НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
"___"_____2016 р.

Дипломна робота

першого (бакалаврського) **рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності 7.05010102, 8.05010102 Інформаційні технології проектування
7.05010103, 8.05010103 Системне проектування
(код та назва напряму підготовки)

на тему: Розробка крос-платформних додатків з використанням фреймворку Xamarin

Виконав (-ла): студент (-ка) 4 курсу, групи ДА-22
(шифр групи)

Сергеєв Єгор Ігорович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, Романов В.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль ст. викладач Британов О.А. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2016 року

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший (Бакалаврський)
(перший (бакалаврський), другий (магістерський) або спеціаліста)

Спеціальність 7.05010102, 8.05010102 Інформаційні технології проектування
7.05010103, 8.05010103 Системне проектування

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

«___»_____2016 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Сергеєву Єгору Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка крос-платформних додатків з використанням фреймворку Xamarin
керівник проекту (роботи) Романов Валерій Володимирович, доцент ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 12 травня 2016 р. № 50-ст

2. Строк подання студентом проекту (роботи) 10.06.2016

3. Вихідні дані до проекту (роботи) _____

Список фреймворків для аналізу та порівняння із Xamarin: Onsen UI, Ionic, PhoneGap, NativeScript, Sencha Touch.

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

1. Поняття крос-платформної розробки. Огляд існуючих рішень;
2. Дослідження технології Xamarin;
3. Методика написання програм на C# з використанням Xamarin.

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

1. Принцип роботи Xamarin.iOS та Xamarin.Android – плакат.
2. Інфраструктура Mono – плакат.
3. Архітектура додатку – плакат.

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 01.02.2016

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2016	
2	Збір інформації	15.02.2016	
3	Дослідження існуючих рішень для написання крос-платформних додатків	28.02.2016	
4	Дослідження технології Xamarin Native	10.03.2016	
5	Дослідження технології Xamarin.Forms	15.03.2016	
6	Розробка додатку	25.03.2016	
7	Тестування додатку	25.04.2016	
8	Оформлення дипломної роботи	31.05.2016	
9	Отримання допуску до захисту та подача роботи в ДЕК	08.06.2016	

Студент _____ Сергеєв Є.І.
(підпис) (ініціали, прізвище)

Керівник роботи _____ Романов В.В.
(підпис) (ініціали, прізвище)

---

* Консультантом не може бути зазначено керівника дипломного проекту (роботи).

АНОТАЦІЯ

бакалаврської дипломної роботи Сергеєва Єгора Ігоровича

на тему: “Розробка крос-платформних додатків з використанням фреймворка Xamarin”

Дипломна робота присвячена розгляду фреймворка Xamarin, крос-платформній розробці з використанням мови програмування C#. Результатом роботи стало дослідження особливостей розробки додатків з використанням Xamarin, зроблений професійний огляд основних підходів, що використовуються при розробці додатків. Були досліджені такі підходи, як Xamarin.Android, Xamarin.Forms, Xamarin.iOS, що дозволяють будувати додатки під різні операційні системи, використовуючи спільну логіку додатку. Було виділено основні недоліки та переваги фреймворка Xamarin та його підходів в розробці крос-платформних додатків. Розглянуто основні патерни розробки та рекомендації, щодо побудови архітектури таких додатків.

Загальний об’єм роботи 74 сторінки, 19 рисунків, 6 таблиць, 17 посилань.

Ключові слова: крос-платформна розробка, C#, Xamarin, iOS, Android, Windows Phone, Universal Windows Platform.

АННОТАЦИЯ

бакалаврской дипломной работе Сергеева Егора Игоревича

на тему: “Разработка кросс-платформенных приложений с использованием фреймворка Xamarin”

Дипломная работа посвящена обзору фреймворка Xamarin, кросс-платформенной разработке с использованием языка программирования C#. Результатом работы стало исследование особенностей разработки приложений с использованием Xamarin, обзор основных подходов, которые используются при разработке приложений. Были исследованы такие подходы, как Xamarin.Android, Xamarin.Forms, Xamarin.iOS, которые позволяют разрабатывать приложения под разные операционные системы, используя общую логику приложения. Были выделены основные недостатки и достоинства фреймворка Xamarin и его подходов в разработке кросс-платформенных приложений. Рассмотрено основные паттерны разработки и рекомендации по построению архитектуры таких приложений.

Общий объём работы: 74 страницы, 19 рисунков, 6 таблиц, 17 ссылок.

Ключевые слова: кросс-платформенная разработка, C#, Xamarin, iOS, Android, Windows Phone, Universal Windows Platform.

ABSTRACT

of the bachelor's thesis of Serheiev Yehor Ihorevich

"Building cross-platform applications using Xamarin framework"

This work is devoted to overview of Xamarin framework, cross-platform develop using C# programming language. Researched main developing features using Xamarin framework and main approaches review using in Xamarin applications development. Approaches such as Xamarin.Android, Xamarin.Forms, Xamarin.iOS were researched, which allows developing for different operating systems. Advantages and disadvantages of Xamarin framework and it's developing approaches were highlighted. Considered main patterns and recommendations for building architecture of Xamarin applications.

The total amount of work is: 74 pages, 19 figures, 6 tables, 17 references.

Keywords: cross-platform developing, C#, Xamarin, iOS, Android, Windows Phone, Universal Windows Platform.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

КПР	Крос-платформна розробка
КП	Крос-платформність
WP	Windows Phone
ОС	Операційна система
ПП	Програмний продукт
СВ	Середовище виконання

ВСТУП

Коли постає питання, яким чином будувати iOS чи Android додатки, багато людей, в першу чергу, згадують такі мови програмування як Swift, Java, або Objective-C. Проте в цій роботі буде розглянуто зовсім іншу технологію.

Xamarin – це унікальний фреймворк який дозволяє будувати такі додатки з використанням однієї мови програмування – C#. З використанням данного фреймворка ми компілюємо (не інтерпретуємо, як, наприклад, при використанні аналогічних JavaScript фреймворків) так званий “нативний” код для цих операційних систем(ОС).

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.

Основною задачею даного дипломного проекту – ознайомлення з фреймворком Xamarin, показати усі його переваги та недоліки, порівняти даний фреймворк з іншими фреймворками, що створені для написання програм під ці ОС. Також буде написана програма, яка демонструє роботу даного фреймворка на різних ОС – Android, WP.

В першому розділі буде розглянуто вже існуючі рішення, для побудови додатків під розглянуті ОС, наприклад Ionic, що використовує JavaScript та Angular, а також RoboVM, який використовує мову програмування Java. Також в першому розділі надано огляд й інших фреймворків, таких як PhoneGap чи Corona SDK, проте їх не буде розглянуто так детально, як Ionic та RoboVM.

В другому розділі наведено та описано основні можливості даного фреймворку. Також буде показано як Xamarin працює всередині (“під капотом”), а також наведено основні способи побудови додатків з використанням Xamarin – Xamarin.iOS, Xamarin.Android та Xamarin.Forms. Також 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#.

В третьому розділі детально розглянуто основні принципи побудови додатків з використанням Xamarin, паттерни, основні конструкції, функції, архітектурні рішення, щодо структури додатку, наприклад, використання паттерна MVVM в WP додатках чи впровадження багатошарової архітектури при проектуванні структури програмного продукту.

1. ПОНЯТТЯ КРОС-ПЛАТФОРМНОЇ РОЗРОБКИ. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Для написання програм під кожну з ОС потрібно витрачати багато ресурсів, а також часу. Оскільки кожна ОС має різні API інтерфейси, специфічну логіку роботи, нативну мову програмування. Для того, щоб створити програмний продукт (ПП), який буде доступний для різних ОС компанії-розробнику потрібно тримати штат програмістів, спеціалізованих для різних ОС, наприклад, Java команду для написання Android програм, C# – команду для написання програм під Windows Phone і т.д. Тому є фреймворки, які дозволяють позбутися від такої залежності і писати код під різні ОС, використовуючи одну мову програмування.

Звичайно, у такого підходу є свої недоліки. По-перше, програми, які написані на нативній мові програмування для даної ОС працюють швидше. По-друге, потрібно знайти спеціалістів, які знають усі платформи, які підтримує даний фреймворк, звісно спеціаліст, який спеціалізується на одній конкретній платформі буде знати саме цю платформу набагато краще.

Проте, при такому підході можна позначити й деякі переваги. Першою такою перевагою, звісно є те, що розробникам не потрібно міняти бізнес логіку ПП, оскільки код пишеться 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, специфічного для платформи[17]

Другою перевагою такого підходу є те, що час, на створення ПП значно скорочується, оскільки потрібно писати менше коду, скорочується й кількість конфліктів між командами.

В даному розділі будуть розглянуті основні поняття крос-платформної розробки, а також – буде проведено розгляд уже існуючих рішень для крос-платформної розробки (КПР) для цих ОС.

1.1 Поняття крос-платформної розробки

Крос-платформність (КП) — властивість програмного забезпечення працювати більш ніж на одній ОС, або апаратній платформі, та технології, що дозволяють досягти такої властивості. Даний підхід дозволяє суттєво скоротити витрати на розробку нового або адаптацію існуючого програмного забезпечення.

Залежно від засобів реалізації поділяється на КС на рівні мов програмування, КП середовища виконання, ОС та апаратної платформи. Основними технологіями досягнення КП є:

1. КП мов програмування

   Даний тип КП досягається шляхом забезпечення незалежності програмного коду від платформи. КП є більшість сучасних високорівневих мов програмування, для яких реалізовані транслятори, що можуть виконуватись на різних платформах.

2. КП на рівні середовища виконання

   КП на рівні середовищ виконання забезпечується реалізацією в цих середовищах можливостей, необхідних програмам незалежно від платформи. Декларований набір таких можливостей прийнято називати «контрактом» — обов'язком, який покладається на середовище, щоб забезпечити виконання програми. Ці обов'язки реалізуються через інтерпретатор, файлові потоки, системні виклики, протоколи, віртуальну машину тощо.

3. КП на апаратному рівні

   КП на апаратному рівні досягається реалізацією однакових машинних команд та форматом їх представлення, систем переривань, механізмів адресації пам'яті, регістрів тощо. Може досягатись шляхом віртуалізації відповідних ресурсів та механізмів.

Розглянемо основні недоліки та переваги КП додатків:

1. Економія бюджету ― використання однієї технології і графічного інтерфейсу значно знижує кількість робочих годин і бюджет проекту;
2. Час розробки ― відсутність унікальних елементів інтерфейсу і одна технологічна платформа скорочує терміни розробки;
3. Підтримка і оновлення продукту ― додавання функціоналу або виправлення помилок відразу для всіх платформ;

4. Спільна логіка додатку — логіка додатку однаково працюватиме для всіх платформ. Написана і налагоджена логіка містить потенційно меншу кількість помилок і розбіжностей в своїй роботі;
5. Повільніша робота додатку;
6. Неможливість спільного використання унікальних особливостей платформи;
7. Незвичний для користувача інтерфейс.

1.2 Огляд аналогічних фреймворків, що використовують інші мови програмування

Xamarin – не єдиний фреймворк, який орієнтується на три ОС – WP, Android, iOS. Звичайно, є ще дуже багато аналогічних фреймворків, що дозволяють писати програми під ці ОС, деякі з них, звичайно, не підтримують усі ці три ОС, проте підтримують дві, проте навіть ці фреймворки слід описати, оскільки вони є альтернативою використанню C# та Xamarin. Досить популярними зараз є аналоги, що використовують Java та JavaScript в своїй роботі, оскільки ці мови популярні у всьому світі. Згідно статистики з сайту dou.ua ці мови по популярності в Україні займають відповідно перше та третє місце. Коротко опишемо найпопулярніші з цих фреймворків, а для деяких з них буде виділено спеціальний розділ, де вони будуть більш детально описані.

