Міністерство освіти і науки України
Державний ВНЗ «Національний гірничий університет»

Факультет інформаційних технологій
(факультет)

Кафедра програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
дипломної роботи

магістра
(назва освітньо-кваліфікаційного рівня)

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)

спеціальність *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

спеціалізація *Програмне забезпечення систем*
(код і назва спеціалізації)

освітній рівень *магістр*
(назва освітнього рівня)

кваліфікація *2132.2 інженер-програміст*
(назва кваліфікації)

на тему: *Дослідження механізмів управління ресурсами в операційних системах Android та iOS для розробки мобільних додатків*

Виконавець:
студент *6* **курсу , групи** *121М-16-1*

(підпис) *Козир А.В.* (прізвище та ініціали)

Керівники	**ПОСАДА, ПРІЗВИЩЕ, ІНІЦІАЛИ**	**Оцінка**	**ПІДПИС**
проекту	*проф. Куваєв В.М.*		
розділів:			
Економічний	*доц. Касьяненко Л.В.*		
Рецензент	*проф. Головко В.І.*		
Нормоконтроль	*доц. Коротенко Л.М.*		

Дніпро
2018

Міністерство освіти і науки України
Державний вищий навчальний заклад
«Національний гірничий університет»

ЗАТВЕРДЖЕНО:
завідувач кафедри

програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

«___» _____ 20 ____ року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

студенту *121М-16-1* *Козир А.В.*
(група) (прізвище та ініціали)

Тема дипломної роботи *Дослідження механізмів управління ресурсами в операційних системах Android та iOS для розробки мобільних додатків*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора Державного ВНЗ «НГУ» від __.__.*2017 р.* № ____*-л*

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – механізми управління ресурсами в операційних системах Android та iOS.

Предмет досліджень – принципи виконання оптимізаційних дії під час розробки мобільного додатку для операційних систем Android та iOS.

Мета НДР – змістовне обґрунтування недоліків сучасного стану виконання оптимізаційних дій серед розробників мобільних додатків. Розробка мобільного додатку, для наочної демонстрації відмінностей та подібностей у використанні ресурсів мобільних пристроїв, що працють під керуванням Android та iOS, у реальному часі. Удосконалення методики проектування мобільних додатків з використанням особливостей їх оптимізації.

Вихідні дані для проведення роботи: результати переддипломної практики, рішення алгоритму для мобільного програмного забезпечення з використанням методів оптимізації.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна результатів, що очікуються, полягає у: проведенні аналізу та виявленні недоліків традиційного підходу до розробки мобільних додатків для операційних систем Android та iOS, а також у створенні удосконаленної методики проектування мобільних додатків з використанням початкової оптимізації.

Практична цінність результатів полягає у: створенні програмних модулів та програмнмх продуктів, які дозволяють оцінити переваги проектування оптимізованих мобільних додатків, розробці мобільного додатку з використанням технології оптимізації динамічних бібліотек.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати досліджень мають бути подано у вигляді, що дозволить побачити реальні переваги у використанні напрацьованих методів оптимізації в реальному мобільному додатку.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець)**
Аналіз стану проблеми у літературі, профільних мережевих ресурсах та онлайн спільнотах. Корпоративних газетах та журналах. Періодичних та неперіодичних виданнях по розробці програмного забезпечення.	15.09.17-15.10.17
Вивчення літератури, теоретичний аналіз проблеми, висунення гіпотези, завдань, формування структури мобільного додатку для операційних систем Android та iOS	15.10.17-15.11.18
Розробка удосконаленного методу оптимізації мобільних додатків під управлінням операційних систем Android та iOS	15.11.17-15.12.17
Узагальнення й систематизація матеріалів, написання магістерської роботи	15.12.17-15.01.18

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект від реалізації результатів роботи очікується нейтральний або майже нейтральний, тому що результат роботи представляє собою виконання дослідження, до цілей якого не входило отримання прибутку від його реалізації. Однак, треба відзначити, що є можливість позитивного ефекту у майбутньому, якщо розроблена удосконалена методика проектування мобільних додатків зарекомендує себе на практиці серед сторонніх розробників мобільних додатків. У такому випадку,

можно очікувати зменшення часу необхідного на проведення оптимізації мобільних додатків після деякого часу від їх розробки. Це можливо завдяки скороченню як матеріальних, так і часових витрат, а також сприяє збільшенню кількості різноманітних девайсів, які успішно підтримують розроблені за даною методикою мобільні додатки, що буде сприяти збільшенню аудиторії потенційних користувачів. Завдяки цьому, кінцевий можливий прибуток з реалізації такого мобільного додатку буде вищим, у порівнянні із стандартно розробленим мобільним додатком.

Соціальний ефект від реалізації результатів роботи очікується позитивним завдяки можливості використання удосконаленнох методики оптимізації мобільних додатків через модель безкоштовного розповсюдження через мережу Internet. У такому випадку, будь-який сторонній розробник мобільних додатків має змогу протестувати розроблений методику та отримати потрібні для себе результати.

7 ДОДАТКОВІ ВИМОГИ

Відповідність оформлення ДСТУ 3008 – 95. “Документація. Звіти у сфері науки і техніки. Структура і правила оформлення ”.

Завдання видав _____ *Куваєв В.М.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Козир А.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *11.09.2017 р.*

Термін подання дипломного проекту до ДЕК _____

Реферат

Пояснительная записка: 127 с., 41 рис., 3 прил., 16 источников.

Объект исследования: механизмы управления ресурсами в операционных системах Android и iOS. Их схожесть, отличия и особенности оптимизации.

Цель магистерской работы: разработка мобильного приложения, для наглядной демонстрации различия и сходства в использовании ресурсов мобильных устройств под управлением операционных систем Android и iOS во время их работы. Совершенствование методики проектирования мобильных приложений с использованием особенностей их оптимизации.

Методы исследования. При решении поставленной задачи использовались научные достижения в областях оптимизации программного обеспечения среди как физических так и программных методов для мобильных операционных систем.

Научная новизна полученных результатов заключается в проведении анализа и выявлении недостатков традиционного подхода к разработке мобильных приложений под управлением операционных систем Android и iOS, а также в создании усовершенствованной методики проектирования мобильных приложений с использованием изначальной оптимизации.

Практическое значение работы заключается в создании программных модулей, программного продукта, которые позволяют оценить преимущества проектирования оптимизированных мобильных приложений, разработке программного обеспечения с использованием технологии оптимизации динамических библиотек.

Область применения. Разработана методология проектирования оптимизированных мобильных приложений может применяться для решения широкого спектра задач, в частности, для создания изначально оптимизированного программного обеспечения.

Значение работы и выводы. Усовершенствованная методика проектирования мобильных приложений с использованием особенностей их оптимизации позволяет проектировать мобильный приложения со значительным сокращением как материальных затрат, так и временных, а также способствует увеличению количества разнообразных девайсов, которые успешно поддерживают разработанные по данной методике мобильные приложения, что способствует увеличению аудитории потенциальных пользователей.

Прогнозы по развитию исследований. Разработать универсальные программные модули, которые могут быть использованы для поддержки проектирования новых мобильных приложений для различных программных платформ. Разработать комплекс программных средств и пользовательский интерфейс для графического представления результатов, сравнительного анализа разработки с использованием различных методов оптимизации мобильных приложений.

В разделе «Экономика» проведены расчеты трудоемкости разработки программного обеспечения, расходов на создание ПО и длительности его разработки, а также проведены маркетинговые исследования рынка сбыта созданного программного продукта.

Список ключевых слов: iOS, Android, OS, Xcode, Android-Studio, RAM, eggPlant, Calabash, Systrace, Swift, ObjectC, Java, ДОДАТОК, СМАРТФОН.

Реферат

Пояснювальна записка: 127 с., 41 рис., 3 додатків., 16 джерел.

Об'єкт дослідження: механізми управління ресурсами в операційних системах Android та iOS.

Мета магістерської роботи: розробка мобільного додатку, для наочної демонстрації відмінності та подібностей у використанні ресурсів мобільних пристроїв на операційних системах Android та iOS під час їх роботи. Удосконалення методики проектування мобільних додатків з використанням особливостей їх оптимізації.

Методи дослідження. Під час вирішення поставленого завдання використовувалися наукові досягнення в областях як фізичних так і програмних методів оптимізацій програмного забезпечення для мобільних операційних систем.

Наукова новизна отриманих результатів полягає у проведенні аналізу та виявленні недоліків традиційного підходу до розробки мобільних додатків для операційних систем Android та iOS, а також у створенні удосконаленної методики проектування мобільних додатків з використанням початкової оптимізації.

Практична цінність полягає у створенні програмних модулів та програмнмх продуктів, які дозволяють оцінити переваги проектування оптимізованих мобільних додатків, розробці мобільного додатку з використанням технології оптимізації динамічних бібліотек.

Область застосування. Розроблена методологія проектування оптимізованих мобільних додатків може застосовуватися для вирішення широкого спектра завдань, зокрема, для створення початково оптимізованого програмного забезпечення.

Значення роботи та висновки. Удосконалена методика проектування мобільних додатків з використанням особливостей їх оптимізації дозволяє проектувати мобільний додатки зі значним скороченням як матеріальних витрат, так і часових, а також сприяє збільшенню кількості різноманітних девайсів, які успішно підтримують розроблені за даною методикою мобільні додатки, що напрямую сприяє збільшенню аудиторії потенційних користувачів.

Прогнози щодо розвитку досліджень. Розробити універсальні програмні модулі, які можуть бути використані для підтримки проектування нових мобільних додатків для різних програмних платформ. Розробити комплекс програмних засобів і призначений для користувача інтерфейс для графічного представлення результатів, виконати порівняльний аналіз розробки з використанням різних методів оптимізації мобільних додатків.

У розділі «Економіка» проведені розрахунки трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки, а також провести маркетингові дослідження ринку збуту створеного програмного продукту.

Список ключових слів: iOS, Android, OS, Xcode, Android-Studio, RAM, eggPlant, Calabash, Systrace, Swift, ObjectC, Java, ДОДАТОК, СМАРТФОН.

The abstract

Explanatory note: 127 p., 41 fig., 3 applications, 16 sources.

Object of research: the mechanisms of resource management in Android and iOS operating systems. Their similarity, differences and features of optimization.

The purpose of the degree project: the development of a mobile application for a clear demonstration of the differences and similarities in the use of mobile resources on Android and iOS operating systems during their work. Improving the design of mobile applications with the use of features of their optimization.

Methods of research. While solving the problem, scientific advances were used in the areas of both physical and software optimization methods for mobile operating systems software.

The scientific novelty of the results are to analyze and identify the disadvantages of the traditional approach to the development of mobile applications for Android and iOS operating systems, as well as to create an advanced methodology for designing mobile applications with the use of initial optimization.

.**The practical value of work** is to create software modules and software products that allow you to assess the benefits of designing optimized mobile applications, developing a mobile application using the technology of optimizing dynamic libraries.

The scope. The developed methodology of designing optimized mobile applications can be used to solve a wide range of tasks, in particular, for the creation of initially optimized software.

The value of the work and conclusions. An advanced methodology for designing mobile applications with the use of their optimization features allows you to design mobile applications with significant reductions both in material costs and in time, and also contributes to the increase in the number of various devices that successfully support the mobile applications developed by this method, which directly contributes to increasing the audience of potential users.

Projections on development research. Develop universal software modules that can be used to support the design of new mobile applications for various software platforms. Develop a set of software tools and a user interface for graphical presentation of results, comparative analysis of development using various methods of optimizing mobile applications.

In section "Economics" calculated the complexity of software development, the cost of creating the software and the duration of its development, and marketing studies market created by the software.

List of keywords: iOS, Android, OS, Xcode, Android-Studio, RAM, eggPlant, Calabash, Systrace, Swift, ObjectC, Java, ДОДАТОК, СМАРТФОН.

Зміст

Перелік скорочень

ПК – персональний комп’ютер

ОС – операційна система

ПО – програмне забезпечення

БД – база даних

СУБД – системи управління базами даних

RAM – оперативна пам’ять

LY – шар зображення

PM – перед головний запуск

AM – після головний запуск

UI – інтерфейс користувача

LV – зіставний елемент інтерфейсу користувача

SD – затримка побудови графіку

CPU – центральний процесор

Вступ

Актуальність роботи. В даний час існує досить велика кількість платформ для розробки і пристроїв, що їх підтримують. З плином часу, прогрес бере своє і все більша кількість людей переходить на смартфони ─ мобільні телефони з сенсорними екранами, які в більшості своїй працюють під управлінням операційної системи Android або iOS. Звичайно, що попит народжує пропозицію, через що майже відразу почалася масова розробка мобільних додатків для цих та інших операційних систем. Однак, така ситуація не означає для більшості розробників такого типу ПЗ те, що при їх розробці будуть використовуватися механізми щодо оптимізації та підвищення стабільності роботи цих додатків для більшості вироблених пристроїв. Головна причина цього явища – відсутність єдиного механізму забезпечення оптимізації під певну операційну систему. Керівникам великих проектів спочатку доводиться витрачати значні матеріальні засоби для придбання і розробки мобільних додатків, їх супроводу, публікації в засобах поширення і т. д. При цьому, роль оптимізації все частіше відходить на другий або третій план, особливо останнім часом, коли стали виробляти смартфони, які практично вже не поступаються за своєю продуктивністю від стаціонарних комп'ютерів. В цьому випадку необхідні дії для проведення оптимізації мобільних додатків можуть бути проігноровані розробником, який замість цього може орієнтуватися на те, що розроблений додаток буде відмінно працювати на самих останніх і, відповідно, найпотужніших смартфонах, що в даний момент представлені на ринку. Перерахування подібних ситуацій можна продовжити, але вже зі сказаного, очевидно, що необхідно якимось чином вирішувати проблеми початкової оптимізації мобільних додатків, щоб уніфікувати всі наявні нині і передбачувані в майбутньому додатки в плані побудови їх правильної оптимізації.

Цілі і завдання дослідження. Метою даної магістерської роботи є створення мобільного додатку для кожної з операційних систем, що наочно продемонструють відмінності і схожість у використанні ресурсів мобільних пристроїв під управлінням операційних систем Android і iOS під час їх роботи.

Удосконалення методики проектування мобільних додатків з використанням особливостей їх оптимізації.

Для досягнення поставленої мети в роботі сформульовані і вирішені такі завдання:

1. Проведення аналізу і виявлення недоліків існуючого підходу до розробки мобільних додатків.
2. Створення методики проектування мобільних додатків на основі використання технології многоваріативності підходу до оптимізації ресурсів смартфону при розробці нового мобільного додатку.

Об'єкт идослідження – механізми управління ресурсами в операційних системах Android та iOS. Їх схожість, відмінності і особливості оптимізації.

Предмет дослідження – методики проектування оптимізованих мобільних додатків.

Ідея роботы полягає в удосконаленні методики розробки початково оптимізованих мобільних додатків за рахунок зниження витрат як матеріальних, так і часових.

Методы дослідження. При вирішенні поставленого завдання використовувалися випробувані і перевірені досвідом досягнення в областях оптимізації розробки і функціонування мобільних додатків і програмного забезпечення для операційних систем Android і iOS.

Наукові положення, очікувані наукові результати.

1. Сформірованний аналіз традиційного підходу до розробки мобільних додатків, а також виявлення недоліків;
2. Створення методики проектування оптимізованих мобільних додатків на основі використання випробуваних і перевірених досвідом досягненнь в областях оптимізації розробки і функціонування мобільних додатків і програмного забезпечення для операційних систем Android та iOS.

Обгрунтованість і достовірність наукових положень

Обгрунтованість і достовірність наукових положень, висновків і рекомендацій магістерської роботи обгрунтована коректністю поставлених проблем і прийнятих припущень при описі процесів, що відбуваються при оптимізації, обґрунтованістю вихідних посилок, достатнім обсягом вибірки даних і верифікованими на явних результатах отриманих шляхом тестування.

Наукова новизна отриманих результатів полягає в створені методики проектування мобільних додатків з впровадженою технологією підходу до розробки з початковою оптимізацією нових мобільних додатків для ОС Android і iOS.

Практичне значення отриманих результатів полягає в розробці двох додатків першого для Android, другого для iOS; що дозволяє оцінити переваги проектування початково оптимізованих додатків, розробки програмного продукту з використанням різних методів оптимізації додатків.

Особливий внесок магістра складається в:

- виборі методів досліджень і технологій реалізації;
- створення мобільних додатків, що реалізують механізми початково оптимізованого підходу до розробки нових мобільних додатків;
- розробці теоретичної частини роботи, в якій досліджені і систематизовані знання про існуючі підходи оптимізації мобільних додатків і досвід їх використання;
- оцінці отриманих результатів.

Апробація результатів магістерської роботи.

Основні положення і результати були докладені та обговорені на студентській науковій конференції.

Структура і обсяг роботи. Робота складається з вступу, трьох розділів і висновків. Містить 114 сторінок друкованого тексту, в тому числі 64 сторінки тексту основної частини з 32 рисунками, списку використаних джерел з 30 найменуваннями на 3 сторінках, 5 додатків на 30 сторінках.

РОЗДІЛ 1.
АНАЛІЗ ПИТАННЯ «АРХІТЕКТУРА, ІСТОРІЯ ТА ВИДИ РЕАЛІЗАЦІЇ МОБІЛЬНИХ ДОДАТКІВ»

1.1 Історія і передумови появи мобільних операційних систем Android та iOS

Ні для кого не секрет, що XXI століття — це століття інформаційних технологій. IT міцно увійшли в наше життя: вони оточують нас всюди. Як правило, людина, натискає педаль газу в своєму автомобілі, може навіть не здогадуватися, що за його рухами стежать сотні датчиків і мікропроцесорів, покликаних полегшити йому життя, а часто і врятувати її. В сучасні машини виробники вбудовують все більш і більш хитромудрі функції. Зважаючи на це з'явилася необхідність зручної взаємодії користувача і всієї навколишньої інфраструктури. З'явилася необхідність у використанні якогось пульта управління електронікою. Інженери Apple, а потім і Google, знайшли рішення. Вони створили операційну систему для телефонів, що дозволяє з легкістю розробляти свої додатки, починаючи від можливості читати електронні книги з екрану мобільних телефонів і закінчуючи управлінням побутовою технікою в «розумному будинку».

Компанія Google пішла далі в своїх амбітних планах і створила відкриту архітектуру Android. Android (Андроїд) — портативна (мережева) операційна система для комунікаторів, планшетних комп'ютерів, електронних книжок, цифрових програвачів, наручних годинників, нетбуків і смартбуков, заснована на ядрі Linux. Спочатку розроблялася компанією Android Inc., яку потім купила Google. останнє десятиліття спостерігається процес активної розробки різних інформаційних технологій рівня підприємства, таких, як перераховані програмні платформи типу Microsoft .NET, CORBA, JAVA і т. д.

