкою ASP.NET, C# та SQL Server. Процес публікації відбувався безпосередньо з середовища Visual Studio, використовуючи інтеграцію з Azure:

В Visual Studio було обрано опцію "Publish" > "Azure" > "Azure App Service".

Створено новий Web App із вказаною назвою, регіоном та ресурсною групою.

Сконфігуровано параметри підключення до бази даних.

Після успішного завантаження застосунок став доступним за унікальним доменом виду: https://mobitrend.azurewebsites.net/

Azure App Service автоматично створює середовище з підтримкою IIS, .NET SDK, SSL-сертифікатами та автозавантаженням застосунку після перезавантаження. Також було налаштовано автоматичне масштабування залежно від навантаження. Вигляд додатку на Azure Portal Web App(Рис. 3.7)

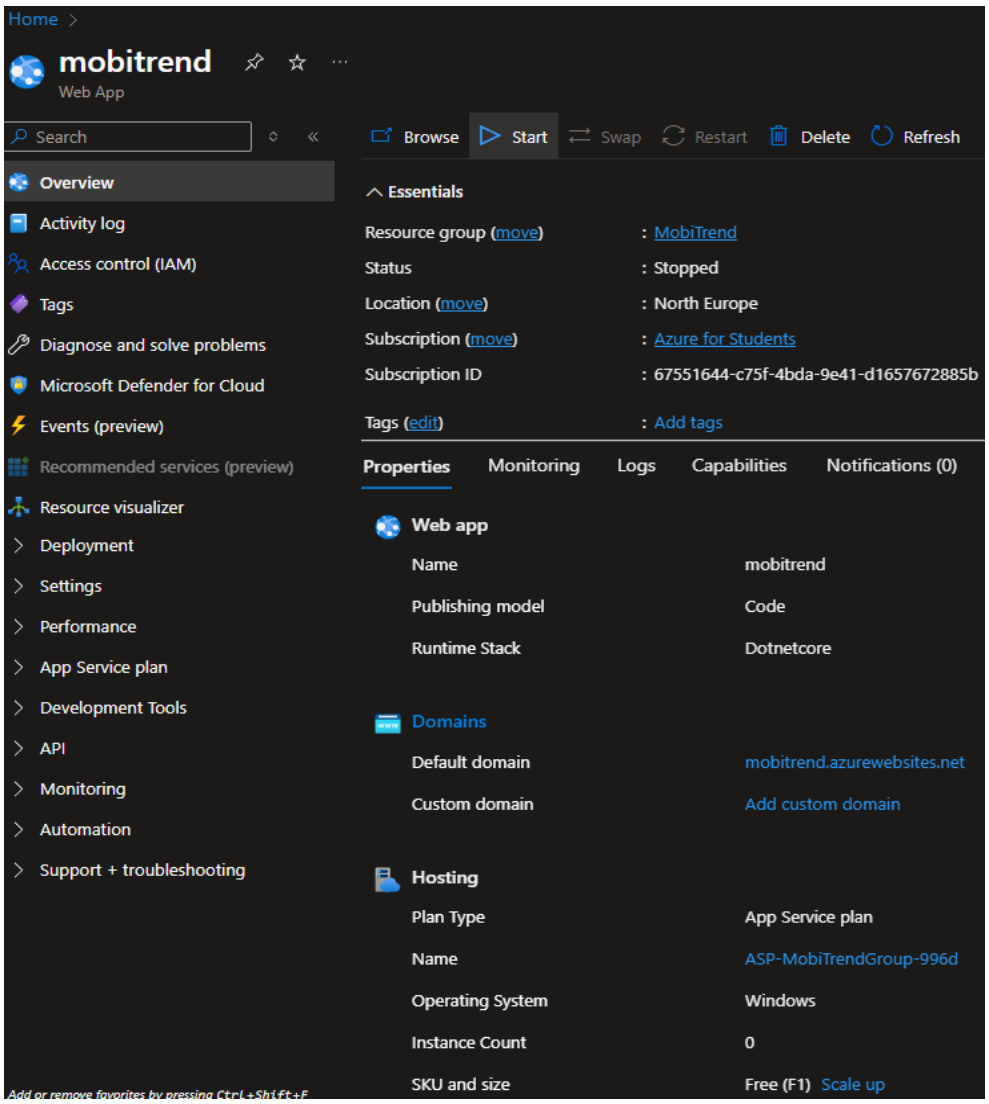


Рисунок 3.7 – Вигляд додатку на Azure Portal Web App.

Щоб забезпечити зручний вхід до системи без реєстрації вручну, в застосунку реалізовано авторизацію через обліковий запис Google з використанням протоколу OAuth 2.0. Одразу після успішної авторизації через Google, у системі виконується перевірка, чи вже існує такий користувач у базі. Якщо ні — створюється новий запис у таблиці Users (або dbo.AspNetUsers, якщо йдеться про Identity).

Це дозволяє зберігати інформацію про кожного авторизованого користувача (ім’я, email, id) і надалі працювати з ним як з зареєстрованим користувачем. (Рис. 3.8)

```
0 references
public IActionResult OnPost(string provider, string returnUrl = null)
{
    var redirectUrl = Url.Page("./ExternalLogin", pageHandler: "Callback", values: new { returnUrl });
    var properties = _signInManager.ConfigureExternalAuthenticationProperties(provider, redirectUrl);
    return new ChallengeResult(provider, properties);
}
0 references
public async Task<IActionResult> OnGetCallbackAsync(string returnUrl = null, string remoteError = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");
    if (remoteError != null)
    {
        ErrorMessage = $"Error from external provider: {remoteError}";
        return RedirectToPage("./Login", new { ReturnUrl = returnUrl });
    }
    var info = await _signInManager.GetExternalLoginInfoAsync();
    if (info == null)
    {
        ErrorMessage = "Error loading external login information.";
        return RedirectToPage("./Login", new { ReturnUrl = returnUrl });
    }
    var result = await _signInManager.ExternalLoginSignInAsync(info.LoginProvider, info.ProviderKey, isPersistent: false, bypassTwoFactor: true);
    if (result.Succeeded)
    {
        _logger.LogInformation("{Name} logged in with {LoginProvider} provider.", info.Principal.Identity.Name, info.LoginProvider);
        return LocalRedirect(returnUrl);
    }
    if (result.IsLockedOut)
    {
        return RedirectToPage("./Lockout");
    }
    else
    {
        ReturnUrl = returnUrl;
        ProviderDisplayName = info.ProviderDisplayName;
        if (info.Principal.HasClaim(c => c.Type == ClaimTypes.Email))
        {
            Input = new InputModel
            {
                Email = info.Principal.FindFirstValue(ClaimTypes.Email)
            };
        }
        return Page();
    }
}
```

Рисунок 3.8 – Підключення Google акаунту.

3.3. Frontend частина проекту

У сучасному веброзробленні зовнішній вигляд і зручність користування є не менш важливими, ніж функціональність самої системи. Фронтенд — це перше, що бачить користувач, саме тому від нього значною мірою залежить загальне враження від сайту.

Під час створення інтерфейсу особливу увагу було приділено поєднанню естетики, логіки взаємодії та доступності. Кожен елемент — кнопка, картка товару чи візуальний блок — спроєктований таким чином, щоб користувач міг інтуїтивно зрозуміти його призначення і виконати дію без додаткових пояснень.

Інтерфейс побудований із розумінням того, що користувачі очікують швидкого доступу до потрібної інформації та зручного механізму взаємодії з функціоналом. Саме тому у фронтенд-частині були реалізовані сучасні UI/UX рішення, що

дозволяють мінімізувати час пошуку, підвищити залученість і спростити процес покупки.

На головній сторінці реалізовано слайдер зі знижками, який динамічно демонструє акційні товари. Це рішення дозволяє одразу звернути увагу на найвигідніші пропозиції та швидко додати товар до кошика. Користувач бачить назву товару, зображення, стару ціну, нову ціну після знижки та кнопку для додавання до кошика. (Рис. 3.9)

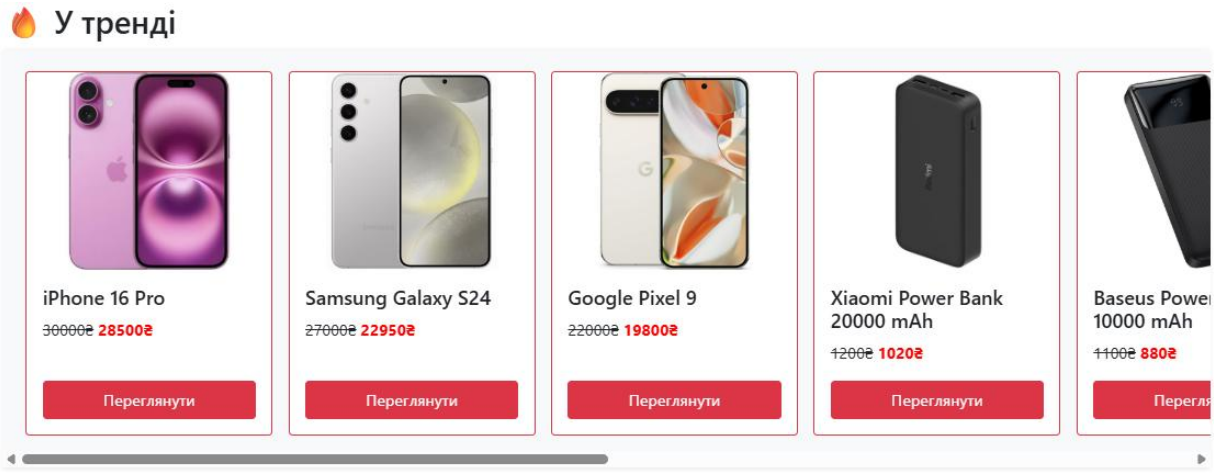


Рисунок 3.9 – Слайдер.

Клік по картці товару може перенаправити користувача на окрему сторінку з можливістю оформити покупку. Таким чином, акційний блок не лише виконує рекламну функцію, а й забезпечує зручну взаємодію з основним функціоналом сайту. (Рис. 3.10)

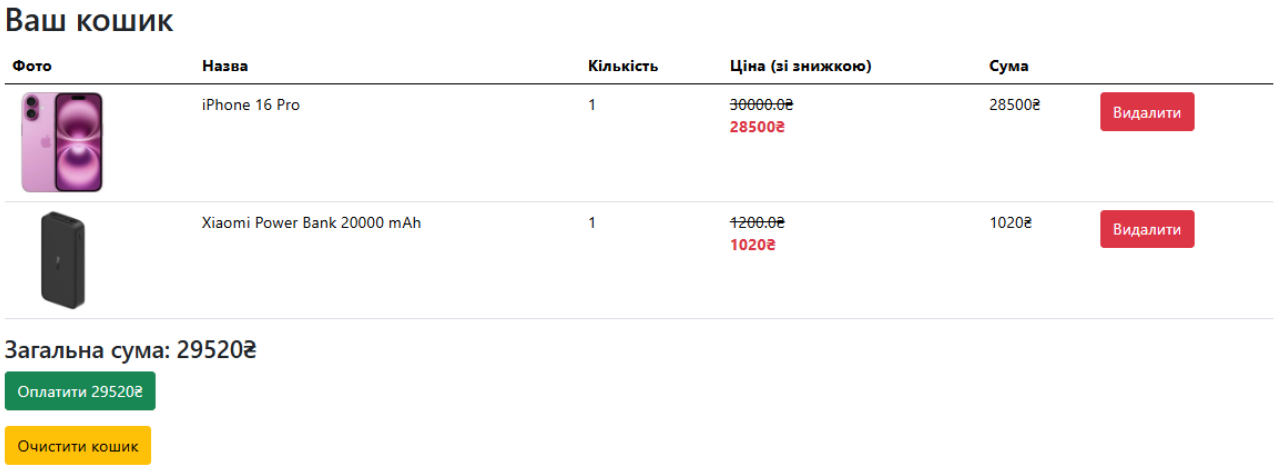


Рисунок 3.10 – Кошик

Сторінка оплат виглядає наступним чином зроблена з допомогою LiqPay. (Рис. 3.11)

