Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп’ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота магістра

на тему: «Розробка крос-платформного додатка "Інтернет-магазин"»

Виконав: студент групи К23-2М

Спеціальність 122 Комп’ютерні науки

Коваленко М.Ю.
(прізвище та ініціали)

Керівник д.е.н., проф. Корнєєв М.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Університет митної справи та фінансів
(місце роботи)

Доцент кафедри кібербезпеки та інфомармаційних технологій
(посада)

к.т.н., доц. Флоров С.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Коваленко М.Ю. Розробка крос-платформного додатка «Інтернет-магазин».

Кваліфікаційна робота на здобуття освітнього ступеня магістр за спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2025.

Об'єктом дослідження є програмні засоби для створення та функціонування інтернет-магазинів.

Предметом дослідження є процес розробки крос-платформних додатків для електронної комерції.

Мета роботи – розробка та програмна реалізація кросплатформного додатка інтернет-магазину з використанням фреймворку Xamarin та архітектурного патерну MVVM.

Робота включає аналіз основних моделей електронної комерції (B2B, B2C, C2C, C2B), дослідження технологій для кросплатформної розробки, зокрема Xamarin, React Native та Flutter, а також порівняння їх переваг і недоліків. Обґрунтовано вибір фреймворку Xamarin для реалізації проекту через його інтеграцію з екосистемою .NET та зручність застосування патерну MVVM, що забезпечує розділення бізнес-логіки та інтерфейсу користувача.

Результатом дослідження є функціональний кросплатформний додаток для інтернет-магазину, який підтримує платформи Android, iOS та Windows. Додаток забезпечує зручний користувацький інтерфейс, ефективний каталог товарів, додавання товарів до кошика та здійснення оплат, відповідаючи сучасним вимогам до продуктивності, надійності та багатоплатформності.

Ключові слова: Xamarin, MVVM, .NET, крос-платформний додаток, електронна комерція, інтернет-магазин.

ABSTRACT

Kovalenko M.Yu. Development of a Cross-Platform "Online Store" Application.

Master's thesis for obtaining the Master’s degree in specialty 122 "Computer Science". – University of Customs and Finance, Dnipro, 2025.

The object of the research is software tools for creating and operating online stores.

The subject of the research is the process of developing cross-platform applications for e-commerce.

The purpose of the work is the development and software implementation of a cross-platform online store application using the Xamarin framework and the MVVM architectural pattern.

The work includes an analysis of the main e-commerce models (B2B, B2C, C2C, C2B), research on technologies for cross-platform development, including Xamarin, React Native, and Flutter, as well as a comparison of their advantages and disadvantages. The choice of the Xamarin framework for project implementation is justified due to its integration with the .NET ecosystem and the convenience of using the MVVM pattern, which ensures the separation of business logic and the user interface.

The result of the research is a functional cross-platform online store application supporting Android, iOS, and Windows platforms. The application provides a user-friendly interface, an efficient product catalog, the ability to add products to the cart, and payment processing, meeting modern requirements for performance, reliability, and cross-platform compatibility.

Keywords: Xamarin, MVVM, .NET, cross-platform application, e-commerce, online store.

ЗМІСТ

ВСТУП

Стрімкий розвиток цифрових технологій значно вплинув на методи ведення бізнесу та організації комерційної діяльності. Однією з найбільш динамічно розвиваючих галузей є електронна комерція (e-commerce), яка охоплює всі аспекти купівлі та продажу товарів або послуг через Інтернет. Сучасні інтернет-магазини пропонують користувачам зручність, широкий асортимент продукції та доступність у будь-який час. У таких умовах розробка ефективних програмних рішень для електронної комерції стає важливим завданням для IT-індустрії.

Інтернет-магазини забезпечують автоматизацію процесів замовлення, оплати та доставки товарів. Водночас, зростання конкуренції на ринку вимагає від таких платформ не лише функціональності, а й підтримки роботи на різних пристроях, що робить актуальною розробку крос-платформних рішень. Використання сучасних технологій, таких як Xamarin або Flutter, дозволяє створювати додатки, які працюють на декількох операційних системах, зберігаючи при цьому єдину кодову базу.

Актуальність теми обумовлена потребою у створенні крос-платформних інструментів, які можуть забезпечити зручність користування, високу продуктивність та адаптацію до вимог різних пристроїв. Розробка інтернет-магазину із застосуванням таких технологій сприяє розширенню аудиторії, підвищенню конкурентоспроможності бізнесу та ефективності його роботи.

Метою дослідження є розробка крос-платформного додатка інтернет-магазину.

Для досягнення мети поставлено такі завдання:

1. Провести аналіз сучасних рішень у галузі електронної комерції.
2. Дослідити методи реалізації кросплатформних додатків і обґрунтувати вибір оптимальної технології.
3. Розробити архітектуру додатка на основі принципів MVVM.

4. Реалізувати функціональність інтернет-магазину, включаючи каталог товарів, управління замовленнями та інтеграцію платіжних систем.

5. Провести тестування додатка для перевірки його продуктивності та надійності.

Об'єктом дослідження є програмні засоби для електронної комерції.

Предметом дослідження є процес розробки крос-платформних додатків.

Практичне значення отриманих результатів полягає у створенні сучасного інструменту для управління комерційною діяльністю, що дозволяє забезпечити зручність для користувачів та зменшити витрати на розробку і підтримку програмного забезпечення. Використання технології Xamarin забезпечує можливість швидкого масштабування та підтримки додатка для роботи на різних платформах.

Наукова новизна роботи полягає у використанні сучасних підходів до розробки інтернет-магазину, які включають застосування кросплатформних технологій для оптимізації процесу розробки та підтримки програмного продукту.

Робота складається зі вступу, трьох розділів, висновків, списку використаної літератури та додатків. Загальний обсяг роботи складає 60 сторінки, містить 33 рисунків.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Основні поняття електронної комерції

Електронна комерція, або «e-commerce», охоплює процес купівлі та продажу товарів або послуг через Інтернет, а також обмін грошовими коштами і даними для здійснення таких операцій. Хоча цей термін часто використовується для позначення продажу фізичних товарів онлайн, він також охоплює будь-які комерційні операції, що відбуваються у мережі. Електронна комерція включає транзакції товарів і послуг, а також всі аспекти онлайн-бізнесу. Щороку цей вид комерційної діяльності набирає популярності, збільшуючи обсяги продажів товарів і розширюючи спектр напрямків та галузей, які вона охоплює. Вона може включати такі процеси, як бронювання та виконання замовлень, фінансові операції через банківські сервіси або системи електронних грошей. Виділяють чотири основні моделі електронної комерції, які описують майже всі можливі транзакції між споживачами та компаніями.

Рисунок 1.1 –Моделі електронної комерції

Модель Business-to-consumer (B2C) передбачає продаж компанією товарів або послуг безпосередньо кінцевому споживачеві, наприклад, придбання взуття в онлайн-магазині. Business-to-business (B2B) полягає у продажу товарів або послуг однією компанією іншій, наприклад, надання програмного забезпечення як послуги для використання іншими підприємствами. Consumer-to-consumer (C2C) стосується продажу товарів або послуг одним споживачем іншому, наприклад, продаж вживаних меблів через платформу eBay. Consumer-to-business (C2B) передбачає, що споживачі пропонують свої товари чи послуги компаніям або організаціям, наприклад, коли блогер рекламує товари компанії своїй аудиторії або фотограф ліцензує свої знімки для комерційного використання [29].

Сьогодні моделі B2B та B2C охоплюють більшість ринків. Проте ефективна електронна комерція потребує наявності таких ключових компонентів:

- Інтернет-платформа (веб-сайт, обліковий запис, онлайн-магазин або цільова сторінка);
- Канали залучення трафіку (SEO, SMM, контекстна та таргетована реклама);
- Системи обробки замовлень і обслуговування клієнтів;
- CRM-системи, відділи продажів і служби підтримки.

Послуги в електронній комерції включають процеси закупівлі, постачання, доставки та повернення товарів. Цей вид комерції охоплює широкий спектр транзакцій: від угод між бізнесом і споживачем до обміну різноманітними товарами.

Основні формати продажів в електронній комерції:

- Роздрібна торгівля — прямий продаж товарів споживачам без посередників.
- Оптова торгівля — продаж великих партій товарів, зазвичай роздрібним продавцям, які надалі реалізують їх кінцевим споживачам.

- Дропшиппінг — продаж товарів, які виготовляються та доставляються замовнику стороннім постачальником.
- Краудфандинг — це процес збору коштів від споживачів до моменту запуску продукту, що дозволяє забезпечити необхідний стартовий капітал для виходу на ринок.
- Підписка передбачає регулярне автоматичне поновлення доступу до товару або послуги, доки клієнт не скасує підписку.
- Фізичні товари — це продукти, які потребують поповнення запасів та фізичної доставки при кожному замовленні.
- Цифрові продукти — це товари, які можна завантажити перед використанням, наприклад, моделі, онлайн-курси чи медіафайли.
- Послуги включають надання певних навичок або експертних знань за плату, де клієнт оплачує час або результати роботи постачальника послуг.

Для здійснення онлайн-транзакцій необхідно створювати спеціалізовані ресурси, які спрощують процес для всіх залучених сторін. Різні моделі електронної комерції підтримують кілька типів інтернет-магазинів.

Процес оформлення замовлення в інтернет-магазині починається з вибору відвідувачем товарів на сайті. Після цього формується замовлення. Якщо користувач уже робив покупки на цьому сайті, він може увійти в обліковий запис, вказавши логін, пароль та адресу доставки. Новим покупцям необхідно надати додаткову контактну інформацію. Далі обирається спосіб доставки, і розраховується кінцева вартість замовлення, включаючи вартість товару та доставку.

Наступним етапом є вибір способу оплати, який зазвичай передбачає перенаправлення на сайт платіжної системи для авторизації та здійснення оплати.

Переваги інтернет-магазинів:

- Доступність 24/7 – інтернет-магазини працюють цілодобово, без вихідних і свят.
- Відсутність потреби у фізичних запасах

можна використовувати модель дропшипінгу або принцип "точно вчасно" з мережею постачальників.

- Економія на приміщенні
- Низькі витрати на запуск
- Глобальне охоплення
- Автономність
- Хакерські атаки.
- Помилки у програмному забезпеченні.
- Легка втрата клієнтів.

Попри можливі труднощі, які можна подолати завдяки ретельному підходу до розробки та обслуговування сайту, інтернет-магазини продовжують завойовувати довіру споживачів і стають все більш популярними. Головна особливість інтернет-магазину полягає у взаємодії продавця та покупця в онлайн-форматі, що забезпечує зручність для обох сторін, але водночас вимагає створення довіри до платформи.

Інтернет-магазини можна класифікувати за декількома критеріями, кожен з яких відображає різні підходи до організації та функціонування електронної комерції. Основними критеріями класифікації є:

1. За моделлю ведення бізнесу:

- Онлайновий магазин — функціонує виключно в цифровому середовищі, без фізичної торгової точки.
- Поєднання офлайнового та онлайнового бізнесу — онлайн-магазин створений на базі вже існуючої фізичної торгової структури.
- Аутсорсинговий магазин — прийом, обробка та доставка замовлень передані стороннім компаніям, тоді як власник магазину виконує лише організаційні функції.
- Дропшипінг — модель, за якої магазин не зберігає товарних запасів, а лише реалізує продукцію постачальників, які безпосередньо доставляють товар кінцевим споживачам.

2. За обсягами продажів:

- Роздрібна торгівля (B2C) — реалізація товарів кінцевим споживачам у невеликих обсягах.
- Оптова торгівля (B2B) — продаж товарів у великих партіях, як правило, іншим підприємствам.

3. За видами продажів:

- B2B (Business-to-Business) — продаж товарів або послуг підприємствам та організаціям.
- B2C (Business-to-Consumer) — продаж товарів або послуг кінцевим споживачам.

4. За способами отримання доходу:

- Прямий продаж товарів і послуг — реалізація продукції безпосередньо від виробника або офіційного представника.
- Партнерські програми — отримання доходу від комісій за продажі сторонніх товарів через афілійовані посилання.
- Продаж інформаційних продуктів — реалізація цифрових товарів, таких як електронні книги, курси, мультимедійний контент, через платний доступ або підписку.

5. За відношенням до постачальників:

- Магазин із власним складом — має реальні запаси продукції та самостійно контролює зберігання та доставку товарів.
- Магазин, що працює за договорами з постачальниками — здійснює продаж без значних власних запасів товарів, використовуючи послуги сторонніх постачальників [4].

6. За способом подання товарів у каталозі:

- Інтернет-вітрина — інформаційний ресурс, що містить дані про товари, але не забезпечує можливість оформлення замовлення безпосередньо через сайт. Зазвичай використовується для демонстрації складних або дорогих товарів.

- Інтернет-магазин із каталогом — структурований веб-ресурс, де представлено каталог товарів з можливістю перегляду карток товарів, додавання їх до кошика, оформлення замовлення та оплати онлайн.
- Онлайн-аукціон — веб-платформа, де покупці можуть робити ставки на товари, а угода укладається після завершення аукціону.

Ці класифікації дають змогу систематизувати підходи до ведення електронної комерції, відображаючи різноманіття бізнес-моделей та стратегій, що використовуються у цифровому середовищі.

1.2 Аналіз методів реалізації кросплатформного застосунку

Мобільні пристрої відіграють ключову роль у сучасному житті, що робить розробку мобільних застосунків надзвичайно актуальною сферою програмування. Існує кілька підходів до створення мобільних застосунків, зокрема: нативні, кросплатформні, прогресивні веб-додатки (PWA) та гібридні рішення.

