math_2020_034

Міністерство освіти і науки України

Чернівецький національний університет

імені Юрія Федьковича

Факультет математики та інформатики
(повна назва інституту/факультету)

Кафедра математичного моделювання
(повна назва кафедри)

Створення мобільного додатку для моніторингу розкладу сеансів в кінотеатрі з використанням Xamarin технологій

Дипломна робота

Рівень вищої освіти - другий (магістерський)

Виконав (ла):
студент (ка) 6 курсу, групи 607
спеціальності 124 – Системний аналіз
(назва спеціальності)
__Миглей_Валентин_Костянтинович__
(прізвище, ім'я та по-батькові)
Керівник к.ф.-м.н., доцент Дорошенко І.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

До захисту допущено:

Протокол засідання кафедри № 6

від „3" грудня 2019 р.

зав. кафедри _____ доц. Піддубна Л.А.

Чернівці – 2019

АНОТАЦІЯ

Магістерська робота присвячена створенню мобільного крос-платформного додатку з використанням фреймворка Xamarin на мові програмування C#. Результатом роботи став додаток для моніторингу сеансів кінотеатрів, перегляд новини кіноіндустрії. Використані базові функції Xamarin фреймворка та його сервіси.

ЗМІСТ

ВСТУП

Xamarin – це унікальний фреймворк який дозволяє будувати додатки з використанням однієї мови програмування – C#. З використанням данного фреймворка компілюється нативний код для операційних систем ОС, Android і WP.

Кожна з цих ОС має свій власний набір можливостей і кожна різниться в здатності писати нативні додатки котрі компілюються в машинний код. Наприклад, ОС Android дозволяє писати код на Java, Windows Phone (WP) – з використанням JavaScript та C#, iOS – Objective-C та Swift. Звичайно є й інші фреймворки, які дозволяють писати додатки на не нативних для платформи мовах програмування, наприклад Corona SDK, PhoneGap та інші, проте при використанні таких фреймворків швидкодія роботи додатку значно знижується, ніж в тих додатках, що використовують нативні засоби розробки.

В першому розділі наведено та описано основні можливості фреймворку Xamarin. Також буде показана внутрішня робота Xamarin та основні способи побудови додатків з використанням Xamarin – Xamarin.iOS, Xamarin.Android та Xamarin.Forms. Зроблено огляд основних компонентів даних технологій, наприклад, activity в Xamarin.Android та page в Xamarin.Forms. Детально розглянуто основні поняття та принципи даних підходів в розробці програмного забезпечення, використовуючи фреймворк Xamarin та мову C#. Також будуть описані деякі програми для полегшення розробки програмного продукту.

В другому розділі описані основні етапи розробки програмного продукту, шляхи та методи їх побудови. Також показані основні частини функціоналу додатку та результат у виглядв скрішотів.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

КПР Крос-платформна розробка

КП Крос-платформність

ПП Програмний продукт

1. ОПИС ТЕХНОЛОГІЙ

1.1. Крос-платформна розробка

1.1.1 Загальне про крос-платформної розробки

Для того щоб написати програми під кожну з ОС окремо потрібно витратити досить багато ресурсів і багато часу. Будь-яка ОС має свої різні API інтерфейси а отже й специфічну логіку роботи і нативну мову програмування. Для створення програмного продукту (ПП) який водночас буде доступний для різних ОС компанії яка займається розробкою ПП потрібно тримати цілий штат програмістів які спеціалізуються для різноманітних ОС, наприклад, команду спеціалістів Java розробників для написання Android програм, команду C# програмістів для написання програми під Windows Phone і т.д. По даній причині існують фреймворки, які дозволяють позбутися від такої залежності і використовувати лише одну мову програмування для того щоб писати код під різні ОС.

Даний підхід має і свої мінуси також. Першим з них є те, що програми для даної ОС які написані на нативній мові програмі функціонують набагато швидше. Другий мінус є те, що потрібно знайти таких спеціалістів, які знають усі платформи, які підтримує даний фреймворк, звісно спеціаліст, який спеціалізується на одній платформі буде знати саме цю платформу набагато краще.

Проте такий підхід має і свої переваги. Одна з найважливіших переваг є те що програмістам не треба міняти бізнес логіку ПП, тому що код пишуть на з використанням однієї мови програмування, звісно, не весь код буде спільним, проте за статистикою (яка надана спеціалістами з Xamarin) такий код займає в середньому 20% від усього програмного коду (рис. 1.1):

Рисунок 1.1. Діаграми відношення коду до специфічної платформи[1]

Ще однією перевагою даного підходу є те, що час, на створення продукту дуже сильно скорочується, оскільки потрібно писати менше коду а це призводить до того що скорочується й кількість конфліктів між командами.

В даному розділі розглядатимуться основні поняття крос-платформної розробки, а також – буде проведено розгляд уже існуючих рішень для крос-платформної розробки.

1.1.2 Поняття крос-платформної розробки

Крос-платформна розробкка це практика розробки програмного продукту або сервісу для різноманітних платформ або середовищ програмного забезпечення. Інженери або розробники використовують різні методи щоб підлаштувати різні операційні системи або середовища для одного додатку або продукту.

Ідея крос-платформної розробки полягає в тому щоб програмний додаток або продукт повинен працювати добре на більше ніж одному специфічному цифровому середовищу. Ця можливість зазвичай використовується для продажу програмного забезпечення для більш ніж однієї специфічної операційної системи, наприклад, для використання на платформах Microsoft та

Apple. З розвитком мобільних пристроїв та інших видів платформ, а також з поширенням технологій з відкритим кодом, таких як Linux, з'явилося все більше видів кросплатформних розробок.

Деякі основні стратегії розвитку rhjc-платформної розробки включають в собі складання різних версій однієї програми для різних операційних систем, або в інших випадках використання файлів різних під-дерев для застосування або пристосування продукту до різних операційних систем. Ще одним важливим підходом є зробити програму абстрактною на певних рівнях, щоб пристосувати різні програмні середовища. Програмне забезпечення подібне цьому можна назвати "агностичною платформою", оскільки воно не цінує і не підтримує одну платформу над іншою. Розробники також можуть використовувати інтерфейси прикладного програмування (API) для налаштування частини програмного забезпечення для певної платформи.

Розглянемо основні недоліки та переваги крос-платформності додатків:

1. Час розробки — відсутність унікальних елементів інтерфейсу і одна технологічна платформа скорочує терміни розробки програмного продукту;
2. Підтримка і оновлення програмного продукту — додавання функціональної частини або виправлення помилок одночасно для всіх платформ;
3. Економія бюджету — використання однієї технології і графічного інтерфейсу дуже сильно знижує кількість робочих годин і бюджет проекту;
4. Спільна логіка проекту — логіка додатку однаково працюватиме для всіх платформ;
5. Досить повільніша робота додатку;
6. Неможливість спільного використання унікальних особливостей платформи;
7. Незвичний для користувача інтерфейс.

Загалом, крос-платформна розробка може зробити програму менш ефективною. Наприклад, вона може вимагати надмірних процесів або папок зберігання файлів для різних систем, які вона повинна підтримувати. Також може знадобитися щоб програма була "скинута" для розміщення менш складних програмних середовищ. Однак у багатьох випадках виробники

програмного забезпечення з'ясували, що з обмеженнями кросплатформної розробки варто боротися, щоб запропонувати додаток або продукт більш широкому набору користувачів.

1.2 Технології Xamarin

Xamarin – це компанія, яка була заснована 2011 році інженерами, які створили Mono, Mono for Android та MonoTouch, що є реалізаціями Common Language Infrastructure (CLI) та Common Language Specifications(яку досить часто називають Microsoft.NET). Історія заснування та розвидку:

На початку XXI століття Microsoft вперше оголосила про створення .NET Framework – спеціального фреймворку, який дозволяв писати програми під операційну систему Windows, використовуючи різноманітні мови програмування. Мігель Де Іказа з компанії Ximian підняв питання про реалізацію даного фреймворку для платформи Linux. Ця ідея розвивалась у вигляді проекту з відкритим кодом, що називається Mono, який був запущений 19 липня 2001 року. Однак, дана ідея хоч і мала розвиток, проте через декілька років, все жтаки розробку Mono було призупинено.

У 2011 році Мігель Де Іказа анонсував в своєму блозі про те, що Mono буде розвиватися далі і підтримуватися компанією Xamarin, яка була створена у тому ж році і основним напрямком розробки даної компанії буде розробка під мобільні пристрої. Він також повідомив про те, що велика частина людей, яка працювала над Mono перейшла до нової компанії.

Наступного року, компанією Xamarin було випущеноXamarin.Mac – плагін для IDE MonoDevelop, який дозволяв розробникам писати додатки на мові програмування C# для операційної системи Apple OS X і публікувати їх в Apple App Store.

Через деякий час було анонсовано створення нової версії Xamarin 2.0. Даний реліз включав в собі два основних компоненти – Xamarin Studio, що заміняла MonoDevelop та плагін для інтеграції у Visual Studio – IDE від Microsoft

для .Net Framework розробників. За допомогою цьому C# розробники змогли створити свої додатки для iOS, Android схожими Windows та Windows Phone(WP).

У 2016 році компанії Xamarin та Microsoft анонсували що Microsoft купили всі права на компанію Xamarin, хоча не називають точну цифру за яку було придбано Xamarin, в одному з журналів припустили, що приблизна сума десь в межах 400 - 500 мільйонів доларів. Поганими новинами стало те, що Xamarin колись придбала усі права на RoboVM, що є аналогічним фреймворком для Xamarin.Forms для Java. Однак, після того як компанія Microsoft придбала Xamarin, Microsoft анонсували, що не будуть в подальшому підтримувати даний продукт.

На конференції Build 2016 було анонсовано те, що Xamarin стає повністю безкоштовним фреймворком який підтримує Microsoft а також буде повністю реліцензійовано Mono.

1.2.1 Основні продукти Xamarin

Основними продуктами, які були створені за цей період часу є:

1. Xamarin Platform Платформа, що дозволяє використовувати нативні API конкретної операційної системи для написання додатків[1].
2. Xamarin.Forms Як вже зазначалося, аналог Ionic, але з використанням C# та мови XAML замість AngularJS для побудови UI[1].
3. Xamarin Test Cloud 26 Даний сервіс дозволяє розробнику протестувати свій додаток на різних телефонах, не маючи його. Достатньо лише мати підключення до інтернету і підписку(дана функція є платною), щоб тестувати свої додатки у великій базі смартфонів[1].
4. Xamarin for Visual Studio Підтримка Xamarin у Visual Studio дозволяє розробнику писати додатки на iOS та Android не використовуючи Mac та Xamarin Studio. Хоча для того, щоб розробляти iOS додатки, все одно потрібно мати підключення до OS X пристрою[1].
5. Xamarin.Mac Фреймворк, що дозволяє писати додатки для Mac[1].

6. Xamarin Studio Основна IDE, що використовується для написання додатків під iOS чи Mac з використанням Xamarin[1].
7. .Net Mobility Scanner Спеціальний сервіс, що дозволяє просканувати бібліотеку, написану на C#, щоб дізнатися, чи підходить вона для додатків на Xamarin.iOS, Xamarin.Android і т.д[1].

1.2.2 Платформа Mono

Mono – платформа розробки з відкритим кодом на основі платформи .NET Framework, дозволяє розробникам розробляти крос-платформні додатки з покращеною продуктивністю розробника. Реалізація Mono .NET заснована на стандартах ECMA для C # і Common Language Infrastructure(CLI)[2].

Даний продукт раніше підтримували такі компанії як Novell, Xamarin, а тепер Microsoft і .NET Foundation. Mono включає в собі як засоби розробки так і інфраструктуру, необхідну для запуску серверних та клієнтських додатків .NET[2].

Основні компоненти Mono:

1. Компілятор C#

Компілятор C# від Mono повністю підтримує усі стандарти C#.

2. Mono runtime

Це середовище виконання яке реалізує ECMA стандарт CLI, надає Just-In-Time компілятор та Ahead-of-Time компілятор, завантажувач бібліотек, збирач сміття (garbage collector), а також систему обробки потоків.

3. Бібліотека класів .NET

Платформа Mono надає повний набір класів, які забезпечують міцну основу для створення додатків. Ці класи сумісні з класами Microsoft .NET Framework.

4. Бібліотека класів Mono

Mono також надає безліч класів, яких немає в бібліотеці базових класів, наданою корпорацією Майкрософт. Вони забезпечують додаткові функціональні можливості, які є корисними, особливо в створенні додатків для

Linux. Деякі приклади класів для GTK +, Zipфайлів, LDAP, OpenGL, Cairo, POSIX і т.д[2].

Основна ідея створення Mono є запуск .NET програм на інших операційних системах, наприклад Linux, OS X і т.д, так як це неможливим з використанням CLR.

1.3 Опис Xamarin.Android та її основні можливості

Xamarin.Android додатки виконуються в середовищі виконання Mono. Mono працює разом з віртуальною машиною Android Runtime (ART). Обидва середовища виконання працюють над ядром Linux і надають різні інтерфейси API для коду програміста, що дозволяє розробникам отримати доступ до даної системи[3].

Розробникам надані простори імен System, System.IO, System.Net та інші бібліотеки класів .NET, для отримання доступу до основних можливостей операційної системи Linux.