Більше 75% смартфонів, проданих в третьому кварталі 2015 року, були оснащені операційною системою Android. Тепер кожен розробник електронного пристрою має можливість переробити Android під свій пристрій, таким чином гарантуючи сумісність свого устаткування зі сторонніми додатками для цієї ОС.

Це виявилося дуже вигідно. Якщо до виходу Android кожен виробник електронного пристрою самостійно писав або купував у кого-то операційну систему, втрачаючи таким чином масу корисних програм, створених програмістами всього світу, то після виходу ОС Android перед виробниками частіше постає питання, яку версію Android їм потрібно підтримувати.

iOS вперше була представлена як iPhone OS 1, «мобільна версія Mac OS», разом з самим першим смартфоном компанії 9 січня 2007 року. Як і «комп'ютерна» операційна система Apple, iOS побудована на двох основних компонентах, розроблених Apple і NeXT — це ядро XNU і система Darwin, яка формально відповідає специфікації SUSv3 «Юнікса», не будучи їм як таковим. Single UNIX Specification — загальна назва для сімейства стандартів, яким повинна задовольняти операційна система, щоб називатися «UNIX». При цьому XNU і Darwin, на відміну від iOS і macOS — системи з відкритим початковим кодом. Початковий код таких програм доступний для перегляду, вивчення та зміни, що дозволяє користувачеві взяти участь в доопрацюванні цієї відкритої програми, використовувати код для створення нових програм і виправлення в них помилок — через запозичення початковоого коду.

Закритість iOS дозволяє Apple контролювати не тільки власний софт, а й сторонніх розробників. Наприклад, незважаючи на безкоштовність iOS SDK і середовища розробки Xcode, кожен розробник змушений платити 99 доларів у рік за можливість використовувати емулятор Apple-пристроїв і розміщувати свої додатки в App Store. До березня 2008 року, коли SDK був викладений у відкритий доступ, сторонні розробники не могли писати програми під iPhone OS в принципі, магазин додатків з'явився тільки в другій версії ОС.

Істотну роль зіграло поява Android та iOS в комунікації між людьми. Установка їх на телефони дала можливість з легкістю розробляти нові моделі мобільних пристроїв, розширюючи функціонал — як телефонів, так і самих операційних систем. Поява програм (додатків), призначених для допомоги користувачеві в самих різних ситуаціях (наприклад, існує додаток, що

використовує вбудовані датчики для вимірювання кутів повороту, швидкості об'єкта і т.д.), призвело до того, що на сьогоднішній день людині, коли вона вирушає в подорож, досить просто мати з собою мобільний пристрій на ОС Android або iOS. Користувачеві надаються послуги бронювання готелів, пошуку авіаквитків, різноманітні додатки-гіди, а спеціальні карти служать для пошуку і прокладки маршруту до пункту призначення. При використанні SIPтелефонії можна мати комунікації на міжміському і міжнародному рівнях [1]. Все це, разом з легкістю розробки додатків, робить розглянуті платформи одними з найбільш перспективних для комунікації в сучасному суспільстві.

1.2 Огляд інструментів, що будуть використані для розробки мобільних додатків на операційних системах Android та iOS

На сьогоднішній день існує великий вибір мов програмування для розробки мобільних додатків. Це пов'язано з тим, що для різних мобільних пристроїв доводиться використовувати різні мови програмування, що обумовлене тим, що мобільні пристрої мають різні операційні системи (ОС).

Цільови платформи (iOS, Android) будуть мати значний вплив на мову розробки, яка буде використовуватися. Наприклад, можна розробляти рідні додатки для кожної платформи або використовувати сторонній інструмент для оптимізації своїх додатків на різних платформах. Другий підхід може заощадити час і зусилля, хоча це може вплинути на зручність використання. Сучасні мобільні пристрої пропонують широкий спектр варіантів розробки.

Процес розробки програмного забезпечення для мобільних пристроїв вимагає підтримувати певні обмеження щодо специфіки роботи додатків у мобільних операційних системах. Особливості роботи мобільних додатків стосуються перш за все:

- на витрати живлення акумуляторної батареї мобільного пристрою;
- обмеження на кількість даних, що передаються через мережу Інтернет;

- зменшену швидкість передачі пакетів в мобільному Інтернет з'єднанні;
- можливість втрати пакетів при передачі в мобільному Інтернет з'єднанні;
- кількість обмінів даними з зовнішніми пристроями через Bluetooth;
- безпеку даних користувача;
- значну фрагментацію версій операційних систем та фреймворків;
- велику різноманітність розмірів екранів мобільних пристроїв;
- обмеження запитів до системи визначення місцезнаходження абонента;
- обмеження на розмір файлу додатку;
- порівняно невеликий ліміт оперативної пам'яті мобільного пристрою;
- порівняно обмежену швидкість передачі даних в мобільному Інтернет з'єднанні.

1.2.1 Інструментарій Android

Розробка додатків для платформи Android ведеться переважно на мові Java. Для створення програм на мові Java необхідне спеціальне програмне забезпечення. Найостанніші версії цього ПО можна завантажити з офіційного сайту розробника, Oracle Corporation. До цього програмного комплексу відносяться такі інструменти як JRE (Java Runtime Environment) і JDK (Java Development Kit). Перший інструмент являє собою середовище виконання — мінімальну реалізацію віртуальної машини, в якій запускається і виконується програмний код на Java. Другий інструмент — це в свою чергу цілий набір інструментів, комплект розробника додатків на мові Java. Насправді, JRE також входить до складу JDK, так само як і різні стандартні бібліотеки класів Java, компілятор javac, документація, приклади коду і різноманітні службові утиліти. Весь цей набір поширюється вільно і має версії для різних ОС, тому будь-хто

може його скачати і використовувати. В JDK не входить інтегроване середовище розробки, передбачається, що її розробник буде встановлювати 15 окремо. Існують численні IDE для Java-розробки, наприклад, NetBeans, IntelliJ IDEA, Borland JBuilder і інші. Таким чином, перш ніж приступити до розробки програми на базі ОС Android, необхідно підготувати інструментарій.

При розробці додатків на базі ОС Android необхідно використовувати середу Android Studio. На сайті IDE можна знайти і скачати Android Studio для своєї платформи (див. рис. 1.1).

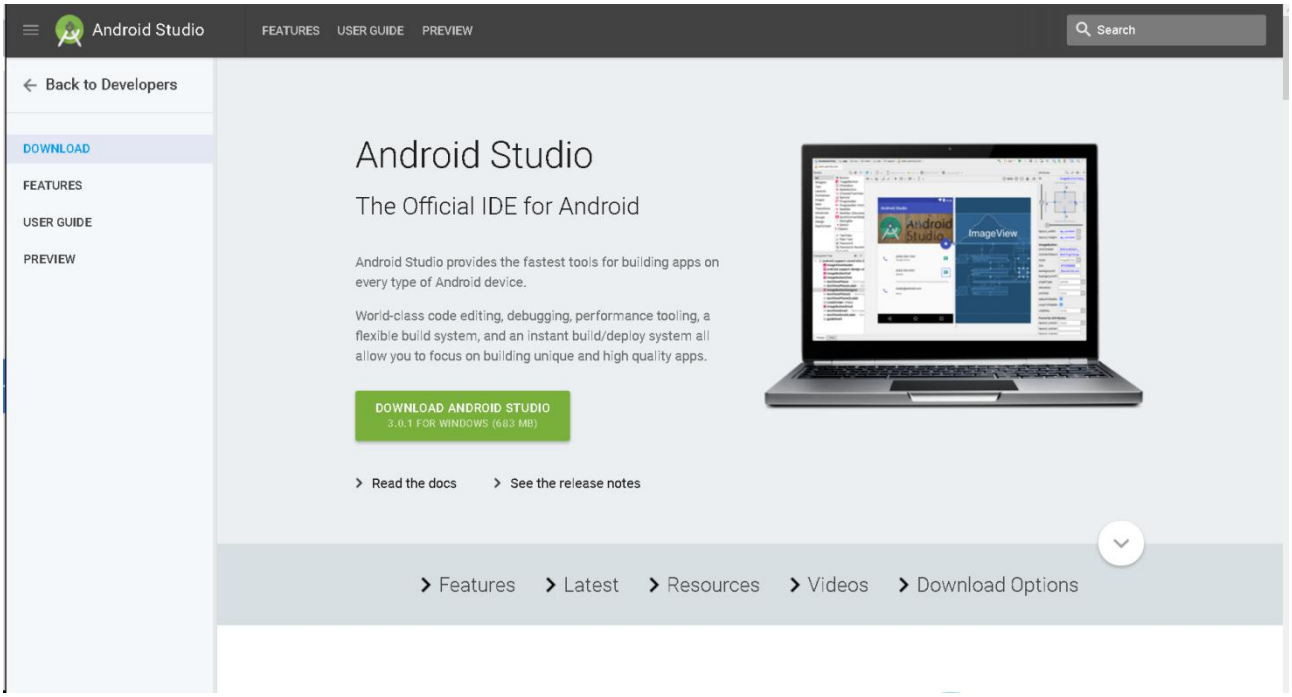


Рис. 1.1. Екран завантаження Android Studio

1.2.2 Інструментарій iOS

Розробка додатків для платформи iOS ведеться переважно на мові Swift та Objective-C. Для створення програм на цих мовах необхідне спеціальне програмне забезпечення. Найостанніші версії цього ПО можна завантажити з офіційного сервісу поширення програмного забезпечення для iOS та macOS — App Store. До цього програмного комплексу відносяться такі інструменти як IDE Xcode. Xcode — це пакет інструментів для розробки додатків під Mac OS X і iPhone OS, розроблений Apple. IDE Xcode поширюється вільно, тому будь-хто

може його скачати і використовувати. Таким чином, перш ніж приступити до розробки програми на базі ОС iOS, необхідно підготувати інструментарій [2].

При розробці додатків на базі ОС iOS необхідно використовувати середу IDE Xcode. Через офіційний сервіс поширення програмного забезпечення для операційних систем iOS та macOS — App Store (див. рис. 1.2).

Рис. 1.2. Екран перегляду Xcode у офіційному сервісі поширення програмного забезпечення для операційних систем iOS та macOS

1.2.3 Огляд середовищ розробки

Середовище розробки Android Studio — інтегроване середовище розробки (IDE) для платформи Android, представлене 16 травня 2013 року на конференції Google I/O менеджером по продукції корпорації Google — Еллі

Паверс 8 грудня 2014 року компанія Google випустила перший стабільний реліз Android Studio 1.0 (див. рис. 1.3) [14].

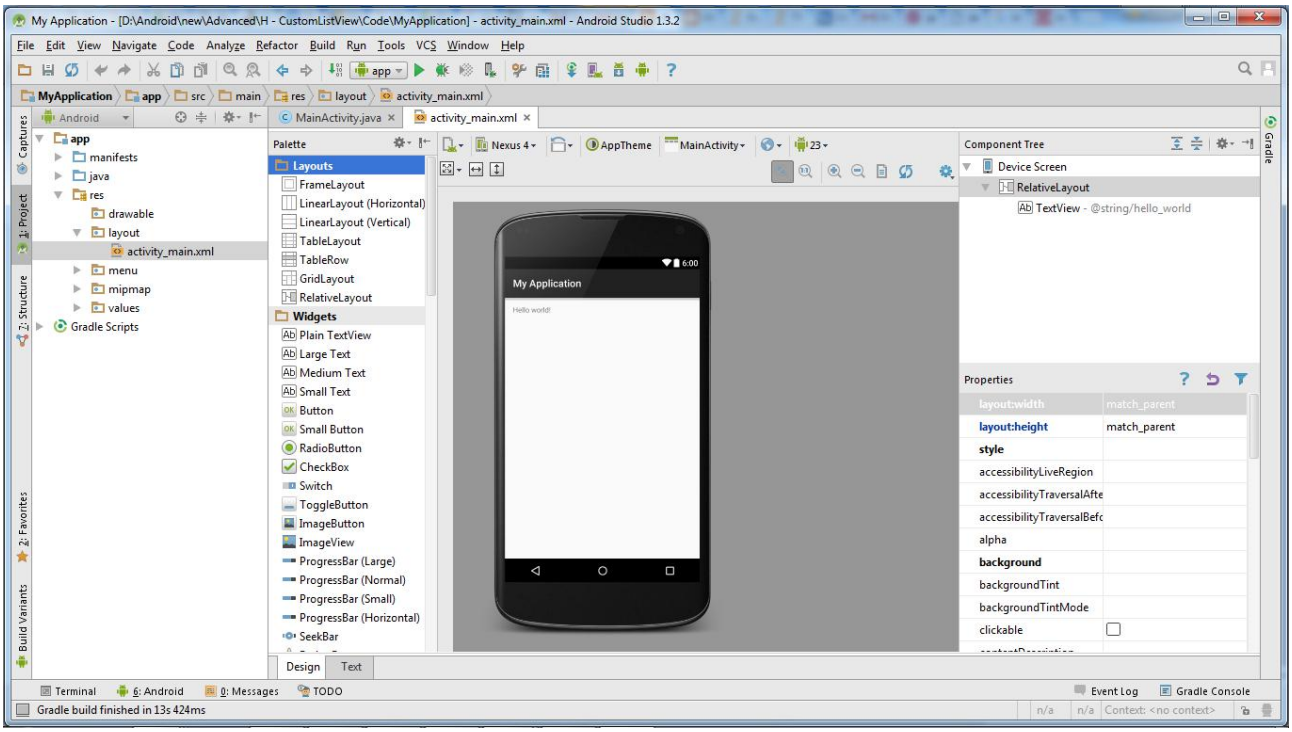


Рис. 1.3. Початковий екран середовища Android Studio

Android Studio прийшло на зміну плагіну ADT для платформи Eclipse. Середовище побудоване на базі вихідного коду продукту IntelliJ IDEA Community Edition, що розвивається компанією JetBrains. Android Studio розвивається в рамках відкритої моделі розробки та поширюється під ліцензією Apache 2.0.

Бінарні складання підготовлені для Linux (для тестування використаний Ubuntu), Mac OS X і Windows. Середовище надає засоби для розробки застосунків не тільки для смартфонів і планшетів, але і для носимих пристроїв на базі Android Wear, телевізорів (Android TV), окулярів Google Glass і автомобільних інформаційно-розважальних систем (Android Auto). Для застосунків, спочатку розроблених з використанням Eclipse і ADT Plugin, підготовлений інструмент для автоматичного імпорту існуючого проекту в Android Studio.

Середовище розробки адаптоване для виконання типових завдань, що вирішуються в процесі розробки застосунків для платформи Android. У тому числі у середовище включені засоби для спрощення тестування програм на сумісність з різними версіями платформи та інструменти для проектування застосунків, що працюють на пристроях з екранами різної роздільності (планшети, смартфони, ноутбуки, годинники, окуляри тощо). Крім можливостей, присутніх в IntelliJ IDEA, в Android Studio реалізовано кілька додаткових функцій, таких як нова уніфікована підсистема складання, тестування і розгортання застосунків, заснована на складальному інструментарії Gradle і підтримуюча використання засобів безперервної інтеграції.

Для прискорення розробки застосунків представлена колекція типових елементів інтерфейсу і візуальний редактор для їхнього компонування, що надає зручний попередній перегляд різних станів інтерфейсу застосунку (наприклад, можна подивитися як інтерфейс буде виглядати для різних версій Android і для різних розмірів екрану). Для створення нестандартних інтерфейсів присутній майстер створення власних елементів оформлення, що підтримує використання шаблонів. У середовище вбудовані функції завантаження типових прикладів коду з GitHub.

До складу також включені пристосовані під особливості платформи Android розширені інструменти рефакторингу, перевірки сумісності з минулими випусками, виявлення проблем з продуктивністю, моніторингу споживання пам'яті та оцінки зручності використання. У редактор доданий режим швидкого внесення правок. Система підсвічування, статичного аналізу та виявлення помилок розширена підтримкою Android API. Інтегрована підтримка оптимізатора коду ProGuard. Вбудовані засоби генерації цифрових підписів. Надано інтерфейс для управління перекладами на інші мови.

Відмітимо деякі особливості будуть пізніше розгорнуті для користувачів так як програмне забезпечення розвивається; наразі, передбачені такі функції:

- Живі макети (layout): живе кодування — подання (rendering) програми в реальному часі.
- Консоль розробника: підказки по оптимізації, допомога по перекладу, стеження за напрямком, агітації та акції—метрики Google аналітики.
- Резерви бета релізів та покрокові релізи.
- Базування на Gradle.
- Android-орієнтований рефакторинг та швидкі виправлення. Lint утиліти для охоплення продуктивності, юзабіліті, сумісності версій та інших проблем.
- Використання можливостей ProGuard та підписів до програм.
- Шаблони для створення поширених Android дизайнів та компонентів.
- Багатий редактор макетів (layouts) що дозволяє користувачам перетягнути і покласти (drag-and-drop) компоненти користувацького інтерфейсу, як варіант, переглянути одночасно макети (layouts) на різних конфігураціях екранів.