Нативні застосунки розробляються спеціально для певної операційної системи або пристрою, що дозволяє їм ефективно використовувати можливості платформи, на якій вони працюють. Найпоширенішими операційними системами для нативної розробки є Android та iOS. Для забезпечення стабільної та коректної роботи додатків їх розробляють мовами програмування, специфічними для кожної платформи: Java або Kotlin для Android і Swift для iOS [1].

Кросплатформні застосунки створюються з використанням проміжної мови програмування, яка не є рідною для жодної з платформ. Код компілюється у відповідний формат операційної системи за допомогою спеціальних фреймворків, таких як Flutter або React Native. Завдяки цьому значна частина програмного коду може бути повторно використана для різних платформ, включаючи iOS та Android.

Прогресивні веб-додатки (PWA) поєднують у собі функціональність нативних застосунків та класичних веб-сайтів. Вони розробляються за допомогою стандартних веб-технологій, таких як HTML5, CSS3 та JavaScript, та можуть бути доступними на різних пристроях. Важливою перевагою PWA є можливість їх індексації пошуковими системами та прямий доступ через браузер. Завантаження таких додатків з магазинів застосунків не передбачено: користувач отримує пропозицію встановити додаток безпосередньо після відвідування сайту через браузер [2].

Гібридні застосунки поєднують характеристики нативних і веб-додатків. Вони створюються із використанням веб-технологій (HTML5, CSS3, JavaScript), проте упаковані у нативну оболонку, що дозволяє їм працювати як звичайні додатки, завантажувані через магазини застосунків. Гібридні застосунки можуть використовувати можливості пристрою через вбудований браузер, однак ці функції залишаються непомітними для кінцевого користувача.

Нативний метод розробки мобільних застосунків передбачає створення додатків, які працюють лише на одній конкретній платформі, наприклад, Android або iOS. Вони розробляються за допомогою мов програмування, специфічних для операційної системи (Java або Kotlin для Android, Swift для iOS). Цей метод підходить для додатків із високими вимогами до продуктивності, складних програм з великою кількістю функцій, а також для застосунків, які потребують доступу до апаратних можливостей пристрою, таких як камера або GPS. Прикладами нативних додатків є системні програми, такі як «Калькулятор» та «Нотатки», а також мобільні ігри, зокрема популярна гра з доповненою реальністю *Pokemon Go*, яка спонукає користувачів досліджувати реальний світ у пошуках ігрових персонажів.

Кросплатформний метод розробки дозволяє створювати застосунки, що працюють на кількох платформах одночасно, використовуючи спільний код. Розробка здійснюється за допомогою фреймворків, таких як Flutter або React Native. Такий підхід підходить, коли необхідно створити додаток для iOS та

Android одночасно, проект не є надто складним, бюджет обмежений, а також потрібно швидко вийти на ринок [5]. Прикладами кросплатформних застосунків є:

- *Alibaba* — торговий застосунок, розроблений на Flutter;
- *Facebook* — спочатку був гібридним, проте перейшов на кросплатформний формат;
- *Slack* — корпоративний месенджер, який забезпечує майже нативний досвід користувача;
- *eBay Motors* — мобільний додаток для купівлі-продажу автомобілів, створений на Flutter через обмеження часу розробки [7].

Гібридний метод розробки поєднує елементи нативних і веб-технологій. Це веб-застосунки, створені за допомогою HTML, CSS і JavaScript, упаковані у нативну оболонку, яка дозволяє додатку працювати на мобільних пристроях. Такий підхід використовується для простих проєктів, орієнтованих на широкий спектр пристроїв, або для мінімально життєздатних продуктів (MVP), коли необхідно швидко перевірити бізнес-ідею [12]. Прикладами є:

- *Instagram — дозволяє переглядати збережені зображення в автономному режимі, але для завантаження нових потрібен інтернет;*
- *Gmail — поєднання веб-інфраструктури та нативної мобільної оболонки.*

Прогресивні веб-додатки (PWA) — це веб-додатки, які поєднують у собі функції звичайних сайтів та мобільних застосунків. Вони доступні через браузер, можуть працювати на будь-якому пристрої та підтримують функції офлайн-доступу. PWA доцільно використовувати у випадках, коли необхідно залучити більше трафіку, забезпечити надійний досвід для електронної комерції або скоротити витрати на розробку, оскільки вони дешевші за нативні додатки [13]. Прикладами PWA є:

- *Tinder — версія PWA покращила залученість користувачів порівняно з нативною;*

- *Uber — створений для підтримки старих пристроїв із обмеженими ресурсами;*
- *Pinterest — розробка PWA була обрана після виявлення низької продуктивності їхнього веб-сайту;*
- *Starbucks та AliExpress — для покращення мобільного досвіду покупців.*

Таким чином, вибір методу розробки залежить від вимог до продуктивності, функціональності, бюджету та часових обмежень проекту.

1.3 Висновок до першого розділу

У даному розділі проведено дослідження предметної області електронної комерції, включаючи основні поняття, моделі транзакцій (B2C, B2B, C2C, C2B) та принципи функціонування онлайн-магазинів. Особливу увагу приділено аналізу методів розробки кросплатформних застосунків, серед яких розглянуто нативні, гібридні, PWA та кросплатформні рішення, а також їх переваги та недоліки.

Проаналізовано ключові критерії вибору технологій для створення мобільних застосунків, зокрема продуктивність, підтримку платформ, зручність у розробці та доступність сторонніх бібліотек. Особливий акцент зроблено на фреймворках Xamarin, React Native та Flutter, що є лідерами у сфері кросплатформної розробки.

Метою дослідження є визначення оптимальних підходів для створення кросплатформного інтернет-магазину, який забезпечує ефективну роботу на різних пристроях.

Для досягнення цієї мети було виконано такі завдання:

1. Проведено аналіз наукових джерел щодо електронної комерції та її моделей.
2. Досліджено основні підходи до створення кросплатформних застосунків.

3. Проаналізовано переваги та недоліки фреймворків Xamarin, React Native та Flutter.

4. Визначено ключові вимоги до функціональності майбутнього застосунку.

5. Побудова застосунок на базі фреймоворку Xamarin

6. Проведено тестування проекту.

Отримані результати стали основою для подальших етапів роботи, присвячених технічному обґрунтуванню вибору технологій та проектуванню кросплатформного інтернет-магазину.

РОЗДІЛ 2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ІНТЕРНЕТ-МАГАЗИНУ

2.1 Вибір програмних засобів для реалізації проекту

Щоб Інтернет-магазин був успішний на ринку треба охопитти найбільшу кількість можливих клієнтів тобто був доступний для широкого кола користувачів, він має функціонувати як на пристроях з операційною системою Android, iPhone так і Windows. Традиційні методи розробки додатків для платформ Windows, Google та Apple значно відрізняються за мовами програмування, інструментами та API, що вимагає створення окремих нативних додатків для кожної платформи.

Вирішенням цієї проблеми є кросплатформова розробка, яка мінімізує зусилля, дозволяючи створити єдиний додаток, сумісний як з Windows, Android, так і з iOS. Для кросплатформової розробки використовується проміжна мова програмування, яка не належить до стандартних мов операційних систем пристроїв. Після цього додаток компілюється для відповідних платформ за допомогою фреймворків, таких як Xamarin, React Native або Flutter.

Цей підхід дозволяє використовувати частину, більшість або весь програмний код на кількох цільових платформах, включаючи iOS та Android. Щопівроку з’являються нові кросплатформові фреймворки, але найбільш популярними залишаються React Native, Flutter та Xamarin.

2.1.1 Xamarin

Xamarin був створений у 2011 році Натом Фрідманом, Мігелем де Іказа та Джозефом Хіллом як незалежна платформа для кросплатформної розробки мобільних застосунків. Придбання Microsoft у 2016 році зробило фреймворк ще більш популярним, підвищивши довіру до нього серед розробників завдяки інтеграції з екосистемою компанії. Основною метою створення Xamarin було

спрощення процесу розробки мобільних додатків, оскільки раніше для створення застосунків для Android та iOS доводилося розробляти окремі нативні рішення, що ускладнювало та здорожувало процес.

Завдяки Xamarin розробники отримали змогу створювати єдиний застосунок для кількох платформ одночасно, зберігаючи при цьому продуктивність і нативний вигляд інтерфейсу. Фреймворк пропонує зручні інструменти для прискорення розробки, серед яких Shell, Hot Reload, Hot Restart і Visual. Вони дозволяють:

- Описувати візуальну ієрархію додатка в одному місці;
- Використовувати спрощені елементи для навігації;
- Вбудовано використовувати пошукові компоненти.

Xamarin відповідає стандартам нативної розробки, що робить його привабливим для багатьох розробників. Однак, для додатків із високим рівнем графіки цей фреймворк може бути менш ефективним через різні підходи до візуального компонування на iOS та Android.

Архітектура Xamarin включає засоби для візуального дизайну, зокрема IDE для iOS із підтримкою X-Code. Фреймворк підтримує LINQ для роботи з колекціями, а також використання кастомних делегатів та подій, що дозволяє уникнути обмежень традиційних мов Objective-C та Java.

Xamarin підтримує архітектури MVC та MVVM, що робить його гнучким у використанні. MVVM (Model-View-ViewModel) дозволяє писати спільний код для бізнес-логіки, тоді як MVC (Model-View-Controller) сприяє чіткому розмежуванню між логікою представлення та обробкою даних, оптимізуючи процес розробки.

За допомогою Xamarin.Forms можна створювати кросплатформні застосунки для iOS, Android та Windows, використовуючи спільний код для інтерфейсу користувача. Ця технологія автоматично перетворює елементи на платформно-орієнтовані під час виконання, що особливо зручно для бізнес-додатків, де важлива швидкість розробки.

Переваги використання Xamarin:

- Єдина мова програмування: Використання C# дозволяє створювати додатки для кількох платформ одночасно без необхідності перемикання між середовищами розробки [14].
- Велика спільнота: Понад 50 000 розробників і 3000 компаній активно використовують Xamarin.
- Повторне використання коду: Можливість використання до 96% єдиного коду для Android та iOS.
- Інтеграція з Visual Studio: Фреймворк є вбудованим у середовище Visual Studio, що забезпечує зручний процес розробки.
- Нативний інтерфейс: Додатки зберігають нативний вигляд завдяки використанню платформно-орієнтованих компонентів.
- Підтримка нативних API: Можливість підключення до нативних бібліотек і API конкретної платформи для розширення функціональності.

Недоліки використання Xamarin:

- Висока вартість для компаній: Хоча фреймворк безкоштовний для фізичних осіб, корпоративним клієнтам необхідно купувати ліцензію на Visual Studio.
- Обмеження у створенні складних графічних інтерфейсів: Не рекомендовано для додатків із насиченою графікою через різні підходи до візуального рендерингу.
- Обмежений доступ до бібліотек: Деякі сторонні бібліотеки можуть бути недоступними для Xamarin.
- Часозатратний процес створення UI: Через використання проміжної мови створення користувацького інтерфейсу може бути тривалішим [8].

Відомі додатки, розроблені за допомогою Xamarin:

- HCl Connections
- American Cancer Society
- Fox Airports
- Alaska Airlines

Xamarin залишається одним із провідних рішень для кросплатформної розробки завдяки потужним інструментам, великій спільноті та високій продуктивності.

2.1.2 React Native

React Native — це один із найвідоміших кросплатформних фреймворків для розробки мобільних застосунків, створений Facebook у 2015 році. Його запуск справив справжній прорив у сфері кросплатформної розробки, адже він запропонував можливість створення застосунків для кількох платформ одночасно, використовуючи єдину базу коду. Сьогодні цей фреймворк залишається лідером серед кросплатформних рішень і активно використовується для створення застосунків під Android, Android TV, iOS, macOS та інші
React Native був розроблений для вирішення проблеми високої складності створення мобільних застосунків, яка виникала через необхідність окремої розробки для кожної платформи. Мета фреймворку полягала у створенні рішення, яке дозволило б використовувати спільний код для Android та iOS, зберігаючи при цьому високу продуктивність і нативний вигляд інтерфейсу.

Фреймворк швидко завоював популярність завдяки зручності та доступності для веб-розробників, які вже знайомі з React.js. Його підтримує велика спільнота: понад 90,5 тисяч зірок на GitHub і понад 3000 активних контриб'юторів, що постійно вдосконалюють фреймворк.

React Native базується на архітектурі Flux, що передбачає односпрямований потік даних, спрощуючи керування станом застосунку. Основний принцип — використання компонентів JavaScript для створення елементів інтерфейсу, які потім транслюються у нативні компоненти відповідних платформ.

Для забезпечення високої продуктивності фреймворк підтримує створення власних нативних модулів, написаних на Swift, Kotlin або Java, що дозволяє обробляти складні операції більш ефективно.

React Native також підтримує стилізацію за допомогою підходу, подібного до CSS, що робить його зручним для веб-розробників.

Основні переваги React Native:

- Кросплатформність: До 80% кодової бази можна використовувати повторно для різних платформ, що значно прискорює процес розробки.
- Продуктивність: Використання GPU (графічного процесора) для рендерингу інтерфейсу, що покращує швидкість відображення графіки у порівнянні з фреймворками, які покладаються на CPU.
- Гарячий перезапуск (Hot Reload): Можливість бачити зміни у коді в реальному часі без необхідності перезавантаження додатка.
- Широка підтримка спільноти: Велика кількість бібліотек, готових рішень та активна база розробників спрощують вивчення фреймворку.
- Інтуїтивно зрозумілий підхід: React Native легко засвоїти, особливо для розробників із досвідом роботи з React.js.
- Візуальний інтерфейс: Додатки, створені за допомогою React Native, візуально схожі на нативні, забезпечуючи високу якість UI.
- Expo: Інструмент, що значно спрощує доступ до нативних API без необхідності писати додатковий код для платформоспецифічних функцій.