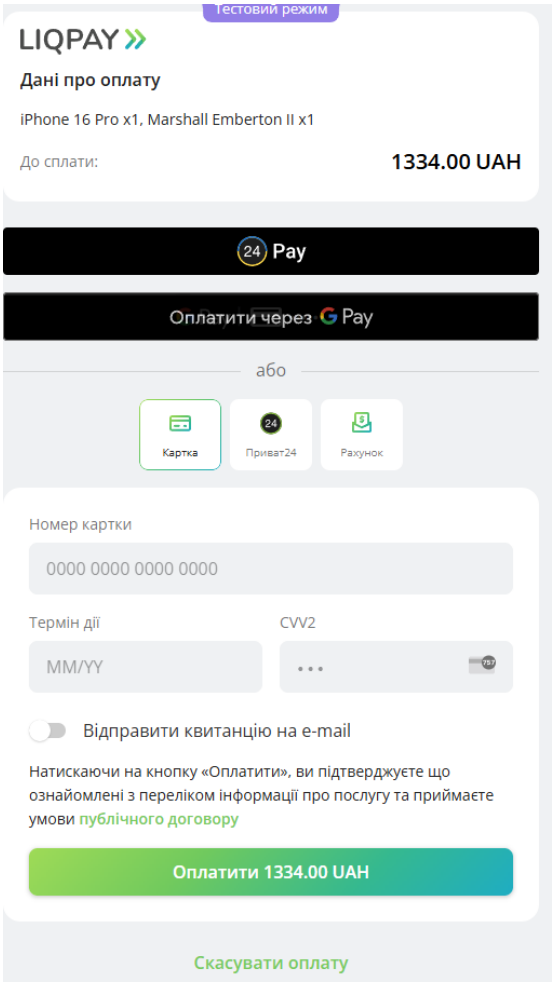


Рисунок 3.11 – Сторінка оплати.

Реалізовано авторизація за допомоги OAuth 2.0, тобто можна логуватись через свій гугл аккаунт або прив’язати гугл акаунт до уже зареєстованого користувача. (Рис. 3.12)

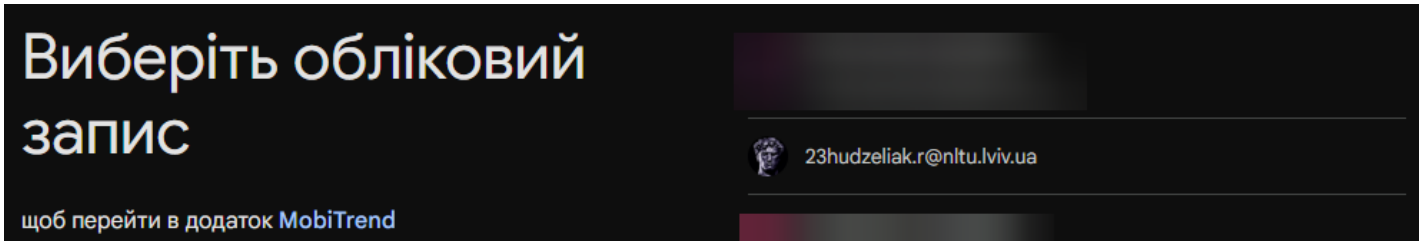


Рисунок 3.12 – Oauth 2.0.

Також, для зручності користувачів, добавлено вибірка по категоріях, пошук по назві, та сортування. (Рис 3.13)

Рисунок 3.13 – Фільтри.

Для зручності користувача в інтерфейсі реалізовано сортування товарів за різними критеріями дозволяє обрати, як саме мають відображатися товари: за назвою (в порядку зростання або спадання) або за ціною. Це дає змогу швидко зорієнтуватися у великій кількості позицій та знайти найактуальніші пропозиції відповідно до особистих уподобань. (Рис. 3.14)

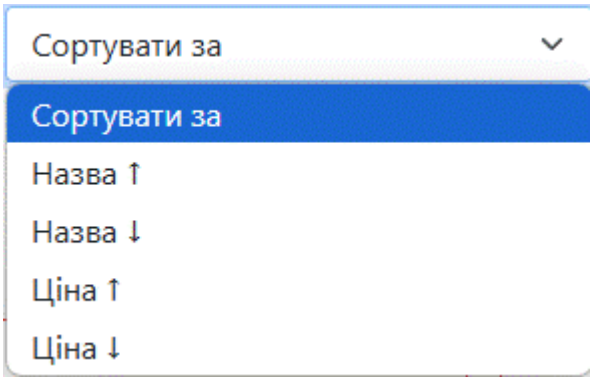


Рисунок 3.14 – Випадаючий список сортування.

Крім сортування, реалізована також фільтрація товарів за категоріями. Список категорій автоматично підтягується з бази даних, що розміщена в Azure SQL. Це гарантує динамічне оновлення фільтрів у разі змін у базі, без необхідності ручного редагування інтерфейсу. Завдяки цьому користувач може легко обрати які переключають сторінку. (Рис 3.15)

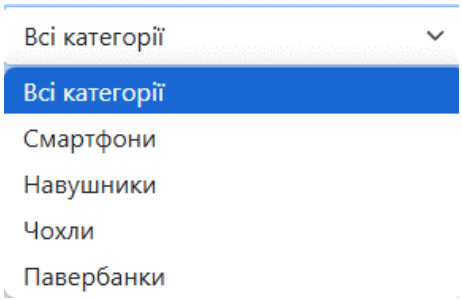


Рисунок 3.15 – Вигляд категорій.

У системі реалізована повноцінна форма реєстрації нового користувача, що є ключовим етапом доступу до персоналізованого функціоналу платформи. Форма складається з трьох основних полів: введення електронної пошти, пароля та

підтвердження пароля. Для зручності користувача передбачено також можливість швидкої авторизації через Google, що суттєво скорочує час реєстрації та зменшує кількість ручного введення даних. (Рис. 3.16)

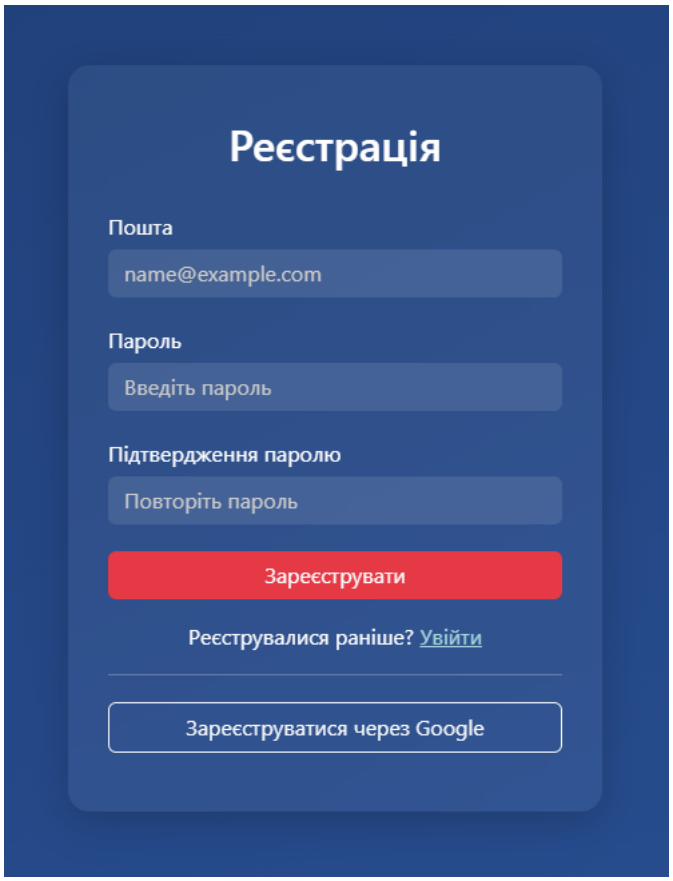


Рисунок 3.16 – Сторінка реєстрації.

Важливою деталлю є наявність валідації: система перевіряє, чи відповідає введений пароль вимогам безпеки, а також чи співпадають поля «Пароль» та «Підтвердження паролю». У разі помилки користувач отримає відповідне повідомлення, що робить процес реєстрації прозорим і зрозумілим. Завдяки чистому дизайну та інтуїтивній структурі, форма реєстрації є не лише візуально привабливою, а й максимально зручною у використанні, що формує позитивне перше враження від взаємодії з системою.

Для реалізації функціоналу реєстрації, авторизації та управління користувачами у проєкті використано ASP.NET Identity — потужну вбудовану систему аутентифікації та авторизації в середовищі ASP.NET. Цей фреймворк дозволяє безпечно зберігати облікові записи користувачів, хешувати паролі, керувати ролями,

підтвердженням електронної пошти та авторизацією через зовнішні провайдери, такі як Google, Facebook тощо. У даному випадку також реалізовано реєстрацію через Google OAuth, що спрощує вхід для користувачів.

Завдяки ASP.NET Identity було мінімізовано потребу в ручному написанні складного коду для безпеки користувацьких даних, а також забезпечено масштабовану й розширювану інфраструктуру для подальшого розвитку системи. Авторизація інтегрована з ASP.NET Identity, що забезпечує безпечну перевірку облікових даних, керування токенами, підтримку cookies, зовнішніх провайдерів і двофакторної аутентифікації при потребі. Усі дії супроводжуються валідацією введених даних, а у разі помилок користувач отримує відповідні повідомлення.

Такий підхід до організації процесу входу забезпечує як безпеку, так і зручність користувачів платформи. (Рис. 3.17)

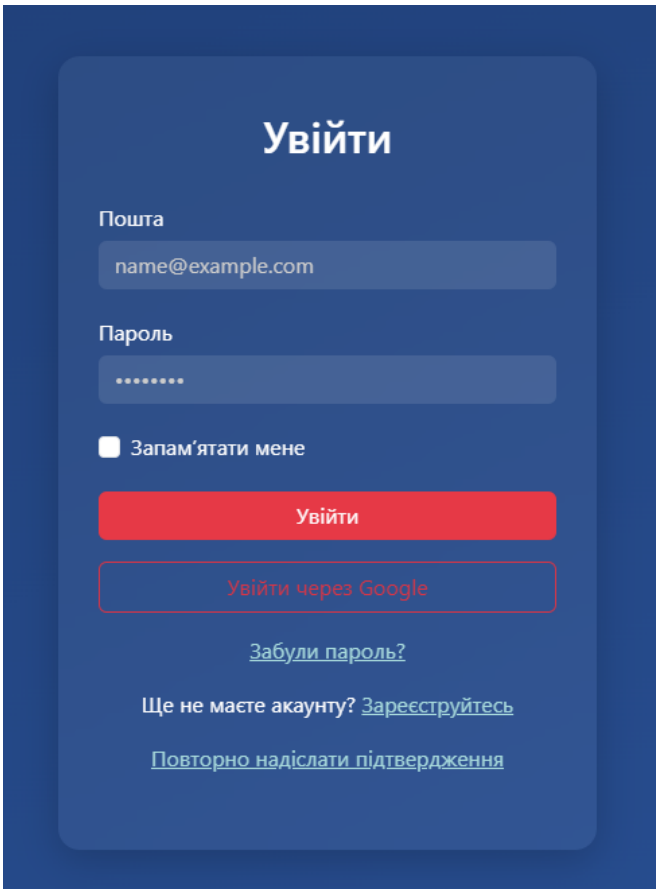


Рисунок 3.17 – Сторінка авторизації.

Після успішної реєстрації користувача система автоматично надсилає лист із підтвердженням електронної пошти. Для реалізації цієї функціональності інтегровано сервіс SendGrid, що забезпечує надійне та швидке надсилання електронних листів.

Після створення облікового запису система генерує унікальний токен підтвердження, який вбудовується у спеціальне посилання. Цей токен разом із адресою користувача передається до шаблону листа, сформованого в HTML-форматі, та надсилається на вказану при реєстрації пошту. Користувач має перейти за отриманим посиланням для підтвердження адреси електронної пошти.

Формат повідомлення адаптований для зручного перегляду як у десктопних, так і мобільних поштових клієнтах. У разі, якщо користувач не отримав листа або випадково його видалив, на сторінці входу реалізована можливість повторної відправки листа для підтвердження.