Середовище розробки Xcode—це пакет інструментів для розробки додатків під Mac OS X і iPhone OS, розроблений Apple. Остання версія Xcode 9.2, безкоштовно поставляється на дистрибутивному диску Mac OS X Install DVD разом з операційною системою Mac OS X 10.6, хоча і не встановлюється за умовчанням. Третя версія не підтримується старими версіями Mac OS, для яких XCode також доступний для безкоштовного звантаження через Apple Developer Connection. Оновлення можна безкоштовно скачати на офіційному сайті підтримки. На сьогодні є єдиним засобом написання «універсальних» (Universal Binary) прикладних програм для Mac OS X. Xcode тісно інтегрований з фреймворком Cocoa. Його використовують і при розробці самої Apple Mac OS X. Цей набір інструментів включає:

Xcode IDE (для кодування, створення і налагодження додатків)(рис. 1.4) [3]:

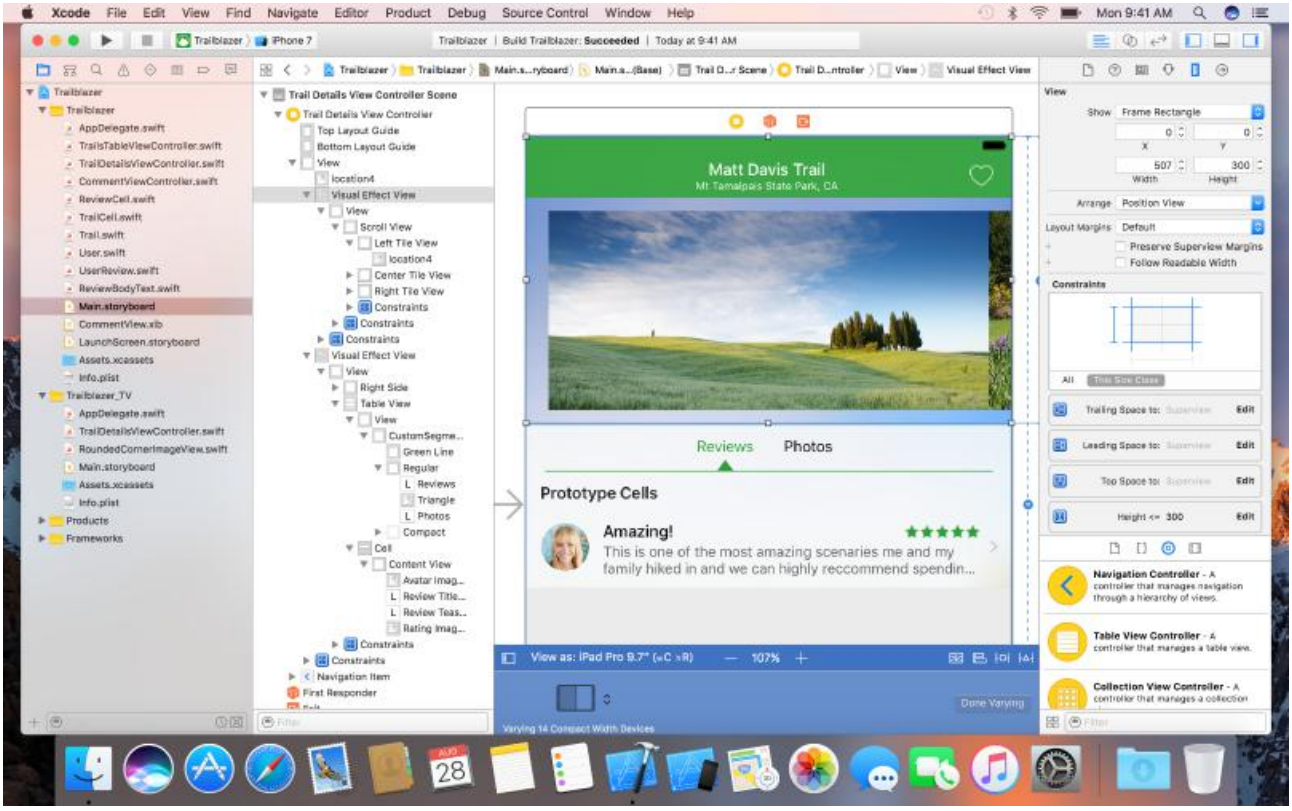


Рис. 1.4. Головний екран Xcode IDE 9.0

Xcode IDE надає все, що потрібно для розробки: від професійних редакторів з функцією автозаповнення коду і Cocoa рефакторінга, до налаштування open-source компіляторів. Xcode IDE розроблений з нуля, щоб можна було скористатися всіма можливостями Cocoa і новітніми технологіями Apple.

Xcode включає велику кількість можливостей, щоб полегшити розробки iOS-додатків, включаючи наступні:

- систему управління проектом для визначення програмних продуктів;
- контекстно-залежний інспектор для перегляду інформації про виділений в коді символі;
- середовище для редагування коду, що включає можливості підсвічування синтаксису, автозаповнення коду та індексацію символів;
- просунуту програму перегляду і пошуку документації Apple;

- просунуту систему збирання проекту з перевіркою залежностей і перевіркою правил складання;
- компілятори GCC, що підтримують мови C, C ++, Objective-C, Objective-C ++ та інші;
- підтримка LLVM і Clang для мов C, C ++ і Objective-C;
- вбудоване налагодження вихідного коду з використанням GDB;
- розподілені обчислення, що дозволяють поширювати великі проекти через кілька мережевих пристроїв;
- інтелектуальна компіляція, яка прискорює час компіляції одного файлу;
- просунуті можливості по налагодженню коду;
- просунуті засоби налагодження, що дозволяють робити глобальні зміни в коді без зміни його поведінки;
- підтримка знімків проекту, які надають легше управління вихідним кодом;
- підтримка запуску засобів продуктивності для аналізу програми;
- підтримка вбудованої системи управління вихідним кодом;
- підтримка AppleScript для автоматизації процесу складання;
- підтримка налагоджувальної інформації в форматах DWARF і Stabs (налагоджувальна інформація для всіх проектів за замовчуванням генерується в форматі DWARF).

Interface Builder (для розробки користувальницького інтерфейсу) (рис. 1.5) [3]:

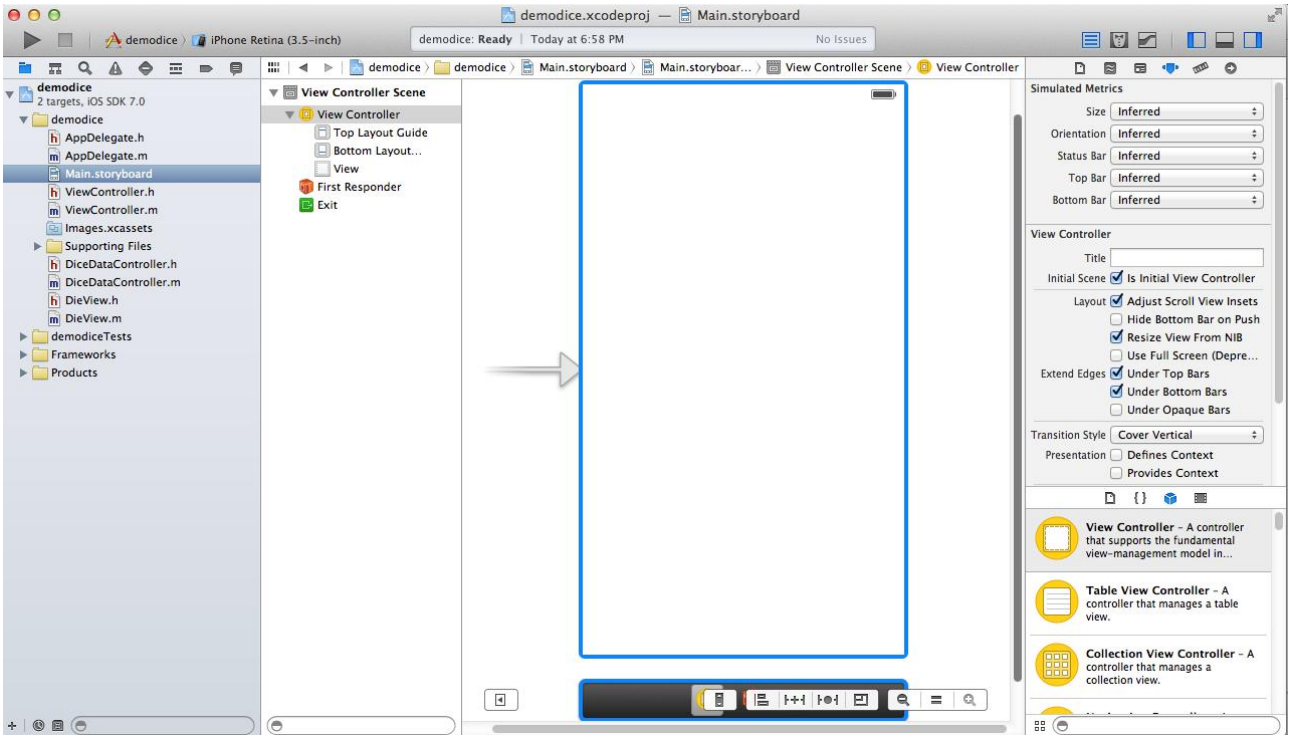


Рис. 1.5. Початковий екран розділу Interface Builder в Xcode IDE 9.0

Interface Builder спрощує створення призначеного для користувача інтерфейсу (UI). З його допомогою можна легко, без написання коду, створити шари з вікон, різні кнопки, повзунки та інші елементи управління. Потім можна перетворити цей прототип UI в реальний додаток, додавши нові можливості. Xcode працює з Interface Builder в режимі реального часу, так що можна спостерігати в графічному інтерфейсі (Interface Builder) те, що розробляється в Xcode.

Використовуючи Interface Builder, можна створювати вікна програми шляхом перетягування в нього попередньо налаштованих компонентів. Компоненти включають стандартні системні елементи управління, такі як перемикачі, текстові поля, кнопки тощо. Компоненти поміщаються вікна, після цього їх можна позиціонувати, перетягуючи по всій величині вікна, налаштовувати атрибути, використовуючи інспектор і встановлювати зв'язки між цими об'єктами і кодом. Вміст вікна зберігається в nib-файл, який є ресурсним файлом особливого формату. Він містить всю інформацію, яка потрібна бібліотеці "UIKit" для створення тих же самих об'єктів у додатку під час

виконання. Завантаження nib-файлу створює версії часу виконання всіх об'єктів, збережених у файлі, налаштовує їх в точності, якими вони були в Interface Builder. Також використовується інформація про взаємозв'язок, зазначена розробником для установки зв'язку між заново створеними об'єктами і вже існуючими в додатку. Ці зв'язки забезпечують код вказівниками на об'єкти nib-файлу, а також надають інформацію самим об'єктам, необхідну для передачі дій користувача вихідному коду. В цілому, використання Interface Builder зберігає величезну кількість часу, що припадає на створення UI. Interface Builder усуває написання коду, необхідного для створення, налаштування і позиціонування об'єктів, які використовуються для розробки інтерфейсу. Тому тут можна побачити, як інтерфейс буде виглядати під час виконання. UI фактично є архівами об'єктів Cocoa, які не вимагають генерації коду. Зміни в інтерфейсі користувача (UI) не вимагають перекомпіляції (перевірки) коду, а зміни в коді не вимагають перекомпіляції UI.

Інструменти для аналізу поведінки і продуктивності (рис. 1.6) [3]:

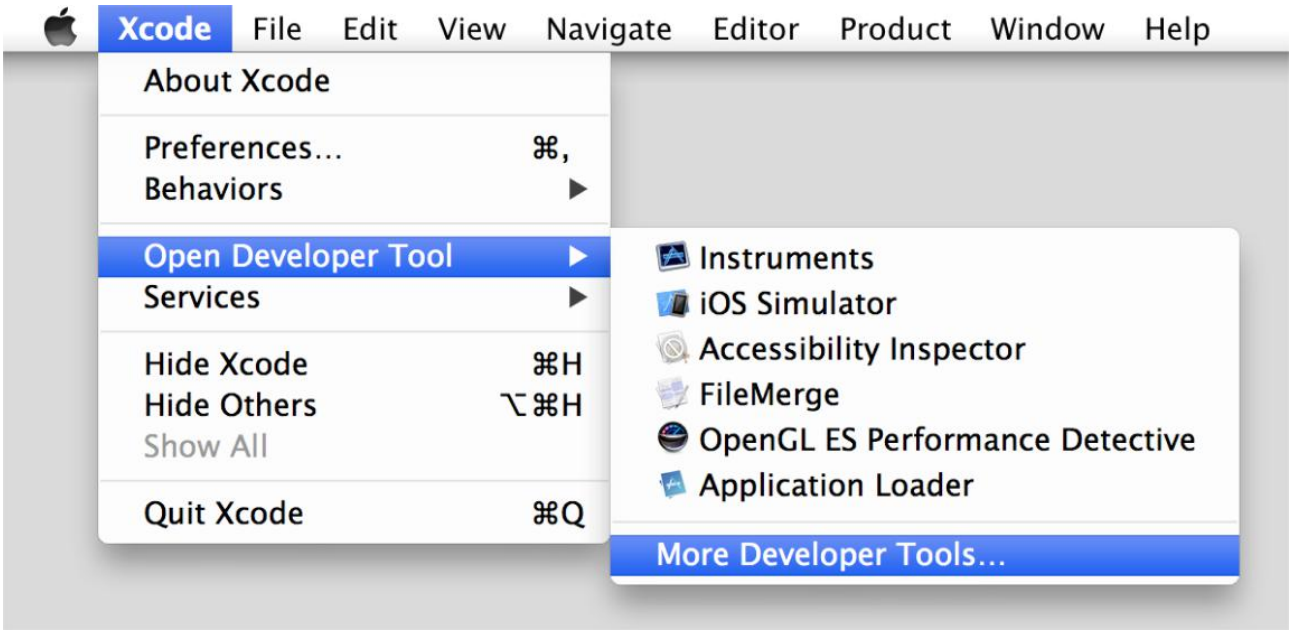


Рис. 1.6. Різноманітні Developer Tools в Xcode IDE 9.0

Величезний світ Mac і iPhone застосувань надає користувачеві великий досвід, на який слід спиратися при створенні своєї програми. Додаток має

містити в собі елегантний користувальницький інтерфейс і оптимальну продуктивність. Developer Tools включають потужні інструменти оптимізації і аналізу (Instruments and Shark), які дозволяють проаналізувати продуктивність iOS-додатків, поки вони виконуються в емуляторі або на пристрої. Developer Tools збирають дані запущеного додатку і відображають їх у графічному вигляді (тимчасова шкала). Вона дозволяє збирати дані про використання пам'яті, дискової активності, мережеву активність і графічної продуктивності мобільного застосування. Тимчасова шкала може відображати всі типи інформації відразу, дозволяючи співвідносити загальну поведінку додатка, а не тільки поведінку за певним параметром. Все це дозволяє легко визначити проблемні зони додатка, і потім перейти до проблемних рядків коду. Developer Tools надають засоби для аналізу поведінки програми з плином часу. Наприклад, вікно середовища Developer Tools дозволяє зберігати дані декількох запусків, дозволяючи тим самим бачити, чи поліпшилося поведінка застосування або воно все ще потребує доопрацювання. Також можна зберігати дані цих запусків в документах і відкривати їх в будь-який час.

1.2.4 Огляд мов програмування для розробки мобільних додатків

1.2.4.1 Опис специфікацій та особливостей мови програмування Java

Java — об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.Переваги:

- Java — це відома мова програмування, велика кількість розробників знають її і не повинні її вивчати;
- складніше застрелити себе з Java, ніж з кодом C / C ++, оскільки він не має арифметики покажчиків;

- Java працює на віртуальній машині, тому немає необхідності перекомпілювати її для кожного телефону, також такий код легко захищати;
- велика кількість інструментів для розробки Java;
- кілька мобільних телефонів вже використовували Java ME, тому Java відома в галузі;
- різниця в швидкості не є проблемою для більшості додатків;

Поліпшення стабільності системи дуже важливо на пристрої, такому як смартфон.

Безпека ще важливіше. Середа Android дозволяє користувачам запускати ненадійні додатки, які можуть використовувати телефон неприйнятним чином без чудової безпеки. Запустивши всі програми на віртуальній машині, можливо гарантувати, що ні один додаток не зможе використовувати ядро ОС, якщо в реалізації VM не існує нестачі. Реалізація VM, в свою чергу, імовірно мала і має невелику, чітко визначену поверхню безпеки.

Можливо, найголовніше, коли програми вже скомпільовані у код для віртуальної машини, їх не потрібно перекомпілювати для нового обладнання. Ринок телефонних чіпів різноманітний і швидко змінюється, так що це дуже важливо.

Крім того, використання Java робить менш ймовірним, що користувачі додатків пишуть самі. Відсутність переповнення буфера, помилки з покажчиками і т. д.

Як вже говорилося, основна проблема полягає в тому, що Android розроблений як переносна ОС для роботи на самих різних апаратних засобах. Він також грунтується на структурі і мові, знайомих багатьом існуючим розробникам мобільних пристроїв.

Нарешті, я б сказав, що це ставка проти майбутнього — незалежно від того, які проблеми з продуктивністю існують, стане недоречним, оскільки апаратне забезпечення покращиться — в рівній мірі, змусивши розробників кодувати

абстракцію, Google може вирвати і змінити базову ОС набагато легше, ніж якби Розробники кодували API POSIX / Unix.

Для більшості додатків накладні витрати на використання мови на основі VM в порівнянні з рідним невеликі (вузьким місцем для додатків, що використовують веб-сервіси, наприклад Twitter, в основному є мережеві). Palm WebOS також демонструє це і в якості основного мови використовується JavaScript, а не Java.

З огляду на, що майже всі віртуальні машини JIT зводяться до власного коду, швидкість вихідного коду часто порівнянна з власною швидкістю. Безліч затримок, пов'язаних з мовами вищого рівня, менше пов'язано з накладними витратами віртуальної машини, ніж з іншими факторами (складний час виконання об'єкта, «безпека», що перевіряє доступ до пам'яті шляхом перевірки кордонів і т. Д.).

Також пам'ятайте, що незалежно від мови, що використовується для написання програми, велика частина фактичної роботи виконується в API нижчого рівня. Мова верхнього рівня часто просто пов'язує виклики API.

Звичайно, є багато винятків з цього правила—гри, аудіо та графічні додатки, які викликають обмеження на апаратне забезпечення телефону. Навіть в iOS розробники часто опускаються на C / C ++, щоб отримати швидкість в цих областях.

Недоліки:

Java, незважаючи на різні способи оптимізації, все ж досить ресурсомісткі і повільна. Причини в наступному:

- автосборка сміття;
- компіляція "на льоту" (Just In Time compilation);
- відмова від таких небезпечних механізмів як: арифметика покажчиків, неявне перетворення типів з втратою точності, функції першого класу.

Автосборка сміття з одного боку звільняє програміста від турботи звільнення пам'яті, з іншого боку відома проблема коли автосборка сміття не спрацював і відбувалася серйозна витік пам'яті.

1.2.4.2 Опис специфікації та особливостей мови програмування Swift

Програмування — основа основ комп'ютерної техніки. Корпорація Apple давно відома своїм умінням задавати тон розвитку індустрії на роки вперед, але з огляду на поступове сходження купертіновцев з ринку професійних рішень, надкушене яблуко асоціюється в першу чергу з споживчими товарами. Однак чергове дослідження громадської думки показує, що розробники ПЗ більш ніж задоволені свіжою пропозицією компанії — мовою програмування Swift. Згідно з інформацією ресурсу Stack Overflow, майже 80 відсотків професіоналів із задоволенням працювали або планують працювати з випущеним не так давно інструментом розробки від Apple (рис. 1.7) [4]. Дослідникам вдалося опитати 26 тисяч відвідувачів з більш ніж 150 країн світу. Близько третини постійно зайнятих в сегменті написання мобільного ПЗ респондентів працюють в основному на платформі iOS, а менше половини також займаються створенням додатків для конкуруючої ОС Android. 20 % опитаних не змогли визначитися, швидше за все, сюди входять фахівці, які регулярно розробляють програмне забезпечення для різних платформ.

