**Кіровоградський державний педагогічний університет**

**імені Володимира Винниченка**

Фізико–математичний факультет

Кафедра інформатики  
  
**Дипломна робота**  
з прикладної математики

на тему:

**ПОРІВНЯННЯ ЗАСОБІВ ДЛЯ КРОСПЛАТФОРМНОЇ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ**

Виконав: студент 4 курсу,

групи 46

напряму підготовки

6.040302 Інформатика

форма навчання денна

Глущенко М.В.

Керівник ??????????

Рецензент

м. Кропивницький – 2017 рік

**ВСТУП**

В останні роки особливе місце на ринку інформаційних технологій займають мобільні пристрої, а число їх користувачів постійно зростає. Більш того, за статистикою аналітичної компанії Gartner в 2013 році обсяг продажів смартфонів вперше перевищив позначку в 50% серед засобів стільникового зв'язку. У зв'язку з цим зростає попит на створення мобільних додатків під різні платформи.

Ще кілька років тому під кросплатформною розробкою розуміли створення контенту для персональних комп'ютерів (ПК), ігрових консолей та комп’ютерів фірми Apple. Тепер кросплатформність, в першу чергу, означає доступність додатку не тільки на ПК, але і на мобільних пристроях з різними операційними системами. На сьогоднішній день, головний напрямок розвитку — це ігри та додатки для смартфонів, планшетів і веб-ігри, які доступні для скачування в цифрових онлайн магазинах та браузерах.

Кросплатформність дозволяє суттєво скоротити витрати на розробку нового або адаптацію існуючого програмного забезпечення.

Уявіть собі мобільний сайт, якому не завжди потрібен доступ до інтернету, а з точки зору дизайну він ближче до мобільних додатків, ніже до веб-сторінок. Приблизно так можна описати кросплатформні додатки.

При кроссплатформній розробці використовуються спеціальні утиліти (фреймворки) для створення програми на основі сімейства мов JavaScript. Вся структура і логіка додатку створюється за допомогою таких інструментів (PhoneGap, Titanium, Xamarin, Cordova і ін.) на JavaScript, а потім обертається в нативний запускаючий елемент, тобто інтегрується в базовий проект для XCode або Android Studio, що дозволяє створювати збірки проекту з однією і тією ж логікою відразу під декілька операційних систем.

Якщо у вас є хороші кросплатформній рішення, то такий підхід дозволить вам вийти на ширшу аудиторію, витративши на це приблизно стільки ж коштів, скільки зазвичай витрачається на розробку продуктів для однієї платформи.

**Актуальність дослідження.** На сучасному етапі розвитку, засоби для кросплатформної розробки мобільних додатків відносно молоді і стрімко розвиваються.

**Об'єктом дослідження** є засоби для кросплатформної розробки мобільних додатків.

**Предметом дослідження** є порівняння засобів для кросплатформної розробки мобільних додатків.

**Мета дослідження** – знайти та дослідити декілька різних фреймворків для кросплатформної розробки мобільних додатків. Визначити їх сильні та слабкі сторони. Дати порівняльну характеристику та розробити мобільний додаток з використанням одного з даних засобів.

Відповідно до мети дослідження були поставлені такі **завдання**:

1. Знайти та проаналізувати наукові публікації на задану тему, та визначити шляхи її розв’язання.
2. На основі отриманих даних, поріняти досліджувані засоби.
3. Детально вивчити роботу одного з досліджуваних фреймворків.
4. Створити програмний додаток з використанням цього фреймворку.

Для розв’язування поставлених завдань застосовувалися такі **методи доcлiдження**: аналіз, дедукція, синтез.

**Практична значимість результатів роботи.** Практичним результатом дипломної роботи є додаток, створений за допомогою використання одного з фреймворків для кросплатформної розробки мобільних додатків, який можна демонстроції можливостей даного фреймворку.

**Структура роботи**. Дипломна робота складається з вступу, двох розділів, висновків, списку використаної літератури та додатків.

У **вступі** обґрунтовано актуальність теми дослідження, визначено його об’єкт, предмет, мету, завдання, методи, наукову новизну, теоретичне й практичне значення; представлена апробація результатів дослідження.