Обмеження та недоліки React Native:

- Великі розміри додатків: Для Android-додатків потрібно включати вбудовану бібліотеку JavaScript, що збільшує їх розмір. iOS-додатки мають менший розмір, але все одно можуть бути більшими за нативні аналоги.
- Неповна кросплатформність: Хоча значна частина коду є спільною, деякі функції потребують написання окремих компонентів для Android та iOS.

- Менша продуктивність для складної графіки: React Native може поступатися нативним рішенням у продуктивності при обробці великих обсягів графіки та анімацій.
- Залежність від нативних платформ: Іноді фреймворк не встигає за оновленнями платформ, що може викликати труднощі з підтримкою додатків.
- Ускладнене налагодження: Процес відлагодження може бути тривалішим, особливо при розробці для Android [9].

Відомі додатки, створені на React Native:

- Pinterest
- Skype
- Tesla
- Wix
- Uber Eats
- Bloomberg

React Native залишається одним із найкращих інструментів для кросплатформної розробки мобільних застосунків завдяки потужному функціоналу, гнучкості та широкій підтримці спільноти. Його переваги, такі як спільна кодова база, гаряче оновлення та глибока інтеграція з нативними платформами, роблять його ідеальним вибором для багатьох компаній. Проте для ресурсомістких додатків із важкими графічними об'єктами може знадобитися комбінування React Native з нативним кодом або використання інших рішень.

2.1.3 Flutter

Flutter — це потужний безкоштовний фреймворк з відкритим вихідним кодом для кросплатформної розробки мобільних застосунків, створений та підтримуваний компанією Google. Він дозволяє створювати нативні інтерфейси для платформ Android, iOS та інших, використовуючи єдину кодову базу. Завдяки своєму інноваційному підходу, Flutter швидко набирає

популярності серед розробників і впритул наблизився до React Native за рівнем використання. Станом на вересень 2021 року фреймворк має понад 82,4 тисячі зірок на GitHub та активно підтримується більш ніж 500 контриб'юторами.

Flutter був офіційно представлений Google у лютому 2018 року на Всесвітньому мобільному конгресі, а перша стабільна версія вийшла 5 грудня 2018 року. Відтоді Google активно інвестує у розвиток цього фреймворку, сприяючи його перетворенню на комплексну та надійну екосистему для кросплатформної розробки.

Хоча спільнота Flutter поступається за масштабами React Native, вона активно зростає. Розробники фреймворку залучені до обговорень на сайтах QA та форумах, що сприяє поширенню технології серед новачків та досвідчених інженерів.

Flutter використовує архітектуру Reactive UI (реактивний інтерфейс користувача) та реалізує підхід односпрямованого потоку даних, натхненний бібліотеками Flux і RefluxJS.

Однією з причин високої продуктивності Flutter є використання мови програмування Dart, яка компілюється безпосередньо у C-бібліотеки, забезпечуючи швидкість роботи, наближену до нативного коду. Це усуває необхідність у JavaScript-містку, що є характерним для інших фреймворків.

Важливою особливістю Flutter є система Hot Reload, яка дозволяє розробникам миттєво бачити результати змін у коді, що значно пришвидшує тестування та налагодження застосунків.

Flutter базується на використанні віджетів — модульних компонентів, які відповідають за візуальні елементи інтерфейсу. Фреймворк має широкий набір вбудованих віджетів для Android (Material Design) та iOS (Cupertino), що дозволяє створювати додатки з нативним виглядом для обох платформ.

Проте залежність від вбудованих віджетів призводить до збільшення розміру застосунку: навіть найпростіший "Hello World" займає приблизно 6,7 МБ через використання вбудованих графічних бібліотек.

Основні переваги:

- Hot Reload: Миттєвий перегляд змін у коді без необхідності перезавантаження додатка.
- Висока продуктивність: Завдяки компіляції Dart у нативний С-код фреймворк забезпечує швидкодію, наближену до нативних застосунків.
- Кросплатформність: Можливість повторного використання до 90% кодової бази між Android та iOS.
- Нативний UI: Віджети Material Design для Android та Cupertino для iOS створюють нативний вигляд застосунків.
- Підтримка Google: Постійний розвиток фреймворку з боку Google та активна спільнота.
- Гнучкість для MVP: Ідеальний для швидкої розробки MVP завдяки спрощеному створенню інтерфейсів.
- Вбудовані віджети: Фреймворк має все необхідне для створення зручного та функціонального інтерфейсу "з коробки".
- Dart: Об'єктно-орієнтована мова програмування, яка легко засвоюється навіть розробниками без досвіду в ній.

Основні недоліки:

- Великий розмір застосунків: Через використання вбудованих бібліотек мінімальний розмір додатків становить 4-6 МБ.
- Обмежена підтримка платформ: Flutter не підтримує Android TV та Apple TV.
- Недостатня кількість сторонніх бібліотек: Деякі функції, доступні у нативних платформах, ще не реалізовані у Flutter.
- Залежність від віджетів: Оскільки фреймворк використовує власні віджети, а не системні, можливі відмінності у вигляді та поведінці на різних платформах [7].

Відомі додатки, створені на Flutter:

- Google Ads
- Alibaba
- Tencent

Flutter є одним із найсучасніших та потужних фреймворків для кросплатформної розробки мобільних додатків, пропонуючи високопродуктивні рішення, зручність у використанні та підтримку з боку Google. Завдяки мові Dart, системі Hot Reload та широкому набору віджетів, Flutter чудово підходить для швидкої розробки як комерційних додатків, так і MVP.

Попри такі недоліки, як збільшений розмір додатків та обмежена кількість бібліотек, Flutter залишається однією з найкращих платформ для створення сучасних мобільних додатків, які виглядають та працюють нативно.

Кожен із фреймворків має унікальні характеристики, що можуть впливати на вибір інструменту залежно від вимог проєкту, рівня продуктивності, складності інтерфейсу, можливостей повторного використання коду та простоти освоєння для розробників. React Native відзначається гнучкістю та великою спільнотою, Flutter — високою продуктивністю та багатим набором віджетів, тоді як Xamarin забезпечує глибоку інтеграцію з .NET та C#.

Нижче наведене детальне порівняння цих фреймворків за ключовими критеріями, що допоможе вибрати оптимальне рішення для конкретного завдання.

Таблиця 2.1 Порівняння крос-платформних технологій

Критерій	React Native	Flutter	Xamarin
Дата запуску	2015, Facebook	2018, Google	2011, Нат Фрідман, придбаний Microsoft у 2016
Мова програмування	JavaScript + React	Dart	C#
Архітектура	Flux (односпрямований потік даних)	Reactive UI (односпрямований потік даних)	MVVM, MVC
Продуктивність	Висока, але залежить від JavaScript-містка	Висока, компіляція у C-бібліотеки	Висока, майже нативна
Кросплатформність	80% повторного використання коду	90% повторного використання коду	До 96% повторного використання
UI-компоненти	Використовує нативні компоненти платформ	Власні віджети (Material, Cupertino)	Платформно-орієнтовані елементи
Простота вивчення	Легкий для веб-розробників з досвідом React	Вимагає вивчення Dart	Підходить для C# розробників
Hot Reload	Так, швидке оновлення коду	Так, дуже швидке оновлення коду	Так, Hot Reload

Підтримка платформ	Android, iOS, macOS, Windows	Android, iOS, macOS, Web	Android, iOS, Windows, macOS
Великий розмір додатків	Відносно великий через JS-місток	Великий (мін. 4-6 МБ через віджети)	Великий через вбудовані бібліотеки
Нативність інтерфейсу	Висока, нативні компоненти	Висока, але через власні віджети	Висока, використання нативних елементів
Підтримка компанії	Facebook	Google	Microsoft
Бібліотеки та модулі	Величезна кількість сторонніх бібліотек	Обмежена кількість, але активно росте	Широкий набір, інтеграція з .NET
Основні проекти	Pinterest, Skype, Tesla, Uber Eats	Google Ads, Alibaba, Tencent	Alaska Airlines, Fox Airports
Сильні сторони	- Велика спільнота	- Висока продуктивність	- Глибока інтеграція з .NET
	- Простота освоєння	- Красивий UI	- Підтримка C#
	- Гнучкість	- Ідеально для MVP	- Велика кросплатформність
Слабкі сторони	- JS-міст знижує продуктивність	- Великий розмір додатків	- Висока вартість для компаній

	- Відставання у підтримці нових фіч платформ	- Обмежена підтримка платформ	- Погано підходить для складної графіки

Вибір між React Native, Flutter та Xamarin залежить від специфіки проєкту та вимог до продуктивності, інтерфейсу та кросплатформності. React Native є оптимальним для швидкого запуску проєктів. Flutter пропонує високу продуктивність та відмінний візуальний інтерфейс, що робить його чудовим вибором для MVP та застосунків із багатим UI. Xamarin, у свою чергу, забезпечує найкращу інтеграцію з екосистемою Microsoft і підходить для C# розробників, але може бути менш ефективним для додатків зі складною графікою. Кожен із фреймворків має свої сильні та слабкі сторони, тому вибір слід базувати на технічних потребах, бюджеті та досвіді команди.

2.2 Вимоги до програмної реалізації

Вимоги до програмної реалізації визначають ключові технічні та функціональні характеристики, необхідні для створення ефективного програмного продукту у вигляді кросплатформного інтернет-магазину. Визначення цих параметрів забезпечує відповідність функціональних можливостей потребам кінцевих користувачів, високий рівень продуктивності та відповідність сучасним стандартам розробки програмного забезпечення.

Функціональні вимоги

Функціональні вимоги передбачають базові можливості, необхідні для забезпечення основної функціональності застосунку:

- Каталогізація товарів: відображення товарів за категоріями, можливість перегляду детальної інформації про товар.

- Управління замовленнями: оформлення, редагування, скасування замовлень та перегляд історії покупок.
- Пошук і фільтрація: можливість пошуку за ключовими словами, фільтрація за категоріями, ціною тощо.

Нефункціональні вимоги

Нефункціональні вимоги спрямовані на забезпечення стабільності, продуктивності та зручності у використанні:

- Продуктивність
- Надійність
- Юзабіліті
- Кросплатформенність: підтримка Android, iOS та Windows.

Архітектурні вимоги

Для забезпечення ефективної організації коду необхідно використовувати архітектурний патерн MVVM (Model-View-ViewModel), що передбачає:

- Чітке розмежування: виділення окремих модулів для моделей даних, обробки бізнес-логіки та відображення інтерфейсу.
- Структура проекту: організація файлів за папками Models, ViewModels, Views, Services, Components.

Технологічні вимоги

Реалізація проекту має базуватися на сучасних інструментах розробки:

- Фреймворк: .NET MAUI або Xamarin.
- Мова програмування: C#.
- Середовище розробки: Visual Studio.

Чітке дотримання зазначених вимог забезпечить створення стабільного, продуктивного та зручного для користувачів програмного продукту, здатного ефективно функціонувати в мультиплатформенному середовищі та відповідати сучасним стандартам програмної розробки.

2.3 Висновки до другого розділу

У даному розділі було проведено аналіз засобів для реалізації інтернет-магазину, зокрема програмних рішень для створення кросплатформних застосунків, таких як Xamarin, React Native та Flutter. Було здійснено порівняльний аналіз цих інструментів за критеріями функціональності, продуктивності, підтримки патернів проектування та зручності використання.

Фреймворк Xamarin обрано основним середовищем для реалізації інтернет-магазину. Такий вибір обумовлений його інтеграцією з екосистемою .NET, використанням мови програмування C#, можливістю створення спільної кодової бази для кількох платформ, підтримкою патерну MVVM, а також високим рівнем продуктивності для бізнес-додатків [11].

Для досягнення поставлених цілей було виконано такі завдання:

1. Проаналізовано доступні програмні засоби для розробки інтернет-магазину.
2. Оцінено функціональні можливості Xamarin, React Native та Flutter.
3. Обґрунтовано вибір Xamarin як оптимального середовища для створення кросплатформного додатка.

Результати даного дослідження стали основою для подальших етапів роботи, присвячених проектуванню та програмній реалізації інтернет-магазину.

РОЗДІЛ 3. РОЗРОБКА КРОС-ПЛАТФОРМНОГО ДОДАТКУ "ІНТЕРНЕТ-МАГАЗИН"

3.1 Актуальність розробки в сучасних умовах

Розробки крос-платформного «інтернет-магазин» в сучасних умовах зумовлена стрімким розвитком цифрових технологій та потребою в автоматизації рутинних процесів. В епоху цифровізації, коли обсяги даних зростають експоненціально, ефективні програмні рішення стають ключовим фактором для підвищення продуктивності та оптимізації робочих процесів.

Зростання вимог до швидкості обробки інформації, надійності зберігання даних і зручності користувацьких інтерфейсів робить розробку сучасного програмного забезпечення надзвичайно важливою. Це сприяє поліпшенню бізнес-процесів, зменшенню витрат часу на виконання завдань та мінімізації помилок, що виникають через людський фактор.

Особливої актуальності набуває розробка систем, які базуються на сучасних архітектурних підходах та забезпечують гнучкість, масштабованість і можливість інтеграції з іншими сервісами. Використання технологій, таких як хмарні обчислення, мікросервісна архітектура та автоматизоване тестування, дозволяє створювати стабільні та безпечні рішення, що відповідають потребам різних галузей.

Крім того, актуальність розробки полягає у підвищенні доступності технологій для користувачів різного рівня підготовки. Простота використання, адаптивний дизайн та багатоплатформність стають важливими критеріями успішного програмного продукту, сприяючи розширенню аудиторії користувачів.