На операційні системі Android, більшість системних можливостей, таких як програвання аудіо файлів, графіки, OpenGL і телефонії не доступні безпосередньо до нативних додатків, вони доступні тільки через Android Java Runtime API, що були реалізовані в 30 просторі імен Java. * або Android. *[3]. Архітектура роботи приблизно така (рис. 1.2):

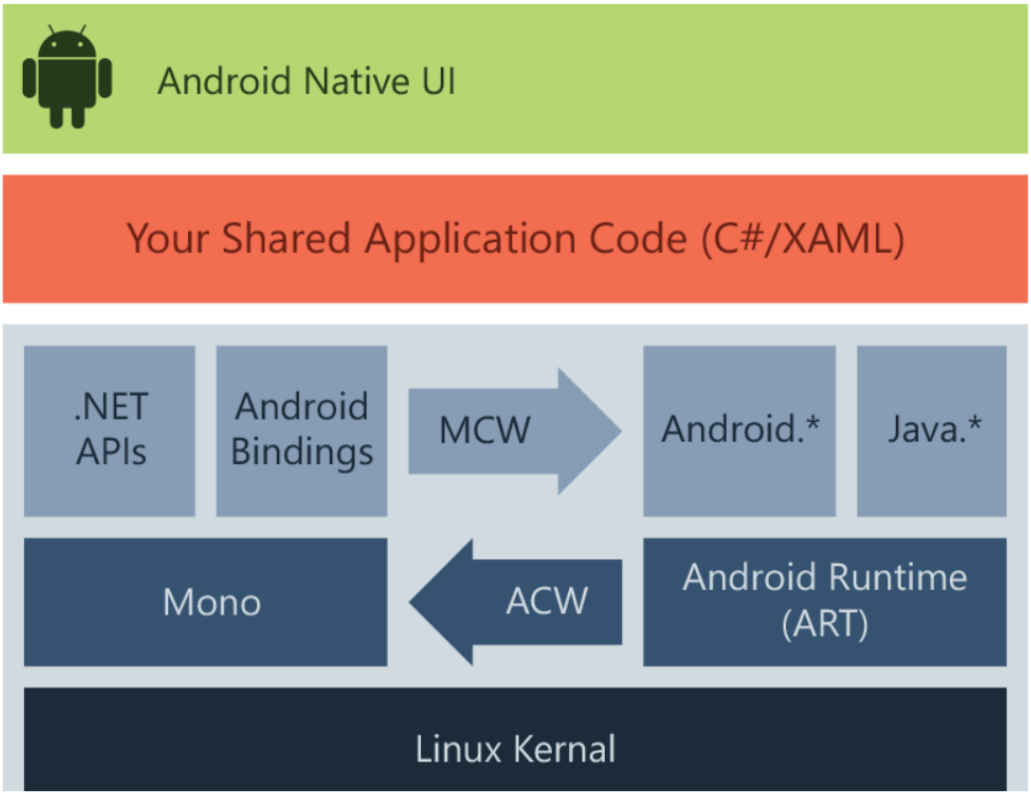


Рисунок 1.2.– Принцип роботи Xamarin.Android[4]

Як видно, в самому центрі розташоване ядро Linux, над яким знаходяться два СВ – Mono та ART, що взаємодіють між собою за допомогою Android Callable Wrappers(ACW). Вже над даними СВ розташовані .NET API та спеціальні прив‘язки до Android.* та Java.* можливостей, що взаємодіють з використанням Managed Callable Wrappers(MCW)[3].

Android Callable Wrappers – спеціальний Java-Native-Interface(JNI) —міст, що використовується ОС Android для виконання коду.

Manager Callable Wrappers – спеціальні обгортки типу JNI для того, щоб виконувати Android код, що був створений з використанням ACW. Кожна така обгортка зберігає у собі глобальне Java посилання(reference) на об‘єкт. Кількість таких об‘єктів є лімітованою. Загалом доступно 52.000 таких посилань під час роботи додатку[3].

Оскільки Xamarin додатки компілюються в нативні додатки, то програмістам доступні усі керуючі об’єкти (controls), що використовуються при розробці додатків з використанням Java – Popup Menus, Views, Date Pickers тощо.

Також схожі додатки можуть використовувати усі можливості API Android – Android Beam, камера, аутентифікація з використанням відбитку пальця, карти, локація, Android.Speech та інші[3].

Кожен додаток Android використовує спеціальні об‘єкти які називаються activity. Наприклад, кожна сторінка додатку на Android і є об’єктом activity, а кожен такий об’єкт має свій цикл існування(lifecycle). Xamarin.Android також підтримує дану концепцію, яка дозволяє описувати Activity з використанням C#. Усі activity в Xamarin.Android будуються спеціальними атрибутами, які дозволяють додавати дані activity до спеціального Android маніфеста, де описуються та додаються основні компоненти додатку. Розглянемо цикл роботи activity[3] (рис. 1.3):

1. OnCreate це метод який виконується при включенні додатку. В данному методі, можна отримувати збережені дані[3].
2. Після OnCreate виконується метод OnStart, що необхідний для встановлення значень для View – тобто ініціалізація користувацького інтерфейсу[3].
3. Метод OnResume викликається завжди після методу OnStart. В даному методі можна ініціалізувати камеру, чи інші ресурси, необхідні для виконання додатку[3].
4. OnPause – метод, що викликається під час того, коли система поставила дану activity в фоновий потік чи дане activity було перекрито іншою activity. В даному методі краще за все зберегти усі необхідні дані та почистити усі ресурси, що використовує додаток[3].
5. OnStop – метод, який викликається, коли користувач вже не бачить Activity. Для того щоб чистити чи зберігати усі ресурси в методі OnPause краще не використовувати метод OnStop[3].
6. OnDestroy – даний метод викликається, коли activity видаляється з пам‘яті пристрою, в деяких ситуаціях ОС Android не викликає цей метод, тому він не є безпечним[3].
7. OnRestart – викликається після OnStop, якщо користувач знову ввімкнув дане Activity[3].

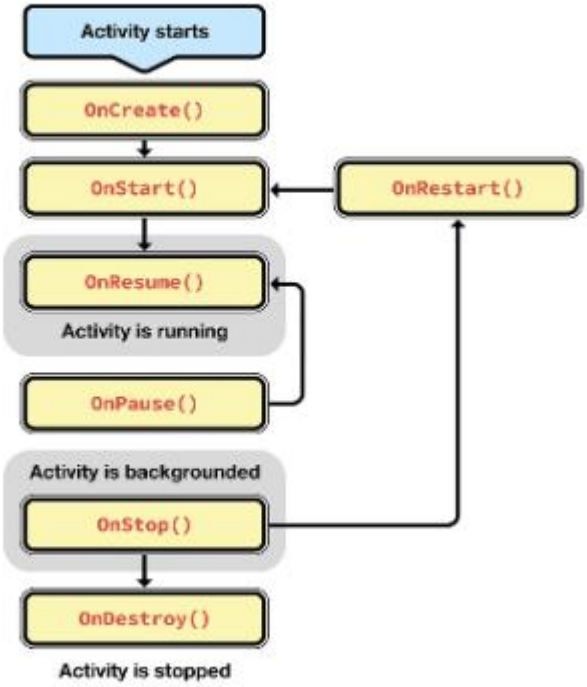


Рисунок 1.3. Схема життєвого циклу додатку[5]

Також Xamarin.Android додатки надають такі можливості:

1. використання фрагментів замість декількох activity для створення більш швидкого додатку, оскільки не потрібно підгружати різні activity у відповідь користувачу;
2. створення спеціальних провайдерів контенту та резолверів контенту, що надають можливість отримувати та надавати дані, а також шукати дані з для інших додатків у системі;
3. створення сервісів, що мають свій життєвий цикл та дозволяють виконання деяких складних задач в іншому потоці, наприклад, скачати статтю з певного сайту. Дані сервіси бувають трьох типів, а також можуть використовуватися іншими додатками у системі, створювати повідомлення для користувача та інше[3];
4. Xamarin.Android підтримує усі API рівні ОС Android, отже розробникам завжди доступні усі можливості даної платформи[3];

5. як і в звичайному додатку під Android, Xamarin.Android надає змогу створювати ресурси – зображення, елементи інтерфейсу, локалізація, аудіо та відео, файлова система та інше.

Отже, Xamarin.Android наділена багатим функціоналом для розробки додатків під ОС Android, нічим не гірші від Java аналогів, дозволяє використовувати усе, що надає платформа, а додатки по швидкодії не програють аналогам.

1.4 Опис Xamarin.iOS та її основні можливості

Технологія Xamarin.iOS відрізняється від Xamarin.Android, в першу чергу це пов‘язано з кардинальними відмінностями даних платформ. Навідміну від Android, iOS не дозволяє використовувати середовища виконання, тому використовувати Mono не вийде. Внутрішня структура додатку для iOS виглядає таким чином (рис. 1.4):

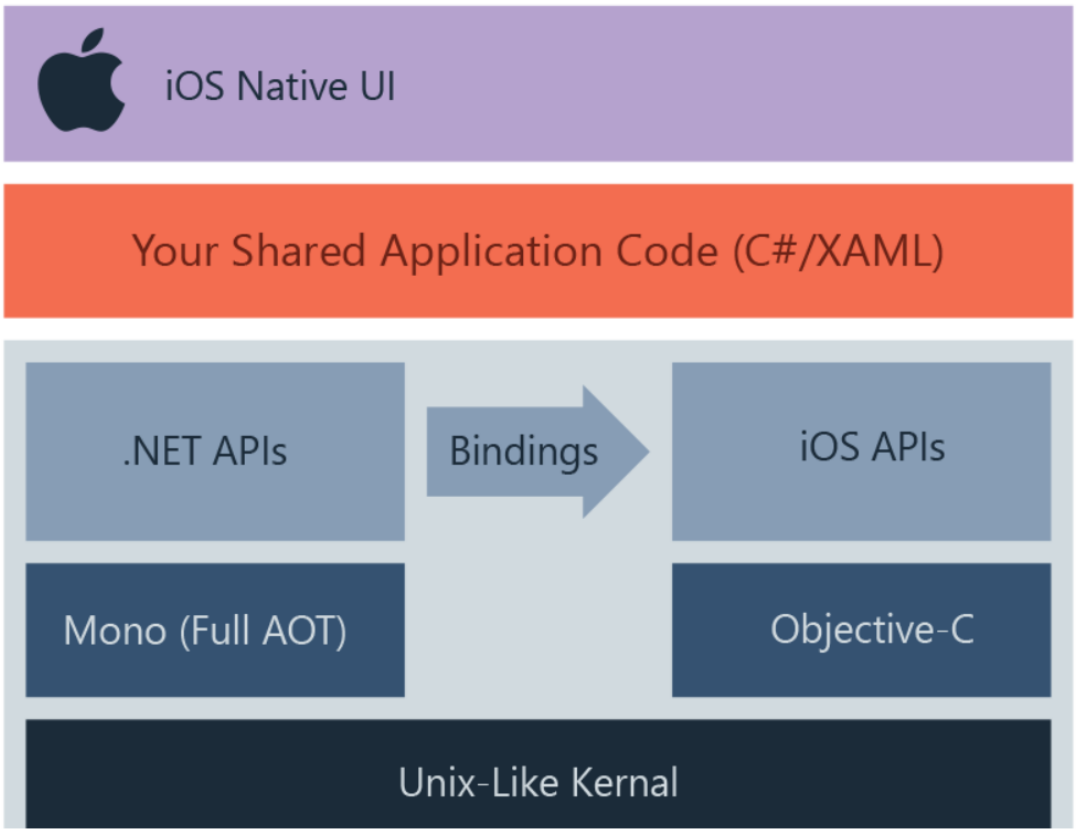


Рисунок 1.4 – Принцип роботи Xamarin.iOS[1]

Отже, як видно з рис 2.5, технологія Xamarin.Android не використовує Mono для додатку. Для iOS використовуються спеціальні прив‘язки, для

нативних компонентів та можливостей даної ОС. Однак, неможливість використання середовища виконання – не проблема, оскільки навідміну від використання JIT(Just-In-Time) компіляції, додаток, що написаний на C# під iOS компілюється одразу в байт код і для даного додатку уже не потрібно використання будь-якого середовища виконання[6].

Як і для Xamarin.Android, Xamarin.iOS дозволяє використання усіх нативних контролів та можливостей платформи та пристрою, на якому виконується програма.

Основними робочими блоками в Xamarin.iOS є event, delegate та protocol. Event викликається під час взаємодії користувача з пристроєм, такі event можна зв‘язати із кнопкою чи іншим елементом управління (UIKit) для створення певної поведінки додатку та реагування на дії користувача, ви можете прив‘язувати до елементів багато подій (events) та присвоювати їм різну поведінку. Protocol – щось на зразок інтерфейсу (interface) в C#, що дозволяє ―дізнатися ОС який метод викликати, не знаючи його поведінку, а delegate використовуються як відповідь (callback) на якусь дію[6]. Життєвий цикл додатку в iOS (рис. 1.5):

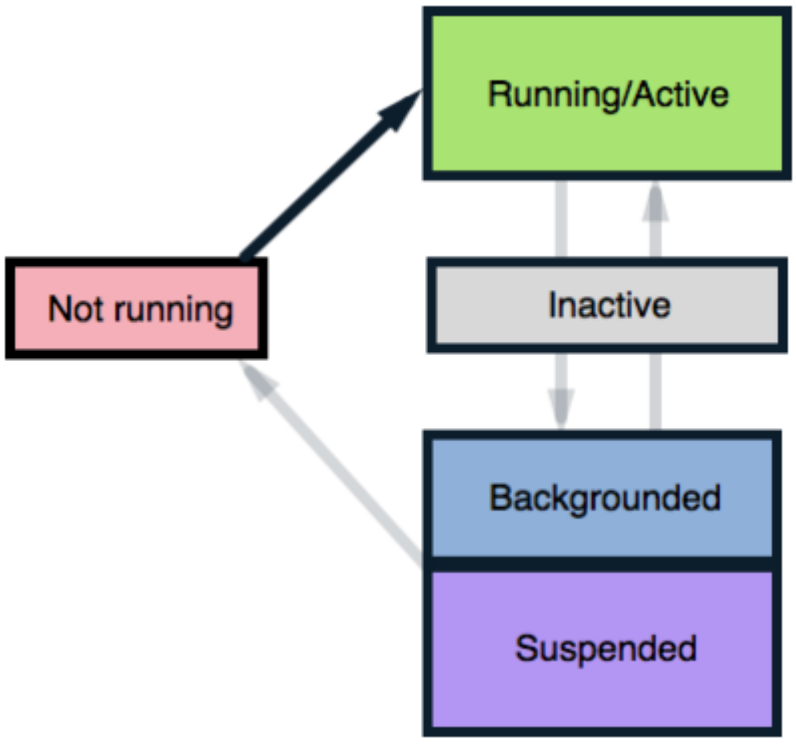


Рисунок 1.5. Життєвий цикл додатку в iOS[6]

Загалом додаток може бути у декількох станах (див. рис. 1.5):

1. Not running – додаток ще не було запущено в пристрої або його процес було знищено через недостату пам‘яті, або користувач закрив додаток;
2. Running/Active – додаток відображається на екрані і виконується;
3. Inactive – додаток було призупинено через мобільний виклик, або інший додаток чи інше;
4. Backgrounded – додаток виконує певну дію і його не видно на екрані (працює в фоновому режимі);
5. Suspended – додаток не має фонових дій, тому його було призупинено системою.

