**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 Паращук С.Д.

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 стають все популярніші та 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-двигун 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.

Не варто використовувати Xamarin якщо:

1. Ви розробляється не мультиплатформний додаток. Ніяка економія часу на вивчення нової мови не варта потенційних проблем.
2. Ви розробляєте GUI орієнтований додаток. Деякі інтерфейсні патерни важко реалізувати на monodroid і дуже важко на monotouch, так як рішення за замовчуванням для тієї чи іншої функції спираються на особливості платформи, які можуть просто не працювати в Xamarin.
3. Ваша програма повинна задовольняти особливим вимогам стабільності. Дійсно, часто виникають проблеми з боку платформи mono, monotouch і monodroid.

**1.3.4 Kony Platform**

Іншою популярною технологією кросплатформної розробки є Kony Platform від компанії Kony. Вона призначена для створення Web, гібридних і нативних додатків для мобільних пристроїв, планшетів і десктопних пристроїв, які написані з єдиним кодом.

Технологія Kony Platform підтримує повний цикл створення додатків, що включає в себе дизайн, розробку, тестування, розгортання додатоку на конкретному пристрої. В результаті чого повний процес розробки програми можна розділити між програмістами і дизайнерами. Вони можуть ділитися прототипами майбутньої програми та обговорювати його.

Великим плюсом є гнучкість технології Kony Platform, яка дає можливість вибирати фреймворки для розробки (JavaScript, нативні засоби - iOS, Android). Додатково технологія надає розробникам утиліти для обміну повідомленнями та сповіщеннями (push services), сервіси синхронізації (sync services). Є зручна система логування, звітів і аналітики.

На відміну від Appcelerator Titanium технологія Kony Platform має можливість налаштування авторизації, прав доступу, підвищуючи тим самим безпеку процесу розробки в цілому.

**1.3.5 IBM Worklight**

IBM Worklight V5 надає відкриту, всеосяжну платформу з розширеними можливостями для розробки мобільних додатків корпоративного класу для смартфонів і планшетів. Ця комплексна платформа, яка спирається на відкриті стандарти і використовує такі технології, як HTML5, CSS3 і Apache Cordova, допомагає організаціям будь-якого масштабу ефективно розробляти, підключати, виконувати і підтримувати засновані на HTML5, гібридні і нативні мобільні додатки.

Worklight підтримує всі етапи життєвого циклу розробки мобільних додатків, включаючи створення, розгортання, виконання і управління, та пропонує інструменти, що допомагають на кожному з цих етапів.

Worklight ефективно використовує засновані на стандартах технології та інструменти і поставляється з всеосяжної середовищем розробки, оптимізованим під мобільні додатки проміжним ПО і інтегрованою консоллю управління та аналізу в поєднанні з безліччю механізмів безпеки. Worklight дозволяє створювати функціонально насичені міжплатформні додатки без перетворення коду і використання спеціалізованих інтерпретаторів і рідкісних мов сценаріїв. Крім того, вона прискорює просування додатків на ринок і знижує вартість розробки та загальну складність, забезпечуючи найкращу взаємодію з користувачами для широкого діапазону мобільних пристроїв.

За допомогою Worklight ви можете ефективно керувати розробкою і супроводом декількох оптимізованих версій своїх мобільних додатків. Вона дозволяє одноманітно вести розробку мобільних додатків, орієнтованих на різні мобільні операційні (Android, iOS і інші). Оскільки значна частина програмного коду використовується в різних операційних системах, можливість застосування готових компонентів і узагальнення не залежних від пристрою частин коду істотно скорочує час розробки. Створені в Worklight мобільні додатки використовують загальні Web-технології на різних пристроях, не перешкоджаючи застосуванню нативних технологій і інструментів, будь то Objective-C, xCode або Android. Це дозволяє додавати нативний код в додатки, орієнтовані на конкретні операційні системи. Ваша програма може звертатися до нативним функцій мобільних пристроїв, використовуючи Apache Cordova або нативні API в поєднанні із загальними фрагментами коду.