Таким чином, у сучасних умовах розробка програмного забезпечення є не лише технічною задачею, а й важливим інструментом для забезпечення конкурентоспроможності компаній, покращення якості послуг і створення нових можливостей для розвитку бізнесу та суспільства в цілому.

3.2 Структура проекту

Додаток частково розроблений за архітектурним патерном MVVM (Model-View-ViewModel), який є стандартним для застосунків та Xamarin у середовищі .NET. Цей патерн забезпечує чітке розділення логіки обробки даних, представлення інтерфейсу користувача та бізнес-логіки, що сприяє спрощенню тестування та підтримки коду.

Структура проєкту (рис. 3.1) містить кілька ключових папок, які відповідають принципам патерну MVVM [15]. Domain містить моделі даних і класи, що відповідають за опис бізнес-логіки, таких як користувачі, товари або замовлення. Вона визначає об'єкти, які використовуються у всьому додатку для обробки та зберігання інформації.

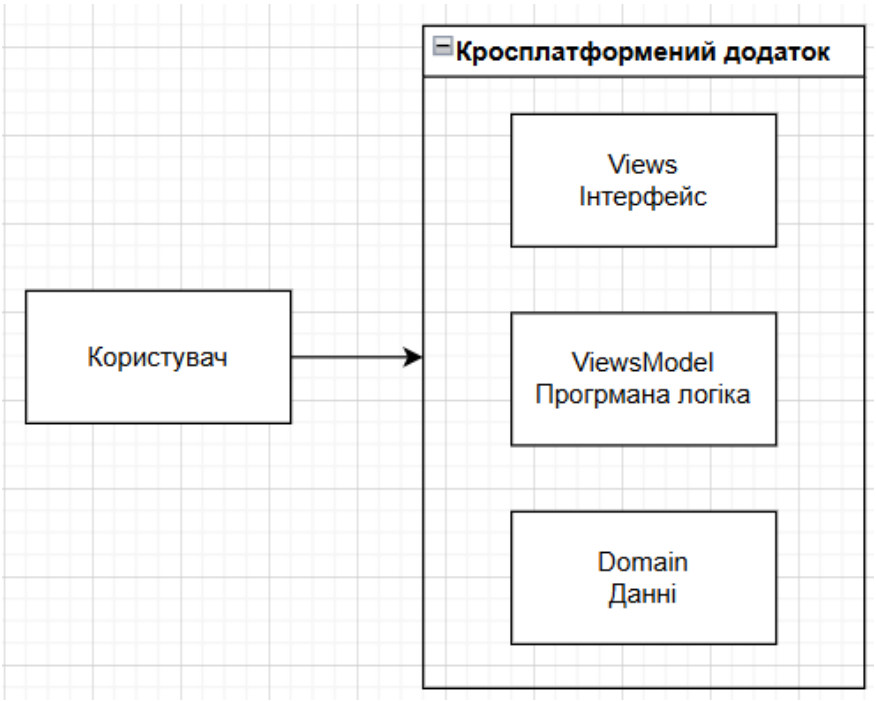

Рисунок 3.1 – Структура проекту

ViewModels містить класи, які виконують роль посередника між моделлю та представленням. Вони відповідають за обробку команд, підготовку даних для відображення у вигляді властивостей, а також за

зв'язування даних (Data Binding) з інтерфейсом користувача. Це забезпечує можливість розділення логіки обробки даних та їх візуалізації.

Views містить XAML-файли, які описують вигляд користувацького інтерфейсу. Вони пов'язані з відповідними ViewModel за допомогою механізму Binding, що дозволяє автоматично оновлювати інтерфейс при зміні даних у ViewModel.

Папка Components містить багаторазово використовувані елементи інтерфейсу, такі як користувацькі контроли або діалогові вікна. Це сприяє повторному використанню коду та полегшує управління інтерфейсними елементами.

Services містить сервіси для обробки даних, підключення до зовнішніх API або роботи з базами даних. Вона може містити також класи для реалізації принципу Dependency Injection (DI), що дозволяє інверсію управління залежностями та спрощує тестування компонентів.

3.3 Розробка додатку

Для розробки додатку LeonaStore за представленою структурою, необхідно правильно розподілити функціонал між основними частинами проєкту, щоб забезпечити чітке розмежування відповідальностей та відповідність архітектурному патерну.

Для реалізації функціональних можливостей класу:

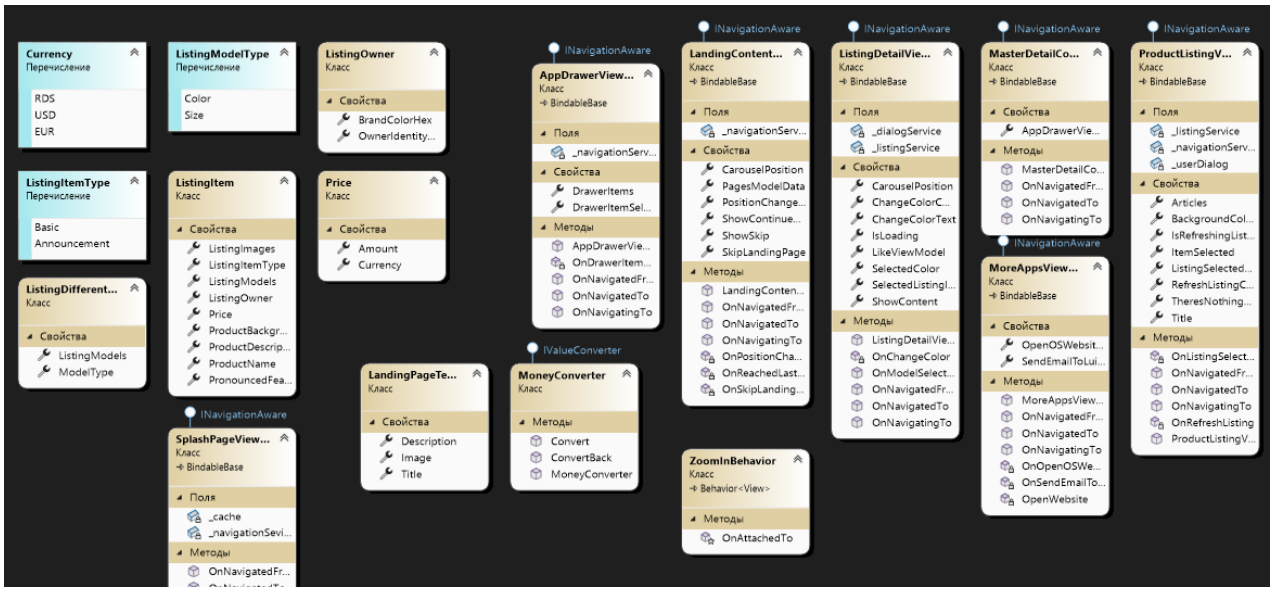

Рисунок 3.2 – Діаграма класів

Спочатку потрібно розглянути, яким чином реалізовані саме функціональні можливост, так додаток відповідає на запити. Для цього використовується група класів ViewModels.

Класи ViewModels відповідають за обробку дій користувача, таких як натискання кнопок, введення даних у форми, вибір елементів у списках. Крім того, ViewModels містять властивості, які використовуються для двостороннього зв'язування даних (Two-way Data Binding). Це означає, що будь-яка зміна даних у моделі автоматично відображається в інтерфейсі, і навпаки — зміни у UI оновлюють відповідні дані у ViewModel.

Важливим аспектом є також підхід до тестування. Оскільки ViewModels не містять коду, пов’язаного із графічним інтерфейсом, їх можна легко тестувати окремо, використовуючи модульні тести для перевірки бізнес-логіки, таких як перевірка правильності виконання команд або оновлення даних після отримання результатів від сервісу (рис. 3.3).

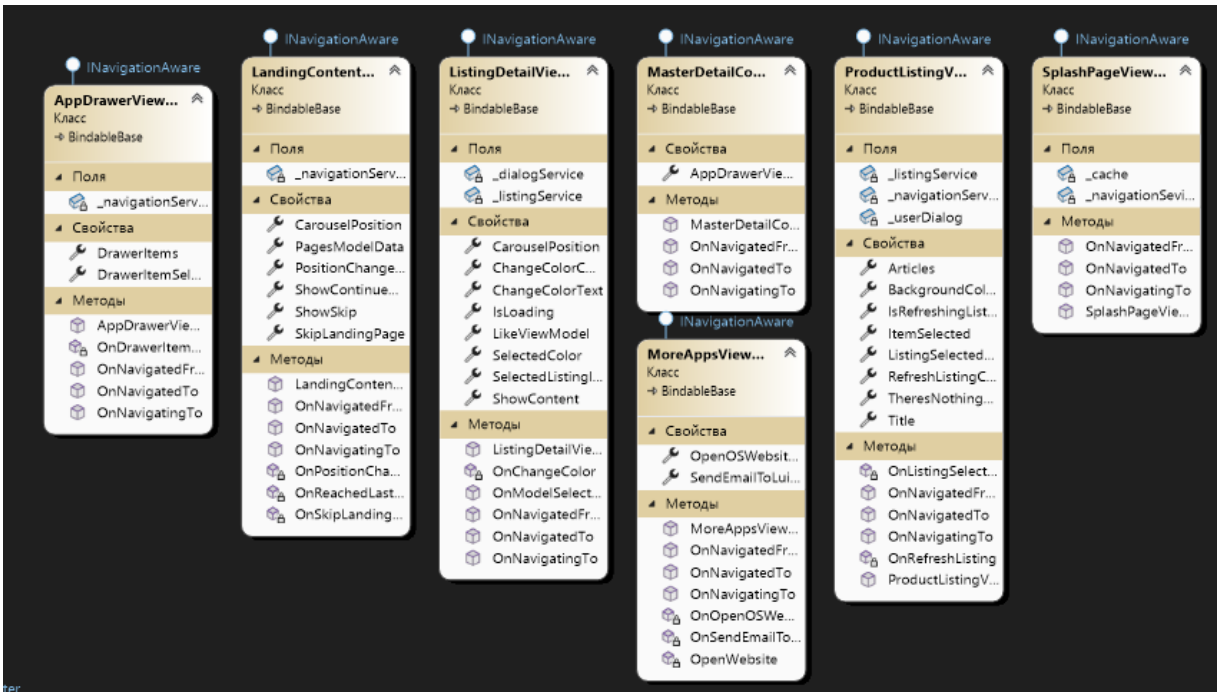

Рисунок 3.3 – Група класів ViewModels

Клас AppDrawerViewModel (рис. 3.4) є частиною рівня ViewModel у застосунку LeonaStore і відповідає за управління навігаційним меню (App Drawer). Він реалізує принципи патерну MVVM (Model-View-ViewModel) та є частиною основного компоненту для обробки взаємодії користувача з бічним меню додатка.

Основним завданням цього класу є зберігання списку елементів меню, які користувач може обирати для навігації між різними екранами застосунку. Це реалізовано через властивість DrawerItems, яка ініціалізується при першій навігації до цього ViewModel. У списку DrawerItems створюються об'єкти типу DrawerItem зі значеннями, такими як Text (назва пункту меню), Icon (шлях до іконки) та ScreenNavigateTo (рядок для навігації до певного екрану).

Рисунок 3.4 – Клас AppDrawerViewModel

Клас реалізує інтерфейс INavigationAware, що є частиною бібліотеки Prism і дозволяє виконувати певні дії під час навігації. Метод OnNavigatedTo (рис. 3.5) використовується для ініціалізації списку DrawerItems, якщо він ще не був створений.

Для обробки натискань на пункти меню використовується команда DrawerItemSelectedCommand, яка створюється через Command<DrawerItem> з використанням асинхронного лямбда-виразу. Ця команда викликає метод OnDrawerItemSelected, який використовує сервіс навігації Prism (INavigationService) для переміщення між екранами застосунку. Конкретний екран для переходу визначається через властивість ScreenNavigateTo у переданому об'єкті DrawerItem.

```
public void OnNavigatedTo(NavigationParameters parameters)
{
    if (DrawerItems == null)
        DrawerItems = new List<DrawerItem>()
        {
            new DrawerItem
            {
                Text = "Home",
                Icon = "ic_home_black_24dp",
                ScreenNavigateTo = LeonaStore.Screens.ProductListing
            },
            new DrawerItem
            {
                Text = "Categories",
                Icon = "ic_subject_black_24dp",
                ScreenNavigateTo = LeonaStore.Screens.ProductListing
            },
            new DrawerItem
            {
                Text = "Today's Deals",
                Icon = "ic_trending_up_black_24dp",
                ScreenNavigateTo = LeonaStore.Screens.ProductListing
            },
            new DrawerItem
            {
                Text = "Your Orders",
                Icon = "ic_list_black_24dp",
                ShowSeparatorAfter = true,
                ScreenNavigateTo = LeonaStore.Screens.ProductListing
            },
            new DrawerItem
            {
                Text = "Customer Service",
                Icon = "ic_help_black_24dp",
                ScreenNavigateTo = LeonaStore.Screens.ProductListing
            },
            new DrawerItem
            {
                Text = "Settings",
                Icon = "ic_settings_black_24dp",
                ScreenNavigateTo = LeonaStore.Screens.ProductListing
            }
```

Рисунок 3.5 – Метод OnNavigatingTo

Особливістю цього ViewModel є чітке розділення логіки від інтерфейсу користувача. Він не містить жодних візуальних елементів, а лише зберігає дані та керує командами, які реагують на дії користувача. Властивість DrawerItems безпосередньо зв'язується (Binding) з елементами інтерфейсу у XAML-файлі, тоді як команда DrawerItemSelectedCommand прив’язана до події натискання на пункт меню.

AppDrawerViewModel забезпечує управління даними та взаємодією користувача, залишаючи відображення даних на рівні View.

