Міністерство освіти і науки України
Університет митної справи та фінансів

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

Кваліфікаційна робота бакалавра
на тему: «Розробка крос-платформенного додатку для замовлення кавових товарів»

Виконав: студент групи ІПЗ20-1
Спеціальність 121 «Інженерія програмного забезпечення»
Мізера Богдан Андрійович
(прізвище та ініціали)
Керівник к.т.н., доц. Ульяновська Ю.В.
(науковий ступінь, вчене звання, прізвище та ініціали)
Рецензент Університет митної справи та фінансів
(місце роботи)
Доцент кафедри кібербезпеки та інформаційних технологій
(посада)
к.т.н., доцент Флоров С.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2024

АНОТАЦІЯ

Мізера Б.А. Розробка крос-платформного додатку для замовлення кавових товарів.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 121 «Інженерія програмного забезпечення» – Університет митної справи та фінансів, Дніпро, 2024.

Дана кваліфікаційна робота присвячена розробці крос-платформного додатку для замовлення кофейних товарів. Розробка додатку проводилась з використанням сучасних технологій програмування, таких як C# та Xamarin. Створений додаток забезпечує автоматизацію процесу замовлення кофейних товарів, надаючи користувачам можливість зручно та ефективно здійснювати замовлення, переглядати асортимент, отримувати інформацію про нові пропозиції та персоналізувати свої замовлення.

У роботі проведено детальний аналіз предметної області, включаючи дослідження сучасних методів та підходів до розробки крос-платформних додатків. Було обрано платформу Xamarin, яка дозволяє створювати додатки з єдиною кодовою базою для різних операційних систем, забезпечуючи високу продуктивність та нативний вигляд додатків.

Розроблене програмне забезпечення включає використання архітектури MVVM, алгоритмів обробки даних та інтеграцію з API кофейних товарів. Додаток забезпечує зручний спосіб перегляду та замовлення кофейних продуктів, отримання інформації про нові пропозиції, а також підвищує ефективність бізнес-процесів через автоматизацію та інтеграцію з системами управління запасами та доставки.

Ключові слова: нативний додаток, крос-платформний додаток, C#, Xamarin, MVVM.

ABSTRACT

Mizera B.A. Development of a Cross-Platform Application for Coffee Product Orders.

Bachelor's thesis for obtaining a degree in Software Engineering, specialty 121. – University of Customs and Finance, Dnipro, 2024.

This bachelor's qualification work is dedicated to the development of a cross-platform application for ordering coffee products. The application was developed using modern programming technologies, such as C# and Xamarin. The created application automates the process of ordering coffee products, providing users with the ability to conveniently and efficiently place orders, browse the assortment, receive information about new offers, and personalize their orders.

The work includes a detailed analysis of the subject area, including the study of modern methods and approaches to the development of cross-platform applications. The Xamarin platform was chosen, allowing the creation of applications with a single codebase for different operating systems, ensuring high performance and a native look and feel.

The developed software includes the use of the MVVM architecture, data processing algorithms, and integration with the coffee product API. The application provides a convenient way to browse and order coffee products, receive information about new offers, and increases business process efficiency through automation and integration with inventory management and delivery systems.

Keywords: native application, cross-platform application, C#, Xamarin, MVVM.

ЗМІСТ

ВСТУП

Актуальність проблеми. Розробка крос-платформного додатку для замовлення кофейних товарів відповідає сучасним вимогам ринку та тенденціям цифровізації бізнесу. У контексті зростаючої популярності мобільних технологій та електронної комерції, споживачі прагнуть до максимальної зручності та швидкості обслуговування, що може бути забезпечено за допомогою інноваційних цифрових рішень. Впровадження крос-платформного додатку дозволить користувачам ефективно замовляти кофейні товари з будь-якого місця і в будь-який час, що значно підвищує лояльність клієнтів та забезпечує більшу конкурентоспроможність на ринку.

Автоматизація процесів замовлення, оплати та доставки через додаток не тільки оптимізує робочі процеси, зменшуючи витрати на обслуговування та адміністрування, але й забезпечує більш високу продуктивність персоналу. Це дозволяє підприємствам швидше реагувати на зміни попиту та пропонувати кращий сервіс своїм клієнтам.

Крім того, крос-платформні додатки відкривають можливості для аналізу великих обсягів даних про поведінку споживачів та їхні переваги. Це дозволяє компаніям краще розуміти потреби своїх клієнтів, адаптувати та персоналізувати свої пропозиції, що, в свою чергу, сприяє збільшенню продажів та покращенню маркетингових стратегій.

Одночасна підтримка декількох платформ забезпечує доступність сервісу для широкого спектра користувачів, незалежно від типу їхніх пристроїв, що є ключовим у підвищенні охоплення ринку та задоволення потреб сучасних споживачів. Таким чином, розробка крос-платформного додатку для замовлення кофейних товарів є стратегічно важливим кроком, який відповідає як поточним, так і майбутнім потребам бізнесу в епоху цифровізації.

Метою роботи є автоматизація роботи продажу кофейних товарів.

Методи дослідження: обробка та аналіз інформації, методи проектування та розробки крос-платформних додатків.

У відповідності до поставленої мети в кваліфікаційній роботі поставлені наступні завдання дослідження:

1. Проаналізувати технічні засоби, що застосовуються для розробки крос-платформних додатків.

2. Розробити архітектуру та функціональні можливості крос-платформного додатку.

3. Розробити крос-платформний додаток з застосуванням технології .Net Xamarin.

4. Провести тестування.

Об'єктом дослідження є розробка програмного забезпечення в сфері надання торгівельних послуг.

Предметом дослідження є апаратно-програмне забезпечення для розробки крос-платформного додатку.

Структура роботи:

- Розділ 1 Аналіз предметної області.
- Розділ 2 Аналіз засобів реалізації крос-платформних додатку для замовлення кофейних товарів.
- Розділ 3 Розробка крос-платформенного додатку для замовлення кавових товарів.

Робота складається зі вступу, 3-х розділів, висновків, списку використаних джерел з 15 найменувань. Обсяг роботи 52 сторінок кваліфікаційної роботи, 51 рисунок.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз публікацій щодо розробки крос-платформних додатків

Крос-платформні програми набули значної популярності в сучасному світі завдяки своїй здатності працювати на кількох операційних системах одночасно, що дозволяє заощадити час і кошти на розробку. Хоча такі програми можуть бути менш гнучкими, ніж нативні, вони надають можливість швидко запускати продукти з широким охопленням аудиторії та помірними витратами. Вибір між крос-платформовою та нативною розробкою залежить від специфічних вимог бізнесу, функціональних можливостей додатка та доступного бюджету, що робить обидва підходи важливими інструментами у цифровізації бізнесу [1].

Крос-платформні програми — це додатки, які можуть працювати на кількох операційних системах одночасно. Завдання програмістів полягає у створенні коду, який функціонує на всіх цих системах.

Цей підхід до розробки дозволяє економити час і гроші. Розробка додатка проходить швидше, оскільки потрібно написати код лише один раз для всіх платформ. У випадку, якщо бізнес замовляє додаток для різних систем окремо, витрати часу і грошей на розробку зростають удвічі.

Проте, крос-платформні додатки мають свої недоліки. Вони менш гнучкі порівняно з нативними додатками, адже важко реалізувати всі функції так, щоб вони добре працювали на різних пристроях. Крім того, кожен магазин додатків має свої вимоги, які потрібно враховувати під час розробки, що може створювати додаткові складнощі [2].

Незважаючи на ці недоліки, крос-платформні додатки залишаються дуже популярними та ефективними. В залежності від бізнес-сфери, можна створити унікальні інструменти, які будуть зручні для користувачів. Швидкий запуск, широке охоплення аудиторії та порівняно невисока вартість розробки

дозволяють швидко реалізувати ідеї, запустити потужні інструменти та цифровізувати бізнес.

Нативні програми створюються для конкретної операційної системи, використовуючи відповідний технологічний стек.

Переваги нативних додатків включають високу продуктивність, максимальне використання можливостей платформи та відмінний користувацький інтерфейс. Такі програми мають кращі шанси на просування в магазинах додатків.

Недоліки нативних додатків включають тривалий час розробки та високу вартість. Крім того, нативні додатки не підтримуються на інших операційних системах, тому для кожної платформи потрібно розробляти окрему версію додатка, що також потребує додаткових витрат часу та коштів [3].

Отже, ключова різниця між нативною та крос-платформовою розробкою додатків полягає у підході до розробки та підтримки додатків. Вибір між нативною та крос-платформовою розробкою залежить від функціональних вимог програми, цільової аудиторії та бюджету. Якщо важливо бути присутнім на двох платформах одночасно, і є обмеження у часі та бюджеті, крос-платформне рішення буде кращим. Якщо ж є час і бюджет, і додаток потребує високої продуктивності та складного функціоналу, варто обрати нативну розробку.

1.2. Аналіз методів та підходів розробки крос-платформних додатків

Для успішного запуску якісної крос-платформної мобільної рамки необхідно дотримуватись декількох ключових кроків. Спочатку визначте цілі, яких має досягнути мобільний додаток. Далі детально опишіть функціональність та функції додатка. Після цього дослідіть продукти своїх конкурентів, щоб зрозуміти їх сильні та слабкі сторони. Наступним кроком створіть прототип або каркас додатка, щоб мати базове уявлення про його

структуру. Обов'язково протестуйте цей прототип, щоб виявити можливі проблеми на ранній стадії. Далі оберіть шлях розвитку, тобто складіть план на майбутнє. Після цього створіть власне мобільний додаток, протестуйте його і, нарешті, запустіть його в експлуатацію.

Крос-платформні додатки значно сприяють бізнесу, оскільки вони можуть збільшити дохід завдяки покращенню продажів, підвищити рівень поінформованості про бренд і стати ефективним інструментом для аналізу задоволеності клієнтів. Саме тому з 2020 року розробка крос-платформних мобільних рішень стала стрімко рости. Головна перевага таких рішень полягає у тому, що додаток розробляється один раз, але працює на різних платформах, таких як Android, iOS чи Windows.

Розробка крос-платформних додатків передбачає використання різних методів і підходів, що дозволяють створювати програми, які можуть працювати на кількох операційних системах одночасно. Ось основні методи розробки крос-платформних додатків:

1) Гібридні додатки (Hybrid Apps)

Гібридні додатки поєднують у собі елементи нативних і веб-додатків. Вони розробляються з використанням веб-технологій, таких як HTML5, CSS та JavaScript, але запускаються в контейнері, що забезпечує доступ до нативних можливостей платформи. Основні інструменти для розробки гібридних додатків:

- Apache Cordova (PhoneGap)
- Ionic
- Framework7 [11]

2) Нативні додатки з крос-платформними фреймворками

Цей підхід передбачає використання фреймворків, які дозволяють писати нативний код для різних платформ з одного коду. Це забезпечує високу продуктивність та доступ до нативних API. Основні інструменти:

- React Native
- Flutter

- Xamarin
- NativeScript

3) Прогресивні веб-додатки (Progressive Web Apps, PWA)

PWA - це веб-додатки, які використовують сучасні веб-технології для забезпечення функціональності, схожої на нативні додатки. Вони працюють у браузері, але можуть бути встановлені як додатки на мобільних пристроях і використовувати деякі нативні можливості, такі як push-сповіщення та робота в офлайн-режимі [12].