Платформа Worklight складається з чотирьох основних складових:

1. IBM Worklight Studio представляє собою інтегроване середовище розробки (IDE) на основі Eclipse, яке дозволяє виконувати всі операції написання та інтеграції коду, необхідні для розробки повнофункціональних мобільних додатків для різних мобільних операційних систем. Користувачі Eclipse легко освоять Worklight Studio, і їм практично не знадобиться додаткове навчання для розробки мобільних додатків за допомогою інтерактивних функцій розробки коду.
2. IBM Worklight Server являє собою сервер на основі Java, який виступає в ролі масштабуючого шлюзу між додатками, зовнішніми сервісами і внутрішньою корпоративною інфраструктурою. Сервер містить функції безпеки, що забезпечують підключення, вилучення даних з декількох джерел і їх обробку, аутентифікацію, пряме оновлення Web- і гібридних додатків, аналіз та оперативне управління. Worklight Server підтримує сервер додатків IBM WebSphere і середовіще виконання Apache для запучку додатків Worklight.
3. IBM Worklight Device Runtime Components містить код, що виконується на стороні клієнта і впроваджує серверні функції в цільове середовище додатків, що розгортаються.
4. IBM Worklight Console являє собою засновану на Web-технології консоль управління, яка підтримує поточний моніторинг і управління сервером Worklight і його додатками, адаптерами і push-повідомленнями. Крім того, цю консоль можна використовувати для управління різними версіями мобільних додатків і для відправки повідомлень користувачам додатків.

Завдяки цим складовим Worklight підтримує широкий набір функцій і можливостей, в число яких входять:

1. Єдине інтерактивне середовище розробки для створення міжплатформених додатків. Worklight Studio надає середовище для розробки мобільних додатків для всіх підтримуваних мобільних платформ. Можна розробляти як мобільні Web-додатки, так і додатки для конкретних операційних систем, включаючи Android, iOS, Blackberry і Windows.
2. Відкритий підхід до інтеграції продуктів сторонніх виробників. У середовище розробки Worklight Studio можна без проблем інтегрувати бібліотеки JavaScript сторонніх виробників, такі як JQuery Mobile, Sencha Touch або Dojo Toolkit. Це дозволяє багаторазово використовувати існуючі ресурси, створені на основі цих бібліотек. У комплект поставки Worklight входить бібліотека Dojo, яку можна використовувати в будь-яких додатках. Для використання бібліотек JQuery і Sencha слід налаштувати їх під час створення мобільного застосування. Можливість використання існуючих ресурсів в розробці нових мобільних додатків може скоротити цикл розробки.
3. Потужна система аутентифікації. Worklight має вбудовану систему аутентифікації, налаштування і використання якої не вимагають особливих зусиль. Аутентифікація може виконуватися на основі форм, cookie-файлів, HTTP-заголовків або адаптерів. Worklight Studio надає редактор, що дозволяє переглядати і змінювати конфігурацію аутентифікації для будь-якої програми. Крім того, передбачена можливість створення спеціальних систем аутентифікації.
4. Загальна база програмного коду для різних операційних систем. Важливою перевагою Worklight є база програмного коду для всіх підтримуваних операційних систем. Коли ви створюєте додаток в Worklight, вона створює стандартне розташування програмного коду в папці з ім'ям common. Велика частина коду програми, необхідна для загальної міжплатформеній функціональності, може створюватися і тестуватися за допомогою цієї загальної кодової бази. Завершивши створення загальної функціональності, ви можете додати в додаток функції конкретної платформи, використовуючи специфічний для цієї платформи програмний код. Це дозволяє багаторазово використовувати загальний код і уникати дублювання навіть при розгортанні додатків, призначених для роботи в декількох операційних системах. Worklight істотно спрощує і робить зручнішии управління кодом і його супровід.
5. Підключення до внутрішньої корпоративної інфраструктури. Адаптери Worklight забезпечують інтеграцію розроблених в Worklight додатків з внутрішньою корпоративною інфраструктурою. Є готові адаптери для підключення до корпоративних СУБД, Web-сервісів і Cast Iron. Крім того, в Worklight Studio можна легко створювати нові адаптери. Можливе створення адаптерів трьох типів:

адаптер SQL;

адаптер HTTP;

адаптер Cast Iron.

Більш того, Worklight Studio пропонує опцію тестування функціональності адаптерів до їх застосування в клієнтських додатках.