Клас LandingContentPageViewModel є частиною рівня ViewModel у додатку та відповідає за управління логікою сторінки вітання (Landing Page). Цей клас використовується для управління даними, які відображаються на сторінці, а також для обробки дій користувача, таких як перехід до головної сторінки або перегортання сторінок каруселі.

Основною властивістю класу є PagesModelData, яка зберігає колекцію моделей даних для сторінок привітання (LandingPageTemplateModel). Кожен

об'єкт цього списку містить заголовок, зображення та опис, які відображаються на екрані для ознайомлення користувача з функціональними можливостями додатку. Дані ініціалізуються у методі OnNavigatedTo, де створюється три об'єкти LandingPageTemplateModel із відповідними текстами та зображеннями.

```
public class LandingContentPageViewModel : BindableBase, INavigationAware
{
    public IList<LandingPageTemplateModel> PagesModelData { get; set; }

    public ICommand SkipLandingPage { get; set; }

    public int CarouselPosition { get; set; }

    readonly INavigationService _navigationService;

    public bool ShowSkip { get; set; }

    public bool ShowContinueButton { get; set; }

    public ICommand PositionChangedCommand { get; set; }

    public LandingContentPageViewModel(INavigationService navigationService)
    {
        _navigationService = navigationService;

        SkipLandingPage = new Command(OnSkipLandingPage);

        PositionChangedCommand = new Command<int>(OnPositionChanged);

        ShowSkip = true;
    }

    void OnPositionChanged(int position)
    {
```

Рисунок 3.6 – Клас LandingContentPageViewModel

Властивість CarouselPosition відповідає за поточний індекс активної сторінки у каруселі. Ця властивість пов'язана з логікою навігації між слайдами. Вона змінюється під час перегортання сторінок і впливає на відображення кнопок "Пропустити" (ShowSkip) і "Продовжити" (ShowContinueButton).

Для обробки дій користувача в класі визначені команди SkipLandingPage та PositionChangedCommand. Команда SkipLandingPage викликає метод OnSkipLandingPage, який здійснює навігацію до головної сторінки застосунку через сервіс навігації Prism (INavigationService). Команда PositionChangedCommand виконує метод OnPositionChanged, який змінює стан кнопок, залежно від поточної сторінки каруселі. Якщо користувач перегортає на останню сторінку (CarouselPosition дорівнює кількості сторінок мінус один), кнопка "Продовжити" стає активною, а кнопка "Пропустити" зникає.

Метод OnReachedLastPage використовується для відображення вітального повідомлення або автоматичного переходу до головної сторінки після досягнення останнього слайда.

Метод OnNavigatedTo (рис. 3.6) використовується для ініціалізації даних сторінки при навігації до даного представлення. Він перевіряє, чи вже була ініціалізована колекція PagesModelData, і якщо вона ще порожня, створює список об'єктів LandingPageTemplateModel. Ці об'єкти містять дані для кожного слайду сторінки привітання: заголовок, зображення та опис. Завдяки цьому метод OnNavigatedTo гарантує, що сторінка буде правильно заповнена даними при кожному її відкритті.

```
public void OnNavigatedTo(NavigationParameters parameters)
{
    if (PagesModelData == null)
    {
        PagesModelData = new List<LandingPageTemplateModel>
        {
            new LandingPageTemplateModel
            {
                Title = "Welcome To Leona",
                Image = "pocket",
                Description = "Buy and Sell like never seen before"
            },
            new LandingPageTemplateModel
            {
                Title = "Modern and Responsive",
                Image = "modern",
                Description = "All your favorite items in one place, instantly searchable"
            },
            new LandingPageTemplateModel
            {
                Title = "Ready to awesome up?",
                Image = "ready",
                Description = "Hit that button below!"
            }
        };
    }
}
```

Рисунок 3.6 – Метод OnNavigatedTo

Методи OnNavigatedFrom та OnNavigatingTo використовуються для збереження стану користувача або виконання перевірок перед навігацією на інші екрани.

Таким чином, LandingContentPageViewModel відповідає за управління сторінкою привітання, включаючи ініціалізацію даних для слайдів, обробку навігації, управління станом кнопок та обробку подій користувача під час взаємодії із каруселлю сторінок.

Наступний клас ListingDetailViewModel (рис. 3.7) є частиною рівня ViewModel і відповідає за керування логікою детальної сторінки товару. Він забезпечує обробку взаємодії користувача з деталями обраного товару, включаючи вибір кольору або моделі товару.

```
public class ListingDetailViewModel : BindableBase, INavigationAware
{
    public ICommand ChangeColorCommand { get; set; }

    readonly IPageDialogService _dialogService;

    readonly IListingService _listingService;

    public LikeViewModel LikeViewModel { get; set; }

    public ListingItem SelectedListingItem { get; set; }

    public string ChangeColorText { get; set; }

    public string SelectedColor { get; set; }

    public int CarouselPosition { get; set; }

    public bool IsLoading { get; set; }

    public bool ShowContent { get; set; }

    public ListingDetailViewModel(LikeViewModel likeViewModel,
                                  IPageDialogService dialogService,
                                  IListingService listingService)
    {
        _listingService = listingService;

        _dialogService = dialogService;

        LikeViewModel = likeViewModel;

        ChangeColorCommand = new Command(OnChangeColor);

        IsLoading = false;

        ShowContent = false;
    }
```

Рисунок 3.7 – Клас ListingDetailViewModel

Конструктор ListingDetailViewModel (рис. 3.7) отримує три залежності через параметри: LikeViewModel для управління функціональністю "вподобань", IPageDialogService для відображення діалогових вікон, а також IListingService для отримання даних про товар. У конструкторі ініціалізується команда ChangeColorCommand, яка прив'язана до методу OnChangeColor для обробки вибору кольору або моделі товару. Також у конструкторі за замовчуванням встановлюються прапори IsLoading та ShowContent, що контролюють відображення контенту під час завантаження даних.

Метод OnChangeColor (рис. 3.8) використовується для ініціалізації діалогового вікна з вибором кольору або моделі товару. Він створює список кнопок (IActionSheetButton) для кожного доступного варіанту товару,

отриманого зі списку SelectedListingItem.ListingModels.ListingModels. Після цього відображається діалогове вікно за допомогою DisplayActionSheetAsync, яке дозволяє користувачеві обрати один із варіантів. При натисканні на кнопку викликається метод OnModelSelected.

```
async void OnChangeColor()
{
    var modelList = new List<IActionSheetButton>();

    foreach (var model in SelectedListingItem.ListingModels.ListingModels)
    {
        modelList.Add(ActionSheetButton.CreateButton(model, () => OnModelSelected(mo
    }

    await _dialogService.DisplayActionSheetAsync($"Select {SelectedListingItem.Listi
}
```

Рисунок 3.8 – Функція OnChangeColor

Метод OnModelSelected встановлює значення вибраного кольору або моделі товару у властивість SelectedColor. Це значення потім може використовуватися для відображення вибраного варіанту товару у користувацькому інтерфейсі.

Метод OnNavigatedTo (рис. 3.9) є частиною інтерфейсу INavigationAware і використовується для завантаження даних при навігації на сторінку. Він встановлює прапор IsLoading у true, щоб показати індикатор завантаження. Далі він перевіряє, чи містяться у параметрах навігації (NavigationParameters) дані про ідентифікатор товару (ProductId). Якщо ідентифікатор знайдено, відбувається виклик методу GetListing із сервісу IListingService для отримання детальної інформації про товар. Після отримання даних оновлюються властивості SelectedListingItem, ChangeColorText (текст кнопки вибору моделі) та SelectedColor (обраний колір). По завершенні завантаження прапор IsLoading встановлюється у false, а ShowContent оновлюється для відображення даних товару.

```
public async void OnNavigatedTo(NavigationParameters parameters)
{
    IsLoading = true;

    ShowContent = !IsLoading;

    object productId = null;

    if (parameters.TryGetValue(ScreensNavigationParameters.ProductId, out productId))
    {
        SelectedListingItem = await _listingService.GetListing(productId as string);

        ChangeColorText = $"Change {SelectedListingItem.ListingModels.ModelType}";

        SelectedColor = SelectedListingItem.ListingModels?.ListingModels.FirstOrDefault();
    }

    IsLoading = false;

    ShowContent = !IsLoading;
}
```

Рисунок 3.9 – Метод OnNavigatedTo

Метод OnNavigatingTo призначений для обробки дій до моменту завершення навігації. Його можна використовувати для попередньої підготовки або перевірки стану сторінки перед відображенням, наприклад, для перевірки авторизації або перевірки, чи завантажені дані.

Метод OnNavigatedFrom також є частиною INavigationAware і використовується для збереження стану сторінки або скидання тимчасових даних при виході зі сторінки. Його можна розширити для збереження прогресу або передачі даних до інших сторінок.

Таким чином, клас ListingDetailViewModel забезпечує управління детальною сторінкою товару, включаючи отримання даних, обробку вибору моделей товару, навігацію між екранами та взаємодію з діалоговими вікнами.

Далі клас MasterDetailContainerViewModel виконує роль контейнера для управління головним інтерфейсом із використанням шаблону "Master-Detail". Його основним завданням є об'єднання функціональності навігаційного меню (AppDrawerViewModel) із основним вмістом сторінки, забезпечуючи єдину точку контролю для управління навігацією та відображенням контенту.

Конструктор класу MasterDetailContainerViewModel приймає як параметр об'єкт AppDrawerViewModel і зберігає його у відповідній властивості AppDrawerViewModel. Це забезпечує доступ до функціональності

навігаційного меню з основного контейнера, дозволяючи передавати дані між головною сторінкою та навігаційною панеллю.

```
public class MasterDetailContainerViewModel : BindableBase, INavigationAware
{
    public AppDrawerViewModel AppDrawerViewModel { get; set; }

    public MasterDetailContainerViewModel(AppDrawerViewModel drawerViewModel)
    {
        AppDrawerViewModel = drawerViewModel;
    }
```

Рисунок 3.10 – Клас MasterDetailContainerViewModel

Метод OnNavigatedFrom (рис. 3.11) реалізує частину інтерфейсу INavigationAware і викликається під час виходу зі сторінки. У цьому методі передається об'єкт NavigationParameters до вкладеного AppDrawerViewModel, що дозволяє передати необхідні дані або виконати збереження стану сторінки перед її закриттям. Це може бути корисним для збереження обраного пункту меню або закриття діалогових вікон.

```
public void OnNavigatedFrom(NavigationParameters parameters)
{
    AppDrawerViewModel.OnNavigatedFrom(parameters);
}

public void OnNavigatedTo(NavigationParameters parameters)
{
    AppDrawerViewModel.OnNavigatedTo(parameters);
}

public void OnNavigatingTo(NavigationParameters parameters)
{
    AppDrawerViewModel.OnNavigatingTo(parameters);
}
```

Рисунок 3.11 – Методи навігації

Метод OnNavigatedTo (рис. 3.11) також реалізує інтерфейс INavigationAware і викликається під час завантаження сторінки. Він передає параметри навігації до AppDrawerViewModel для ініціалізації елементів меню або підготовки сторінки до відображення. Наприклад, це може включати

завантаження списку категорій товарів або персоналізованого привітання для користувача після авторизації.

Метод OnNavigatingTo (рис. 3.11) викликається до завершення навігації на сторінку та може використовуватися для перевірки умов перед переходом. Його виклик через AppDrawerViewModel може включати перевірку доступу до сторінки, наприклад, перевірку, чи авторизований користувач перед відображенням контенту або чи всі необхідні дані завантажені.

Даний клас сприяє підтримці принципу інверсії залежностей, оскільки делегує більшу частину логіки управління навігацією вкладеному AppDrawerViewModel.

Наступним клас MoreAppsViewModel (рис. 3.12) призначений для управління функціональністю сторінки "Додаткові застосунки". Він реалізує базовий набір команд для взаємодії користувача із зовнішніми ресурсами, такими як відкриття веб-сайтів і відправка електронних листів.

```
public class MoreAppsViewModel : BindableBase, INavigationAware
{
    public ICommand SendEmailToLuisCommand { get; set; }

    public ICommand OpenOSWebsiteCommand { get; set; }

    public MoreAppsViewModel()
    {
        SendEmailToLuisCommand = new Command(OnSendEmailToLuis);

        OpenOSWebsiteCommand = new Command<string>(OnOpenOSWebsite);
    }
```

Рисунок 3.12 – Клас MoreAppsViewModel

Конструктор класу MoreAppsViewModel (рис. 3.12) ініціалізує дві команди: SendEmailToLuisCommand і OpenOSWebsiteCommand. Команда SendEmailToLuisCommand прив’язана до методу OnSendEmailToLuis, який викликає відкриття стандартного поштового клієнта для відправки листа розробнику. Команда OpenOSWebsiteCommand приймає рядок url як параметр і використовує його для виклику методу OnOpenOSWebsite, який відкриває вказану веб-сторінку у стандартному браузері пристрою.

Метод OnSendEmailToLuis (рис. 3.13) викликає Device.OpenUri, який відкриває поштовий клієнт із заздалегідь підготовленою електронною адресою " user@gmail.com ". Це дозволяє користувачеві швидко зв’язатися з розробником для зворотного зв’язку або запитів щодо застосунку.