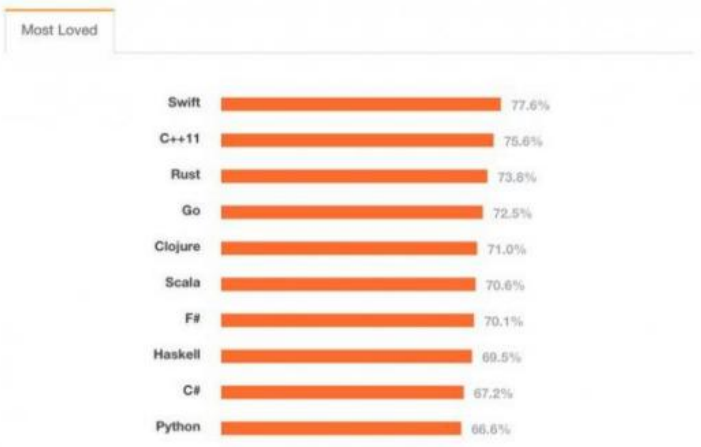


Рис. 1.7. Оцінки мов програмування

Swift ─ багатопарадигмова компільована мова програмування, розроблена компанією Apple для того, щоб співіснувати з Objective C і бути стійкішою до помилкового коду. Swift була представлена на конференції розробників WWDC 2014. Мова побудована з LLVM компілятором, включеного у Xcode 6 beta. Безкоштовний посібник мови програмування Swift доступний для завантаження у магазині iBooks.

Компілятор Swift побудований з використанням технологій вільного проекту LLVM. Swift успадковує найкращі елементи мов C і Objective-C, тому синтаксис звичний для знайомих з ними розробників, але водночас відрізняється використанням засобів автоматичного розподілу пам'яті і контролю переповнення змінних і масивів, що значно збільшує надійність і безпеку коду.

При цьому Swift-програми компілюються у машинний код, що дозволяє забезпечити високу швидкодію. За заявою Apple, код Swift виконується в 1.3 рази швидше коду на Objective-C. Замість збирача сміття Objective-C в Swift використовуються засоби підрахунку посилань на об'єкти, а також надані у LLVM оптимізації, такі як автовекторизація. Мова також пропонує безліч сучасних методів програмування, таких як замикання, узагальнене програмування, лямбда-вирази, кортежі і словникові типи, швидкі операції над колекціями, елементи функційного програмування. 26 Основним застосуванням Swift є розробка користувацьких застосунків для MacOS X і Apple iOS з використанням фреймворка Cocoa і Cocoa Touch. При цьому Swift надає об'єктну модель, сумісну з Objective-C. Вихідний код мовою Swift може змішуватися з кодом на C і Objective-C в одному проекті [5]. Swift щільно інтегрований у власницьке середовище розробки Xcode і не може бути використаний відособлено на платформах, відмінних від OS X. Окремо варто відзначити, що Swift від компанії Apple не варто плутати з досить давно розроблюваною скриптовою мовою Swift, націленої на багатонитеве програмування і поставленого під вільною ліцензією Apache.

1.2.4.3 Опис специфікації та особливостей мови програмування Objective-C

Objective-C — рефлективна, високорівнева об'єктно-орієнтована мова програмування загального призначення, розроблена у вигляді набору розширень стандартної C. Розроблена компанією Apple, використовується в основному у Mac OS X та GNUStep — середовищах, розроблених на основі стандарту OpenStep, та Cocoa — бібліотеки компонентів для розробки програм. Програму на Objective-C що не використовує цих бібліотек можна скомпілювати для будь-якої платформи, яку підтримує gcc компілятор з підтримкою Objective-C. Objective-C є розширенням C і тому будь-яку програму на C можна скомпілювати компілятором Objective-C. ООП в Objective-C включає інтерфейси, класи, категорії. Реалізовано одиничне, невіртуальне спадкування. Немає єдиного базового класу для всіх об'єктів. Всі методи в класі — віртуальні. Категорія — це парадигма, яка дозволяє описувати інтерфейс з методами, які «необов'язково» імплементувати. Синтакс Objective-C породжений одночасно від C та Smalltalk. Від останньої взято основний семантичний конструкт мови — замість виклику методу об'єктові надсилається повідомлення. Наприклад, якщо клас об'єкта obj 27 імплементує метод doJob то говориться що об'єкт відкликається на повідомлення doJob. Щоб надіслати повідомлення doJob цьому об'єктові потрібно написати: [obj doJob]; Такий механізм дозволяє надсилати повідомлення навіть до тих об'єктів які не підтримують їх обробки. Такий підхід відрізняється від тих, що використовуються в статично типізованих мовах C++ чи Java.

Мови програмування не вмирають швидко, а студії розробники, які чіпляються за згасаючі парадигми, вмирають. В даний момент Swift не тільки витісняє Objective-C, коли мова йде про розробки додатків під OS X, iOS і WatchOS, але і в недалекому майбутньому замінить C для внутрішнього програмування, для платформ Apple [6].

Завдяки кільком ключовим особливостям, Swift має потенціал стати єдиною мовою програмування, для створення захоплюючих, гнучких додатків або програм на багато років.

У Apple великі надії на Swift. Компанія настільки ефективно оптимізувала компілятор і саму мову в принципі, що багато можливостей ще тільки належить розкрити. Якщо вірити офіційній документації по Swift, то можна з впевненістю сказати, що Swift «призначений для зльоту від "hello, world" до цілої операційної системи» [7]. Виділимо основні причини:

1. Swift більш читабельна мова, ніж Objective-C:

У Objective-C є всі болячки, які можуть бути у мови, побудованої на C. Для диференціації ключових слів і типів від C типів, Objective-C вводить нові ключові слова, використовуючи символ @. Так як Swift не побудований на C, то він може об'єднати всі ключові слова і видалити численні символи @ перед кожним Objective-C типом або перед пов'язаною з об'єктом ключовим словом.

Swift переглядає загальноприйняті умови успадкування. Таким чином, більше не потрібно ставити кому в кінці рядка або писати круглі дужки, щоб оточити умовні вирази всередині операторів if / else [8]. Найбільшою перевагою Swift перед Objective-C є те, що виклики методу не розташовуються усередині один одного. Для виклику функцій і методів в Swift використовується стандартний, розділений комами, список параметрів в круглих дужках. В результаті отримуємо мову зі спрощеним синтаксисом і граматикою.

Код в Swift дуже нагадує природну англійську мову, на додаток до інших сучасних популярних мов програмування. Читабельність коду в Swift полегшує роботу для програмістів JavaScript, Java, Python, C #, C ++, на відміну від Objective-C.

2. Swift легше підтримувати:

Спадщина утримує Objective-C позаду: мова не може розвиватися без розвитку C. C вимагає від програмістів підтримки 2 кодових файлів для поліпшення часу установки і ефективності створення додатка — вимога, яка переноситься на Objective-C.

Swift скасовує вимогу двох-файлів. Xcode і компілятор LLVM може з'ясувати залежність і виконати покрокові зміни автоматично в Swift 1.2. Середовище розробки Xcode і компілятор LLVM (Low Level Virtual Machine) може з'ясувати залежність і виконати покрокові зміни автоматично в Swift. Тому, поділу змісту від тіла стає справою минулого. Swift поєднує в собі заголовок Objective-C (.h) і файли реалізації (.m) в одному файлі коду (.swift). Двох файлова система Objective-C накладає додаткову роботу на програмістів - і це робота, яка відволікає їх від більш важливих завдань. У Objective-C потрібно вручну синхронізувати імена методів і коментарі між файлами [3].

Xcode і компілятор LLVM можуть працювати непомітно для програміста, знижуючи його навантаження. Використовуючи Swift, програмісти роблять менше допоміжних дій і можуть витратити більше часу на створення логіки 29 додатка. Swift відмовляється від шаблонної праці і покращує якість коду, підтримування коментарів і функцій [9].

3. Swift ― безпечніша мова:

Одним з цікавих моментів в Objective-C можна вважати спосіб обробки вказівників, особливо nil (NULL). У Objective-C нічого не трапиться, якщо спробувати викликати метод зі змінною покажчика nil (неініціалізованих) [10]. Вираз або рядок коду стає невиконуваним і може здатися, що вираз спрацює, але насправді буде повно помилок. Нездійсненність призводить до непередбачуваної поведінки, що є ворогом програмістів, які намагаються знайти і виправити випадкові збої або зупинити поведінку, яка відхиляється від норми.

Опціональні типи дають можливість існування в Swift коді nil (опціонального значення), що говорить про можливість створення помилки компілятора при написанні поганого коду. Це створює короткий цикл зворотного зв'язку і дозволяє програмістам писати програми більш впевнено. Проблеми можуть бути усунені при вже написаному коді, що значно зменшує кількість часу і грошей, які витрачаються на виправлення помилок, в порівнянні з передачею вказівника на NSError в Objective-C [9].

Традиційно в Objective-C, якщо значення було повернуто з методу, то програміст був зобов'язаний зафіксувати поведінку вказівника повернутої змінної (з використанням коментарів і найменування методу). В Swift опціональні типи і типи значень дозволяють явно визначити метод, якщо значення існує, або якщо воно може бути опціональним (тобто, значення може існувати або може бути nil).

Для забезпечення передбачуваної поведінки, Swift викликає помилку при виконанні коду, якщо використовується опціональна змінна nil. Ця помилка забезпечує узгоджену поведінку, яке полегшує процес усунення помилок, тому що змушує програміста вирішити проблему відразу. Ця помилка виникне на 30 тому рядку коду, де була використана опціональна змінна nil. Це означає, що помилка буде виправлена своєчасно або її вдасться уникнути зовсім.

4. Незалежне управління пам'яттю:

Swift уніфікований так, як ніколи не був Objective-C. Підтримка автоматичного підрахунку посилань (ARC) є повною з процедурних і об'єктно-орієнтованих шляхах коду. У Objective-C ARC підтримується всередині Cocoa API і об'єктно-орієнтованого коду, але він не доступний для C коду і API, наприклад Core Graphics [3]. Це означає, що програміст повинен взяти на себе управління пам'яттю при роботі з Core Graphics APIs і з іншими старішими API, доступних на iOS. Величезні витоки пам'яті, які програміст може мати в Objective-C, неможливі в Swift.

Програмісту не доводиться думати про пам'ять для кожного цифрового об'єкта, який він створює. Тому що ARC обробляє всі управління пам'яттю під час компіляції, і витрати, які пішли б на управління пам'яттю, тепер можуть бути сфокусовані на основній лозіці додатка і нових можливостях. Це відбувається тому, що ARC в Swift працює і на процесуальному, і на об'єктно- орієнтованому коді, і тепер не потрібно контекстних переходів для програмістів, навіть якщо вони пишуть код, який розрахований на більш старі API. Це є проблемою для поточної версії Objective-C.

Автоматичне і високопродуктивне управління пам'яттю було проблемою, яку Apple змогли вирішити і довели, що це може підвищити продуктивність. З іншого боку і в Objective-C, і в Swift немає залежності від темпу роботи Garbage Collector, який використовується для очищення невикористаної пам'яті, як в Java, Go або C #. Це важливий фактор для будь-якої мови програмування, що використовується для чутливої графіки і наданні входу для користувача, особливо на таких пристроях як iPhone, Apple Watch або IPad (де затримка в роботі сприймається як розчарування і змушує користувача думати, що програма не працює).

5. Swift вимагає меншу кількість коду:

Swift зменшує кількість коду, необхідного для повторюваних заяв і рядків (рис. 1.8) [9]. У Objective-C, працюючи з текстовими рядками, для того, щоб об'єднати дві частини інформації, потрібно зробити безліч кроків. Для додавання двох рядків Swift використовує оператор «+». Ця особливість відсутня у Objective-C. Така підтримка для об'єднаних символів і рядків має важливе значення для будь-якої мови програмування, яка використовує відображення тексту для користувача на екрані.

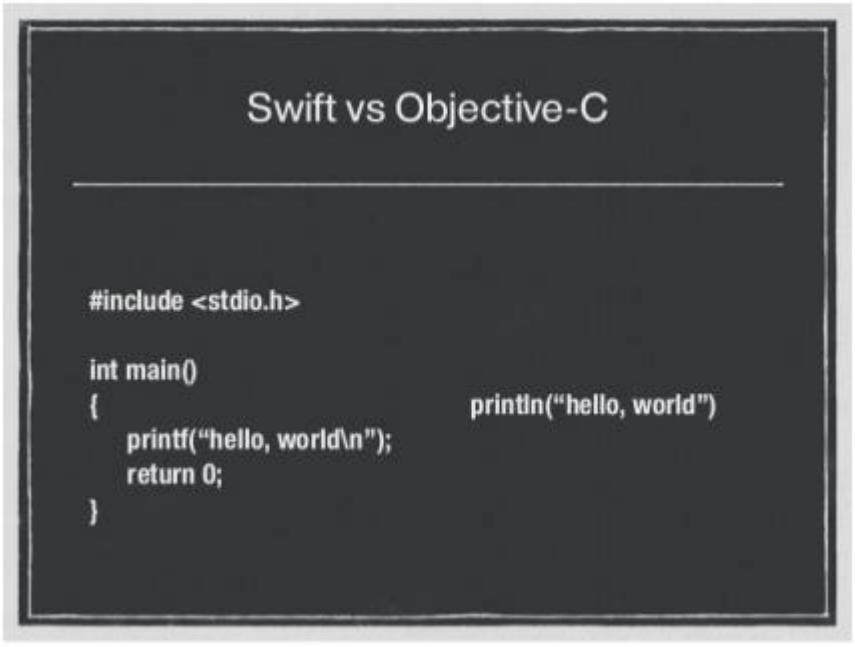


Рис. 1.8. Порівняння Swift та Objective-C

Система типів в Swift знижує складність в написанні коду, так як компілятор може з'ясовувати типи. Наприклад: Objective-C вимагає, щоб програмісти запам'ятовували спеціальні маркери рядків (%S, %d, %@) і використовували розділений комами список. Swift підтримує інтерполяцію рядків, що усуває необхідність запам'ятовування символів і дозволяє програмістам вставляти змінні, такі як назва ярлика або кнопки, безпосередньо у вбудований рядок користувача. Тип виведення системи і рядки інтерполяції зменшують кількість помилок, які поширені в Objective-C. У Objective-C, якщо порушити порядок розташування або використовувати неправильний символ, мобільний додаток працювати не буде. Swift ж знову зменшує кількість написання коду вбудованій підтримці для обробки текстових рядків і даних.

6. Swift швидше:

Видалення застарілих конвенцій набагато поліпшило «двигун» Swift. Тестування продуктивності коду в Swift як і раніше вказує на те, що Apple продовжує покращувати швидкість роботи додатків на Swift (рис. 1.9) [7].

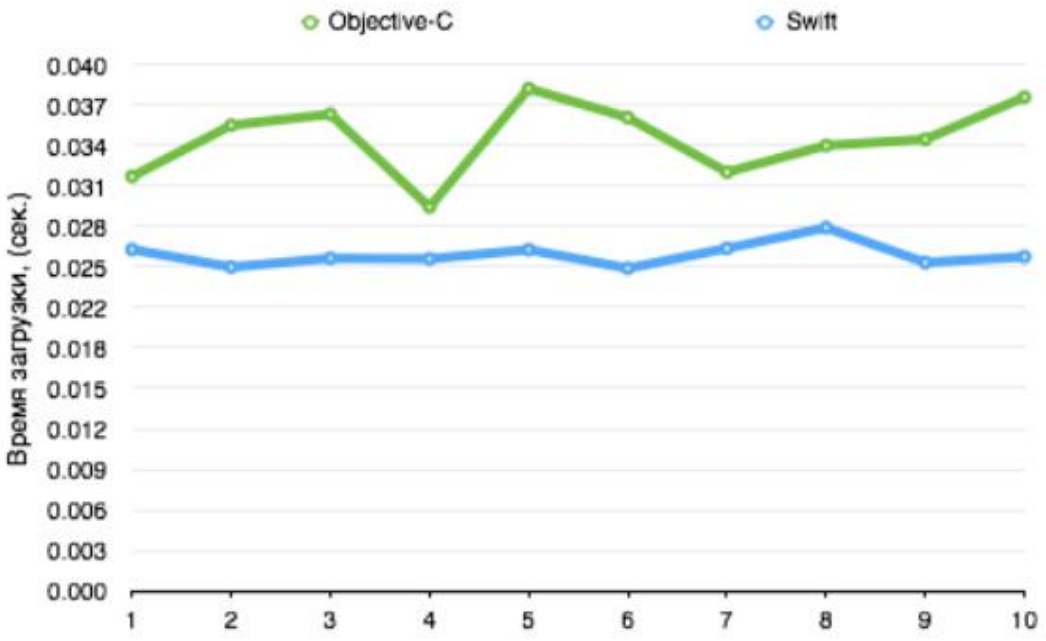


Рис. 1.9. Швидкість роботи додатків

Відповідно до даних Primate Labs, за показниками популярного тесту GeekBench (крос-платформний еталонний тест для вимірювання швидкодії процесора і підсистеми пам'яті комп'ютера) Swift наблизився до експлуатаційних характеристик C++ по обмеженню швидкості обчислень з використанням алгоритму Мандельброта (знаходження множини Мандельброта) в грудні 2014 року [11]. У лютому 2015 року Primate Labs виявили, що Xcode 6.3 Beta поліпшив продуктивність алгоритму GEMM в Swift (алгоритм обмеженої пам’яті з послідовним доступом великих масивів (a memory-bound algorithm with sequential access of large arrays)) до коефіцієнта 1,4. Початкова імплементація FFT (алгоритму обмеженої пам'яті з випадковим доступом великих масивів) ― поліпшення продуктивності в 2,6 рази. Swift показав і інші поліпшені показники при подальшому тестуванні: 8,5-кратне підвищенням для FFT алгоритму (залишивши C++ тільки з приростом продуктивності в 1,1 рази). Також, Swift перевершує C++ для алгоритму Мандельброта з коефіцієнтом в 1,03 [12].

Практично, Swift на одному рівні з C++ для FFT і алгоритму Мандельброта. Відповідно до даних Primate Labs алгоритм GEMM показав, що компілятор Swift не може векторизувати код так, як може компілятор C++. У C++ є невелика перевага, але Apple обіцяють все виправити в наступних версіях Swift.