1. Консоль адміністратора для управління всіма додатками. У Worklight є працююча в браузері консоль управління, яку можна використовувати для розгортання, управління і адміністрування всіх додатків і адаптерів за допомогою одного інтерфейсу. Консоль адміністратора використовується для управління додатками для всіх підтримуваних мобільних операційних систем. Крім того, вона дозволяє управляти різними версіями додатка, робити налаштування push-повідомлень і формувати звіти про активнісь користувачів.
2. Уніфіковані push-повідомлення. Push-повідомлення використовуються для відправки повідомлень на пристрої із встановленими додатками Worklight. Push-повідомлення можна налаштувати так, щоб вони розсилалися на пристрої незалежно від того, чи працює додаток на пристрої у фоновому режимі чи ні. Можна налаштувати розсилку повідомлень за допомогою Android C2DM або Apple APNS для пристроїв, що використовують Android або iOS відповідно. Worklight дозволяє відправляти повідомлення на всі пристрої, на підгрупу пристроїв або на один пристрій.
3. Можливість автономного шифрування. Worklight пропонує можливість шифрування даних, які треба зберегти на пристрої в зашифрованому вигляді. Worklight API визначає стан з'єднання програми, що дозволяє останньому вжити відповідних заходів, якщо воно працює в автономному режимі. Потім можна використовувати механізм шифрованого кеша Worklight для збереження важливих даних на пристроях в зашифрованому форматі, щоб забезпечити захист інформації, що зберігається в пристроях інформації. Ці вбудовані функції забезпечують доступність додатків і аутентифікацію в автономному режимі. Дана можливість відповідає вимогам створення захищених мобільних додатків з урахуванням доступності мережі.
4. Пряме оновлення і дистанційне відключення. Після установки додатків Worklight на пристрій, їх можна безпосередньо оновлювати при внесені змін в додаток, розгорнутий на сервері Worklight. Ця функція називається прямим оновленням. Якщо додаток Worklight запущено на пристрої у фоновому режимі, він перевіряє наявність оновлень на сервері Worklight. Якщо оновлення доступні, сервер Worklight передає поновлення в пристрій, стежачи за тим, щоб на ньому працювала поточна версія програми. Ця функція економить багато часу і зусиль, які зазвичай витрачаються на встанвлення оновлених версій. Крім того, консоль Worklight має опцію відключення будь-якої програми, роботу якої потрібно з якоїсь причини призупинити. Якщо додаток відключено, користувачам цієї програми можна відправляти повідомлення з поточною інформацією про доступність.
5. Захист додатків. Worklight пропонує кілька функцій захисту додатків. Аутентифікацію кожного додатку, що підключається до сервера Worklight можна налаштувати так, щоб модифіковані програми, не мали доступу до сервера Worklight. Цю функцію можна налаштувати за допомогою властивості testAppAuthenticity в конфігурації дескриптора будь-якої програми. Функція захищає додаток Worklight від можливих несанкціонованих змін при редистрибуції. Щоб уникнути поширення додатків з неавторизованими змінами можна використовувати обфускацію коду.
6. Збір даних для аналізу. Worklight пропонує функцію збору даних для аналізу. Дані можна збирати як на рівні сервера, так і на рівні пристрою. Збір можна налаштувати так, щоб отримані дані можна було використовувати для аналізу за допомогою різних інструментів для складання звітів. Функція складання звітів Worklight допомогає знайти активних користувачів за допомогою консолі адміністратора. Надбудови Business Intelligence Reporting Tool (BIRT) для Eclipse дозволяють створювати прості аналітичні звіти, наприклад, звіти по новим завантаженням певних програм, звіти про закономірності звертання до додатків, звіти про щоденні відвідини і щоденні звернення. Отримані аналітичні дані можна експортувати в корпоративні системи звітності. Ця функція допомагає вирішувати завдання аналізу і звітності в масштабі підприємства.

**1.3.6 Unity**

Unity - крос-платформний ігровий двигун для розробки двомірних і тривимірних додатків та ігор під різні платформи. У Unity є дві версії: безкоштовна і платна. Відрізняються вони набором можливостей, які можуть сильно знадобитися при розробці гри. По-перше, безкоштовна версія Unity3d підтримує тільки Android, Web Player, PC-платформи. Повна версія дозволяє розробникам викладати своє творіння під всі найвідоміші платформи, такі як: PC, Linux, Mac, Windows Store, IOS, Android, Windows Phone 10 Store, Blackberry 10, Wii U, PS3, Xbox 360, PS4, Xbox One. Є можливість робити софт для VR (Virtual Reality), тобто під окуляри та шолом віртуальної реальності: Hololens, Oculus Rift, StarVR та інші, а також писати програми для Kinect 2.0, LeapMotion. Повну версію Unity можна адаптувати під свої потреби: наприклад, якщо вас цікавить розробка під Kinect 2.0, ви можете прибрати деякі елементи меню в інтерфейсі і доповнити його своїми надбудовами, які полегшать розробку.

