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 IT новин»

Виконав: студент групи ІП320-1
Спеціальність 121 «Інженерія програмного забезпечення»
Лисиця Станіслав Юрійович
(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 Кли В.Ю.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2024

АНОТАЦІЯ

Лисиця С.Ю. Розробка крос-платформного додатку ІТ новин.

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

Кваліфікаційна робота присвячена розробці крос-платформного додатку для новин ІТ-сфери. У сучасному суспільстві мобільні пристрої стали невід'ємною частиною повсякденного життя, що зумовлює необхідність розробки додатків, здатних працювати на різних операційних системах, таких як iOS та Android.

В процесі дослідження було проведено аналіз сучасних методів та інструментів розробки крос-платформних додатків, зокрема фреймворків .NET MAUI, React Native, Flutter та інших. Вибір .NET MAUI обґрунтовано його високою продуктивністю, підтримкою сучасних мов програмування та широкими можливостями інтеграції.

В роботі реалізовано функціональний крос-платформний додаток, який дозволяє користувачам отримувати актуальні новини ІТ-сфери на будь-якому мобільному пристрої. Додаток було розроблено з використанням .NET MAUI, C# та Visual Studio Community 2022.

Результати роботи можуть бути використані для подальшого розвитку подібних додатків та автоматизації збору інформації в інших сферах. Виконана робота демонструє можливості крос-платформних технологій та їх ефективність у сучасній розробці програмного забезпечення.

Ключові слова: крос-платформний додаток, ІТ-новини, .NET MAUI, C#, Visual Studio.

ABSTRACT

Lysytsia S.Yu. Development of a Cross-Platform IT News Application.

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

The qualification work is devoted to the development of a cross-platform application for IT news. In modern society, mobile devices have become an integral part of daily life, necessitating the development of applications capable of operating on various operating systems such as iOS and Android.

The research involved analyzing modern methods and tools for developing cross-platform applications, including frameworks such as .NET MAUI, React Native, Flutter, and others. The choice of .NET MAUI is justified by its high performance, support for modern programming languages, and extensive integration capabilities.

The work implemented a functional cross-platform application that allows users to receive up-to-date IT news on any mobile device. The application was developed using .NET MAUI, C#, and Visual Studio Community 2022.

The results of the work can be used for further development of similar applications and the automation of information collection in other areas. The completed work demonstrates the capabilities of cross-platform technologies and their efficiency in modern software development.

Keywords: cross-platform application, IT news, .NET MAUI, C#, Visual Studio.

ЗМІСТ

ВСТУП 5

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 7

1.1. Аналіз публікацій щодо розробки крос-платформних додатків 7

1.2. Аналіз методів та підходів розробки крос-платформних додатків 9

1.3. Висновок до першого розділу 10

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ КОРС-ПЛАТФОРМНИХ ДОДАТКІВ 12

2.1. Вибір програмних засобів для реалізації проекту 12

2.2. Програмні засоби для розробки крос-платформного додатку 13

2.3. Висновок до другого розділу 21

РОЗДІЛ 3. РОЗРОБКА КРОСПЛАТФОРМНОГО ДОДАТКУ ІТ-НОВИН 22

3.1 Концепція створення кросплатформенного додатку 22

3.2 Структура проекту 22

3.3 Інструменти розробки 23

3.4 Розробка додатку 27

3.5 Тестування додатку 46

3.6 Висновок до третього розділу 49

ВИСНОВОК 51

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 53

ВСТУП

Актуальність проблеми. Сучасне суспільство активно користується мобільними пристроями для доступу до інформації. Смартфони та планшети стали невід'ємною частиною повсякденного життя, що зумовлює необхідність розробки додатків, здатних працювати на різних операційних системах, таких як iOS та Android. Це дозволяє охопити максимально широку аудиторію користувачів, надаючи їм можливість отримувати новини в будь-який час та в будь-якому місці. Крос-платформні додатки забезпечують таку універсальність.

Крос-платформний підхід дозволяє розробникам оперативно реагувати на зміни та нові вимоги ринку, забезпечуючи тим самим конкурентоспроможність продукту. Використання сучасних крос-платформних фреймворків, таких як .NET MAUI значно підвищує ефективність процесу розробки. Фреймворки дозволяють створювати єдину кодову базу для всіх платформ, що суттєво скорочує час та витрати на розробку. Завдяки єдиній кодовій базі для різних операційних систем, зміни та виправлення можуть бути внесені лише один раз, що знижує ризик помилок та забезпечує стабільну роботу додатку на всіх платформах, що значно спрощує роботу розробників та забезпечує високу якість та надійність програмного забезпечення.

Уніфікований дизайн, який можна створити завдяки крос-платформним фреймворкам. Єдиний дизайн, впізнаваний на різних платформах, забезпечує кращий користувацький досвід. Користувачі можуть легко орієнтуватися в додатку, оскільки інтерфейс залишається знайомим незалежно від операційної системи, яку вони використовують. Це сприяє зручності використання додатку та підвищує задоволення користувачів від взаємодії з ним.

Метою роботи є автоматизація роботи збору новин IT-сфери.

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 MAUI.

4. Провести тестування.

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

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

Структура роботи:

- Розділ 1 Аналіз існуючих рішень.
- Розділ 2 Аналіз засобів крос-платформних додатків
- Розділ 3 Розробка крос-платформного додатку ІТ-новин

Робота складається зі вступу, 3-х розділів, висновків, списку використаних джерел з 18 найменувань. Обсяг роботи 60 сторінок кваліфікаційної роботи, 39 рисунків, 2 таблиці.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз публікацій щодо розробки крос-платформних додатків

Розробка додатків розвивається стрімкими темпами вибір між нативними та крос-платформними підходами є важливим аспектом цього процесу. Нативні платформи забезпечують високу продуктивність та доступ до всіх можливостей операційної системи, а крос-платформні фреймворки мають свої значні переваги. З огляду на постійний розвиток обох підходів, доцільно регулярно аналізувати існуючі та нові крос-платформні рішення, щоб обрати найбільш ефективний інструментарій для конкретних проєктів.

Переваги Крос-Платформних Фреймворків

1. Економія часу

Розробники можуть швидше створювати та розгортати додатки, використовуючи гібридні рішення, що дозволяє швидше виходити на ринок. Це особливо важливо в умовах високої конкуренції, коли час виходу продукту на ринок може стати вирішальним фактором.

2. Зниження витрат

Використання концепції "пиши один раз, запускай скрізь" дозволяє значно зменшити витрати на розробку. Замість того, щоб мати окремі команди розробників для iOS та Android, можна використовувати одну команду, яка працює з єдиним кодом для обох платформ. Знижує витрати на розробку, спрощує підтримку та розвиток додатка [4].

3. Уніфікована кодова база

Крос-платформні фреймворки дозволяють мати єдину кодову базу для кількох операційних систем. Спрощує процес розробки та підтримки, оскільки всі зміни вносяться в один код, який потім адаптується для різних платформ.

4. Ширше охоплення аудиторії

Використовуючи крос-платформний підхід, можна створювати додатки, які працюють на різних платформах, що дозволяє охопити більшу аудиторію

розгортаючи один додаток на iOS, Android та веб-платформах. Таким чином, розробники можуть максимізувати охоплення своєї аудиторії та збільшити кількість користувачів свого додатка [1].

5. Поліпшене обслуговування та розгортання

Оновлення додатків можуть синхронізуватися на всіх платформах та пристроях, що значно економить час і кошти. Якщо в загальній кодовій базі виявлено помилку, її виправляють лише один раз, і це виправлення автоматично застосовується до всіх платформ.

6. Швидший вихід на ринок та оновлення

Концепція "пиши один раз, запускай скрізь" дозволяє швидко вносити зміни в код, що сприяє швидшому виходу продукту на ринок та покращує залучення клієнтів. Це особливо важливо в умовах високої конкуренції, коли швидкість виведення нових функцій на ринок може стати вирішальним фактором для успіху продукту.

7. Єдиний дизайн

Крос-платформні фреймворки дозволяють створювати уніфікований дизайн, який легко розпізнається користувачами на різних платформах. Це забезпечує кращий користувацький досвід (UX), оскільки користувачі можуть легко розпізнавати елементи інтерфейсу та розуміти їх функціональність незалежно від платформи, на якій вони використовують додаток.

Незважаючи на численні переваги, крос-платформна розробка має свої недоліки, які необхідно враховувати:

1. Проблеми загальної продуктивності

Крос-платформні додатки можуть працювати менш ефективно, що негативно впливає на користувацький досвід. Проблеми з продуктивністю можуть виникати через необхідність адаптації одного коду для кількох платформ, що іноді призводить до компромісів у продуктивності.

2. Обмежений доступ до нативних бібліотек

Використання специфічних для платформ бібліотек може бути ускладнене або неможливе, що обмежує функціональні можливості додатків, які вимагають

використання специфічних функцій або бібліотек, доступних лише на одній платформі [2].

1.2. Аналіз методів та підходів розробки крос-платформних додатків

У цьому контексті розглянемо основні методи та підходи до крос-платформної розробки.

Основні підходи до крос-платформної розробки

1) Гібридні додатки

Гібридні додатки створюються за допомогою веб-технологій, таких як HTML, CSS та JavaScript, і вбудовуються в нативні контейнери. Вони дозволяють розробникам використовувати один код для різних платформ. Основні інструменти для створення гібридних додатків включають Cordova, Ionic та Framework7.

2) Нативні скриптові фреймворки

Цей підхід включає використання мов програмування, які можуть бути інтерпретовані або скомпільовані у нативний код. Популярні фреймворки включають React Native та Flutter.

- React Native

Використовує JavaScript та дозволяє розробникам писати нативні компоненти для iOS та Android. React Native має велику спільноту та підтримку від Facebook [5].

- Flutter

Використовує мову Dart і пропонує високопродуктивні компоненти для створення нативних інтерфейсів. Flutter має підтримку від Google і забезпечує високу швидкість розробки та продуктивність [6].

3) Прогресивні веб-додатки (PWA)

Прогресивні веб-додатки використовують сучасні веб-технології для створення додатків, які можуть працювати як веб-додатки, так і як нативні

мобільні додатки. PWA використовують технології, такі як Service Workers та Web App Manifests [14].

Таблиця 1.1

Порівняння підходів

Підхід	Основні інструменти	Переваги	Недоліки
Гібридні додатки	Cordova, Ionic, Framework7	Швидкий розвиток, низькі витрати, доступ до нативних API через плагіни	Зниження продуктивності, обмежений доступ до нативних функцій
Нативні скриптові фреймворки	React Native, Flutter	Висока продуктивність, доступ до нативних API, активна спільнота	Великий розмір додатка, необхідність вивчення нових мов
Прогресивні веб-додатки (PWA)	HTML, CSS, JavaScript	Легкість у розгортанні, робота офлайн, зручність у підтримці	Обмежений доступ до нативних функцій, знижена продуктивність

1.3. Висновок до першого розділу

У першому розділі проведено всебічний аналіз існуючих рішень щодо розробки крос-платформних додатків. Розглянуто переваги та недоліки крос-платформних фреймворків, таких як React Native, Flutter, та інших, у порівнянні з нативними підходами. Зроблено акцент на ефективності, економії часу, зниженні витрат та уніфікації кодової бази, що дозволяє створювати додатки для різних платформ з мінімальними затратами.