Завдяки використанню SendGrid, гарантовано високу доставлюваність листів, підтримку логування, а також можливість масштабування при зростанні кількості користувачів. Такий підхід відповідає сучасним вимогам безпеки та дозволяє уникнути створення фальшивих або анонімних облікових записів. (Рис. 3.18)

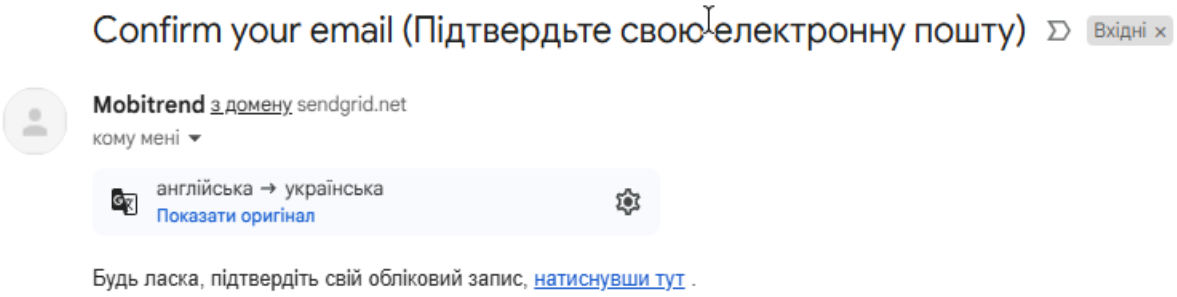


Рисунок 3.18 – Сторінка підтвердження.

На додаток, у вебдодатку реалізовано адміністративну панель, яка забезпечує ефективне керування товарами інтернет-магазину. Адміністратор має доступ до окремого інтерфейсу, спеціально створеного для управління асортиментом продукції. Через цей інтерфейс адміністратор може додавати нові товари, редагувати існуючі записи та видаляти непотрібні елементи з каталогу.

Для зручності взаємодії було створено таблицю з переліком товарів, що дозволяє швидко отримати повну інформацію про кожен продукт. У таблиці відображаються такі характеристики: фото товару, його назва, категорія, кількість на складі, ціна, розмір знижки, ціна зі знижкою, а також короткий опис. Для кожного товару передбачені кнопки «Редагувати» та «Видалити», які дають змогу адміністратору оперативно вносити зміни або вилучати товар із бази.

При розробці цієї панелі було використано комбінацію технологій Bootstrap для адаптивного та зручного інтерфейсу, а також ASP.NET MVC з Razor Pages для забезпечення динамічної генерації сторінок та взаємодії з серверною частиною. Для спрощення кодування та покращення повторного використання елементів інтерфейсу були створені універсальні компоненти для відображення та редагування товарів, що значно полегшує підтримку та розширення функціоналу.

Такий підхід забезпечує адміністраторам зручний, інтуїтивно зрозумілий інструмент для управління всіма аспектами товарного каталогу магазину, від додавання нових позицій до оперативного оновлення наявної інформації (Рис. 3.19).

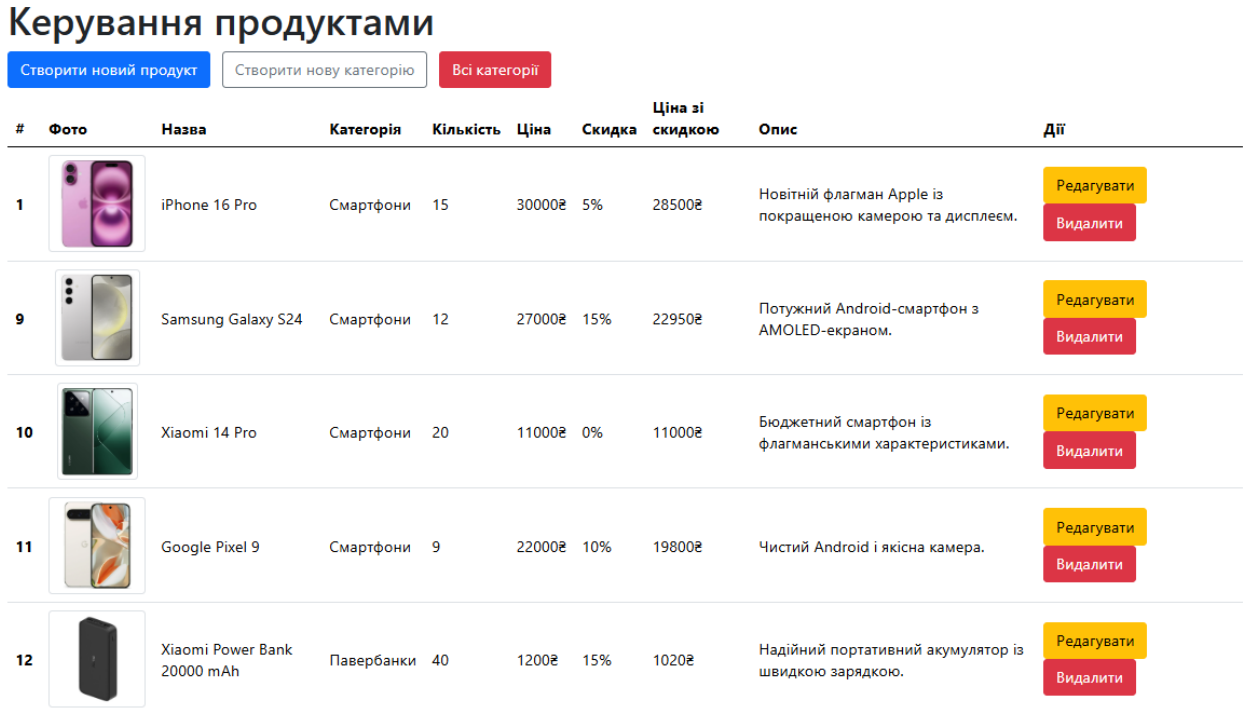


Рисунок 3.19 – Адміністративна панель.

При редагуванні запису, елемент редагується в базі даних та автоматично повертає оновлений елемент. У нашому випадку це продукт. Щоб не робити оновлення таблиці вручну, все оновляється самостійно, що дає більш зручний інтерфейс. (Рис. 3.20)

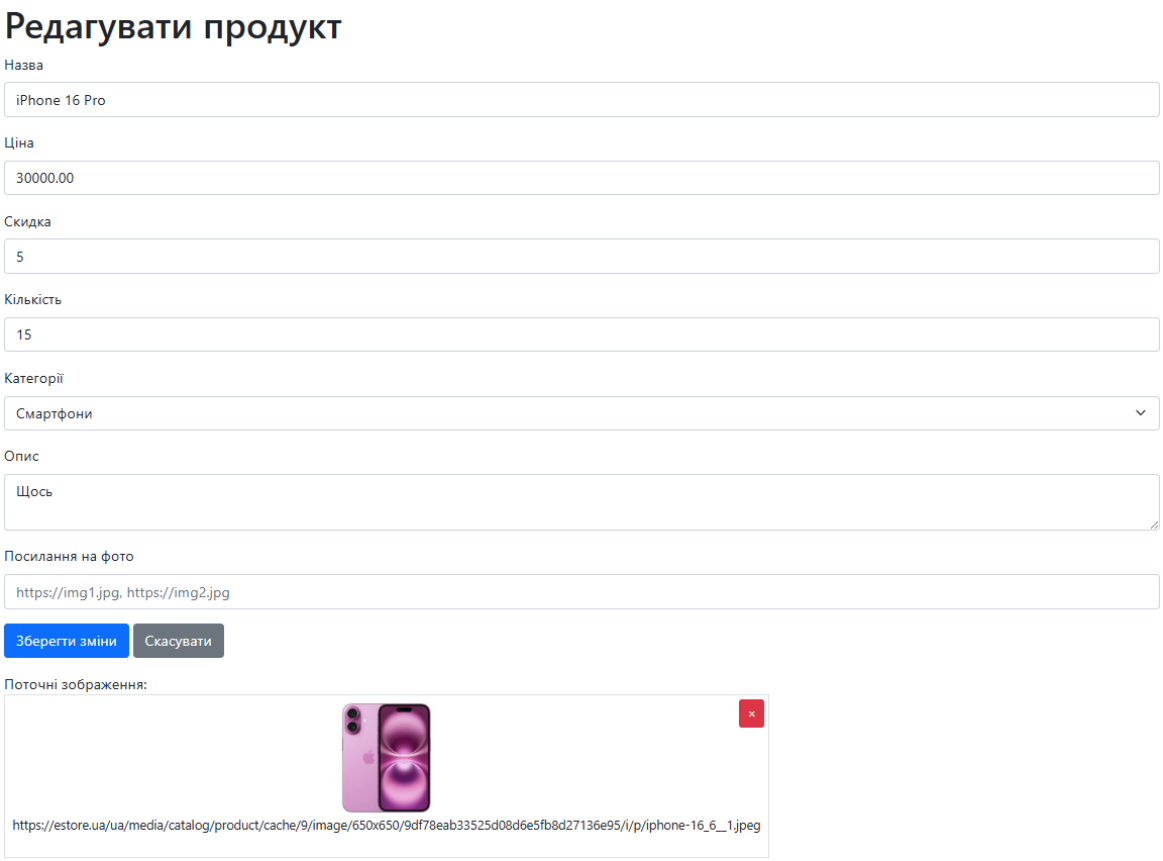


Рисунок 3.20 – Вікно редагування продукту.

Форма додавання нового продукту на сайті — це функціональний інтерфейс, що дозволяє адміністраторам швидко та зручно додавати нові товари до каталогу інтернет-магазину. Цей інструмент представлений у вигляді інтерактивної вебформи з полями для заповнення ключової інформації про товар.

Використання такої форми значно спрощує процес керування асортиментом продукції. Адміністратору не потрібно взаємодіяти безпосередньо з базою даних або використовувати складні адміністративні панелі — додати новий продукт можна за кілька кліків. Форма містить обов’язкові поля для введення назви товару, його ціни, розміру знижки, кількості одиниць на складі, посилання на фотографію продукту,

категорії товару та необов’язкового опису. Така структура форми забезпечує повноту та уніфікацію даних, що зберігаються у базі даних магазину.

Важливою перевагою є наявність зручної кнопки «Додати», яка автоматично створює новий запис у базі та відображає його у загальному списку товарів без необхідності перезавантаження сторінки. Це сприяє підвищенню швидкості роботи адміністратора та зменшує час обслуговування контенту магазину. Додатково передбачено кнопку «Назад», яка дає змогу адміністратору швидко повернутися до головної сторінки керування товарами без втрати даних.

Форма була реалізована за допомогою технологій Bootstrap для забезпечення адаптивного та зручного дизайну, а також ASP.NET MVC та Razor Pages для обробки серверної логіки та збереження даних у базі. Запровадження цієї форми дозволило створити інтуїтивно зрозумілий і функціональний інструмент для адміністраторів магазину. Це сприяє покращенню загальної ефективності управління товарним каталогом та забезпечує більш гнучку роботу з асортиментом.

Для реалізації даної форми було розроблено окремий компонент, який відповідає за створення нового товару та обробку відповідних даних (Рис. 3.20).