4) Інтерпретовані додатки

Цей метод передбачає використання інтерпретаторів, які виконують код додатка безпосередньо на пристрої користувача. Такі додатки можуть працювати на різних платформах без необхідності перекомпіляції. Основні інструменти:

- JavaScript
- Python
- RubyMotion

5) Розробка на основі єдиного коду (Single Codebase Development)

Цей метод використовує фреймворки, що дозволяють створювати додатки для різних платформ з використанням єдиного коду. Це знижує витрати на розробку та підтримку, оскільки весь код написаний один раз і може використовуватися на всіх платформах. Основні інструменти:

- Kotlin Multiplatform
- Haxe
- Unity

1.3. Висновок до першого розділу

У першому розділі було проведено всебічний аналіз предметної області, зокрема дослідження сучасних методів та підходів до розробки кросплатформних додатків. Здійснено детальний розгляд існуючих рішень, що

дозволило визначити ключові тенденції та технології, які використовуються для створення таких додатків.

Розглянуто основні методи розробки крос-платформних додатків, включаючи гібридні додатки, нативні додатки з крос-платформними фреймворками, прогресивні веб-додатки (PWA) та інтерпретовані додатки. Кожен з цих підходів має свої переваги та недоліки, які детально проаналізовано.

Основною перевагою крос-платформних додатків є можливість одночасної роботи на декількох операційних системах, що дозволяє значно знизити витрати на розробку та забезпечити швидкий запуск продукту з широким охопленням аудиторії. Проте, такі додатки можуть бути менш гнучкими порівняно з нативними, що може ускладнити реалізацію специфічних функцій на різних пристроях.

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ КРОС-ПЛАТФОРМНИХ ДОДАТКУ ДЛЯ ЗАМОВЛЕННЯ КОФЕЙНИХ ТОВАРІВ

2.1. Вибір програмних засобів для реалізації проекту

Вибір правильних програмних засобів для реалізації проекту є ключовим етапом у розробці кросплатформних додатків, оскільки від цього залежить успіх і ефективність кінцевого продукту. Сучасні інструменти пропонують різноманітні можливості та функціонал, що дозволяють розробникам створювати додатки, які працюють на різних платформах, забезпечуючи при цьому високу продуктивність та відмінний користувацький досвід. Розглянемо основні фреймворки та інструменти, які допоможуть у створенні якісних кросплатформних додатків.

1) React Native використовує JavaScript як основну мову програмування для створення багатоплатформних додатків. Однією з основних переваг цього фреймворку є можливість написання модулів на Java, C та Swift. Крім того, React Native дозволяє редагувати зображення та відео, що робить його унікальним серед інших фреймворків. Цей інструмент використовували такі великі компанії як Instagram та Facebook [7, 8].

2) Flutter – це SDK з відкритим вихідним кодом, розроблений компанією Google, що дозволяє створювати додатки для різних платформ з однієї бази коду. Він базується на мові програмування Dart, яку легко освоїти розробникам, знайомим з JavaScript та Java. Google широко використовує Flutter для своїх нових проектів [6].

3) Adobe PhoneGap є простим у використанні фреймворком для створення мобільних додатків на основі HTML5, CSS та JavaScript. Основною перевагою PhoneGap є доступ до хмарних рішень і численних плагінів, що розширюють функціональність додатків.

4) NativeScript – це інструмент з відкритим вихідним кодом, призначений для створення кросплатформних додатків на iOS та Android. Він

підтримує JavaScript та інші мови, що перекладаються на JavaScript, такі як TypeScript.

5) Xamarin, розроблений Microsoft, дозволяє створювати додатки для більшості платформ, використовуючи єдиний код на .NET. Додатки, створені за допомогою Xamarin, виглядають як рідні, що забезпечує найкращий користувацький досвід. Xamarin також пропонує інструменти для створення, розгортання та налагодження додатків [7].

6) Sencha Touch дозволяє створювати додатки, що працюють як у браузерах, так і на мобільних пристроях. Цей інструмент інтегрується з ext JS, що дозволяє створювати додатки, які можуть обробляти великі обсяги даних.

7) Appcelerator Titanium – це інструмент для створення кросплатформних додатків для Android, iOS та Windows. Він надає багато корисних інструментів для веб-технологій, що допомагають скоротити час розробки додатків.

Вибір залежить від конкретних потреб проекту, але кожен з цих інструментів може значно спростити процес розробки та забезпечити відмінний користувацький досвід.

Для реалізації проекту було обрано Xamarin, C#, Visual Studio Community 2022.

2.2. Програмні засоби для розробки крос-платформного додатку

Xamarin – це платформа з відкритим вихідним кодом, призначена для розробки сучасних додатків для iOS, Android і Windows за допомогою .NET. Основною інновацією, яку Xamarin привнесла у світ мобільної розробки, є забезпечення кросплатформенності коду. Платформа Xamarin надає крос-платформенну підтримку для трьох основних платформ: iOS, Android та Windows. Додатки можуть бути написані з можливістю спільного використання до 80% їх коду, а бібліотека Xamarin.Essentials пропонує

універсальний API для доступу до загальних ресурсів на всіх трьох платформах. Це може значно знизити витрати на розробку та скоротити час виходу на ринок для мобільних розробників, орієнтованих на ці популярні платформи.

Функціонально платформа Xamarin складається з декількох субплатформ:

- Xamarin.Android - бібліотеки для створення додатків для ОС Android;
- Xamarin.Mac - бібліотеки для створення додатків для Mac OS X;
- Xamarin.iOS - бібліотеки для створення додатків для iOS.

Ці субплатформи відіграють важливу роль, оскільки через них додатки можуть направляти запити до прикладних інтерфейсів на пристроях під управлінням ОС Android або iOS. Завдяки цим субплатформам можна створювати окремі додатки для Android та iOS, але найважливішою особливістю платформи є можливість створювати кросплатформені додатки, що використовують одну логіку для всіх платформ. Можна визначити візуальний інтерфейс, один раз прив'язати до нього певну логіку на C#, і все це буде працювати на Android і iOS [13].

Платформа Xamarin дозволяє розробникам створювати на мові C# мобільні додатки для операційних систем iOS та Android. Вона базується на сторонній відкритій реалізації платформи .NET під назвою Mono. Рівень бізнес-логіки може бути написаний один раз і використовуватись на всіх мобільних платформах. Взаємодія з інтерфейсом користувача (UI) та API відрізняється на різних мобільних платформах, тому рівень користувацького досвіду зазвичай унікальний для кожної платформи [10].

Інструментарій Xamarin.Forms розширює можливості Xamarin, спрощуючи крос-платформену мобільну розробку шляхом написання спільного коду для забезпечення необхідного користувацького інтерфейсу, а також для реалізації бізнес-логіки. Для розробки бізнес-логіки додатків використовується мова C#. Бізнес-логіка може включати обмін даними з сервером, роботу з внутрішньою базою даних, конвертацію даних тощо.

Як і в додатках для універсальної платформи Windows (UWP), у Xamarin.Forms використовується мова розмітки XAML для одноразового визначення користувацького інтерфейсу для всіх платформ через абстрагування компонентів інтерфейсу, специфічних для тієї чи іншої платформи.

XAML включає основні чотири категорії елементів: панелі, елементи управління, елементи пов'язані з документом та графічні об’єкти. Все створене або реалізоване за допомогою XAML може бути виражене через C#, однак ключовим аспектом технології є зменшення складності інструментів, що використовуються для обробки XAML.

Додатки, створені на основі Xamarin.Forms, формують UI з нативних віджетів платформи, тому є зручними та привабливими, а також ідеально підходять для різних платформ. XAML-файли можна також створювати і редагувати за допомогою редактора і інструментів візуального конструювання, наприклад, Microsoft Visual Studio. Процес розробки додатків схожий на розробку на основі Windows.Forms і UWP.

2.3. Патерни проектування

Проєктні патерни відіграють вирішальну роль у розробці веб-додатків, сприяючи структурованості, зрозумілості та масштабованості коду. Вони дозволяють розробникам застосовувати перевірені рішення для типових проблем, що виникають під час створення програмного забезпечення. Серед найпоширеніших патернів у веб-розробці можна виділити Model-View-Controller (MVC) та Model-View-ViewModel (MVVM). Кожен з них має свої переваги, недоліки та специфічну сферу застосування, що робить їх незамінними інструментами для сучасних розробників. Розглянемо їх детальніше [14].

MVVM (Model-View-ViewModel)

MVVM - це патерн проектування, призначений для створення структурованих та масштабованих додатків, особливо в контексті сучасних фреймворків та бібліотек. Основна мета MVVM полягає в розділенні представлення інтерфейсу користувача та бізнес-логіки, що спрощує розробку, тестування та підтримку додатка.

1. Model - відповідає за управління даними додатка, взаємодіючи з базою даних і виконуючи бізнес-логіку.

2. View - відповідає за відображення даних користувачу через інтерфейс (HTML, XAML тощо) не містить бізнес-логіки.

3. ViewModel - виступає як посередник між View та Model. Він отримує дані з Model, обробляє їх і надає View у зручному для відображення форматі. ViewModel також обробляє події від View і викликає відповідні методи Model. Основною перевагою ViewModel є можливість двосторонньої прив'язки даних, що автоматично оновлює View при зміні даних у ViewModel і навпаки.

Переваги MVVM:

1. Розділення обов'язків

MVVM забезпечує чітке розділення обов'язків між даними, логікою та інтерфейсом користувача, що полегшує розробку та підтримку додатка.

2. Модульність

Кожен компонент (Model, View, ViewModel) може бути розроблений, протестований і підтримуваний незалежно, що сприяє більшій масштабованості та зрозумілості коду.

3. Двостороння прив'язка даних

Однією з основних переваг MVVM є можливість двосторонньої прив'язки даних між View і ViewModel, що зменшує необхідність вручну оновлювати інтерфейс при зміні даних.

4. Повторне використання коду

ViewModel і Model можуть бути повторно використані у різних частинах додатка або навіть у різних проектах, що підвищує ефективність розробки.

2.3. Висновок до другого розділу

У другому розділі було проведено аналіз засобів реалізації крос-платформних додатків для замовлення кавових товарів. Розглянуто основні програмні засоби та інструменти, які можуть бути використані для розробки таких додатків, а також детально описано процес вибору оптимальних технологій для реалізації проекту.

Розглянуто найпопулярніші фреймворки для створення крос-платформних додатків, такі як React Native, Flutter, Adobe PhoneGap, NativeScript, Xamarin та інші. Кожен з цих інструментів має свої переваги та особливості, які можуть впливати на ефективність розробки, продуктивність додатків та зручність користувацького інтерфейсу.