У **першому** розділі розглянуто декілька фреймворків для кросплатформної розробки мобільних додатків, їх можливості та порівняльна характеристика кожного з них.

У **другому** розділі описаний процес розробки мобільного додатку з використанням одного з фреймворків.

1. **КРОСПЛАТФОРМНІСТЬ**

**1.1 Загальні положення**

Не та давно, кросплатформна розробка передбачала створення одного контенту, який би працював і на персональних комп'ютерах (ПК), і на ігрових консолях, на комп’ютерах Mac, та ОС Linux. Зараз під кросплатфрмністю, в першу чергу розуміють доступність додатку не тільки на ПК, але і на мобільних пристроях, керованих різними операційними системами. На сьогоднішній день, головний напрямок розвитку — це ігри та додатки для смартфонів, планшетів і веб-ігри, які доступні для скачування в цифрових онлайн магазинах та браузерах.

При кросплатформній розробці мобільних додатків суттєво скорочуються витрати на розробку нового або адаптацію існуючого програмного забезпечення.

Щоб краще зрозуміти, що таке кросплатформний додаток, спроуйте уявити собі мобільний сайт, для користування яким не обов’яеово потрібно матідостув до мережі Інтеренет, а його зовнішній вігляд більше схожий на мобільній додаток, ніж на веб-сторіноку.

При кросплатформній розробці використовуються спеціальні інструменти (Unity, PhoneGap, Xamarin), які дозволяють створювати додатки відразу для декількох мобільних операційних систем. Каркас і логіка подібних додатків спочатку програмуються в кросплатформному інструменті, а далі код інтерпретується в проект XCode або Android Studio.

**1.2 Порівняння нативного та кросплатформного підходів**

Існує 2 основних підходи до розробки мобільніх додатків - нативний та кросплатформний. Давайте розглянемо та півняємо кожен з них.

**1.2.1 Нативний підхід**

Нативними додатками є ті, з якими ви стикаєтеся з першого дня використання пристрою. Це встановлені за замовчуванням браузер, поштовий клієнт, адресна книга, будильник, календар і інші стандартні програми.

Якщо розробники в процесі написання програми користуються прийнятою для конкретної платформи мовою програмування, будь то Objective-C і Swift для iOS або Java для Android, такий додаток буде називатися нативним (від англ. Native - рідний, природний). Нативні додатки можуть отримувати доступ до всіх служб, сервісів і функцій телефону: камери, мікрофону, геолокатору, акселерометру, календарем, медіафайлів, повідомлень і так далі.

При нативній розробці, програмісти використовують оригінальні мови програмування і інструменти мобільної операційної системи. Розробка iOS додатків ведеться в інтегрованому в OS X і iOS середовищі Xcode, на мовах Objective-C, Swift, C і С++. Для розробки додатків під Android використовується середовище Android Studio і мова Java. Середовище розробки це спеціальний комплекс інструментів, створений для максимально зручного написання коду, проектування інтерфейсу, налагодження, моніторингу та компіляції програми.