Проаналізовано методи та підходи до розробки крос-платформних додатків, такі як гібридні додатки, нативні скриптові фреймворки та прогресивні веб-додатки (PWA). Детально розглянуто інструменти для кожного підходу, їх переваги та недоліки, а також особливості їх використання.

Таким чином, проведений аналіз дозволяє зробити висновок, що для ефективної розробки крос-платформного додатка важливо враховувати специфіку проекту та обирати відповідні інструменти і підходи, які забезпечать баланс між продуктивністю, витратами та часом розробки.

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ КОРС-ПЛАТФОРМНИХ ДОДАТКІВ

2.1. Вибір програмних засобів для реалізації проекту

Сьогодні серед найпопулярніших фреймворків для створення мобільних додатків можна виділити Kotlin Multiplatform Mobile, React Native, Flutter та Xamarin/MAUI. Для порівняння фреймворків було обрано такі критерії, як продуктивність роботи, наявність відкритого вихідного коду, спосіб побудови інтерфейсу користувача (UI), зрілість фреймворку (термін його існування), наявність великої спільноти користувачів, документація, підтримувані мови програмування та вартість використання [3].

При обмежених термінах розробки варто звернути увагу на кросплатформні рішення, які дозволяють створювати уніфікований інтерфейс користувача, що допомагає заощадити час. Однак уніфікований інтерфейс може мати недоліки, такі як візуальна «не нативність» додатку та можливе зниження продуктивності.

Розглянемо детальніше кожен з фреймворків:

React Native використовує блоки власного інтерфейсу користувача, які збираються за допомогою бібліотеки JavaScript від React. Розробники можуть писати код один раз на JavaScript, а React Native забезпечує створення версій для різних платформ. Проте продуктивність React Native нижча порівняно з нативними програмами, що може вплинути на досвід взаємодії користувача. Додатки React Native більші за розміром, що може бути незручним для користувачів старих або бюджетних телефонів. Крім того, налаштування додатків для кожної платформи вимагає додаткового коду [9].

Flutter має високу швидкість, зручність та простоту, що робить його підходящим як для новачків, так і для досвідчених програмістів. Однак є обмеження щодо використання рідкісних рідних бібліотек для ОС та створення програм, що напряму спілкуються з IoT пристроями через Bluetooth.

Розробникам доведеться використовувати налаштовані канали платформи або залежати від сторонніх плагінів, що може ускладнювати розробку [8].

Xamarin/.NET MAUI також має девіз «пиши один раз, запускай скрізь», але стикається з подібними проблемами, такими як розмір додатку та труднощі роботи з рідними бібліотеками платформ. Оновлення для Android або iOS потребують часу для інтеграції, що може затримувати оновлення додатків і впливати на репутацію компанії. Створення додатків із складними графічними елементами або анімацією за допомогою Xamarin є важким завданням [7, 10].

Kotlin Multiplatform Mobile дозволяє повторно використовувати спільну бізнес-логіку та створювати платформозалежний інтерфейс користувача. Це рішення підходить для розробників, які прагнуть створювати додатки для Android та iOS без дублювання роботи, зберігаючи принцип DRY (Don't repeat yourself) та нативний інтерфейс користувача для кожної з платформ. Сучасні додатки потребують сучасних архітектур, і Kotlin Multiplatform Mobile дозволяє досягти цього, поділяючи шари програми [11, 12].

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

2.2. Програмні засоби для розробки крос-платформного додатку

.NET 6 є важливою віхою у розвитку платформи .NET, об'єднуючи різні технології та інструменти в єдину кросплатформену екосистему. Цей реліз був створений для забезпечення високої продуктивності, зручності використання та гнучкості для розробників, дозволяючи створювати різноманітні типи додатків, включаючи веб-додатки, десктопні програми, мобільні додатки, ігри та хмарні рішення.

Однією з основних характеристик .NET 6 є підтримка кросплатформеності. Ця платформа дозволяє розробникам писати код, який може виконуватись на різних операційних системах, таких як Windows, macOS

та Linux, що робить її дуже зручною для розробки сучасних додатків. Розробники можуть використовувати єдину базу коду для різних платформ, зменшуючи час та витрати на розробку.

.NET 6 інтегрує декілька важливих компонентів, включаючи ASP.NET Core, Entity Framework Core, та Xamarin, що дозволяє створювати високопродуктивні веб-додатки, працювати з базами даних та створювати мобільні додатки. ASP.NET Core, наприклад, є потужним фреймворком для створення веб-додатків і API, який забезпечує високу продуктивність і масштабованість. Entity Framework Core надає зручні інструменти для роботи з базами даних, дозволяючи розробникам легко виконувати CRUD-операції (створення, читання, оновлення, видалення).

Окрім цього, .NET 6 включає в себе значні покращення в продуктивності. Компанія Microsoft постійно працює над оптимізацією платформи, щоб забезпечити швидке виконання коду та зменшити використання ресурсів. Це робить .NET 6 ідеальним вибором для створення високонавантажених систем, які потребують максимальної продуктивності.

Розширені можливості інструментів розробки також є важливою частиною .NET 6. Інтеграція з Visual Studio забезпечує зручне середовище для розробників, з потужними засобами налагодження, автодоповненням коду та іншими функціями, що покращують продуктивність роботи. Також .NET 6 підтримує нові функції мови програмування C#, включаючи покращення у роботі з шаблонами, спрощення синтаксису та нові можливості для асинхронного програмування.

Безпека є ще одним ключовим аспектом .NET 6. Платформа включає в себе нові механізми захисту даних, забезпечуючи надійний захист додатків від різних загроз. Розробники можуть використовувати вбудовані функції для аутентифікації та авторизації, шифрування даних та інших заходів безпеки.

MAUI (Multi-platform App UI) — це фреймворк, розроблений компанією Microsoft для створення кросплатформних додатків. Він є наступником Xamarin.Forms і дозволяє розробляти додатки для Android, iOS, macOS та

Windows, використовуючи єдину кодову базу. MAUI інтегрується з .NET 6, надаючи розробникам можливість створювати високопродуктивні сучасні додатки з єдиним користувацьким інтерфейсом.

Однією з ключових переваг MAUI є підтримка кількох платформ. Це значно зменшує час і зусилля, необхідні для створення додатків, оскільки розробники можуть писати код один раз і запускати його на різних платформах. Інтеграція з .NET 6 надає доступ до всіх переваг цього потужного фреймворку, включаючи високу продуктивність, багатий набір бібліотек і підтримку різних мов програмування, таких як C# і F#. MAUI також дозволяє об'єднати всі цільові платформи в єдиний проект, що спрощує управління кодовою базою та дозволяє легко впроваджувати зміни для всіх платформ.

Фреймворк MAUI підтримує як спільний користувацький інтерфейс для всіх платформ, так і платформозалежні налаштування, що дозволяє створювати додатки з нативним виглядом і відчуттям. MAUI є проектом з відкритим вихідним кодом, що дозволяє спільноті розробників вносити зміни, виправляти помилки і додавати нові функціональні можливості. Для опису інтерфейсу користувача використовується XAML (Extensible Application Markup Language), що забезпечує декларативний підхід до створення UI. Також підтримується створення інтерфейсу за допомогою C#.

Зрілість фреймворку є ще однією його перевагою. MAUI є наступником Xamarin.Forms, який існує з 2014 року і має велику спільноту користувачів та розробників, що забезпечує надійність і тривалу підтримку фреймворку. Microsoft забезпечує детальну документацію для MAUI, включаючи приклади, керівництва та API-документацію, а також активну підтримку з боку спільноти розробників. Основною мовою програмування для MAUI є C#, яка забезпечує високу продуктивність і зручність розробки.

Проте MAUI має і свої недоліки. Додатки на MAUI можуть бути більшими за розміром порівняно з нативними додатками. Випуск нових версій операційних систем може вимагати оновлення MAUI для підтримки нових функціональностей, що може викликати затримки. Хоча MAUI має високу

продуктивність, у деяких випадках нативні додатки можуть працювати швидше через відсутність накладних витрат на кросплатформні абстракції.

Отже, MAUI (Multi-platform App UI) є потужним фреймворком для створення кросплатформних додатків, який поєднує в собі переваги .NET 6 і надає гнучкість у створенні користувацького інтерфейсу. Він підходить для розробників, які прагнуть зменшити час та зусилля на створення додатків для кількох платформ, зберігаючи при цьому нативний вигляд та продуктивність.

Visual Studio Community 2022 пропонує потужні інструменти для відладки та тестування додатків. Відладчик дозволяє виконувати покрокову відладку, аналізувати значення змінних у реальному часі, встановлювати точки зупинки та досліджувати стек викликів. IDE підтримує модульне тестування і інтеграцію з популярними тестовими фреймворками, такими як NUnit, MSTest і xUnit, надаючи засоби для автоматизованого тестування, що підвищує якість програмних продуктів.

Інтеграція з Git та GitHub забезпечує зручну роботу з системами контролю версій безпосередньо з Visual Studio. Це дозволяє легко створювати, клонувати, комітити та пушити репозиторії, вирішувати конфлікти та працювати з гілками коду. Visual Studio Community 2022 також інтегрується з хмарними платформами, такими як Microsoft Azure, що дозволяє легко створювати, розгортати та керувати хмарними додатками.

Visual Studio підтримує велику кількість розширень, що дозволяють додавати нові функціональні можливості та налаштовувати IDE відповідно до потреб розробника. Marketplace Visual Studio пропонує тисячі розширень для різних мов програмування, фреймворків та інструментів. Завдяки інтеграції з Xamarin, Visual Studio Community 2022 дозволяє розробляти кросплатформні мобільні додатки для iOS та Android з використанням єдиного коду на C#. Це значно спрощує процес розробки та підтримки мобільних додатків [13].

Visual Studio Community 2022 підтримує розробку ігор з використанням Unity та Unreal Engine, що дозволяє створювати високоякісні 2D та 3D ігри. Крім того, Visual Studio Community 2022 підтримує сучасні технології та

фреймворки, такі як .NET 6, C# 10, ASP.NET Core, Blazor та інші, надаючи розробникам можливість використовувати найновіші інструменти та методики для створення інноваційних додатків.

Visual Studio Community 2022 є безкоштовною для індивідуальних розробників, студентів, відкритих проектів та навчальних закладів, що робить її доступною для широкого кола користувачів. Інтеграція з іншими продуктами Microsoft, такими як Azure, Office 365 та GitHub, дозволяє використовувати широкий спектр можливостей та сервісів у єдиному середовищі розробки, що підвищує продуктивність та спрощує процес розробки. Велика та активна спільнота розробників забезпечує доступ до численних ресурсів, включаючи документацію, форуми, відеоуроки та блоги, що полегшує процес навчання та вирішення проблем.