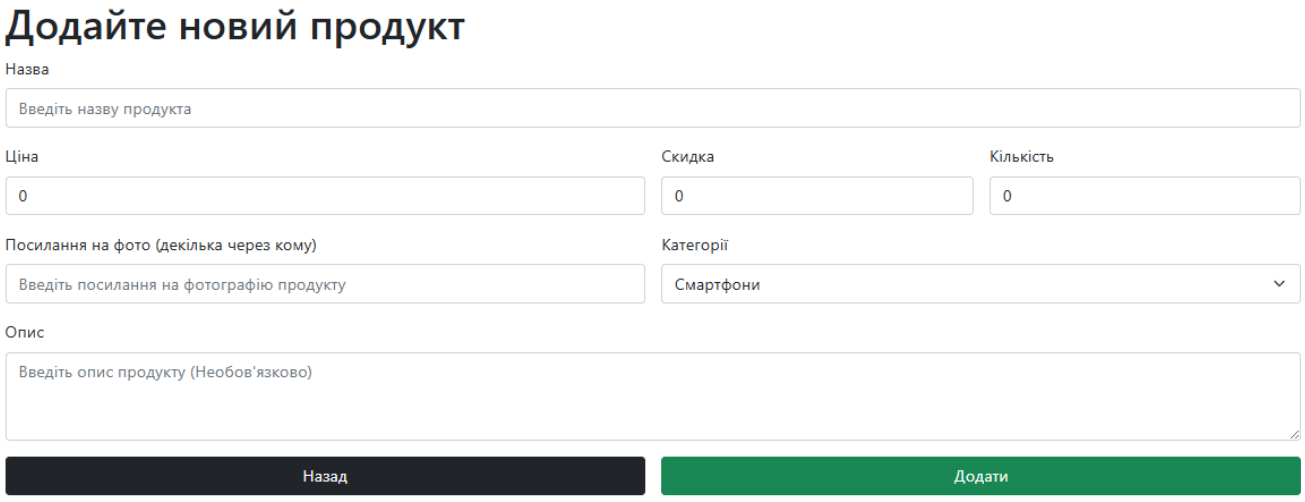


Рисунок 3.20 – Вікно додавання продукту.

ВИСНОВКИ

У процесі розробки вебсайту для купівлі смартфонів я використав сучасний стек технологій: SendGrid для автоматичного надсилання підтверджень пошти, LiqPay для зручної і безпечної оплати товарів, ASP.NET у парі з Razor для створення динамічного і швидкого сервісу, а також Azure для хостингу сайту і зберігання бази даних. Для забезпечення безпеки і зручності користувачів я інтегрував OAuth 2 для авторизації через Google, що дозволяє швидко і безпечно входити до системи.

Основною метою було створити інтуїтивно зрозумілий і зручний вебсайт, що сприяє швидкому пошуку та купівлі смартфонів. Впроваджені сучасні технології забезпечують високий рівень безпеки, стабільності та швидкодії платформи. Автоматичні розсилки підтверджень через SendGrid підвищують довіру користувачів і сприяють ефективній комунікації, а інтеграція LiqPay забезпечує простий і безпечний процес оплати.

Застосування Azure дозволяє легко масштабувати сайт відповідно до зростаючих потреб, а використання Razor і ASP.NET дає змогу швидко оновлювати і підтримувати функціонал. Інтеграція OAuth 2 з Google зробила процес авторизації швидким і зручним, що позитивно впливає на досвід користувачів.

Загалом, створений мною сайт є сучасним, надійним і ефективним інструментом для покупки смартфонів, що враховує потреби користувачів і використовує передові технології для забезпечення високої якості сервісу. Це дає можливість користувачам легко, швидко і безпечно здійснювати покупки, отримуючи при цьому високий рівень комфорту і задоволення від роботи з платформою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. **Freeman A.** *Pro ASP.NET Core MVC.* Sixth Edition, 2016. — 1018 с
2. **Freeman A.** *Pro ASP.NET Core MVC 2.* Apress, 2017. — 1018 с.
3. **Price J.** *C# 12 and .NET 8 – Modern CrossPlatform Development Fundamentals.* Eighth Edition, 2023. — 1098 с
4. **Freeman A.** *Essential Angular for ASP.NET Core MVC 3.* Second Edition, 2019. — 351 с
5. **Albahari J.** *C# 12.0 in a Nutshell.* The Definitive Reference, 2023. — 1083 с
6. **Lock A.** *ASP.NET Core in Action.* Third Edition, 2023. — 984 с
7. Microsoft Docs. *ASP.NET Core Documentation.* URL: https://learn.microsoft.com/en-us/aspnet/core/ (дата звернення: 10.06.2025).
8. Microsoft Docs. *C# Documentation.* URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 10.06.2025).
9. Microsoft Docs. *Azure Documentation.* URL: https://learn.microsoft.com/en-us/azure/ (дата звернення: 10.06.2025).
10. Auth0 Docs. *OAuth 2.0 and OpenID Connect.* URL: https://auth0.com/docs/authenticate/protocols/oauth(дата звернення: 10.06.2025).
11. LiqPay API Documentation. URL: https://www.liqpay.ua/doc(дата звернення: 10.06.2025).
12. SendGrid Docs. *SendGrid API Documentation.* URL: https://docs.sendgrid.com/api-reference/ (дата звернення: 10.06.2025).
13. Microsoft Docs. *Razor Pages Documentation.* URL: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/ (дата звернення: 10.06.2025).
14. Bootstrap Documentation. URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/(дата звернення: 10.06.2025).

ДОДАТКИ

Program.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using SimpleShopApp.Services;
using WebApplication1.Data;
using Microsoft.AspNetCore.Identity.UI.Services;
using SimpleShopApp.Services.CartService;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Authentication.Google;
var builder = WebApplication.CreateBuilder(args);
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new
InvalidOperationException("Connection string 'DefaultConnection' not found.");
builder.Services.AddDbContext<ApplicationDbContext>(options =>
   options. UseSqlServer(connectionString));  // UseSqlite
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
builder.Services.AddDefaultIdentity<IdentityUser>(options =>
{
options.SignIn.RequireConfirmedAccount = true ;
options.User.RequireUniqueEmail = true;
})
   .AddRoles<IdentityRole>()
   .AddEntityFrameworkStores<ApplicationDbContext>();
builder.Services.AddControllersWithViews();
builder.Services.AddAuthentication()
.AddCookie()
.AddGoogle(options =>
{
   IConfigurationSection googleAuthNSection =
      builder.Configuration.GetSection("Authentication:Google");

   options.ClientId = googleAuthNSection["ClientId"];
   options.ClientSecret = googleAuthNSection["ClientSecret"];
});
builder.Services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();
builder.Services.AddTransient<Microsoft.AspNetCore.Identity.UI.Services.IEmailSender, EmailSender>();
builder.Services.AddScoped<ICartService, CartService>();
builder.Services.AddScoped<IOrderService, OrderService>();
builder.Services.AddSession();
var app = builder.Build();
if (app.Environment.IsDevelopment())
{
   app.UseMigrationsEndPoint();
}
else
{
   app.UseExceptionHandler("/Home/Error");
   app.UseHsts();
}
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseSession();
app.UseAuthentication();
app.UseAuthorization();
app.MapControllerRoute(
```

```
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
using (var scope = app.Services.CreateScope())
{
    var roleManager = scope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole>>();
    var roles = new[] { "Admin", "Manager", "Member" };
    foreach (var role in roles)
    {
        if (!await roleManager.RoleExistsAsync(role))
            await roleManager.CreateAsync(new IdentityRole(role));
    }
}

using (var scope = app.Services.CreateScope())
{
    var userManager = scope.ServiceProvider.GetRequiredService<UserManager<IdentityUser>>();
    string email = "admin@admin.com";
    string password = "Qwerty12!";
    if (await userManager.FindByEmailAsync(email) == null)
    {
        var user = new IdentityUser()
        {
            UserName = email,
            Email = email,
            EmailConfirmed = true
        };
        await userManager.CreateAsync(user, password);

        await userManager.AddToRoleAsync(user, "Admin");
    }
}
app.MapRazorPages();
app.Run();
```

appsettings.json

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=tcp:mobitrend.database.windows.net,1433;Initial Catalog=Mobitrenddb;Persist Security
Info=False;User
ID=rosa;Password=Qwerty12;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection
Timeout=30;"
    //"DefaultConnection": "Data Source=app.db"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "Authentication": {
    "Google": {
      "ClientId": "345131131816-n0p9ie13rb0v9i7e5i9vak2t526u7ghh.apps.googleusercontent.com",
      "ClientSecret": "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"
    }
  }
}
```

Areas/Identity/Pages/Account

ConfirmEmail.cshtml

```
@page
@model ConfirmEmailModel
@{
    Layout = null;
    ViewData["Title"] = "Підтвердження пошти";
}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>@ViewData["Title"]</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site-custom.css" />
</head>
<body>
    <div class="login-card">
        <h2>@ViewData["Title"]</h2>
        <hr />
        @if (Model.StatusMessage != null)
        {
            var isSuccess = Model.StatusMessage.Contains("success", StringComparison.OrdinalIgnoreCase);
            <p class="@(isSuccess ? "status-success" : "status-error")">
                Дякую за підтвердження електронної пошти
            </p>
        }
        <div class="mt-3">
            <a asp-area="Identity" asp-page="/Account/Login"> Увійти </a>
        </div>
    </div>
</body>
</html>
```

Login.cshtml

```
@page
@model SimpleShopApp.Areas.Identity.Pages.Account.LoginModel
@{
    Layout = null;
    ViewData["Title"] = "Log in";
}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>@ViewData["Title"]</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site-custom.css" />
</head>
<body>
    <div class="login-card">
        <h2>Увійти</h2>
        <form method="post">
            <input type="hidden" asp-for="ReturnUrl" />
```

```
        <div asp-validation-summary="ModelOnly" class="text-danger" role="alert"></div>
        <div class="form-group">
          <label asp-for="Input.Email">Пошта</label>
          <input asp-for="Input.Email" class="form-control" placeholder="name@example.com" />
          <span asp-validation-for="Input.Email" class="text-danger"></span>
        </div>
        <div class="form-group">
          <label asp-for="Input.Password">Пароль</label>
          <input asp-for="Input.Password" class="form-control" placeholder="••••••••" />
          <span asp-validation-for="Input.Password" class="text-danger"></span>
        </div>
        <div class="form-check mb-3">
          <input asp-for="Input.RememberMe" class="form-check-input" />
          <label asp-for="Input.RememberMe" class="form-check-label">Запам’ятати мене</label>
        </div>
        <button type="submit" class="btn btn-primary w-100 mb-3">Увійти</button>
      </form>
      <form method="post" asp-page-handler="ExternalLogin" asp-route-provider="Google" asp-route-
returnUrl="@Model.ReturnUrl">
        <button type="submit" class="btn btn-outline-danger w-100">
          <i class="bi bi-google"></i> Увійти через Google
        </button>
      </form>
      <div class="text-center mt-3">
        <p><a asp-page="./ForgotPassword">Забули пароль?</a></p>
        <p>Ще не маєте акаунту? <a asp-page="./Register" asp-route-
returnUrl="@Model.ReturnUrl">Зареєструйтесь</a></p>
        <p><a asp-page="./ResendEmailConfirmation">Повторно надіслати підтвердження</a></p>
      </div>
    </div>
</body>
</html>
```

Register.cshtml

```
@page
@model RegisterModel
@{
    Layout = null;
    ViewData["Title"] = "Register";
}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>@ViewData["Title"]</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site-custom.css" />
</head>
<body>
    <div class="login-card">
      <h2>Реєстрація</h2>
      <form id="registerForm" asp-route-returnUrl="@Model.ReturnUrl" method="post">
        <div asp-validation-summary="ModelOnly" class="text-danger mb-3" role="alert"></div>
        <div class="form-group">
          <label asp-for="Input.Email">Пошта</label>
          <input asp-for="Input.Email" class="form-control" autocomplete="username" placeholder="name@example.com"
/>
```

```
            <span asp-validation-for="Input.Email" class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="Input.Password">Пароль</label>
            <input asp-for="Input.Password" class="form-control" autocomplete="new-password" placeholder="Введіть
пароль" />
            <span asp-validation-for="Input.Password" class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="Input.ConfirmPassword">Підтвердження паролю</label>
            <input asp-for="Input.ConfirmPassword" class="form-control" autocomplete="new-password"
placeholder="Повторіть пароль" />
            <span asp-validation-for="Input.ConfirmPassword" class="text-danger"></span>
        </div>
        <button id="registerSubmit" type="submit" class="btn btn-primary w-100 mb-3">Зареєструвати</button>
        <div class="text-center">
            <p>Реєструвалися раніше? <a asp-page="./Login" asp-route-returnUrl="@Model.ReturnUrl">Увійти</a></p>
        </div>
    </form>
    @if ((Model.ExternalLogins?.Count ?? 0) > 0)
    {
        <hr class="text-white" />
        <div class="text-center">
            <form id="external-account" asp-page="./ExternalLogin" asp-route-returnUrl="@Model.ReturnUrl"
method="post">
                @foreach (var provider in Model.ExternalLogins!)
                {
                    <button type="submit" class="btn btn-outline-light w-100 my-1" name="provider" value="@provider.Name">
                        Зареєструватися через @provider.DisplayName
                    </button>
                }
            </form>
        </div>
    }
</div>
@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
</body>
</html>
```