Unity3d має дуже простий Drag and Drop інтерфейс, який освоюється за місяць. Весь двигун написаний на англійській мові. Русифікації Unity немає. Unity розбитий на кілька вікон: Hierarchy, де знаходяться назви всіх об'єктів на сцені, які можна групувати і легко переходити по ним, Scene, де можна розглянути певну сцену під потрібним вам ракурсом, Inspector, який допоможе з налаштуванням виділеного об'єкта, Project, де видно всі матеріали проекту, Toolbar (або меню з інструментами).

Unity підтримує дві мови: C# (найбільш використовувана) і Javascipt. Розробнику необхідно знати одну з мов досконало, а іншу на середньому рівні, так як деякі моменти Unity робить тільки на одній з двох мов, або це робиться набагато важче, ніж на іншій мові. Передостання версія Unity, а саме Unity 4, підтримувала мову програмування Boo (діалект Phyton), але його прибрали з 5-ої версії, так як їм практично ніхто не користувався, та й документації не було. Розрахунки фізики в Unity виконує NVIDIA PhysX, яка, до речі, відмінно з цим справляється. Зовсім недавно NVIDIA представила одну цікаву річ - NVIDIA Flex, яку, можливо, в майбутньому вмонтують в ігрові двигуни.

Об'єкти в Unity можуть бути порожніми, містити компоненти, з якими взаємодіють скрипти, можуть бути названі одним і тим же ім'ям, їм можуть бути присвоєні теги, які служать для того, щоб скрипт знайшов потрібний об'єкт. До об'єктів в Unity можна привласнити колайдери: Box Collider - куб, в який потрапляє модель об'єкта, Sphere Collider - сфера, Character Collider - коллайдер, який був спеціально введений в Unity для використання персонажів, Mesh Collider - коллайдер, створений за мешу, тобто повторює геометрію об'єкта, Wheel Collider - коллайдер для коліс, Terrain Collider - коллайдер для Terrain - майданчики, яку використовують для відображення землі.

Анімувати моделі в Unity3d можна декількома способами: створення анімацій в спеціальних програмах, наприклад: 3Ds Max, Blender і інші, а можна і в самому Unity3d, так як редактор Unity має компонент для їх створення. Матеріали в Unity відіграють важливу роль. Імпортовані текстури в Unity3d прикріпити до об'єкту не можна, необхідно створити матеріал, який можна привласнювати ігровому об'єкту. До призначеним матеріалу шейдерам будуть присвоєні текстури. Шейдери можна редагувати прямо в Unity3d. Unity дозволяє генерувати нормал-мапи (normal-map), лайт-мапи (light-map), різні альфа-канали та mip-рівні. У повній версії Unity можливо повне налаштвання шейдерів, а в безкоштовній - ні.

Unity має дві дуже важливі особливості: Occlusion Culling і Level Of Detail. Обидві речі дозволяють сильно знизити навантаження на центральний процесор, завдяки грамотній деталізації. Наприклад, в іграх жанру 2D і 3D Runner при подоланні певної дистанції все, що було позаду вас, видаляється, а то, що попереду вас, генерується. Таким чином, при тривалій грі ваше Пристрій не захаращує непотрібна інформація. Occlusion Culling не візуалізує геометрію і колайдери об'єктів, що знаходяться не в полі зорі камери, а Level Of Detail замінює деталізовані об'єкти, що знаходяться далеко від гравця, на менш деталізовані, причому розробник сам налаштовує цю систему. Тобто скромний проект може дозволити виставити величезні значення в Level of Detail, коли AAA-проекти виставляють його на мінімум.

Unity 5 володіє величезною кількістю переваг перед іншими ігровими двигунами. Спільнота Unity на сьогоднішній момент є найбільшою в світі. На офіційному сайті є спеціальний розділ, в якому можна знайти статистику по ігровим двигунам. За цими даними Unity використовують понад 50% розробників відеоігор. 20% належать Unreal Engine, решта - 30%. Для розробки 2D або 3D інді-ігор Unity підходить за всіма параметрами. У Unity дуже просто будувати проекти. Причому можна створити один проект під безліч платформ, що дуже сильно полегшує процес для розробників.

Розробка ААА-проектів в Unity - найскладніший процес. По-перше, будь-який скрипт в Unity відразу тягне за собою купу помилок, які в майбутньому необхідно виправити, або переписати скрипт заново. По-друге, все ще має погану оптимізацією. Весь контент, який стоїть у вас у вікні Project, але не стоїть у вас на сцені, буде зібраний, а значить, що гра буде важити в рази більше, ніж передбачалося. У двигуні є ряд проблем зі скроллингом. При наближенні до об'єкту в певний момент камера наближається повільніше. Якщо вам потрібно максимально близько наблизитися до землі, то іноді це буває дуже складно зробити. Швидше за все, в найближчих оновленнях скролінг виправлять, або навчать їм користуватися, що теж добре. У Unity є проблеми з мультиплеерною складовою. Але, якщо у вас є достатню досвіду, то він налаштовується дуже просто.