Найпопулярніші аналоги:

1. Ionic

   За останні кілька років, Ionic зарекомендував себе як лідер в області розвитку гібридних мобільних додатків. Команда Ionic зберігає структуру оновлення шляхом адаптації до останніх тенденцій, випереджаючи конкурентів. Його найближчі конкуренти мають комерційну ліцензію, в той час як Ionic безкоштовний для користування з відкритим вихідним кодом, крім того, його екосистема

стала настільки велика, що ви можете легко знайти тонни ресурсів, щоб почати роботу в найкоротші терміни. Мова програмування – JavaScript[4].

2. Onsen UI

   Onsen UI є відносно новим фреймворком, але є основним JavaScript конкурентом для Ionic. Onsen UI – проект з відкритим кодом та ліцензією APACHE. Onsen UI також використовує AngularJS директиви і фреймворк Topcoat для більшості компонентів графічного інтерфейсу. Для любителів JQuery, даний фреймворк поставляється упакованим з компонентами на основі JQuery. Onsen UI містить велику колекцію готових до використання компонентів, що дозволяє створювати мобільні, планшетні, а також настільні версії своїх додатків, використовуючи HTML5 і JavaScript і використовуючи PhoneGap і Cordova для побудови нативних додатків[4].

3. Sencha Touch

   Sencha Touch – це JavaScript фреймворк, що працює на HTML5 та CSS3, надаючи API-інтерфейси, анімації і компонентів, які сумісні з існуючими мобільними платформами та браузерами. Sencha Touch підтримує як Cordova так і PhoneGap; ви можете скомпілювати додаток, і виставити його до відповідного маркету для кожної ОС. Крім того, Sencha Touch надає набір тем для iOS, Android, Blackberry, Windows Phone, Tizen, а також безліч інших ОС, щоб ваш додаток був схожий на нативний[4].

4. RoboVM

   Аналог Xamarin, що використовує мову програмуванняJava. В 2015 році даний проект був придбаний компанією Xamarin. Перший і поки що, єдиний фреймворк для Java, що дозволяє створювати додатки для

iOS та Android[4]. Детальніше цей фреймворк буде розглянуто в наступному розділі.

5. NativeScript

   NativeScript – це бібліотека, що дозволяє створювати КП додатки, використовуючи XML, CSS, JavaScript. NativeScript вирішує ту ж задачу, що і вже всім відомий Phonegap (створення КП додатків), але підходи у них різні. Phonegap використовує ядро браузера, щоб відобразити графічний інтерфейс (фактично створюється веб-сторінка), NativeScript використовує нативний рендерінг та елементи графічного інтерфейсу. Наступне важлива відмінність: щоб отримати доступ до камери, GPS і т. д. в PhoneGap необхідно встановлювати спеціальні плагіни, в той час як Native script надає доступ одразу. Варто підкреслити, що додатки можна писати для Android 4.2 і вище, і для iOS 7.1 і вище[4].

6. PhoneGap

   Безкоштовний фреймворк, що дозволяє створити додатки для мобільних пристроїв використовуючи JavaScript, HTML5 та CSS3, без необхідності знання нативних мов програмування (наприклад, Objective-C), під всі мобільні операційні системи (iOS, Android, WP і т. д.). Готовий додаток компілюється у вигляді пакетів для кожної мобільної операційної системи[4].

1.3 Ionic – JavaScript фреймворк

Ionic – це JavaScript фреймворк, що використовується для створення гібридних мобільних додатків. Перший реліз фреймворку пройшов в листопаді 2013. Ionic побудований на AngularJS і використовує його функціонал для оперування DOM, в той час як Ionic надає користувальницький інтерфейс[6].

Одна з яскравих переваг у використанні Ionic для розробки мобільних додатків – це те, що розробнику не потрібно використовувати нові для нього технології, а розробляти додатки з JavaScript та HTML5. Час розробника – це сьогодні найдорожчий ресурс. Якщо є можливість його скоротити, то потрібно використовувати цю можливість.

У пакет з Ionic входить набір інструментів і віджетів для створення додатків. Це анімовані сторінки, спливаючі вікна, кнопки, слайдери і багато іншого. Всі елементи адаптовані в першу чергу для роботи на мобільному пристрої, хоча по суті ніщо не забороняє використовувати Ionic для роботи з інтернетом.

Як бачимо, Ionic використовує власний набір UI елементів для розробки додатків під ці платформи, наприклад, ion-list, а для логіки додатку використовується AngularJS.

Тепер порівняємо Xamarin та Ionic:

1. Ionic з часу свого створення – повністю відкритий та безкоштовний фреймворк, Xamarin до 2016 року був платним фреймворком (поки його не придбала компанія Microsoft). Хоча і можна було створювати додатки безкоштовно, але вони мали певні обмеження на розмір коду та ін.;
2. Ionic використовує один і той же синтаксис для кожної ОС, тобто один раз написаний код буде працювати на різних ОС, це досягається за рахунок того, що Ionic використовує ядро браузера в своїй роботі. В Xamarin також можна зробити код, що буде працювати для всіх платформ однаково з використанням Xamarin.Forms та XAML, проте можливості такої програми обмежені, оскільки неможливо використовувати усі можливості кожної з ОС;

3. Код Xamarin – нативний код, оскільки кожна команда транслюється в нативний байт код, зрозумілий платформі. Використовуючи Ionic такого досягти неможливо;
4. Xamarin використовує компільовану мову C#, Ionic додатки використовують JavaScript для роботи, що є інтерпретованою мовою програмування, а отже працює повільніше, на додачу Ionic використовує CSS, що дуже повільно виконується, а CSS анімація сильно навантажує процесор телефону;
5. Даний пункт випливає з другого пункту. Написати додаток, використовуючи Ionic, можна набагато швидше.
6. Для написання додатку під iOS, використовуючи Xamarin, розробник обов'язково повинен мати Mac, використовуючи Ionic, використання Mac є необов'язковим.

Отже, як видно з даних пунктів, серед даних фреймворків немає явного лідера, оскільки Ionic в чомусь кращий ніж Xamarin та навпаки. Звичайно, можна повністю проігнорувати третій пункт порівняння, якщо замість Ionic використати NativeScript, проте спільнота даного фреймворка значно менша ніж спільнота Ionic.

1.4 RoboVM – JVM фреймворк

Ще одним цікавим фреймворком, який не слід ігнорувати є RoboVM. Даний фреймворк дозволяє писати iOS додатки з використанням JVM мов програмування. Потрібно зазначити, що даний фреймворк є не просто Java фреймворком, а JVM фреймворком, а отже, мови програмвання сумісні із JVM також сумісні і з фреймворком.

В основі RoboVM лежить ahead-of-time компілятор. Це інструмент, який може бути викликаний або з командного рядка, з таких інструментів, як Maven або Gradle, або також з IDE, таких як Eclipse, або IntelliJ Idea. Він приймає Java байт-код і переводить його в машинний код для конкретної операційної

системи і типу процесора. Як правило, це iOS і тип процесора ARM, але RoboVM також здатний генерувати код для Mac OSX і Linux, що працюють на x86 процесорах[5].

Підхід ahead-of-time дуже сильно відрізняється від того, який використовується в традиційних віртуальних машинах, як Oracle Hotspot. Такі JVM зазвичай читають Java байт-код під час виконання і якимось чином виконують інструкції віртуальної машини, що містяться в байткоді. Щоб прискорити цей процес, віртуальна машина використовує техніку, названу just-in-time компіляцією. Простіше кажучи, цей процес перетворює інструкції віртуальної машини в машинний код для поточного фізичного процесора в перший раз, коли метод викликається програмою[5].

Через технічні обмеження, just-in-time компіляція неможлива для додатків iOS. Єдиною альтернативою є використання інтерпритатора, який є занадто повільним і енергоємним, або використовувати ahead-of-time компіляцію, як в RoboVM (рис. 1.3). Процес компіляції ahead-of-time відбувається під час компіляції на машині розробника, так під час виконання на пристрої iOS або tvOS, що генерує машинний код, який працює швидше, ніж код, що побудований на Objective-C[5].

Потрібно зазначити, що підхід ahead-of-time використовується і в Xamarin.iOS, що ще раз підтверджує схожість підходів в розрібці програмного забезпечення, що застосовуються в даних фреймворках. На рис. 1.3 зображено принцип роботи RoboVM при компіляції бібліотек класів у повноцінний додаток, що працює під ОС iOS.

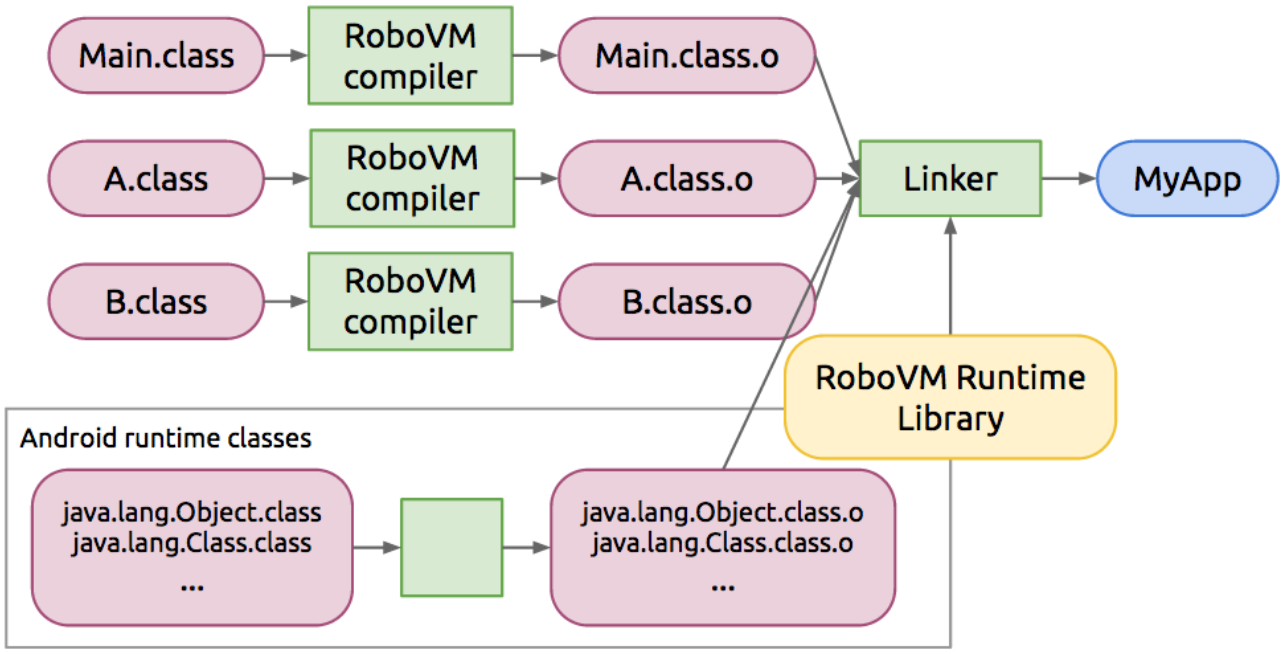

Рисунок 1.2 – Принцип роботи RoboVM[5]

Порівняння Xamarin та RoboVM:

1. Дані фреймворки мають однаковий принцип роботи, проте використовують різні мови програмування;
2. Для розробки на Android RoboVM використовує нативну мову програмування – Java;
3. Використовуючи RoboVM неможливо написати додаток для Windows та WP, проте, в майбутньому, можливо, таку можливість буде додано до даного фреймворка;
4. Як і для Xamarin, щоб писати iOS додатки, необхідно мати Mac;
5. Досить схожий синтаксис, оскільки використовуються дві схожі мови програмування.

Отже, як видно з даних пунктів, серед даних фреймворків немає явного лідера, оскільки RoboVM за принципами дуже схожий на Xamarin. Як в ситуації з Ionic, вибір фреймворка залежить лише від вподобань розробника.

1.5 Висновок до розділу 1

У даному розділі було роглянуто основні поняття КП, а також недоліки та переваги даного підходу для створення ПП.

Також, продемонстровано основні фреймворки, що дозволяють побудувати КП додатки аналогічні Xamarin. В основному такі фреймворки базуються на мові JavaScript, де більшість використовує ядро браузера для створення додатків, тобто фактично з таким підходом розробник не створює нативний додаток, а використовує усі можливості браузера для рендерингу HTML та CSS.