RegisterConfirmation.cshtml

```
@page
@model RegisterConfirmationModel
@{
    Layout = "_AuthLayout";
    ViewData["Title"] = "Підтвердження реєстрації";
}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>@ViewData["Title"]</title>
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;600&display=swap" rel="stylesheet">
    <link rel="stylesheet" href="~/css/site-custom.css" />
</head>
```

```
<body>
    <div class="login-card">
        <h1>@ViewData["Title"]</h1>
        <p>Будь ласка, перевірте свою електронну пошту, щоб підтвердити свій аккаунт.</p>
    </div>
</body>
</html>
```

Data

ApplicationDbContext.cs

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

namespace WebApplication1.Data
{
    public class ApplicationDbContext : IdentityDbContext
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }
        public DbSet<Product> Products { get; set; }
        public DbSet<ProductImage> ProductImage { get; set; }
        public DbSet<Category> Categories { get; set; }
        public DbSet<Order> Orders { get; set; }
        public DbSet<OrderItem> OrderItems { get; set; }
        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);
        }
    }
}
```

DTOs

CardItemDto.cs

```
namespace SimpleShopApp.DTOs
{
    public class CartItemDto
    {
        public int ProductId { get; set; }
        public string ProductName { get; set; } = null!;
        public decimal Price { get; set; }
        public int Discount { get; set; }
        public int Quantity { get; set; }
        public string? ImageUrl { get; set; }
    }
}
```

LiqPayCallbackDto.cs

```
namespace SimpleShopApp.DTOs
{
    public class LiqPayCallbackDto
    {
        public string Status { get; set; } = "";
        public string OrderId { get; set; } = "";
```

```
        public decimal Amount { get; set; }
        public string Email { get; set; } = "";
        public string Description { get; set; } = "";
        public string PublicKey { get; set; } = "";
    }
}
```

Extensions

SessionExtensions.cs

```
using Newtonsoft.Json;
namespace SimpleShopApp.Extensions
{
    public static class SessionExtensions
    {
        public static void SetObjectAsJson(this ISession session, string key, object value)
        {
            session.SetString(key, JsonConvert.SerializeObject(value));
        }

        public static T? GetObjectFromJson<T>(this ISession session, string key)
        {
            var value = session.GetString(key);
            return value == null ? default : JsonConvert.DeserializeObject<T>(value);
        }
    }
}
```

Models

Category.cs

```
using System.ComponentModel.DataAnnotations;
namespace SimpleShopApp.Models
{
    public class Category
    {
        public int Id { get; set; }
        [Required(ErrorMessage = "Вкажіть назву категорії")]
        public string Name { get; set; } = null!;
        public ICollection<Product>? Products { get; set; }

    }
}
```

ErrorViewModel.cs

```
namespace WebApplication1.Models
{
    public class ErrorViewModel
    {
        public string? RequestId { get; set; }

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

Order.cs

```
using Microsoft.AspNetCore.Identity;
namespace SimpleShopApp.Models
```

```
{
    public class Order
    {
        public int Id { get; set; }
        public string UserId { get; set; } = null!;
        public IdentityUser? User { get; set; }
        public DateTime OrderDate { get; set; } = DateTime.Now;
        public ICollection<OrderItem> Items { get; set; } = new List<OrderItem>();
    }
}
```

OrderItem.cs

```
namespace SimpleShopApp.Models
{
    public class OrderItem
    {
        public int Id { get; set; }
        public int OrderId { get; set; }
        public Order? Order { get; set; }
        public int ProductId { get; set; }
        public Product? Product { get; set; }
        public int Quantity { get; set; }
        public decimal PriceAtPurchase { get; set; }
    }
}
```

Product.cs

```
using System.ComponentModel.DataAnnotations;
namespace SimpleShopApp.Models
{
    public class Product
    {
        public int Id { get; set; }
        [Required(ErrorMessage = "Назва обов'язкова")]
        public string Name { get; set; } = null!;
        [Range(1, double.MaxValue, ErrorMessage = "Ціна не може бути нижче 0")]
        public decimal Price { get; set; }
        [Range(0, int.MaxValue, ErrorMessage = "Кількість не може бути від’ємною")]
        public int Quantity { get; set; }
        [Range(0, 100, ErrorMessage = "Скидка повинна бути від 0 до 100")]
        public int Discount { get; set; }
        public string? Description { get; set; }
        public string? ImageUrl { get; set; }
        [Required(ErrorMessage = "Категорія обов'язкова")]
        public int CategoryId { get; set; }
        public Category? Category { get; set; }
        public bool IsTrending { get; set; }
        public ICollection<ProductImage> Images { get; set; } = new List<ProductImage>();
    }
}
```

ProductCatalogViewModel.cs

```
using Microsoft.AspNetCore.Mvc.Rendering;
namespace SimpleShopApp.Models
{
    public class ProductCatalogViewModel
    {
```

```
        public List<Product> Products { get; set; }
        public List<SelectListItem> Categories { get; set; }
        public int? SelectedCategoryId { get; set; }
        public string SearchQuery { get; set; }
        public string SortOrder { get; set; } // "name_asc", "name_desc", "price_asc", "price_desc"
        public List<Product> TrendingProducts { get; set; }
        public List<Product> OnSaleProducts { get; set; }
        public int? CurrentPage { get; set; }
        public int? TotalPages { get; set; }
    }
}
```

ProductImage

```
namespace SimpleShopApp.Models
{
    public class ProductImage
    {
        public int Id { get; set; }
        public string ImageUrl { get; set; } = null!;
        public int ProductId { get; set; }
        public Product Product { get; set; } = null!;
    }
}
```

Views/Cart

Index.cshtml

```
@using System.Globalization
@model List<CartItemDto>

<h2>Ваш кошик</h2>

@if (!Model.Any())
{
    <p>Кошик порожній.</p>
}
else
{
    <table class="table">
        <thead>
            <tr>
                <th>Фото</th>
                <th>Назва</th>
                <th>Кількість</th>
                <th>Ціна (зі знижкою)</th>
                <th>Сума</th>
                <th></th>
            </tr>
        </thead>
        <tbody>
            @{
                decimal total = 0;
            }
            @foreach (var item in Model)
            {
                var discountedPrice = Math.Round(item.Price * (100 - item.Discount) / 100);
                var itemTotal = discountedPrice * item.Quantity;
                total += itemTotal;
```

```
            <tr>
                <td><img src="@item.ImageUrl" class="img-fluid" width="100"/></td>
                <td>@item.ProductName</td>
                <td>@item.Quantity</td>
                <td>
                    @if (item.Discount > 0)
                    {
                        <del>@(item.Price)₴</del> <br />
                        <span class="text-danger fw-bold">@discountedPrice₴</span>
                    }
                    else
                    {
                        @($"{item.Price}₴")
                    }
                </td>
                <td>@itemTotal₴</td>
                <td>
                    <a asp-action="Remove" asp-route-id="@item.ProductId" class="btn btn-danger">Видалити</a>
                </td>
            </tr>
        }
    </tbody>
</table>
<h4>Загальна сума: @Math.Round(total)₴</h4>
<form asp-action="StartPayment" method="post">
    <input type="hidden" name="totalAmount" value="@total.ToString("F2", CultureInfo.InvariantCulture)" />
    <button type="submit" class="btn btn-success">Оплатити @total₴</button>
</form>
<a asp-action="Clear" class="btn btn-warning">Очистити кошик</a>
}
```

Pay.cshtml

```
@using SimpleShopApp.Services.LiqPay
@model List<CartItemDto>
@{
    var data = ViewData["Data"];
    var signature = ViewData["Signature"];
}
<form method="POST" accept-charset="utf-8" action="https://www.liqpay.ua/api/3/checkout">
    <input type="hidden" name="data" value="@data" />
    <input type="hidden" name="signature" value="@signature" />
    <button style="border: none; background: #77CC5D; color: white; font-weight: bold; padding: 10px 20px; border-radius:
6px;">
        <img src="https://static.liqpay.ua/buttons/logo-white.svg" style="vertical-align: middle;" />
        <span style="margin-left: 10px;">Оплатити</span>
    </button>
</form>
```

Views/Category

Create.cshtml

```
@model Category
@{
    ViewData["Title"] = "Створити категорію";
}
<h2>@ViewData["Title"]</h2>
<form asp-action="Create" method="post">
    <div class="form-group">
```

```
        <label asp-for="Name" class="control-label">Назва категорії</label>
        <input asp-for="Name" class="form-control" />
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <button type="submit" class="btn btn-primary mt-3">Зберегти</button>
    <a asp-action="Manage" asp-controller="Home" class="btn btn-secondary mt-3">Назад</a>
</form>
@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

Views/Home

Index.cshtml

```
@model ProductCatalogViewModel
@inject SignInManager<IdentityUser> SignInManager
@inject UserManager<IdentityUser> UserManager
@if (User.IsInRole("Admin"))
{
    <div class="alert alert-info text-center mt-4">
        <h5>Привіт Адміністратор!</h5>
    </div>
}
<div class="container py-4">
    <div class="text-center mb-4">
        <h1 class="display-5">Каталог продуктів</h1>
        <p class="lead">Знайди найкращий товар для себе!</p>
    </div>
    <form method="get" asp-controller="Home" asp-action="Index" class="row g-3 mb-5 justify-content-center">
        <div class="col-md-3">
            <select name="categoryId" class="form-select" asp-items="Model.Categories">
                <option value="">Всі категорії</option>
            </select>
        </div>
        <div class="col-md-3">
            <select name="sortOrder" class="form-select">
                <option value="">Сортувати за</option>
                <option value="name_asc" selected="@(Model.SortOrder == "name_asc")">Назва ↑</option>
                <option value="name_desc" selected="@(Model.SortOrder == "name_desc")">Назва ↓</option>
                <option value="price_asc" selected="@(Model.SortOrder == "price_asc")">Ціна ↑</option>
                <option value="price_desc" selected="@(Model.SortOrder == "price_desc")">Ціна ↓</option>
            </select>
        </div>
        <div class="col-md-3">
            <input type="text" name="searchQuery" class="form-control" placeholder="Знайти продукт..."
value="@Model.SearchQuery" />
        </div>
        <div class="col-md-2 d-grid">
            <button type="submit" class="btn btn-primary">Застосувати</button>
        </div>
    </form>
    @if (Model.OnSaleProducts.Any())
    {
        <h2 class="mt-5">🔥 У тренді</h2>
        <div class="scrolling-wrapper d-flex overflow-auto pb-3 bg-light p-4 rounded shadow-sm mb-5">
            @foreach (var product in Model.OnSaleProducts)
            {
                var discountedPrice = Math.Round(product.Price * (100 - product.Discount) / 100);
```