Переваги Unity:

1. IDE: поєднання редактора сцен (в комплексі загального редактора) з редактором ігрових об'єктів і редакторів скриптів. Додатково додаються генератори дерев і террейнов.
2. Покращені можливості скриптинга, в Unity доступні три мови: JavaScript, C # і різновид Python's Boo.
3. Кросплатформеність - як уже згадувалося вище, підтримуються Windows, MacOS, Wii, iPhone, iPod, iPad, Android, PS3 і XBox 360, не всі з яких, звичайно, доступні в безкоштовній ліцензії. Ну і веб-плагін, звичайно, забувати не варто.
4. Сучасний рівень графіки, здатний конкурувати з іншими двигунами. Unity, безумовно, програє UnrealEngine за кількістю реалізованих можливостей. Однак Unity володіє такими можливостями, як deferred освітлення, стандартний набір постпроцессінгових ефектів, SSAO, прискорена опрацювання лайтмапів.
5. Дуже добре написаний фізичний двигун.
6. Масштабованість і продуктивність. Більшу частину простих процесів двигун обробляє на чудовому рівні.
7. Запуск будь-якої програми на Unity в веб-плагіні.
8. Невисока ціна за повну ліцензійну версію для серйозного веб-розробника.

Недоліки Unity:

1. Закритість коду. Неможливість отримання вихідних кодів двигуна навіть за наявності ліцензії.
2. Неможливість доповнення фізики двигуна сторонніми можливостями. Ви не зможете додати в двигун сторонню фізику, або SpeedTree.

Реальні мінуси складно визначити з першого погляду. Двигун продуктивний, стабільний і легкий в застосуванні.

У більшості малочисленних команд розробників комп'ютерних ігор основною проблемою часто ставав саме двигун. Малій команді досить складно писати свій двигун з нуля. Потрібен повноцінний безкоштовний двигун, і потрібен він відразу. Програміст починає шукати безкоштовні рішення (Ogre, Irrlicht). Ці двигуни не так вже й погані (Torchlight написаний на Ogre), але складні в освоєнні і вимагають цілої команди. Звичайно, можна звернутися до наборів типу GameMaker, але серйозну гру з його допомогою зібрати складно.

В Unity вже завершений пайплайн, готовий рендерер, фізика, аудіо, мережева взаємодія та багатомовність.

Варто дати Unity шанс - цей двигун підкуповує недосвідченого користувача своєю зручністю, гнучкістю, можливостями і швидкістю розробки.

**1.3.7 Ionic**

Ionic - це html5 фреймворк 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.

Одна з яскравих переваг у використанні Ionic для розробки мобільних додатків - це те що вам не потрібно використовувати невідомі для вас технології. Якщо ви вже розробляєте для вебу, то ви напевне знаєте HTML, CSS і JavaScript - а це саме ті технології, які лежать в основі розробки під Ionic. Час розробника - це сьогодні найдорожчий ресурс. Якщо є можливість його скоротити, то потрібно використовувати цю можливість.

У пакет 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 вебом

Установка Ionic потребує наявності Node.js і NPM. Якщо у вас вже вони встановлені, то процедура запуску Ionic не займе багато часу.

Крім Ionic автори цього фреймворка працюють над іншими проектами пов'язаними з розробкою мобільних додатків. В кінцевому підсумку всі вони створюють загальну екосистему і досить тісно пов'язані один з одним. Разом з тим, ніщо не зупиняє розробника від використання тільки деяких проектів, а не всіх відразу.

Ionicons це іконочние шрифт. Самі іконки виконані під дизайн ios7 - дуже легкі, з тонкими лініями. Дуже приємний шрифт, я думаю, що йому прекрасно знайдеться застосування при веб-розробці, а не тільки в мобільних додатках.

ngCordova - це бібліотека написана спеціально для AngularJS, вона надає доступ до Cordova API. По суті це означає, що за допомогою цієї бібліотеки у розробника з'являється можливість створювати гібридні програми з доступом до нативного функціоналу смартфона: камера, спалах, GPS і т.д. При цьому немає необхідності використовувати весь ionic - можна включити в проект тільки цю бібліотеку і розробляти додаток тільки з нею.