Вибір програмних засобів для реалізації проекту був здійснений на основі таких критеріїв, як продуктивність роботи, наявність відкритого вихідного коду, спосіб побудови інтерфейсу користувача, зрілість фреймворку, наявність великої спільноти користувачів, документація та підтримувані мови програмування. За результатами порівняння було обрано платформу Xamarin, яка дозволяє створювати крос-платформні додатки з єдиною кодовою базою, забезпечуючи при цьому високу продуктивність та нативний вигляд додатків.

Xamarin пропонує широкі можливості для розробки, включаючи підтримку C# та .NET, що дозволяє використовувати один і той же код для створення додатків для iOS, Android та Windows. Крім того, інструментарій Xamarin.Forms спрощує процес створення інтерфейсу користувача, забезпечуючи одноразове визначення UI для всіх платформ. Використання MVVM (Model-View-ViewModel) патерну у Xamarin.Forms значно спрощує розробку та тестування додатків, забезпечуючи чітке розділення між логікою представлення, бізнес-логікою та даними.

РОЗДІЛ 3. РОЗРОБКА КРОС-ПЛАТФОРМЕННОГО ДОДАТКУ ДЛЯ ЗАМОВЛЕННЯ КАВОВИХ ТОВАРІВ

3.1. Концепція створення крос-платформенного додатку

Розробка кросплатформенного додатку для замовлення кофейних товарів є актуальною через зростаючу потребу сучасних споживачів у зручних та ефективних способах взаємодії з постачальниками товарів та послуг. Такий додаток забезпечує доступність замовлень у будь-який час і з будь-якого місця, підтримує різні операційні системи, покращує обслуговування клієнтів завдяки інтуїтивно зрозумілому інтерфейсу та персоналізації замовлень, підвищує ефективність бізнес-процесів через автоматизацію та інтеграцію з системами управління запасами та доставки, надає можливість аналізу даних і розробки маркетингових стратегій, стимулює повторні покупки через пуш-повідомлення та спеціальні пропозиції, і забезпечує конкурентні переваги на ринку, сприяючи формуванню позитивного іміджу різноманітних брендів.

Крім цього, даний додаток може стати своєрідним маркетплейсом, що дозволить будь-якій людині отримати можливість вести власний бізнес завдяки продажу кавових виробів. Користувач також може отримувати повідомлення про знижки в додатку, що може стати стимулом для купівлі товару. Такий додаток значно розширить можливості самих продавців, які як правило рекламують власну продукції через інші популярні додатки як instagram, які мають змішану аудиторію, і тому важко спрогнозувати, який буде попит на дану продукцію, враховуючи те яким чином пощастить продавцю потрапити в рекомендації саме тієї аудиторії яку потребує виробник або продавець товарів кавових виробів. Таким чином даний додаток дозволить створити власне коло інтересів навколо конкретного виду товарів.

3.2. Структура проекту

Технологія XAMARIN надає широкий вибір, щодо проектування майбутньої архітектури проекту. Для побудови архітектури потрібно визначити яким чином і на скільки складною буде бізнес-логіка даного додатку та його масштаб. Одним з популярних патернів проектування додатку є архітектура MVVM (MVVM (Model-View-ViewModel). Даний патерн відносно простіше реалізовується ніж інші архітектури як: VIPER, MVC та інші.

Загальна структура проекту має наступний вигляд (рис. 3.1):

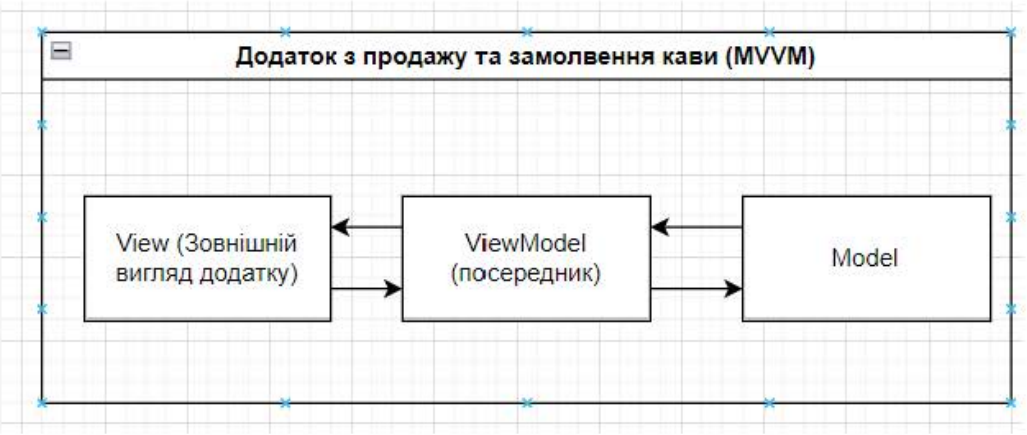


Рисунок 3.1 – Загальна структура проекту

Патерн MVVM (Model-View-ViewModel) є архітектурним підходом, який використовується для розробки програмного забезпечення, що розділяє логіку представлення, бізнес-логіку та дані. Model відповідає за управління даними та бізнес-логікою програми, забезпечуючи доступ до даних та їх обробку. View відповідає за відображення користувацького інтерфейсу та взаємодію з користувачем. ViewModel виступає посередником між Model і View, забезпечуючи зв'язок між ними та перетворення даних Model у форму, яка підходить для відображення у View.

Так як окрім однієї операційної платформи, XAMARIN надає можливість побудувати додаток одночасно на декілька платформ та запускати для відлагодження помилок під час пильнування (рис. 3.2).

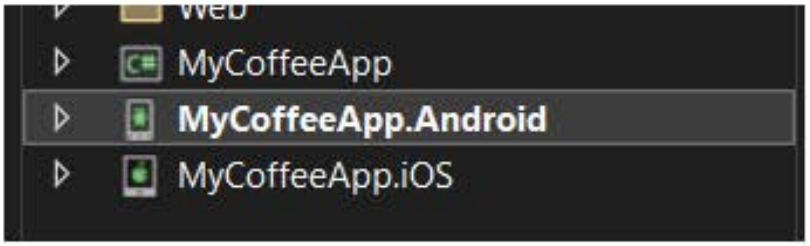


Рисунок 3.2 – Сформовані проекти для різних платформ

3.3. Технології розробки кросплатформенного додатку

Технологія Xamarin.Forms - дозволяє розробникам створювати інтерфейси користувача для всіх трьох платформ з використанням спільного коду. Це значно скорочує час та зусилля, необхідні для розробки та підтримки додатків на різних платформах. Xamarin.iOS та Xamarin.Android забезпечують доступ до нативних API платформ, дозволяючи створювати додатки з повною функціональністю і однаковим виглядом. Використання MVVM (Model-View-ViewModel) патерну в Xamarin.Forms значно спрощує процес розробки та тестування додатків. Зв'язок між View та ViewModel здійснюється через механізм прив'язки даних (data binding), що дозволяє автоматично оновлювати інтерфейс користувача при зміні даних у ViewModel.

Крім того, Xamarin включає в себе потужний інструмент для тестування — Android SDK, який дозволяє автоматизувати тестування мобільних додатків на різних пристроях і платформах. Це допомагає забезпечити високу якість додатків та їх стабільну роботу.

Рисунок 3.3 – Технологія Xamarin

Xamarin автоматично підтримує технологію Android SDK (Software Development Kit). Це набір інструментів і бібліотек, необхідних для розробки додатків для операційної системи Android. Android SDK надає розробникам всі необхідні ресурси для створення, тестування, налагодження та розгортання додатків, включаючи Android Studio — офіційне інтегроване середовище розробки (IDE) від Google.

Ціним дана технологія тим, що вона надає можливість створювати віртуальні середовища для тестування мобільних додатків на різні пристрої та з різною роздільною здатністю (рис. 3.4).

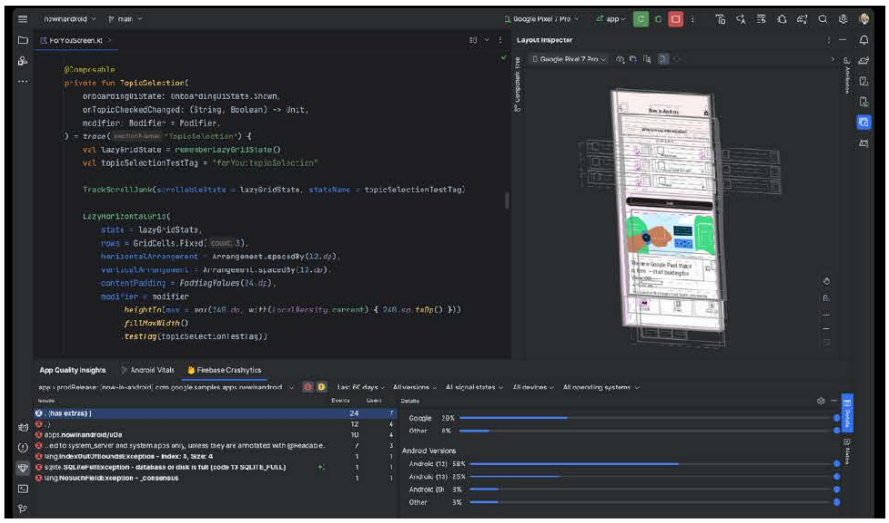


Рисунок 3.4 – Розробка за допомогою Android SDK

Крім того які технології та бібліотеки використовуються для розробки того чи іншого проекту. Важливим є визначення середовище розробки, яке б дозволяло використовувати потрібні бібліотеки та технології. На ринку є

багато так званих IDE. Але, якщо розробнику потрібно розробляти додаток використовуючи мову програмування C#, то найпопулярнішим рішення є саме Visual Studio. Visual Studio — це інтегроване середовище розробки (IDE) яке використовується для створення програмного забезпечення на різних платформах. Воно підтримує велику кількість мов програмування, включаючи C#, C++, Python, JavaScript, TypeScript, і багато інших, і забезпечує потужний набір інструментів для розробки, тестування та налагодження додатків. Visual Studio є популярним серед розробників завдяки своїй функціональності, продуктивності та зручності використання (рис. 3.5).