Дана схема роботи набагато легша ніж для ОС Android, оскільки не має ―зайвих методів і більш зрозуміла. Тепер розглянемо, як програміст з використанням Xamarin.iOS може оброблювати ці стани. Коли додаток змінює свій стан операційна система сповіщає додаток через event методи в спеціальному класі AppDelegate[6] (рис. 1.6):

1. OnActivated – додаток вперше запускається, або додаток повертається з фонового стану. В даний метод пишеться логіка, яку потрібно виконати, коли додаток запускається[6];
2. OnResignActivation – коли додаток потрібно прервати та перейти до фонового стану викликається даний метод, наприклад, під час дзвінку на телефон;
3. DidEnterBackground – коли додаток переходить у фоновий стан, ОС дає приблизно 5 секунд на те, щоб зберегти усі дані, в даному методі можна їх зберегти[6];
4. WillEnterForeground – додаток переходить з фонового чи призупиненого стану в запущений стан, тобто, користувач знову взаємодіє з додатком;
5. WillTerminate – додаток вимикається, причини даної поведінки описано в минулому параграфі.

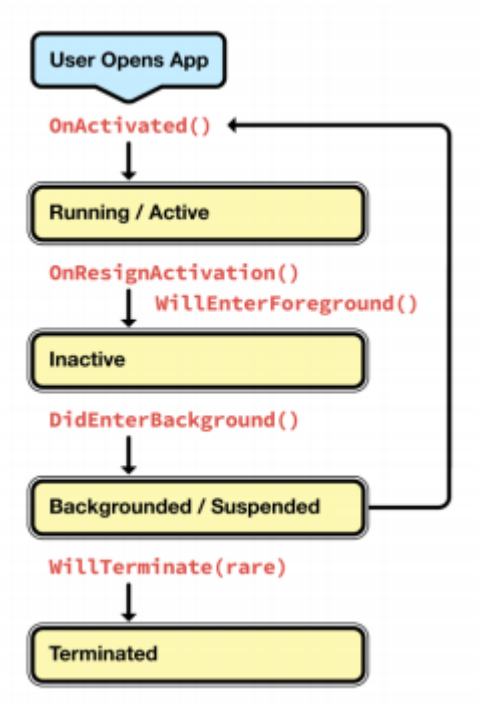


Рисунок 1.6 – Схема викликів обробників подій зміни стану додатку в Xamarin.iOS[6]

Таким чином, було розглянуто основні особливості Xamarin.iOS, дана технологія відрізняється від Xamarin.Android та надає усі можливості пристрою та ОС для програміста.

1.5 Опис Xamarin.Forms та її основні можливості

Xamarin.Forms – це зовсім несхожа технологія на дві інші, що були описані раніше. Дана технологія більше схожа на JavaScript фреймворки, які було описано в першому розділі. Різниця в тому, що Xamarin.Forms дозволяє писати нативні додатки з більшою кількістю спільного коду. Наприклад, інтерфейс програми буде спільний для усіх додатків(по структурі та по написанню, проте матиме дещо різний вигляд на платформах) оскільки для побудови інтерфейсу користувача використовується спеціальна мова розмітки – XAML (рис. 1.7). Тобто розробнику не потрібно знати особливості кожної платформи, розробники із Xamarin все зробили за вас.

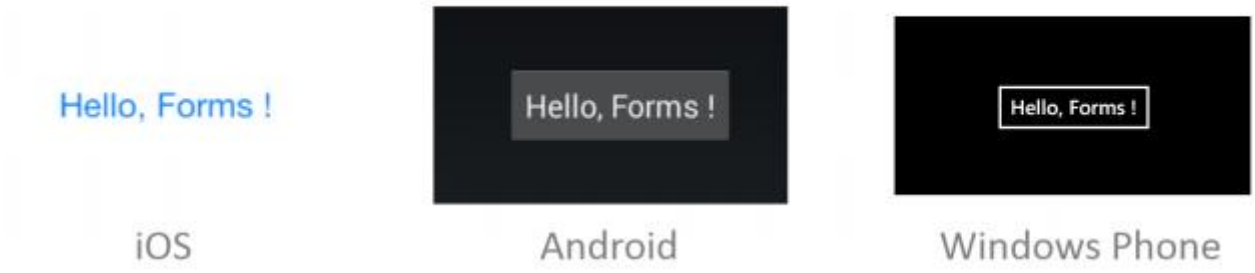


Рисунок 1.7 – Стандартний вигляд кнопок для кожної платформи в Xamarin.Forms[7]

Xamarin.Forms надає розробнику абстракції графічного інтерфейсу, що використовуються для кожної платформи. Дана технологія використовує нативні компоненти під час роботи додатку, тобто код графічного інтерейсу на різній платформі має різне підґрунтя, а саме його буде перетворено на нативний код окремої конкретної платформи. Код Xamarin.Forms має можливість взаємодіяти з операційною системою, використовувати її програмний інтерфейс[8].

Основними типами компонентами, що використовуються в розробці з використанням Xamarin.Forms є:

1. Представлення

Представлення – це основний блок графічного інтерфейсу, бо саме з цих елементів його побудовано. Прикладом представлення є кнопка, поле вводу тощо.

2. Макет

Макети визначають, як представлення будуть розташовані на екрані, їх розмір при різних розмірах екрану, поведінку представлень в певних ситуаціях;

3. Сторінка

Сторінка має дві основні функції. Перша - сторінка представляє собою контейнер для представлень та макетів, тобто сторінка і є екраном телефону, також сторінка має і іншу функцію – навігаційну. Саме сторінка дозволяє перехід від однієї сторінки до іншої[7].

Основними блоками, що дозволяють писати логіку додатку є клас Device, центр обміну повідомленнями(message center) та сервіс залежностей (dependency service). Клас Device надає методи для використання платформоспецифічних можливостей додатку та побудови поведінки додатку з точки зору ОС. Центр обміну повідомленнями надає можливість використовувати методи publish/subscribe – тобто підписуватися на якусь подію, що сталася в ОС, чи, наприклад, зіставити метод, що буде спрацьовувати при натисканні кнопки. [7].

1.6 Програма SourceTree та її основні функції

SourceTree – безкоштовний візуальний клієнт системи управління версіями Git та Mercurial, який працює на Windows, Mac OS X. SourceTree полегшує взаємодію з репозиторіями Git та Mercurial, для того щоб ви могли сконцентруватись на кодуванні. Візуалізація та управління репозиторіями через простий інтерфейс SourceTree.[9]

За допомогою даної програми дуже легко зберігати актуальні версії продукту. Також дуже легко та зручно робити нові вітки для добавлення нового функціоналу продукту водночас робити це паралельно актуальній версії продукту(див рис. 1.8). В кінці, для того щоб новий готовий функціонал продукту добавити до актуальної версії, треба лише доліпити вітку до вітки яка зберігає актуальну версію та яка називається зазвичай develop(див рис. 1.9)

Ще одною перевагою даної програми є те що у випадку коли був внесений неправильний або не працюючій функціонал в актуальну версію програми, завжди є можливість повернутись до попередньої актуальної версії а зміни які було ненароком чи навмисно внесені в актуальну версію програми будуть відмінені повністю або зі змогою зберегти зміни які були внесені в окремій вітці локально.

Дана програма дає можливість створити декілька видів віток кожна з яких відповідає своїй меті:

1. feature – вид гілки яка є відокремленою версією актуальної вітки призначена для того щоб створювати в ній нові функціонали до актуальної вурсії програми;
2. develop – вид гілки яка зберігає актуальну версію програми. В ній доліплюються готові feature з новим функціоналом;
3. release – вид гілки яка зберігає вже готові для розгортки актуальної версії програми;
4. master – вид гілки яка призначена для того щоб зберігати актуальну версію програми після зробленого релізу актуальної версії.

Найчастіші функції які виконуються за допомогою даної програми:

1. Branch – функція за допомогою якої можливо створити нову вітку починаючі з останньої актуальної версії(develop). Дана вітка не буде вносити ніякі зміни в develop а буде як окрема версія-вітка з добавленим функціоналу;
2. Commit – функція яка дозволяє зберегти внесені зміни в будь-якій з видів гілок;
3. Push – функція яка призначена для того щоб відправити збережені(commited) зміни до репозиторію Git або Mercurial;
4. Pull – функція призначена для того щоб витягнути всі зміни відповідної вітки з репозиторія Git або Mercurial;
5. Fetch – функція для оновлення візуального клієнта;
6. Merge – функція призначена для того щоб доліплювати відповідні вітки до develop;
7. Stash – функція яка дозволяє зберігати локально в окремому файлі зміни певної гілки які не були збережені в локальному репозиторії але їх треба зберегти для подальшого використання.

Програма SourceTree є дуже комфортним та надійним візуальним інтерфейсом для того щоб всі зміни внесені в продукт були правильно і легко збережені або певним чином оброблені.

1.7 Програма Postman та її основні функції

Postman це свого роду швейцарський ніж, який дозволяє створювати і виконувати запити, документувати й моніторити Ваші сервіси в одному місці[10].

Приклад надсилання запитів:

Якщо Ви встановлюєте Postman вперше, команда розробки надає Вам колекцію “Postman Echo”. Це набір збережених запитів (і відповідей), організованих логічно. Postman Echo передбачена для легкого старту тестування API із заздалегідь налаштованими запитами, від яких Ви вільно можете відштовхуватися[10].

Для виконання елементарних запитів достатньо:

1. вибрати тип запиту
2. вбити запит у відповідне поле …
3. … або заповнити параметри через форму
4. натиснути кнопку “Send”

Якщо відкрити цей набір, перейти в “Методи запитів” і потім в “Запити GET”, всі збережені дані відобразяться в центральній частині вікна Postman. Тепер натисніть “Відправити”[10](рис. 1.10).

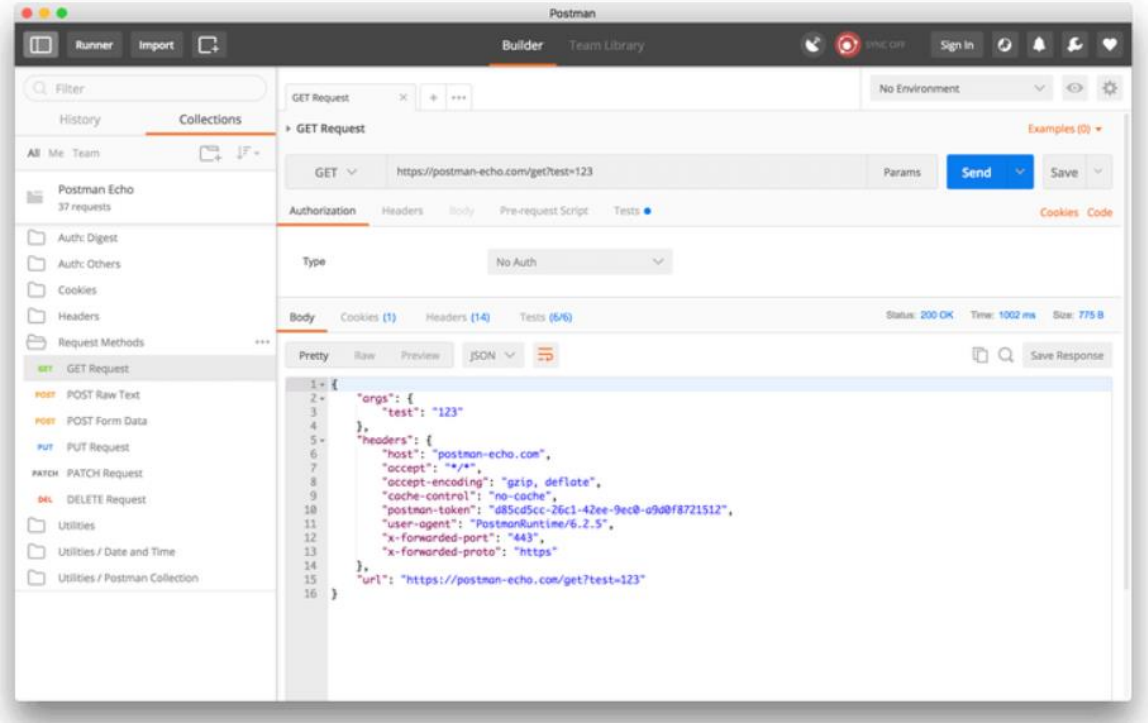


Рисунок 1.10 – Результат запиту[10]

Postman — зарекомендував себе відмінним засобом підмоги в повсякденній рутині API-тестування. Він позбавляє вас від купи текстових файлів з збереженими запитами, позбавляє від нудних і непоказних консольних запитів.

1.8 Висновок до розділу 1

В даному розділі було розглянуто основні можливості Xamarin для побудови нативних додатків під конкретні платформи, а саме Xamarin.Android, Xamarin.iOS та Xamarin.Forms, зазначено основні принципи роботи Xamarin для кожної ОС, а саме як компілюються додатки та з чим взаємодіють, та що використовують — під капотом, як видно, усі підходи в Xamarin кардинально відрізняються для кожної з оглянутих платформ, наприклад, ОС Android використовує ARM (раніше – Dalvik), для своєї роботи, iOS взагалі забороняє використання середовищ виконання, що унеможливлює використання Mono для данної платформи. Як видно, ця різниця зумовлена самими платформами, хоча вони і побудовані на ядрі Linux.

Окрім можливостей Xamarin як фреймворка, було розглянуто і історію створення даного фреймворка, його розвиток, та подальший розвиток.

Ще однією цікавою темою, що була розглянута, звичайно є Mono – середовище виконання, аналог CLR під Windows для Linux.

Також у цьому розділі були розглянуті дві програми які є дуже корисними у розробці ПП.