**Переваги** нативного підходу:

1. Швидкість роботи програми. Оскільки додаток створюється з використанням оригінальних інструментів розробки (XCode, Android Studio і т.д.), одержаний в результаті компіляції проекту код є оптимальним для даної платформи. Додаток отримує повну апаратну підтримку пристрою (наприклад, обробка зображень здійснюється окремим процесором, спеціально для цього призначеним - GPU), використовується многопоточность для реалізації складних завдання і завантаження контенту в фоні, в процесі розробки програмісти можуть вимірювати швидкість роботи всіх ділянок коду і при необхідності їх оптимізувати. В їх розпорядженні є інструменти для моніторингу використання оперативної пам'яті, пошуку можливих втрат і т.д.
2. Гнучкість в реалізації. На відміну від обмежень в побудові інтерфейсу і складності візуальних ефектів, що накладаються фреймворками для кросплатформної збірки проектів, в разі нативної розробки реалізувати можна все, на що здатні технології тієї чи іншої мобільної операційної системи.
3. Використання останніх технологій, залежно від кросплатформних фреймворків. Новий програмний та апаратний функціонал, наданий компаніями-виробниками обладнання й операційної системи, стає доступний для реалізації відразу після випуску відповідних оновлень. Наприклад, в iOS 9 закладена можливість пошуку усередині додатків: в кожному з них повинен бути реалізований спеціальний метод, який повертає результати за певним пошуковим запитом. В результаті для тих додатків, в яких цей функціонал реалізований, доступна можливість пошуку контенту через системний розділ пошуку в iOS, там же, де здійснюється пошук додатків, контактів, подій та іншої інформації. У випадку з кросплатформеною розробкою для реалізації подібного функціоналу доведеться чекати не тільки релізу iOS 9, але і поновлення відповідного фреймворка, причому коли з'явиться підтримка тих чи інших нових можливостей і чи з'явиться взагалі, передбачити неможливо.
4. Легкість і якість тестування. Крім згаданому в п. 1 інструментарію для контролю використання додатком апаратних ресурсів пристрою, в розпорядженні розробників і тестувальників є цілих комплекс технологій. По-перше, знову ж таки, всі параметри системи в процесі роботи програми контролюються автоматично. Якщо додаток почав використовувати більше пам'яті ніж очікується або більше ресурсів центрального процесора, це не залишиться непоміченим. По-друге, можливості в широкому застосуванні юніт-тестів - автоматичного тестування практично кожного методу в додатку. Якщо якась частина додатка перестала працювати коректно внаслідок будь-яких змін коду, нова версія просто не скомпілюється, а програміст відразу побачить причину. По-третє, доступні широкі можливості в інтеграції систем віддаленого моніторингу помилок: в кожен нативний проект вбудовується відповідний функціонал, який дозволяє побачити помилку і її причину, що виникла на пристрої будь-якого користувача.⋅
5. Повна підтримка з боку магазинів додатків App Store, Google Play, Microsoft Store і т.д. Компанії зацікавлені в тому, щоб користувачі отримували максимально позитивний досвід при використанні додатків на відповідних платформах. Це означає що програма має виглядати максимально якісно (тобто якщо у екрану висока роздільна здатність, а зображення розпливчасті, в App Store додаток просто не пропустять), має працювати настільки швидко, наскільки це можливо (якщо додаток відображає невеликий список елементів за 20-30 секунд - його так само не пропустять), і взагалі все повинно бути красиво і зручно. Якщо якісь з цих параметрів занадто низькі або взагалі не виконані, програму не пропустять в магазин. Якщо ж вони віконані на високому рівні, чого вкрай складно домогтися при кросплатформеному підході, а часто і зовсім неможливо в принципі, то ваш додаток ніколи не буде розглянутій відповідними компаніями для розміщення в спеціальних рекламних розділах(featured). Серед додатків, які перебувають в featured-розділах App Store і Google Play, немає жодного, зробленого за допомогою кросплатформних технологій, за винятком ігрових проектів, в яких інтерфейс не є системним.

**Недоліки** нативного підходу:

1. Час розробки - для кожної платформи потрібно створити окремій додаток, що займе набагато більше часу.
2. Персонал - для кожної платформи потрібен окремий програміст.
3. Бюджет - чим більше платформ повинен охопити додаток, тим більше коштів потрібно витратити на його розробку.

**1.2.2 Кросплатформний підхід**

При кросплатформній розробці використовуються засоби, які дозволяють створювати додатки відразу для декількох мобільних операційних систем. Каркас і логіка подібних додатків спочатку програмуються в цюму інструменті, а потім код трансюється у відповідні проекти операційних систем(наприклад XCode або Android Studio).

Більшість фахівців, що створюють такі додатки, користуються фреймворком PhoneGap. Його особливість полягає в тому, що він дозволяє відкрити додатком доступ до апаратних і програмних можливостей платформи. Також кросплатформна розробка можлива на таких технологіях, як Xamarin, Unity і інших, але вони не так популярні для розробки додатків, як веб-технології.

Ринок пропозицій зростає. Статистика продажів мобільних додатків показує, що рік від року користувачі гаджетів все частіше змінюють стандартні сервіси на альтернативні. Так, рідний менеджер завдань замінюється на Wunderlist, поштовий клієнт - на додаток Mailbox, Evernote виявляється переважно стандартних заміток.