Деяким фреймворкам було приділено більше уваги, а саме Ionic та RoboVM. Досить цікавим для вивчення є фреймворк RoboVM, що використовує мову програмування Java, споріднену мові C#. Ще однією перевагою даного фреймворка є те, що його викупила компанія Xamarin, а, отже найкращі підходи будуть взяті з обох цих фреймворків, тобто будуть розвиватися паралельно та на одному принципі. Порівняно ці два фреймворка із Xamarin, виділено основні недоліки та переваги кожного, в порівнянні з Xamarin. Можна зробити висновок, що вибір фреймворка залежить лише від вподобань розробника, подобається JavaScript – обирайте Ionic чи інший аналог, що використовує дану мову програмування, програмуєте на Java, тоді RoboVM – ваш вибір. Ну і для C#, звісно, найкращий вибір – Xamarin.

2. ДОСЛІДЖЕННЯ ТЕХНОЛОГІЇ XAMARIN

В даному розділі буде розглянуто сам фреймворк Xamarin. Історію його створення, ідею, розробників, а також основні підходи, що реалізовані розробниками Xamarin, для створення КП додатків. Згідно висновків з минулого розділу, Xamarin є перспективним фреймворком, що не уступає своїм конкурентам.

Спочатку, буде розглянуто основні можливості Xamarin, які додатки можна зробити, використовуючи Xamarin, що саме лежить в основі Xamarin, розглянуто Mono.Android – реалізацію .NET framework для написання програм під Android.

Далі будуть більш детально описані такі підходи як Xamarin.Android, Xamarin.Forms, Xamarin.iOS, кожен з яких відрізняється в принципах побудови програм, оскільки дозволяє орієнтуватися під певну ОС. Найцікавішим є, звичайно, Xamarin.Forms, що дозволяє писати код одразу під три платформи – Android, iOS, WP, схоже на те, що дозволяє робити Ionic , проте цей підхід не є панацеєю від усього, адже цього підходу є свої недоліки, про які буде розказано у відповідному пункті, присвяченому Xamarin.Forms.

Може виникнути таке питання, навіщо взагалі використовувати C# та Xamarin, адже можна писати програми на Java чи на C++ та Swift, що є досить перспективними та цікавими мовами програмування. Дійсно, але мова C# має багату історію, цікаві конструкції, фреймворки, велику спільноту. Тому хотілося б написати основні та найбільш цікаві можливості даної мови:

1. Інкапсульовані сигнатури методів, звані делегатами, які підтримують типобезпечні повідомлення про події.
2. Властивості(properties), що виступають в ролі методів доступу для закритих змінних-членів.
3. Атрибути з декларативними метаданими про типах під час виконання.

4. Вбудовані коментарі XML-документації.
5. LINQ, що пропонує вбудовані можливості запитів в різних джерелах даних.
6. Підтримка асинхронних операцій
7. Використання TPL – Task Parallel Library в .NET
8. Використання широкого спектру класів .NET та фреймворків

Як видно, дана мова має багато гарних можливостей, вона схожа на C++ та легша в засвоєнні.

2.1 Технологія Xamarin

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

На початку XXI століття Microsoft вперше оголосила про створення .NET Framework – спеціальний фреймворк, що дозволяв писати програми під ОС Windows, використовуючи різні мови програмування. Мігель Де Іказа з компанії Ximian підняв питання про реалізацію даного фреймворку для платформи Linux. Ця ідея мала розвиток у вигляді проекту з відкритим вихідним кодом, що називається Mono, який був запущений 19 липня 2001 року. Однак, дана ідея хоч і мала розвиток, проте через декілька років, все ж-таки 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, що була створена у тому ж році і основним напрямком розробки даної компанії буде розробка під мобільні пристрої. Він також повідомив про те, що значна частина людей, що працювала над Mono перейшла до нової компанії.

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

Згодом було анонсовано створення нової версії Xamarin 2.0. Даний реліз влюкчав у себе два основні компоненти – Xamarin Studio, що заміняла MonoDevelop та плагін для інтеграції у Visual Studio – IDE від Microsoft для .Net Framework розробників. Завдяки цьому C# розробники змогли створювати свої додатки для iOS, Android аналогічно Windows та Windows Phone(WP).

У 2016 році компанії Xamarin та Microsoft анонсували те, що Microsoft підписала угоду, за якою вона повністю купує компанію Xamarin, хоча точну цифру за яку було придбано Xamarin не називають, в одному з журналів припустили, що сума варіюється десь в межах 400 - 500 мільйонів доларів. Однак, є й погані новини, як було зазначено у першому розділі, Xamarin колись придбала усі права на RoboVM, що є аналогічним фреймворком для Xamarin.Forms для Java. Однак, після придбання компанії Xamarin компанією Microsoft, Microsoft анонсували, що не будуть підтримувати даний продукт і усі контракти буде анульовано після 30 квітня 2017 року.

На конференції Build 2016 було анонсовано те, що Xamarin тепер є повністю безкоштовним фреймворком, підтримується Microsoft, також буде повністю реліцензійовано Mono.

Основними продуктами, що було створено за цей період часу є:

1. Xamarin Platform

   Платформа, що дозволяє використовувати нативні API конкретної операційної системи для написання додатків.
2. Xamarin.Forms

   Як вже зазначалося, аналог Ionic, але з використанням C# та мови XAML замість AngularJS для побудови UI.
3. Xamarin Test Cloud

Даний сервіс дозволяє розробнику протестувати свій додаток на різних телефонах, не маючи його. Достатньо лише мати підключення до інтернету ну і підписку(дана функція є платною), щоб тестувати свої додатки у великій базі смартфонів.

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.

5. Xamarin.Mac

   Фреймворк, що дозволяє писати додатки для Mac.

6. Xamarin Studio

   Основна IDE, що використовується для написання додатків під iOS чи Mac з використанням Xamarin.

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.

2.1.1 Платформа Mono

Mono – платформа розробки з відкритим вихідним кодом на основі платформи .NET Framework, дозволяє розробникам створювати КП додатки з поліпшеною продуктивностю розробника. Реалізація Mono .NET заснована на стандартах ECMA для C # і Common Language Infrastructure(CLI)[7].

Даний продукт раніше підтримувався такими компаніями як Novell, Xamarin, а тепер Microsoft і .NET Foundation. Mono включає в себе як засоби розробки так і інфраструктуру, необхідну для запуску клієнтських і серверних додатків .NET.

Основні компоненти Mono:

1. Компілятор C#

   Компілятор C# від Mono повністю підтримує стандарти C#(1.0-6.0).

2. Mono runtime

   Це середовище виконання(CB) 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 +, Zip-файлів, LDAP, OpenGL, Cairo, POSIX і т.д[7].

Основна ідея створення Mono – запуск .NET програм на інших ОС, наприклад Linux, OS X і т.д, оскільки це неможливо з використанням CLR. Тому компанією Novell була створена така інфраструктура (рис. 2.1):

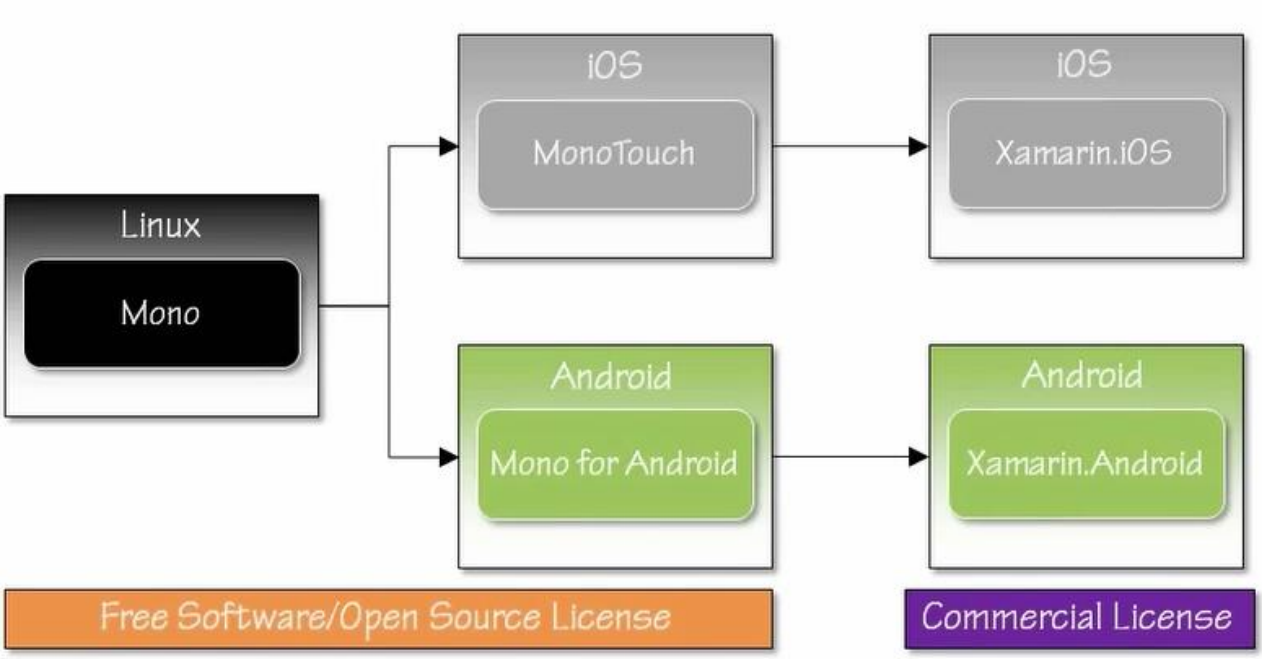

Рисунок 2.1 – Ієрархія Mono[17]

Як бачимо, розробникам з компанії Novell було створено реалізацію CLR та CLI для ядра Linux, тобто усі додатки типу ASP.NET, ADO.NET і т.д. можна запускати на даній платформі. Також було створено аналогічні копії Mono під інші ОС – iOS, Android, названі відповідно MonoTouch та Mono for Android, оскільки дані ОС також працюють над ядром Linux. Усі ці продукти є безкоштовними та з відкритим кодом, однак з деяких причин їх розробку було призупинено, тому частина розробників перейшла з компанії Novell та заснували компанію Xamarin, продовживши роботу над новими продуктами – Xamarin.iOS та Xamarin.Android, що стали комерційними. Як вже було зазначено в розділі раніше, з 2016 року Xamarin.Android, Xamarin.iOS стали повністю безкоштовними та підтримуються компанією Microsoft. Принцип роботи Mono схожий на принцип роботи .NET CLR (рис. 2.2):

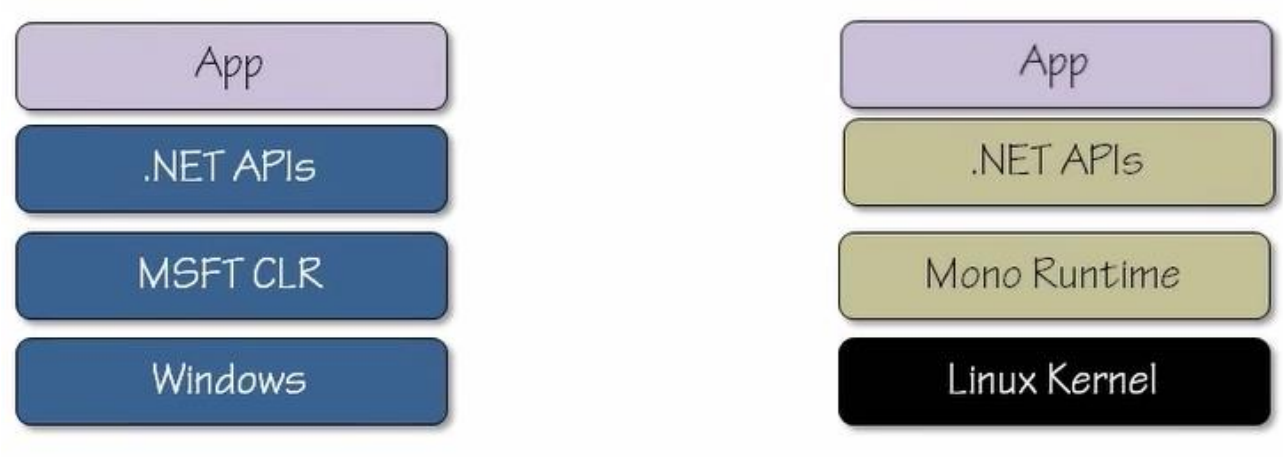

Рисунок 2.2 – Порівняння принципів роботи Mono та CLR[17]

Mono дозволяє використовувати увесь стек технологій, що й CLR, однак, з тією різницею, що додатки на Mono можна запускати не лише під Windows. Mono використовується не лише для Linux, але й для ОС Android, про що буде згадано в наступному розділі. Хоча iOS також побудований на ядрі Linux, проте Mono в даній ОС не використовується у явному вигляді і далі буде пояснено чому.