Visual Studio Community 2022 є потужним і універсальним середовищем розробки, яке надає розробникам всі необхідні інструменти для створення високоякісних додатків для різних платформ. Її безкоштовність, багатий набір можливостей, підтримка сучасних технологій та інтеграція з хмарними сервісами роблять її ідеальним вибором як для початківців, так і для досвідчених розробників.

2.3. Патерни проектування

Проєктні патерни відіграють важливу роль у розробці веб-додатків, забезпечуючи структурованість, зрозумілість та масштабованість коду. Вони дозволяють розробникам застосовувати перевірені рішення для типових проблем, що виникають під час створення програмного забезпечення. Серед найбільш популярних патернів, що використовуються у веб-розробці, можна виділити Model-View-Controller (MVC) та Model-View-ViewModel (MVVM). Кожен з цих патернів має свої переваги, недоліки та специфічну сферу застосування, що робить їх незамінними інструментами для сучасних розробників. Давайте розглянемо кожен з них детальніше.

MVC (Model-View-Controller) - це патерн проектування, що використовується для створення організованих та структурованих додатків, особливо в контексті веб-розробки. Він розділяє логіку додатка на три взаємодіючі компоненти: Model, View, та Controller. Основна мета MVC - розділити внутрішнє представлення інформації від способу її подання та взаємодії з користувачем.

1. Model - відповідає за управління даними додатка.
2. View - відповідає за відображення даних користувачу.
3. Controller - виступає як посередник між Model і View.

Переваги MVC:

1. Розділення обов'язків

MVC забезпечує чітке розділення обов'язків між різними компонентами додатка, що полегшує його розробку, тестування та підтримку.

2. Модульність

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

3. Гнучкість

Завдяки розділенню логіки додатка, MVC дозволяє легко змінювати та оновлювати окремі частини додатка без впливу на інші компоненти.

4. Повторне використання коду

Компоненти Model і View можуть бути повторно використані у різних частинах додатка або навіть у різних проектах [16].

MVVM (Model-View-ViewModel) - це патерн проектування, що використовується для створення структурованих та масштабованих додатків, особливо в контексті сучасних фреймворків та бібліотек, таких як WPF, Angular та React. Основна мета MVVM - розділення представлення інтерфейсу користувача та бізнес-логіки, що полегшує 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 користувача (HTML, XAML тощо). View 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 знижує 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 [17].

Результатом цього аналізу є таблиця, що надає порівняння основних характеристик патернів MVC та MVVM, показуючи їх переваги та недоліки, а також допомагаючи визначити, який патерн може бути найкращим вибором для конкретного проекту залежно від його вимог.

Таблиця 2.1

Порівня патернів проектування

Характеристика	MVC	MVVM
Компоненти	Model, View, Controller	Model, View, ViewModel
Розділення обов'язків	Чітке розділення логіки, представлення та контролю	Чітке розділення даних, логіки та інтерфейсу
Модульність	Висока: кожен компонент може бути розроблений, протестований і підтримуваний незалежно	Висока: кожен компонент може бути розроблений, протестований і підтримуваний незалежно
Гнучкість	Легке оновлення окремих частин додатка	Легке оновлення окремих частин додатка
Повторне використання коду	Можливість повторного використання Model і View у різних частинах додатка	Можливість повторного використання ViewModel і Model у різних частинах додатка
Двостороння прив'язка даних	Відсутня	Присутня: автоматичне оновлення View при зміні даних у ViewModel і навпаки
Простота тестування	Відносно проста	Відносно проста
Основне використання	Веб-додатки, серверні додатки	Клієнтські додатки, додатки з насиченим UI
Переваги	1. Чітке розділення обов'язків 2. Висока модульність 3. Гнучкість	1. Чітке розділення обов'язків. 2. Висока модульність. 3. Двостороння

	4. Повторне використання коду	прив'язка даних. 4. Повторне використання коду
Недоліки	1. Відсутність двосторонньої прив'язки даних 2. Може бути складним для початківців	1. Може бути складним для початківців 2. Потребує більше зусиль для налаштування двосторонньої прив'язки даних

2.3. Висновок до другого розділу

У другому розділі проведено аналіз засобів реалізації крос-платформних додатків. Розглянуто популярні фреймворки, такі як React Native, Flutter, Xamarin/.NET MAUI та Kotlin Multiplatform Mobile, з точки зору продуктивності, можливостей, наявності відкритого вихідного коду, способу побудови інтерфейсу користувача, зрілості фреймворку, підтримки різних мов програмування та вартісних аспектів.

Особливу увагу приділено .NET MAUI, який обрано для реалізації проекту. .NET MAUI інтегрується з .NET 6, що забезпечує високу продуктивність, кросплатформеність та підтримку сучасних мов програмування, таких як C#. Проаналізовано також Visual Studio Community 2022, яке пропонує потужні інструменти для відладки та тестування додатків, підтримує інтеграцію з Git та GitHub, а також надає можливості для створення крос-платформних мобільних додатків.

Загалом, проведений аналіз демонструє, що .NET MAUI разом з .NET 6 та Visual Studio Community 2022 є оптимальним вибором для розробки крос-платформного додатка, забезпечуючи високу продуктивність, гнучкість, зручність у використанні та підтримку сучасних технологій.

РОЗДІЛ 3. РОЗРОБКА КРОСПЛАТФОРМНОГО ДОДАТКУ ІТ-НОВИН

3.1 Концепція створення кросплатформенного додатку

У сучасному світі інформаційних технологій, що швидко розвиваються, споживачі інформації стикаються з постійно зростаючим потоком новин та подій. Особливо це стосується сфери інформаційних технологій (ІТ), де кожного дня з'являються нові розробки, технологічні досягнення та інноваційні рішення. У такому контексті актуальність розробки кросплатформенного додатку для новинного месенджера, який висвітлює актуальні новини та технології ІТ, є надзвичайно високою.

Сучасні користувачі очікують отримувати інформацію миттєво та без зусиль. Традиційні новинні ресурси часто не можуть забезпечити оперативність та зручність, які надає мобільний додаток. Кросплатформенний додаток дозволяє користувачам отримувати новини у будь-який час та в будь-якому місці, що значно підвищує їхню проінформованість та забезпечує можливість своєчасного реагування на зміни у сфері ІТ.

Кросплатформенний додаток може забезпечити високу ступінь персоналізації контенту, що є важливим для користувачів, які спеціалізуються у певних галузях ІТ. Використання алгоритмів машинного навчання та штучного інтелекту дозволяє відстежувати інтереси користувачів та надавати їм найбільш релевантні новини та статті. Це підвищує якість споживаного контенту та задоволеність користувачів.

3.2 Структура проекту

Проект .NET MAUI (Multi-platform App UI) надає розробникам можливість створювати кросплатформенні додатки з використанням єдиної кодової бази. Основна структура проекту MAUI ґрунтується на основі архітектури MVVM (Model-View-ViewModel).

Рисунок 3.1 – Архітектура MVVM

Компонент Model відповідає за управління даними додатку, бізнес-логікою та правилами взаємодії з джерелами даних, такими як бази даних або веб-сервіси. Model містить лише дані та логіку, необхідну для їх обробки.

Компонент View відповідає за візуальне представлення даних користувачеві та взаємодію з ним. View відображає інформацію та забезпечує інтерфейс для користувацького вводу. Представлення це файли XAML у .NET MAUI, що визначають структуру інтерфейсу користувача, елементи керування, стилі та шаблони.

ViewModel виступає як посередник, який абстрагує View від деталей реалізації Model, надаючи дані у формі, зручній для відображення.

3.3 Інструменти розробки

Для створення додатків з використанням архітектурного патерну MVVM часто використовуються такі інструменти, як Visual Studio, на мові програмування C# та платформа .NET MAUI.

Visual Studio є потужним інтегрованим середовищем розробки (IDE), розробленим компанією Microsoft. Це один з найпопулярніших інструментів для розробки програмного забезпечення, який підтримує широкий спектр мов програмування та технологій.

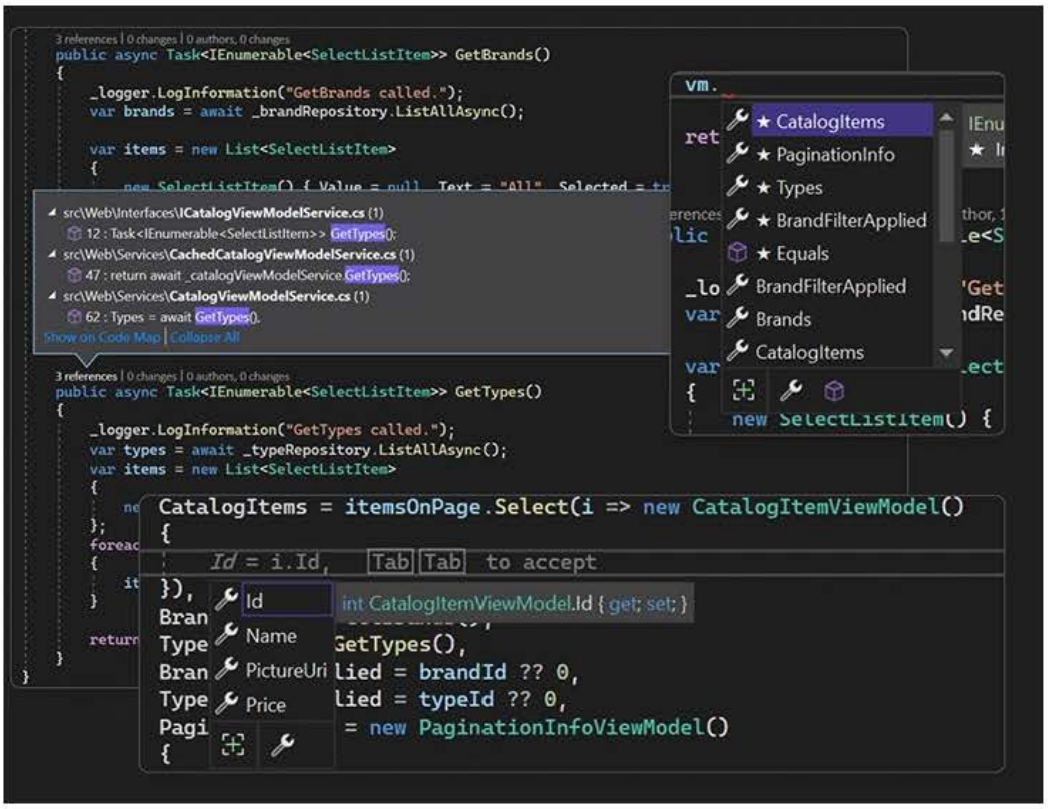


Рисунок 3.2 – Середовище розробки Visual Studio

Visual Studio забезпечує розширені можливості для редагування коду, включаючи підсвічування синтаксису, автозавершення, рефакторинг та інтеграцію з системами контролю версій. Також надає тісну інтеграцію з платформою .NET, що дозволяє легко створювати, збирати та розгортати додатки на основі .NET.

Підтримка таких технологій, як .NET MAUI, Xamarin та інших дозволяє розробникам створювати кросплатформенні додатки для різних операційних систем.