Замовнику важливо знати переваги та недоліки кожного з підходів і не завищувати очікування, роблячи вибір. Проводити порівняльний аналіз буде доречно по ряду критеріїв.

**Переваги** кросплатформного підходу:

1. Потрібно менше ресурсів для реалізації програми відразу під кілька платформ. В цьому власне і суть кросплатформного підходу - один і той же код працює і на iOS, і на Android, і на інших системах. Програмістів, які займаються проектом, потрібно в декілька разів менше. Дизайнер робить тільки один набір графіки. Все це знижує кількість робочих годин і як наслідок бюджет проекту.
2. Менший час на розробку. За рахунок відсутності унікальних елементів інтерфейсу і простіших технологій, час на створення простих продуктів як правило менше.
3. Спрощений цикл оновлення продукту. Якщо в проект потрібно щось додати або виправити якусь помилку, то це робиться відразу для всіх платформ, на яких поширюється проект.
4. Можливість використання мобільної версії сайту. Більшість кроссплатформенних рішень використовують сімейство JavaScript мов, тому якщо у вас вже є мобільна версія сайту, значна частина коду і матеріалів може бути використана в додатку без змін.
5. Використання єдиної логіки. Логіка, закладена в роботу програми, буде працювати стабільно та однаково на всіх платформах. Досить часто це може бути і мінусом через різну архітектуру операційних систем (яскравий приклад - кнопка Назад в навігації між екранами: в Android передбачена апаратна кнопка Back для цих цілей, в iOS - рух пальцем від лівої частин екрана або ж наявність кнопки в лівій частини навігаційної панелі; якщо кнопку не робити зовсім, користувачі iOS не зможуть повернутися назад; якщо зробити, але не на тому місці і з нестандартнім виглядом - користувачам iOS буде незвично і незручно, якщо зробити як в iOS, буде незвично користувачам Android), однаково написана і вілагоджена один раз логіка містить потенційно меншу кількість помилок і розбіжностей в своїй роботі: вам не доведеться проробляти подвійну роботу з пошуку проблем на кожній платформі.

**Недоліки** кросплатформного підходу

1. Повільна швидкість роботи додатка.
2. Не використовуються унікальні особливості платформи.
3. Незвичний для користувача інтерфейс.

Таким чином, кросплатформні рішення **вигідні** у випадках, якщо:

1. Потрібно написати прототип додатка під кілька платформ за короткий термін, але при цьому не планується цей прототип розвивати далі.
2. При розробці складних ігор є потужні кросплатформні рішення на кшталт Unity3D, які мають багатий вбудований функціонал для розробників.
3. Якщо додаток бере всю інформацію з сервера, анімація і інтерфейс не дуже важливі, а логіка вже реалізована на сервері
4. Якщо ви можете змиритися зі скромною продуктивністю, а контент завантажується з Інтернету.

**1.2.3 Гібридний підхід**

Як видно, планка для входу в більш ніж перспективну область розробки мобільних додатків значно знизилася. Хтось може подумати, що тепер верстальники, які не йдуть далі перевірених HTML і CSS, будуть віднімати хліб у справжніх програмістів. Інші бачать за кросплатформним підходом майбутнє, в якому час і витрати на розробку додатків будуть повністю оптимізовані. З обох сторін знайдуться аргументи, що пояснюють, чому правильним є саме цей, а не інший підхід до розробки.

Але коли ми говоримо про рішення певних завдань, ефективніше буде ці підходи скомбінувати - використовувати кросплатформні переваги HTML для оформлення контенту, а вимогливі до швидкості відгуку меню і елементи управління зробити нативними, витративши на це мінімум зусиль, часу і бюджету. Такі додатки називаються гібридними. У цьому випадку тільки обсяг нативного коду визначає, якому підходу більше відповідає розробка програми.