2.2 Xamarin.Android. Опис та основні можливості

Xamarin.Android додатки виконуються в СВ Mono. Mono працює пліч-о-пліч з віртуальною машиною Android Runtime (ART). Обидва СВ працюють над ядром Linux і надають різні інтерфейси API для коду програміста, що дозволяє розробникам отримати доступ до даної системи[9].

Розробникам надано простори імен System, System.IO, System.Net та інші бібліотеки класів .NET, щоб отримати доступ до основних можливостей операційної системи Linux.

На Android, більшість системних можливостей, таких як аудіо, графіки, OpenGL і телефонії не доступні безпосередньо до нативних додатків, вони доступні тільки через Android Java Runtime API, що були реалізовані в

просторі імен Java. * або Android. *[9]. Архітектура роботи приблизно така (рис. 2.3):

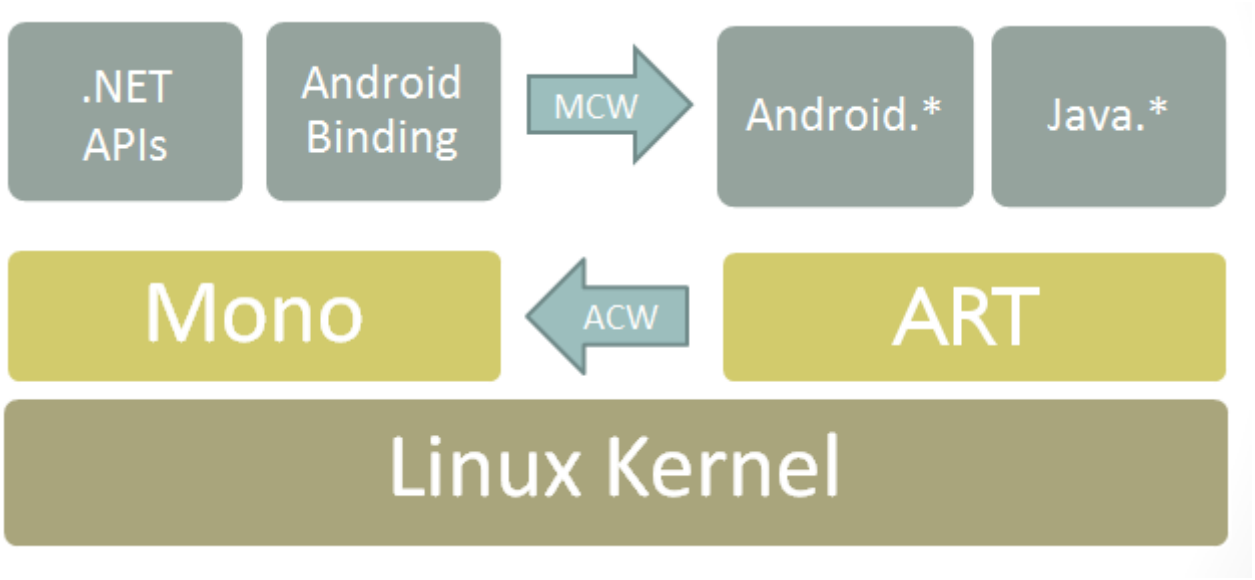

Рисунок 2.3 – Принцип роботи Xamarin.Android[16]

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)[9].

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[9].

Так як Xamarin додатки компілюються в нативні додатки, то програмістам доступні усі контроли (controls), що використовуються при написанні додатків з використанням Java – Popup Menus, Views, Date Pickers і т.д.

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

Кожен додаток Android працює з використанням спеціальних об’єктів, названих activity. Наприклад, кожна сторінка додатку на Android і є activity, а кожне таке activity має свій цикл існування(lifecycle). Xamarin.Android також підтримує цю концепцію, дозволяючи описувати Activity з використанням C#. Усі activity в Xamarin.Android обрамляються спеціальними атрибутами, що дозволяють додавати дані activity до спеціального Android маніфеста, де описуються та додаються основні компоненти додатку. Розглянемо цикл роботи activity (рис. 2.4):

1. При включенні додатку викликається метод OnCreate, в данному методі, можна отримувати збережені дані, наприклад, якщо додаток було пере запущено.
2. Згодом викликається метод OnStart, що необхідний для встановлення значень для View – тобто ініціалізація користувацького інтерфейсу.
3. Метод OnResume викликається завжди після методу OnStart, в даному методі можна ініціалізувати камеру, чи інші ресурси, необхідні для виконання додатку.
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.

5. OnStop – метод, що викликається, коли користувач вже не бачить дану Activity, краще не використовувати метод OnStop, а чистити та зберігати усі ресурси в методі OnPause.
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.
7. OnRestart – викликається після OnStop, якщо користувач знову ввімкнув дане Activity.

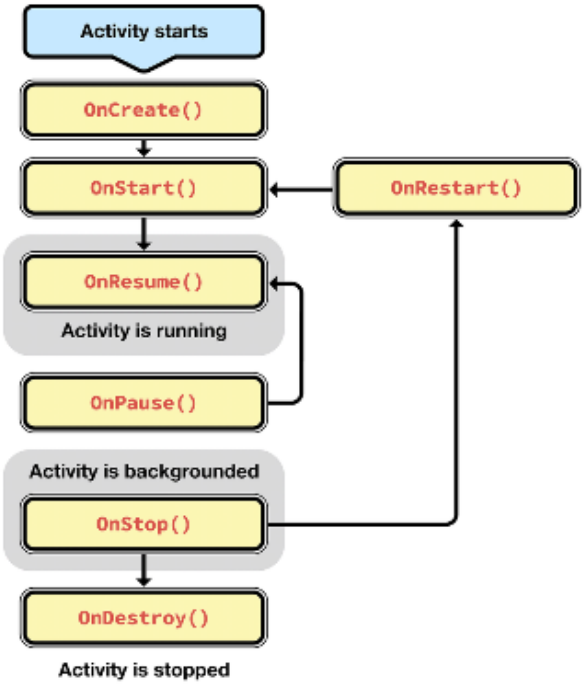

Рисунок 2.4 – Схема життєвого циклу додатку[15]

Також Xamarin.Android додатки надають такі можливості:

1. Створення спеціальних провайдерів контенту та резолверів контенту, що надають можливість отримувати та надавати дані, а також шукати дані з для інших додатків у системі;

2. 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. Використання фрагментів замість декількох activity для створення більш швидкого додатку, оскільки не потрібно підгружати різні activity у відповідь користувачу;
4. Xamarin.Android підтримує усі API рівні ОС Android, отже розробникам завжди доступні усі можливості даної платформи;
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, що не гірші від Java аналогів, дозволяє використовувати усе, що надає платформа, а додатки по швидкодії не програють аналогам.

2.3 Xamarin.iOS. Опис та 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. 2.5):

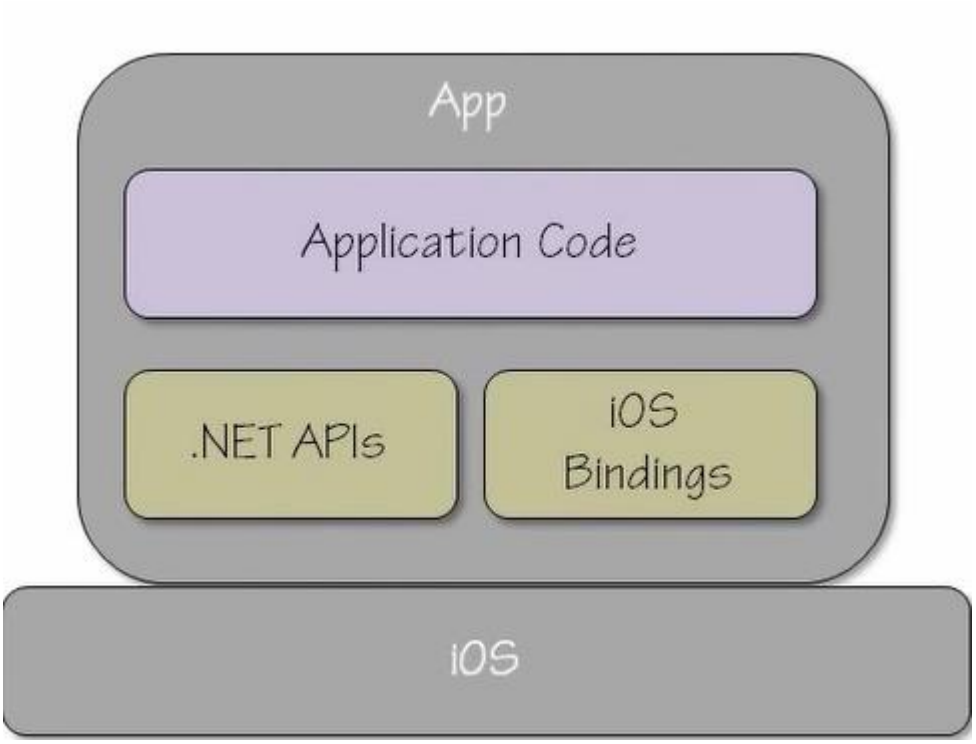

Рисунок 2.5 – Принцип роботи Xamarin.iOS[17]

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.

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[10]. Життєвий цикл додатку в iOS (рис. 2.6):

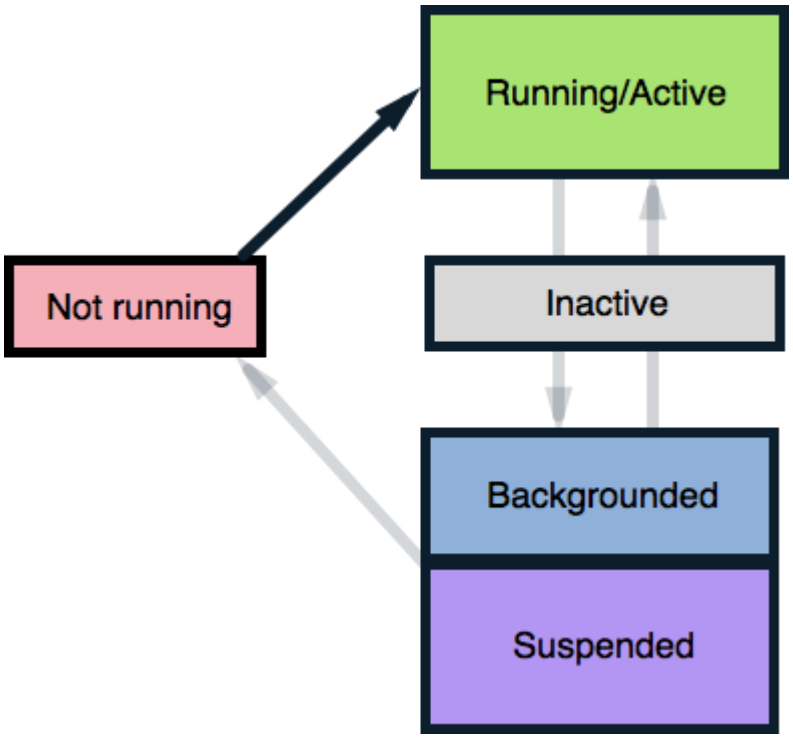

Рисунок 2.6 – Життєвий цикл додатку в iOS[10]

Загалом додаток може бути у декількох станах (див. рис. 2.6):

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 (рис. 2.7):

1. OnActivated – додаток вперше запускається, або додаток повертається з фонового стану. В даний метод пишеться логіка, яку потрібно виконати, коли додаток запускається;
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;
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.