В даному випадку для розробки проекту логічно використовувати мову програмування C#, так як вона вже інтегрована в технології .NET MAUI, Xamarin. Крім інтеграції дана мова програмування має декілька переваг порівняно з іншими мовами. Перш за все коли згадують C#, це те, що вона підтримує об’єктно-орієнтоване програмування (ООП). C# підтримує основні принципи ООП, такі як інкапсуляція, наслідування та поліморфізм, що

дозволяє створювати модульний та код, що може використовуватися повторно.

Рисунок 3.3 – Мова програмування C#

C# підтримує сучасні парадигми програмування, такі як асинхронне програмування, лямбда-вирази та LINQ (Language Integrated Query), що сприяє підвищенню продуктивності розробників [15].

NET MAUI (Multi-platform App UI) - — це сучасна платформа для створення кросплатформених додатків, що дозволяє розробникам додатки для різних платформ Android, iOS, Windows та MacOS. MAUI є наступником технології Xamarin.Forms і забезпечує ще більшу гнучкість та продуктивність [19].

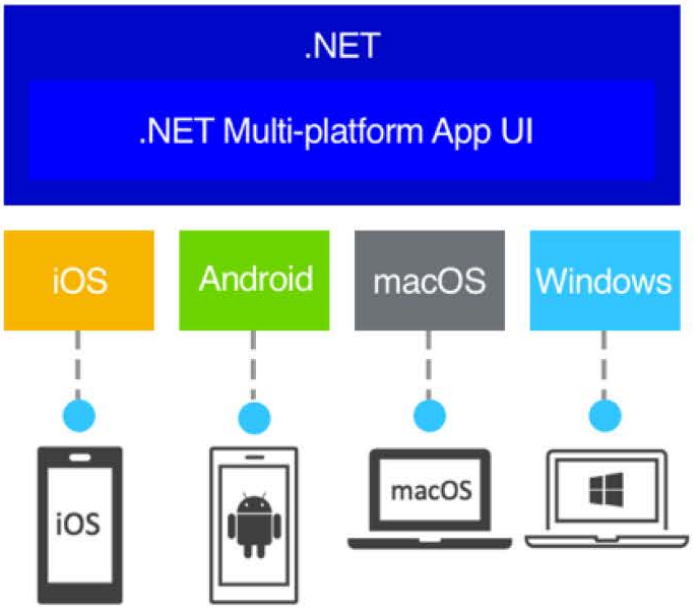


Рисунок 3.4 – NET MAUI

Платформа MAUI глибоко інтегрована з .NET, що надає розробникам доступ до великої кількості бібліотек та інструментів, таких як .NET Standard, Entity Framework та інші. Єдиний проект .NET MAUI спрощує та уніфікує процес кросплатформної розробки, незалежно від цільових операційних систем. Єдиний проект .NET MAUI надає наступні можливості:

1) Проект .NET MAUI підтримує розробку для різних платформ, таких як Android, iOS, macOS і Windows, в межах одного спільного проекту.
2) Розробники можуть легко обирати цільову платформу для налагодження та запуску додатків .NET MAUI завдяки інтегрованим інструментам.
3) Усі файли ресурсів, такі як зображення, шрифти та стилі, зберігаються в одному проекті, що спрощує їх управління та доступ.
4) За потреби розробники можуть використовувати API-інтерфейси та інструменти для конкретних платформ, що забезпечує більшу гнучкість та функціональність.
5) Додаток має одну кросплатформну точку входу, що забезпечує єдине місце для ініціалізації та запуску додатку незалежно від цільової платформи.

В рамках розробки кросплатформених додатків за допомогою .NET MAUI, Android SDK є невід'ємною частиною розробки разом з Android-API та інструментів для створення, налагодження та тестування додатків на платформі Android. SDK включає в себе інструменти для створення та компіляції додатків, а також набір бібліотек, що підтримують взаємодію з апаратними можливостями пристрою, такими як камера, GPS, сенсори та інші. MAUI. Інтеграція Android SDK, дозволяє розробникам створювати інтерфейси користувача, що відповідають стандартам Android, з використанням XAML для визначення UI елементів та C# для логіки додатку (рис. 3.5).

Рисунок 3.5 – Android SDK

Крім API, Android SDK, надає можливість створювати віртуальні машини для зручної демонстрації додатку на різних пристроях.

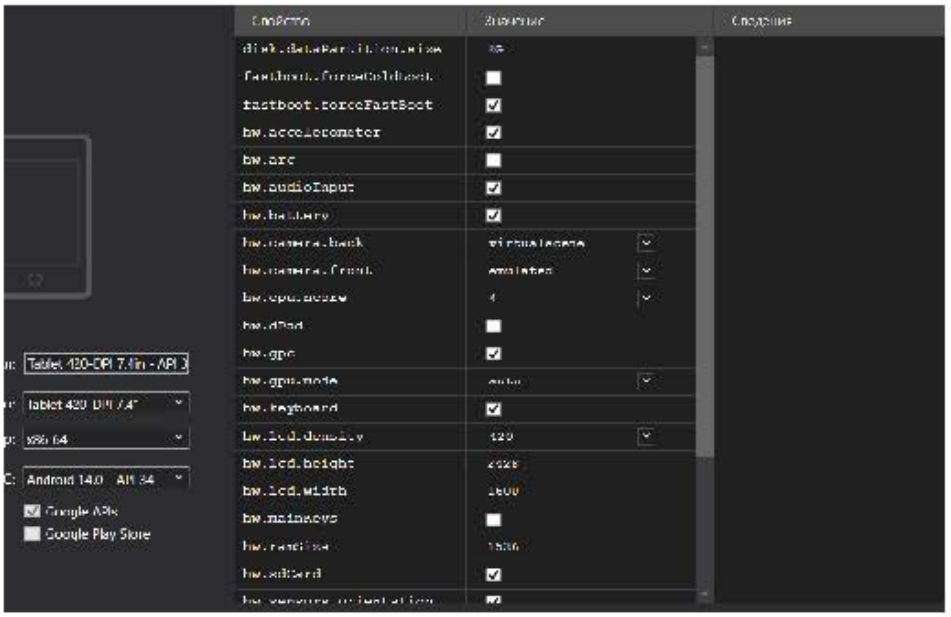


Рисунок 3.6 – Приклад налаштування віртуального пристрою

3.4 Розробка додатку

Наступний крок розглянути основні компоненти програми, розробити структуру MVVM, та віжповідно впроавдити технології для отримання даних з сайту новин.

Під час розробки програма отримала наступну структуру класів [18]:

Рисунок 3.7 – Діаграма класів проекту

Спочатку потрібно розглянути всі класи які пов'язані з архітектурою MVVM, а саме: StoryModel, NewsViewModel, StoryDataTemplate.

Клас StoryDataTemplate є користувацьким шаблоном даних, що успадковується від класу DataTemplate.

```
using static CommunityToolkit.Maui.Markup.GridRowsColumns;

namespace HackerNews;

2 references
public class StoryDataTemplate : DataTemplate
{
    1 reference
    public StoryDataTemplate() : base(CreateGrid)
    {

    }

    1 reference
    static Grid CreateGrid() => new()
    {
        RowSpacing = 1,

        RowDefinitions = Rows.Define(
            (Row.Title, 20),
            (Row.Description, 20),
            (Row.BottomPadding, 1)),

        Children =
        {
            new Label()
                .Row(Row.Title).Top()
                .Font(size: 16).TextColor(ColorConstants.TextCellTextColor)
                .Paddings(10, 0, 10, 0)
                .Bind(Label.TextProperty, static (StoryModel m) => m.Title, mode: BindingMode.OneTime),

            new Label()
                .Row(Row.Description)
                .Font(size: 13).TextColor(ColorConstants.TextCellDetailColor)
                .Bind(Label.TextProperty, static (StoryModel m) => m.Description, mode: BindingMode.OneTime)
        }
    };
```

Рисунок 3.8 – Клас DataTemplate

Цей клас призначений для представлення даних у вигляді сітки (Grid) з певною структурою та стилем. Конструктор класу викликає базовий конструктор, передаючи йому делегат, що створює сітку.

StoryDataTemplate() це конструктор класу, який викликає базовий конструктор з делегатом, що посилається на метод CreateGrid. Це забезпечує автоматичне створення сітки при ініціалізації шаблону.

Рисунок 3.9 – Конструктор класу StoryDataTemplate

Метод CreateGrid є статичним методом, який повертає об'єкт Grid з визначеною структурою рядків та дочірніми елементами. В даному методі використовуються наступні властивості:

1) RowSpacing: Властивість, що визначає відстань між рядками у сітці, встановлена на 1.
2) RowDefinitions: Визначає структуру рядків у сітці, використовуючи метод Rows.Define. Задаються три рядки: Title, Description та BottomPadding з висотами відповідно 20, 20 і 1.
3) Children: Колекція дочірніх елементів, яка містить два елементи Label.

```
1 reference
static Grid CreateGrid() => new()
{
    RowSpacing = 1,

    RowDefinitions = Rows.Define(
        (Row.Title, 20),
        (Row.Description, 20),
        (Row.BottomPadding, 1)),

    Children =
    {
        new Label()
            .Row(Row.Title).Top()
            .Font(size: 16).TextColor(ColorConstants.TextCellTextColor)
            .Paddings(10, 0, 10, 0)
            .Bind(Label.TextProperty, static (StoryModel m) => m.Title, mode: BindingMode.OneTime),

        new Label()
            .Row(Row.Description)
            .Font(size: 13).TextColor(ColorConstants.TextCellDetailColor)
            .Bind(Label.TextProperty, static (StoryModel m) => m.Description, mode: BindingMode.OneTime)
    }
};
```

Рисунок 3.10 – Метод CreateGrid

Клас також містить внутрішнє перелічення Row, яке визначає назви рядків для сітки: Title, Description та BottomPadding. Це перелічення полегшує визначення та використання рядків у сітці.

Клас NewsViewModel успадковується від базового класу BaseViewModel. Цей клас відповідає за управління даними та логікою представлення новин у додатку. Основні функції цього класу включають обробку запитів до сервісів новин, аналіз тексту та управління інтерфейсом користувача за допомогою диспетчера.

```
11 references
partial class NewsViewModel : BaseViewModel
{
    readonly IDispatcher _dispatcher;
    readonly TextAnalysisService _textAnalysisService;
    readonly HackerNewsAPIService _hackerNewsAPIService;

    readonly WeakEventManager _pullToRefreshEventManager = new();

    [ObservableProperty]
    bool _isListRefreshing;

    0 references
    public NewsViewModel(IDispatcher dispatcher,
                         TextAnalysisService textAnalysisService,
                         HackerNewsAPIService hackerNewsAPIService)
    {
        _dispatcher = dispatcher;
        _textAnalysisService = textAnalysisService;
        _hackerNewsAPIService = hackerNewsAPIService;
```

Рисунок 3.11 – Клас NewsViewModel

Конструктор NewsViewModel(IDispatcher dispatcher, TextAnalysisService textAnalysisService, HackerNewsAPIService hackerNewsAPIService) Ініціалізує новий екземпляр класу NewsViewModel з переданими залежностями.