Які ситуації призводять до злиття підходів? Припустимо, що клієнтові потрібна звичайна стрічка новин, де не буде нічого, крім тексту і зображень. Виходячи з цього завдання, розробник приймає рішення використовувати кросплатформний підхід. Але якщо через деякий час замовник побажає, щоб додаток зберігав велику кількість даних або обробляв звук і графіку, завдання ускладнюється. Для цих цілей потрібно писати нативний код під кожну конкретну платформу, і колись повністю кросплатформний додаток перетворюється в гібридний.

Є поширена помилка, що за будь-якої іконкою на робочому столі користувача чекає нативний додаток. Ця помилка пустила коріння настільки глибоко, що навіть в професійних колах грішать формулюваннями на кшталт «нативний phonegap-додаток». Але на робочий стіл можна вивести навіть ярлик для сайту, тому ікона нічого не гарантує, і по ту сторону з однаковою ймовірністю може виявитися як нативной додаток, так і будь-який інший.

**1.2.4 Залежність від платформи**

Могло скластися враження, що кросплатформеному додатком в рівній мірі комфортно на всіх платформах, аж до самих непопулярних. Але щоб це твердження відповідало дійсності, під кожну платформу, можливо, доведеться писати частину додаткового коду. У випадку ж нативних додатків можна розраховувати на їх відмінну роботу, але для кожної платформи потрібно розробляти свою версію.

**1.2.5 Дизайн інтерфейсу**

Залишити поза увагою гайдлайни в контексті розробки мобільних додатків неможливо. Гайдлайни - це цінні вказівки від компаній-виробників платформ для розробників мобільних додатків, спрямовані на те, щоб підігнати їх дизайн і функціональність під стандарти. Гайдлайни - це фундамент, на якому ґрунтується психологія і комфорт користувачів платформи. Простіше кажучи, елементи інтерфейсу мають звичний зовнішній вигляд і розташування.

Мовне середовище, в якому розробляються нативні додатки, володіє необхідними інструментами для створення звичного користувачеві інтерфейсу. Інша ситуація з веб-технологіями: щоб зробити кросплатформний додаток схожим на нативний, доведеться докласти чимало зусиль. Різні кросплатформні фреймворки (Framework 7, Sencha Touch, Kendo UI, Ionic і інші) допомагають з тим або іншим ступенем достовірності імітувати нативний інтерфейс, але частіше за все чутливість, швидкість анімації, ефекти і дизайн будуть іншими. Цьому і присвячений наступний пункт.

**1.2.6 Користувацький досвід**

Перше, чого на підсвідомому рівні чекає користувач від додатку - це чутливість. За дією користувача тут же слідує відповідна реакція, прокрутка сторінки і анімація протікають плавно і без підвисань. Кросплатформні додатки в цьому плані значно поступаються нативним, а часто взагалі залишаються поза конкуренцією, і це їх головна проблема.

Також користувач впевнений в тому, що кожен елемент управління, кожна ікона матимуть стандартний вигляд і положення на екрані. Для різних платформ ці стандарти будуть різними, і якщо кросплатформний додаток зроблений по гайдлайнам iOS, то користувачам Android це доставить дискомфорт, і навпаки.

Одним з найяскравіших прикладів може стати кнопка Back: це типова для Android функція, яка не має аналога на iOS. Тому, коли ви створюєте кросплатформний додаток, компромісів в цій ситуації може бути тільки два: або дизайн єдиний для обох платформ, і користувачі однієї з них змушені пристосовуватися, або ви створюєте два різних дизайни з урахуванням особливостей кожної платформи. По суті, у другому випадку створюються два додатки, але на одній кросплатформеній мові.

**1.2.7 Обмеження**

Нативний додаток, написаний під конкретну платформу, відчуває себе її повноправним мешканцем, отримуючи максимальний доступ до всіх функцій і сервісів пристрої. Проектуючи кросплатформний додаток, розробник враховує тільки можливості фреймворка, що накладає свої обмеження.

Проблемою моде стати і те, що у фреймворків є безліч версій, і чим старіша версія, тим більше обмежень накладається. У будь-якому випадку, кросплатформеному додатк відкриті двері далеко не до всіх можливостей платформи. Але і не завжди виникає необхідність в повній інтеграції - її глибина залежить від завдань, які має вирішувати додаток.