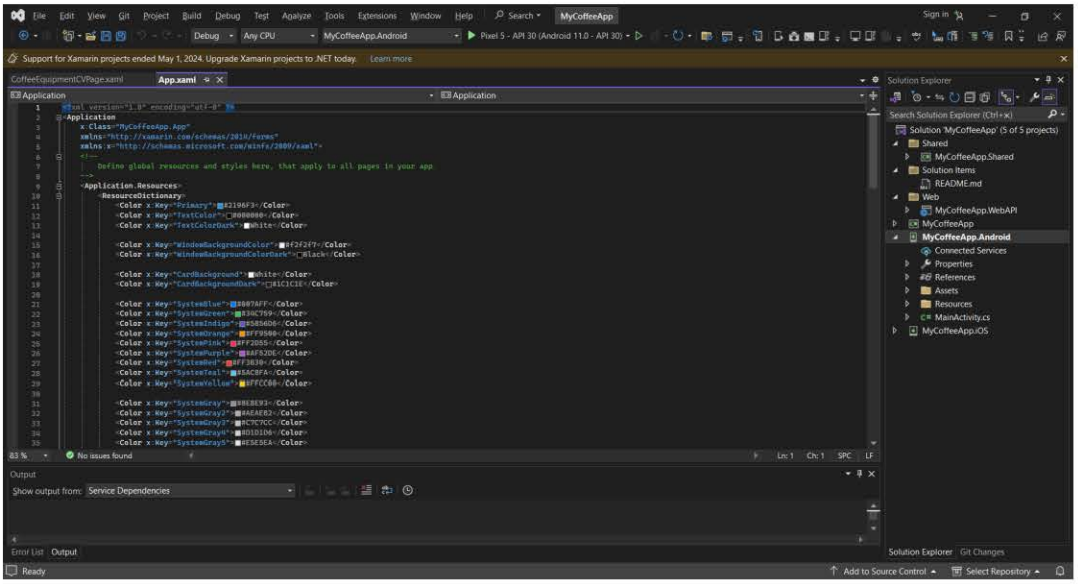


Рисунок 3.5 – Середовище розробки Visual Studio

Вбудовані інструменти для DevOps, такі як Azure DevOps, дозволяють автоматизувати збірку, тестування та розгортання додатків в хмарному середовищі.

Для створення візуального інтерфейсу Microsoft надає мову розмітки XAML. XAML (Extensible Application Markup Language) — це мова розмітки, яка використовується для створення інтерфейсів користувача в технологіях, таких як WPF (Windows Presentation Foundation), UWP (Universal Windows Platform), Xamarin.Forms та .NET MAUI. XAML дозволяє розробникам

описувати елементи інтерфейсу користувача та їхню логіку аналогічно, як це робить html (рис. 3.6):

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:viewmodels="clr-namespace:MyCoffeeApp.ViewModels"
             x:Class="MyCoffeeApp.Views.AnimationPage"
             x:DataType="viewmodels:ImageCacheViewModel"
             Title="Animations"
             xmlns:fontawesome="clr-namespace:FontAwesome"
             xmlns:helpers="clr-namespace:MyCoffeeApp.Helpers"
    BackgroundColor="{AppThemeBinding
                      Dark={StaticResource WindowBackgroundColorDark},
                      Light={StaticResource WindowBackgroundColor}}">
    <ContentPage.BindingContext>
        <viewmodels:ImageCacheViewModel />
    </ContentPage.BindingContext>

    <StackLayout>
        <Button Text="Animate"
                IsEnabled="{Binding IsNotBusy}"
                Command="{Binding RefreshLongCommand}"/>

        <Label Text="{x:Static fontawesome:FontAwesomeIcons.Spinner}"
               FontFamily="FAS"
               x:Name="LabelLoad"
               IsVisible="{Binding IsBusy}"
               FontSize="80"
               Style="{StaticResource LabelLarge}"
               HorizontalOptions="Center"/>
    </StackLayout>

</ContentPage>
```

Рисунок 3.6 – Приклад XAML-розмітки

3.4. Розробка кросплатформенного додатку

Спочатку потрібно визначити загальну структуру проекту, та побудувати діаграму класів, для повного розуміння всіх функціональних компонентів додатку (рис. 3.7) [15].

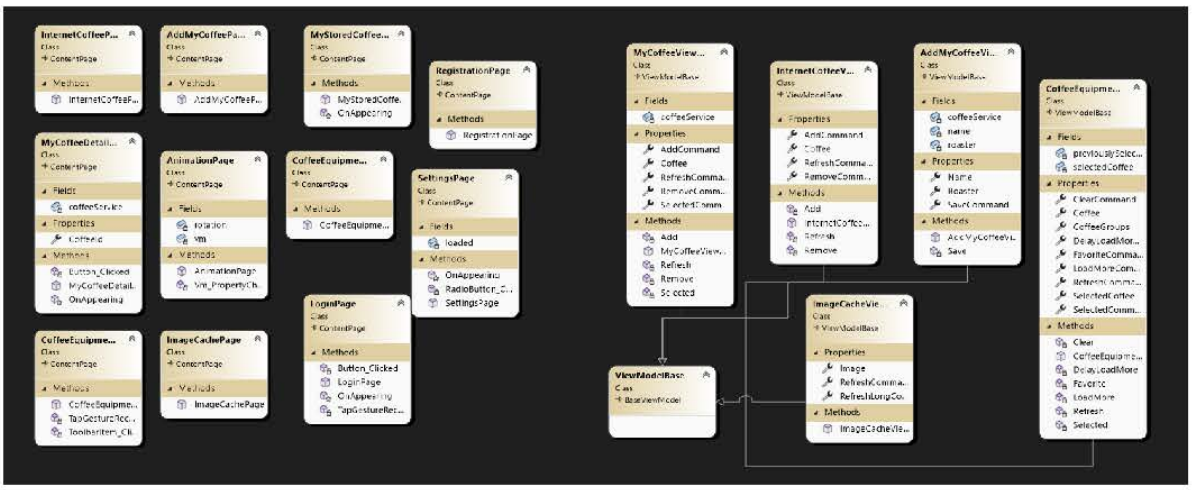


Рисунок 3.7 – Діаграма класів проекту

Звернувши увагу на діаграму, вона візуально поділена на 2 частини. Перша частина представляє модель ViewModel:

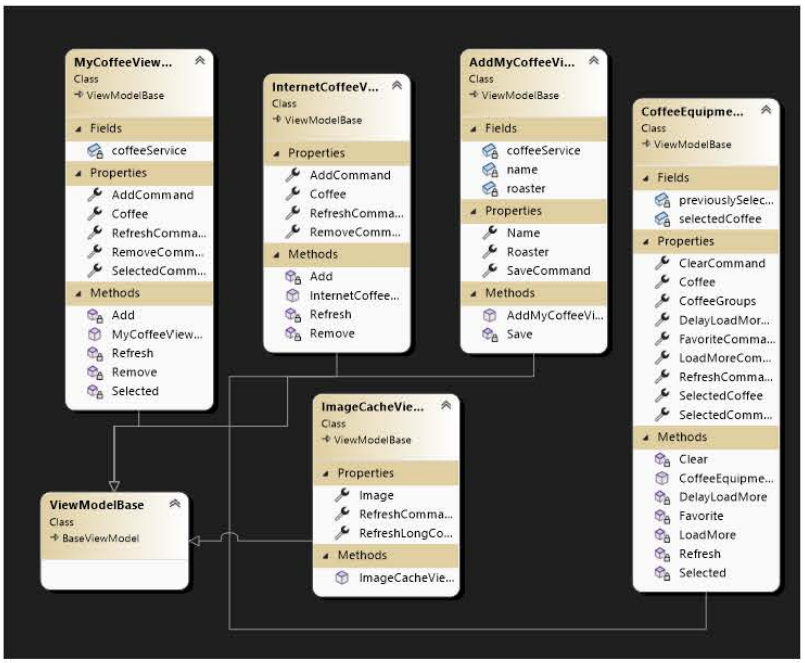


Рисунок 3.8 – Класи, що відносяться до моделі ViewModel

Інша частина діаграми присвячена класам, що відносяться до представлень, та частіше використовуються для обробки подій.

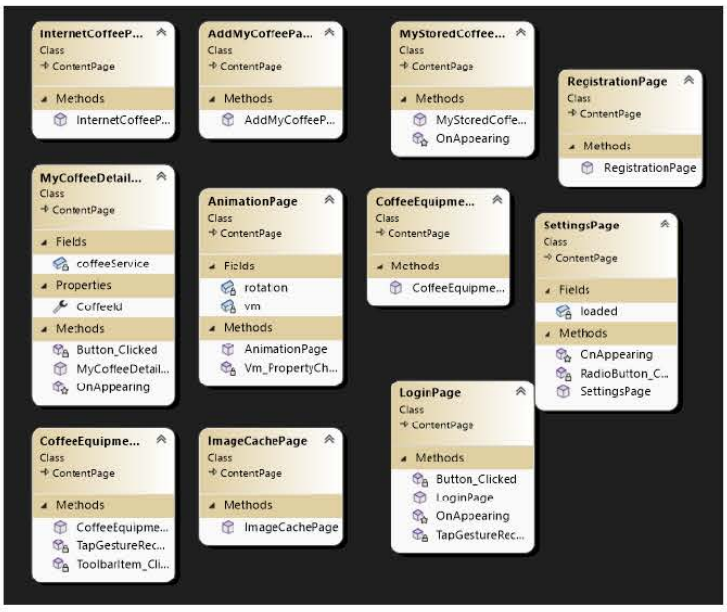


Рисунок 3.9 – Класи представлень (Views)

Наступним кроком потрібно розглянути роботу важливих класів, що надають функціональність додатку.

Перший клас, який потрібно розглянути це – InternetCoffeeViewModel.

```
namespace MyCoffeeApp.ViewModels
{
    1 reference
    public class InternetCoffeeViewModel : ViewModelBase
    {
        3 references
        public ObservableRangeCollection<Coffee> Coffee { get; set; }
        1 reference
        public AsyncCommand RefreshCommand { get; }
        1 reference
        public AsyncCommand AddCommand { get; }
        1 reference
        public AsyncCommand<Coffee> RemoveCommand { get; }


        0 references
        public InternetCoffeeViewModel()
        {

            Title = "Internet Coffee";

            Coffee = new ObservableRangeCollection<Coffee>();


            RefreshCommand = new AsyncCommand(Refresh);
            AddCommand = new AsyncCommand(Add);
            RemoveCommand = new AsyncCommand<Coffee>(Remove);
        }

        1 reference
```

Рисунок 3.10 – Клас InternetCoffeeViewModel

Згідно назви можна одразу визначити, що даний клас є частиною архітектури MVVC. Серед важливих функцій даного класу варто зазначити:

- Add(): Відображає діалогові вікна для введення назви кави, додає нову каву за допомогою InternetCoffeeService та оновлює список даних (рис. 3.11).

```
1 reference
async Task Add()
{
    var name = await App.Current.MainPage.DisplayPromptAsync("Name", "Name of coffee");
    var roaster = await App.Current.MainPage.DisplayPromptAsync("Roaster", "Roaster of coffee");
    await InternetCoffeeService.AddCoffee(name, roaster);
    await Refresh();
}
```

Рисунок 3.11 – Функція Add

- Remove(Coffee coffee): Видаляє каву за її ідентифікатором через InternetCoffeeService (рис. 3.12).

```
1 reference
async Task Remove(Coffee coffee)
{
    await InternetCoffeeService.RemoveCoffee(coffee.Id);
    await Refresh();
}
```

Рисунок 3.12 – Функція видалення кави

- Refresh(): Оновлює список кави, встановлюючи індикатор завантаження (IsBusy), після закінчення дії таймера протягом 2 секунд, очищує поточну колекцію та додає свіжі дані з InternetCoffeeService.