Делегат EventHandler<string> використовується для обробки подій під час невдалого оновлення списку новин (Pull to Refresh). Подія PullToRefreshFailed дозволяє визначати події, які сигналізують про невдале оновлення даних з серверу. Використовується для визначення помилок в обробці запиту. Метод _pullToRefreshEventManager.AddEventHandler та _pullToRefreshEventManager.RemoveEventHandler керують додаванням і видаленням обробників подій відповідно.

```
public event EventHandler<string> PullToRefreshFailed
{
    add => _pullToRefreshEventManager.AddEventHandler(value);
    remove => _pullToRefreshEventManager.RemoveEventHandler(value);
}
```

Рисунок 3.12 – Обробник подій PullToRefreshFailed

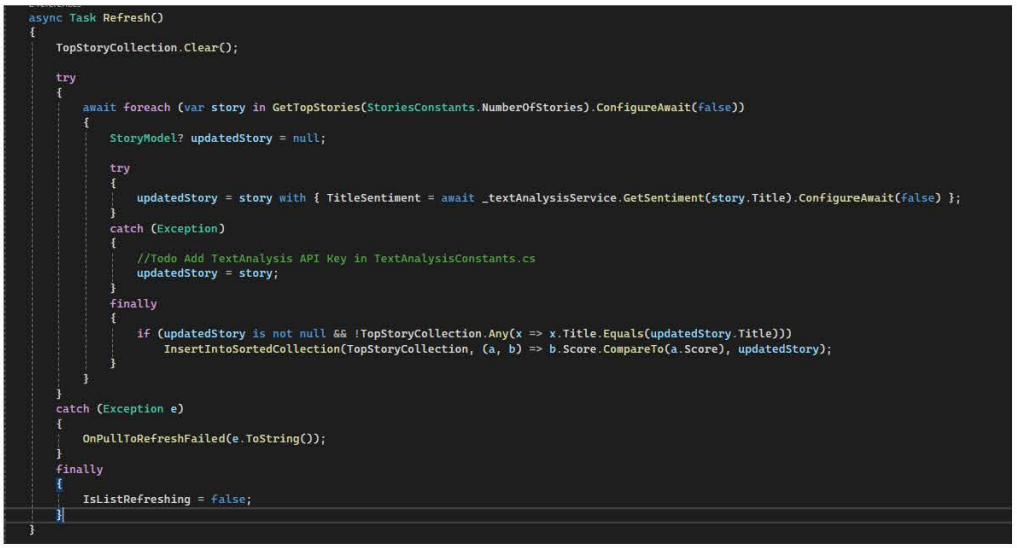
TopStoryCollection – це властивість типу ObservableCollection<StoryModel>, яка зберігає колекцію моделей новин. ObservableCollection забезпечує автоматичне оновлення UI при зміні в колекції. Ця колекція є основним джерелом даних для відображення новин в інтерфейсі користувача.

Метод InsertIntoSortedCollection (рис. 3.13) вставляє елемент у вказану колекцію ObservableCollection, зберігаючи порядок сортування, визначений переданим делегатом Comparison. Спочатку метод перевіряє, чи є колекція порожньою. Якщо так, елемент просто додається до колекції. В іншому випадку, метод проходить по всіх елементах колекції, знаходячи індекс, де новий елемент повинен бути вставлений для збереження порядку. Як тільки знайдено відповідне місце, новий елемент вставляється на цей індекс.

```
6 references
public ObservableCollection<StoryModel> TopStoryCollection { get; } = new();

1 reference
static void InsertIntoSortedCollection<T>(ObservableCollection<T> collection, Comparison<T> comparison, T modelToInsert)
{
    if (collection.Count is 0)
    {
        collection.Add(modelToInsert);
    }
    else
    {
        int index = 0;
        foreach (var model in collection)
        {
            if (comparison(model, modelToInsert) >= 0)
            {
                collection.Insert(index, modelToInsert);
                return;
            }

            index++;
        }

        collection.Insert(index, modelToInsert);
    }
}
```

Рисунок 3.13 – Метод InsertIntoSortedCollection

Метод Refresh (рис. 3.14) є асинхронним методом, який оновлює колекцію новин TopStoryCollection. Він позначений атрибутом [RelayCommand], що дозволяє викликати його як команду з інтерфейсу користувача. Метод починається з очищення колекції новин, видаляючи всі попередні елементи: TopStoryCollection.Clear(). Метод намагається отримати новини за допомогою асинхронного ітератора GetTopStories. Кількість новин визначається константою StoriesConstants.NumberOfStories. Для кожної новини виконується аналіз тексту заголовка за допомогою сервісу _textAnalysisService. Якщо виникає виключення (наприклад, відсутній API ключ), то обробляється новина в оригінальному вигляді.



```
async Task Refresh()
{
    TopStoryCollection.Clear();

    try
    {
        await foreach (var story in GetTopStories(StoriesConstants.NumberOfStories).ConfigureAwait(false))
        {
            StoryModel? updatedStory = null;

            try
            {
                updatedStory = story with { TitleSentiment = await _textAnalysisService.GetSentiment(story.Title).ConfigureAwait(false) };
            }
            catch (Exception)
            {
                //Todo Add TextAnalysis API Key in TextAnalysisConstants.cs
                updatedStory = story;
            }
            finally
            {
                if (updatedStory is not null && !TopStoryCollection.Any(x => x.Title.Equals(updatedStory.Title)))
                    InsertIntoSortedCollection(TopStoryCollection, (a, b) => b.Score.CompareTo(a.Score), updatedStory);
            }
        }
    }
    catch (Exception e)
    {
        OnPullToRefreshFailed(e.ToString());
    }
    finally
    {
        IsListRefreshing = false;
    }
}
```

Рисунок 3.14 – Метод Refresh

Якщо оновлена історія не є порожньою та її заголовок не дублюється в колекції, вона вставляється в колекцію TopStoryCollection у відповідному порядку за допомогою методу InsertIntoSortedCollection.

Функція GetTopStories (рис. 3.15) є асинхронним методом, який повертає IAsyncEnumerable з об'єктів StoryModel. Цей метод дозволяє отримувати топові історії з сервісу HackerNewsAPIService. Спочатку метод отримує список ідентифікаторів новин за допомогою методу GetTopStoryIDs. Потім для кожного ідентифікатора створюється завдання для отримання відповідної новини. Ці завдання додаються до списку getTopStoryTaskList. У циклі while метод очікує завершення будь-якого з завдань за допомогою Task.WhenAny. Після завершення завдання його видаляють зі списку, отримують відповідну новину і повертають її за допомогою ключового слова yield return. Цей процес триває доти, доки в списку є завдання і не досягнуто максимальну кількість історій, вказану параметром storyCount.

```
1 reference
async IAsyncEnumerable<StoryModel> GetTopStories(int? storyCount = int.MaxValue)
{
    var topStoryIds = await _hackerNewsAPIService.GetTopStoryIDs().ConfigureAwait(false);
    var getTopStoryTaskList = topStoryIds.Select(_hackerNewsAPIService.GetStory).ToList();

    while (getTopStoryTaskList.Any() && storyCount-- > 0)
    {
        var completedGetStoryTask = await Task.WhenAny(getTopStoryTaskList).ConfigureAwait(false);
        getTopStoryTaskList.Remove(completedGetStoryTask);

        var story = await completedGetStoryTask.ConfigureAwait(false);
        yield return story;
    }
}
```

Рисунок 3.15 – Функція GetTopStories

Метод ObservableCollectionCallback забезпечує доступ до колекції ObservableCollection. Цей метод використовується для забезпечення коректного доступу до колекції в багатопотокових додатках. Метод приймає параметри collection, context, accessMethod і writeAccess. Основна функція методу полягає в тому, щоб виконати accessMethod в контексті головного потоку за допомогою диспетчера _dispatcher. Це забезпечує коректне виконання операцій з колекцією в багатопотоковому середовищі.

Метод OnPullToRefreshFailed є допоміжним методом, який викликається у випадку невдалого оновлення списку новин (Pull to Refresh). Цей метод приймає повідомлення про помилку як параметр і викликає обробник подій _pullToRefreshEventManager.HandleEvent з переданим повідомленням та назвою події PullToRefreshFailed. Це дозволяє підписаним обробникам подій отримувати повідомлення про помилку та відповідним чином реагувати на неї.

```
1 reference
void ObservableCollectionCallback(IEnumerable collection, object context, Action accessMethod, bool writeAccess)
{
    _dispatcher.Dispatch(accessMethod);
}

1 reference
void OnPullToRefreshFailed(string message) => _pullToRefreshEventManager.HandleEvent(this, message, nameof(PullToRefreshFailed));
```

Рисунок 3.16 – Методи ObservableCollectionCallback, OnPullToRefreshFailed

Клас StoryModel є записом (record) у мові C#, що забезпечує зручну і структуровану модель для представлення новини з сервісу. Клас включає в себе кілька властивостей, методів і обчислюваних властивостей, що забезпечують зручний доступ до даних і їх обробку.

Конструктор класу StoryModel приймає шість параметрів:

1) Id (типу long): унікальний ідентифікатор.
2) By (типу string): ім'я автора.
3) Score (типу long): кількість балів (рейтингу).
4) Time (типу long): час створення.
5) Title (типу string): Назва новини.
6) Url (типу string): URL посилання на новину.

Властивість Description повертає текстове представлення об'єкта, використовуючи метод ToString().

Властивість CreatedAt повертає об'єкт DateTimeOffset, що відповідає за час створення історії, за допомогою методу UnixTimeStampToDateTimeOffset.

```
13 references
public record StoryModel(long Id, string By, long Score, long Time, string Title, string Url)
{
    1 reference
    public string Description => ToString();

    1 reference
    public DateTimeOffset CreatedAt => UnixTimeStampToDateTimeOffset(Time);

    1 reference
    public string TitleSentimentEmoji => TitleSentiment switch
    {
        TextSentiment.Negative => EmojiConstants.SadFaceEmoji,
        TextSentiment.Neutral => EmojiConstants.NeutralFaceEmoji,
        TextSentiment.Mixed => EmojiConstants.NeutralFaceEmoji,
        TextSentiment.Positive => EmojiConstants.HappyFaceEmoji,
        null => string.Empty,
        _ => throw new NotSupportedException()
    };
```

Рисунок 3.17 – Клас StoryModel

Метод ToString перевизначає стандартну поведінку методу для повернення форматованого рядка, що назву новини, кількість балів, ім'я автора та вік новини (як давно була створена).