**1.2.8 Безпека**

Для всіх популярних браузерів існує стандартний безпечний протокол передачі даних - HTTPS. Але якщо потрібен особливий рівень шифрування, рішення цієї проблеми лягає на розробника. Забезпечення надійного захисту даних можливо тільки при нативній розробці, так як це пов'язано з математичними обрахунками, а подібні операції вимагають максимально ефективного використання апаратних ресурсів.

**1.2.9 Обслуговування та підтримка**

Комплексне обслуговування нативних додатків для двох платформ (пошук і виправлення помилок, оновлення і будь-яка незначна зміна) в середньому займає в декілька разів більше ресурсів через необхідність більшої кількості фахівців для різних платформ. З кросплатформним додатком може працювати один розробник.

**1.3 Огляд та порівняння існуючих засобів для кросплатформної розробки**

**1.3.1 Phonegap (Cordova)**

Існує плутанина між Phonegap і Cordova. Phonegap і Cordova мають такий самий взаємозв'язок, як Google Chrome і Webkit, відповідно. Apache Cordova це opensource проект, тоді як PhoneGap являє собою інфраструктуру, побудованою навколо Cordova. Крім того, інфраструктура Phonegap додатково надає такі опції як платна підтримка, онлайн-навчання, а також хмарні веб-сервіси, що дозволяють зібрати кінцевий додаток без необхідності інсталяції на локальному комп'ютері розробника. PhoneGap декларується як більш стабільна версія.

Phonegap - це сукупність API пристроїв, що дозволяє розробнику отримати доступ до нативних можливостей мобільного пристрою, таких як камера або акселерометр, із JavaScript коду. При використанні Phonegap, мобільний додаток можна створити не написавши і рядка нативного коду (Java, Objective-C, C #), а тільки з використанням HTML, CSS, і JavaScript. На відміну від web-сайту, немає необхідності завантажувати по мережі вихідний код сторінки, всі веб-ресурси вже присутні на смартфоні. Завдяки цьому, і досягається кросплатформність додатку.

Однак, додатки, створені за допомогою Phonegap, все ж упаковуються як нативні з використанням SDK платформи і можуть бути доступні для установки з цифрового магазину додатків будь-якого пристрою. На кожній платформі для відображення HTML і інтерпретації JavaScript використовується елемент користувацького інтерфейсу на основі Webkit. Наприклад, на платформі Android - це WebView. Також Phonegap має зручні утиліти командного рядка, що дозволяють при наявності відповідного SDK цільової платформи створити «скелет» додатка, в який залишиться лише додати свої HTML, CSS, і JavaScript файли, зібрати додаток або запустити його в емуляторі. Утиліти командного рядка доступні для операційних систем Windows, Linux і MacOS.

Phonegap має модульну структуру, тобто за замовчуванням бібліотека містить лише можливості для отримання версії платформи і назви пристрою. Доступ до решти можливостей платформи реалізується в плагінах. Одним з переваг такого підходу є зменшення часу завантаження програми, так як завантажуються лише плагіни, безпосередньо використовувані в додатку. Список необхідних плагінів розробник вказує в файлі конфігурації. До того ж, модульна структура дозволяє з легкістю створювати власні плагіни, розширюючи тим самим можливості платформи Phonegap.

**1.3.2 Appcelerator Titanium**

Titanium це фреймворк з відкритим вихідним кодом, який дозволяє створювати мобільні додатки для мобільних платформ, включаючи iOS, Android, BlackBerry OS, і Tizen, на основі єдиного JavaScript коду для всіх платформ. У лютому 2013 року сайт «Business Insider» підрахував, що приблизно 10% смартфонів по всьому світу мають встановлені додатки, створені за допомогою Appcelerator Titanium. За стану на 2013 рік, зареєстровано понад п'ятсот тисяч розробників Appcelerator.

Ключовим компонентом Titanium є комплект розробки програмного забезпечення Titanium SDK, який розповсюджується за ліцензією Apache. Appcelerator представляє сукупність фреймворку, заснованого на Titanium SDK, який реалізує шаблон проектування MVC і Titanium Studio, інтегрованого пропрієтарного середовища розробки, яке поширюється безкоштовно.