```
3 references
async Task Refresh()
{
    IsBusy = true;

    await Task.Delay(2000);

    Coffee.Clear();

    var coffees = await InternetCoffeeService.GetCoffee();

    Coffee.AddRange(coffees);

    IsBusy = false;
}
```

Рисунок 3.13 – Функція Refresh

Наступний клас AddMyCoffeeViewModel, використовується для зберігання наступних атрибутів: Name та Roaster представляють назву та обсмажувача кави відповідно. Вони використовують метод SetProperty, який автоматично викликає оновлення інтерфейсу при зміні значень.

Метод Save: Асинхронний метод, який перевіряє, чи властивості name та roaster не є пустими або складаються лише з пробілів. Якщо вони заповнені, метод додає нову каву через coffeeService та виконує навігацію назад до попередньої сторінки за допомогою Shell.Current.GoToAsync("..").

```
1 reference
public class AddMyCoffeeViewModel : ViewModelBase
{

    string name, roaster;
    2 references
    public string Name { get => name; set => SetProperty(ref name, value); }
    0 references
    public string Roaster { get => roaster; set => SetProperty(ref roaster, valu

    1 reference
    public AsyncCommand SaveCommand { get; }
    ICoffeeService coffeeService;
    0 references
    public AddMyCoffeeViewModel()
    {
        Title = "Add Coffee";
        SaveCommand = new AsyncCommand(Save);
        coffeeService = DependencyService.Get<ICoffeeService>();
    }

    1 reference
    async Task Save()
    {
        if(string.IsNullOrWhiteSpace(name) ||
            string.IsNullOrWhiteSpace(roaster))
        {
            return;
        }

        await coffeeService.AddCoffee(name, roaster);

        await Shell.Current.GoToAsync("..");
    }
}
```

Рисунок 3.14 – Клас AddMyCoffeeViewModel

Далі наступний клас CoffeeEquipmentViewModel, який має ряд важливих функцій, крім цього він на початку зберігає важливу колекцію даних та асинхронних процесів для роботи додатку (рис. 3.15):

```
namespace MyCoffeeApp.ViewModels
{
    1 reference
    public class CoffeeEquipmentViewModel : ViewModelBase
    {
        12 references
        public ObservableRangeCollection<Coffee> Coffee { get; set; }
        5 references
        public ObservableRangeCollection<Grouping<string, Coffee>> CoffeeGroups { get; }

        1 reference
        public AsyncCommand RefreshCommand { get; }

        1 reference
        public AsyncCommand<Coffee> FavoriteCommand { get; }
        1 reference
        public AsyncCommand<object> SelectedCommand { get; }

        1 reference
        public Command LoadMoreCommand { get; }
        1 reference
        public Command DelayLoadMoreCommand { get; }
        1 reference
        public Command ClearCommand { get; }

        0 references
        public CoffeeEquipmentViewModel()
        {
```

Рисунок 3.15 – Клас CoffeeEquipmentViewModel

Перший метод даного класу - Favorite(Coffee coffee), Асинхронний метод, який додає об'єкт кави до улюблених. Якщо переданий об'єкт coffee є null, метод повертається без виконання подальших дій (рис. 3.16).

```
1 reference
async Task Favorite(Coffee coffee)
{
    if (coffee == null)
        return;

    await Application.Current.MainPage.DisplayAlert("Favorite", coffee.Name, "OK");

}
```

Рисунок 3.16 – Метод Favorite

Selected(object args) - асинхронний метод, який викликається при виборі об'єкта кави. Приймає аргумент args, який приводиться до типу Coffee. Далі викликає навігацію до сторінки AddMyCoffeePage.

```
1 reference
async Task Selected(object args)
{
    var coffee = args as Coffee;
    if (coffee == null)
        return;

    SelectedCoffee = null;


    await AppShell.Current.GoToAsync(nameof(AddMyCoffeePage));
    //await Application.Current.MainPage.DisplayAlert("Selected"

}
```

Рисунок 3.17 – Метод Selected

Для завантаження додаткових даних використовується функція LoadMore. Вона перевіряє кількість об'єктів кави, якщо більше 20, то просто припиняє виконання функції, в іншому випадку додає нові об'єкти (рис. 3.18):

```
void LoadMore()
{
    if (Coffee.Count >= 20)
        return;

    var image = "coffeebag.png";
    Coffee.Add(new Coffee { Roaster = "Yes Plz", Name = "Sip of Sunshine", Image = image });
    Coffee.Add(new Coffee { Roaster = "Yes Plz", Name = "Potent Potable", Image = image });
    Coffee.Add(new Coffee { Roaster = "Yes Plz", Name = "Potent Potable", Image = image });
    Coffee.Add(new Coffee { Roaster = "Blue Bottle", Name = "Kenya Kiambu Handege", Image = image });
    Coffee.Add(new Coffee { Roaster = "Blue Bottle", Name = "Kenya Kiambu Handege", Image = image });

    CoffeeGroups.Clear();

    CoffeeGroups.Add(new Grouping<string, Coffee>("Blue Bottle", Coffee.Where(c => c.Roaster == "Blue Bottle")));
    CoffeeGroups.Add(new Grouping<string, Coffee>("Yes Plz", Coffee.Where(c => c.Roaster == "Yes Plz")));
}
```

Рисунок 3.18 – Функція до завантаження об'єктів

Остання функція використовується для очищення об'єктів Coffee (рис. 3.19):

```
1 reference
void Clear()
{
    Coffee.Clear();
    CoffeeGroups.Clear();
}
```

Рисунок 3.19 – Функція для очищення об'єктів

В проекті також використовуються функції для управління зображеннями та їх кешування використовуючи наступний спеціалізований клас (рис. 3.20):

```
3 references
public  class ImageCacheViewModel : ViewModelBase
{
    2 references
    public UriImageSource Image { get; set; } =
        new UriImageSource
        {
            Uri = new Uri("https://images.wsdot.wa.gov/sw/005vc00032.jpg"),
            CachingEnabled = true,
            CacheValidity = TimeSpan.FromMinutes(1)
        };

    1 reference
    public Command RefreshCommand { get; }

    0 references
    public ImageCacheViewModel()
    {
        RefreshCommand = new Command(() =>
        {
            Image = new UriImageSource
            {
                Uri = new Uri("https://images.wsdot.wa.gov/sw/005vc00032.jpg"),
                CachingEnabled = true,
                CacheValidity = TimeSpan.FromMinutes(1)
            };
            OnPropertyChanged(nameof(Image));
        });

        RefreshLongCommand = new AsyncCommand(async () =>
        {
            IsBusy = true;
            await Task.Delay(5000);
            IsBusy = false;
        });

    }

    1 reference
    public AsyncCommand RefreshLongCommand { get; }

}
```

Рисунок 3.20 – Клас ImageCacheViewModel

В даному класі використовуються 2 команди:

RefreshCommand: команда для оновлення зображення. При виконанні створює новий об'єкт UriImageSource з тими ж параметрами, що й початковий, і викликає метод OnPropertyChanged, щоб повідомити інтерфейс користувача про зміну властивості Image.

RefreshLongCommand: асинхронна команда для імітації тривалого процесу оновлення. Встановлює властивість IsBusy в true, очікує 5 секунд, після чого встановлює IsBusy в false.

Далі розглянемо представлення, їх класи , та яким чином реалізований функціонал реагує на події, що відбуваються в результаті користування цим додатком.

Перше представлення InternetCoffeePage, практично не має функціоналу. Використовується тільки для завантаження:

```
IsRefreshing="{Binding IsBusy, Mode=OneWay}"
ItemsSource="{Binding Coffee}"
RefreshCommand="{Binding RefreshCommand}"
SelectionMode="None"
Style="{StaticResource CoffeeListView}"
CachingStrategy="RecycleElement">
<ListView.ItemTemplate>
    <DataTemplate x:DataType="model:Coffee">
        <ViewCell>
            <ViewCell.ContextActions>
                <MenuItem
                    Command="{Binding Source={x:Reference CoffeePage}, Path=Bin
                    CommandParameter="{Binding .}"
                    IsDestructive="True"
                    Text="Delete" />
            </ViewCell.ContextActions>
            <Grid Padding="10">
                <Frame CornerRadius="20" HasShadow="True">
                    <StackLayout Orientation="Horizontal">
                        <Image Source="{Binding Image}" WidthRequest="66" />
                        <StackLayout VerticalOptions="Center">
                            <Label
                                FontSize="Large"
                                Text="{Binding Name}"
                                VerticalOptions="Center" />
                            <Label
                                FontSize="Large"
                                Text="{Binding Roaster}"
                                VerticalOptions="Center" />
                            <Label
                                FontSize="Small"
                                Text="{Binding Id}"
                                VerticalOptions="Center" />
                        </StackLayout>
                    </StackLayout>
                </Frame>
            </Grid>
        </ViewCell>
    </DataTemplate>
</ListView.ItemTemplate>
istView>
```

Рисунок 3.21 – Розмітка представлення InternetCoffeePage

Наступне представлення AddMyCoffeePage, аналогічно попередньому представленню не має функціоналу, є тільки завантаження xaml. розмітки (рис. 3.22).

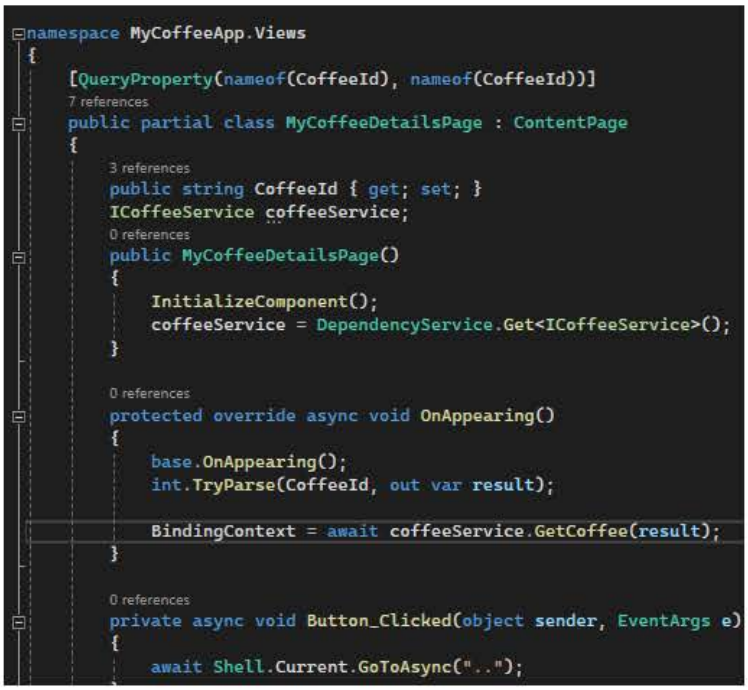
```
<ContentPage
    x:Class="MyCoffeeApp.Views.AddMyCoffeePage"
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:viewmodel="clr-namespace:MyCoffeeApp.ViewModels"
    Title="{Binding Title}"
    x:DataType="viewmodel:AddMyCoffeeViewModel"
    BackgroundColor="{AppThemeBinding
                    Dark={StaticResource WindowBackgroundColorDark},
                    Light={StaticResource WindowBackgroundColor}}"
    Shell.PresentationMode="Modal">
    <ContentPage.BindingContext>
        <viewmodel:AddMyCoffeeViewModel />
    </ContentPage.BindingContext>
    <ContentPage.Content>
        <StackLayout Padding="20">
            <Label Text="Name:"
                   Style="{StaticResource LabelMedium}"/>
            <Entry Text="{Binding Name}" />
            <Label Text="Roaster:"
                   Style="{StaticResource LabelMedium}" />
            <Entry Text="{Binding Roaster}" />