2. АЛГОРИТМ НАПИСАННЯ ПРОГРАМИ З ВИКОРИСТАННЯМ XAMARIN

2.1. Тестування та підготовка даних

2.1.1 Опис джерел та права доступу

Для створення даного продукту використовувались дані для розробників з ресурсу https://api.kino-teatr.ua/. Даний ресурс містить у собі задокументований список методів отримання даних сайту https://kino-teatr.ua/. З вище згаданого списку були взяті такі методи:

- Метод для отримання списку міст;
- Метод для отримання списку кінотеатрів у певному місті;
- Метод для отримання списку залів вибраного кінотеатра;
- Метод для отримання списку фільмів;
- Метод для отримання списку прем’єр;
- Метод для отримання одного фільму по ідентифікатору;
- Метод щоб отримати трейлер фільму по ідентифікатору;
- Метод для отримання списку акторів;
- Метод для отримання інформації про одного актора по ідентифікатору;
- Метод щоб отримати список усіх фільмів в місті починаючи з сьогоднішнього дня;

Цей ресурс містить набагато більшу кількість методів які в подальшому будуть використовуватись в проекті, але вище наведені методи є основним та найважливішим ресурсом для отримання даних які оброблюються в проекті.

Також, для того щоб мати доступ до інформації яка міститься у кожному методі окремо потрібен доступ. Кожен метод який є згаданим вище називається викликом(call) який далі буде описуватись. Виклик містить у собі певні правила, і одним із них є ключ який можливо отримати тільки від розробників які працюють з вище згаданим ресурсом. В ресурсі показані виклики які містять тестовий ключ який не дозволяє отримати дані. Для того щоб не порушувати умови домовленості з розробниками даного ресурсу, я показуватиму загальний вигляд виклику та його основні правила використовуючи тестовий ключ.

2.1.2 Опис та підготовка списку викликів

Як вище було згадано, основним джерелом інформації для додатку є виклики. Для того щоб протестувати та побачити дані які приходять з ресурсу використовується програма Postman. Для того щоб виконати виклик потрібно взяти один метод з вище наведених з ресурсу, вставити його у потрібне поле, вибрати тип виклику, поміняти ключ на потрібний та натиснути Send(рис. 2.1).

Рисунок 2.1 – Загальний вигляд виклику

На цьому малюнку показано виклик за допомогою якого можливо отримати список усіх фільмів. Цей виклик містить певні правила по яких він побудований:

1. GET - вид виклику (у нашому випадку виклик на отримання даних);
2. http://api.kino-teatr.ua - Базовий URL на який посилається виклик і яким є основним ресурсом усіх даних;
3. /rest – правило яке є розділом який відповідає за збереження масиву підрозділів;
4. /films – правило яке є підрозділом який зберігає у собі данні у вигляді об’єктів які в свою чергу містять дані певного фільму;
5. ?apiKey= - назва правила яке повинно дорівнювати ключу доступу до даних;
6. apiKey – ключ доступу до даних(тестовий);
7. &size= - назва правила яке повинно дорівнювати кількості об’єктів які міститиме результат виклику;
8. 100 – кількість повернених об’єктів;
9. Send – кнопка яка відповідає за те щоб побудоване правило виконалось і повернуло результат.

Після того як було натиснене кнопка Send, виклик оброблюється. Час оброблення виклику залежить від вказаної кількості об’єктів які повинні

повернутись. Після завершення оброблення виклику, як результат повернеться JSON об’єкт (рис. 2.2).

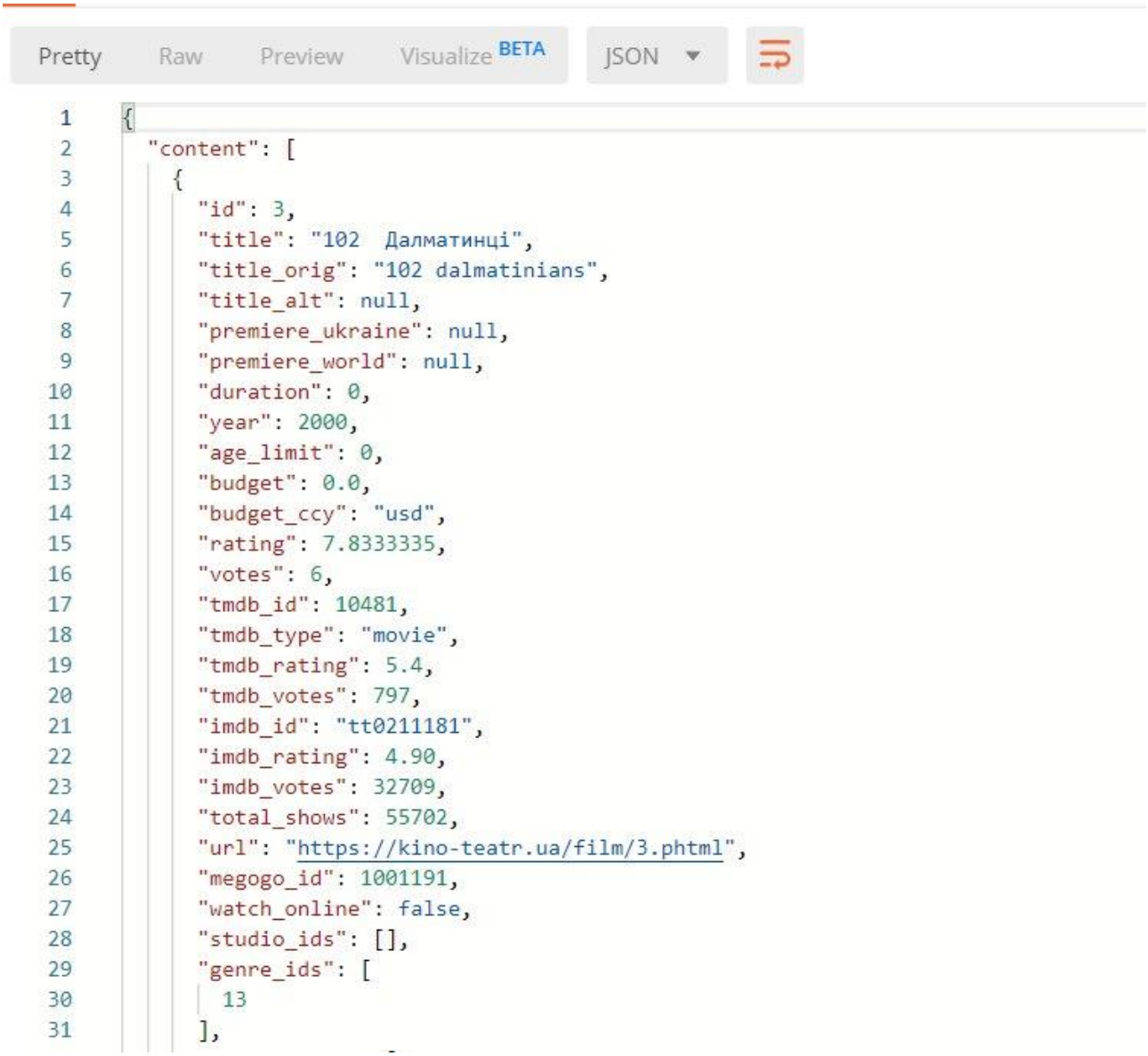


Рисунок 2.2. JSON об’єкт з даними про фільми

Отриманий об’єкт є об’єктом який містить у собі поле content, який у свою чергу є масивом який включає в собі кількість об’єктів яка була вказана в виклику. Об’єкти які містяться в content є моделями які мають поля, кожна з яких відповідає певній інформації про фільм.

2.2. Побудова додатку

2.2.1. Створення проекту

Для того щоб створити новий Xamarin.Forms проект потрібно інсталювати Visual Studio 2019 якщо він ще не встановлений. Якщо вже встановлений або закінчили його інсталювати, тоді щоб створити новий проект потрібно зробити декілька легких кроків:

1. Запустити Visual Studio 2019 та на початковій сторінці вибрати Create new project(рис 2.3);

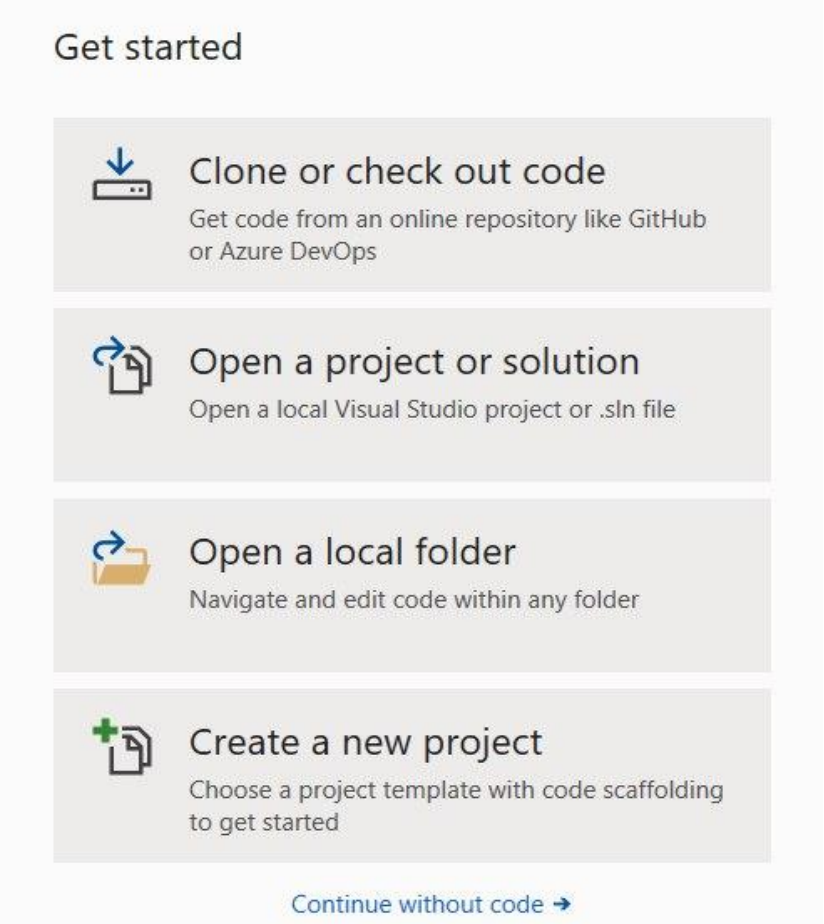


Рисунок 2.3. Частина початкової сторінки

2. У вікні в якому відкрилась знайти поле пошуку та ввести в ній Xamarin для того щоб знайти потрібний проект. Зі списку результату пошуку вибрати Mobile App (Xamarin.Forms) (рис. 2.4). Натиснути Next;

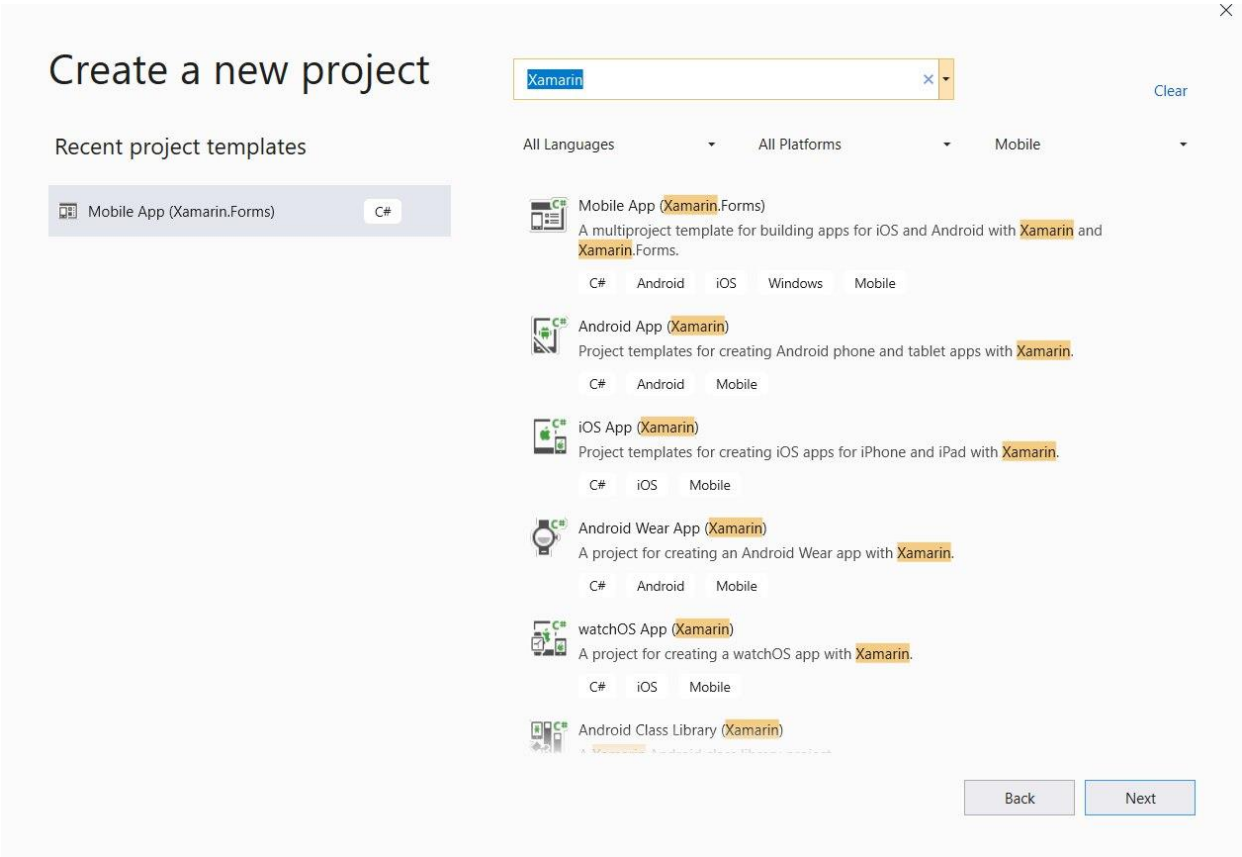


Рисунок 2.4. Вікно створення нового проекту

3. У вікні налаштування проекту задати назву проекту, вибрати шлях для його збереження та задати назву рішення(не обов’язково)(рис. 2.5). Натиснути Next;