Метод GetAgeOfStory приймає параметр storyCreatedAt типу DateTimeOffset і повертає рядок, що представляє час, який минув з моменту створення історії, у зручному для користувача форматі. Метод обчислює часовий інтервал між поточним часом (DateTimeOffset.UtcNow) і часом створення історії, зберігаючи результат у змінній timespanSinceStoryCreated. Потім за допомогою оператора switch (з використанням шаблонів), метод визначає відповідний формат рядка в залежності від тривалості інтервалу: якщо інтервал менше години, повертається кількість хвилин; якщо інтервал між однією і двома годинами – повертається "1 година"; якщо інтервал між двома годинами і добою – повертається кількість годин; якщо інтервал між однією і двома добами – повертається "1 день"; якщо інтервал перевищує дві доби – повертається кількість днів. Це забезпечує зручне і зрозуміле відображення часу, який минув з моменту створення історії, для користувачів.

```
1 reference
public override string ToString() => $"{TitleSentimentEmoji} {Score} Points by {By}, {GetAgeOfStory(CreatedAt)} ago";

1 reference
static string GetAgeOfStory(DateTimeOffset storyCreatedAt)
{
    var timespanSinceStoryCreated = DateTimeOffset.UtcNow - storyCreatedAt;

    return timespanSinceStoryCreated switch
    {
        TimeSpan storyAge when storyAge < TimeSpan.FromHours(1) => $"{Math.Ceiling(timespanSinceStoryCreated.TotalMinutes)} minutes",

        TimeSpan storyAge when storyAge >= TimeSpan.FromHours(1) && storyAge < TimeSpan.FromHours(2) => $"{Math.Floor(timespanSinceStoryCreated.TotalHours)} hour",

        TimeSpan storyAge when storyAge >= TimeSpan.FromHours(2) && storyAge < TimeSpan.FromHours(24) => $"{Math.Floor(timespanSinceStoryCreated.TotalHours)} hours",

        TimeSpan storyAge when storyAge >= TimeSpan.FromHours(24) && storyAge < TimeSpan.FromHours(48) => $"{Math.Floor(timespanSinceStoryCreated.TotalDays)} day",

        TimeSpan storyAge when storyAge >= TimeSpan.FromHours(48) => $"{Math.Floor(timespanSinceStoryCreated.TotalDays)} days",

        _ => string.Empty,
    };
}
```

Рисунок 3.18 - Метод GetAgeOfStory

Метод UnixTimeStampToDateTimeOffset забезпечує простий і ефективний спосіб перетворення Unix-міток часу в об'єкти DateTimeOffset, які зручні для використання в .NET додатках. Це перетворення дозволяє працювати з датами і часом у більш зрозумілому і зручному форматі, забезпечуючи сумісність з іншими компонентами системи, що використовують DateTimeOffset.

```
1 reference
static DateTimeOffset UnixTimeStampToDateTimeOffset(long unixTimeStamp)
{
    var dateTimeOffset = new DateTimeOffset(1970, 1, 1, 0, 0, 0, 0, default);
    return dateTimeOffset.AddSeconds(unixTimeStamp);
}
```

Рисунок 3.19 – Метод UnixTimeStampToDateTimeOffset

Далі розглянемо наступний клас NewsPage.

```
class NewsPage : BaseContentPage<NewsViewModel>
{
    readonly IBrowser _browser;
    readonly IDispatcher _dispatcher;

    0 references
    public NewsPage(IBrowser browser,
                    IDispatcher dispatcher,
                    NewsViewModel newsViewModel) : base(newsViewModel, "Новини")
    {
        _browser = browser;
        _dispatcher = dispatcher;

        BindingContext.PullToRefreshFailed += HandlePullToRefreshFailed;

        Content = new RefreshView
        {
            RefreshColor = Colors.Black,

            Content = new CollectionView
            {
                BackgroundColor = Color.FromArgb("#FFFFFF")
```

Рисунок 3.20 – Клас NewsPage

Клас NewsPage є частиною представлення у архітектурному патерні MVVM і відповідає за відображення новин у додатку. Він успадковується від базового класу і використовує NewsViewModel для прив'язки даних та управління подіями. Клас також взаємодіє з диспетчером та браузером для виконання асинхронних операцій та обробки подій.

Конструктор класу NewsPage приймає три параметри: IBrowser для роботи з браузером, IDispatcher для виконання операцій у головному потоці, та NewsViewModel для прив'язки даних.

Конструктор класу NewsPage приймає три параметри: IBrowser для роботи з браузером, IDispatcher для виконання операцій у головному потоці, та NewsViewModel для прив'язки даних. Він викликає базовий конструктор з

переданим newsViewModel та заголовком сторінки "Новини". Поля _browser та _dispatcher ініціалізуються переданими значеннями. У конструкторі налаштовується BindingContext для обробки подій PullToRefreshFailed через метод HandlePullToRefreshFailed. Вміст сторінки задається за допомогою RefreshView, що включає в себе CollectionView для відображення новин з відповідними зпосиланнями до властивостей TopStoryCollection, IsListRefreshing та RefreshCommand з NewsViewModel.

```
0 references
public NewsPage(IBrowser browser,
                IDispatcher dispatcher,
                NewsViewModel newsViewModel) : base(newsViewModel, "Новини")
{
    _browser = browser;
    _dispatcher = dispatcher;

    BindingContext.PullToRefreshFailed += HandlePullToRefreshFailed;

    Content = new RefreshView
    {
        RefreshColor = Colors.Black,

        Content = new CollectionView
        {
            BackgroundColor = Color.FromArgb("F6F6EF"),
            SelectionMode = SelectionMode.Single,
            ItemTemplate = new StoryDataTemplate(),

        }.Bind(CollectionView.ItemsSourceProperty, static (NewsViewModel vm) => vm.TopStoryCollection)
         .Invoke(collectionView => collectionView.SelectionChanged += HandleSelectionChanged)

    }.Bind(RefreshView.IsRefreshingProperty, static (NewsViewModel vm) => vm.IsListRefreshing)
     .Bind(RefreshView.CommandProperty, static (NewsViewModel vm) => vm.RefreshCommand);
}
```

Рисунок 3.21 – Конструктор класу NewsPage

Метод OnAppearing є перевизначенням (override) методу базового класу, що викликається під час відображення сторінки. Цей метод забезпечує ініціалізацію та перевірку стану колекції новин при появі сторінки. Метод починається з виклику базового методу OnAppearing для забезпечення базової логіки відображення сторінки: base.OnAppearing(). Далі перевіряє, чи є вміст сторінки об'єктом RefreshView, який містить CollectionView. Якщо колекція ItemsSource у CollectionView є порожньою або не ініціалізованою (визначається за допомогою методу IsNullOrEmpty), встановлюється властивість IsRefreshing у true для запуску процесу оновлення.

```
0 references
protected override void OnAppearing()
{
    base.OnAppearing();

    if (Content is RefreshView refreshView
        && refreshView.Content is CollectionView collectionView
        && IsNullOrEmpty(collectionView.ItemsSource))
    {
        refreshView.IsRefreshing = true;
    }

    static bool IsNullOrEmpty(in IEnumerable? enumerable) => !enumerable?.GetEnumerator().MoveNext() ?? true;
}
```

Рисунок 3.22 – Метод OnAppearing

Метод HandleSelectionChanged є асинхронним обробником подій, який викликається при зміні вибраного елемента у CollectionView. Основна мета цього методу – обробка вибору новини та відкриття її у браузері. ArgumentNullException.ThrowIfNull(sender) - Метод перевіряє параметр sender на null і встановлює виключення, якщо sender дорівнює null. Якщо перший вибраний елемент є об'єктом StoryModel і має непорожній URL, відкривається веб-браузер з вказаними параметрами. Якщо URL відсутній, відображається попередження про недійсну статтю.

```
1 reference
async void HandleSelectionChanged(object? sender, SelectionChangedEventArgs e)
{
    ArgumentNullException.ThrowIfNull(sender);

    var collectionView = (CollectionView)sender;
    collectionView.SelectedItem = null;

    if (e.CurrentSelection.FirstOrDefault() is StoryModel storyModel)
    {
        if (!string.IsNullOrEmpty(storyModel.Url))
        {
            var browserOptions = new BrowserLaunchOptions
            {
                PreferredControlColor = ColorConstants.BrowserNavigationBarTextColor,
                PreferredToolbarColor = ColorConstants.BrowserNavigationBarBackgroundColor
            };

            await _browser.OpenAsync(storyModel.Url, browserOptions);
        }
        else
        {
            await DisplayAlert("Invalid Article", "ASK HN articles have no url", "OK");
        }
    }
}
```

Рисунок 3.23 – Метод HandleSelectionChanged

Метод HandlePullToRefreshFailed є обробником подій, який викликається у разі невдалого оновлення списку новин.

```
1 reference
void HandlePullToRefreshFailed(object? sender, string message) =>
    _dispatcher.DispatchAsync(() => DisplayAlert("Refresh Failed", message, "OK"));
```

Рисунок 3.24 – Метод HandlePullToRefreshFailed

Клас ShellWithLargeTitles є спеціалізованим класом, який успадковується від ShellRenderer. Він надає користувацькі реалізації для відстеження та рендерингу сторінок у додатку на основі Shell. Клас забезпечує налаштування відображення заголовків у навігаційних елементах. Даний клас має декілька методів CreatePageRendererTracker та CreateShellSectionRenderer.

```
5 references
public class ShellWithLargeTitles : ShellRenderer
{
    0 references
    public static IShellPageRendererTracker? Tracker { get; set; }

    0 references
    protected override IShellPageRendererTracker CreatePageRendererTracker()
    {
        if (Tracker is not null)
            throw new InvalidOperationException("This should have been cleared out by CustomShellSectionRenderer");

        return Tracker = new CustomShellPageRendererTracker(this);
    }

    0 references
    protected override IShellSectionRenderer CreateShellSectionRenderer(ShellSection shellSection)
    {
        return new CustomShellSectionRenderer(this);
    }
}
```

Рисунок 3.25 – Клас ShellWithLargeTitles

Метод CreatePageRendererTracker перевизначає базовий метод для створення відстежувача рендерингу сторінок. Якщо Tracker не дорівнює null, кидається виключення InvalidOperationException з повідомленням про те, що відстежувач повинен був бути очищений попереднім рендерером секцій. Якщо Tracker дорівнює null, створюється новий екземпляр CustomShellPageRendererTracker і присвоюється властивості Tracker.

```
0 references
protected override IShellPageRendererTracker CreatePageRendererTracker()
{
    if (Tracker is not null)
        throw new InvalidOperationException("This should have been cleared out by CustomShellSectionRenderer");

    return Tracker = new CustomShellPageRendererTracker(this);
}
```

Рисунок 3.26 – Метод CreatePageRendererTracker

Метод CreateShellSectionRenderer перевизначає базовий метод для створення рендерера секцій Shell. Повертається новий екземпляр CustomShellSectionRenderer, передаючи поточний об'єкт ShellWithLargeTitles як параметр.

```
0 references
protected override IShellSectionRenderer CreateShellSectionRenderer(ShellSection shellSection)
{
    return new CustomShellSectionRenderer(this);
}
```

Рисунок 3.27 – Метод CreateShellSectionRenderer

Клас CustomShellSectionRootHeader успадковується від ShellSectionRootHeader і забезпечує додаткову логіку для обробки обертання екрана та оновлення даних.

```
3 references
public class CustomShellSectionRootHeader : ShellSectionRootHeader
{
    volatile bool _isRotating;

    1 reference
    public CustomShellSectionRootHeader(IShellContext shellContext) : base(shellContext)
    {
    }

    0 references
    public override void ViewDidLayoutSubviews()
    {
        base.ViewDidLayoutSubviews();

        if (_isRotating)
            Invoke(CollectionView.ReloadData, 0);

        _isRotating = false;
    }


    0 references
    public override void ViewWillTransitionToSize(CGSize toSize, IUIViewControllerTransitionCoordinator coordinator)
    {
        base.ViewWillTransitionToSize(toSize, coordinator);
        _isRotating = true;
    }
}
```