```
            var imageUrl = product.Images.FirstOrDefault()?.ImageUrl ?? "/images/no-image.png";
            <div class="card me-3 border-danger" style="min-width: 250px;">
                <img src="@imageUrl" class="card-img-top" style="height: 200px; object-fit: contain;" alt="@product.Name">
                <div class="card-body">
                    <h5 class="card-title">@product.Name</h5>
                    <p class="card-text">
                        <del>@product.Price.ToString("F0")₴</del>
                        <span style="color: red; font-weight: bold;">@discountedPrice₴</span>
                    </p>
                    <a asp-controller="Product" asp-action="Details" asp-route-id="@product.Id" class="btn btn-danger w-
100">Переглянути</a>
                </div>
            </div>
        }
    </div>
}
@if (!Model.Products.Any())
{
    <div class="alert alert-warning text-center">Товарів не знайдено.</div>
}
else
{
    <div class="row row-cols-1 row-cols-md-3 g-4 ">
        @foreach (var item in Model.Products)
        {
            var imageUrl = item.Images.FirstOrDefault()?.ImageUrl ?? "/images/no-image.png";
            <div class="col">
                <div class="card h-100 @(item.Discount > 0 ? "border-danger" : "")">
                    <div class="position-relative">
                        <div class="d-flex justify-content-center align-items-center" style="height: 250px;">
                            <img src="@imageUrl" class="@(item.Quantity == 0 ? "grayscale" : "")" alt="@item.Name"
style="object-fit: contain; max-height: 100%; max-width: 100%;" />
                        </div>
                        @if (item.Discount > 0)
                        {
                            <span class="badge bg-danger position-absolute top-0 end-0 m-2">
                                -@item.Discount%
                            </span>
                        }
                    </div>
                    <div class="card-body @(item.Quantity == 0 ? "text-muted" : "")">
                        <h5 class="card-title">@item.Name</h5>
                        <p class="card-text">
                            @if (item.Discount > 0)
                            {
                                var discountPrice = item.Price * (100 - item.Discount) / 100;
                                <span><del>@item.Price.ToString("F0")₴</del> <span style="color: red; font-weight:
bold;">@discountPrice.ToString("F0")₴</span></span>
                            }
                            else
                            {
                                <span>@item.Price.ToString("F0")₴</span>
                            }
                            <br />
                            Кількість: @item.Quantity <br />
                            @item.Category?.Name
                        </p>
```

```
                    @if (item.Quantity == 0)
                    {
                        <a asp-controller="Product" asp-action="Details" asp-route-id="@item.Id" class="btn btn-primary w-100">Немає в наявності</a>
                    }
                    else
                    {
                        <a asp-controller="Product" asp-action="Details" asp-route-id="@item.Id" class="btn btn-primary w-100">Переглянути</a>
                    }
                </div>
            </div>
        </div>
    }
</div>
<nav aria-label="Пагінація товарів">
    <ul class="pagination justify-content-center mt-4">
        @for (int i = 1; i <= Model.TotalPages; i++)
        {
            <li class="page-item @(i == Model.CurrentPage ? "active" : "")">
                <a class="page-link"
                   href="@Url.Action("Index", new { page = i, categoryId = Model.SelectedCategoryId, searchQuery = Model.SearchQuery, sortOrder = Model.SortOrder })">
                    @i
                </a>
            </li>
        }
    </ul>
</nav>
}
</div>
```

Manage.cshtml

```
@model List<Product>
<h1>Керування продуктами</h1>
<a asp-controller="Product" asp-action="Create" class="btn btn-primary">Створити новий продукт</a>
<a asp-controller="Category" asp-action="Create" class="btn btn-outline-secondary ms-2"> Створити нову категорію</a>
<button class="btn btn-danger ms-2" type="button" data-bs-toggle="collapse" data-bs-target="#categoryList">
    Всі категорії
</button>
@{
    var categories = ViewBag.Categories as List<SimpleShopApp.Models.Category>;
}
@if (categories != null && categories.Any())
{
    <div class="collapse mt-3" id="categoryList">
        <div class="card card-body">
            <h5>Категорії</h5>
            <ul class="list-group">
                @foreach (var cat in categories)
                {
                    <li class="list-group-item d-flex justify-content-between align-items-center">
                        @cat.Name
                        <form asp-controller="Category" asp-action="Delete" asp-route-id="@cat.Id" method="post"
                              onsubmit="return confirm('Ви впевнені, що хочете видалити цю категорію?')">
                            <button type="submit" class="btn btn-sm btn-danger">Видалити</button>
                        </form>
```

```
                </li>
            }
        </ul>
    </div>
</div>
}
else
{
    <p class="text-muted">Категорії не знайдено.</p>
}
<table class="table">
    <thead>
        <tr>
            <th scope="col">#</th>
            <th scope="col">Фото</th>
            <th scope="col">Назва</th>
            <th scope="col">Категорія</th>
            <th scope="col">Кількість</th>
            <th scope="col">Ціна</th>
            <th scope="col">Скидка</th>
            <th scope="col">Ціна зі скидкою</th>
            <th scope="col">Опис</th>
            <th scope="col">Дії</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
        {
            var discountedPrice = Math.Round(item.Price * (100 - item.Discount) / 100);
            var imageUrl = item.Images.FirstOrDefault()?.ImageUrl ?? "/images/no-image.png";
            <tr class="align-middle">
                <th scope="row">@item.Id</th>
                <td style="width: 100px; height: 100px;">
                    <div class="d-flex justify-content-center align-items-center" style="width: 100px; height: 100px;">
                        <img src="@imageUrl"
                            alt="@item.Name"
                            class="img-thumbnail @(item.Quantity == 0 ? "grayscale" : "")"
                            style="object-fit: contain; max-width: 100%; max-height: 100%;" />
                    </div>
                </td>
                <td>@item.Name</td>
                <td>@item.Category.Name</td>
                <td>@item.Quantity</td>
                <td>@item.Price.ToString("F0")₴</td>
                <td>@item.Discount%</td>
                <td>@discountedPrice₴</td>
                <td>@item.Description</td>
                <td>
                    <a asp-controller="Product" asp-action="Edit" asp-route-id="@item.Id" class="btn btn-warning me-
5">Редагувати</a>
                    <a asp-action="Delete" asp-route-id="@item.Id" class="btn btn-danger">Видалити</a>
                </td>
            </tr>
        }
    </tbody>
</table>
```

Privacy.cshtml

```
@{
    ViewData["Title"] = "Політика конфіденційності";
}
<h1>@ViewData["Title"]</h1>
<p>Використайте цю сторінку, щоб детально описати політику конфіденційності вашого сайту.</p>
```

Views/Order

History.cshtml

```
@model List<SimpleShopApp.Models.Order>
<h2>Історія покупок</h2>
<div style="max-height: 500px; overflow-y: auto; padding-right: 10px;">
    @foreach (var order in Model)
    {
        <div class="card my-3">
            <div class="card-header">
                Замовлення #@order.Id — @order.OrderDate.ToString("g")
            </div>
            <div class="card-body">
                @if (order.Items != null)
                {
                    @foreach (var item in order.Items)
                    {
                        <p>
                            <strong>@(item.Product?.Name ?? "—")</strong> — @item.Quantity шт —
@item.PriceAtPurchase.ToString("F0")₴
                        </p>
                        <p>
                            Загальна ціна: @((item.PriceAtPurchase * item.Quantity).ToString("F0"))₴
                        </p>
                    }
                }
                else
                {
                    <p>Це замовлення не містить товарів.</p>
                }
            </div>
        </div>
    }
</div>
```

Views/Payment

Success.cshtml

```
@{
    ViewData["Title"] = "Оплата успішна";
}
<h2>Дякуємо за покупку!</h2>
<p>Ваше замовлення успішно оформлено.</p>
<a href="/Order/History" class="btn btn-primary">Перейти до історії замовлень</a>
```

Views/Product

Create.cshtml

```
@model Product
@{
    ViewData["Title"] = "Додати продукт";
    var categories = ViewBag.Categories as SelectList;
```

```
}
<partial name="_ValidationScriptsPartial" />
 <h1>Додайте новий продукт</h1>
 <form class="row g-3" asp-action="Create" method="post">
   <div class="col-md-12">
      <label asp-for=Name class="form-label">Назва</label>
      <input asp-for=Name type="text" class="form-control" placeholder="Введіть назву продукта">
      <span asp-validation-for="Name" class="text-danger"></span>
   </div>
   <div class="col-md-6">
      <label asp-for=Price class="form-label">Ціна</label>
      <input asp-for=Price type="number" class="form-control" placeholder="Введіть ціну продукта (₴)">
      <span asp-validation-for="Price" class="text-danger"></span>
   </div>
   <div class="col-md-3">
      <label asp-for=Discount class="form-label">Скидка</label>
      <input asp-for=Discount type="number" class="form-control" placeholder="Введіть скидку продукта (%)">
      <span asp-validation-for="Discount" class="text-danger"></span>
   </div>
   <div class="col-md-3">
      <label asp-for=Quantity class="form-label">Кількість</label>
      <input asp-for=Quantity type="number" class="form-control" placeholder="Введіть кількість продукту">
      <span asp-validation-for="Quantity" class="text-danger"></span>
   </div>
   <div class="col-md-6">
      <label asp-for=ImageUrl class="form-label">Посилання на фото (декілька через кому)</label>
      <input asp-for=ImageUrl type="url" class="form-control" placeholder="Введіть посилання на фотографію
продукту">
      <span asp-validation-for="ImageUrl" class="text-danger"></span>
   </div>
   <div class="col-md-6">
      <label asp-for="CategoryId" class="form-label">Категорії</label>
      <select asp-for="CategoryId" class="form-select" asp-items="categories">
         <option disabled>Виберіть категорію...</option>
      </select>
      <span asp-validation-for="CategoryId" class="text-danger"></span>
   </div>
   <div class="col-md-12">
      <label asp-for=Description class="form-label">Опис</label>
      <textarea asp-for=Description class="form-control" rows="3" placeholder="Введіть опис продукту
(Необов'язково)"></textarea>
      <span asp-validation-for="Description" class="text-danger"></span>
   </div>
   <div class="col-6">
      <a asp-controller="Home" asp-action="Manage" class="btn btn-dark w-100">Назад</a>
   </div>
   <div class="col-6">
      <button type="submit" class="btn btn-success w-100">Додати</button>
   </div>
</form>
```

Details.cshtml

```
@model Product
@{
   ViewData["Title"] = Model.Name;
}
<div class="container mt-4">
```

```
    <div class="row">
        <div class="col-md-5 text-center">
            <img id="mainImage" src="@(Model.Images.FirstOrDefault()?.ImageUrl ?? " /images/no-image.png")"
class="img-fluid mb-3" alt="@Model.Name" style="max-height: 400px;" />
            <div class="d-flex justify-content-center gap-2">
                @foreach (var img in Model.Images)
                {
                    <img src="@img.ImageUrl" onclick="document.getElementById('mainImage').src = this.src"
                        style="height: 80px; width: 80px; object-fit: cover; cursor: pointer;" class="border rounded" />
                }
            </div>
        </div>
        <div class="col-md-7">
            <h2>@Model.Name</h2>
            <p><strong>Категорія:</strong> @Model.Category?.Name</p>
            <p><strong>Кількість:</strong> @Model.Quantity</p>
            <p><strong>Опис:</strong> @Model.Description</p>
            @if (Model.Discount > 0)
            {
                var discountPrice = Model.Price * (100 - Model.Discount) / 100;
                <p><del>@Model.Price.ToString("F0")₴</del> <span class="text-danger fw-
bold">@discountPrice.ToString("F0")₴</span></p>
            }
            else
            {
                <p>@Model.Price.ToString("F0")₴</p>
            }
            @if (Model.Quantity > 0)
            {
                <a asp-controller="Cart" asp-action="Add" asp-route-id="@Model.Id" class="btn btn-success">Додати до
кошика</a>
            }
            else
            {
                <button class="btn btn-secondary" disabled>Немає в наявності</button>
            }
        </div>
    </div>
</div>
```

Edit.cshtml

```
@model Product
@{
    ViewData["Title"] = "Редагувати продукт";
    var categories = ViewBag.Categories as SelectList;
}
<h1>Редагувати продукт</h1>
<form asp-action="Edit" method="post">
    <input type="hidden" asp-for="Id" />
    <div class="mb-3">
        <label asp-for="Name" class="form-label">Назва</label>
        <input asp-for="Name" class="form-control" required />
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <div class="mb-3">
        <label asp-for="Price" class="form-label">Ціна</label>
        <input asp-for="Price" class="form-control" required />
```