Весь вихідний код програми розгортається на мобільний додаток, де він інтерпретується JavaScript-двигуном. Mozilla's Rhino використовується на Android і BlackBerry, і Javascriptcore від Apple використовується на iOS. Був анонсований перехід на JavaScript-движок V8 від Google, який дозволить істотно збільшити продуктивність Appcelerator додатків, коли перехід буде завершено. Завантаження додатків займає більше часу, ніж для додатків, створених із застосуванням нативного SDK, так як інтерпретатор і необхідні бібліотеки повинні бути завантажені до початку виконання вихідного коду на пристрої.

Деякі розробники відзначають, що хоча використання Titanium і дає швидкі результати, роблячи його відмінним вибором для прототипування, існують проблеми, пов'язані з кросплатформною поведінкою API, стабільністю і управлінням пам'яттю, які змусили їх переписати додаток з використанням нативного SDK. Однак, станом на 21 липня 2013 року, в магазинах додатків було зареєстровано понад п'ятдесят п'ять тисяч додатків, створених за допомогою Appcelerator. Багато розробники називають в якості основних причин для вибору Titanium швидкість розробки, нативний користувацький інтерфейс і використання JavaScript в якості мови розробки програм.

У червні 2011 Appcelerator випустив Titanium Studio і Titanium Mobile 1.7. Titanium Studio - середовище розробки з повністю відкритим вихідним кодом, яка є похідною від Aptana Studio, придбаної компанією «Appcelerator» в січні 2011 року. У квітні 2010 року Appcelerator розширила лінійку Titanium продуктів новим фреймворком - Titanium Tablet. Titanium Tablet спирається на існуючу підтримку iPhone, але також включає нативну підтримку елементів призначеного для користувача інтерфейсу специфічних для iPad. Підтримка планшетів на базі операційної системи Android також присутня.

Базові можливості Appcelerator Titanium включають в себе багатоплатформовий API для доступу до нативних компонентів користувацького інтерфейсу, таких як: навігаційні панелі, меню, діалогові вікна, і до нативним можливостей пристрою: доступ до файлової системи, мережевих з'єднань, геолокації, акселерометру і мап.

До переваг цієї технології можна віднести відсутність необхідності вивчати будь-яку мобільну платформу і мову програмування, використовуючи досить поширений JavaScript. Крім цього, в результаті застосування даної технології можна отримати додаток, що використовує нативні елементи користувацького інтерфейсу, що позитивно впливає на його швидкості роботи та інтуїтивності взаємодії з точки зору користувача. З іншого боку, розробники Appcelerator Titanium говорять про те, що лише 60% коду є абсолютно переносимим, в інших випадках для підтримки іншої платформи доведеться враховувати її специфіку і писати під неї унікальний код, що явно не відповідає повною мірою поняттю кросплатформності.

**1.3.3 Xamarin**

Xamarin - це фреймворк для кроссплатформенной розробки мобільних додатків (iOS, Android, Windows Phone) з використанням мови C#. Ідея дуже проста. Ви пишете код на своїй улюбленій мові, із застосуванням всіх звичних для вас можливостей C#, наприклад LINQ, лямбда-виразів, generic і async. При цьому ви маєте повний доступ до всіх можливостей SDK платформи і рідного механізму створення UI, отримуючи на виході додаток, якbq, строго кажучи, нічим не відрізняється від нативних і (по завіреннях розробників) не поступається їм у продуктивності.

Фреймворк складається з декількох основних частин:

1. Xamarin.IOS - бібліотека класів для C#, що надає розробнику доступ до iOS SDK.
2. Xamarin.Android - бібліотека класів для C#, що надає розробнику доступ до Android SDK.
3. Компілятори для iOS і Android;
4. IDE Xamarin Studio.
5. Плагін для Visual Studio.

Xamarin заснований на open-source реалізації платформи .NET - Mono. Ця реалізація включає в себе власний компілятор C#, середовище виконання, а також основні .NET бібліотеки. Мета проекту - дозволити запускати програми, написані на мові C#, на операційних системах, відмінних від Windows - Unix-системах, Mac OS та інших. Важливо, що розробкою Xamarin займаються ті ж люди, що і розробкою Mono. Тобто, розробкою Xamarin займається не компанія Microsoft з усіма випливаючими плюсами і мінусами.