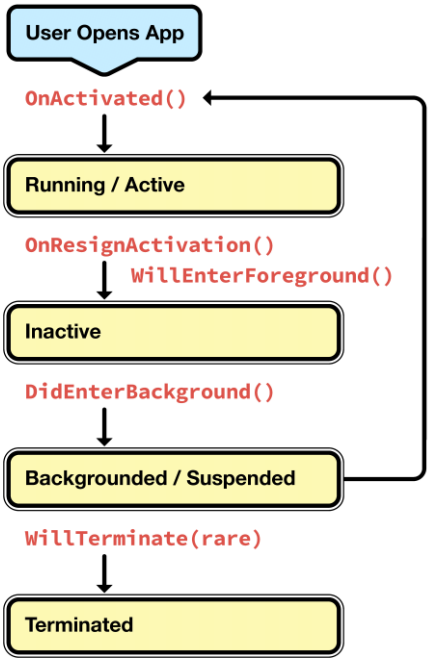

Рис. 2.7 – Схема викликів обробників подій зміни стану додатку в Xamarin.iOS[10]

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

2.4 Xamarin.Forms. Опис та 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. 2.8). Тобто розробнику не потрібно знати особливості кожної платформи, розробники із Xamarin все зробили за вас.

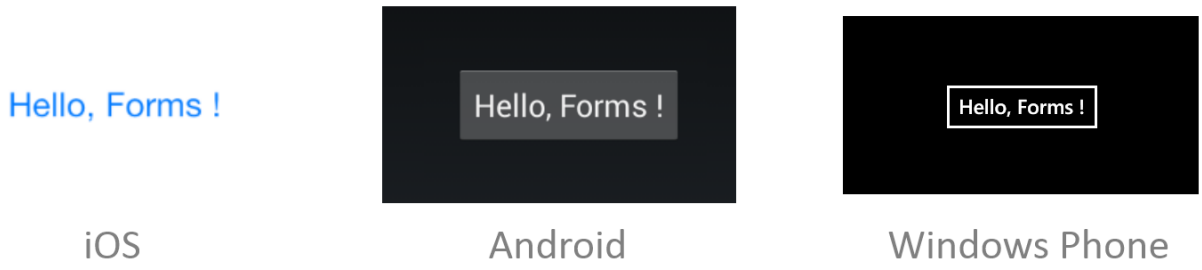

Рис. 2.8 – Стандартний вигляд кнопок для кожної платформи в Xamarin.Forms[11]

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[3].

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 і т.д.
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.

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 платформо-специфічних 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. [11].

2.5 Висновок до розділу 2

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, щ обула розглянута, звичайно є Mono – середовище виконання, аналог CLR під Windows для Linux.

3. МЕТОДИКА НАПИСАННЯ ПРОГРАМ НА C# З ВИКОРИСТАННЯМ XAMARIN

Xamarin – обширний фреймворк, в даний фреймворк входить дуже багато бібліотек, а також доступні і бібліотеки-розширення, створені програмістами для використання в своїх проектах, що є у вільному доступі. Тобто інфраструктура даного фреймворка постійно розширюється та вдосконалюється. Проте, при описі певної технології потрібно дивитися не лише на його інфраструктуру, а й на його зручність для розробника, поріг входження. Саме для цього і призначений даний розділ – не оглянути Xamarin з точки зору ОС, які підтримуються фреймворком, основних можливостей, що реалізовано у фреймворку, його взаємодію з ОС та компонентами, а оглянути фреймворк з точки зору розробника, тобто, які основні концепції використовуються, та як вони реалізовані.

В даному розділі буде розглянуто основні принципи розробки з використанням технології Xamarin, основна частина якого буде приділена паттернам програмування та порівнянню побудови додатків з використанням портативної бібліотеки класів(portable class library) або спільного проектку (shared project), порівняно та описано переваги та недоліки використання Xamarin Native та Xamarin.Forms в розробці програмних продуктів.

3.1 Основні підходи для написання додатків на Xamarin

З найпершого релізу фреймворка Xamarin в ньому було доступно такі два блоки – Xamarin.iOS та Xamarin.Android – як вже зазначалося, ці частини фреймворка дозволяють розробляти додатки під відповідні платформи з використанням єдиної мови програмування, що, звичайно сильно полегшує процес розробки програмного продукту, та дозволяє використовувати спільну бізнес логіку додатку та, наприклад, логіку доступу до SQLite бази даних, що встановлено на мобільному пристрої чи планшеті. Також використання єдиної

логіки додатку веде за собою скорочення часу на розробку, що є гарною перевагою в порівнянні з використанням різних мов програмування, проте є й недоліки, основним з яких є складність для розробника. Розробнику потрібно гарно знати платформи Android та iOS щоб писати дійсно складні та функціональні додатки, тобто потрібно витратити дуже багато часу на те, щоб почати створювати конкурентоспроможні програмні продукти. Xamarin.Android та Xamarin.iOS є частиною підходу розробки, що називається Xamarin Native, тобто написання чистого нативного додатку з використанням C#.

Однак, з плином часу та вдосконаленням інфраструктури Xamarin Native розробниками з компанії Xamarin було вирішено створити ще одну частину свого фреймворка – Xamarin.Forms. Даний підхід є набагато простішим для розробника, оскільки він полегшує роботу з різними операційними системами. Звісно, розробнику і досі потрібно писати певний платформо-залежний код, проте з використанням даного підходу даного коду стане набагато менше, оскільки в Xamarin.Forms використовується XAML (Extensible application markup language) для побудови графічного інтерфейсу, що дозволяє винести увесь код пов’язаний з графічним дизайном у спільну бібліотеку, тобто розробник тепер не повинен знати специфіку інтерфейсу для кожної платформи, достатньо лише знати XAML та пов’язані з ним класи, даний підхід, тобто отримуємо зв’язку C# та XAML, що схожа на JavaScript та CSS. На рисунку порівняно схему роботи даних підходів (рис. 3.1):

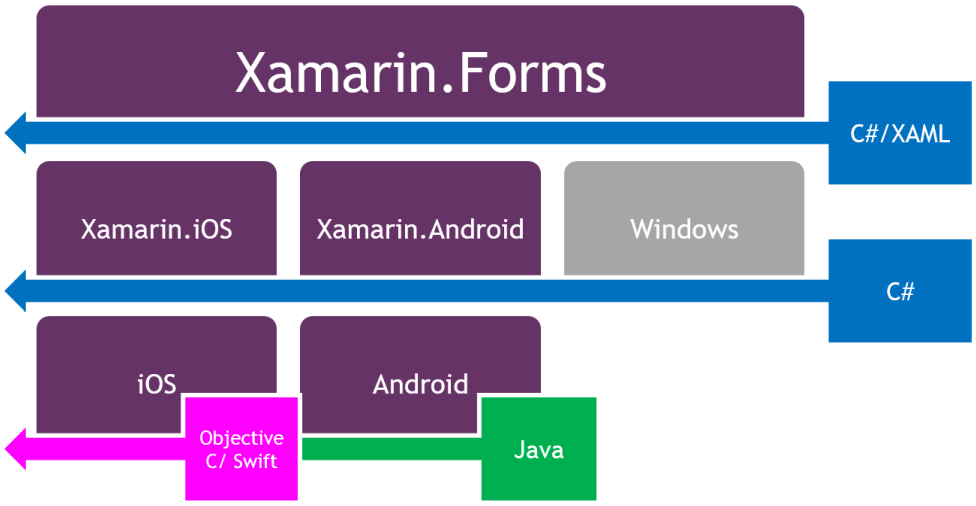

Рисунок 3.1 – Схема покриття Xamarin.Forms та Xamarin Native[14]

Як видно з рисунка, Xamarin.Forms повністю “покриває” усі три платформи та дозволяє розробнику зосередитися на бізнес логіці додатку та його структурі і економити час розробки. Однак, не усе так гарно як могло б здаватися, звичайно, такий додаток буде крос-платформним, однак даний додаток матиме багато обмежень з точки зору функціональної частини додатку, оскільки неможливо об’єднати усі можливості платформ і використовувати як одну спільну, адже вони реалізовані по-різному та й деякі з них не наявні у інших платформах.

Оскільки тема Xamarin.Forms досить обширна, розглянемо основні недоліки та переваги цієї технології:

1. Дуже схожа на стандартні Windows технології;
2. Спільна логіка для всіх додатків;
3. Не потрібно вглиблюватись в можливості конкретної платформи
4. Нижча швидкодія, порівняно з Xamarin.iOS, Xamarin.Android;
5. Неповна реалізація усіх можливостей платформ;
6. Компромісні рішення для реалізації функціоналу, що відрізняється на різних платформах;
7. Можлива різна поведінка додатку для платформ.

Тобто, точних рекомендацій для розробника, який з даних підходів використовувати – Xamarin Native чи Xamarin.Forms немає, усе залежить від бажання та можливостей розробника, масштабів проекту. Є лише декілька критеріїв селекції одного з цих підходів – час розробки та можливості додатку. Якщо програмний продукт повинен мати широкий функціонал та використовувати усі можливості програмного інтерфейсу (api) платформи, то гарним вибором, звичайно буде Native підхід, якщо ж основним критерієм є час розробки, то єдиним логічним вибором є Xamarin.Forms.

Тепер розглянемо підходи структурування та побудови логіки додатку при написанні програмного коду. Оскільки додаток має певні частини додатку мають спільну логіку, розробник може винести її у спільну бібліотеку чи спеціальний проект, або реалізовувати весь функціонал окремо для кожної платформи та повторювати логіку, що не рекомендується. При розробці на Xamarin розрізняють два основних підходи для структурування проекту – портативна бібліотека класів, тобто така бібліотека, код якої може виконуватися на різних пристроях під різними ОС та загальний проект – такий проект, що містить спільний код та ресурси для специфічних платформ. З опису здається, що дані підходи схожі, проте вони зовсім різні, розглянемо їх відмінності в наступних підрозділах.