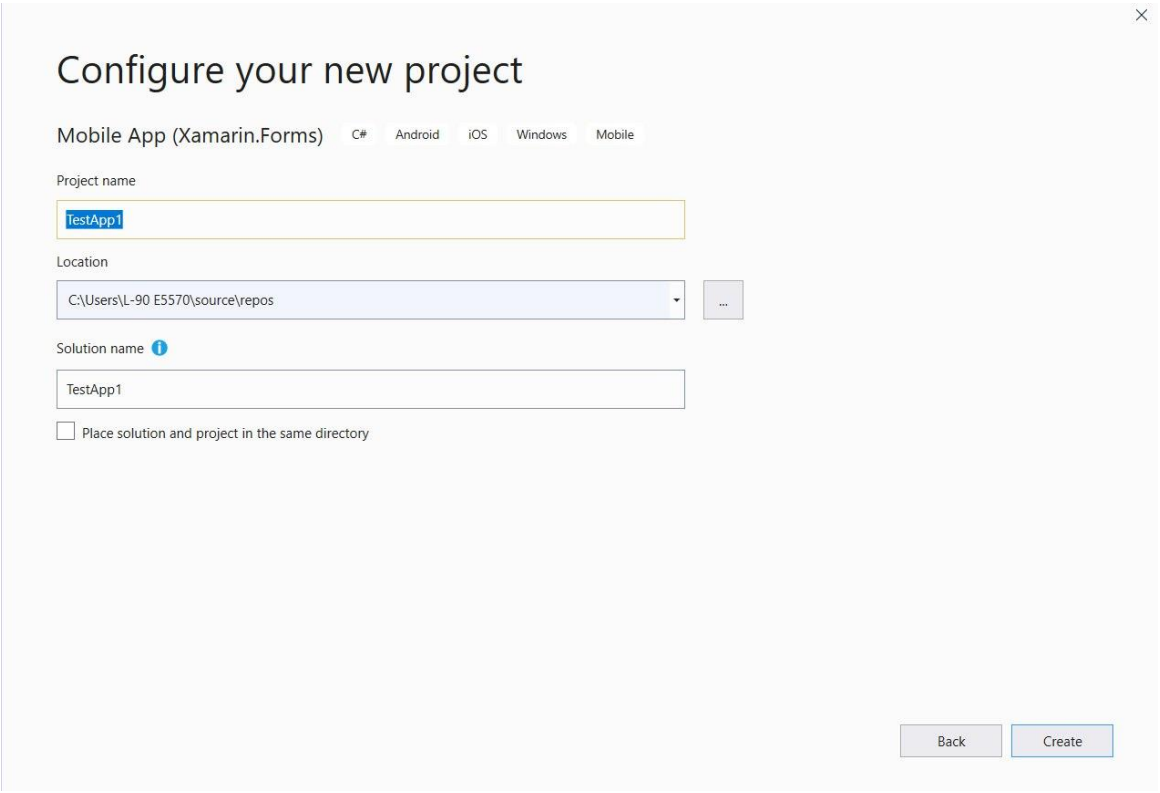


Рисунок 2.5. Вікно налаштування нового проекту

4. У вікні в якому відкрилась вибрати бажаний шаблон проекту, вибрати платформи під які буде створений проект та натиснути OK(рис. 2.6);

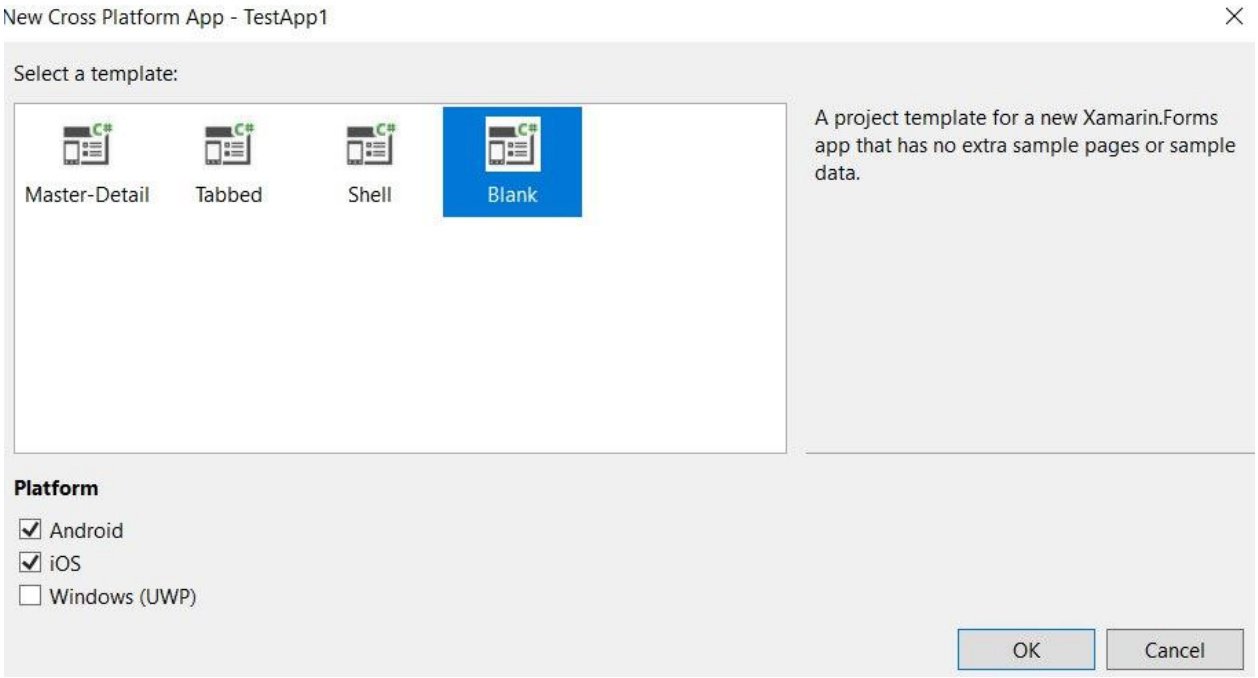


Рисунок 2.6. Вікно вибору шаблону

Після виконання усіх кроків відкриється новий проект готовий для того щоб він був виконаний на мобільних пристроях та готовий до того щоб у ньому додавався новий функціонал.

2.2.2. Підготовка моделей та моделей відображення

Для того щоб отримати дані з зовнішнього ресурсу потрібно першим кроком підготувати моделі які будуть ідентичними тим об’єктам які містяться у потрібному виклику. Таким чином при десеріалізації JSON об’єкту потрібно вказати під яку модель буде відбуватись десеріалізація. Моделі зберігатимуть інформацію одного об’єкта який міститься у виклику. Поля моделі повинні абсолютно точно співпадати з полями об’єкту(рис. 2.7-2.8) але модель не обов’язково повинна містити у собі всі поля які містить об’єкт. Всі моделі показані в Додаток 1.

```
9 references
class MovieInfo
{
    1 reference
    public int Id { get; set; }
    1 reference
    public string Title { get; set; }
    0 references
    public string Title_Orig { get; set; }
    0 references
    public DateTime? Premiere_Ukraine { get; set; }
    0 references
    public DateTime? Premiere_World { get; set; }
    0 references
    public int? Duration { get; set; }
    1 reference
    public int? Year { get; set; }
    0 references
    public int? Age_Limit { get; set; }
    0 references
    public decimal? Budget { get; set; }
    0 references
    public string Budget_Ccy { get; set; }
    1 reference
    public decimal? Rating { get; set; }
    0 references
    public int? Votes { get; set; }
    0 references
    public int? Total_Shows { get; set; }
    0 references
    public string Url { get; set; }
    0 references
    public int[] Studio_Ids { get; set; }
```

Рисунок 2.7 – Модель фільму

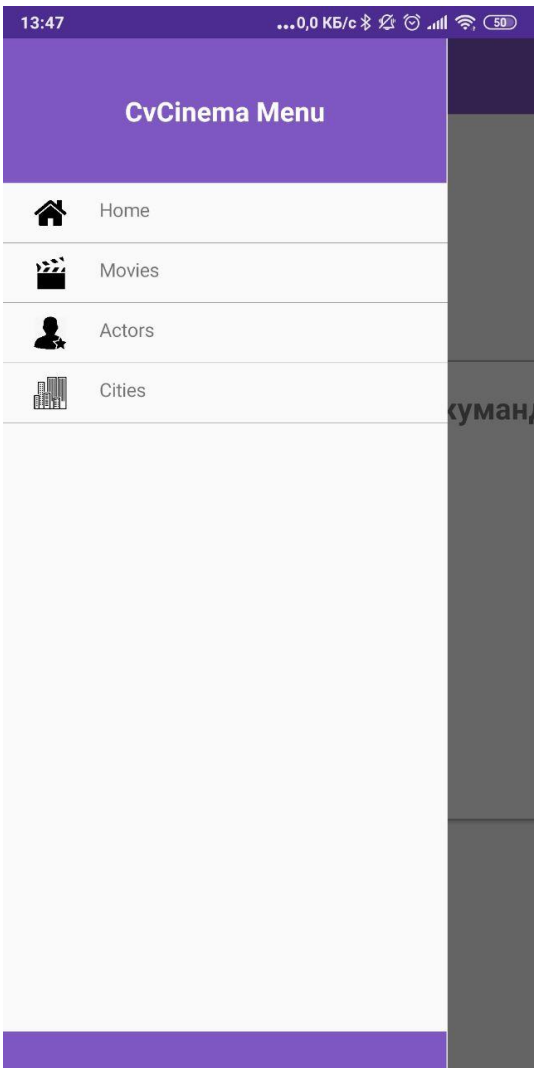

```
"id": 3,
"title": "102  Далматинці",
"title_orig": "102 dalmatinians",
"title_alt": null,
"premiere_ukraine": null,
"premiere_world": null,
"duration": 0,
"year": 2000,
"age_limit": 0,
"budget": 0.0,
"budget_ccy": "usd",
"rating": 7.8333335,
"votes": 6,
"tmdb_id": 10481,
"tmdb_type": "movie",
"tmdb_rating": 5.4,
"tmdb_votes": 797,
"imdb_id": "tt0211181",
"imdb_rating": 4.90,
"imdb_votes": 32709,
"total_shows": 55702,
"url": "https://kino-teatr.ua/film/3.phtml",
"megogo_id": 1001191,
"watch_online": false,
"studio_ids": [],
```


Рисунок 2.8 – Об’єкт виклику

Модель відображення служить для того щоб в цьому класі підготувати дані для відображення їх на сторінці. Також в проекті, в кожній моделі відображення були побудовані функції за допомогою який були витягнуті дані з зовнішнього ресурсу та збережені в моделі відображення. В даній функції використовується клас HttpClient який є базовим класом для відправлення HTTP-запитів і отримання HTTP-відповідей від ресурсу із вказаним URL[11]. За його допомогою дані з зовнішнього ресурсу витягуються і оброблюються в нашій моделі відображення. Всі моделі відображення відображені в Додаток 2.

2.2.3. Бокового меню

Для того щоб добавити бокове меню використовується MasterDetailPage. Xamarin.Forms MasterDetailPage являється сторінкою, яка керує двума сторінками зав’язаних даними – головною сторінкою, яка показує елементи, та сторінкою властивостей, яка показує властивості елементів головної сторінки[12].



Ця сторінка містить об’єкт ListView який заповнюється даними в XAML. Для цього його властивості ItemSource присвоюється масив екземплярів MasterPageItem. Кожен об’єкт MasterPageItem визначає властивості Title, IconSource та TargetType.

Об’єкт DataTemplate присвоюється властивості ListView.ItemTemplate для відображення кожного об’єкту MasterPageItem. Об’єкт DataTemplate містить об’єкт ViewCell який складається з об’єктів Image та Label. Об’єкт Image відображає значення властивості IconSource, а об’єкт Label відображає значення властивості Title для кожного об’єкту MasterPageItem.

Для даної сторінки задані властивості Title та IconImageSource. Якщо у сторінки відомостей є заголовок, на ній появиться значок. Детальний код сторінки міститься у Додаток 3.

2.2.4. Домашня сторінка

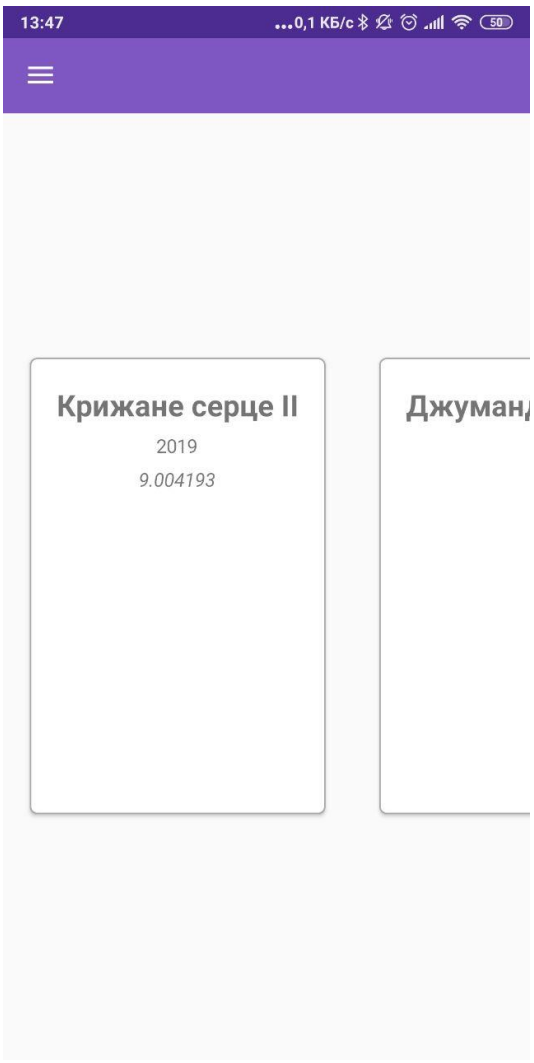
На домашній сторінці показано список прем’єр яких показують в кінотеатрах міста Чернівці починаючи з сьогоднішнього дня. Список представлений у вигляді CollectionView.

CollectionView визначає наступні властивості які керують макетом:

- ItemsLayout типу IItemsLayout який вказує на використаний макет;
- ItemSizingStrategy типу IItemSizingStrategy яка вказує на приблизну стратегію мірок елементів.

CollectionView відображає свої елементи в горизонтальному списку, присвоїв властивості ItemLayout значення HorizontalList.

Дана колекція містить собі об’єкти які були витягнуті з виклику спеціально для того щоб витягнути всі прем’єри фільмів вказаного міста починаючі з сьогоднішнього дня. Ці об’єкти містять таку інформацію як назва фільму, рік прем’єри та рейтинг фільму.