1. Менше зіткнень імен з проектами відкритого вихідного коду:

Існує одна проблема, яка переслідує код Objective-C ― це відсутність формальної підтримки простору імен (namespaces), які були створені як вирішення проблеми C++ для зіткнень при назві файлів. Коли це зіткнення відбувається в Objective-C, це вважається помилкою лінкера (редактора зв'язків), і додаток не може працювати. Обхідні шляхи існують, але вони мають потенційні підводні камені. Існує загальна згода на використання дво- або трибуквених префіксів, для відмінності написаного коду Objective-C кимось, і від такого ж коду для Facebook.

Swift дає можливість використання неявних просторів імен, що дозволяє одному і тому ж файлу з кодом використовуватися в численних проектах, без ризиків збою і вимог найменувань, таких як NSString (NextStep ― компанія

Стіва Джобса, після його звільнення з Apple) або CGPoint (Core Graphics). В кінцевому рахунку, ця функція в Swift сприяє більшій продуктивності для програмістів і означає, що їм не доведеться займатися додатковою роботою, яка існує в Objective-C. Також можна замітити, що в Swift є зв'язок з простими іменами, такими як Array, Dictionary і String, замість NSArray, NSDictionary, і NSString, які народилися через відсутність простору імен в Objective-C.

В Swift простори імен засновані на цілях коду. Це означає, що програмісти можуть диференціювати класи або значення, використовуючи ідентифікатор простору імен. Це дуже вагома зміна в Swift. Вона значно полегшує включення проектів з відкритим вихідним кодом, а також додавання рамок і бібліотек в вихідний код. Простори назв дозволяють різним програмним компаніям створювати ті ж імена файлів, не турбуючись про 34 зіткненнях при інтеграції проектів з відкритим вихідним кодом. Тепер і Facebook, і Apple можуть використовувати файл об'єктного коду FlyingCar.swift без будь-яких помилок або ризиків збою.

2. Swift підтримує динамічні бібліотеки:

Найбільше зміна в Swift, яка не отримало достатньої уваги — це перехід від статичних бібліотек, які оновлюються тільки при великих оновлення (iOS8, iOS7 тощо), до динамічних бібліотек. Динамічні бібліотеки — це виконувані шматки коду, які можуть бути приєднані до додатка. Ця функція дозволяє додаткам поточної версії Swift зв'язатися з новішими версіями мови в зв'язку з його постійним розвитком.

Розробник надає додаток разом з бібліотеками, обидві з яких підписані електронним підписом і мають сертифікат розвитку для забезпечення цілісності (hello, NSA). Це означає, що Swift може розвиватися швидше ніж iOS, який в свою чергу висуває вимоги для сучасної мови програмування. Зміни в бібліотеках включаються в останнє оновлення, в додаток на App Store, і все чудово працює.

Динамічні бібліотеки ніколи не підтримувалися iOS до запуску Swift і iOS8, хоча динамічні бібліотеки підтримувалися на Mac протягом дуже довгого часу.

Динамічні бібліотеки є зовнішніми для виконання додатками, але включені в пакет програми при завантаженні з App Store. Це зменшує початковий розмір програми, так як воно завантажується в пам'ять, а зовнішній код додається тільки коли використовується.

Можливість відкласти завантаження в мобільному додатку або у вбудованому додатку на Apple Watch поліпшить ефективність відтворення для користувача. Це одна з відмінностей, яке робить екосистему iOS більш гнучкою. Apple була зосереджена на завантаженні тільки аресурсів і тепер компілювання і зв’язування коду відбувається на льоту. Завантаження в ході роботи зменшує кількість часу очікування, поки ресурс не буде відображений на екрані. 35

Динамічні бібліотеки Swift дозволяють проводити зміни і поліпшення простіше, ніж коли-небудь раніше. Програмістам більше не потрібно чекати iOS релізів, щоб отримати поліпшення Apple для Swift.

3. Спонукання до інтерактивного програмування:

Одним із важливих нововведень Swift, в порівнянні з Objective-C, є можливість писати код і переглядати результати в реальному часі. Ми можемо внести деякі зміни в програмі і відразу побачити результат, так що нам не потрібно перекомпільовувати і перезапускати програму. Це має назву Interactive Playgrounds. Також тут можна використовувати Quick Look, який відображає графіку або список результатів, Timeline Assistant, який допомагає експериментувати з кодом UI (інтерфейсом користувача) або продивлятися повний цикл створення анімацій. Даний код можна перенести в основний проект (рис. 1.10) [1].

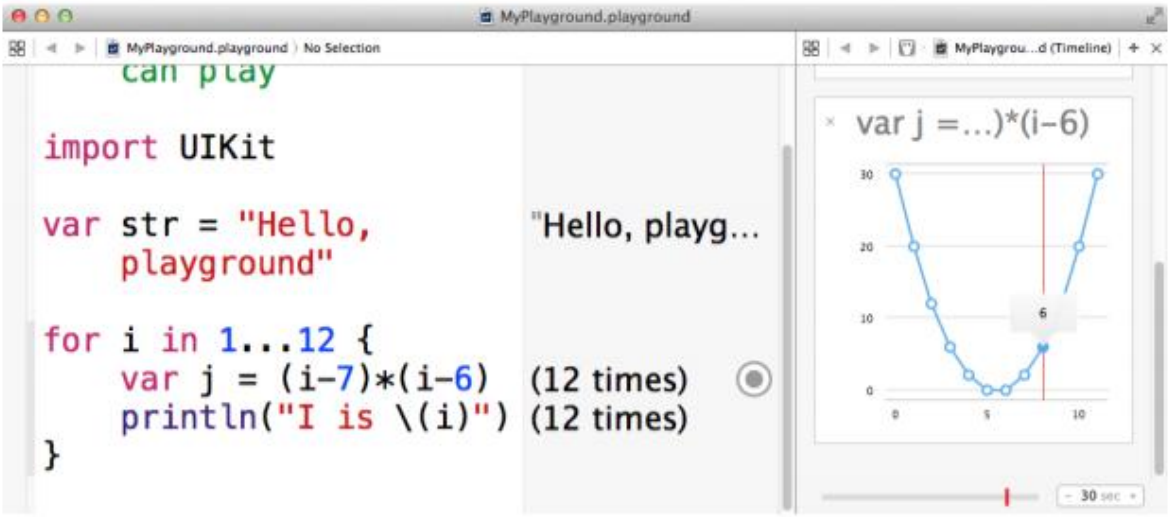


Рис. 1.10. Interactive Playgrounds

Apple додала вбудоване виконання коду під час його написання, щоб допомогти програмістам створювати шматки коду або писати алгоритми і зразу ж отримувати результат, що може поліпшити швидкість написання коду, тому що та модель, яка традиційно потрібна програмісту, може бути замінена візуалізацією даних в пісочниці. Програмування — повторюваний процес, і кожен strain може бути зменшений або використаний на додаток до творчого процесу. Це зробить роботу програмістів більш продуктивною і звільнить їх від зайвої роботи, яку додавали програмістам традиційні компілятори.

Можна відмітити, що Interactive Playgrounds не так значимі для новачків, як для досвідчених програмістів. Наприклад, показуючи роботу змінної в Interactive Playgrounds, програмісту-новачку важко зрозуміти необхідність використання змінної Float замість змінної Int. Необхідність стає очевидною, коли цю змінну можна показати в додатку, що скролл запам'ятовує останнє положення в новинах Facebook. Також, починаючи з Xcode 7 можна створювати коментарі до коду, використовуючи форматований текст з жирним шрифтом або курсивом, маркований список, вбудовані зображення і посилання. Таким чином, Interactive Playgrounds розроблені для того, щоб зробити програмування більш інтерактивним і доступним.

4. Swift — це майбутнє, на яке може вплинути будь-хто:

Objective-C нікуди не зникне, але і не буде зазнавати великих змін, завдяки появі Swift. Швидше за все, деякі зміни перекочують в Objective-C, але спадщина Objective-C в C означає, що воно лише буде все поглинати. Swift дає можливість суспільству вплинути на мову, яка буде використовуватися для створення програмного забезпечення, вбудованих систем і інших девайсів, таких як Apple Watch.

1.3 Основні принципи оптимізації мобільних додатків для операційних систем Android та iOS

Кожного разу, коли розробники мобільних додатків для обох платформ Android та iOS стикаються з проблемами продуктивності або шукають вирішення цих проблем, вони дотримуються таких правил:

- Завжди роби заміри — оптимізація навмання це завжди погана ідея. Після того як подивитися однакові анімації кілька разів, почне здаватися, що вони йдуть швидше. Числа не брешуть. Використовуйте спеціальні інструменти для замірів показників, і виміряйте кілька разів, як працює додаток до і після змін.
- Використовуй повільні пристрої—якщо ви дійсно є бажання знайти всі слабкі місця, повільні пристрої більше допоможуть в цьому. З більш новими і потужними пристроями можна не так хвилюватися через питання продуктивності, але не всі користувачі користуються останніми і найкращими моделями.
- Компроміси — оптимізація продуктивності побудована на компромісах. Оптімізія одного елементу, але ціною іншого. У багатьох випадках цим іншим може бути час, витрачений на пошуки і виправлення, або якість растрових зображень, або обсяг даних, які повинні бути збереженні в певних структурах. Потрібно бути готовим до таких жертв.

Висновок

Тема розробки мобільних застосувань для платформи Android та iOS є досить цікавою і представляє собою широке поле для дослідження в галузі розробки мобільного програмного забезпечення та методів його оптимізації. Актуальність теми підкреслюється широким спектром можливостей для втілення ідей у вигляді мобільного додатку. В даному розділі було проаналізовані основні інструменти для розробки мобільного програмного забезпечення під платформи Android та iOS. Також було доведено і обгрунтовано правильність вибору засобів для проектування. На основі проведеного аналізу встановлено, що найкращим середовищем розробки є Android Studio та Xcode, а мовами програмування — Java та Swift.

РОЗДІЛ 2.
ВИКОРИСТАННЯ ІСНУЮЧИХ МЕТОДІВ ОПТИМІЗАЦІЇ ПРИ ПРОЕКТУВАННІ МОБІЛЬНОГО ДОДАТКУ

2. 1. Огляд існуючого інструментарію, що використовується для реалізації оптимізації мобільних додатків для операційних систем Android та iOS

При виконанні оптимізації розробникам будь-якого рівня потрібні спеціальні інструменти, за допомогою яких розробник буде мати можливість записати отримані результати, переглянути попередні результати і т.д.
Потрібно описати декілька з цих інструменів, що будуть використанні при проектуванні мобільного додатку.

2.1.1 Calabash

Calabash — фреймворк, розроблений Xamarin, для автоматичного приймального тестування, Він дозволяє автоматизувати, написані на Cucumber, приймальні тестування призначеного для користувача інтерфейсу, для роботи на iOS і Android-додатках [13].

Основні переваги Calabash:

- Користувач може проводити виконання тест кейсів Calabash на більш ніж 1000 реальних мобільних пристроях в Xamarin Test Cloud, спеціальної хмарної лабораторії для тестування.
- Ясна і проста документація, що стосується роботи програми.
- Підтримка Cucumber для проведення розробки тестування, заснованої на функціонуванні (Behavior-Driven Development).
- Паралельне виконання тест кейсів.
- Широко підтримуються такі мобільні функції як жести, прокрутка і т.п.
- Прокрутка може бути автоматизована.

Недоліки Calabash:

- Тест кейси написані тільки на мові Ruby.

- Необхідна спеціальна цільова підготовка для тестування iOS додатки.
- Відсутні функції запису і відтворення.
- Відсутні інструменти генерації коду.

2.1.2 eggPlant

eggPlant включає в себе безліч інструментів для задоволення всіх потреб тестування — від функціонального до навантажувального, від мобільного до десктопного, від тестування цифрових версій до тестування застарілих версій. Інструменти eggPlant можуть функціонувати як самостійно, так і використовуючи інструменти тестування інших виробників, діючи разом в єдиному середовищі [15].

Основні переваги Calabash:

- Проста інтеграція з програмним забезпеченням для управління якістю.
- Керований режим запису.
- eggPlant Test Cases можуть запускатися без нагляду і безпосередньо з командного рядка.
- Підтримка всіх основних платформ: Android, iOS, BlackBerry, Windows Phone і Symbian.
- Взаємодіє з системою точно так же, як і користувач.
- Не потрібні ніякі плагіни.
- Немає необхідності чекати поновлення інструменту для підтримки, недавно випущеної ОС, або мови розробки.

Недоліки Calabash:

- Відсутність ідентифікатора нативного об'єкта.
- Зображення, зафіксовані в одній операційній системі, не можуть працювати в інших ОС.

- eggPlant не надто популярний на ринку. Тому дуже складно знайти фахівців, що мають навички програмування автоматизованих тест кейсів;
- Eggplant в порівнянні з іншими конкурентними інструментами, такими як Selenium Webdriver.

2. 2. Визначення головних архітектурних особливостей для розробки мобільних додатків під керуванням операційних систем Android та iOS

2.2.1 Архітектура iOS

Архітектура iOS схожа з базовою архітектурою операційної системи Mac OS X. На найвищому рівні iOS являє собою проміжний шар між обладнанням пристроя та додатками, які відображаються на екрані (рис. 2.1). Дуже рідко доводиться створювати додатки, які будуть звертатися до обладнання безпосередньо. Замість цього, додатки взаємодіють з обладнанням через набір чітко визначених системних інтерфейсів, які захищають додатки від змін обладнання. Ця абстракція дозволяє дуже легко писати програми, які коректно працюють на пристроях з різними апаратними можливостями.

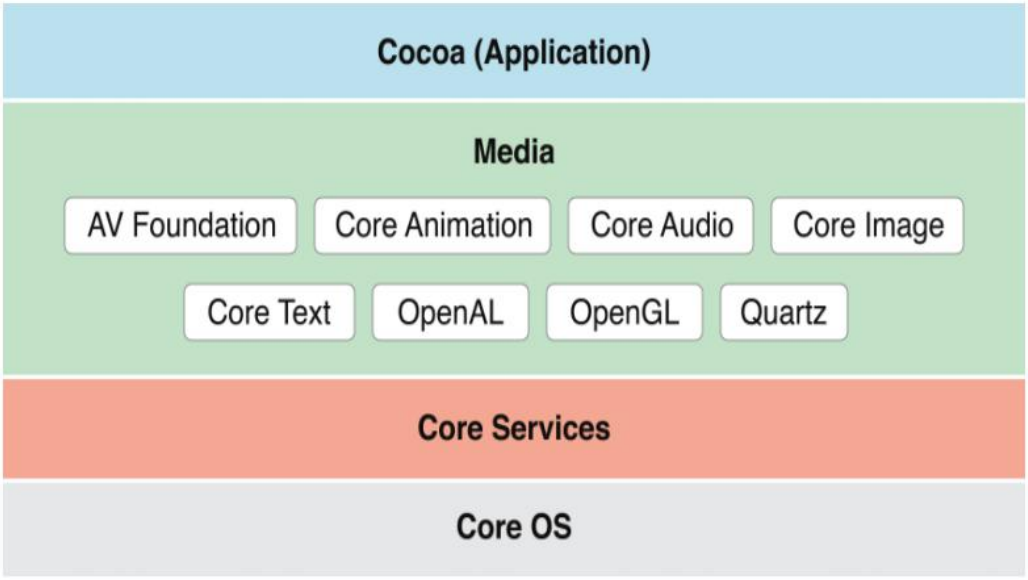


Рис. 2.1. Архітектура платформи iOS

Хоча додатки в цілому захищені від змін обладнання, але доводиться враховувати відмінності між пристроями при написанні коду. Наприклад, у деяких пристроїв є камера, а у деяких її немає. Якщо додаток вміє працювати при наявності або відсутності якоїсь функції, то використовуючи інтерфейс відповідної бібліотеки можна визначити, доступна ця функція чи ні. Додатки, яким потрібна наявність певного обладнання, повинні декларувати цю вимогу в файлі зі списком властивостей додатки (Info.plist). Реалізація технологій iOS може бути представлена у вигляді набору шарів (див. рис. 2.1). На самому нижньому шарі операційної системи знаходяться основні служби та технології, від яких залежать всі додатки; на більш високих рівнях знаходяться складніші служби і технології.

При написанні додатків всюди, де це можливо, рекомендується використовувати бібліотеки вищого рівня, ніж бібліотеки низького рівня. Бібліотеки вищого рівня написані для того, щоб забезпечити об'єктно-орієнтовані абстракції для низькорівневих структур. Ці абстракції істотно полегшують написання коду, тому що вони зменшують обсяг коду, який необхідно написати і приховують досить складні функції, такі як сокети і потоки. І хоча вони приховують низькорівневі функції, ці функції як і раніше доступні для розробників. Розробники, які вважають за краще використовувати низькорівневі бібліотеки або хочуть скористатися наявними можливостями, яких не надають високорівневі бібліотеки, можуть їх використовувати.

Основні переваги iOS:

- простота роботи;
- висока стабільність;
- безпечність;
- відсутність вірусів;
- відсутність фрагментованості;
- регулярне оновлення;
- зручність розробки додатків;
- широкий вибір програм та ігор.

Основні недоліки:

• прив’язка до апаратного забезпечення одного виробника;

• загальна «закритість» системи.

Управління пам’яттю ARC (automatic reference counting) використовується для відстеження та управління пам'яттю мобільного застосування. ARC автоматично звільняє пам'ять, яка використовувалася екземпляром класу, коли ці екземпляри більше не потрібні.

Кожен раз, коли створюється екземпляр класу, ARC виділяє шматок пам'яті для зберігання інформації цього екземпляра. Цей шматок пам'яті містить інформацію про тип екземпляра, про його значення і про будь-які властивості, пов'язаних з ним.

Додатково, коли екземпляр більше не потрібен, ARC звільняє пам'ять, використану під цей екземпляр, і направляє цю пам'ять туди, де вона потрібна. Це свого роду гарантія того, що непотрібні екземпляри не будуть займати пам'ять.

Однак, якщо ARC звільнить пам'ять використовуваного екземпляра, то доступ до властивостей або методів цього екземпляра буде неможливий. Якщо спробувати отримати доступ до цього екземпляра, то додаток швидше за все видасть помилку і зупинить свою роботу.

Для того, щоб потрібний екземпляр не пропав, ARC веде облік кількості властивостей, констант, змінних, які посилаються на кожен екземпляр класу. ARC не звільнить екземпляр, якщо є хоча б одне активне посилання.

Для того щоб це було можливо, кожен раз як привласнюється екземпляр властивості, константі або змінній створюється strong reference (сильний зв'язок) з цим екземпляром. Такий зв'язок називається "сильним", так як він міцно тримається за цей екземпляр і не дозволяє йому звільниться до тих пір, поки залишаються сильні зв'язки.