3.1.1 Портативна бібліотека класів (Portable class library)

Портативна бібліотека класів використовується для структурування лише програмного коду, а саме класів та інтерфейсів, що можуть виконуватися на кожній з платформ, що підтримується фреймворком, в основному в даному проекті зберігають бізнес логіку додатку – менеджери та сервіси, репозиторії та їх інтерфейси, тобто шар доступу до даних додатку (data access layer) . Уся логіка, яку можливо реалізувати лише з використанням платформно-специфічних конструкцій будується у вигляді інтерфейсів, реалізація яких створюється у конкретному проекті, наприклад, Android проекті. Даний підхід є найкращим вибором для великих проектів, оскільки уся специфічна логіка

зберігається там, де потрібно і структуру проекту та реалізації легше підтримувати, однак, даний підхід має і свої недоліки, наприклад те, що з таким підходом розробнику потрібно переписувати багато коду, а отже код програми буде більшим ніж у програмного продукту, що використовує загальний проект. На рисунку (рис 3.2) зображена типова структура проекту, в якому наявна портативна бібліотека класів[8]:

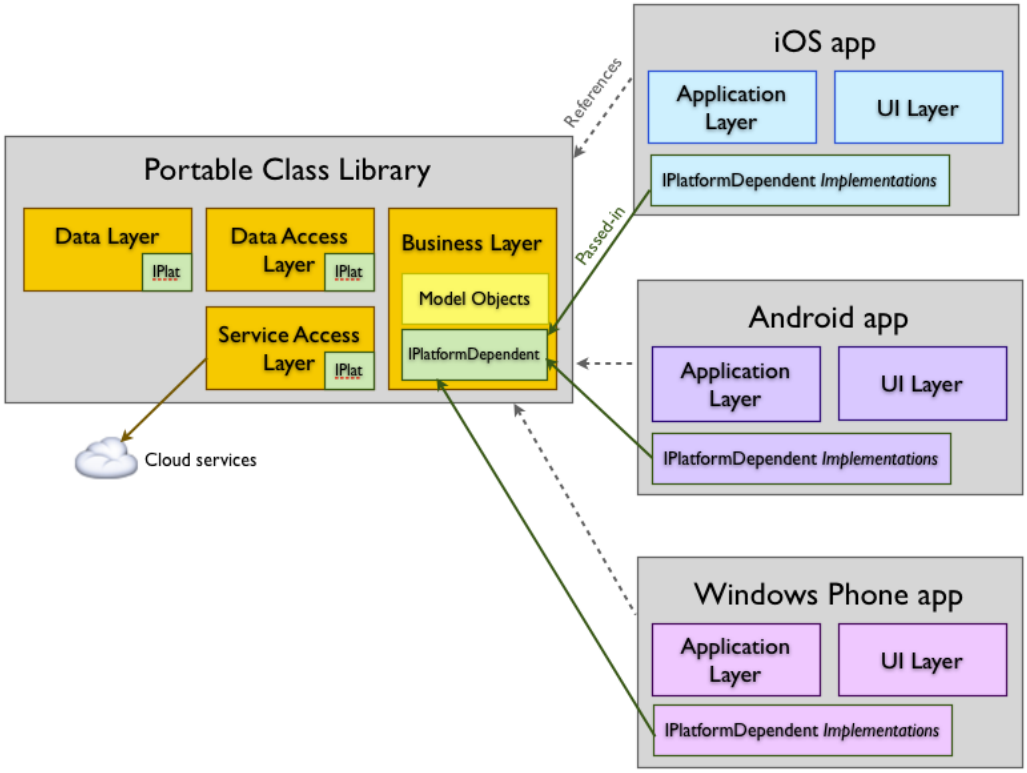

Рисунок 3.2 – Структура додатку з портативною бібліотекою класів[12]

3.1.2 Спільний проект (Shared project)

Ще одним варіантом структурування коду програми є так званий спільний проект (shared project) – це також бібліотека класів як і портативна бібліотека класів. Даний підхід має ту ж структуру, проте має інший принцип написання коду в ньому. Код більше не структурується з використанням інтерфейсів та їх конкретних реалізацій (хоча інтерфейси і використовуються,

проте вони реалізуються з використанням класів одразу в цій самій бібліотеці, а не в бібліотеці для конкретної платформи). Даний підхід реалізується за допомогою препроцесорних директив “#if”, “#elseif”, “#endif”[8]. Тобто, якщо певний функціонал має розбіжності в реалізації, достатньо лише написати препроцесорну директиву та описати платформо-специфічний код всередині конструкції цієї директиви. Звісно, така бібліотека класів повністю завантажується в загальну збірку для кожної з платформ і при компіляції виокремлюються лише потрібні команди для цієї платформи.

Такий підхід простіший ніж підхід, описаний раніше, проте код, написаний таким чином важче читати, через використання директив, якщо певний клас використовує багато функціоналу, специфічного для різних платформ, то код даного класу набагато гірше підтримувати ніж код, що написано у спільній бібліотеці класів, тому даний підхід використовують рідше. Найкраще використовувати цей варіант для маленьких проектів, або для проектів, де розбіжностей в реалізації для певної платформи мало, і використання директив зводиться до мінімуму. На відміну від раніше описаного способу спільний проект повністю завантажується у збірку мобільного проекту, а не постачається як окрема бібліотека, тобто мобільний проект не зсилається на бібліотеку, а містить всі необхідні класи в собі. Мінусом такого підходу ще є й те, що при зміні логіки, що міститься в класі, який описано в спільному проекті потрібно перекомпілювати весь мобільний проект, а при використанні портативної бібліотеки класів – лише певну бібліотеку.

Структуру програмного продукту з використанням спільного проекту(рис. 3.3):

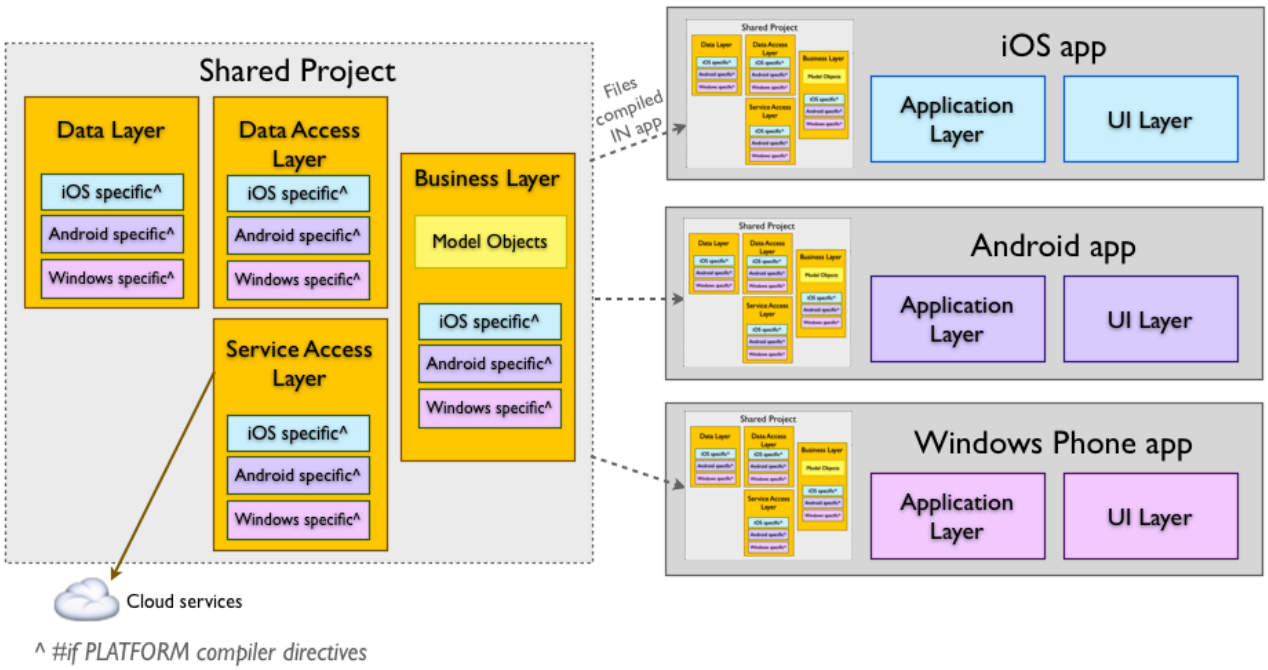

Рисунок 3.3 – Структура програми з використанням спільного проекту[12]

3.2 Основні рекомендації для побудови додатків з використанням Xamarin

Кожен великий проект використовує паттерни для кращої структуризації коду та його підтримки в майбутньому, паттерн – це “рекомендація” розробникам для написання якісного коду, тобто це не панацея і будь-який розробник в праві відмовитися від використання паттернів. Мобільна розробка не є виключенням, тут також використовуються багато паттернів, однак, розглянемо основні із них, що використовуються при написанні програм з використанні Xamarin і рекомендовані самими розробниками Xamarin:

1. Розділення структури додатку на шари, використання так званої багатошарової архітектури (n-tier architecture). Загалом, в мобільному додатку мають бути присутні такі шари – данних(DL), доступу до данних(DAL), бізнес шар(BLL), доступу до сервісів(SAL), шар додатку (AL) та шар користувацького інтерфейсу(UIL)[8];

2. Використання паттерна MVC(модель представлення контроллер), даний паттерн використовується ще й у веб-додатках для розшарування логіки, графічного інтерфейсу та доступу до даних[8];
3. Використання паттерна DI (вставка залежностей)[8], класи в програмі не повинні залежати від конкретної реалізації, а від інтерфейсів, а самі залежності повинен “вставляти” контейнер;
4. Використання паттерна MVVM(Model-View-ViewModel)[8] даний патерн використовується в розробці під Windows та Xamarin.Forms, розділяє модель та представлення, використовуючи спеціальні класи;
5. Одинак(singleton) – гарним прикладом використання даного патерна в мобільній розробці та в Xamarin є використання бази данних SQLite, оскільки об’єкт бази може бути лише один.
6. Репозиторій(repository) – даний патерн використовується як метод доступу до об’єктів бази даних.

Хоча мобільні додатки повинні розділяти свою структуру на певні шари, вони не обов’язково повинні мати усі з них, наприклад, додаток, що не використовує базу данних не повинен мати шару данних та доступу до данних, а у досить простих додатках, що використовують базу даних ці шари можна об’єднати, також в таких додатках можна не використовувати репозиторії, оскільки немає джерела данних.

Паттерн DI потрібно також використовувати лише у великих додатках, в яких у будь-який час може знадобитися зміна певного функціоналу, наприклад замість використання ORM(Object relational mapper) фреймворка типу Entity Framework чи SQLite.NET потрібно використовувати ADO.NET, тому в додатку не використовуються залежні класи, а лише класи, які використовують інтерфейси в своїй роботі.

Паттерн MVVM досить популярний і існує багато його реалізацій не лише під Windows та Xamarin.Forms, даний паттерн схожий на MVC проте має свої відмінності. На відміну від MVC в даному випадку view та vm(ViewModel)

взаємодіють між собою завдяки спеціальним командам та прив'язкам даних, чого немає в паттерні MVC (рис. 3.4):

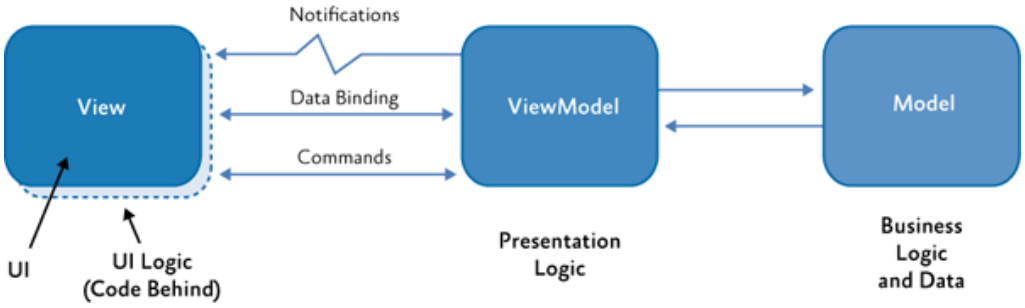

Рисунок 3.4 – Паттерн MVVM[13]

3.3 Висновок до розділу 3

В даному розділі було розглянуто основні принципи розробки програмного забезпечення з використанням Xamarin Native та Xamarin.Forms, проте на відміну від минулого розділу, дані підходи було розглянуто не з технічної точки зору а з програмної, тобто плюси та мінуси використання кожної з технологій. Було показано, що використання Xamarin Native є більш вигідним при побудові комплексних додатків, що повинні використовувати повний стек технологій програмного інтерфейсу ОС, обирати Xamarin.Forms слід лише у випадку, коли час розробки важливий і не потрібне використання усіх можливостей ОС.

Також було розглянуто підходи для побудови додатку з використанням Xamarin – спільного проекту та портативної бібліотеки класів, було розглянуто основні принципи їх використання, переваги та недоліки. Зазначено, що підхід з використанням портативної бібліотеки класів є кращим для великих по структурі проекту, оскільки код читається легше.

Ще однією розкритою темою даного розділу є основні рекомендації для структурування проекту та паттерни, які використовуються при розробці

додатків для мобільних пристроїв. Було розглянуто 6 паттернів, що використовуються в Xamarin-додатках, а також обґрунтовано їх використання в деяких ситуаціях. Розглянуто принцип поділу проекту на шари, та показано, що додаток не обов’язково повинен мати усіх їх, адже у деяких ситуаціях вони просто не потрібні чи можуть бути об’єднані в один.