У висновку хочу сказати, що Ionic framework - це дуже цікавий інструмент для створення гібридних додатків. У комплексі з AngularJS він надає широкий спектр можливостей для розробника.

**1.3.8 Qt**

Qt - це багатоплатформовий фреймворк для розробки ПЗ на мові програмування C++ (і не тільки). Також є і для Ruby - QtRuby, для Python - PyQt, PHP - PHP-Qt і інших мов програмування. Розробляється компанією Trolltech з 1996 року.

З використанням цього фреймворка написано безліч популярних програм: 2ГІС для Android, Kaspersky Internet Security, Virtual Box, Skype, VLC Media Player, Opera та інші. KDE - це одне з оточень робочого столу з безліччю програм для Linux написано з використанням фреймворку Qt.

Qt повністю об'єктно-орієнтований та кросплатформний. Він дає можливість розробляти платформо-незалежне ПО, написаний код можна компілювати для Linux, Windows, Mac OS X і інших операційних систем. Включає в себе безліч класів для роботи з мережею, базами даних, класи-контейнери, а також для створення графічного інтерфейсу і безліч інших.

Qt використовує MOC (Meta Object Compiler) для попередньої компіляції програм. Оригінальний текст програми обробляється MOC, який шукає в класах програми макрос Q\_OBJECT і переводить вихідний код в мета-об'єктний код, після чого мета-об'єктний код компілюється компілятором C++. MOC розширює функціональність фреймворка, завдяки йому додаються такі поняття, як слоти і сигнали.

В Qt є величезний набір віджетів (Widget), таких як: кнопки, прогрес бари, перемикачі, checkbox, і інші - вони забезпечують стандартну функціональність GUI (графічний інтерфейс користувача). Дозволяє використовувати весь функціонал призначеного для користувача інтерфейсу - меню, контекстні меню, drag & drop.

Qt має середовище розробки Qt Creator. Вона включає в себе Qt Designer, за допомогою якого можна створювати графічний інтерфейс. Візуальне створення інтерфейсу дозволяє легко і просто створювати інтерфейс, перетягуючи різні віджети (випадають списки, кнопки, перемикачі) на форму.

Qt поставляється разом з Qt Assistant - це величезний інтерактивний довідник, що містить в собі інформацію по роботі з Qt. На жаль повністю не перекладений російською. До складу Qt також входить Qt Linguist, яка дозволяє локалізувати додаток для різних мов.

Бібліотека Qt складається з різних модулів, які підключаються за допомогою директиви #include. До складу входять:

1. QtCore - класи ядра бібліотеки Qt, вони використовуються іншими модулями.
2. QtGui - модуль містить компоненти графічного інтерфейсу.
3. QtNetwork - модуль містить класи для роботи з мережею. У нього входять класи для роботи з протоколами FTP, HTPP, IP і іншими.
4. QtOpenGL - модуль містить класи для роботи з OpenGL.
5. QtSql - містить класи для роботи з різними базами даних з використанням мови SQL.
6. QtSvg - містить класи, що дозволяють працювати з даними Scalable Vector Graphics (SVG).
7. QtXml - класи для роботи з XML.
8. QtScript - класи для роботи з Qt Scripts.

Є і інші модулі.

В даний момент Qt розповсюджується за 3-ма ліцензіями: Qt Commercial (власницька), GNU GPL, GNU LGPL.

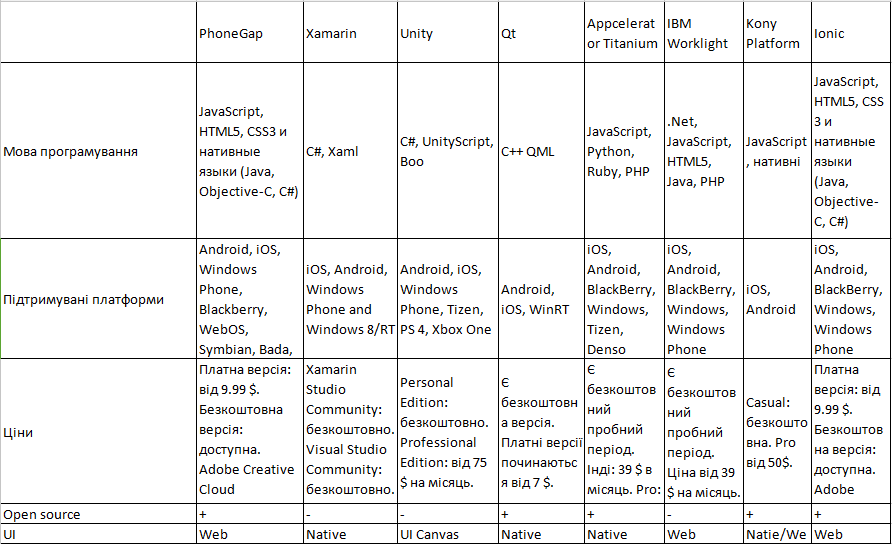
В даний час Qt фреймворк активно розвивається. Має інтуїтивно зрозуміле API, величезну документацію з великою кількістю прикладів, найпотужнішу середу розробки QtCreator і додатковий інструментарій. Цей фреймворк вартий Вашої уваги.