Наприклад, нехай є клас Person, який визначає константну властивість name:

```
class Person {
let name: String
```

```
init(name: String) {
self.name = name print("\(name) is being initialized")
}
deinit {
print("\(name) is being deinitialized")
}
```

Клас Person має ініціалізатор, який встановлює властивість name екземпляра і виводить повідомлення для відображення того, що йде ініціалізація. Так само клас Person має деініціалізатор, який виводить повідомлення, коли екземпляр класу звільняється.

Наступний шматок коду визначає три змінні класу Person?, який використовується для установки декількох посилань до нового екземпляру Person в наступних шматках коду. Так як ці змінні опціонального типу Person?, а не Person, вони автоматично ініціалізуються зі значенням nil, і не мають жодних посилань на екземпляр Person:

```
var reference1: Person?
var reference2: Person?
var reference3: Person?
```

Тепер можна створити екземпляр класу Person і привласнити його однією з цих трьох змінних:

```
reference1 = Person(name: "John Appleseed")
// Prints "John Appleseed is being initialized"
```

Повідомлення "John Appleseed is being initialized" виводиться під час того, як викликається ініціалізатор класу Person. Це підтверджує той факт, що відбулася ініціалізація.

Так як новий екземпляр класу Person було присвоєно змінній reference1, значить тепер існує сильне посилання між reference1 і новим екземпляром класу Person. Тепер у цього екземпляра є як мінімум одне сильна посилання, значить ARC тримає під Person пам'ять і не звільняє її.

Якщо надати іншим змінним той же екземпляр Person, то додасться два сильних посилання до цього екземпляра:

reference2 = reference1

reference3 = reference1

Тепер екземпляр класу Person має три сильні посилання. Якщо знищите два з цих трьох посилань (включаючи і первісне посилання), присвоємо nil двом змінним, то залишиться одне сильне посилання, і екземпляр Person не буде звільнений:

reference1 = nil

reference2 = nil

ARC не звільнить екземпляр класу Person до тих пір, поки залишається останнє сильне посилання, знищивши яку вказуємо на те, що екземпляр більше не використовується:

reference3 = nil

// Prints "John Appleseed is being deinitialized"

Цикл життя UIViewController

Більшість додатків під iOS пишуться з використанням фреймворку UIKit, а значить використовують UIViewController. Відповідно патерну проектування MVC (рис. 2.2) UIViewController являється контроллером, який зв’язує модель даних з візуальним представленням. Тобто, саме View — це просто аркуш паперу, на який можна добавити такі елементи як кнопки, toolbars, text views тощо. Коли натискається кнопка, то повідомлення про це отримує саме UIViewController, і саме він контролює всю діяльність, яка там відбувається. Кожен UIViewController контролює одне View. Також на UIViewController «покладено» обов'язок контролю не тільки над своїм життєвим циклом, але і над циклом життя власного UIView.

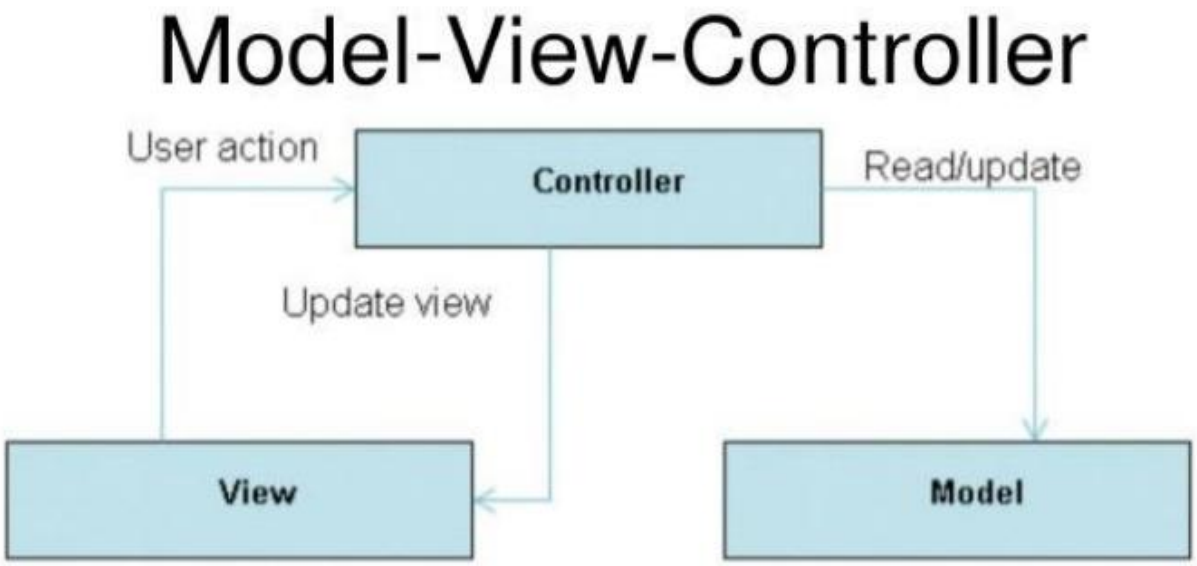


Рис. 2.2. Паттерн проектування MVC

Життєвий цикл являє собою наступне (рис. 2.3):

- Створення UIViewController:
  - init
  - initWithNibName
- Знищення UIViewController:
  - dealloc 45
- Створення UIView:
  - isViewLoaded
  - loadView
  - viewDidLoad
  - initWith
- Обробка зміни стану UIView:
  - viewDidLoad
  - viewWillAppear
  - viewDidAppear
  - viewWillDisappear
  - viewDidDisappear
  - viewDidUnload

- Знищення UIView:
  - viewDidUnload
- Обробка браку пам'яті
  - didReceiveMemoryWarning

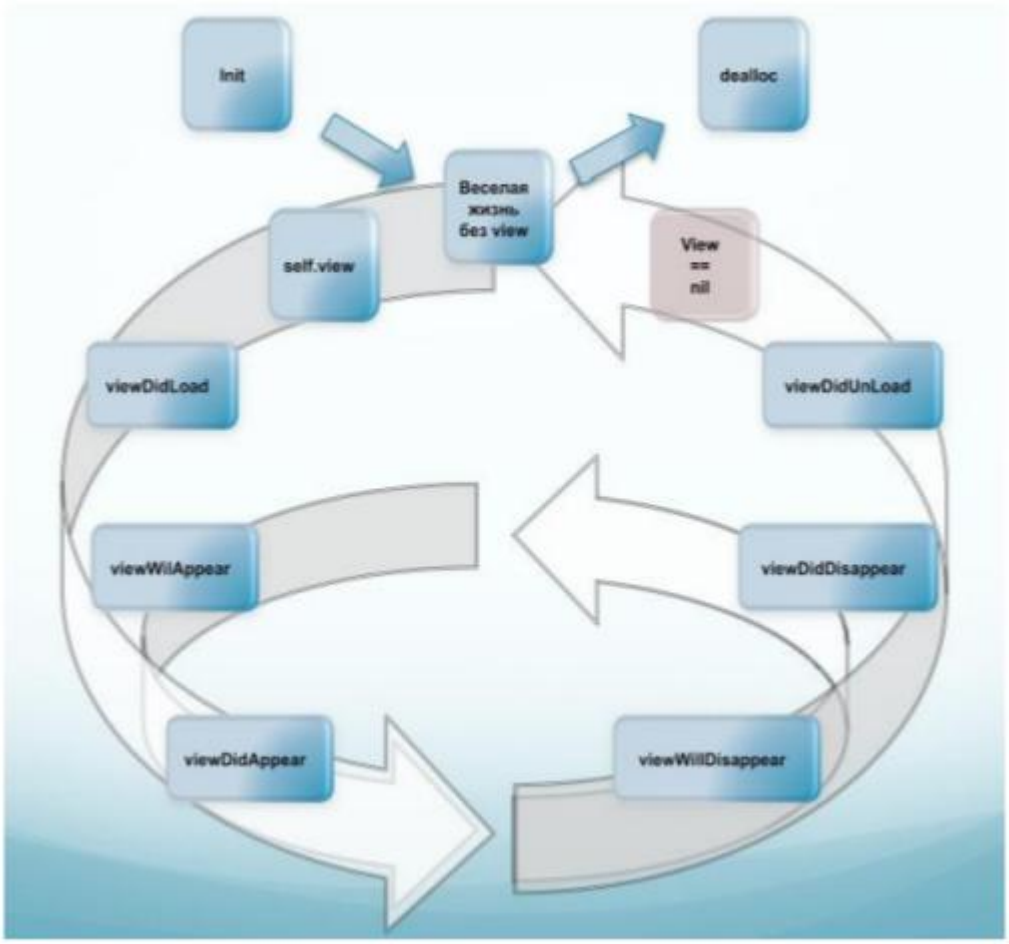


Рис. 2.3. Цикл життя UIViewController

Початок «життя» — незалежно від того, як саме інціалізується контролер (UIViewController) (з підключення .nib файлу або нормально), відбувається виклик методу init, в якому і з'являється значення змінної self, вірніше, її значення перестає бути nil і замінюється вказівником на конкретний блок пам'яті. Ініціалізується тільки сам контролер. Ніякого UIView не існує (хоча він і ініціалізований, тобто звернення до self.view можливо). Якщо використовуються .nib файли, то ніяких зв’язків на цьому етапі ще немає.

Відповідно, спроба створення або звернення до візуальних компонентів призведе до помилки. У будь-який момент можна перевірити, завантажений UIView чи ні (відповідно, можна звертатися до візуальних елементів) за допомогою методу isViewLoaded. Головна його особливість у тому, що він не «провокує» завантаження UIView, якщо ще не завантажений.

Створення UIView — у загальному випадку, створення образу (його завантаження в пам'ять) починається в той момент, коли відбудеться будь-яке звернення до self.view. Буде викликаний метод loadView, результатом якого буде або створений порожній UIView або з елементами з .nib файлу. Саме в цей момент котроллер і його UIView стають повністю доступними. По закінченню завантаження буде викликаний метод viewDidLoad. Якщо інтерфейс створюється з використанням коду, то власні елементи інтерфейсу можна створювати , додавати і налаштовувати як в методі loadView, так і в методі viewDidLoad. Якщо використовуються .nib файли, то для настройки елементів інтерфейсу доступний тільки метод viewDidLoad.

Життя UIView складається з таких етапів:

- viewDidLoad —UIView завантажений і повністю готовий до роботи. Тут відбувається так звана «друга частина ініціалізації контролера».
- viewWillAppear — буде викликаний перед тим, як почати відображення UIView. У разі, якщо для відображення використовується анімація — перед початком анімації.
- viewDidAppear — буде викликаний після закінчення відображення UIView. У разі, якщо для відображення використовується анімація — після закінчення анімації відображення.
- viewWillDisappear — теж, що і viewWillAppear, тільки коли UIView припиняє відображення.
- viewDidDisappear — теж, що і viewDidAppear, тільки коли UIView припиняє відображення.

- viewDidUnload — UIView вивантажено з пам'яті але не елементи інтерфейсу, які створювалися. Саме в цьому методі їх потрібно знищити.

Кінець життєвого шляху контролера настає, коли контролер буде знищений (вказівник на нього буде nil), ARC очистить виділену для нього пам'ять. Перед цим буде викликаний метод dealloc. Саме в ньому нобхідно виконати очистку і обнулення всіх вказівників, які створювалися для використання всередині контролера.

Обробка браку пам'яті за допомогою методу didReceiveMemoryWarning, що викликається системою при нестачі пам'яті. За замовчуванням, реалізація цього методу для контролера, який не перебуває у видимій області, викличе звільнення і видалення UIView, що, в свою чергу, призведе до виклику viewDidUnload.

Storyboard — це середовище для розробки користувацького інтерфейсу мобільного застосування (рис. 2.4).

Рис. 2.4. Ієрархія зв’язків UIView

Головна перевага Storyboard — не маючи уявлення, як працює дане мобільне застосування, розробник бачить, як виглядають його сторінки і як вони пов’язані між собою. Якщо створюється додаток з великою кількістю різних

сторінок, то Storyboard дозволяють істотно зменшити кількість коду для їх з’єднання, щоб забезпечити перехід від однієї сторінки до іншої. Раніше розробникам доводилося створювати окремі файли (файли nib або xib) інтерфейсу для кожного окремого view controller (сторінки). Але тепер додатки використовують єдиний Storyboard, який включає в себе дизайн всіх сторінок і який визначає взаємодію між ними.

Storyboards мають ряд переваг:

- У Storyboard дуже зручно спостерігати загальну картину застосування і взаємозв'язків між його сторінками.
- У Storyboard можна стежити за будь-чим, тому що загальний дизайн програми міститься в одному єдиному файлі, а не розподілений між декількома файлами nib.
- Storyboards можуть описувати переходи між різними вікнами. Ці переходи називаються “segues”, які створюються шляхом з'єднання двох сторінок прямо в Storyboard. Завдяки цим segues використовується менше коду для користувацького інтерфейсу.
- Storyboards полегшують вашу роботу з table view, з cell. Можна створювати таблиці практично повністю в Storyboards, але ж знову таки це саме те, що зменшує в рази код, який би довелося писати.
- Storyboards спрощують роботу при використання AutoLayout. Storyboards — потужна функція, яка дозволяє визначати математичні взаємозв’язки між елементами, які мають певні розміри і позиції, і так само спрощує роботу по відображенню мобільного застосування на різних пристроях з різними розширеннями екрану.

AutoLayout (автомакет, автокомпановка) — це механізм розташування елементів користувача на екрані. AutoLayout використовується для побудови динамічних призначених для користувача інтерфейсів, масштабованих і адаптованих до різних форматів і розширень екранів пристроїв, а також до їх орієнтацій. AutoLayout прийшов на зміну системі «пружин і розтяжок», які

застосовуються в попередніх версіях iOS SDK. Також AutoLayout робить інтернаціоналізацію більш простим завданням, розміщувати текст змінної довжини на екрані стає простіше, також підтримуються мови з напрямком письма справа наліво, такі як іврит і арабська. Робота AutoLayout заснована на зв'язках (constraints), які встановлюють геометричні відносини між уявленнями призначеного для користувача інтерфейсу. Наприклад, можна створити зв'язок, який говорить: «текстова мітка повинна бути закріплена на деякій відстані від лівого краю батьківського уявлення і з'єднана з лівим краєм кнопки, з додавання між ними проміжку в 10 px». AutoLayout бере задані зв'язки і математично обчислює ідеальні позиції і розміри для всіх уявлень. Розробнику не потрібно більше встановлювати 50 розміри уявлень вручну, задавати їх координати розташування ─ AutoLayout бере цю роботу на себе. Створювати зв'язки можна як програмно, так і за допомогою Interface Builder.

2.2.2 Архітектура Android

До складу Android входить комплект базових додатків: клієнти електронної пошти і SMS, календар, різні карти, браузер, програма для управління контактами і багато іншого. Всі додатки, що запускаються на платформі Android написані на мові Java (рис 2.4). Android дозволяє використовувати всю міць API, використовуваного в додатках ядра.

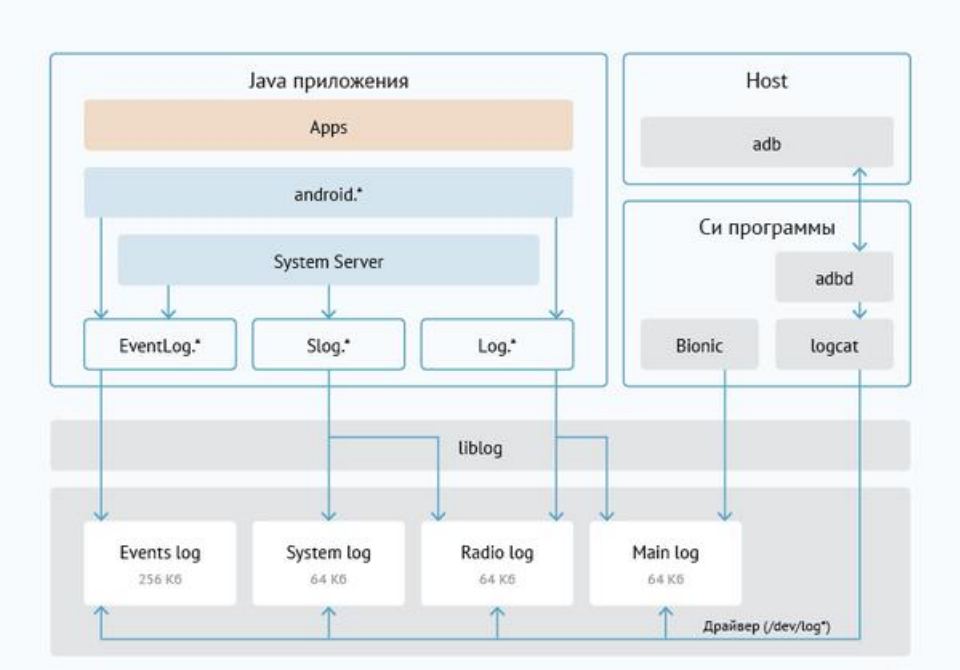


Рис. 2.5. Архітектура ОС Android

Архітектура побудована таким чином, що будь-який додаток може використовувати вже реалізовані можливості іншої програми за умови, що останнім відкриє доступ на використання своєї функціональності. Таким чином, архітектура реалізує принцип багаторазового використання компонентів ОС і додатків.

Основою всіх додатків є набір систем і служб:

1. Система уявлень (View System) ─ це багатий набір уявлень з розширюється функціональністю, який служить для побудови зовнішнього вигляду додатків, що включає такі компоненти, як списки, таблиці, поля введення, кнопки і т.п.
2. Контент-провайдери (Content Providers) ─ це служби, які дозволяють додаткам отримувати доступ до даних інших додатків, а також надавати доступ до своїх даних.
3. Менеджер ресурсів (Resource Manager) призначений для доступу до строкових, графічним і іншим типам ресурсів.

4. Менеджер повідомлень (Notification Manager) дозволяє будь-якому додатком відображати призначені для користувача повідомлення в рядку статусу.
5. Менеджер дій (Activity Manager) управляє життєвим циклом додатків і надає систему навігації по історії роботи з діями.