            <ActivityIndicator
                HorizontalOptions="Center"
                IsRunning="{Binding IsBusy}"
                IsVisible="{Binding IsBusy}"
                VerticalOptions="Center" />

            <Button
                Padding="20,0"
                Command="{Binding SaveCommand}"
                HorizontalOptions="Center"
                IsEnabled="{Binding IsNotBusy}"
                Text="Save"
                VerticalOptions="EndAndExpand"
                Style="{StaticResource ButtonOutline}"/>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Рисунок 3.22 – Представлення AddMyCoffeePage

Клас MyCoffeeDetailsPage представляє сторінку з деталями про конкретний об'єкт кави. Він успадковується від класу ContentPage, що забезпечує базову функціональність сторінки у мобільному додатку. Клас також використовує атрибут QueryProperty для передачі параметрів навігації (рис. 3.23).



```
namespace MyCoffeeApp.Views
{
    [QueryProperty(nameof(CoffeeId), nameof(CoffeeId))]
    7 references
    public partial class MyCoffeeDetailsPage : ContentPage
    {
        3 references
        public string CoffeeId { get; set; }
        ICoffeeService coffeeService;
        0 references
        public MyCoffeeDetailsPage()
        {
            InitializeComponent();
            coffeeService = DependencyService.Get<ICoffeeService>();
        }

        0 references
        protected override async void OnAppearing()
        {
            base.OnAppearing();
            int.TryParse(CoffeeId, out var result);

            BindingContext = await coffeeService.GetCoffee(result);
        }

        0 references
        private async void Button_Clicked(object sender, EventArgs e)
        {
            await Shell.Current.GoToAsync("..");
```

Рисунок 3.23 – Клас MyCoffeeDetailsPage

Клас використовує наступні властивості:

- CoffeeId - властивість типу string, що представляє ідентифікатор кави, який передається як параметр навігації. Цей ідентифікатор використовується для отримання даних про конкретну каву з сервісу.
- coffeeService - інтерфейс ICoffeeService, що забезпечує доступ до даних про каву. Ініціалізується за допомогою DependencyService, який дозволяє здійснювати ін'єкцію залежностей.

Дане представлення MyCoffeeDetailsPage теж має наступну xaml розмітку (рис. 3.24):

```
<ContentPage
    x:Class="MyCoffeeApp.Views.MyCoffeeDetailsPage"
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:model="clr-namespace:MyCoffeeApp.Shared.Models;assembly=MyCoffee
    Title="{Binding Name}"
    x:DataType="model:Coffee"
    BackgroundColor="{AppThemeBinding
                      Dark={StaticResource WindowBackgroundColorDark},
                      Light={StaticResource WindowBackgroundColor}}">
    <ContentPage.Content>
        <StackLayout Padding="20">
            <Label Text="{Binding Name, StringFormat='Name: {0}'}"
                   Style="{StaticResource LabelMedium}"/>
            <Label Text="{Binding Roaster, StringFormat='Roaster: {0}'}"
                   Style="{StaticResource LabelMedium}" />
            <Image HorizontalOptions="Center" Source="{Binding Image}" />

            <Button Clicked="Button_Clicked" Text="Done"
                   Style="{StaticResource ButtonOutline}"/>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Рисунок 3.24 – Xaml-розмітка MyCoffeeDetailsPage

Далі наступне представлення AnimationPage (рис. 3.25).

```
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class AnimationPage : ContentPage
{
    ImageCacheViewModel vm;

    readonly Animation rotation;

    public AnimationPage()
    {
        InitializeComponent();

        rotation = new Animation(v => LabelLoad.Rotation = v,
            0, 360, Easing.Linear);

        vm = (ImageCacheViewModel)BindingContext;

        vm.PropertyChanged += Vm_PropertyChanged;
    }

    private void Vm_PropertyChanged(object sender, System.ComponentModel.P
    {
        if(e.PropertyName == nameof(vm.IsBusy))
        {
            if(vm.IsBusy)
            {
                //animate
                rotation.Commit(this, "rotate", 16, 1000, Easing.Linear,
                    (v, c) => LabelLoad.Rotation = 0,
                    () => true);
            }
            else
            {
                //stop
```

Рисунок 3.25 – Клас AnimationPage

Конструктор AnimationPage ініціалізує компонент сторінки, створює анімацію обертання для елемента LabelLoad і встановлює прив'язку до моделі представлення. Також додає обробник події для зміни властивостей моделі представлення. Розмітка даного класи виглядає наступним чином:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:viewmodels="clr-namespace:MyCoffeeApp.ViewModels"
             x:Class="MyCoffeeApp.Views.AnimationPage"
             x:DataType="viewmodels:ImageCacheViewModel"
             Title="Animations"
             xmlns:fontawesome="clr-namespace:FontAwesome"
             xmlns:helpers="clr-namespace:MyCoffeeApp.Helpers"
    BackgroundColor="{AppThemeBinding
                      Dark={StaticResource WindowBackgroundColorDark},
                      Light={StaticResource WindowBackgroundColor}}">
    <ContentPage.BindingContext>
        <viewmodels:ImageCacheViewModel />
    </ContentPage.BindingContext>

    <StackLayout>
        <Button Text="Animate"
                IsEnabled="{Binding IsNotBusy}"
                Command="{Binding RefreshLongCommand}"/>

        <Label Text="{x:Static fontawesome:FontAwesomeIcons.Spinner}"
               FontFamily="FAS"
               x:Name="LabelLoad"
               IsVisible="{Binding IsBusy}"
               FontSize="80"
               Style="{StaticResource LabelLarge}"
               HorizontalOptions="Center"/>
    </StackLayout>

</ContentPage>
```

Рисунок 3.26 – Xaml-розмітка AnimationPage

Представлення CoffeeEquipmentCVPage відсутній функціонла в класі. Використовується для завантаження сторінко.

```
using Xamarin.Forms.Xaml;

namespace MyCoffeeApp.Views
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    4 references
    public partial class CoffeeEquipmentCVPage : ContentPage
    {
        0 references
        public CoffeeEquipmentCVPage()
        {
            InitializeComponent();
        }
    }
}
```

Рисунок 3.27 – Клас CoffeeEquipmentCVPage

CoffeeEquipmentCVPage має наступний вигляд xaml-розмітки (рис. 3.29):

Рисунок 3.29 – Xaml-розмітка CoffeeEquipmentCVPage

Клас MainActivity інтегрує основні функції для запуску додатку на платформі Android, включаючи ініціалізацію Xamarin.Forms та обробку дозволів. Класи Toaster та Environment забезпечують додаткові сервісні функції для відображення сповіщень та налаштування.

```
using System;

using Android.App;
using Android.Content.PM;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
using MyCoffeeApp.Services;
using Xamarin.Essentials;
using Xamarin.Forms;
using MyCoffeeApp.Droid;
using MyCoffeeApp.Helpers;
using AndroidX.Core.View;
using System.Threading.Tasks;

[assembly: Dependency(typeof(Toaster))]
[assembly: Dependency(typeof(MyCoffeeApp.Droid.Environment))]

namespace MyCoffeeApp.Droid
{
    [Activity(Label = "MyCoffeeApp", Theme = "@style/MainTheme", MainLauncher = true, Configura
    0 references
    public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsAppCompatActivity
    {
        0 references
        protected override void OnCreate(Bundle savedInstanceState)
        {
            TabLayoutResource = Resource.Layout.Tabbar;
            ToolbarResource = Resource.Layout.Toolbar;

            base.OnCreate(savedInstanceState);

            Xamarin.Essentials.Platform.Init(this, savedInstanceState);
            global::Xamarin.Forms.Forms.Init(this, savedInstanceState);
            LoadApplication(new App());
        }
        0 references
```

Рисунок 3.30 – Клас MainActivity

Метод даного класу OnCreate становлює ресурси для вкладок та інструментів та ініціалізує Xamarin.Essentials та Xamarin.Forms (рис. 3.31).

```
0 references
protected override void OnCreate(Bundle savedInstanceState)
{
    TabLayoutResource = Resource.Layout.Tabbar;
    ToolbarResource = Resource.Layout.Toolbar;

    base.OnCreate(savedInstanceState);

    Xamarin.Essentials.Platform.Init(this, savedInstanceState);
    global::Xamarin.Forms.Forms.Init(this, savedInstanceState);
    LoadApplication(new App());
}
```

Рисунок 3.31 – Метод OnCreate

Клас Environment реалізує інтерфейс IEnvironment і відповідає за налаштування кольору статусної панелі на платформі Android. Для управління кольором використовується функція SetStatusBarColor (рис. 3.32).

```
1 reference
public class Environment : IEnvironment
{
    3 references
    public async void SetStatusBarColor(System.Drawing.Color color, bool darkStatusBarTint)
    {
        if (Build.VERSION.SdkInt < Android.OS.BuildVersionCodes.Lollipop)
            return;

        var activity = Platform.CurrentActivity;
        var window = activity.Window;

        //this may not be necessary(but may be fore older than M)
        window.AddFlags(Android.Views.WindowManagerFlags.DrawsSystemBarBackgrounds);
        window.ClearFlags(Android.Views.WindowManagerFlags.TranslucentStatus);


        if (Build.VERSION.SdkInt >= Android.OS.BuildVersionCodes.M)
        {
            await Task.Delay(50);
            WindowCompat.GetInsetsController(window, window.DecorView).AppearanceLightStatusBars = darkStatusBarTint
        }

        window.SetStatusBarColor(color.ToPlatformColor());
    }
}
```

Рисунок 3.32 – Клас Environment

Останній важливий клас даного проекту - це CoffeeService. Даний клас в собі містить важливі функції які перевизначені відповідним інтерфейсом ICoffeeService (рис. 3.33).

```
using MyCoffeeApp.Services;
using MyCoffeeApp.Shared.Models;
using SQLite;
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Essentials;
using Xamarin.Forms;

[assembly:Dependency(typeof(CoffeeService))]
namespace MyCoffeeApp.Services
{
    1 reference
    public class CoffeeService : ICoffeeService
    {
        SQLiteAsyncConnection db;
        4 references
        async Task Init()
        {
            if (db != null)
                return;

            // Get an absolute path to the database file
            var databasePath = Path.Combine(FileSystem.AppDataDirectory, "MyData.db");

            db = new SQLiteAsyncConnection(databasePath);

            await db.CreateTableAsync<Coffee>();
        }

        2 references
        public async Task AddCoffee(string name, string roaster)
        {
            await Init();
            var image = "coffeebag.png";
            var coffee = new Coffee
            {
```