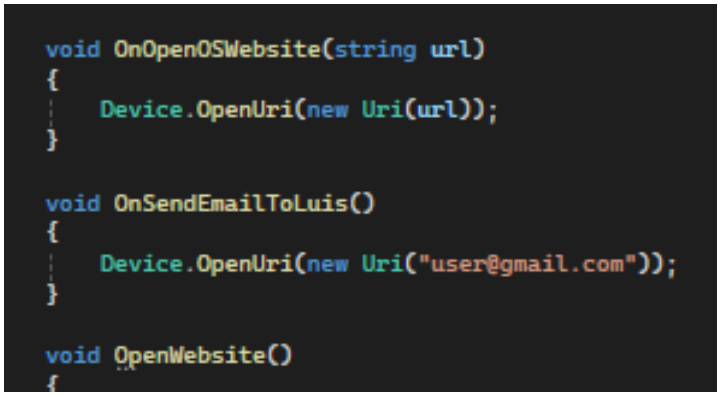

Рисунок 3.13 – Методи OnSendEmailToLuis, OnOpenOSWebsite

Метод OnOpenOSWebsite (рис. 3.13) також використовує Device.OpenUri, але вже для відкриття переданого URL у браузері пристрою. Це може бути використано для відкриття офіційного веб-сайту застосунку або інших сторінок, пов'язаних із розробником або продуктом.

Метод OpenWebsite створений для додаткової логіки відкриття веб-сайтів. Він використовується для відкриття конкретних сторінок додатку або реалізації логіки перевірки доступності ресурсу перед його відкриттям.

Метод OnNavigatedTo використовується для завантаження даних при переході на сторінку. Він розширений для ініціалізації контенту сторінки, завантаження списку інших застосунків розробника та підготовки рекламних матеріалів.

Метод OnNavigatedFrom виконує функцію очищення ресурсів або скидання стану сторінки при виході користувача. Це може включати скидання таймерів, збереження прогресу або відписку від подій.

Метод OnNavigatingTo призначений для підготовки сторінки до відображення перед завершенням навігації. Його можна розширити для

перевірки параметрів навігації, таких як передача URL або контактних даних для автоматичного відображення певної інформації при відкритті сторінки.

Клас MoreAppsViewModel забезпечує основні функції для взаємодії користувача із зовнішніми ресурсами, такими як веб-сторінки та електронна пошта.

Далі клас ProductListingViewModel (рис. 3.14) є частиною рівня ViewModel відповідає за управління логікою відображення списку товарів. Він реалізує функціональність завантаження, оновлення та навігації до детальної інформації про обраний товар, дотримуючись принципів патерну MVVM.

```
public class ProductListingViewModel : BindableBase, INavigationAware
{
    public string Title { get; set; }
    public Color BackgroundColor { get; set; }
    public IList<ListingItem> Articles { get; set; }
    public ICommand RefreshListingCommand { get; set; }
    public ICommand ListingSelectedCommand { get; set; }
    public bool IsRefreshingListing { get; set; }
    readonly INavigationService _navigationService;
    readonly IListingService _listingService;
    public ListingItem ItemSelected { get; set; }
    readonly IUserDialogs _userDialog;
    public bool TheresNothingToShow { get; set; }

    public ProductListingViewModel(INavigationService navigationService,
                                   IListingService listingService,
                                   IUserDialogs userDialog)
    {
        _userDialog = userDialog;

        _listingService = listingService;

        _navigationService = navigationService;

        RefreshListingCommand = new Command(async()=> await OnRefreshListing());

        ListingSelectedCommand = new Command<ListingItem>(OnListingSelected);
    }
```

Рисунок 3.14 – Клас ProductListingViewModel

Конструктор класу ProductListingViewModel (рис. 3.14) приймає три залежності: INavigationService для управління навігацією між сторінками, IListingService для отримання даних про товари та IUserDialogs для відображення діалогових повідомлень користувачу.

У конструкторі ініціалізуються команди RefreshListingCommand для оновлення списку товарів та ListingSelectedCommand для обробки натискання на окремий товар у списку.

Метод OnListingSelected (рис. 3.16) викликається при виборі користувачем товару зі списку. Він перевіряє, чи обраний товар не є null, після чого здійснює навігацію до сторінки з детальною інформацією про товар (Screens.ListingDetail). У параметрах передається назва товару (ProductId), що дозволяє сторінці деталізації отримати всі необхідні дані для відображення товару.

```
async void OnListingSelected(ListingItem item)
{
    if (item == null)
        return;

    await _navigationService.NavigateAsync($"{Screens.ListingDetail}",
                                           new NavigationParameters($"{ScreensNav

    ItemSelected = null;
}
```

Рисунок 3.15 – Метод OnListingSelected

Метод OnRefreshListing (рис. 3.16) відповідає за оновлення списку товарів. Спочатку він перевіряє, чи пристрій має підключення до інтернету за допомогою бібліотеки CrossConnectivity. Якщо з'єднання відсутнє, користувач отримує повідомлення через _userDialog.Alert, а завантаження скасовується. Якщо ж підключення активне, метод викликає сервіс IListingService для завантаження актуального списку товарів. Якщо список товарів порожній або недоступний, прапор TheresNothingToShow встановлюється у true.

```
async Task OnRefreshListing()
{

    if (!CrossConnectivity.Current.IsConnected)
    {
        _userDialog.Alert("You must be connected to the internet :(.", "Oh no!", "Ok!");

        IsRefreshingListing = false;

        TheresNothingToShow = true;

        return;
    }

    IsRefreshingListing = true;

    Articles = await _listingService.GetAllListings();

    TheresNothingToShow = Articles == null;

    IsRefreshingListing = false;
}
```

Рисунок 3.16 – Метод OnRefreshListing

Метод OnNavigatedTo (рис. 3.17) викликається при переході на сторінку списку товарів. Він встановлює значення заголовка сторінки (Title) та кольору фону (BackgroundColor). Якщо список товарів ще не завантажений, метод ініціалізує його завантаження через виклик OnRefreshListing. Якщо у параметрах навігації передано ProductId, здійснюється автоматичний перехід на сторінку детального перегляду товару.

```
public async void OnNavigatedTo(NavigationParameters parameters)
{
    Title = "Shop";

    BackgroundColor = Color.FromHex("F5F5F5");

    if(Articles == null)
        await OnRefreshListing();

    object productId;

    if (parameters.TryGetValue(ScreensNavigationParameters.ProductId, out productId))
    {
        await _navigationService.NavigateAsync($"{Screens.ListingDetail}",
                                               new NavigationParameters($"{ScreensNavig
    }
}
```

Рисунок 3.17 – Метод OnNavigatedTo

Метод OnNavigatedFrom може виконує дій перед виходом зі сторінки, скидання стану списку товарів або збереження даних користувача.

Метод OnNavigatingTo обробляє дії, які потрібно виконати до завершення переходу на сторінку.

Таким чином, ProductListingViewModel забезпечує повноцінне управління сторінкою списку товарів, включаючи завантаження даних, перевірку інтернет-з'єднання, обробку вибору товарів та навігацію між сторінками.

Останній клас SplashPageViewModel (рис. 3.18) відповідає за управління логікою екрану завантаження (Splash Screen). Він використовується для обробки початкової навігації після запуску застосунку та перевірки, чи користувач вперше відкриває додаток.

```
public class SplashPageViewModel : BindableBase, INavigationAware
{
    readonly INavigationService _navigationSevice;
    readonly ICache _cache;

    public SplashPageViewModel(INavigationService navigationService, ICache cache)
    {
        _cache = cache;

        _navigationSevice = navigationService;
    }
```

Рисунок 3.18 – Клас SplashPageViewModel

Конструктор класу SplashPageViewModel (рис. 3.18) приймає дві залежності: INavigationService для управління навігацією між сторінками та ICache для роботи з кешованими даними. Ці сервіси зберігаються у приватних полях _navigationService та _cache для подальшого використання в методах класу. Конструктор відповідає лише за ініціалізацію залежностей та не виконує додаткових дій.

Метод OnNavigatedTo (рис. 3.19) є основним у цьому класі та викликається автоматично під час завантаження сторінки. Він починає свою роботу з очікування (Task.Delay) на 2 секунди, що імітує екран завантаження, дозволяючи користувачеві бачити анімацію або логотип під час ініціалізації застосунку. Після завершення затримки метод перевіряє, чи користувач запускає додаток вперше. Це відбувається за допомогою виклику методу _cache.GetObjectAsync<bool>, який звертається до збережених даних у локальному сховищі через ключ CacheKeys.NewUserKey.

```
public async void OnNavigatedTo(NavigationParameters parameters)
{
    await Task.Delay(TimeSpan.FromSeconds(2));

    var isNotFirstTimeUser = await _cache.GetObjectAsync<bool>(CacheKeys.NewUserKey);

    if (isNotFirstTimeUser)
        await _navigationSevice.NavigateAsync(new
            Uri($"{Screens.AbsoluteURI}/{Screens.MasterDetailContainer}/{Screens.LeonaNavigationPage}/{
    else
        await _navigationSevice.NavigateAsync(new
            Uri($"{Screens.AbsoluteURI}/{Screens.LandingContentPage}", UriKind.Absolute));

    await _cache.Insert(CacheKeys.NewUserKey, true);
}
```

Рисунок 3.19 – Метод OnNavigatedTo

Якщо користувач уже відкривав додаток раніше (isNotFirstTimeUser дорівнює true), відбувається навігація на головний екран додатка (ProductListing) через використання сервісу _navigationService. У цьому випадку застосунок перенаправляє користувача на основний екран магазину, минаючи сторінку привітання. Якщо ж користувач відкриває застосунок вперше, виконується навігація до сторінки привітання (LandingContentPage).

Після завершення навігації метод виконує збереження у кеші (_cache.Insert) значення true для ключа CacheKeys.NewUserKey. Це гарантує, що при наступному відкритті застосунок пропустить сторінку привітання та перейде одразу до основного інтерфейсу.

Метод OnNavigatedFrom реалізує стандартний інтерфейс INavigationAware й використовується для очищення ресурсів або скидання тимчасових даних під час виходу зі сторінки.

Метод OnNavigatingTo також є частиною інтерфейсу INavigationAware і також використовується для попередньої обробки даних перед завершенням навігації на сторінку.

3.4 Розробка користувацького інтерфейсу

Для розробки візуально привабливого користувацького інтерфейсу в додатках, побудованих на платформі Xamarin.Forms або MAUI, використовується мова розмітки XAML (eXtensible Application Markup Language).

XAML також підтримує механізм прив’язування даних (Data Binding), що є ключовим аспектом для реалізації патерну MVVM (Model-View-ViewModel). За допомогою прив’язок можна пов’язувати елементи інтерфейсу безпосередньо із властивостями у ViewModel, що дозволяє автоматично оновлювати інтерфейс при зміні даних.

Важливою функцією XAML також є підтримка стилів та ресурсів. Визначення глобальних стилів дозволяє застосовувати однакове оформлення

для кількох елементів інтерфейсу, що сприяє покращенню візуальної єдності додатка.

Далі потрібно розглянути декілька вікон, що проектуються за допомогою XAML.

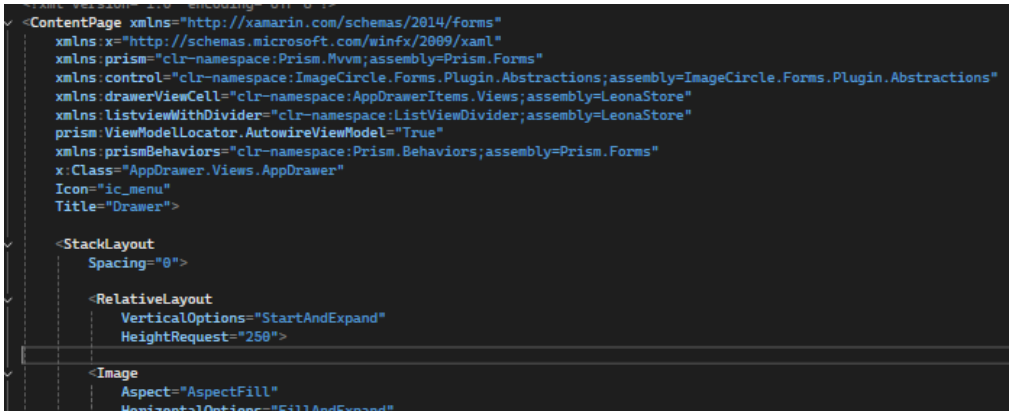

Рисунок 3.20 – Розмітка XAML

Наступна XAML-розмітка (рис. 3.20) представляє сторінку бічного меню (App Drawer) для застосунку, створену за допомогою Xamarin.Forms із використанням архітектурного патерну MVVM. Вона відповідає за візуальне оформлення та інтерактивність навігаційного меню, включаючи відображення профілю користувача та списку навігаційних пунктів.

Верхній рівень розмітки представлений елементом ContentPage, який є контейнером для основного вмісту сторінки. Використовується простір імен prism:ViewModelLocator.AutowireViewModel="True", що вказує фреймворку Prism автоматично створити екземпляр відповідного ViewModel (AppDrawerViewModel) для цієї сторінки. Сторінка має іконку меню (Icon="ic_menu") та заголовок Title="Drawer", які використовуються для відображення у заголовку сторінки або панелі навігації.