Кожен об’єкт зі списку має властивість щоб на нього можливо було клікати. При виконанні цієї дії можливий перехід на сторінку з детальною інформацією про фільм. Навігація здійснюється за допомогою ієрархічній навігації NavigationPage.

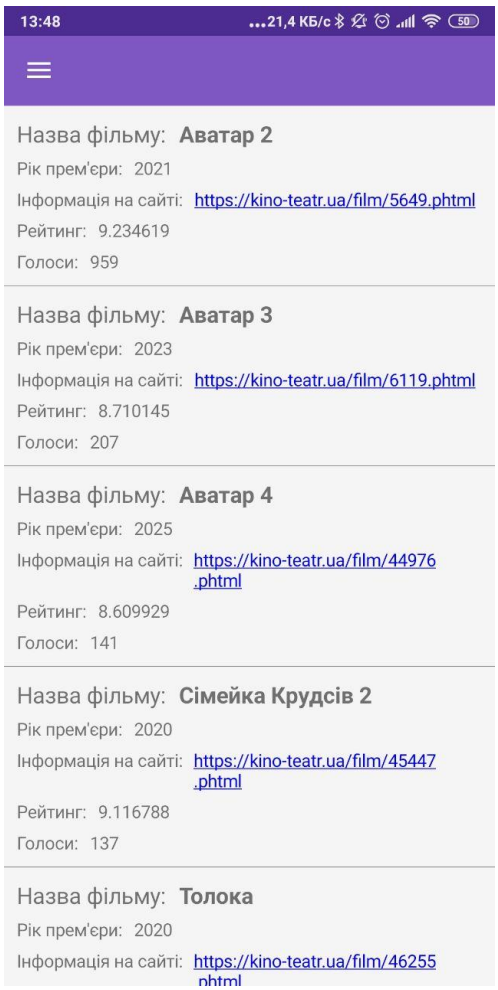
NavigationPage це клас який забезпечує ієрархічну навігацію, за допомогою якої користувач має змогу переходити по сторінкам вперед та назад на своє бажання. Цей клас реалізує навігацію на основі сітки об’єктів Page по методу ЛІФО (остання поступила – перша обслугована). Детальний код сторінки міститься у Додаток 4.

2.2.5. Сторінки зі списком фільмів, акторів, міст, кінотеатрів та залів

Сторінки містить у собі списки об’єктів витягнутих за допомогою викликів які витягують вказану кількість фільмів, акторів, міст, кінотеатрів та залів. Такими викликами є:

- https://api.kino-teatr.ua/rest/films?apiKey=apiKey8&size=100 – виклик, який витягує з зовнішнього ресурсу 100 фільмів;
- https://api.kino-teatr.ua/rest/persons?apiKey=apiKey&size=24 – виклик, який витягує 24 акторів;
- https://api.kino-teatr.ua/rest/cities?apiKey=apiKey&size=150 – виклик, який витягує 150 міст;
- https://api.kino-teatr.ua/rest/city/1/cinemas?apiKey=apiKey – виклик, який витягує всі кінотеатри по ідентифікатору міста;
- https://api.kino-teatr.ua/rest/cinema/50/halls?apiKey=apiKey – виклик, який витягує всі зали по ідентифікатору кінотеатру.

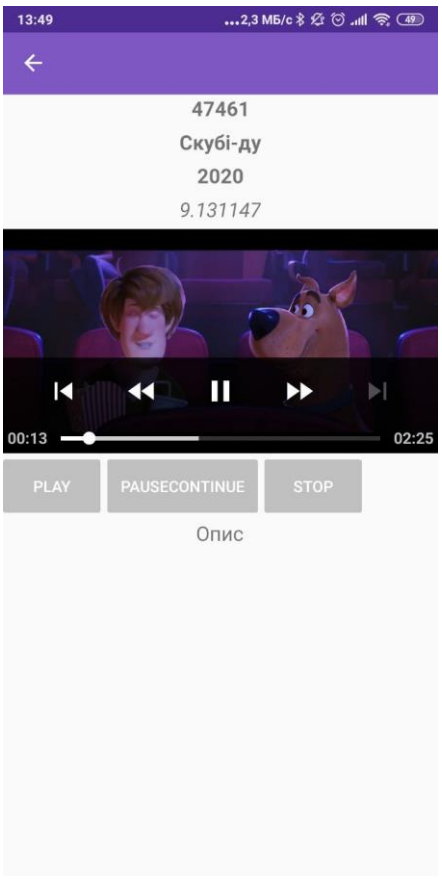
Список відображається за допомогою ListView який являє собою представлення для представлення списку даних, особливо довгих списків, які вимагають прокрутки.



Кожен елемент зі списку є клікабельним та при взаємодії з ним можливий перехід до сторінки з детальною інформацією про фільм та яка також містить трейлер фільму якщо він є. Перехід між сторінками здійснюється за допомогою ієрархічній навігації NavigationPage описаній вище. Детальний код сторінки міститься у Додаток 5.

2.2.6. Сторінка з інформацією про один фільм

Сторінка призначена для того щоб показати детальну інформацію про фільм а також трейлер фільму. В ній показується назва фільму, рік прем’єри рейтинг та плеєр в якому програється трейлер. Плеєр є плагінами Plugin.MediaManager та Plugin.MediaManager.Forms. Це спеціальний плагін який використовує нативний плеєр для програвання відео та аудіо. Також даний плеєр є ідеальним рішенням для програвання відео з зовнішніх ресурсів які надають спеціальний URL із відеом. Детальний код сторінки міститься у Додаток 6.


13:49
...2,3 МБ/с
47461
Скубі-ду
2020
9.131147
00:13
02:25
PLAY
PAUSECONTINUE
STOP
Опис

Висновки

У даній магістерскій роботі були розгорнуті методи тестування потрібних викликів та підготовка даних для подальшого їх використання. Також було показано як створювати новий Xamarin.Forms проект, за якими кроками це можливо зробити.

У роботі представлено готові функціонали продукту, описано їх реалізацію та функціональність. Описані виклики, з яких витягувалися дані, їх методи обробки, збереження та методи передачі даних на екран пристрою. Також до кожного функціоналу показані результати у вигляді скрінів.

СПИСОК ЛІТЕРАТУРИ

1. Building Cross-Platform iOS/Android Apps with Xamarin, Visual Studio, and C# - Part 1 by Jim Wilson – Режим доступу : https://app.pluralsight.com/library/courses/cross-platform-ios-android-visualstudio-csharp/table-of-contents/.
2. About mono – Режим доступу: http://www.mono-project.com/docs/aboutmono/.
3. Xamarin.Android application fundamentals – Режим доступу : https://developer.xamarin.com/guides/android/application_fundamentals/
4. Xamarin.Android architecture – Режим доступу : https://developer.xamarin.com/guides/android/under_the_hood/architecture/
5. Activity lifecycle – Режим доступу : https://developer.xamarin.com/guides/android/application_fundamentals/activit
6. Xamarin.iOS application fundamentals – Режим доступу : https://developer.xamarin.com/guides/ios/application_fundamentals/.
7. Working with Xamarin.Forms – Режим доступу : https://developer.xamarin.com/guides/xamarin-forms/working-with/
8. Charles Petzold. Creating Mobile Apps with Xamarin.Forms First Edition / Petzold C. – Redmond : Microsoft Press, 2016. – 1187 с
9. Материал из Национальной библиотеки им. Н. Э. Баумана – Режим доступу: https://ru.bmstu.wiki/SourceTree#.D0.9A.D0.BE.D0.BC.D0.BC.D0.B5.D0.BD.D1.82.D0.B8.D1.80.D0.BE.D0.B2.D0.B0.D0.BD.D0.B8.D0.B5
10. ЗНАЙОМТЕСЬ POSTMAN — ДЛЯ ТЕСТУВАЛЬНИКА MUST HAVE – Режим доступу: https://www.quality-assurance-group.com/znajomtes-postman-dlya-testuvalnyka-must-have/
11. HttpClient Class – Режим доступу: https://docs.microsoft.com/ru-ru/dotnet/api/system.net.http.httpclient?view=netframework-4.8

12. Xamarin.Forms Master-Detail Page – Режим доступу: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/navigation/master-detail-page

ДОДАТКИ

Додаток 1

```
using System;
using System.Collections.Generic;
using System.Text;

namespace CvCinemaProject.Models
{
    class ActorInfo
    {
        public int Id { get; set; }
        public string First_Name_Orig { get; set; }
        public string Last_Name_Orig { get; set; }
        public DateTime BirthDate { get; set; }
        public decimal Rating { get; set; }
        public int RatingSum { get; set; }
        public int RatingVotes { get; set; }
        public int NumComments { get; set; }
        public string Biography { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Text;

namespace CvCinemaProject.Models
{
    class Cinema
    {
        public int Id { get; set; }
        public int City_Id { get; set; }
        public string Name { get; set; }
        public string Site { get; set; }
        public string Phone { get; set; }
        public string Latitude { get; set; }
        public string Longitude { get; set; }
```

```
        public string Address { get; set; }
        public string Notice { get; set; }
        public int Num_Photos { get; set; }
        public string Description { get; set; }
        public string Url { get; set; }
    }
}
using System;
using System.Collections.Generic;
using System.Text;

namespace CvCinemaProject.Models
{
    class City
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Country_Id { get; set; }
    }
}
using System;
using System.Collections.Generic;
using System.Text;

namespace CvCinemaProject.Models
{
    class Country
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
using System;
using System.Collections.Generic;
using System.Text;

namespace CvCinemaProject.Models
{
    class FilmStudio
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
using System;
```

```
using System.Collections.Generic;
using System.Text;

namespace CvCinemaProject.Models
{
   class Genre
   {
      public int Id { get; set; }
      public string Name { get; set; }
   }
}
using System;
using System.Collections.Generic;
using System.Text;

namespace CvCinemaProject.Models
{
   class MovieInfo
   {
      public int Id { get; set; }
      public string Title { get; set; }
      public string Title_Orig { get; set; }
      public DateTime? Premiere_Ukraine { get; set; }
      public DateTime? Premiere_World { get; set; }
      public int? Duration { get; set; }
      public int? Year { get; set; }
      public int? Age_Limit { get; set; }
      public decimal? Budget { get; set; }
      public string Budget_Ccy { get; set; }
      public decimal? Rating { get; set; }
      public int? Votes { get; set; }
      public int? Total_Shows { get; set; }
      public string Url { get; set; }
      public int[] Studio_Ids { get; set; }
      public int[] Genre_Ids { get; set; }
      public int[] Country_Ids { get; set; }
      public int[] Distributor_Ids { get; set; }
      //public string Description { get; set; }

      //public List<FilmStudio> FilmStudios { get; set; }
      //public List<Country> Countries { get; set; }
   }
}
using System;
using System.Collections.Generic;
```

```
using System.Text;

namespace CvCinemaProject.Models
{
   class MoviePremiere
   {
      public int Id { get; set; }
      public DateTime Begin { get; set; }
      public DateTime End { get; set; }
      public int Film_Id { get; set; }
      public int Hall_Id { get; set; }
   }
}
using System;
using System.Collections.Generic;
using System.Text;

namespace CvCinemaProject.Models
{
   class MovieTrailer
   {
      public int Id { get; set; }
      public string Title { get; set; }
      public int Film_Id { get; set; }
      public string Language { get; set; }
      public int Duration { get; set; }
      public DateTime Date { get; set; }
      public string Url { get; set; }
      public int Order { get; set; }
   }
}
using System;
using System.Collections.Generic;
using System.Text;

namespace CvCinemaProject.Models
{
   class Sale
   {
      public int Id { get; set; }
      public string Name { get; set; }
      public int Cinema_Id { get; set; }
      public bool isThreeD { get; set; }
   }
}
```