```
    <span asp-validation-for="Price" class="text-danger"></span>
  </div>
  <div class="mb-3">
    <label asp-for="Discount" class="form-label">Скидка</label>
    <input asp-for="Discount" class="form-control" required />
    <span asp-validation-for="Discount" class="text-danger"></span>
  </div>
  <div class="mb-3">
    <label asp-for="Quantity" class="form-label">Кількість</label>
    <input asp-for="Quantity" class="form-control" required />
    <span asp-validation-for="Quantity" class="text-danger"></span>
  </div>
  <div class="mb-3">
    <label asp-for="CategoryId" class="form-label">Категорії</label>
    <select asp-for="CategoryId" class="form-select" asp-items="categories" required>
      <option disabled>Виберіть категорію...</option>
    </select>
    <span asp-validation-for="CategoryId" class="text-danger"></span>
  </div>
  <div class="mb-3">
    <label asp-for="Description" class="form-label">Опис</label>
    <textarea asp-for="Description" class="form-control"></textarea>
    <span asp-validation-for="Description" class="text-danger"></span>
  </div>
  <div class="mb-3">
    <label asp-for="ImageUrl" class="form-label">Посилання на фото</label>
    <input asp-for="ImageUrl" class="form-control" placeholder="https://img1.jpg, https://img2.jpg"/>
    <span asp-validation-for="ImageUrl" class="text-danger"></span>
  </div>
  <form asp-action="AddImage" asp-controller="Product" method="post" class="mb-3">
    @Html.AntiForgeryToken()
    <input type="hidden" name="productId" value="@Model.Id" />
    <button type="submit" class="btn btn-primary">Зберегти зміни</button>
    <a asp-controller="Home" asp-action="Manage" class="btn btn-secondary">Скасувати</a>
  </form>
  @if (Model.Images != null && Model.Images.Any())
  {
    <div class="mb-3">
      <label>Поточні зображення:</label>
      <div class="d-flex flex-wrap gap-2">
        @foreach (var img in Model.Images)
        {
          <div class="border p-2 text-center position-relative">
            <img src="@img.ImageUrl" style="width: 120px; height: 120px; object-fit: cover;" class="mb-1" />
            <p class="small text-break">@img.ImageUrl</p>
            <form asp-action="DeleteImage" method="post" class="position-absolute top-0 end-0 m-1">
              <input type="hidden" name="imageId" value="@img.Id" />
              <button type="submit" class="btn btn-sm btn-danger" title="Видалити зображення">×</button>
            </form>
          </div>
        }
      </div>
    </div>
  }
</form>
```

Views/Shared

_Layout.cshtml

```
@inject SignInManager<IdentityUser> SignInManager
@inject UserManager<IdentityUser> UserManager
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" />
<link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;600&display=swap" rel="stylesheet" />
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>@ViewData["Title"] - MobiTrend</title>
   <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
   <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
   <link rel="stylesheet" href="~/WebApplication1.styles.css" asp-append-version="true" />
</head>
<body>
   <header>
      <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
         <div class="container-fluid">
            <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">МобіТренд</a>
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-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="Home" asp-action="Index">Головна</a>
                  </li>
                  @if (User.IsInRole("Admin") || User.IsInRole("Manager"))
                  {
                     <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-
action="Manage">Керування</a>
                     </li>
                  }
                  <li class="nav-item">
                     <a class="nav-link text-dark" asp-area="" asp-controller="Cart" asp-action="Index">Кошик</a>
                  </li>
                              <li class="nav-item">
                     <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-
action="Privacy">Конфіденційність</a>
                  </li>
                  @if (User.Identity.IsAuthenticated)
                  {
                     <li class="nav-item">
                        <a class="nav-link" asp-controller="Order" asp-action="History">Історія замовлень</a>
                     </li>
                  }
               </ul>
               <partial name="_LoginPartial" />
            </div>
         </div>
      </nav>
   </header>
```

```
    <div class="container">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

Controllers

CartController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using SimpleShopApp.DTOs;
using SimpleShopApp.Extensions;
using SimpleShopApp.Models;
using SimpleShopApp.Services.CartService;
using SimpleShopApp.Services.LiqPay;
using System.Globalization;
using WebApplication1.Data;
namespace SimpleShopApp.Controllers
{
    public class CartController : Controller
    {
        private readonly ApplicationDbContext _context;
        private readonly ICartService _cartService;
        public CartController(ApplicationDbContext context, ICartService cartService)
        {
            _context = context;
            _cartService = cartService;
        }
        [HttpPost]
        public IActionResult StartPayment(decimal totalAmount)
        {
            var orderId = Guid.NewGuid().ToString();
            var cart = HttpContext.Session.GetObjectFromJson<List<CartItemDto>>("Cart");
            if (cart == null || !cart.Any())
                return RedirectToAction("Index", "Cart");
            var description = string.Join(", ", cart.Select(c => $"{c.ProductName} x{c.Quantity}"));
            var amount = Math.Round(totalAmount).ToString("F0");
            var (data, signature) = LiqPayHelper.GeneratePaymentData(
                publicKey: "sandbox_i20147184055",
                privateKey: "sandbox_93Fkg63oYxuKokh48w1b96PnfEOjRvysVxNKlviq",
                amount: amount,
                description: description,
                orderId: orderId,
                baseUrl: $"{Request.Scheme}://{Request.Host}"
            );
            ViewData["Data"] = data;
            ViewData["Signature"] = signature;
            return View("Pay");
        }
        public IActionResult Index()
```

```
        {
            var cartItems = _cartService.GetCartItems();
            return View(cartItems);
        }
        public async Task<IActionResult> Add(int id)
        {
            var product = await _context.Products
                .Include(p => p.Images)
                .FirstOrDefaultAsync(p => p.Id == id);
            if (product != null)
            {
                _cartService.AddToCart(product);
            }
            return RedirectToAction("Index");
        }
        public IActionResult Remove(int id)
        {
            _cartService.RemoveFromCart(id);
            return RedirectToAction("Index");
        }
        public IActionResult Clear()
        {
            _cartService.ClearCart();
            return RedirectToAction("Index");
        }
    }
}
```

CategoryController.cs

```
using Microsoft.AspNetCore.Mvc;
using SimpleShopApp.Models;
using WebApplication1.Data;
namespace SimpleShopApp.Controllers
{
    public class CategoryController : Controller
    {
        private readonly ApplicationDbContext _context;
        public CategoryController(ApplicationDbContext context)
        {
            _context = context;
        }
        [HttpGet]
        public IActionResult Create()
        {
            return View();
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public IActionResult Create(Category category)
        {
            if (!ModelState.IsValid)
                return View(category);
            _context.Categories.Add(category);
            _context.SaveChanges();
            return RedirectToAction("Manage", "Home");
        }
        [HttpPost]
```

```
    [ValidateAntiForgeryToken]
    public IActionResult Delete(int id)
    {
        var category = _context.Categories.Find(id);
        if (category == null)
            return NotFound();
        var hasProducts = _context.Products.Any(p => p.CategoryId == id);
        if (hasProducts)
        {
            TempData["Error"] = "Неможливо видалити категорію, до якої прив’язані продукти.";
            return RedirectToAction("Index");
        }
        _context.Categories.Remove(category);
        _context.SaveChanges();
        return RedirectToAction("Manage", "Home");
    }
  }
}
```

HomeController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.Data.SqlClient;
using Microsoft.EntityFrameworkCore;
using SendGrid.Helpers.Mail;
using SimpleShopApp.Models;
using System.Diagnostics;
using WebApplication1.Data;
using WebApplication1.Models;
namespace WebApplication1.Controllers
{
  public class HomeController : Controller
  {
    private readonly ILogger<HomeController> _logger;
    private readonly ApplicationDbContext _context;
    public HomeController(ILogger<HomeController> logger, ApplicationDbContext context)
    {
        _logger = logger;
        _context = context;
    }
    public async Task<IActionResult> IndexAsync(int? categoryId, string searchQuery, string sortOrder)
    {
        int pageSize = 9;
        int page = int.TryParse(Request.Query["page"], out int p) ? p : 1;
        var productsQuery = _context.Products
            .Include(p => p.Category)
            .Include(p => p.Images)
            .AsQueryable();
        if (categoryId.HasValue)
            productsQuery = productsQuery.Where(p => p.CategoryId == categoryId.Value);
        if (!string.IsNullOrEmpty(searchQuery))
            productsQuery = productsQuery.Where(p => p.Name.Contains(searchQuery));
        productsQuery = sortOrder switch
        {
            "name_asc" => productsQuery.OrderBy(p => p.Name),
            "name_desc" => productsQuery.OrderByDescending(p => p.Name),
```

```
        "price_asc" => productsQuery.OrderBy(p => (double)p.Price),
        "price_desc" => productsQuery.OrderByDescending(p => (double)p.Price),
        _ => productsQuery
    };
    int totalItems = await productsQuery.CountAsync();
    int totalPages = (int)Math.Ceiling(totalItems / (double)pageSize);
    var pagedProducts = await productsQuery
        .Skip((page - 1) * pageSize)
        .Take(pageSize)
        .ToListAsync();
    var onSaleProducts = _context.Products
        .Where(p => p.Discount > 0)
        .Include(p => p.Images)
        .ToList();
    var viewModel = new ProductCatalogViewModel
    {
        Products = pagedProducts,
        Categories = _context.Categories.Select(c => new SelectListItem
        {
            Text = c.Name,
            Value = c.Id.ToString(),
            Selected = c.Id == categoryId
        }).ToList(),
        SelectedCategoryId = categoryId,
        SearchQuery = searchQuery,
        SortOrder = sortOrder,
        OnSaleProducts = onSaleProducts,
        CurrentPage = page,
        TotalPages = totalPages
    };
    return View(viewModel);
}
public IActionResult Manage()
{
        var products = _context.Products
        .Include(p => p.Images)
        .Include(p => p.Category)
        .ToList();
    ViewBag.Categories = _context.Categories.ToList();
    return View(products);
}
public IActionResult Create()
{
    return View();
}
[HttpPost]
public IActionResult Create(Product model)
{
    if (!ModelState.IsValid)
        return View(model);
    _context.Products.Add(model);
    _context.SaveChanges();
    return RedirectToAction("Manage");
}
[Authorize(Roles = "Admin")]
public IActionResult Delete(int id)
{
```

```
            var item = _context.Products.Find(id);
            if (item == null) return NotFound();
            _context.Products.Remove(item);
            _context.SaveChanges();
            return RedirectToAction("Manage");
        }
        public IActionResult Privacy()
        {
            return View();
        }
        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }
    }
}
```

OrderController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using SimpleShopApp.Services;
namespace SimpleShopApp.Controllers
{
    [Authorize]
    public class OrderController : Controller
    {
        private readonly IOrderService _orderService;
        private readonly UserManager<IdentityUser> _userManager;
        public OrderController(IOrderService orderService, UserManager<IdentityUser> userManager)
        {
            _orderService = orderService;
            _userManager = userManager;
        }
        public async Task<IActionResult> History()
        {
            var userId = _userManager.GetUserId(User);
            var orders = await _orderService.GetOrdersByUserAsync(userId);
            return View(orders);
        }
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

PaymentController.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using SimpleShopApp.DTOs;
using SimpleShopApp.Models;
using SimpleShopApp.Services;
using SimpleShopApp.Services.CartService;
using System.Text;
using System.Text.Json;
```

```
using WebApplication1.Data;
namespace SimpleShopApp.Controllers
{
    public class PaymentController : Controller
    {
        private readonly ApplicationDbContext _context;
        private readonly ICartService _cartService;
        private readonly IOrderService _orderService;
        private readonly UserManager<IdentityUser> _userManager;
        public PaymentController(ApplicationDbContext context, ICartService cartService, IOrderService orderService,
UserManager<IdentityUser> userManager)
        {
            _context = context;
            _orderService = orderService;
            _cartService = cartService;
            _userManager = userManager;
        }
        public IActionResult Index()
        {
            return View();
        }
        [HttpPost]
        [IgnoreAntiforgeryToken]
        public async Task<IActionResult> Callback()
        {
            var data = Request.Form["data"];
            var signature = Request.Form["signature"];
            var liqpayData = DecodeLiqPayData<LiqPayCallbackDto>(data!);
            if (liqpayData.Status == "success")
            {
                var userId = ExtractUserIdFromOrderId(liqpayData.OrderId);
                var cartItems = _cartService.GetCartItems();
                if (cartItems.Any())
                {
                    var order = new Order
                    {
                        UserId = userId,
                        OrderDate = DateTime.Now,
                        Items = cartItems.Select(ci => new OrderItem
                        {
                            ProductId = ci.ProductId,
                            Quantity = ci.Quantity,
                            PriceAtPurchase = ci.Price * (100 - ci.Discount) / 100
                        }).ToList()
                    };
                    _context.Orders.Add(order);
                    await _context.SaveChangesAsync();
                    _cartService.ClearCart();
                }
            }
            return Ok();
        }
        public async Task<IActionResult> Success()
        {
            var userId = _userManager.GetUserId(User);
            if (string.IsNullOrEmpty(userId))
            {
```