Рисунок 3.33 – Клас CoffeeService

Функція Init забезпечує ініціалізацію бази даних. Вона перевіряє, чи вже існує підключення до бази даних (db). Якщо підключення вже існує, функція просто повертається без результатів. Якщо підключення ще не створено, функція визначає абсолютний шлях до файлу бази даних, використовуючи метод Path.Combine і директрію FileSystem.AppDataDirectory. Потім вона створює асинхронне підключення до SQLite бази даних за цим шляхом і створює таблицю Coffee, якщо вона ще не існує, за допомогою методу CreateTableAsync (рис. 3.34).

```
async Task Init()
{
    if (db != null)
        return;

    // Get an absolute path to the database file
    var databasePath = Path.Combine(FileSystem.AppDataDirectory, "MyData.db");

    db = new SQLiteAsyncConnection(databasePath);

    await db.CreateTableAsync<Coffee>();
}
```

Рисунок 3.34 – Функція Init

Функція AddCoffee відповідає за додавання нового об'єкта кави до бази даних. Вона викликає метод Init для забезпечення ініціалізації бази даних. Потім створює новий об'єкт Coffee з вказаними іменем. Після цього функція додає новий об'єкт кави в базу даних за допомогою методу InsertAsync і отримує ідентифікатор вставленого запису (рис. 3.35).

```
2 references
public async Task AddCoffee(string name, string roaster)
{
    await Init();
    var image = "coffeebag.png";
    var coffee = new Coffee
    {
        Name = name,
        Roaster = roaster,
        Image = image
    };

    var id = await db.InsertAsync(coffee);
}
```

Рисунок 3.35 – Функція AddCoffee

Функція RemoveCoffee видаляє об'єкт кави з бази даних за його ідентифікатором. Вона також починає з виклику методу Init для забезпечення ініціалізації бази даних. Після цього викликає метод DeleteAsync, передаючи йому тип об'єкта Coffee і ідентифікатор запису, який потрібно видалити (рис. 3.36).

```
2 references
public async Task RemoveCoffee(int id)
{

    await Init();

    await db.DeleteAsync<Coffee>(id);
}
```

Рисунок 3.36 - Функція RemoveCoffee

Функція GetCoffee повертає всі об'єкти кави з бази даних. Вона викликає метод Init для ініціалізації бази даних. Потім використовує метод Table<Coffee>().ToListAsync для асинхронного отримання всіх записів з таблиці Coffee і перетворює їх у список. Повертає список об'єктів кави як результат (рис. 3.37).

```
2 references
public async Task<IEnumerable<Coffee>> GetCoffee()
{
    await Init();

    var coffee = await db.Table<Coffee>().ToListAsync();
    return coffee;
}
```

Рисунок 3.37 - Функція GetCoffee

Функція GetCoffee з параметром id повертає конкретний об'єкт кави з бази даних за вказаним ідентифікатором. Вона викликає метод Table<Coffee>().FirstOrDefaultAsync, який шукає перший об'єкт, що відповідає заданому ідентифікатору. Повертає знайдений об'єкт кави як результат (рис. 3.38).

```
2 references
public async Task<Coffee> GetCoffee(int id)
{
    await Init();

    var coffee = await db.Table<Coffee>()
        .FirstOrDefaultAsync(c => c.Id == id);

    return coffee;
}
```

Рисунок 3.38 – Функція GetCoffee(int id)

3.5. Тестування додатку

Для зручності тестування додатку Visual Studio дозволяє завантажувати проект на різні платформи зокрема на Android та IOS. Для завантаження віртуалізованої машини потрібно спочатку відкрити диспетчер пристроїв Android:

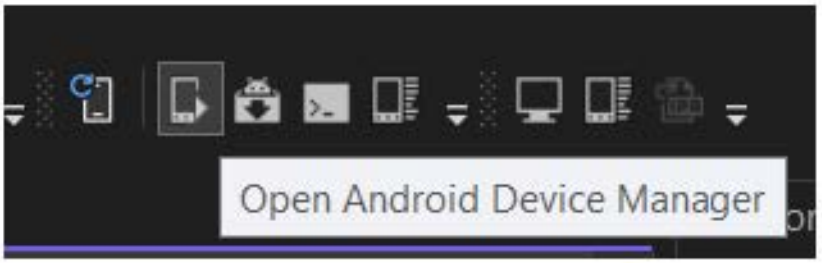


Рисунок 3.39 – Кнопка Open Android Device Manager

Після відкриття можна вибрати вже існуючий пристрій:

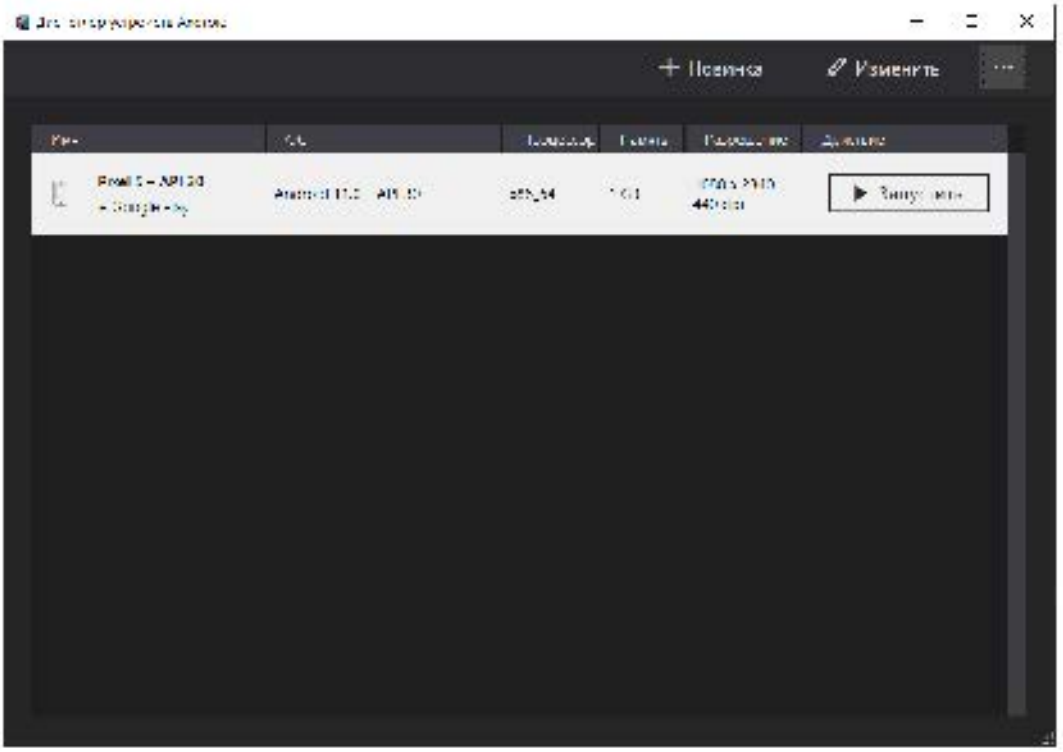


Рисунок 3.40 – Вибір віртуального пристрою

Якщо в розробника є бажання він може додати інший пристрій, вибравши потрібні йому параметри починаючи від операційної системи до вибору процесора (рис. 3.41):

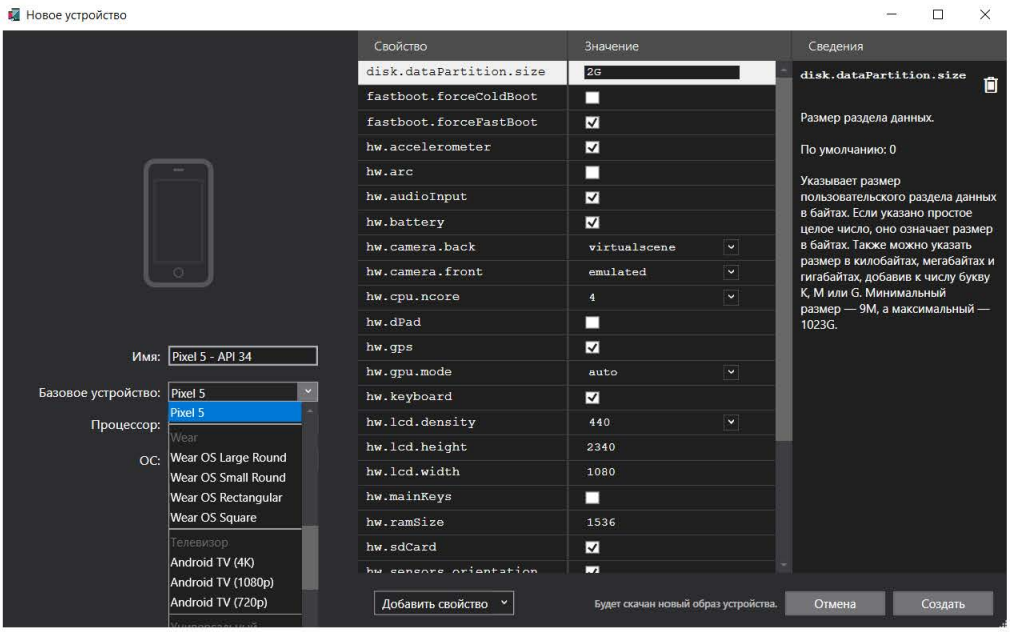


Рисунок 3.41 – Створення нового віртуального пристрою

Для тестування обрано пристрій Pixel 5 на операційні системі Android, запускаємо віртуальне середовище (рис. 3.42):

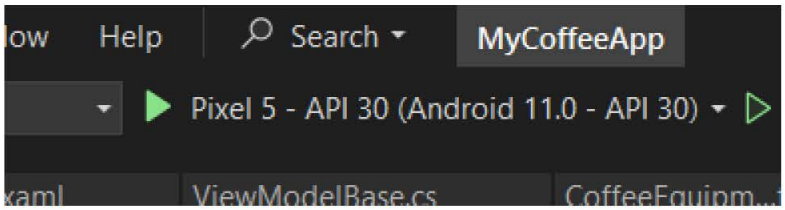


Рисунок 3.42 – Кнопка запуску проекту

На екрані в користувача повинен з’явитися вікно з обраним пристроєм (рис. 3.43):

Рисунок 3.43 – Поява віртуального пристрою

Xamarin дозволяє за допомогою Android SDK тестувати проекти відразу з комп'ютера розробника, що не потребує додаткових мобільних пристроїв. Далі потрібно знайти в меню знайти відповідну програму та відкрити її:

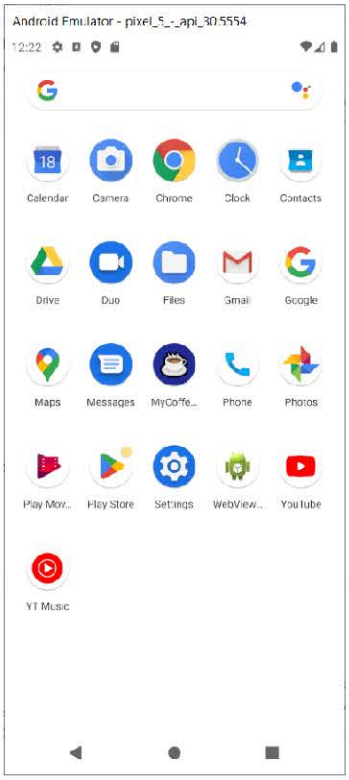


Рисунок 3.44 – Відкриття меню з програмами

Далі знаходимо потрібну програму та запускаємо її (рис. 3.45):