З точки зору виконання додатків між iOS і Android є одна ключова відмінність - спосіб їх попередньої компіляції. Як відомо, для виконання додатків в Android використовується віртуальна Java-машина Dalvik. Нативні додатки, які пишуться на Java, компілюються в якийсь проміжний байт-код, який інтерпретується Dalvik`ом в команди процесора в момент виконання програми (тобто аналогічно тому, як працює CLR в .NET). Це так звана just-in-time компіляція (компіляція на льоту). В iOS використовується інша модель компіляції - ahead-of-time (компіляція перед виконанням). Xamarin враховує цю різницю, надаючи окремі компілятори для кожної з цих платформ, які дозволяють на виході отримувати справжні, нативні додатки, які виконуються поза контекстом браузера і можуть використовувати всі апаратні і програмні ресурси платформи.

Для iOS ситуація проста - ніякої віртуальної машини немає і програмний код повинен бути просто заздалегідь скомпільований в машинний. Для цієї мети використовується AOT компілятор Mono.

Для Android все трохи складныше. При компіляції програми відбувається переклад коду з C# в проміжний байт-код, зрозумілий віртуальній машині Mono і сама ця віртуальна машина також додається в упакований додаток. І Mono і Dalvik написані на мови С і працюють поверх ядра Linux (а ми пам'ятаємо, що Android заснована на Linux). Ви вже розумієте, що відбувається. При запуску програми на Android обидві віртуальні машини починають працювати пліч-о-пліч і обмінюються даними через спеціальний механізм wrapper`ів.

Все це працює так само красиво, як і звучить. Тим не менш, є кілька зауважень:

1. Практично всі нативні класи реалізують інтерфейс IDisposable. І це не проста формальність: втрати пам'яті цілком реальні.
2. У monotouch вас, можливо, здивує відсутність таких об'єктів, як GSize, CGRect та інше. Вони замінені відповідними структурами і класами з простору імен System.Drawing.
3. Звичайно, виключення, що виникають всередині платформи, виводяться через MonoTouch.Foundation.MonoTouchException і Java.Lang.Throwable, але не завжди. Цілком реальна ситуація, коли виняткова ситуація виникає в коді фреймворка. Більш того: помилка переповнення стека часто просто критично завершує роботу додатку, додаток, без обробки виняткової ситуації.
4. Деякі елементи API просто не працюють. Наприклад, події AnimationStart, AnimationEnd в ViewFlipper (Android). Monotouch до того ж має цікаву особливість: деякі методи були перейменовані відповідно до правил .NET. Наприклад, метод UIApplication.didFinishLaunchingWithOptions перетворився на UIApplication.FinishedLaunching.
5. Багато елементів в monodroid і в monotouch представлені як проксі до нативних об'єктів. І їх життєвий цикл має слабкий зв'язок один з одним. Тому необхідно завжди пам'ятати про Dispose, інакше можливі втрати пам'яті.
6. Налагоджувач досить повільний і не завжди стабільний. Однак, розробники постійно його покращують.

Xamarin варто використовувати, якщо:

1. Ваша програма повинна містити великий обсяг мультиплатформенного коду. Рішення проблеми дублювання логіки часто важливіше потенційних проблем роботи з monotouch і monodroid.
2. Вам необхідно розробити в стислі терміни додаток під кілька платформ. Знову ж, повторне використання коду в різних платформах дозволяє відчутно прискорити розробку. Проте, варто пам'ятати, що на рішення багатьох проблем можуть піти десятки людино-годин.
3. Вам потрібно розробити невеликий додаток-прототип під IOS, але ви не знаєте ObjectiveC або Swift. При розробці прототипу, рахунок йде на години, які не варто витрачати на вивчення нової мови.
4. Ви стартапер або інді розробник. У такому випадку, ви можете не реалізовувати проблемні можливості, а кросплатформність дозволить зекономити дорогоцінний час.