Рисунок 3.28 – Клас CustomShellSectionRootHeader

Клас CustomShellSectionRootRenderer (рис. 3.29) використовується для рендереру секцій, який успадковується від ShellSectionRootRenderer та розширює його функціональність для забезпечення покращеного відображення та обробки інтерфейсу користувача. Клас використовує додаткові відступи безпечної зони (UIEdgeInsets) для коректного розташування елементів у різних умовах, таких як обертання пристрою. Він також інтегрує рендерер заголовків секцій (CustomShellSectionRootHeader) і забезпечує асинхронну обробку відображення дочірніх контролерів. Властивості та методи цього класу налаштовані для підтримки специфічних потреб додатку, включаючи управління контекстом оболонки, взаємодію з рендерером секцій і оновлення представлень.

```
3 references
public class CustomShellSectionRootRenderer : ShellSectionRootRenderer
{
    readonly CustomShellSectionRenderer _customShellSectionRenderer;

    UIEdgeInsets _additionalSafeArea = UIEdgeInsets.Zero;

    1 reference
    public CustomShellSectionRootRenderer(ShellSection shellSection, IShellContext shellContext, Custom
    {
        ShellSection = shellSection;
        _customShellSectionRenderer = customShellSectionRenderer;
    }

    2 references
    public ShellSection ShellSection { get; }

    2 references
    public CustomShellSectionRootHeader? CustomShellSectionRootHeader { get; set; }

    2 references
    IShellSectionController ShellSectionController => ShellSection;

    0 references
    public override void AddChildViewController(UIViewController childController)
    {
        base.AddChildViewController(childController);
        UpdateAdditionalSafeAreaInsets(childController);
    }

    0 references
    public override void ViewDidLayoutSubviews()
    {
        base.ViewDidLayoutSubviews();
        UpdateAdditionalSafeAreaInsets();
```

Рисунок 3.29 – Клас CustomShellSectionRootRenderer

Клас CustomShellSectionRenderer (рис. 3.30) успадковується від ShellSectionRenderer та додає додаткову функціональність для підтримки великих заголовків і відстеження рендерингів сторінок. Клас ініціалізує контекст оболонки (IShellContext) та делегата навігації

(UINavigationControllerDelegate), оновлює заголовки секцій за допомогою методу UpdateLargeTitle та використовує словник для зберігання трекерів сторінок. Він перевизначає методи для обробки змін властивостей оболонки, запитів навігації, додавання нових контролерів. Клас також включає метод SnagTracker для управління трекерами рендерингу сторінок та підтримує специфічні властивості та забезпечує узгоджене відображення та інтерфейс користувача.

```
2 references
class NavDelegate : UINavigationControllerDelegate
{
    readonly UINavigationControllerDelegate _navDelegate;
    readonly CustomShellSectionRenderer _self;

    1 reference
    public NavDelegate(UINavigationControllerDelegate navDelegate, CustomShellSectionRenderer customShellSectionR
    {
        _navDelegate = navDelegate;
        _self = customShellSectionRenderer;
    }

    // This is currently working around a Mono Interpreter bug
    // if you remove this code please verify that hot restart still works
    // https://github.com/xamarin/Xamarin.Forms/issues/10519
    [Export("navigationController:animationControllerForOperation:fromViewController:toViewController:")]
    [Foundation.Preserve(Conditional = true)]
    0 references
    public new IUIViewControllerAnimatedTransitioning? GetAnimationControllerForOperation(UINavigationController
    {
        return null;
    }

    0 references
    public override void DidShowViewController(UINavigationController navigationController, [Transient] UIViewCon
    {
        _navDelegate.DidShowViewController(navigationController, viewController, animated);
    }

    0 references
    public override void WillShowViewController(UINavigationController navigationController, [Transient] UIViewCo
    {
        _navDelegate.WillShowViewController(navigationController, viewController, animated);
```

Рисунок 3.30 – Клас CustomShellSectionRenderer

Клас ShellAttachedProperties (рис. 3.31) є статичним класом, що містить прикріплені властивості для керування великими заголовками у додатках на основі Shell. Цей клас забезпечує зручний спосіб встановлення та отримання значення властивості PrefersLargeTitles для елементів BindableObject.

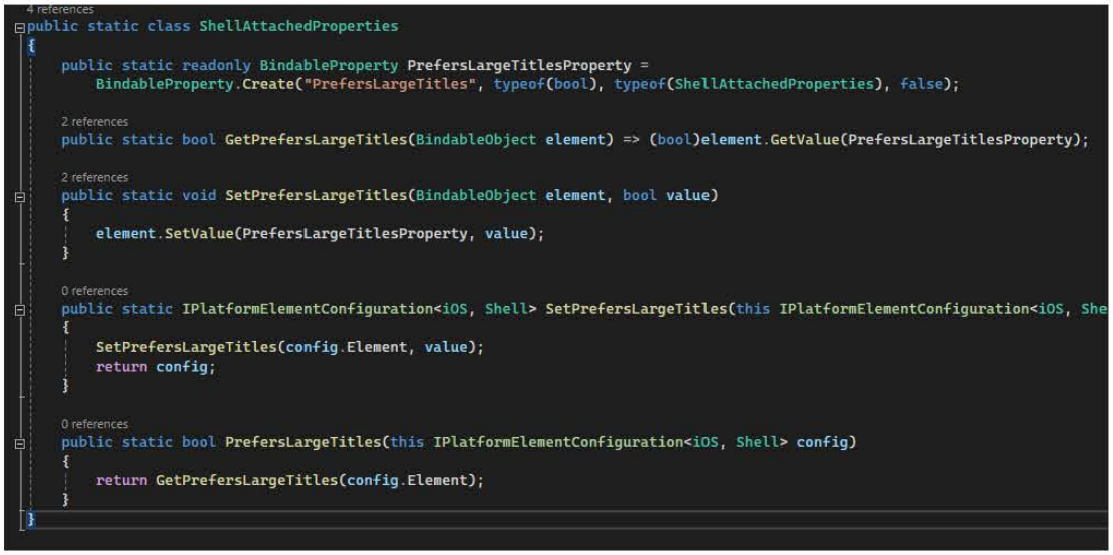


Рисунок 3.31 – Клас ShellAttachedProperties

Метод SetPrefersLargeTitles використовується для встановлення значення властивості PrefersLargeTitles для переданого елемента BindableObject (рис. 3.30).

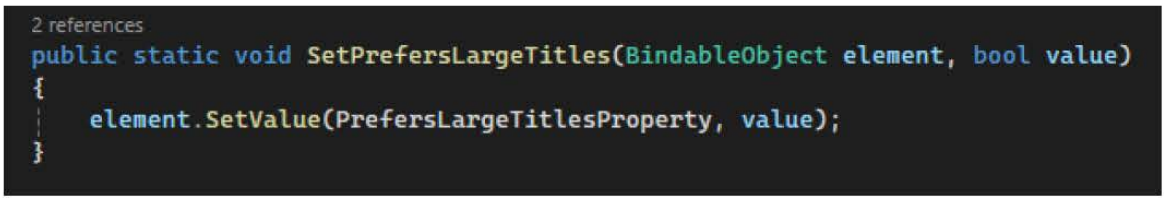


Рисунок 3.32 – Метод SetPrefersLargeTitles

SetPrefersLargeTitles (розширений метод для IPlatformElementConfiguration) - метод для встановлення значення властивості PrefersLargeTitles для конфігурації платформи iOS на основі Shell (рис. 3.31).

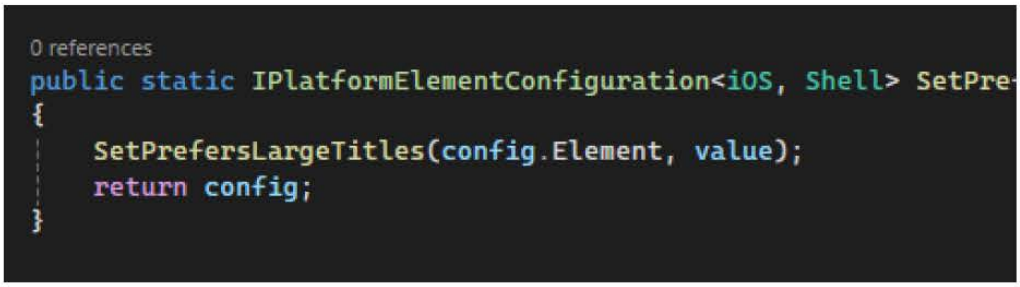


Рисунок 3.33 – Метод SetPrefersLargeTitles

Клас App (рис. 3.34) є головним класом додатку, який успадковується від Application. Він відповідає за ініціалізацію додатку, налаштування глобальних ресурсів та встановлення головної сторінки додатку. Конструктор класу App приймає один параметр AppShell (екземпляр оболонки додатку), ініціалізує словник ресурсів та встановлює головну сторінку додатку.

```
3 references
class App : Application
{
    0 references
    public App(AppShell appshell)
    {
        Resources = new ResourceDictionary()
        {
            new Style<Shell>(
                (Shell.NavBarHasShadowProperty, true),
                (Shell.TitleColorProperty, ColorConstants.NavigationBarTextColor),
                (Shell.DisabledColorProperty, ColorConstants.NavigationBarTextColor),
                (Shell.UnselectedColorProperty, ColorConstants.NavigationBarTextColor),
                (Shell.ForegroundColorProperty, ColorConstants.NavigationBarTextColor),
                (Shell.BackgroundColorProperty, ColorConstants.NavigationBarBackgroundColor)).ApplyToDerivedTypes(true),

            new Style<NavigationPage>(
                (NavigationPage.BarTextColorProperty, ColorConstants.NavigationBarTextColor),
                (NavigationPage.BarBackgroundColorProperty, ColorConstants.NavigationBarBackgroundColor)).ApplyToDerivedTypes(true)
        };

        MainPage = appshell;
    }
}
```

Рисунок 3.34 – Клас App

Клас ColorConstants є статичним класом, що містить константи кольорів, які використовуються у різних частинах додатку для забезпечення узгодженого стилю. Всі кольори визначені як властивості типу Color і призначені для використання у навігаційних елементах, текстових полях.

```
public static class ColorConstants
{
    2 references
    public static Color NavigationBarBackgroundColor { get; } = Color.FromArgb("008000");
    5 references
    public static Color NavigationBarTextColor { get; } = Colors.Black;

    1 reference
    public static Color TextCellDetailColor { get; } = Color.FromArgb("828282");
    1 reference
    public static Color TextCellTextColor { get; } = Colors.Black;

    1 reference
    public static Color BrowserNavigationBarBackgroundColor { get; } = Color.FromArgb("00FF00");
    1 reference
    public static Color BrowserNavigationBarTextColor { get; } = Color.FromArgb("3F3F3F");
}
```

Рисунок 3.35 – Клас ColorConstants

3.5 Тестування додатку

Щоб впенитися роботі даного проекту, розробнику потрібно провести тестування всього функціоналу додатку.