Всередині ContentPage знаходиться основний контейнер StackLayout (рис. 3.21) із вертикальним розміщенням елементів (Spacing="0"). Він містить два основних підконтейнери: RelativeLayout для відображення інформації про

профіль користувача та ListviewDivider для відображення списку пунктів меню.

```
<StackLayout
    Spacing="0">

    <RelativeLayout
        VerticalOptions="StartAndExpand"
        HeightRequest="250">

    <Image
        Aspect="AspectFill"
        HorizontalOptions="FillAndExpand"
        Source="https://cdn.pixabay.com/photo/2017/04/06/17/43/water-2208931_960_720.jpg"
        RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToParent, Property=Height, Factor=0,Constant=0}"
        RelativeLayout.XConstraint="{ConstraintExpression Type=RelativeToParent, Property=Width, Factor=0, Constant=0}"
        RelativeLayout.WidthConstraint="{ConstraintExpression Type=RelativeToParent, Property=Width,Factor=1,Constant=0}"
        RelativeLayout.HeightConstraint="{ConstraintExpression Type=RelativeToParent, Property=Height, Factor=1,Constant=0}"/>

    <Image
        Aspect="AspectFill"
        HorizontalOptions="FillAndExpand"
        BackgroundColor="Black"
        Opacity="0.4"
        RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToParent, Property=Height, Factor=0,Constant=0}"
        RelativeLayout.XConstraint="{ConstraintExpression Type=RelativeToParent, Property=Width, Factor=0, Constant=0}"
        RelativeLayout.WidthConstraint="{ConstraintExpression Type=RelativeToParent, Property=Width,Factor=1,Constant=0}"
        RelativeLayout.HeightConstraint="{ConstraintExpression Type=RelativeToParent, Property=Height, Factor=1,Constant=0}"/>

    <StackLayout
        HorizontalOptions="FillAndExpand"
        VerticalOptions="CenterAndExpand"
        Padding="20,0,0,0"
        Spacing="16"
        RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToParent, Property=Height, Factor=0,Constant=0}"
        RelativeLayout.XConstraint="{ConstraintExpression Type=RelativeToParent, Property=Width, Factor=0, Constant=0}"
        RelativeLayout.WidthConstraint="{ConstraintExpression Type=RelativeToParent, Property=Width,Factor=1,Constant=0}"
        RelativeLayout.HeightConstraint="{ConstraintExpression Type=RelativeToParent, Property=Height, Factor=1,Constant=0}">
```

Рисунок 3.21 – Контейнер StackLayout

Контейнер RelativeLayout використовується для створення секції з інформацією про користувача, висота якої обмежена параметром HeightRequest="250". В ньому спочатку розміщене зображення фонової картинки (Image) із властивістю AspectFill для повного заповнення області. Ця картинка завантажується з мережі через атрибут Source та URL-адресу. Поверх фонового зображення накладено ще один елемент Image із чорним напівпрозорим фоном (Opacity="0.4"), що створює затемнення для покращення читабельності тексту, який буде розміщений поверх цього фону.

Далі всередині RelativeLayout знаходиться StackLayout, який містить елементи для відображення профільної інформації користувача. Це зображення профілю, представлене елементом CircleImage із закругленими краями. Зображення має розмір 100x100 пікселів і прив'язане до ресурсу profile. Властивості BorderColor та BorderThickness вимикають обводку навколо зображення, а Aspect="AspectFit" гарантує, що зображення відображатиметься пропорційно.

Друга частина сторінки, яка відповідає за відображення пунктів меню, реалізована за допомогою спеціалізованого списку listviewWithDivider:ListviewDivider.

```
<listviewWithDivider:ListviewDivider
    ItemsSource="{Binding DrawerItems}"
    SeparatorColor="Transparent"
    HasUnevenRows="true"
    SeparatorVisibility="None"
    BackgroundColor="Transparent"
    IsPullToRefreshEnabled="false"
    HorizontalOptions="FillAndExpand"
    VerticalOptions="FillAndExpand">
    <ListView.ItemTemplate>
        <DataTemplate>
            <drawerViewCell:AppDrawerItem/>
        </DataTemplate>
    </ListView.ItemTemplate>
    <ListView.Behaviors>
        <prismBehaviors:EventToCommandBehavior EventName="ItemSelected"
            Command="{Binding DrawerItemSelectedCommand}" EventArgsParameterPath="SelectedItem"/>
    </ListView.Behaviors>
</listviewWithDivider:ListviewDivider>
</StackLayout>
ontentPage>
```

Рисунок 3.22 – Список listviewWithDivider:ListviewDivider

Цей компонент отримує дані зі списку DrawerItems, що прив'язані до властивості ItemsSource="{Binding DrawerItems}" у відповідному ViewModel. Параметри SeparatorColor="Transparent" та SeparatorVisibility="None" відключають стандартні роздільники елементів списку, а BackgroundColor="Transparent" робить фон прозорим, щоб зберегти візуальну цілісність оформлення.

Для відображення кожного елемента списку використовується шаблон DataTemplate, який посилається на кастомний елемент drawerViewCell:AppDrawerItem. Це, ймовірно, окремий користувацький елемент, що містить власний дизайн для кожного пункту меню, включаючи іконку та текст.

Важливим функціональним елементом є механізм обробки подій. У секції ListView.Behaviors застосовується поведінка prismBehaviors:EventToCommandBehavior, яка дозволяє прив’язати подію ItemSelected до команди DrawerItemSelectedCommand у ViewModel. Це

забезпечує автоматичне виконання команди при виборі пункту меню, що дозволяє відокремити логіку взаємодії від візуального представлення.

Наступна сторінка це сторінка товару (ListingDetail). Вона відповідає за візуальне відображення інформації про обраний товар, включаючи галерею зображень, назву, ціну, опис, а також інтерактивні елементи, такі як кнопка покупки та можливість змінювати вибраний варіант товару.

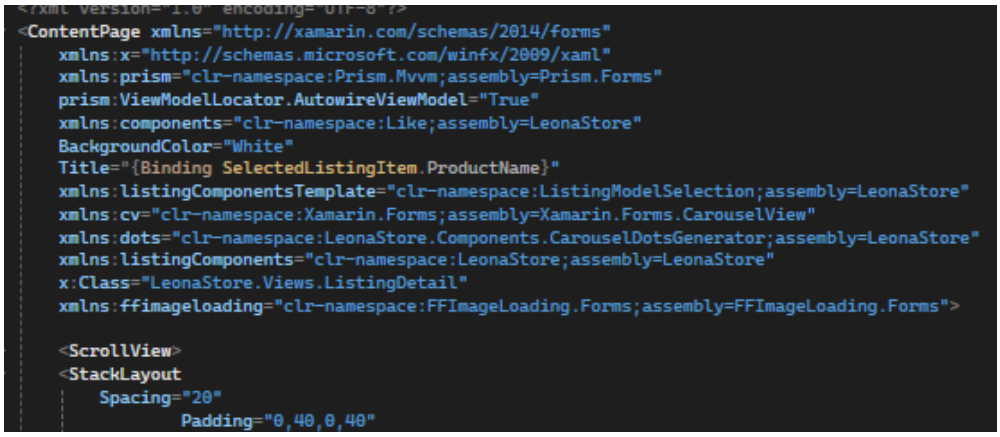

Рисунок 3.23 – Елемент ContentPage

Верхній рівень цієї сторінки представлений елементом ContentPage (рис. 3.23), де використовується системний параметр prism:ViewModelLocator.AutowireViewModel="True", що дозволяє автоматично зв'язати сторінку із відповідним ViewModel (ListingDetailViewModel). Атрибут Title="{Binding SelectedListingItem.ProductName}" встановлює динамічний заголовок сторінки, що оновлюється залежно від вибраного товару. Фон сторінки заданий через властивість BackgroundColor="White".

Основний вміст сторінки розміщено всередині ScrollView, що дозволяє прокручувати контент у випадку, якщо даних більше, ніж вміщається на екрані. Весь контент вкладений у StackLayout з вертикальним розміщенням елементів і відступами (Padding="0,40,0,40"), що забезпечує гармонійний простір між компонентами.

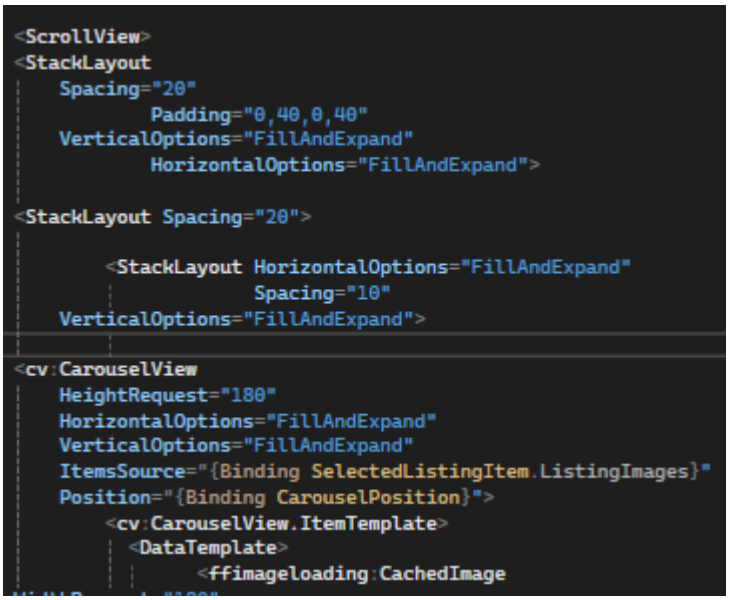

Рисунок 3.24 – Компонент ScrollView

Перша секція сторінки містить карусель зображень товару, реалізовану через CarouselView. Вона отримує дані зі списку зображень через ItemsSource="{Binding SelectedListingItem.ListingImages}". Для відображення кожного зображення використовується DataTemplate, який містить елемент ffimageloading:CachedImage. Це оптимізований компонент для завантаження зображень, який підтримує кешування та автоматичне масштабування (DownsampleToViewSize="true").

Під каруселлю зображень розміщений кастомний елемент dots:CarouselDotsGenerator, який створює індикатори поточної сторінки каруселі у вигляді точок. Кількість точок відповідає кількості зображень у колекції, а активна точка змінюється через двосторонній зв’язок CurrentPage="{Binding CarouselPosition}".

Далі розміщений ActivityIndicator, який служить для відображення процесу завантаження даних. Він стає видимим (IsVisible="{Binding IsLoading}") лише під час завантаження товару, а колір індикатора налаштовується через Color="{StaticResource leonaColor}".

Наступний блок представлений BoxView — це горизонтальна лінія, яка використовується як роздільник між секціями сторінки. Вона має прозорість

Opacity="0.2" і відображається лише тоді, коли завантаження завершено (IsVisible="{Binding ShowContent}").

Далі йде секція з основною інформацією про товар, включаючи назву, можливість поставити "вподобайку" та обрати колір товару (рис. 3.25). Текст "NEW ARRIVAL" відображається у вигляді підзаголовка, а нижче розташована назва товару (SelectedListingItem.ProductName) у великому шрифті (FontSize="26"). Поруч з назвою розміщений кастомний елемент components:LikeView, який прив’язаний до LikeViewModel і використовується для додавання товару у список обраного.

```
</StackLayout>
<StackLayout Spacing="16" Padding="40,0,40,0" IsVisible="{Binding ShowContent}">

    <Label FontAttributes="Bold" Text="NEW ARRIVAL"
        TextColor="{StaticResource leonaColor}"/>

    <StackLayout Orientation="Horizontal">

    <Label Text="{Binding SelectedListingItem.ProductName}"
        TextColor="Black"
        HorizontalOptions="StartAndExpand"
        VerticalOptions="CenterAndExpand"
        FontSize="26"/>

        <components:LikeView HorizontalOptions="EndAndExpand"
            BindingContext="{Binding LikeViewModel}"
            VerticalOptions="CenterAndExpand"/>

    </StackLayout>

    <StackLayout Orientation="Horizontal" IsVisible="{Binding ShowContent}">
```

Рисунок 3.25 – Секція для фідбеку

Наступний StackLayout містить елементи для вибору кольору товару. Властивість Text="{Binding SelectedColor}" відображає поточний вибраний колір, а текст ChangeColorText використовується як підпис для кнопки зміни кольору. Для обробки натискання на цей блок використовується TapGestureRecognizer, прив'язаний до команди ChangeColorCommand у ViewModel.

Секція з ціною товару (рис. 3.26) містить елемент Label для відображення ціни (SelectedListingItem.Price), де використовується спеціальний конвертер для форматування валюти (Converter={StaticResource

moneyConverter}). Поруч розташований кастомний компонент listingComponents:ListingModelSelection, який дозволяє користувачеві обрати варіанти товару, наприклад, розмір або конфігурацію. Візуальний шаблон для відображення цих варіантів заданий через DataTemplate, який містить компонент listingComponentsTemplate:ListingModelSelectionTemplate.

```
<StackLayout
        Orientation="Horizontal" IsVisible="{Binding ShowContent}">

        <Label TextColor="Black"
            Text="{Binding SelectedListingItem.Price, Converter={StaticResource moneyConverter}}"
            FontSize="18"/>

        <listingComponents:ListingModelSelection
            HorizontalOptions="EndAndExpand"
            ItemsSource="{Binding SelectedListingItem.PronouncedFeatures}">

        <listingComponents:ListingModelSelection.DataTemplate>
            <DataTemplate>
                <listingComponentsTemplate:ListingModelSelectionTemplate/>
            </DataTemplate>
        </listingComponents:ListingModelSelection.DataTemplate>

        </listingComponents:ListingModelSelection>
</StackLayout>
```

Рисунок 3.26 – Секція з Описом товару

Нижня секція сторінки містить опис товару (SelectedListingItem.ProductDescription) та кнопку для здійснення покупки (Button із текстом "Shop Now!"). Кнопка має білий текст, контрастний фоновий колір (leonaColor), та параметри розміру (WidthRequest="70" та HeightRequest="60").

Загалом, ця XAML-розмітка створює детальну сторінку товару з інтерактивними елементами, такими як карусель зображень, вибір кольору, ціна, опис та кнопка покупки.

Остання сторінка – це сторінка привітання LandingContentPage) для застосунку LeonaStore, побудовану з використанням Xamarin.Forms. Вона відповідає за відображення інтерактивної стартової сторінки з каруселлю слайдів, можливістю пропуску та кнопкою для продовження роботи із застосунком.