```
            return RedirectToAction("Index", "Home");
        }
        var cartItems = _cartService.GetCartItems();
        if (cartItems == null || !cartItems.Any())
        {
            return RedirectToAction("Index", "Home");
        }
        await _orderService.CreateOrderAsync(userId, cartItems);
        _cartService.ClearCart();
        return View();
    }
    public IActionResult Result()
    {
        return View();
    }
    private static T DecodeLiqPayData<T>(string base64)
    {
        var json = Encoding.UTF8.GetString(Convert.FromBase64String(base64));
        return JsonSerializer.Deserialize<T>(json)!;
    }
    private string ExtractUserIdFromOrderId(string orderId)
    {
        var parts = orderId.Split('_');
        return parts.Length >= 2 ? parts[1] : "";
    }
  }
}
```

ProductController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.CodeAnalysis.Elfie.Serialization;
using Microsoft.EntityFrameworkCore;
using SimpleShopApp.Models;
using System.Globalization;
using WebApplication1.Data;
using CsvHelper;
namespace SimpleShopApp.Controllers
{
  public class ProductController : Controller
  {
    private readonly ApplicationDbContext _context;
    public ProductController(ApplicationDbContext context)
    {
        _context = context;
    }
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> AddImage(int productId, string imageUrl)
    {
        if (string.IsNullOrWhiteSpace(imageUrl))
            return RedirectToAction("Edit", new { id = productId });
        var product = await _context.Products
            .Include(p => p.Images)
            .FirstOrDefaultAsync(p => p.Id == productId);
        if (product == null)
            return NotFound();
```

```
    var urls = imageUrl.Split(',', StringSplitOptions.RemoveEmptyEntries | StringSplitOptions.TrimEntries);
    foreach (var url in urls)
    {
        product.Images.Add(new ProductImage { ImageUrl = url });
    }
    await _context.SaveChangesAsync();
    return RedirectToAction("Edit", new { id = productId });
}
[HttpPost]
public async Task<IActionResult> DeleteImage(int imageId)
{
    var image = await _context.ProductImage.FindAsync(imageId);
    if (image == null)
        return NotFound();
    int productId = image.ProductId;
    _context.ProductImage.Remove(image);
    await _context.SaveChangesAsync();
    return RedirectToAction("Edit", new { id = productId });
}
public async Task<IActionResult> Edit(int id)
{
    var product = await _context.Products
    .Include(p => p.Images)
    .FirstOrDefaultAsync(p => p.Id == id);
    if (product == null) return NotFound();
    ViewBag.Categories = new SelectList(_context.Categories, "Id", "Name", product.CategoryId);
    return View(product);
}
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(Product product)
{
    if (!ModelState.IsValid)
    {
        ViewBag.Categories = new SelectList(_context.Categories, "Id", "Name", product.CategoryId);
        return View(product);
    }
    var existing = await _context.Products.FindAsync(product.Id);
    if (existing == null) return NotFound();
    existing.Name = product.Name;
    existing.Price = product.Price;
    existing.Discount = product.Discount;
    existing.Description = product.Description;
    existing.Quantity = product.Quantity;
    existing.CategoryId = product.CategoryId;
    if (!string.IsNullOrEmpty(product.ImageUrl))
    {
        var urls = product.ImageUrl
            .Split(',', StringSplitOptions.RemoveEmptyEntries | StringSplitOptions.TrimEntries);
        foreach (var url in urls)
        {
            existing.Images.Add(new ProductImage
            {
                ImageUrl = url,
                ProductId = existing.Id
            });
        }
```

```
    }
    await _context.SaveChangesAsync();
    return RedirectToAction("Manage", "Home");
}
public IActionResult Details(int id)
{
    var product = _context.Products
        .Include(p => p.Category)
        .Include(p => p.Images)
        .FirstOrDefault(p => p.Id == id);
    if (product == null)
    {
        return NotFound();
    }
    return View(product);
}
public IActionResult Create()
{
    ViewBag.Categories = new SelectList(_context.Categories, "Id", "Name");
    return View(new Product());
}
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Create(Product model)
{
    if (!ModelState.IsValid)
    {
        ViewBag.Categories = new SelectList(_context.Categories, "Id", "Name", model.CategoryId);
        return View(model);
    }
    var product = new Product
    {
        Name = model.Name,
        Price = model.Price,
        Quantity = model.Quantity,
        Discount = model.Discount,
        Description = model.Description,
        CategoryId = model.CategoryId,
    };
    if (!string.IsNullOrEmpty(model.ImageUrl))
    {
        var urls = model.ImageUrl.Split(',', StringSplitOptions.RemoveEmptyEntries | StringSplitOptions.TrimEntries);
        foreach (var url in urls)
        {
            product.Images.Add(new ProductImage { ImageUrl = url });
        }
    }
    _context.Products.Add(product);
    _context.SaveChanges();

    return RedirectToAction("Manage", "Home");
}
public IActionResult Manage()
{
    var products = _context.Products.Include(p => p.Category).ToList();
    return View(products);
}
```

```
    }
}
```

Services/CartService

```
using SimpleShopApp.DTOs;
using SimpleShopApp.Models;
namespace SimpleShopApp.Services.CartService
{
    public interface ICartService
    {
        List<CartItemDto> GetCartItems();
        void AddToCart(Product product);
        void RemoveFromCart(int productId);
        void ClearCart();
    }
}

using SimpleShopApp.DTOs;
using SimpleShopApp.Models;
using System.Text.Json;
namespace SimpleShopApp.Services.CartService
{
    public class CartService : ICartService
    {
        private readonly IHttpContextAccessor _httpContextAccessor;
        private const string SessionKey = "Cart";

        public CartService(IHttpContextAccessor httpContextAccessor)
        {
            _httpContextAccessor = httpContextAccessor;
        }
        public List<CartItemDto> GetCartItems()
        {
            var session = _httpContextAccessor.HttpContext?.Session;
            var data = session?.GetString(SessionKey);
            return data == null ? new List<CartItemDto>() : JsonSerializer.Deserialize<List<CartItemDto>>(data)!;
        }
        public void AddToCart(Product product)
        {
            var items = GetCartItems();
            var existingItem = items.FirstOrDefault(i => i.ProductId == product.Id);

            if (existingItem != null)
            {
                existingItem.Quantity++;
            }
            else
            {
                items.Add(new CartItemDto
                {
                    ProductId = product.Id,
                    ProductName = product.Name,
                    Price = product.Price,
                    Discount = product.Discount,
                    Quantity = 1,
                    ImageUrl = product.Images.FirstOrDefault()?.ImageUrl
                });
```

```
            }
            SaveCart(items);
        }
        public void RemoveFromCart(int productId)
        {
            var items = GetCartItems();
            var item = items.FirstOrDefault(i => i.ProductId == productId);
            if (item != null)
            {
                items.Remove(item);
                SaveCart(items);
            }
        }
        public void ClearCart()
        {
            SaveCart(new List<CartItemDto>());
        }

        private void SaveCart(List<CartItemDto> items)
        {
            var session = _httpContextAccessor.HttpContext?.Session;
            session?.SetString(SessionKey, JsonSerializer.Serialize(items));
        }
    }
}
```

Services/LiqPay

LiqPayHelper.cs

```
using System.Security.Cryptography;
using System.Text;
using System.Text.Json;
namespace SimpleShopApp.Services.LiqPay
{
    public class LiqPayHelper
    {
        public static string Base64Encode(string text)
        {
            return Convert.ToBase64String(Encoding.UTF8.GetBytes(text));
        }
        public static string GetSignature(string privateKey, string data)
        {
            var signatureString = privateKey + data + privateKey;
            using var sha1 = SHA1.Create();
            var hash = sha1.ComputeHash(Encoding.UTF8.GetBytes(signatureString));
            return Convert.ToBase64String(hash);
        }
        public static (string data, string signature) GeneratePaymentData(
            string publicKey,
            string privateKey,
            string amount,
            string description,
            string orderId,
            string baseUrl
        )
        {
            var payload = new Dictionary<string, object>
        {
            { "version", "3" },
```

```
            { "action", "pay" },
            { "amount", amount },
            { "currency", "UAH" },
            { "description", description },
            { "order_id", orderId },
            { "public_key", publicKey },
            { "language", "uk" },
            { "sandbox", 1 },
            { "server_url", $"{baseUrl}/Payment/Callback" },
            { "result_url", $"{baseUrl}/Payment/Success" }
        };
            var json = JsonSerializer.Serialize(payload);
            var data = Base64Encode(json);
            var signature = GetSignature(privateKey, data);
            return (data, signature);
        }
    }
}
```

Services/OrderService

```
using SimpleShopApp.DTOs;
using SimpleShopApp.Models;
namespace SimpleShopApp.Services
{
    public interface IOrderService
    {
        Task<List<Order>> GetOrdersByUserAsync(string userId);
        Task CreateOrderAsync(string userId, List<CartItemDto> cartItems);
    }
}
using Microsoft.EntityFrameworkCore;
using SimpleShopApp.DTOs;
using SimpleShopApp.Models;
using WebApplication1.Data;
namespace SimpleShopApp.Services
{
    public class OrderService : IOrderService
    {
        private readonly ApplicationDbContext _context;
        public OrderService(ApplicationDbContext context)
        {
            _context = context;
        }
        public async Task<List<Order>> GetOrdersByUserAsync(string userId)
        {
            return await _context.Orders
                .Where(o => o.UserId == userId)
                .Include(o => o.Items)
                    .ThenInclude(i => i.Product)
                .OrderByDescending(o => o.OrderDate)
                .ToListAsync();
        }
        public async Task CreateOrderAsync(string userId, List<CartItemDto> cartItems)
        {
            var order = new Order
            {
                UserId = userId,
                OrderDate = DateTime.Now,
```

```
            Items = cartItems.Select(i => new OrderItem
            {
                ProductId = i.ProductId,
                Quantity = i.Quantity,
                PriceAtPurchase = i.Price * (1 - i.Discount / 100m)
            }).ToList()
        };
        _context.Orders.Add(order);
        await _context.SaveChangesAsync();
    }
  }
}
```

AuthMessageSenderOptions

```
namespace SimpleShopApp.Services
{
  public class AuthMessageSenderOptions
  {
    public string? SendGridKey { get; set; }
  }
}
```

EmailSender.cs

```
using System.Net;
using System.Net.Mail;
using SendGrid.Helpers.Mail;
using SendGrid;
using Microsoft.AspNetCore.Identity.UI.Services;
namespace SimpleShopApp.Services
{
  public class EmailSender : IEmailSender
  {
    private readonly string _apiKey = "SG.jCkyk5ovTfGGsxcpNFGt-Q.YZW2KFjMHfmH1_3f9c5l-
buXv7wJTTGEAGmGT_XNYqY";
    public async Task SendEmailAsync(string subject, string toEmail, string username, string message)
    {
      var apiKey = "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";
      var client = new SendGridClient(apiKey);
      var from = new EmailAddress("email@gmail.com", "Mobitrend");
      var to = new EmailAddress(toEmail, username);
      var plainTextContent = message;
      var htmlContent = "";
      var msg = MailHelper.CreateSingleEmail(from, to, subject, plainTextContent, htmlContent);
      var response = await client.SendEmailAsync(msg);
    }
    public async Task SendEmailAsync(string email, string subject, string htmlMessage)
    {
      var client = new SendGridClient(_apiKey);
      var from = new EmailAddress("rostyk.gydzelyak@gmail.com", "Mobitrend");
      var to = new EmailAddress(email);
      var msg = MailHelper.CreateSingleEmail(from, to, subject, "", htmlMessage);
      await client.SendEmailAsync(msg);
    }
  }
}
```