4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту – мультимедійного сервісу, призначеного для слухання музики, альбомів та збірок пісень. Інтерфейс користувача був розроблений за допомогою XAML, HTML, AXML, основною мовою програмування є мова C#, розробка велася у середовищі розробки Microsoft Visual Studio 2015. Інтерфейс користувача створений за допомогою технології Xamarin.

Програмний продукт призначено для використання на персональних комп'ютерах з операційною системою Windows та мобільних пристроях з доступом до інтернет під управлінням операційної системи Windows Phone, Android.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

1. Визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.
2. для кожної функції визначаються повні річні витрати й кількість робочих часів.
3. для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.
4. після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

4.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

1. Програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;
2. Забезпечувати високу швидкість обробки великих об’ємів даних у реальному часі;
3. Забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;
4. Передбачати мінімальні витрати на впровадження програмного продукту.

4.1.1 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який дозволяє завантаження альбомів та пісень з музичними файлами. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір мови програмування;

F_2 – вибір оптимальної СКБД;

F_3 – інтерфейс користувача.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) мова програмування C#;

б) Мова програмування JavaScript.

Функція F_2:

а) Microsoft SQL Server;

б) Oracle.

Функція F_3:

а) інтерфейс користувача, створений за технологією Xamarin;

б) інтерфейс користувача, створений за технологією Ionic.

4.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

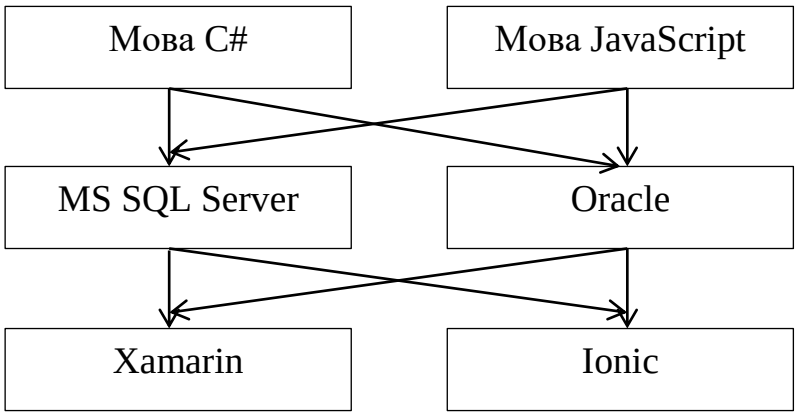

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Як видно з даної морфологічної карти, для реалізації ПП можна використовувати мову JavaScript, що є дуже популярною мовою для розробки в наш час, однак ще одним варіантом є використання мови C#, різниця в тому, що мова C# компілюється, а JavaScript – інтерпретована мова програмування. Щодо використання бази даних для даного проекту, то не має значення яку з баз можна обрати і вибір бази даних не залежить від вибору мови програмування, а от вибір фреймворка для реалізації інтерфейсу користувача – залежить від мови програмування, тому для C# єдиною альтернативою є Xamarin, а для JavaScript – фреймворк Ionic.

Розглянемо недоліки та переваги обраних технологій та побудуємо позитивно-негативну матрицю (табл. 4.1):

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	А	Компілюється	Потребує гарного знання особливостей платформи
	Б	Займає менше часу при написанні коду	Низька швидкодія
F2	А	Більш дешева вартість корпоративної ліцензії	Необхідність додаткової інсталяції, низький рівень користувацької підтримки
	Б	Подовжений термін користувацької підтримки	Більш висока вартість корпоративної ліцензії. Необхідність додаткової інсталяції
F3	А	Швидкий інтерфейс	Для кожної платформи потрібно знання нативних засобів створення інтерфейсу
	Б	Однаковий інтерфейс для кожної з платформ	Низька швидкодія CSS, що використовується для написання інтерфейсу

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція *F1:*

Оскільки програма потребує швидкого часу відгуку та швидкої обробки данних, то оберемо мову C# для створення програмного продукту.

Функція *F2:*

Вибір СКБД не відіграє велику роль у даному програмному продукті, тому вважаємо варіанти а) та б) гідними розгляду.

Функція *F3:*

Оскільки, програмний продукт реалізується на мові C#, використовуємо варіант А як єдиний можливий.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1а – F2а – F3а
2. F1а – F2б – F3а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.2 Обґрунтування системи параметрів ПП

4.2.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

1. X1 – швидкодія мови програмування;
2. X2 – об’єм пам’яті для збереження даних;
3. X3 – час обробки даних;
4. X4 – потенційний об’єм програмного коду.

X1: Відображає швидкодію операцій залежно від обраної мови програмування.

X2: Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми.

X3: Відображає час, який витрачається на дії.

X4: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

4.2.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 4.2.

Таблиця 4.2 – Основні параметри ПП

Назва Параметра	**Умовні позначення**	**Одиниці виміру**	**Значення параметра**		
			гірші	**Середні**	**Кращі**
Швидкодія мови програмування	X1	Оп/мс	2000	11000	19000
Об’єм пам’яті для збереження даних	X2	Мб	32	16	8
Час обробки запитів користувача	X3	мс	5000	2900	1000
Потенційний об’єм програмного коду	X4	кількість строк коду	15000	7000	5000

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

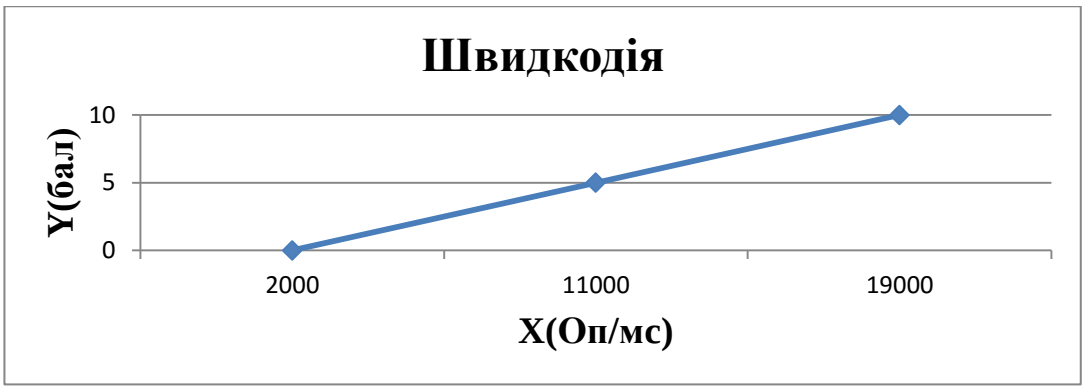

Рисунок 4.2 – X1, швидкодія мови програмування

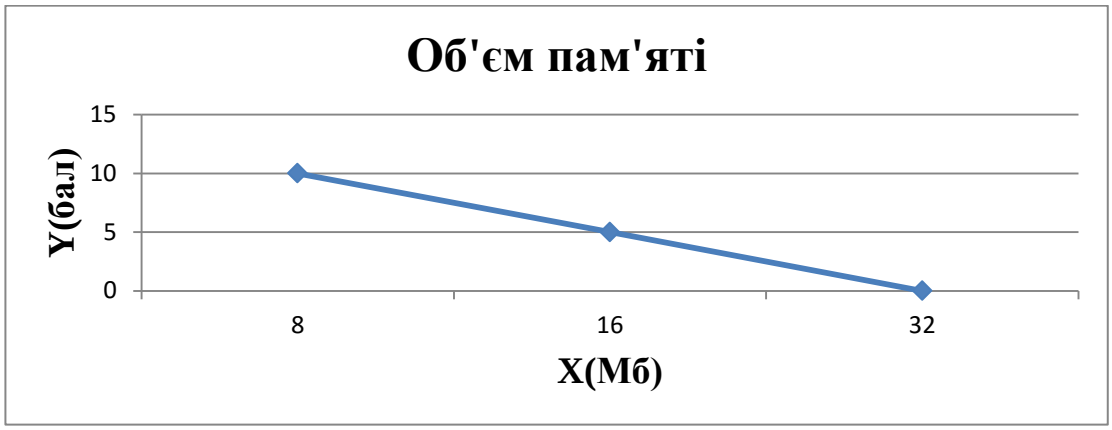

Рисунок 4.3 – X2, об’єм пам’яті для збереження даних

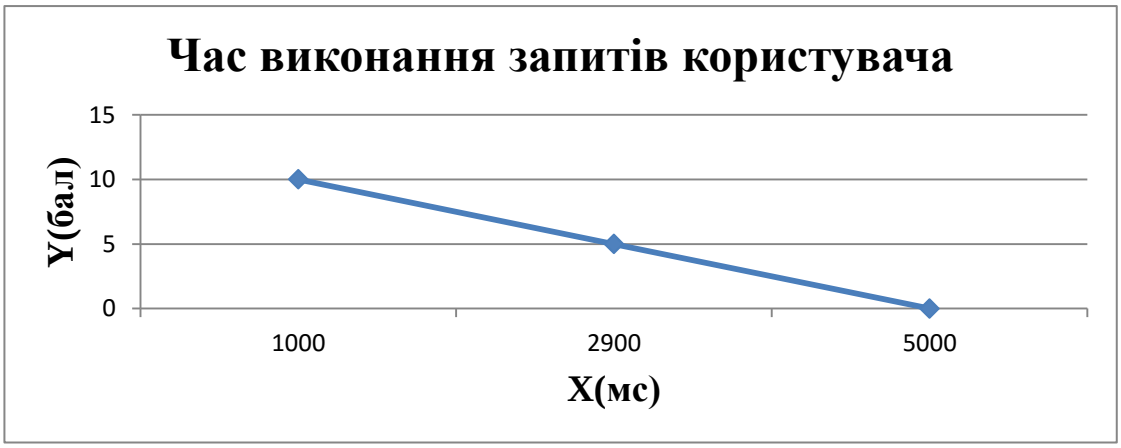

Рисунок 4.4 – X3, час виконання запитів користувача

Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.2.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

1. Визначення рівня значимості параметра шляхом присвоєння різних рангів;
2. Перевірку придатності експертних оцінок для подальшого використання;
3. Визначення оцінки попарного пріоритету параметрів;
4. Обробку результатів та визначення коефіцієнту значимості.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	1	2	2	1	1	1	2	10	-7.5	56.25
X2	Об’єм пам’яті для збереження даних	Мб	1	2	1	1	2	2	1	10	-7.5	56.25
X3	Час обробки запитів користувача	Мс	3	3	1	3	2	2	2	16	-1.5	2.25
X4	Потенційний об’єм програмного коду	кількість строк коду	5	3	6	5	5	5	5	34	16.5	272.25
	Разом		10	10	10	10	10	10	10	70	0	387

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17.5$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 387$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 * 387}{7^2(4^3 - 4)} = 1.57 > W_k = 0.67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0.67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	=	=	>	=	<	<	>	=	1
X1 і X3	<	<	>	<	>	<	=	<	0.5
X1 і X4	<	<	<	<	<	<	<	<	0.5
X2 і X3	<	<	=	>	=	=	>	=	1
X2 і X4	<	<	<	<	<	<	<	<	0.5
X3 і X4	<	<	<	<	<	<	<	<	0.5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{де } b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{де } b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Для кожного з параметрів розрахуємо коефіцієнти вагомості, із зазначенням проміжних результатів обчислення, при чому будемо проводити ітерації доти, поки різниця між результатами не буде різнитися не більше ніж на 2%, результати показано в таблиці(табл.. 4.5), для даного варіанту, кількість ітерацій не перевищила трьох.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1.0	1.0	0.5	0.5	3	0.1875	11.25	0.186	43	0.188
X2	1.0	1.0	1.0	0.5	3.5	0.2185	13.25	0.219	49.875	0.218
X3	1.5	1.0	1.0	0.5	4	0.25	14.75	0.244	55.5	0.244
X4	1.5	1.5	1.5	1.0	5.5	0.344	21.25	0.351	80.125	0.35
Всього:					16	1	60.5	1	228,5	1

4.3 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2*(об’єм пам’яті для збереження даних) та *X1* (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3 (*час обробки даних) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 1000 мс або варіанту б) 80мс.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} ,$$

де n – кількість параметрів;$K_{вi}$– коефіцієнт вагомості i–го параметра;B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1)	А	11000	5	0.188	0.94
F2(X3)	А, Б	3600	2.8	0.244	0.68
F2(X4)	А	9000	4	0.35	1.4
	Б	12000	2	0.35	0.7
F3(X2)	А	16	5	0.218	1.09

За даними з таблиці 4.6 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

K_{K1} = 0.94 + 0.68 + 1.4 + 1.09 = 4.11

K_{K2} = 0.94 + 0.68 + 0.7 + 1.09= 3.41

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.4 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;
2. Розробка програмної оболонки;

При цьому варіант 1 має додаткове завдання:

3. Використання стандартного ORM фреймворку;

А варіант 2 має інше додаткове завдання:

4. Використання та ознайомлення із Oracle ORM.