**1.4 Висновки**

Як правило, вихід будь-якого бізнесу в інтернет протікає за наступним сценарієм: спочатку компанія запускає сайт, потім його адаптують під мобільні пристрої, і якщо спостерігається приріст трафіку, з'являється сенс закріпитися серед власників мобільних гаджетів, і компанія випускає додаток.

Порівнювати мобільний сайт і додаток немає сенсу - другий однозначно виграє за рахунок широти своїх можливостей і швидкого інтерфейсу, взаємодіяти з яким через телефон або планшет набагато комфортніше. Крім того, додаток може працювати без постійного підключення до інтернету.

Всі розглянуті фреймворки для кросплатфрмної розробки доволі непогано показали себе в при розробці мобільних додатків, і заняли свое місце в цій ніші. Більшість з них основані на веб-технологіях (JavaScript, HTML5, CSS). Окремо можна виділити фреймворк Unity, який дуже добре підходить для написання ігор та 3D додатків. Xamarin та Cordova підходять як для початківців, так і досвідчених програмістів через свою простоту в налаштуванні та коистуванні, але водночас являють собою потужні інструменти для розробки серйозних кросплатформних мобільних додатків, які можуть конкурувати навіть з нативними. IBM Worklight надає користувачам відкриту, всеосяжну платформу з розширеними можливостями для розробки мобільних додатків корпоративного класу. Фрейморк Kony Platform хороший тим, що підтримує повний цикл створення додатків, що включає в себе дизайн, розробку, тестування, розгортання додатоку на конкретному пристрої, в результаті чого повний процес розробки програми можна розділити між багатьма програмістами.



Таблиця порівняння фреймворків

**2. ПРОЦЕС РОЗРОБКИ ПРОГРАМНОГО ДОДАТКУ**

Для розробки даного програмного додатку було вибрано фреймворк для кросплатформної розробки мобільних додатків Apachi Cordova, через його зручність, різноманість плагінів для роботі з пристроями смартфона, та використанням веб-технологій(JavaScript для реалізації функціоналу та HTML5 та CSS розмітки сторінки). Було вирішено не використовувати фреймворки для роботи з графікою, а писати на чистому JavaScript. Також були використані бібліотеки мови JavaScript — jQuery для прискорення написання програмного коду.

**2.1 Встановлення Cordova**

Для роботи фреймворк потребує наявності втановленого серверного фреймворка JavaScript - NodeJS та зособу для контролю версій Git. Після цого встановлюється сам фреймворк Cordova.

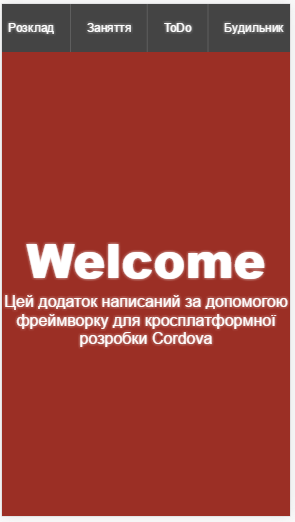
**2.2 Налаштування седовища**

Для початку роботи потрібно створити проект. Для цьго в командний рядок вводиться команда *$ cordova create hello com.example.hello HelloWorld,* де hello - і’мя майбутньої програми. Після створення проекту потрібно додати платформи, під які буде створюватись додаток. Для цього в командний рядок вводиться команда *$ cordova platform add [platform-name].* Щоб скомпылювати програму потрібно ввести команду *$ cordova build.* Для запуску додатку використовується команда *$ cordova run [platform-name].*

**2.3 Розробка додатку**

Додаток повинен виконувати функції: введеня розкладу занять, введення розкладу дзвінків, веденя заміток по кожнму предмету з функцією нагадування, функція будильника на основі розкладу дзвінків та ToDo-list.

Спочатку було реалізовано початкову сцену привітання користувача та меню додатку.