Платформа Android включає набір C / C ++ бібліотек, використовуваних різними компонентами ОС. Для розробників доступ до функцій цих бібліотек реалізований через використання Application Framework. Нижче представлені деякі з них:

1. System C library ─ BSD-реалізація стандартної системної бібліотеки C (libc) для вбудованих пристроїв, заснованих на Linux.
2. Media Libraries ─ бібліотеки, засновані на PacketVideo's OpenCORE, призначені для підтримки програвання і записи популярних аудіо і відео форматів (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG і т.п.).
3. Surface Manager ─ менеджер поверхонь управляє доступом до підсистеми відображення 2D- і 3D- графічних шарів.
4. LibWebCore ─ сучасний движок web-браузера, який надає всю міць вбудованого Android-браузера.
5. SGL ─ двигун для роботи з 2D-графікою.
6. 3D libraries ─ двигун для роботи з 3D-графікою, заснований на OpenGL ES 1.0 API.
7. FreeType ─ бібліотека, призначена для роботи зі шрифтами.
8. SQLite ─ потужний легкий двигун для роботи з реляційними БД.

До складу Android входить набір бібліотек ядра, які надають більшу частину функціональності бібліотек ядра мови Java.

Платформа використовує оптимізовану, регістр-орієнтовану віртуальну машину Dalvik, на відміну від неї стандартна віртуальна машина Java ─ стек-орієнтована. Кожна програма запускається в своєму власному процесі, зі своїм власним примірником віртуальної машини. Dalvik використовує формат

Dalvik Executable (*.dex), оптимізований для мінімального використання пам'яті додатком. Це забезпечується такими базовими функціями ядра Linux, як організація потокової обробки і низькорівневе управління пам'яттю. Байт-код Java, на якому написані ваші програми, компілюються в dex-формат за допомогою утиліти dx, що входить до складу SDK.

Android заснований на ОС Linux версії 2.6, тим самим платформі доступні системні служби ядра, такі як управління пам'яттю і процесами, забезпечення безпеки, робота з мережею і драйверами. Також ядро служить шаром абстракції між апаратним та програмним забезпеченням.

2. 3. Огляд напоширеніших методів оптимізації мобільних додатків під керуванням операційної системи iOS

2.3.1 Оптимізація ресурсів в iOS

При складанні додатків під iOS для оптимізації ресурсів використовується скрипт iphoneos-optimize з набору XCode. Працює він відмінно, але якщо копнути глибше, то стає ясно, що деякі файли не перетискаються, а інші хоч і трохи зменшуються, але все-одно далекі від ідеалу. Можна сказати, що завдання скрипта зробити файли більш сумісними з iPhone, щоб вони швидше читалися або розпаковувалися, але швидше за все це мало сенс лише на старих iPhone 1 і іже з ними, а вже на процесорах 1 ГГц з ARM 7 це відверто не актуальне.

За допомогою простих оптимізацій і парочки програм з набору MacPorts можливо домогтися істотного зменшення PNG і JPG картинок в кінцевій програмі, а при бажанні і інших видів даних.

Що саме виконує ключ-iphone неможливо впізнати, але за великим рахунком це і не так важливо. При детальному розгляді видно, що якщо pngcrush повернув помилку або не зміг зменшити файл, то тимчасовий файл видаляється, а основний залишається без змін. Під час оптимізації, так відбувалося з якимось файлом browser.png:

Recompressing browser.png

Total length of data found in IDAT chunks = 433949

IDAT length with method 120 (fm 1 zl 9 zs 1) = 512501

Best pngcrush method = 120 (fm 1 zl 9 zs 1) for browser_iphone.png

(18.10% IDAT increase)

(18.11% filesize increase)

CPU time used = 2.569 seconds (decoding 0.040,

encoding 2.490, other 0.039 seconds)

Без ключа-iphone ситуація була кращою і файл зменшився:

Recompressing browser.png

Total length of data found in IDAT chunks = 433949

IDAT length with method 1 (fm 0 zl 4 zs 0) = 740769

IDAT length with method 2 (fm 1 zl 4 zs 0) = 611778

IDAT length with method 3 (fm 5 zl 4 zs 1) = 485419

IDAT length with method 9 (fm 5 zl 2 zs 2) = 743935

IDAT length with method 10 (fm 5 zl 9 zs 1) = 427514

Best pngcrush method = 10 (fm 5 zl 9 zs 1) for browser_tmp.png

(1.48% IDAT reduction)

(1.47% filesize reduction)

CPU time used = 3.949 seconds (decoding 0.176,

encoding 3.766, other 0.007 seconds)

Інший шлях—GPL утиліта optipng. З MacPorts вона встановлюється командою: sudo port install optipng

Результат роботи утиліти з тим же самим browser.png:

** Processing: browser_opti.png

1024x1024 pixels, 4x8 bits/pixel, RGB+alpha

Input IDAT size = 433949 bytes

Input file size = 434043 bytes

Trying:

zc = 9 zm = 8 zs = 1 f = 5 IDAT size = 427390

Selecting parameters:

zc = 9 zm = 8 zs = 1 f = 5 IDAT size = 427390

Output IDAT size = 427390 bytes (6559 bytes decrease)

Output file size = 427484 bytes (6559 bytes = 1.51% decrease)

Як можно побачити, розмір файлу став ще менше, ніж у pngcrush, а швидкість роботи при цьому помітно вище. У деяких інших випадках розрив ще більше помітний.

Найважливіше те, що отримані PNG файли відмінно відкриваються на iPhone і iPad, ніяких візуальних спотворень в них немає, різниця в швидкості відкриття відсутня або не помітна.

Додатково можна дописати в скрипт інші розширення (JPEG, JFIF, JPE, JIF, etc.), додати зменшення якості та ін. Пралельно з цим можна оптимізувати бази даних SQLite за допомогою такої команди:

sqlite3 database.sqlite vacuum;

Команда створить базу з усіма даними, викинувши різне сміття, залишки старих транзакцій і т.п.

Насправді, є ще як мінімум два цікавих методу зменшення розміру PNG файлів. Перший не зовсім канонічний і може бути сприйнятий як єресь, але факт є факт: компілятор Android (точніше apktool) вміє відслідковувати картинки з малою кількістю квітів і переводити їх в Paletted формат. Більш того, навіть повнокольорові PNG можуть стати ще менше. Досить додати потрібні зображення в папку res / drawable робочого проекту, зібрати його і потім списати оптимізовані файли з папки bin / res / drawable. Звичайно, це вимагає деяких навичок роботи з Android SDK і не зовсім відноситься до розробки під iOS.

Другий метод більш універсальний: зменшити колірний обхват файлу з RGBA8888 до RGB565 або RGBA4444. При цьому розмір може як зрости через dithering, так і помітно зменшиться у випадку з мальованими зображеннями. Для цих операція я написав власну консольную утиліту, але її розгляд лежить вже за межами цієї статті.

2.3.2 Оптимізація часу запуску

Очевидно, що з двох додатків з однаковим набором функцій користувач вибере те, яке швидко запускається. Якщо альтернативи немає, а під час запуску

програми довго, користувача це буде дратувати, він буде рідше повертатися в ваш додаток, писати погані відгуки. І, навпаки, якщо під час запуску швидко, то все буде здорово.

Не варто забувати і про обмеження часу запуску в 20 секунд, при перевищенні якого система перериває завантаження вашої програми. На слабких пристроях ці 20 секунд досить реально перевищити.

Почнемо з того, що є час запуску і як його заміряти. Час запуску ─ це час від натискання користувачем на іконку програми і до моменту, коли програма готова до використання.

У найпростішому випадку можна вважати, що програма готова до використання після закінчення функції DidFinishLaunching, тобто коли завантажений основний інтерфейс програми. Однак якщо ваше застосування повинне на старті кудись сходити за даними, повзаімодействовать з базами даних, оновити UI, це теж доводиться враховувати. Тому що вважати закінченням запуску ─ особиста справа кожного розробника.

Визначилися з тим, що є час запуску, починаємо його заміряти. Виявляємо, що час дуже сильно скаче (рис 2.6).

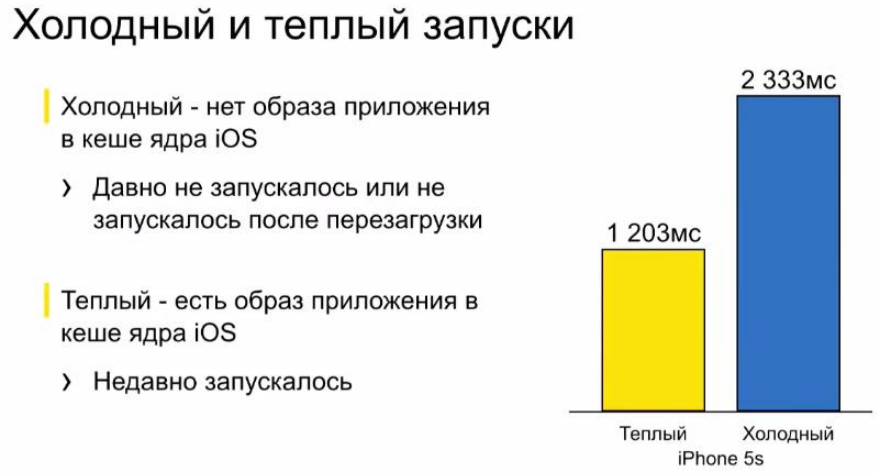


Рис. 2.6. Різниця в часі при теплому та холодному старті

Тут потрібно ввести поняття холодного та теплого запуску. Холодний запуск ─ це коли додаток давно завершило свою роботу і було видалено з кешу операційної системи. Холодний запуск завжди відбувається після перезавантаження програми, але, в принципі, так WWDC рекомендує

моделювати його в своїх тестах. Теплий запуск — це коли додаток було завершено нещодавно.

Різниця виникає через те, що запуск складається з двох великих етапів:

- підготовка образу додатки;
- запуск коду — коли запускається функція main.

При підготовці образу додатки система повинна:

- завантажити всі динамічні бібліотеки, для кожної динамічної бібліотеки перевірити її цифровий підпис, відобразити її ВАП;
- оскільки бібліотека може бути розташована в будь-якому місці пам'яті, потрібно поправити покажчики, підставити адреси невідомих символів в інших бібліотеках;
- створити контекст Objective-C, тобто зареєструвати C-класи, селектори, категорії, уникализировать селектори, додатково там виконується й інша робота;
- проініціалізувати класи, здійснити телефонний дзвінок + load і конструктора глобальних змінних C ++.

У разі холодного запуску цей pre-main може бути на порядок довше, ніж в разі теплого запуску (рис 2.7).

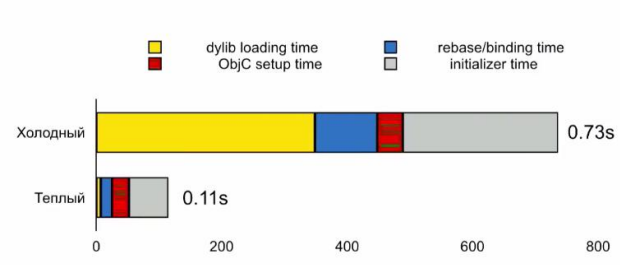


Рис. 2.7. Заміри часу запуска зіставних частин додатку при теплому та холодному старті

За рахунок цього і виходить така велика різниця між холодним і теплим запуском.

Особливо сильно збільшується час завантаження динамічних бібліотек в разі Swift-a. Таким чином, виконуючи замір запуску додатку, потрібно

обов'язково враховувати не тільки той відрізок, коли код працює, але і pre-main, коли система збирає веб-додаток, а також враховувати холодний запуск.

Замір pre-main — нетривіальне завдання, оскільки наш код там не працює. На щастя, в останніх iOS (9 і 10) Apple додала змінну оточення DYLD_PRINT_STATISTICS, при включенні якої в консоль виводиться статистика роботи системного завантажувача (рис 2.8).

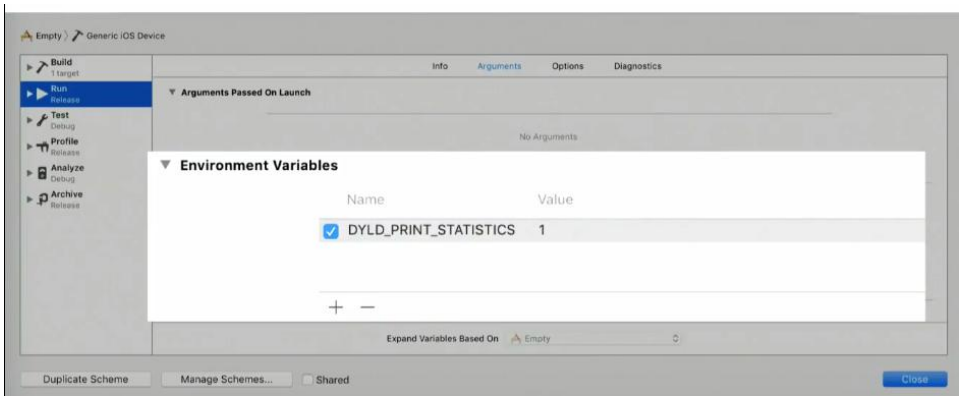


Рис. 2.8. Додавання змінної оточення

```
Total pre-main time: 677.82 milliseconds (100.0%)
         dylib loading time: 520.33 milliseconds (76.7%)
        rebase/binding time:  54.31 milliseconds (8.0%)
            ObjC setup time:  43.16 milliseconds (6.3%)
           initializer time:  59.99 milliseconds (8.8%)
           slowest intializers :
             libSystem.B.dylib :   4.35 milliseconds (0.6%)
                       TestApp :  74.83 milliseconds (11.0%)
```

Рис. 2.9. Результат виконання додатку з включенною змінною оточення

Виводиться повний час pre-main і далі поетапно:

- час завантаження динамічних бібліотек;
- час rebase / binding — тобто правки покажчиків і зв'язування;
- час створення ObjC контексту;

- час ініціалізації — там, де + load та глобальні змінні.

Потрібно прагнути побачити час запуску саме порожнього проекту, оскільки швидше, ніж з порожнім проектом, додаток запускатися не може.

Тут натрапив на неприємний сюрприз мови Swift. Візьмемо простий додаток на Objective-C, заміряємо його pre-main (рис 2.10).

```
Total pre-main time:  19.40 milliseconds (100.0%)
         dylib loading time:   0.72 milliseconds (3.7%)
        rebase/binding time:   1.14 milliseconds (5.8%)
            ObjC setup time:   5.02 milliseconds (25.9%)
           initializer time:  12.51 milliseconds (64.4%)
           slowest intializers :
             libSystem.B.dylib :  10.27 milliseconds (52.9%)
                CoreFoundation :   0.66 milliseconds (3.4%)
```

Рис. 2.10. Результат виконання додатку побудованого на мові Object-C

На мові Object-C час завантаження динамічних бібліотек буде менше мілісекунди. Робимо таке ж додаток на Swift, запускаємо на тому ж пристрої (рис 2.11):

```
Total pre-main time: 232.19 milliseconds (100.0%)
         dylib loading time: 209.33 milliseconds (90.1%)
        rebase/binding time:   7.98 milliseconds (3.4%)
            ObjC setup time:   8.49 milliseconds (3.6%)
           initializer time:   6.37 milliseconds (2.7%)
           slowest intializers :
             libSystem.B.dylib :   3.14 milliseconds (1.3%)
```

Рис. 2.11. Результат виконання додатку побудованого на мові Swift

Завантаження динамічних бібліотек — 200 мілісекунд — на 2 порядки більше. Щоб розібратися чому так, потрібно порівняти лог-файли обох додатків.

Порівнюємо логи проектів на Objective-C і Swift. Можно побачити, що в обох випадках завантажуються 146 динамічних бібліотек, які є системними, і бінарний додаток. Але в Swift додатково завантажуються ще дев'ять підозрілих бібліотек з бандла додатків ― з папки Frameworks, що називаються libswift ***. Dylib. Це swift standard libraries.

Якщо подивитися на лог компіляції додатку, то один з останніх етапів― це coping swift standard libraries. Потрібно зрозуміти звудки вони беруться.

Справа в тому, що swift дуже швидко розвивається, і його розробники не заморочуються дотриманням зворотного бінарної сумісності. Тому якщо ви зібрати який-небудь модуль на swift 3.0, то навіть на swift 3.01 не буде можливості його використовувати. Компілятор напише, що цього робити не можна. Swift ще не може бути частиною iOS, адже інакше на старих iOS новий swift запускатися не буде. Тому додатки завжди тягнуть за собою Swift runtime ― бібліотеки підтримки swift-у, на відміну від Objective-C, який вже давно є частиною системи.

Тому ми отримуємо наступні висновки:

- хто б що не говорив, завантаження системних бібліотек оптимізована (я сам витратив багато часу на те, щоб остаточно в цьому переконатися). Навіть якщо у вас 200 або 300 системних бібліотек вантажиться, це все одно не впливає на час запуску, так що не витрачайте час, намагаючись скоротити їх кількість;
- а ось бібліотеки з бандла додатків вантажаться довго. На жаль, до них відносяться swift standard libraries;
- навіть порожнє додаток на iPhone вантажиться секунду (при холодному запуску). Відповідно, ваше додаток не може завантажуватися менше секунди;
- коли-небудь ця проблема зникне. Уже готується swift 4, і swift standard libraries просто стануть частиною системи, а додатки перестануть за собою ці бібліотеки тягати.

Припустимо, розроблений додаток містить багато подів, написаних на мові swift, і десь усередині себе використовує MapKit з iOS SDK (рис 2.12).

```
...
use_frameworks!

target :OriginalApp do
    pod 'AlamofireObjectMapper'
    pod 'AlamofireImage'
    pod 'FacebookLogin'
    pod 'ObjectMapper'
    pod 'PhoneNumberKit'
    pod 'UIImageEffects'
    pod 'pop'
    pod 'KissXML'
    pod 'MTDates'
    pod 'Punycode-Cocoa'
    pod 'YandexSpeechKit'
    pod 'YandexMapKit'
end
```

Рис. 2.12. Початковий набір подів

Починаемо замір часу запуску (рис 2.13).

```
Total pre-main time: 741.74 milliseconds (100.0%)
         dylib loading time: 627.80 milliseconds (84.6%)
        rebase/binding time:  28.87 milliseconds (3.8%)
            ObjC setup time:  36.00 milliseconds (4.8%)
           initializer time:  49.04 milliseconds (6.6%)
           slowest intializers :
             libSystem.B.dylib :   4.57 milliseconds (0.6%)
                  OriginalApp :  28.50 milliseconds (3.8%)
```

Рис. 2.13. Результат зміру часу запуску додатку

Час запуску в три рази вище, ніж у порожнього додатку. Така ситуація складається за рахунок динамічних бібліотек, тобто у порожнього додатку вони завантажуються 200 мілісекунд, а у розробленоого — вже 600.