Завдання 1 за ступенем новизни відноситься до групи Б, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 2; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \qquad (5.1)$$

де T_P – трудомісткість розробки ПП;$K_П$ – поправочний коефіцієнт;$K_{СК}$ – коефіцієнт на складність вхідної інформації; K_M – коефіцієнт рівня мови програмування;$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм; $K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни Б та групи складності алгоритму 2, трудомісткість дорівнює: T_P =27 людино-днів. Поправочний коефіцієнт, який враховує вид банку даних, для першого завдання: $K_П$ = 0.9. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1: $K_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ}$ = 0.8. Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$T_1 = 27 \cdot 0.9 \cdot 0.8 = 19.44$ людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто T_P =19 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ = 0.8:

$T_2 = 19 \cdot 0.9 \cdot 0.8 = 13.68$ людино-днів.

Для третього завдання (використовується алгоритм третьої групи складності, ступінь новизни Г):

Тр = 8 людино-днів;

Кп = 0.6; Кст = 1;

То = $8 \cdot 0.6 \cdot 1 = 4.8$.

Для четвертого завдання (використовується алгоритм другої групи складності, ступінь новизни Г з використанням перемінної інформації):

Тр = 12 людино-днів;

Кп = 0.72; Кст = 0.8;

То = $12 \cdot 0.72 \cdot 0{,}8 = 6.91$.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$T_I = (19.44 + 13.68 + 4.8) \cdot 8 = 303.36$ людино-годин;

$T_{II} = (19.44 + 13.68 + 6.91) \cdot 8 = 320.24$ людино-годин;

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 29000 грн., один фінансовий аналітик з окладом 15000грн. Визначимо зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} \text{ грн.},$$

де М – місячний оклад працівників;T_m – кількість робочих днів тиждень;t – кількість робочих годин в день.

$$С_ч = \frac{29000 + 29000 + 15000}{3 \cdot 21 \cdot 8} = 144.84 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д,$$

де $С_ч$– величина погодинної оплати праці програміста;$Т_i$ – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 144.84 \cdot 303.36 \cdot 1.2 = 52726.39$ грн.

II. $С_{ЗП} = 144.84 \cdot 320.24 \cdot 1.2 = 55660.27$ грн.

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 52726.39 \cdot 0.22 = 11599.8$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 55660.27 \cdot 0.22 = 12245.26$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 29000 грн., з коефіцієнтом зайнятості 0.2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 29000 \cdot 0.2 = 69600$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 69600 \cdot (1 + 0.2) = 83520$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 83520 \cdot 0.22 = 18374.4$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 10000 = 2875$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 10000 \cdot 0.05 = 575$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0.9 = 1706.4$ годин,

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706.4 \cdot 0.2 \cdot 1.56 = 518.18$ грн.,

де N_C – середньо-споживча потужність приладу; $K_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$C_Н = Ц_{ПР} \cdot 0.67 = 10000 \cdot 0.67 = 6700$ грн.

Тоді, річні експлуатаційні витрати будуть:

$C_{ЕКС} = C_{ЗП} + C_{ВІД} + C_А + C_Р + C_{ЕЛ} + C_Н$

$C_{ЕКС} = 83520 + 18374.4 + 2875 + 575 + 518.18 + 6700 = 112562.52$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$C_{М\text{-}Г} = C_{ЕКС} / T_{ЕФ} = 112562.52/1706.4 = 65.96$ грн/год.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$C_М = C_{М\text{-}Г} \cdot T$

I. $C_М = 65.96 \cdot 303.36 = 20009.62$ грн.;

II. $C_М = 65.96 \cdot 320.24 = 21123.03$ грн.;

Накладні витрати складають 67% від заробітної плати:

$C_Н = C_{ЗП} \cdot 0.67$

I. $C_Н = 20009.62 \cdot 0.67 = 13406.44$ грн.;

II. $C_Н = 21123.03 \cdot 0.67 = 14152.43$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_Н$

I. $C_{ПП} = 52726.39 + 11599.8 + 20009.62 + 13406.44 = 97742.25$ грн.;

II. $С_{ПП}$ = 55660.27 + 12245.26 + 21123.03 + 14152.43 = 103180.99 грн.

4.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj},$$

$К_{ТЕР1}$ = 4.11/ 97742.25 = $0.42 \cdot 10^{-4}$;

$К_{ТЕР2}$ = 3.41 / 103180.99 = $0.33 \cdot 10^{-4}$;

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 0.42 \cdot 10^{-4}$.

4.6 Висновки до розділу 4

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що

залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 0.42 \cdot 10^{-4}$.

Цей варіант реалізації програмного продукту має такі параметри:

1. мова програмування – C#;
2. СКБД MS SQL Server;
3. інтерфейс користувача, створений за технологією Xamarin.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

ВИСНОВКИ

У даній дипломній роботі було розглянуто фреймворк Xamarin, що дозволяє розроблювати додатки з використанням C# під такі ОС як Android, iOS, Windows Phone, Windows. При використанні даного фреймворка для кожної ОС використовуються її нативні компоненти, наприклад, для Android використовуються усі стандартні класи, що описані для мови Java, проте переписані для мови C#, також використовується мова розмітки AXML. Завдяки тому, що Xamarin використовує мову C#, то додаток компілюється, а не інтерпретується, що суттєво підвищує швидкість роботи додатку, при написанні додатку під ОС Android чи Windows(Windows Phone) використовується just-in-time компіляція, однак для iOS використання віртуальних машин заборонено, тому Xamarin використовує ahead-of-time компіляцію, даний підхід відрізняється від, наприклад, JavaScript фреймворків, де використовується інтерпретатор.

В першому розділі розглянуто та дано характеристику інших аналогічних рішень для побудови крос-платформних додатків, таких як PhoneGap, Sencha Touch, OnsenUI. Порівняно Xamarin з такими відомими фреймворками як Ionic та RoboVM, що використовують мови JavaScript та Java відповідно. Зроблено висновок, що серед данних рішень немає явного лідера, адже Ionic перемагає своєю простотою, а RoboVM є аналогом Xamarin для Java та використовує нативну мову програмування для ОС Android, проте не надає можливості написання додатку під Windows чи Windows Phone. Зроблено висновок, що вибір потрібного рішення залежить лише від проекту, що потрібно реалізувати.

У другому розділі було описано такий проект як Mono – віртуальну машину на якій працює Xamarin під ОС Android. Розглянуто історію створення фреймворку, його основні принципи роботи під різні платформи, показано, що хоча Android та iOS використовують ядро Linux, недостатньо використовувати лише Mono для побудови мобільних додатків, тому було створено MonoTouch

та Mono Android, що працюють під iOS та Android відповідно. Показано основні стани роботи додатків, або, як ще можна назвати, життєвий цикл додатків, що працюють на розглянутих ОС, дійшли висновку, що життєвий цикл Android додатків набагато складніший ніж у Windows Phone та iOS. Також, розглянуто проект Xamarin.Forms, що дозволяє збільшити кількість спільного коду для додатку, що працює під розглянуті ОС та ознайомлено з основними структурними частинами Xamarin.Forms, тобто блоками, з яких будується додаток на кожну з платформ. Зроблено висновок, що найкраще за все використовувати Xamarin.Forms в невеликих проектах, де не потрібно використовувати спеціальні можливості ОС, наприклад Android Speech чи HealthKit та EventKit для iOS.

У третьому розділі було розглянуто основні принципи побудови додатку, їх переваги та недоліки з точки зору розробки – витрат та часу на створення програмного продукту. Також показано основні підходи в побудові додатків під різні платформи – спільного проекту(shared project) та портативної бібліотеки класів (portable class library). Та розглянуто основні патерни, що використовуються в розробці мобільних додатків, таких як MVVM, observer та інші, а також рекомендації щодо побудови архітектури додатку.

Можна зробити висновок, що Xamarin – перспективний фреймворк, використання якого дає багато переваг в циклі створення мобільних додатків, скорочуючи розробку та зменшуючи кількість коду за рахунок спільної бізнес логіки та методів доступу до даних, а з використанням Xamarin.Forms дає й можливість розробляти спільний інтерфейс користувача для додатків. Хоча підхід, що використовується в Xamarin має й свої недоліки, основним з яких є те, що програмісту потрібно мати гарні знання одразу трьох різних ОС, що мають мало спільного, тобто розробник повинен мати високу кваліфікацію, хоча цей недолік зникає з роками досвіду.

ПЕРЕЛІК ПОСИЛАНЬ

1. Jonathan Peppers. Xamarin Cross-platform Application Development Second Edition / Peppers J. – Birmingham : Packt, 2015. – 459 с.
2. Nilanchala Panigrahy. Xamarin Mobile Application Development for Android Second Edition / Panigrahy N. – Birmingham : Packt, 2015. – 296 с.
3. Charles Petzold. Creating Mobile Apps with Xamarin.Forms First Edition / Petzold C. – Redmond : Microsoft Press, 2016. – 1187 с.
4. 10 Best Hybrid Mobile App UI Frameworks: HTML5, CSS and JS – Режим доступу : http://noeticforce.com/best-hybrid-mobile-app-ui-frameworks-html5-js-css/. – Дата доступу : 29.03.2016.
5. RoboVM documentation – Режим доступу: http://docs.robovm.com/. – Дата доступу : 3.04.2016.
6. Ionic documentation – Режим доступу: http://ionicframework.com/docs/. – Дата доступу : 3.04.2016.
7. About mono – Режим доступу: http://www.mono-project.com/docs/about-mono/. – Дата доступу : 5.04.2016.
8. Xamarin application fundamentals – Режим доступу : http://developer.xamarin.com/guides/crossplatform/application_fundamentals/. – Дата доступу : 20.04.2016.
9. Xamarin.Android application fundamentals – Режим доступу : https://developer.xamarin.com/guides/android/application_fundamentals/. – Дата доступу : 13.04.2016.
10. Xamarin.iOS application fundamentals – Режим доступу : https://developer.xamarin.com/guides/ios/application_fundamentals/. – Дата доступу : 13.04.2016.
11. Working with Xamarin.Forms – Режим доступу : https://developer.xamarin.com/guides/xamarin-forms/working-with/. – Дата доступу : 15.04.2016.

12. Xamarin sharing code options – Режим доступу : https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/sharing_code_options/. – Дата доступу: 15.04.2016.
13. Presentation patterns: MVC, MVP, PM, MVVM – Режим доступу : https://manojjaggavarapu.wordpress.com/2012/05/02/presentation-patterns-mvc-mvp-pm-mvvm/. – Дата доступу : 16.04.2016.
14. Native Mobile Platform Breakdown: A Guide to Xamarin, iOS, and Android – Режим доступу : http://our.componentone.com/2016/04/12/native-mobile-platform-breakdown-a-guide-to-xamarin-ios-and-android/. – Дата доступу : 16.04.2016.
15. Activity lifecycle – Режим доступу : https://developer.xamarin.com/guides/android/application_fundamentals/activity_lifecycle/. – Дата доступу : 13.04.2016.
16. Xamarin.Android architecture – Режим доступу : https://developer.xamarin.com/guides/android/under_the_hood/architecture/ . – Дата доступу : 13.04.2016.
17. 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-visual-studio-csharp/table-of-contents/. – Дата доступу : 25.03.2016.