Верхній рівень цієї розмітки представлений елементом ContentPage із автоматичним зв'язуванням ViewModel через prism:ViewModelLocator.AutowireViewModel="True". Це дозволяє автоматично підключити LandingContentPageViewModel без необхідності задавати його вручну у коді code-behind (рис. 3.27). Фон сторінки заданий через BackgroundColor="{StaticResource leonaColor}", що використовує ресурс із палітри застосунку.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"
    xmlns:prismBehaviors="clr-namespace:Prism.Behaviors;assembly=Prism.For
    xmlns:behaviors="clr-namespace:Behaviors;assembly=LeonaStore"
    xmlns:template="clr-namespace:LeonaStore.Views.LandingPage.LandingPage
    xmlns:cv="clr-namespace:Xamarin.Forms;assembly=Xamarin.Forms.CarouselV
    xmlns:dots="clr-namespace:LeonaStore.Components.CarouselDotsGenerator;
    BackgroundColor="{StaticResource leonaColor}"
    prism:ViewModelLocator.AutowireViewModel="True"
    x:Class="LeonaStore.Views.LandingContentPage">

    <RelativeLayout>

        <StackLayout
            RelativeLayout.XConstraint =
      "{ConstraintExpression Type=RelativeToParent,
                             Property=X,
                             Factor=0,
                             Constant=0}"
    RelativeLayout.YConstraint =
      "{ConstraintExpression Type=RelativeToParent,
                             Property=Y,
                             Factor=0,
                             Constant=0}"
        RelativeLayout.WidthConstraint =
      "{ConstraintExpression Type=RelativeToParent,
                             Property=Width,
                             Factor=1,
```

Рисунок 3.27 – Елемен ContentPage та підключення LandingContent

Основний контейнер сторінки — RelativeLayout. Цей макет дозволяє розміщувати елементи за допомогою відносних координат, що надає гнучкість у компонуванні. Всі вкладені елементи (StackLayout, CarouselView тощо) використовують прив'язку через RelativeLayout.XConstraint та YConstraint, яка розтягує їх на весь екран.

У верхній частині сторінки розміщений StackLayout (рис. 3.28), що містить кнопку Label із текстом "Skip". Вона дозволяє користувачеві пропустити слайди привітання та одразу перейти до головної частини застосунку. Кнопка стилізована з використанням напівпрозорого білого

кольору (TextColor="#99FFFFFF") та жирного шрифту (FontAttributes="Bold"). До елемента прив’язано команду SkipLandingPage за допомогою TapGestureRecognizer, яка викликає відповідний метод у ViewModel. Крім того, реалізовано поведінку behaviors:HideViewIf, яка приховує кнопку, якщо властивість ShowSkip у ViewModel встановлена у false.

```
    <StackLayout
        RelativeLayout.XConstraint =
  "{ConstraintExpression Type=RelativeToParent,
                         Property=X,
                         Factor=0,
                         Constant=0}"
RelativeLayout.YConstraint =
  "{ConstraintExpression Type=RelativeToParent,
                         Property=Y,
                         Factor=0,
                         Constant=0}"
    RelativeLayout.WidthConstraint =
  "{ConstraintExpression Type=RelativeToParent,
                         Property=Width,
                         Factor=1,
                         Constant=0}"
RelativeLayout.HeightConstraint =
  "{ConstraintExpression Type=RelativeToParent,
                         Property=Height,
                         Factor=1,
                         Constant=0}"
    Padding="30"
    Orientation="Horizontal">

        <Label
            x:Name="SkipLabel"
            FontAttributes="Bold"
            Text="Skip"
            TextColor="#99FFFFFF"
            FontSize="20"
            HorizontalOptions="EndAndExpand"
            VerticalOptions="StartAndExpand">

            <Label.GestureRecognizers>
                <TapGestureRecognizer Command="{Binding SkipLandingPage}"/>
            </Label.GestureRecognizers>

        <Label.Behaviors>
                <behaviors:HideViewIf HideControlIf="{Binding ShowSkip}"/>
            </Label.Behaviors>
        </Label>

    </StackLayout>
```

Рисунок 3.28 – Елемент StackLayout

Центральний елемент сторінки — це CarouselView, який відображає слайди привітання. Властивість ItemsSource="{Binding PagesModelData}" прив’язана до списку слайдів у ViewModel. Кожен слайд представлений через шаблон DataTemplate, де використовується користувацький компонент LandingPageViewTemplate. Для керування відображенням активного слайду використовується властивість Position="{Binding CarouselPosition}". Крім того, для обробки події зміни сторінки (PositionSelected) використовується поведінка prismBehaviors:EventToCommandBehavior, яка викликає команду PositionChangedCommand із передачею нового індексу слайду.

В нижній частині сторінки розташований ще один StackLayout для управління нижніми елементами керування. Він містить кнопку "Get Started" (ContinueButton) та індикатор поточної сторінки (CarouselDotsGenerator). Кнопка "Get Started" використовується для завершення перегляду слайдів і переходу до основної частини застосунку. Вона має білий текст (TextColor="White") і напівпрозорий фон, який визначається ресурсом likeLeonaColorButALittleBitMoreOpaque. Прив’язка до команди SkipLandingPage забезпечує, що натискання кнопки викличе перехід до головної сторінки застосунку. Для приховування кнопки до моменту, поки користувач не перегорне всі слайди, використовується поведінка behaviors:HideViewIf із прив’язкою до властивості ShowContinueButton.

Індикатор поточної сторінки (dots:CarouselDotsGenerator) розміщений під кнопкою "Get Started". Він відображає кількість слайдів у вигляді точок, де активна точка відповідає поточній сторінці в каруселі. Властивість ItemsSource="{Binding PagesModelData}" синхронізує кількість точок з кількістю слайдів, а CurrentPage="{Binding CarouselPosition}" забезпечує зміну активної точки при перемиканні слайдів.

3.5 Тестування програмного продукту

Для тестування програмного продукту, користувачу потрыбно встановити даний додаток на власний пристрій. Після цього він повинен відкрити додаток.

Спочатку програма відображає стаартовий логотип магазину (рис. 3.29):

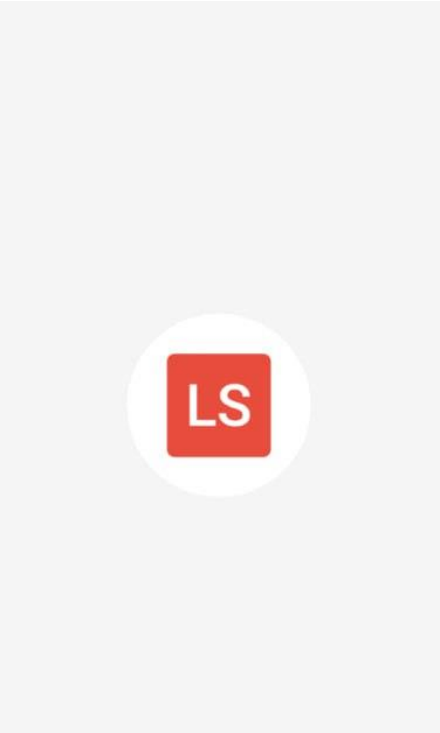

Рисунок 3.29 – Стартовий логотип магазину

Після завантаження сторінки та логотипу з’являється головний екран додатку в якому відображаються доступні товари для користувача (рис. 3.30):

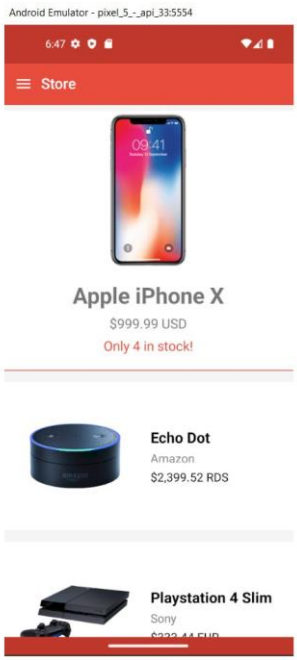

Рисунок 3.30 – Вікно магазину

В користувача є можливість відкрити меню в якому він може переглянути детальну інформацію про додаток, зайти в налаштування додатку обрати певні категорії товарів (рис. 3.31).

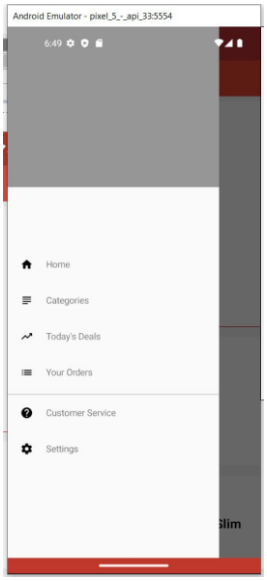

Рисунок 3.31 – Бокове меню додатку

При виборі конкретного товару, перед користувачем відкривається сторінка цього ж товару: його загальний опис, ціна та моделі різних кольорів, та кнопка отримати товар.

Рисунок 3.32 – Сторінка товару

3.6 Висновок до третього розділу

В результаті було розроблено кросплатформений додаток із використанням фреймворку Xamarin.Forms та архітектурного патерну MVVM. Додаток забезпечує зручний користувацький інтерфейс з інтерактивною сторінкою привітання, детальним відображенням товарів, функцією вибору моделей і кольорів, а також системою навігаційного меню. Завдяки використанню XAML для опису інтерфейсу, досягнуто чіткого розділення логіки бізнесу та візуального представлення, що покращує підтримуваність і масштабованість коду. Реалізація команд, прив’язок даних і поведінок забезпечує гнучку взаємодію користувача з додатком, а використання зовнішніх бібліотек, таких як FFImageLoading та Prism, дозволяє оптимізувати роботу із зображеннями та спростити управління навігацією.

ВИСНОВОК

У процесі виконання кваліфікаційної роботи на тему «Розробка крос-платформного додатка "Інтернет-магазин"» було проведено комплексне дослідження предметної області електронної комерції, методів розробки кросплатформних застосунків та сучасних технологій програмної розробки. Робота включала аналіз існуючих підходів до створення мобільних застосунків, дослідження переваг та недоліків фреймворків Xamarin, React Native та Flutter, обґрунтування вибору технології для реалізації проєкту, а також розробку програмної реалізації та її тестування.

Основними завданнями, які були вирішені під час виконання роботи, стали:

1. Проведення аналізу моделей електронної комерції, включаючи B2B, B2C, C2C та C2B.
2. Огляд сучасних фреймворків для кросплатформної розробки, таких як Xamarin, React Native та Flutter.
3. Обґрунтування вибору Xamarin як основного фреймворку для реалізації інтернет-магазину через його інтеграцію з .NET та підтримку мови C#.
4. Розробка кросплатформного застосунку за архітектурним патерном MVVM, що забезпечило розділення бізнес-логіки, даних та інтерфейсу користувача.
5. Тестування розробленого програмного продукту для перевірки відповідності функціональним та нефункціональним вимогам, таким як стабільність роботи, продуктивність та зручність використання.

У ході роботи було обґрунтовано вибір платформи Xamarin для реалізації проєкту завдяки можливості повторного використання до 96% коду між платформами Android, iOS та Windows, а також через зручну інтеграцію з середовищем Visual Studio.

У процесі роботи були сформовані вимоги до програмного забезпечення:

- Функціональні вимоги: каталогізація товарів, управління замовленнями, пошук та фільтрація товарів.
- Нефункціональні вимоги: кросплатформність, продуктивність, стабільність роботи, підтримка архітектури MVVM.

Розроблений інтернет-магазин було успішно протестовано, що підтвердило його відповідність вимогам до функціональності, продуктивності та зручності використання. Застосунок може бути використаний для подальшого вдосконалення, зокрема через додавання інтеграції з хмарними сервісами, функцій аналітики та розширення функціональності.

Таким чином, поставлена мета щодо розробки кросплатформного інтернет-магазину була досягнута, а отримані результати можуть бути використані для подальших досліджень у сфері електронної комерції та розробки програмного забезпечення.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Пилипенко В.О., Петросян Р.В. Огляд кросплатформенних фреймворків для розробки мобільного додатку [Електронний ресурс] – Режим доступу: https://conf.ztu.edu.ua/wp-content/uploads/2022/04/37.pdf

2. Мадрі Д. Weebly. Website planet. – [Електронний ресурс] – Режим доступу: https://www.websiteplanet.com/uk/website-builders/weebly/

3. Hermes D. Xamarin Mobile Application Development: Cross-Platform C# and Xamarin.Forms Fundamentals / Dan Hermes., 2015. – 432 с

4. Turban E., Outland J., King D., Lee J.K., Liang T.-P. Electronic Commerce 2018: A Managerial and Social Networks Perspective. 2018

5. Cross platform app frameworks in 2021. URL: https://www.netsolutions.com/insights/cross-platform-appframeworks-in-2021

6. Документація ReactNative. URL: https://reactnative.dev

7. Документація Flutter. URL: https://flutter.dev

8. Документація Xamarin. URL: https://docs.microsoft.com/en-us/xamarin/

9. Why you should (or shouldn’t) use React Native. URL: https://www.conceptatech.com/blog/why-you-should-orshouldnt-use-react-native

10. What is Flutter. URL: https://lanars.com/blog/what-is-flutter

11. Why use Xamarin? URL: https://www.sam-solutions.com/blog/xamarin-cross-platform-development/

12. Гібридні мобільні додатки та їх переваги. URL: https://web4u.in.ua/blog/g-bridn-mob-l-n-dodatki-ta-hperevagi-18

13. PWA, або Прогресивні веб-додатки. URL: https://smile-ukraine.com/ua/pwa/introduction

14. C# Programming Language. Geekforgeeks. – [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/csharp-programming-language/

15. Introduction to MVVM Architecture – [Електронний ресурс] – Режим доступу: https://medium.com/@onurcem.isik/introduction-to-mvvm-architecture-5c5558c3679