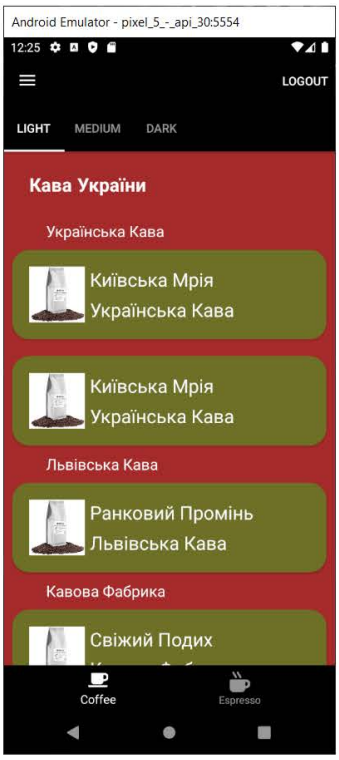


Рисунок 3.45 – Відкриття програми

Користувач на своєму екрані може спостерігати декілька пропозицій у виборі фірми виробника та кави, орієнтуючись по назві

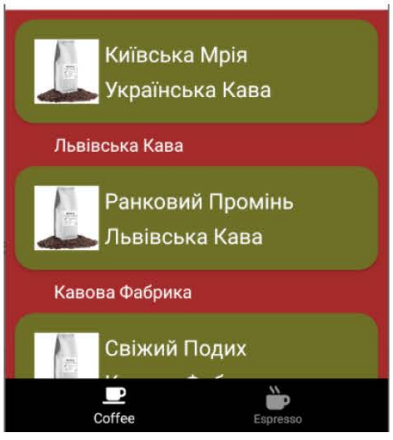


Рисунок 3.46 – Назви фірми та продукту

Якщо користувач обере каву та певний час затисне в меню, то з’явиться додаткові елементи інтерфейсу, за допомогою яких користувач може або видалити пропозицію або відмітити як фаворита (рис. 3.47):

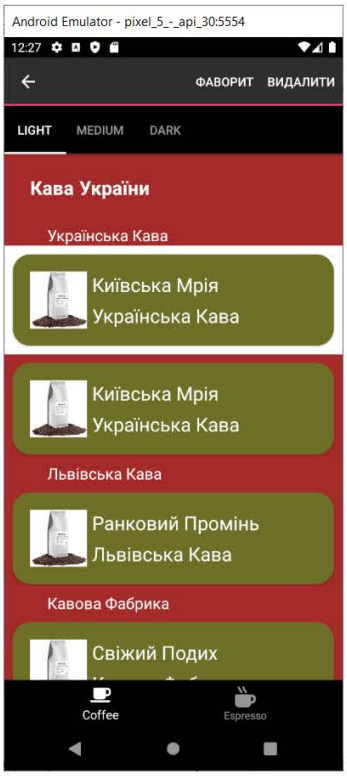


Рисунок 3.47 – Видалення пропозиції

Після вибору з’явиться на екрані наступне повідомлення, в якому зазначено вибір користувача:

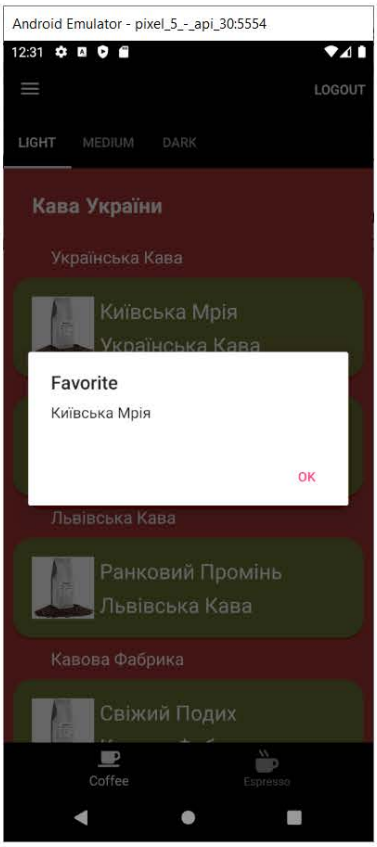


Рисунок 3.48 – Повідомлення про додавання кави до улюблених

Якщо користувачу пропозиції не вистачає то він може натиснути на кнопку «Завантажити» (рис. 3.49):

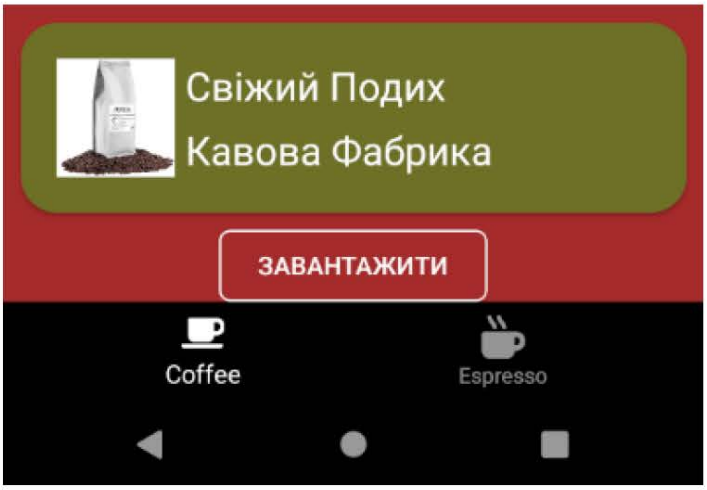


Рисунок 3.49 – До завантаження пропозицій

Користувачу також доступне бокове меню, в якому може перейти на інші сторінки додатку:

Рисунок 3.50 – Відкриття бокового меню

Для прикладу можна перейти до збереженої кави користувача:

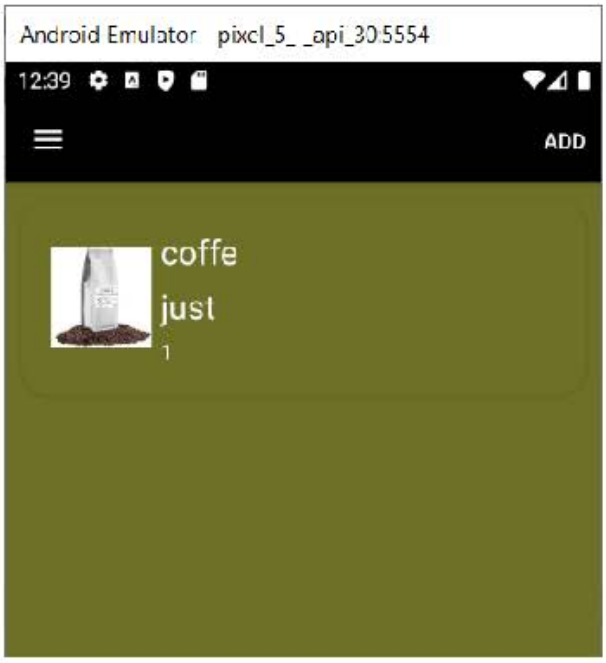


Рисунок 3.51 – Список збереженої кави

3.6. Висновок до третього розділу

Результатом розробки є крос-платформенний додаток з продажу кави, який забезпечує доступ до широкого асортименту кавових продуктів для користувачів на різних мобільних платформах, таких як iOS та Android. Використання сучасних технологій, таких як Xamarin.Forms для розробки

інтерфейсу, дозволяє створити додаток з інтуїтивно зрозумілим інтерфейсом, швидким доступом до даних і можливістю персоналізації замовлень. Додаток забезпечує користувачам зручний спосіб перегляду та вибору кавових продуктів, отримання інформації про нові пропозиції від вітчизняних виробників. Крім того, інтеграція функцій кешування зображень та відображення анімацій підвищує користувацький досвід, роблячи додаток привабливим і легким у використанні.

Розглянута структура проекту, описані програмні технології розробки. для повного розуміння всіх функціональних компонентів додатку побудовані діаграми класів та описана їх робота. Наведено елементи програмного коду, які демонструють роботу функцій, відповідальних за роботу розробленого програмного забезпечення. Проведено тестування додатку з описом основних етапів його роботи.

ВИСНОВОК

Кваліфікаційна робота присвячена розробці крос-платформного додатку для замовлення кавових товарів, що є актуальним завданням у контексті сучасного розвитку мобільних технологій. Метою роботи було створення зручного та функціонального інструменту для автоматизації процесу замовлення кавових товарів, забезпечуючи доступність додатку для користувачів на різних операційних системах.

У першому розділі роботи було проведено детальний аналіз предметної області, зокрема дослідження сучасних методів та підходів до розробки крос-платформних додатків. Було визначено ключові тенденції та технології, які використовуються для створення таких додатків. Проведено аналіз різних підходів до розробки, таких як гібридні додатки, нативні додатки з крос-платформними фреймворками, прогресивні веб-додатки (PWA) та інтерпретовані додатки. Зроблено висновок, що вибір між крос-платформною та нативною розробкою залежить від специфічних вимог проекту, цільової аудиторії та бюджету.

У другому розділі роботи було проаналізовано та обрано оптимальні технології для реалізації проекту. Розглянуто найпопулярніші фреймворки для створення крос-платформних додатків, такі як React Native, Flutter, Adobe PhoneGap, NativeScript, Xamarin та інші. На основі порівняння різних фреймворків було обрано платформу Xamarin, яка дозволяє створювати крос-платформні додатки з єдиною кодовою базою, забезпечуючи високу продуктивність та нативний вигляд додатків.

У третьому розділі було детально описано процес розробки крос-платформного додатку для замовлення кавових товарів. Реалізовано основні компоненти додатку, включаючи архітектуру MVVM, алгоритми обробки даних та інтеграцію з API кавових товарів. Проведене тестування проекту підтвердило його функціональність та відповідність вимогам. Додаток

забезпечує зручний спосіб перегляду та замовлення кавових продуктів, отримання інформації про нові пропозиції та персоналізацію замовлень.

Розроблений проект демонструє високу ефективність та відповідає сучасним вимогам індустрії програмного забезпечення. Використання кросплатформних технологій дозволило досягти високої гнучкості та адаптивності коду, що полегшує його подальшу підтримку та розвиток. Результати роботи можуть бути корисними для інших розробників та дослідників у сфері програмної інженерії, які прагнуть створювати інноваційні продукти, що відповідають сучасним тенденціям та очікуванням користувачів.

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

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

4. Документація Kotlin Multiplatform Mobile. URL: https://kotlinlang.org/docs/multiplatform.html

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

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

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

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

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

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

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

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

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

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

15. Що таке UML-діаграми? – [Електронний ресурс] – Режим доступу: https://evergreens.com.ua/ua/articles/uml-diagrams.html