*</head>*

*<body>*

*<div id="mainmenu">*

*<ul>*

*<li><a id="rozklad"> Розклад </a></li>*

*<li><a id="chas"> Заняття </a></li>*

*<li><a id="todo"> ToDo </a></li>*

*<li><a id="alarm"> Будильник </a></li>*

*</ul>*

*</div>*

*<div class="testt" id="welcomediv"><br><br><br><br><br><br><br><br><br><br><br><br><p id="www">Welcome</p><p id="ccc">Цей додаток написаний за допомогою фреймворку для кросплатформної розробки Cordova</p></div>*

*html, body {*

*min-width: 100%;*

*min-height: 100%;*

*//background-color: red;*

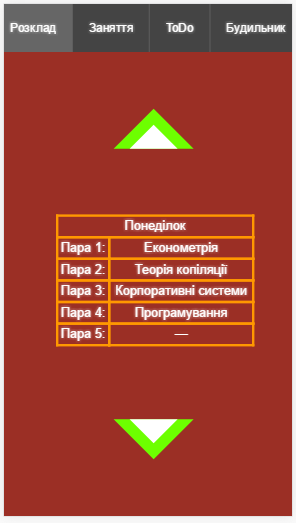
*text-shadow: 0px 0px 6px rgba(255,255,255,0.7);*

*color: white;*

*font-family: "Arial";*

*}*

Далі було розроблено пункт меню для ввелення та переглядання розкладу занять. Для цього було використано бібліотеку Bootstrap та Material.



*<div class="testt" id="rozkladdiv">*

*<div class="rozkswitch arrow up" id="rozkleft"></div>*

*<div id="day1"></div>*

*<div id="day2"></div>*

*<div id="day3"></div>*

*<div id="day4"></div>*

*<div id="day5"></div>*

*<div class="rozkswitch arrow down" id="rozkright"></div>*

*</div>*

*.day {*

*text-align: center;*

*}*

*#day2, #day3, #day4, #day5 {*

*display: none;*

*text-align: center;*

*float: left;*

*// margin: 50% 13%;*

*position: absolute;*

*top: 35%;*

*left: 18%;*

*}*

*.rozkswitch {*

*height: 50px;*

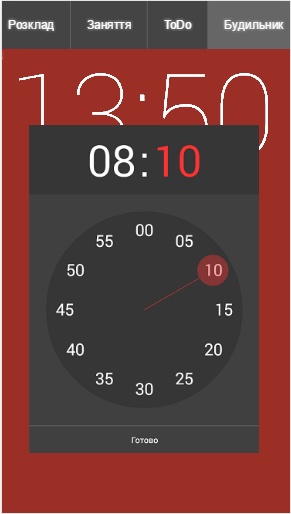
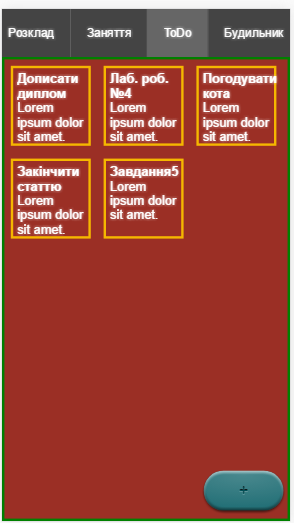
*width: 50px;*

*border: solid black;*

*position: absolute;*

*}*

Далі були розроблені фіункція будильника та RoDo-list. Для них довелось використовувати плагіни для з репозиторію Cordova для взаємодії з функціями смартфону.

*$('#todo').click(function(){*

*$('#' + blockId).fadeOut('fast');*

*$('#tododiv').fadeIn('fast');*

*blockId = $('#tododiv').attr('id');*

*});*

*$('#alarm').click(function(){*

*$('#' + blockId).fadeOut('fast');*

*$('#alarmdiv').fadeIn('fast');*

*blockId = $('#alarmdiv').attr('id');*

*});*

*$('#tododiv').html($('#tododiv').html()+ "<div class='todotask'><strong>Дописати диплом</strong><br>Lorem ipsum dolor sit amet.</div>"+*

*"<div class='todotask'><strong><br>Lorem ipsum dolor sit amet.</div>"+*

*"<div class='todotask'><strong><br>Lorem ipsum dolor sit amet.</div>"+*

*"<div class='todotask'><strong></strong><br>Lorem ipsum dolor sit amet.</div>");*

*$('#tododiv').html( $('#tododiv').html()+"<div class='todotask'><strong></strong><br>Lorem ipsum dolor sit amet.</div>");*