Отже спочатку потрібно завантажити додаток:

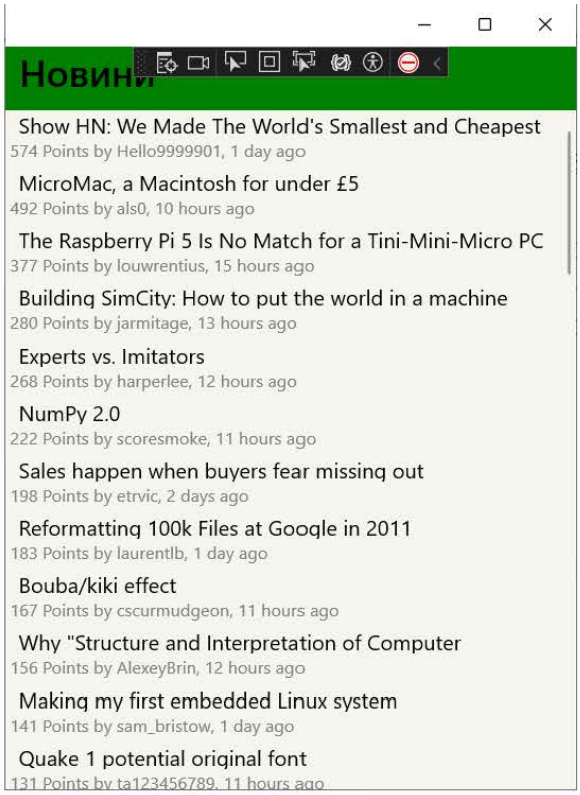


Рисунок 3.36 - Відкриття додатку

Користувач має отримати додаток в якому він спостерігає стрічку з новин. Кожна новина супроводжується її назвою, час коли вона була завантажена та кількість балів, про що свідчить популярність даної новини та зацікавленість користувачів в даній статті:

Рисунок 3.37 – Зображення новини

Під час перегляду користувач може прогорнути в низ за допомогою слайдера:

Рисунок 3.38 – Елемент інтерфейсу Slider

Під час прокрутки слайдеру, користувач може спостерігати як новини з часом додаються та оновлюються (рис. 3.39):

Рисунок 3.37 – Перегляд новин

Останнім кроком це перевірка посилання на новину стрічку. Для цього користувачу потрібно обрати потрібну новину (рис. 3.38):

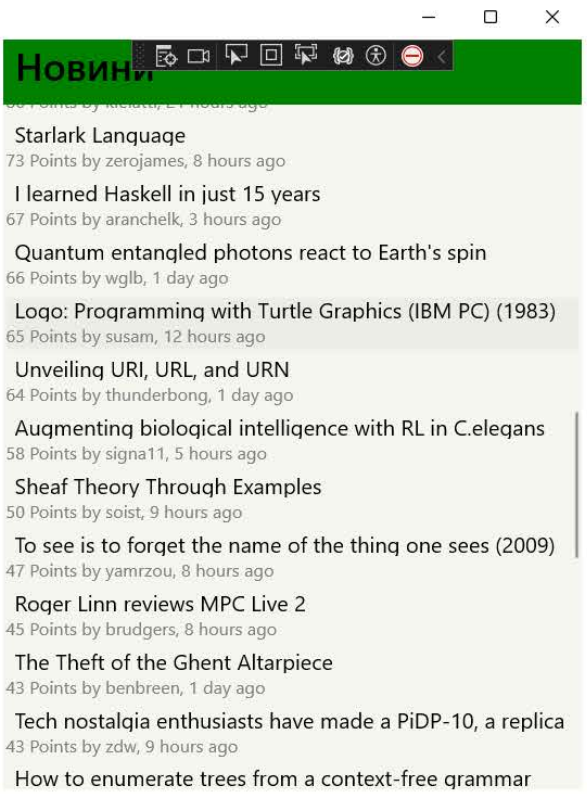


Рисунок 3.38 – Вибір новини

Далі потрібно клацнути на елемент інтерфейсу, що відрити браузер з новиною (рис. 3.39):

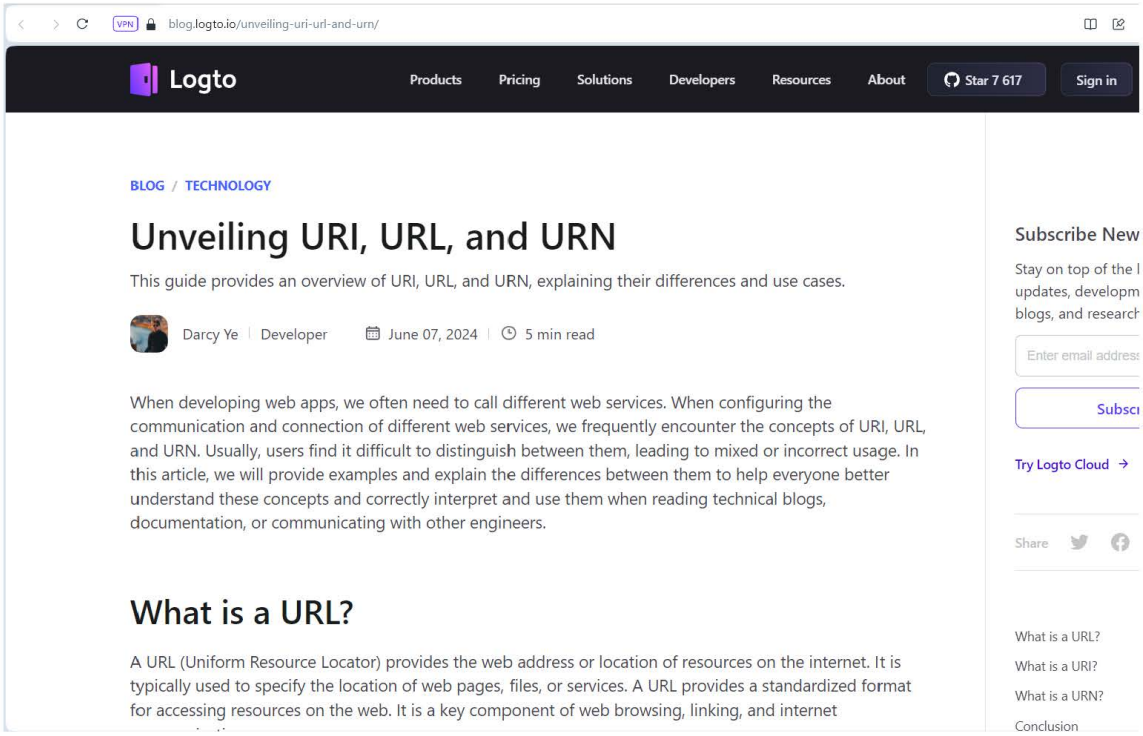


Рисунок 3.39 – Відкриття новини в браузері

В результаті назва новини в програмі та на сайті статті повністю співпадають. Тобто програма цілком виконує властні функції

Також дана програма має широкий простір для вдосконалення. Наприклад додати елементи персоналізації: тобто додавати можливість змінювати колір фону, або додати власний профіль

3.6 Висновок до третього розділу

В результаті практичної частини проекту було успішно розроблено крос-платформенний додаток, який демонструє високу функціональність та зручність у використанні. Використання .NET MAUI як основної платформи для розробки дозволило ефективно об'єднати кодову базу, зменшити витрати на розробку та підтримку, а також забезпечити узгоджений користувацький досвід незалежно від пристрою. Інтеграція з різноманітними API та службами, такими як Android SDK, забезпечила доступ до сучасних можливостей апаратного забезпечення та специфічних функцій платформи. Ретельне

опрацювання інтерфейсу користувача, врахування особливостей кожної платформи та реалізація інтуїтивно зрозумілої навігації сприяли створенню додатку

Таким чином, розроблений крос-платформенний додаток не лише виконує поставлені завдання, але й демонструє потенціал для подальшого розширення та вдосконалення, відкриваючи нові можливості для користувачів та розробників.

ВИСНОВОК

Кваліфікаційна робота присвячена розробці кросплатформного додатку IT новин, що є актуальним завданням у контексті сучасного розвитку мобільних технологій. Метою роботи було створення зручного та функціонального інструменту для автоматизації збору та подання новин IT-сфери, що забезпечить користувачам доступ до актуальної інформації незалежно від використовуваної операційної системи.

У першому розділі роботи було проведено детальний аналіз предметної області, зокрема дослідження сучасних методів та підходів до розробки кросплатформних додатків. Особливу увагу було приділено аналізу існуючих рішень, що дозволило визначити ключові тенденції та технології, які використовуються для створення таких додатків.

Проаналізовано методи та підходи до розробки крос-платформних додатків, такі як гібридні додатки, нативні скриптові фреймворки та прогресивні веб-додатки (PWA). Детально розглянуто інструменти для кожного підходу, їх переваги та недоліки, а також особливості їх використання [14].

Було встановлено, що вибір правильних інструментів та технологій є критичним для успішної реалізації проекту, оскільки це забезпечує гнучкість та продуктивність розробки.

У другому розділі роботи було обрано оптимальні програмні засоби для реалізації проекту. Сьогодні серед найпопулярніших фреймворків для створення мобільних додатків можна виділити Kotlin Multiplatform Mobile, React Native, Flutter та Xamarin/MAUI. Для порівняння фреймворків було обрано такі критерії, як продуктивність роботи, наявність відкритого вихідного коду, спосіб побудови інтерфейсу користувача (UI), зрілість фреймворку (термін його існування), наявність великої спільноти користувачів, документація, підтримувані мови програмування та вартість використання.

Для розробки додатку було обрано фреймворк .NET MAUI, мову програмування C# та середовище розробки Visual Studio Community 2022. Використання .NET MAUI забезпечило потужні можливості для створення кросплатформних додатків з єдиною кодовою базою, що значно зменшило витрати на розробку та підтримку.

У третьому розділі було детально описано процес розробки додатку для IT новин. Реалізовано основні компоненти, такі як архітектура MVVM, алгоритми обробки даних та інтеграція з API новин. Було розроблено та протестовано ключові елементи додатку, включаючи відображення новин, взаємодію користувачів з інтерфейсом та обробку запитів до серверу, що забезпечило плавність та зручність користування додатком.

Проведене тестування проекту підтвердило його функціональність та відповідність вимогам. Результати тестування показали, що додаток ефективно обробляє дані, забезпечує надійне зберігання інформації та пропонує користувачам зручний спосіб отримання новин IT-сфери.

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

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. Why You Should Consider Kotlin Multiplatform. URL: https://x-team.com/blog/kotlin-multiplatform/

12. KMM: common code for iOS and Android, DOU. URL: https://medium.com/globant/kotlin-multiplatform-mobilekmm-code-sharing-between-android-and-ios-9a9af66e2655

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

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

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

16. MVC Framework Introduction – [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/mvc-framework-introduction/

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

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