Додаток 2

```
using CvCinemaProject.Models;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Net.Http;
using System.Text;

namespace CvCinemaProject.ViewModels
{
   class ActorsListViewModel
   {
      ObservableCollection<ActorInfo> listOfActors = new
ObservableCollection<ActorInfo>();
      public ObservableCollection<ActorInfo> ListOfActors { get { return
listOfActors; } }

      public async void LoadData()
      {
         string url = "https://api.kino-teatr.ua/rest/persons?apiKey=apiKey&size=10";

         try
         {
            HttpClient client = new HttpClient();
            client.BaseAddress = new Uri(url);
            var response = await client.GetAsync(client.BaseAddress);
            response.EnsureSuccessStatusCode();

            var content = await response.Content.ReadAsStringAsync();
            JObject o = JObject.Parse(content);

            //ObservableCollection<ActorInfo> listOfActors = new
ObservableCollection<ActorInfo>();
            var actorsStr = o.SelectToken(@"$.content");
            foreach (var actor in actorsStr)
            {

listOfActors.Add(JsonConvert.DeserializeObject<ActorInfo>(actor.ToString()));
            }

         }
         catch (Exception)
```

```
            {
            }
        }

    }
}
using CvCinemaProject.Models;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Net.Http;
using System.Text;
using System.Windows.Input;
using Xamarin.Forms;

namespace CvCinemaProject.ViewModels
{
    class ActorViewModel : INotifyPropertyChanged
    {
        private string firstname;
        private string lastname;
        private DateTime birthdate;
        private decimal rating;
        private string biography;
        public string FirstName
        {
            get { return firstname; }
            private set
            {
                firstname = value;
                OnPropertyChanged("FirstName");
            }
        }
        public string LastName
        {
            get { return lastname; }
            private set
            {
                lastname = value;
                OnPropertyChanged("LastName");
            }
        }
```

```
public DateTime BirthDate
{
   get { return birthdate; }
   private set
   {
      birthdate = value;
      OnPropertyChanged("BirthDate");
   }
}
public decimal Rating
{
   get { return rating; }
   private set
   {
      rating = value;
      OnPropertyChanged("Rating");
   }
}
public string Biography
{
   get { return biography; }
   private set
   {
      biography = value;
      OnPropertyChanged("Biography");
   }
}
public async void LoadData(int id)
{
   string url = "https://api.kino-teatr.ua/rest/person/" + id + "?apiKey=apiKey";

   try
   {
      HttpClient client = new HttpClient();
      client.BaseAddress = new Uri(url);
      var response = await client.GetAsync(client.BaseAddress);
      response.EnsureSuccessStatusCode();

      var content = await response.Content.ReadAsStringAsync();
      JObject o = JObject.Parse(content);

      var actorInfo = JsonConvert.DeserializeObject<ActorInfo>(o.ToString());

      this.FirstName = actorInfo.First_Name_Orig;
      this.LastName = actorInfo.Last_Name_Orig;
```

```
            this.BirthDate = actorInfo.BirthDate;
            this.Rating = actorInfo.Rating;
            this.Biography = actorInfo.Biography;
        }
        catch (Exception)
        {
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;
    public void OnPropertyChanged(string prop = "")
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(prop));
        }
    }
  }
}
using CvCinemaProject.Models;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Net.Http;
using System.Text;

namespace CvCinemaProject.ViewModels
{
  class CinemasListViewModel
  {
    ObservableCollection<Cinema> listOfCinemas= new
ObservableCollection<Cinema>();
    public ObservableCollection<Cinema> ListOfCinemas{ get { return
listOfCinemas; } }

    public async void LoadData(int id)
    {
        string url = "https://api.kino-teatr.ua/rest/city/" + id+
"/cinemas?apiKey=apiKey";

        try
        {
            HttpClient client = new HttpClient();
```

```
            client.BaseAddress = new Uri(url);
            var response = await client.GetAsync(client.BaseAddress);
            response.EnsureSuccessStatusCode();

            var content = await response.Content.ReadAsStringAsync();
            JObject o = JObject.Parse(content);

            var cinemasStr = o.SelectToken(@"$.content");
            foreach (var cinema in cinemasStr)
            {

listOfCinemas.Add(JsonConvert.DeserializeObject<Cinema>(cinema.ToString()));
            }

        }
        catch (Exception)
        {
        }
    }
  }
}
using CvCinemaProject.Models;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Net.Http;
using System.Text;

namespace CvCinemaProject.ViewModels
{
  class CitiesListViewModel
  {
    ObservableCollection<City> listOfCities= new ObservableCollection<City>();
    public ObservableCollection<City> ListOfCities { get { return listOfCities; } }

    public async void LoadData()
    {
      string url = "https://api.kino-teatr.ua/rest/cities?apiKey=apiKey&size=100";

      try
      {
        HttpClient client = new HttpClient();
        client.BaseAddress = new Uri(url);
```

```
            var response = await client.GetAsync(client.BaseAddress);
            response.EnsureSuccessStatusCode();

            var content = await response.Content.ReadAsStringAsync();
            JObject o = JObject.Parse(content);

            var cityStr = o.SelectToken(@"$.content");
            foreach (var city in cityStr)
            {

listOfCities.Add(JsonConvert.DeserializeObject<City>(city.ToString()));
            }
        }
        catch (Exception)
        {
        }
    }
  }
}
using CvCinemaProject.Models;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Net.Http;
using System.Text;

namespace CvCinemaProject.ViewModels
{
  class CountriesListViewModel
  {
    ObservableCollection<Country> listOfCountries = new
ObservableCollection<Country>();
    public ObservableCollection<Country> ListOfCountries { get { return
listOfCountries; } }

    public async void LoadData()
    {
      string url = "https://api.kino-
teatr.ua/rest/countries?apiKey=apiKey&size=100";

      try
      {
```

```
            HttpClient client = new HttpClient();
            client.BaseAddress = new Uri(url);
            var response = await client.GetAsync(client.BaseAddress);
            response.EnsureSuccessStatusCode();

            var content = await response.Content.ReadAsStringAsync();
            JObject o = JObject.Parse(content);

            var countryStr = o.SelectToken(@"$.content");
            foreach (var country in countryStr)
            {

listOfCountries.Add(JsonConvert.DeserializeObject<Country>(country.ToString()));
            }
        }
        catch (Exception)
        {
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;
    public void OnPropertyChanged(string prop = "")
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(prop));
        }
    }
  }
}
using CvCinemaProject.Models;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Net.Http;
using System.Text;

namespace CvCinemaProject.ViewModels
{
  class MoviePremiersListViewModel
  {
```

```
        ObservableCollection<MovieInfo> listOfMoviePremiere = new
ObservableCollection<MovieInfo>();
        public ObservableCollection<MovieInfo> ListOfMoviePremiere { get { return
listOfMoviePremiere; } }

        public async void LoadData()
        {
            string url = "https://api.kino-
teatr.ua/rest/city/24/shows?apiKey=apiKey&size=10&detalization=FULL";
            try
            {
                HttpClient client = new HttpClient();
                client.BaseAddress = new Uri(url);
                var response = await client.GetAsync(client.BaseAddress);
                response.EnsureSuccessStatusCode();

                var content = await response.Content.ReadAsStringAsync();
                JObject o = JObject.Parse(content);

                var premiereStr = o.SelectToken(@"$.content");
                foreach (var premiere in premiereStr)
                {

                    var mp =
JsonConvert.DeserializeObject<MoviePremiere>(premiere.ToString());
                    string murl = "https://api.kino-teatr.ua/rest/film/" + mp.Film_Id +
"?apiKey=apiKey";
                    HttpClient client2 = new HttpClient();
                    client2.BaseAddress = new Uri(murl);
                    var response2 = await client2.GetAsync(client2.BaseAddress);
                    response2.EnsureSuccessStatusCode();

                    var content2 = await response2.Content.ReadAsStringAsync();
                    JObject ob = JObject.Parse(content2);

listOfMoviePremiere.Add(JsonConvert.DeserializeObject<MovieInfo>(ob.ToString(
)));
                }

            }
            catch (Exception)
            {
            }
        }
    }
```

```
}
using CvCinemaProject.Models;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Net.Http;
using System.Text;

namespace CvCinemaProject.ViewModels
{
   class MoviesListViewModel
   {
      ObservableCollection<MovieInfo> listOfMovies = new
ObservableCollection<MovieInfo>();
      public ObservableCollection<MovieInfo> ListOfMovies { get { return
listOfMovies; } }

      public async void LoadData()
      {
         string url = "https://api.kino-
teatr.ua/rest/films/premieres?apiKey=apiKey&size=100";
         try
         {
            HttpClient client = new HttpClient();
            client.BaseAddress = new Uri(url);
            var response = await client.GetAsync(client.BaseAddress);
            response.EnsureSuccessStatusCode();

            var content = await response.Content.ReadAsStringAsync();
            JObject o = JObject.Parse(content);

            var movieStr = o.SelectToken(@"$.content");
            foreach (var movie in movieStr)
            {

listOfMovies.Add(JsonConvert.DeserializeObject<MovieInfo>(movie.ToString()));
            }

         }
         catch (Exception)
         {
         }
      }
```

```
    }
}
using CvCinemaProject.Models;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Net.Http;
using System.Text;

namespace CvCinemaProject.ViewModels
{
    class MovieTrailerListViewModel
    {
        ObservableCollection<MovieTrailer> listOfMovieTrailers = new
ObservableCollection<MovieTrailer>();
        public ObservableCollection<MovieTrailer> ListOfMovieTrailers { get { return
listOfMovieTrailers; } }

        public async void LoadData()
        {
            string url = "https://api.kino-
teatr.ua/rest/film/51101/trailers?apiKey=apiKey&size=10";

            try
            {
                HttpClient client = new HttpClient();
                client.BaseAddress = new Uri(url);
                var response = await client.GetAsync(client.BaseAddress);
                response.EnsureSuccessStatusCode();

                var content = await response.Content.ReadAsStringAsync();
                JObject o = JObject.Parse(content);

                var movieTrailerStr = o.SelectToken(@"$.content");
                foreach (var movieTrailer in movieTrailerStr)
                {

listOfMovieTrailers.Add(JsonConvert.DeserializeObject<MovieTrailer>(movieTraile
r.ToString()));
                }
            }
            catch (Exception)
            {
```

```
                throw;
            }
        }
    }
}
using CvCinemaProject.Models;
using MediaManager;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Net.Http;
using System.Text;
using System.Windows.Input;
using Xamarin.Forms;

namespace CvCinemaProject.ViewModels
{
    class MovieViewModel : INotifyPropertyChanged
    {
        private int id;
        private string title;
        private int? year;
        public decimal? rating;
        public string description;
        public string trailerUrl;
        public int Id
        {
            get { return id; }
            private set
            {
                id = value;
                OnPropertyChanged("Id");
            }
        }
        public string Title
        {
            get { return title; }
            private set
            {
                title = value;
                OnPropertyChanged("Title");
            }
```

```
}
public int? Year
{
    get { return year; }
    private set
    {
        year = value;
        OnPropertyChanged("Year");
    }
}
public decimal? Rating
{
    get { return rating; }
    private set
    {
        rating = value;
        OnPropertyChanged("Rating");
    }
}
public string Description
{
    get { return description; }
    private set
    {
        description = value;
        OnPropertyChanged("Description");
    }
}
public string TrailerUrl
{
    get { return trailerUrl; }
    private set
    {
        trailerUrl = value;
        OnPropertyChanged("TrailerUrl");
    }
}
public async void LoadData(int id)
{
    string url = "https://api.kino-teatr.ua/rest/film/" + id + "?apiKey=apiKey";

    try
    {
        HttpClient client = new HttpClient();
        client.BaseAddress = new Uri(url);
```

```
            var response = await client.GetAsync(client.BaseAddress);
            response.EnsureSuccessStatusCode();

            var content = await response.Content.ReadAsStringAsync();
            JObject o = JObject.Parse(content);

            var movieInfo =
JsonConvert.DeserializeObject<MovieInfo>(o.ToString());

            this.Id = movieInfo.Id;
            this.Title = movieInfo.Title;
            this.Year = movieInfo.Year;
            this.Rating = movieInfo.Rating;
            //this.Description = movieInfo.Description;
        }
        catch (Exception)
        {
        }
    }
    public ICommand LoadDataCommand { get; private set; }
    public MovieViewModel()
    {
        this.LoadDataCommand = new Command(PlayButton);
    }

    void PlayButton()
    {
            CrossMediaManager.Current.Play(TrailerUrl);
    }

    public async void LoadTrailer(int id)
    {
        string url = "https://api.kino-teatr.ua/rest/film/" + id +
"/trailers?apiKey=apiKey&size=1";

        try
        {
            HttpClient client = new HttpClient();
            client.BaseAddress = new Uri(url);
            var response = await client.GetAsync(client.BaseAddress);
            response.EnsureSuccessStatusCode();

            var content = await response.Content.ReadAsStringAsync();
            JObject o = JObject.Parse(content);
```

```
            var movieStr = o.SelectToken(@"$.content[0]");

            var movieInfo =
JsonConvert.DeserializeObject<MovieTrailer>(movieStr.ToString());

            this.TrailerUrl = movieInfo.Url;
        }
        catch (Exception)
        {
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
    public void OnPropertyChanged(string prop = "")
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(prop));
        }
    }
  }
}
using CvCinemaProject.Models;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Net.Http;
using System.Text;

namespace CvCinemaProject.ViewModels
{
  class SalesListViewModel
  {
    ObservableCollection<Sale> listOfSales = new ObservableCollection<Sale>();
    public ObservableCollection<Sale> ListOfSales { get { return listOfSales; } }
    public async void LoadData(int id)
    {
        string url = "https://api.kino-teatr.ua/rest/cinema/" + id +
"/halls?apiKey=apiKey";

        try
        {
            HttpClient client = new HttpClient();
            client.BaseAddress = new Uri(url);
```

```
            var response = await client.GetAsync(client.BaseAddress);
            response.EnsureSuccessStatusCode();

            var content = await response.Content.ReadAsStringAsync();
            JObject o = JObject.Parse(content);

            var salesStr = o.SelectToken(@"$.content");
            foreach (var sale in salesStr)
            {
               listOfSales.Add(JsonConvert.DeserializeObject<Sale>(sale.ToString()));
            }

         }
         catch (Exception)
         {
         }
      }
   }
}
```

Додаток 3

```
using CvCinemaProject.Pages.ActorsListPage;
using CvCinemaProject.Pages.CitiesPage;
using CvCinemaProject.Pages.CountriesPage;
using CvCinemaProject.Pages.FirstPage;
using CvCinemaProject.Pages.HomePage;
using CvCinemaProject.Pages.MoviePage;
using CvCinemaProject.Pages.MoviesListPage;
using CvCinemaProject.Pages.MovieTrailerListPage;
using CvCinemaProject.Pages.SecondPage;
using System;
using System.Collections.Generic;
using System.Text;
using Xamarin.Forms;

namespace CvCinemaProject
{
   class MasterPage : ContentPage
   {
      public ListView ListView { get { return listView; } }

      ListView listView;

      public MasterPage()
```

```
{
    var masterPageItems = new List<MasterPageItem>();
    masterPageItems.Add(new MasterPageItem
    {
        Title = "Home",
        IconSource = "home.png",
        TargetType = typeof(HomePage)
    });
    masterPageItems.Add(new MasterPageItem
    {
        Title = "Movies",
        IconSource = "movieIcon2.png",
        TargetType = typeof(MoviesListPage)
    });
    masterPageItems.Add(new MasterPageItem
    {
        Title = "Actors",
        IconSource = "actor.png",
        TargetType = typeof(ActorsListPage)
    });
    masterPageItems.Add(new MasterPageItem
    {
        Title = "Cities",
        IconSource = "citiesIcon.png",
        TargetType = typeof(CitiesPage)
    });

    listView = new ListView
    {
        Header = new StackLayout {
            Padding = 30,
            BackgroundColor = Color.FromHex("#7e57c2"),
            Children = {
                new Label {
                    Margin = 10,
                    VerticalTextAlignment = TextAlignment.Center,
                    HorizontalTextAlignment = TextAlignment.Center,
                    Text = "CvCinema Menu",
                    TextColor = Color.FromHex("#ffffff"),
                    FontSize = 20,
                    FontAttributes = FontAttributes.Bold
                }
            }
        },
        ItemsSource = masterPageItems,
```

```
            ItemTemplate = new DataTemplate(() => {
                var grid = new Grid { Padding = new Thickness(5, 10, 10, 10) };
                grid.ColumnDefinitions.Add(new ColumnDefinition { Width = new
GridLength(60) });
                grid.ColumnDefinitions.Add(new ColumnDefinition { Width =
GridLength.Star });

                var image = new Image();
                image.SetBinding(Image.SourceProperty, "IconSource");
                var label = new Label { VerticalOptions =
LayoutOptions.FillAndExpand };
                label.SetBinding(Label.TextProperty, "Title");

                grid.Children.Add(image);
                grid.Children.Add(label, 1, 0);

                return new ViewCell { View = grid };
            }),

            Footer = new StackLayout
            {
                Margin = new Thickness(0, 450, 0, 0),
                Padding = 22,
                BackgroundColor = Color.FromHex("#7e57c2")
            },
            //BackgroundColor = Color.FromHex("#027BCE"),
            SeparatorVisibility = SeparatorVisibility.Default,
            SeparatorColor = Color.Gray
        };
        BackgroundColor = Color.FromHex("F5F56");
        IconImageSource = "hamburger.png";
        Title = "Personal Organiser";
        Content = new StackLayout
        {
            Children = { listView }
        };
    }
  }
}
```

Додаток 4

```
using CvCinemaProject.ViewModels;
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace CvCinemaProject.Pages.HomePage
{
   [XamlCompilation(XamlCompilationOptions.Compile)]
   public partial class HomePage : ContentPage
   {
      MoviePremiersListViewModel viewModel;
      public HomePage()
      {
         InitializeComponent();
         viewModel = new MoviePremiersListViewModel();
         viewModel.LoadData();
         this.BindingContext = viewModel;
      }

      public async void GoToMovieInfoPage(object sender,
SelectionChangedEventArgs e)
      {
         var myListView = (CollectionView)sender;
         var myItem = myListView.SelectedItem;
         var idProp = myItem.GetType().GetProperty("Id");
         int id = (int)(idProp.GetValue(myItem, null));

         await Navigation.PushAsync(new MoviePage.MoviePage(id));
      }
   }
}
<?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:d="http://xamarin.com/schemas/2014/forms/design"
          xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
          mc:Ignorable="d"
          x:Class="CvCinemaProject.Pages.HomePage.HomePage">
   <CollectionView ItemsSource="{Binding ListOfMoviePremiere}"
                ItemsLayout="HorizontalList"
                SelectionMode="Single"
                SelectionChanged="GoToMovieInfoPage">
```

```
        <CollectionView.ItemTemplate>
            <DataTemplate>
                <StackLayout>
                    <Frame HasShadow="True"
                        BorderColor="DarkGray"
                        CornerRadius="5"
                        Margin="20"
                        HeightRequest="300"
                        HorizontalOptions="Center"
                        VerticalOptions="CenterAndExpand">
                        <StackLayout>
                            <Label Text="{Binding Title}"
                                FontAttributes="Bold"
                                FontSize="Large"
                                HorizontalOptions="Center"
                                VerticalOptions="Center" />
                            <!--<Image Source="{Binding ImageUrl}"
                                Aspect="AspectFill"
                                HeightRequest="150"
                                WidthRequest="150"
                                HorizontalOptions="Center" />-->
                            <Label Text="{Binding Year}"
                                HorizontalOptions="Center" />
                            <Label Text="{Binding Rating}"
                                FontAttributes="Italic"
                                HorizontalOptions="Center"
                                MaxLines="5"
                                LineBreakMode="TailTruncation" />
                        </StackLayout>
                    </Frame>
                </StackLayout>
            </DataTemplate>
        </CollectionView.ItemTemplate>
    </CollectionView>
</ContentPage>
```

Додаток 5

```
using CvCinemaProject.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using CvCinemaProject.Pages.FirstPage;
```

```
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using System.Windows.Input;

namespace CvCinemaProject.Pages.ActorsListPage
{
   [XamlCompilation(XamlCompilationOptions.Compile)]
   public partial class ActorsListPage : ContentPage
   {
      public ICommand ToActorInfoPage { protected set; get; }

      ActorsListViewModel viewModel;

      public ActorsListPage()
      {
         InitializeComponent();
         viewModel = new ActorsListViewModel();
         viewModel.LoadData();

         //this.ToActorInfoPage = new Command(GoToActorInfoPage);
         this.BindingContext = viewModel;
      }
      public async void GoToActorInfoPage(object sender,
ItemTappedEventArgs e)
      {
         var myListView = (ListView)sender;
         var myItem = myListView.SelectedItem;
         var idProp = myItem.GetType().GetProperty("Id");
         int id = (int)(idProp.GetValue(myItem, null));

         await Navigation.PushAsync(new FirstPage.FirstPage(id));
      }
   }
}
<?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:d="http://xamarin.com/schemas/2014/forms/design"
          xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
          mc:Ignorable="d"
          x:Class="CvCinemaProject.Pages.ActorsListPage.ActorsListPage">
   <StackLayout>
      <ListView HasUnevenRows="True"
```

```
            ItemsSource="{Binding ListOfActors}"
            ItemTapped="GoToActorInfoPage">
          <ListView.ItemTemplate>
            <DataTemplate>
              <ViewCell>
                <ViewCell.View>
                  <StackLayout>
                    <Label Text="{Binding First_Name_Orig}" FontSize="18"
/>
                    <Label Text="{Binding Last_Name_Orig}" />
                    <Label Text="{Binding BirthDate}" />
                    <Label Text="{Binding Rating}"/>
                    <!--<Label Text="{Binding Description}"/>-->
                  </StackLayout>
                </ViewCell.View>
              </ViewCell>
            </DataTemplate>
          </ListView.ItemTemplate>
        </ListView>
      </StackLayout>
</ContentPage>
using CvCinemaProject.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace CvCinemaProject.Pages.CinemasPage
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class CinemasPage : ContentPage
    {
        CinemasListViewModel viewModel;
        public CinemasPage(int id)
        {
            InitializeComponent();
            viewModel = new CinemasListViewModel();
            viewModel.LoadData(id);
            this.BindingContext = viewModel;
        }
        async void GoToSalesPage(object sender, EventArgs e)
```

```
        {
            var myListView = (ListView)sender;
            var myItem = myListView.SelectedItem;
            var idProp = myItem.GetType().GetProperty("Id");
            int id = (int)(idProp.GetValue(myItem, null));

            await Navigation.PushAsync(new SalesPage.SalesPage(id));
        }
    }
}
<?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:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             mc:Ignorable="d"
             x:Class="CvCinemaProject.Pages.CinemasPage.CinemasPage">
    <StackLayout>
        <ListView HasUnevenRows="True"
                  ItemsSource="{Binding ListOfCinemas}"
                  ItemTapped="GoToSalesPage">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <ViewCell.View>
                            <StackLayout>
                                <Label Text="{Binding Name}" FontSize="18" />
                                <Label Text="{Binding Site}" />
                                <Label Text="{Binding Phone}" />
                                <Label Text="{Binding Address}" />
                                <Label Text="{Binding Url}" />
                            </StackLayout>
                        </ViewCell.View>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </StackLayout>
</ContentPage>
using CvCinemaProject.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace CvCinemaProject.Pages.CitiesPage
{
   [XamlCompilation(XamlCompilationOptions.Compile)]
   public partial class CitiesPage : ContentPage
   {
      CitiesListViewModel viewModel;
      public CitiesPage()
      {
         InitializeComponent();
         viewModel = new CitiesListViewModel();
         viewModel.LoadData();
         this.BindingContext = viewModel;
      }
      async void GoToCinemasPage(object sender, EventArgs e)
      {
         var myListView = (ListView)sender;
         var myItem = myListView.SelectedItem;
         var idProp = myItem.GetType().GetProperty("Id");
         int id = (int)(idProp.GetValue(myItem, null));

         await Navigation.PushAsync(new CinemasPage.CinemasPage(id));
      }
   }
}
<?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:d="http://xamarin.com/schemas/2014/forms/design"
          xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
          mc:Ignorable="d"
          x:Class="CvCinemaProject.Pages.CitiesPage.CitiesPage">
   <StackLayout>
      <ListView HasUnevenRows="True"
           ItemsSource="{Binding ListOfCities}"
           ItemTapped="GoToCinemasPage">
         <ListView.ItemTemplate>
            <DataTemplate>
               <ViewCell>
                  <ViewCell.View>
```

```
                    <StackLayout>
                        <Label Text="{Binding Name}" FontSize="18" />
                    </StackLayout>
                </ViewCell.View>
            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
</StackLayout>
</ContentPage>
using CvCinemaProject.ViewModels;
using CvCinemaProject.Pages.MoviePage;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
using Xamarin.Essentials;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace CvCinemaProject.Pages.MoviesListPage
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class MoviesListPage : ContentPage
    {
        public ICommand TapCommand => new
Command<string>(OpenBrowser);
        MoviesListViewModel viewModel;
        public MoviesListPage()
        {
            InitializeComponent();
            viewModel = new MoviesListViewModel();
            viewModel.LoadData();
            this.BackgroundColor = Color.FromHex("#F5F5F6");
            this.BindingContext = viewModel;
        }
        public async void GoToMovieInfoPage(object sender,
ItemTappedEventArgs e)
        {
            var myListView = (ListView)sender;
            var myItem = myListView.SelectedItem;
            var idProp = myItem.GetType().GetProperty("Id");
            int id = (int)(idProp.GetValue(myItem, null));
```

```
            await Navigation.PushAsync(new MoviePage.MoviePage(id));
        }
        void OpenBrowser(string url)
        {
            Device.OpenUri(new Uri(url));
        }
    }
}
<?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:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             mc:Ignorable="d"
             x:Class="CvCinemaProject.Pages.MoviesListPage.MoviesListPage">
    <StackLayout>
        <ListView HasUnevenRows="True"
                  SeparatorColor="Gray"
                  SeparatorVisibility="Default"
                  ItemsSource="{Binding ListOfMovies}"
                  ItemTapped="GoToMovieInfoPage">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <ViewCell.View>
                            <StackLayout Padding="10">
                                <StackLayout Orientation="Horizontal">
                                    <Label Text="Назва фільму: " FontSize="18"/>
                                    <Label Text="{Binding Title}" FontSize="18"
FontAttributes="Bold" />
                                </StackLayout>
                                <StackLayout Orientation="Horizontal">
                                    <Label Text="Рік прем'єри: " />
                                    <Label Text="{Binding Year}" />
                                </StackLayout>
                                <StackLayout Orientation="Horizontal">
                                    <Label Text="Інформація на сайті: " />
                                    <Label>
                                        <Label.FormattedText>
                                            <FormattedString>
                                                <Span Text="{Binding Url}"
                                                      TextColor="Blue"
                                                      TextDecorations="Underline">
```

```
                                    <Span.GestureRecognizers>
                                        <TapGestureRecognizer
Command="{Binding TapCommand}"

CommandParameter="{Binding Url}" />
                                    </Span.GestureRecognizers>
                                </Span>
                            </FormattedString>
                        </Label.FormattedText>
                    </Label>
                </StackLayout>
                <StackLayout Orientation="Horizontal">
                    <Label Text="Рейтинг: " />
                    <Label Text="{Binding Rating}" />
                </StackLayout>
                <StackLayout Orientation="Horizontal">
                    <Label Text="Гласи: " />
                    <Label Text="{Binding Votes}" />
                </StackLayout>
            </StackLayout>
        </ViewCell.View>
    </ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</StackLayout>
</ContentPage>
using CvCinemaProject.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace CvCinemaProject.Pages.SalesPage
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class SalesPage : ContentPage
    {
        SalesListViewModel viewModel;
        public SalesPage(int id)
        {
```

```
            InitializeComponent();
            viewModel = new SalesListViewModel();
            viewModel.LoadData(id);
            this.BindingContext = viewModel;
        }
    }
}
<?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:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             mc:Ignorable="d"
             x:Class="CvCinemaProject.Pages.SalesPage.SalesPage">
    <StackLayout>
        <ListView HasUnevenRows="True"
             ItemsSource="{Binding ListOfSales}">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <ViewCell.View>
                            <StackLayout>
                                <Label Text="{Binding Name}" FontSize="18" />
                            </StackLayout>
                        </ViewCell.View>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </StackLayout>
</ContentPage>
```

Додаток 6

```
using CvCinemaProject.ViewModels;
using MediaManager;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;
```

```
namespace CvCinemaProject.Pages.MoviePage
{
   [XamlCompilation(XamlCompilationOptions.Compile)]
   public partial class MoviePage : ContentPage
   {
      MovieViewModel viewModel;

      public MoviePage(int id)
      {
         InitializeComponent();
         viewModel = new MovieViewModel();
         viewModel.LoadData(id);
         viewModel.LoadTrailer(id);
         this.BindingContext = viewModel;
      }
      private void StopButton(object sender, EventArgs e)
      {
         CrossMediaManager.Current.Stop();
      }
      private void ContinueButton(object sender, EventArgs e)
      {
         CrossMediaManager.Current.PlayPause();
      }
   }
}
<?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:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             mc:Ignorable="d"
             x:Class="CvCinemaProject.Pages.MoviePage.MoviePage"
             xmlns:mm="clr-
namespace:MediaManager.Forms;assembly=MediaManager.Forms">
   <StackLayout>
      <Label FontSize="Medium" FontAttributes="Bold" Text="{Binding Id}"
HorizontalOptions="Center" />
      <Label FontSize="Medium" FontAttributes="Bold" Text="{Binding
Title}" HorizontalOptions="Center" />
      <Label FontSize="Medium" FontAttributes="Bold" Text="{Binding
Year}" HorizontalOptions="Center"/>
      <Label FontSize="Medium" FontAttributes="Italic" Text="{Binding
Rating}" HorizontalOptions="Center"/>
      <mm:VideoView HeightRequest="202"
```

```
                WidthRequest="202"
                ShowControls="True"/>
        <StackLayout Orientation="Horizontal">
            <Button Text="Play"  Command="{Binding LoadDataCommand}"
BackgroundColor="Silver" TextColor="White"/>
            <Button Text="PauseContinue" Clicked="ContinueButton"
BackgroundColor="Silver" TextColor="White"/>
            <Button Text="Stop" Clicked="StopButton"
BackgroundColor="Silver" TextColor="White"/>
        </StackLayout>
        <Label FontSize="Medium" Text="Опис" HorizontalOptions="Center"/>
        <Label BackgroundColor="GhostWhite"
            FontSize="Medium"
            FontAttributes="Italic"
            Text="{Binding Description}"
            HorizontalOptions="Center"/>
    </StackLayout>
</ContentPage>
```