Як прибрати завантаження зайвих динамічних бібліотек, які приходять від подів? Для цього ми можемо скористатися плагіном для cocoa pods, який називається cocoapods-amimono. Він патчіть скрипти і xcconfig, які генеруються подами, так, щоб після компіляції подовскіх бібліотек залишилися об'єктнікі влінковать відразу ж в бінарний файл програми, і таким чином позбутися від необхідності лінковки з динамічними бібліотеками. Як ним скористатися?

У Podfile додаємо використання плагіна і post-install. Таким чином, час запуску зменшується практично в два рази: час завантаження динамічних бібліотек падає з 600 секунд до 320 (рис 2.14).

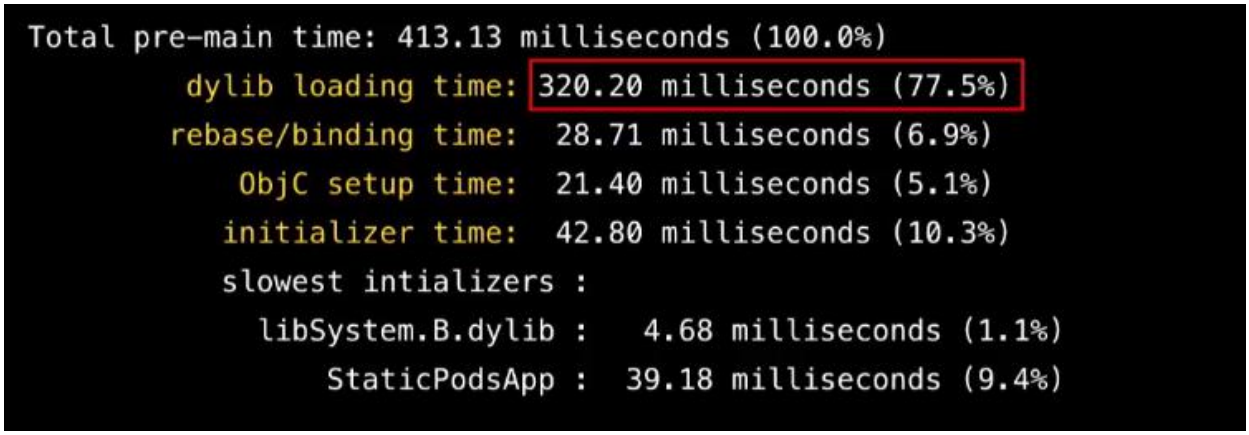


Рис. 2.14. Результат виконання додатку з плагіном для cocoa pods

Такий результат досягнутий за рахунок того, що зникли всі динамічні бібліотеки від подів.

На жаль, існують декілька недоліків цього рішення, пов'язаних з тим, що у нього немає великого ком'юніті (люди робили його практично під себе):

- cocoapods-amimono пропускає інтеграцію подів, що поставляються фреймворками, тобто такі поди потрібно самому додати в додаток і виконати інтеграцію;

- в такому методі немає контролю за тим, які поди вмержіть в бінарний файл, а які залишити як є;
- таргети, назва яких містять «test», відпускаються.

Однак, для багатьох розробників, переваги цього методу оптимізації перекривають його недоліки.

2. 4. Огляд напоширеніших методів оптимізації мобільних додатків під керуванням операційної системи Android

Android Studio була вдосконалена в останній час, було додано все більше і більше інструментів, щоб допомогти розробнику знайти та проаналізувати проблеми, пов'язані з продуктивністю. Вкладка «Пам'ять» у вікні Android-Studio показує нам зміну обсягу даних в куче за течією часу (рис 2.15).

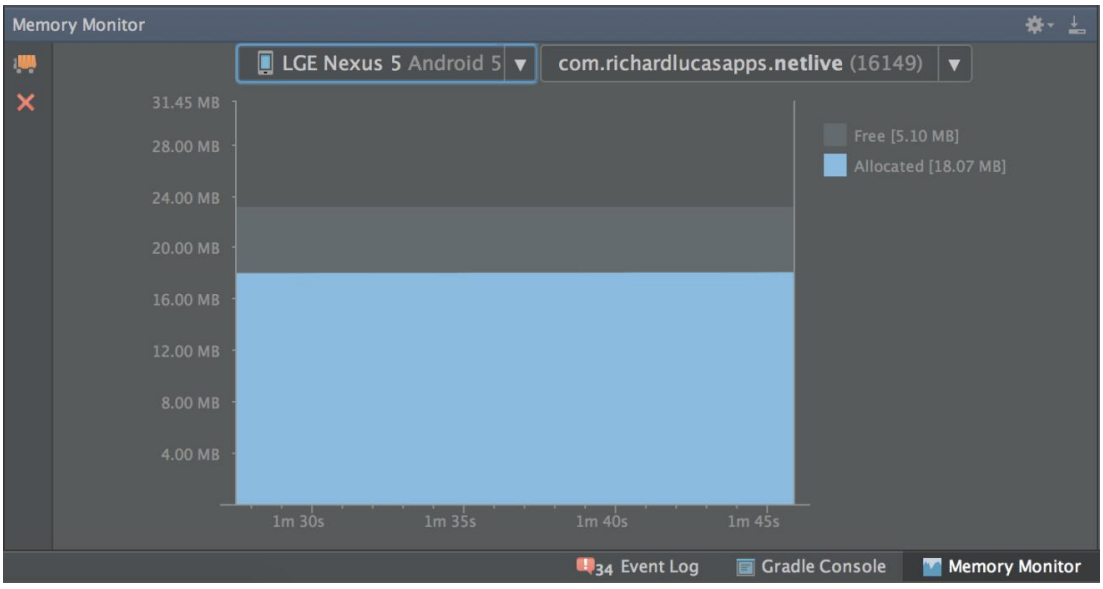


Рис. 2.16. Вкладка «Пам'ять» у вікні Android-Studio

2. 4. 1. Профіліровка GPU

Новым добавлением в Android Studio 1.4 є профілірование графічного процесора рендерінгу. Під вікном Android перейдіть на вкладку GPU, і ви побачите графік, що показує час, який потрібно, щоб зробити кожен кадр на екрані (рис 2.16).

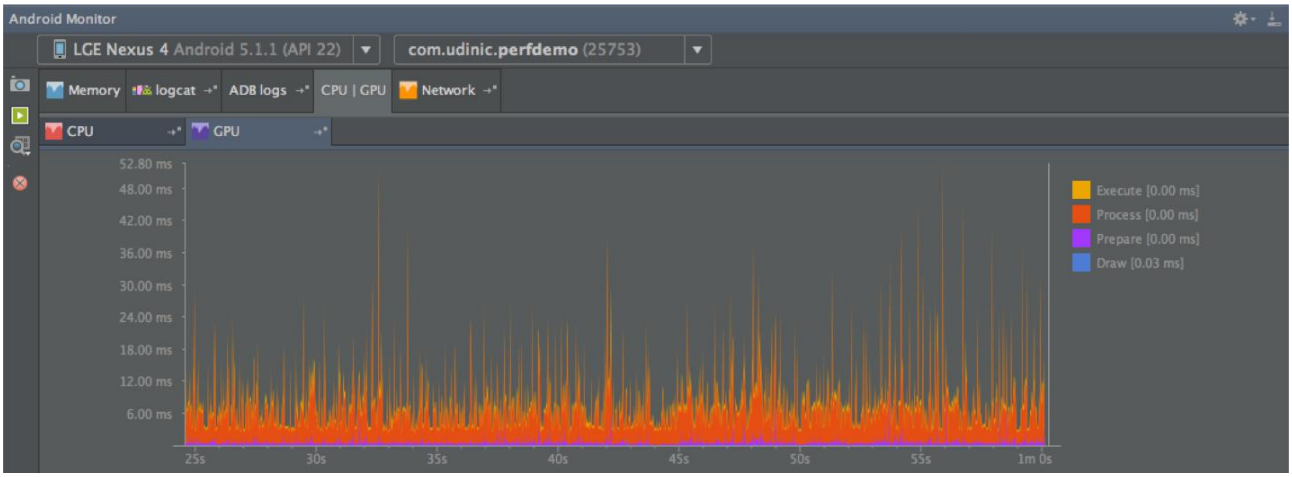


Рис. 2.16. Результат виконання трасування додатку

Кожна панель на графіку представляє собою один відзнятий кадр, а колір представляє різні етапи процесу.

Draw (синій) — представляє метод View # onDraw (). Ця частина будує об'єкти DisplayList, які потім будуть конвертовані в OpenGL команди, зрозумілі для GPU. Високі показники можуть бути через використовуваних складних представлень, що вимагають більшого часу для побудови їх списку відображення, або багато представлень стали недійсними в короткий проміжок часу.

Підготовка (фіолетовый) — у Lollipop було додано ще один потік, щоб допомогти швидше виготовити інтерфейс користувача. Він називається RenderThread. Він відповідає за перетворення списків відображення в OpenGL команди та відправлення їх графічному процесору. Як тільки це відбувається, потоки UI можуть переходити до обробки наступного кадру. Час, витрачений потоком UI, щоб передавати всі необхідні ресурси RenderThread, відображається на даній стадії. Якщо довгі списки відображення, цей крок може зайняти більше часу.

Процес (червоний) — виконання списків відображення для створення OpenGL команд. Цей крок може зайняти більше часу, якщо списки довгі або складні, тому що потрібно перерізати безліч елементів. Представлення може

бути перерісовано, так як стало недійсним або було відкрито після пересування схованого його представлення.

Виконання (жовтий) ─ відправка команд OpenGL на GPU. Цей етап може тривати довго, так як CPU відправляє буфер з командами на GPU, очікуючи отримати зворотний чистий буфер для наступного кадру. Кількість буферів обмежено, і якщо GPU занадто зайнятий, процесор буде чекати, доки висвободиться перший. Тому, якщо наблюдаються високі значення на цьому кроці, це може означати, що GPU був зайнятий рисуванням інтерфейсу, який може бути надто складним, для того щоб бути відображеним за коротший час.

2. 4. 2. Апаратне прискорення

Коли аппаратне прискорення було включено в Honeycomb, розробники отримали нову модель показу для відображення програми на екрані. Це включає структуру DisplayList, яка записує команди показу представлень для більш швидкого отримання зображення. Але є ще одна цікава особливість, про яку розробники зазвичай забувають ─ рівні представлення.

Використовуючи їх, можливо виготовити представлення в екранному буфері і маніпулювати з ним. Це чудово підходить для анімації, тому що стає можливо анімітувати складні представлення швидше. Без рівнів анімація представлених стане недоступною після зміни властивостей анімації (таких як x-координата, масштаб, альфа-значення та ін.). Для складних представлень ця недієздатність поширюється на представлення потомків, і вони в свою чергу будуть перерисовуваних себе, що є дорогою операцією. Виконуючи рівні представлення, спираючись на технічні засоби, текстура для нашого представлення створюється в графічному процесорі. Існує декілька операцій, які можливо застосувати до текстури, без необхідності її заміни, такі як зміна положення по осям x / y, обертання, альфа та інші. Все це означає, що можливо анімінувати складні представлення на екрані без їх заміни в процесі анімації! Це робить анімацію більш гладкою.

Висновок

Оптимізація це, з однієї сторони, інструментарій та технологія його використання, що дозволяє навіть розробникам середньої кваліфікації долучитися до виконання оптимізаційних процессів над своїми проектами, а з іншої пошаговий метод вдосконалення фінальних результатів роботи розробленого мобільного додатку, що отримується тільки через певну кількість часу та спроб.

В останній час, завдяки інтеграції все більшої кількості інструментів для оптимізації мобільних додатків у середи розробки, а також численним функціям та можливостям, які вони предоставляють, цей інструментарій був вибран в якості основи майбутньої розробки.

У час оптимізаційні процеси виконуються вже після того, як мобільний додаток розроблений, а в інших випадках, випущений в реліз. Таким чином, сторонні розробники економлять на процесі розробки мобільних додатків, відсуваючи такий важливий момент у розробці мобільного дадтку, як їх оптимізація на інший раз.

РОЗДІЛ 3.

ВИКОРИСТАННЯ ВДОСКОНАЛЕНОГО МЕТОДУ ОПТИМІЗАЦІЇ НА РОЗРОБЛЕНИХ ДОДАТКАХ

2. 5. Загальний план розробки мобільних додатків

Проектування та дизайн. На цьому етапі використовуємо прототипи, які можуть бути створені на листку А4 від руки. На них потрібно наглядно продумати, як буде відбуватися навігаці у додатку.

При розробці дизайну обов'язково використовуються гайдлайни. Гайдлайн в загальному розумінні ― це документ, який випускає компанія, і за яким дизайнери і розробники розуміють принцип побудови взаємодії додатка з користувачем. Наприклад, для iOS кнопки треба робити круглими, а для Android ― квадратними. Таким чином результат роботи дизайнера найчастіше складається з макетів, гайдлайни і нарізки графіки.

Макети найкраще подавати «облінкований», наприклад за допомогою ProtoTypr, щоб була зрозуміла логіка переходів. Гайдлайни містять в собі інформацію про відступи, розмірах, візуальних ефектах, механіці анімації та ін. Цей етап можна пропустити, якщо у вашому проекті один дизайнер і один розробник. Третя частина результату ― нарізка графіки. Вона повинна містити мінімум необхідних графічних ресурсів (піклуємося про загальну вагу додатків), мати версії для різних екранів різних розмірів (рис 3.1).

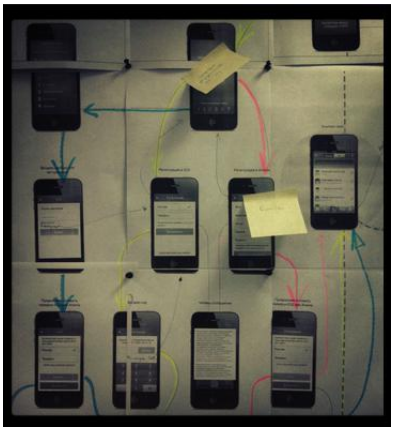


Рис. 3.1. Гайдлайну для мобільного додатку

Розробка. Після отримання макетів, гайдлайнів та нарізки, починається робота розробника. Починаємо розробку всього того, що придумали, і очікуємо ранній результат. Це не означає, що робота над архітектурою і призначеним для користувача інтерфейсом закінчена. Іноді з'являються цікаві ідеї, які вносять корективи в початковий план. Коли розробка завершена, настає стадія тестування.

Тестування. Існує чимала кількість способів протестувати розроблені додатоки. У мобільній розробці тестувальник—це людина, навколо якої одні телефони. Наприклад, є величезна шафа, в якій лежать як старі телефони, так і найсвіжіші новинки. Всередині потрібно намагатися тестувати тест-кейсами. Якщо впроваджується новий функціонал, по її опису складається тест-план. Існують сервіси, що допомагають в тестуванні. Деякі з них були розглянуті у другому розділі дипломної роботи. У цьомму випадку доцільно використовувати фреймворк Calabash [13].

Отже, формулювання поетапної розробки виглядає наступним чином:

- Виконання проектування та створення дизайну;
- Розробка кодів мобільних додатків;

2. 6. Програмне забезпечення

Для разробки додатку під керуванням операційної системи Android, потрібно завантажити та встановити Android-Studio IDE, її можливо знайти на офіційному сайті [14]. Більш детальніший опис про цю IDE можливо знайти у першому розділі дипломної роботи.

Android-Studio має наступні вимоги до комп’ютеру: операційна система Microsoft Windows 7/8/10 (32 або 64 біт), 3 Гб RAM або оперативної пам’яті, 2 Гб свободного місця на жорсткому диску комп’ютера.

З перерахованих вище вимог, можна зробити висновок, що Android-Studio ― це зручне середовище розробки, яке можливо встановлювати на більшість сучасних комп'ютерів.

Для розробки додатку під керуванням операційної системи iOS потрібно завантажити та встановити Xcode IDE, її можливо знайти на офіційному онлайн магазині під назвою App Store [3]. Більш детальніший опис про цю IDE можливо знайти у першому розділі дипломної роботи.

Xcode має наступні вимоги до комп’ютеру: операційна система Mac OS Sierra або краще, 4 Гб RAM або оперативної пам’яті, від 2 Гб свободного місця на жорсткому диску комп’ютера.

З перерахованих вище вимог, можна зробити висновок, що Xcode ― це зручне середовище розробки, однак, щоб його встановити та користуватися потрібно мати комп’ютер з операційною системою iMac. Такі комп’ютери мають дуже велику початкову вартість, що суттєво зменшує кількість активних розробників, що мають бажання почати розробляти мобільні додатки під керівництвом операційної системи iOS.

2. 7. Пояснення прикладу виконання розроблених методів оптимізації на прикладі отриманих мобільних додатків

Для демонстрации ефективності виконання розробленого методу оптимізації мобільних додатків, розроблені методи оптимізації будуть застосовані до розроблених мобільних додатків під керуванням операційних систем Android та iOS. Розроблені додатки мають змогу продемонструвати параметри роботи до та після застосування розробленого методу оптимізації, що дає можливість проаналізувати ефективність застосування розробленого методу.

2. 8. Розробка та побудова додатку під керуванням операційної системи iOS

Розробка додатку почалась із створення робочого шаблону, який буде використаний далі (рис 3.2).

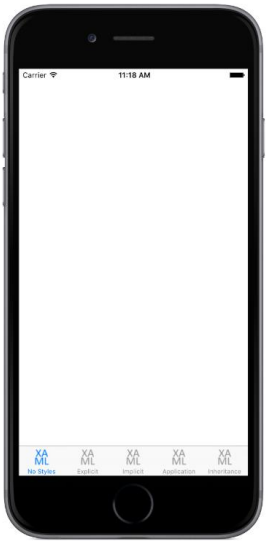


Рис. 3.2. Початковий шаблон додатку

На наступномі етапі була спроектована та розроблена частина додатку, що відповідає за представлення інформації о використанні існуючих ресурсів смартфону у реальному часі (рис 3.3). Показники змінюються, якщо користувач заповнює чи звільняє оперативну пам’ять пристрою, однак цей процес важно продемонструвати при тестовій експлуатації.

Рис. 3.3. Вид відображення рівня завантаженності мобільного пристрою

Наступним кроком є доповнення отриманного результату шляхом збільшення числа показників, що відповідають за основні елементи смартфону. Отриманий результат представлений на рисунку 3.4.

Рис. 3.4. Фінальний результат розробленого мобільного додатку під керуванням мобільної операційної системи iOS

Відображені показники під час виконання додатку:

- clang — виконання прямих команд центрального процесору найнижчого рівня;
- kernel_task — кількість працюючих потоків головного процесору;
- ;aunchd — загальна кількість працюючих додатків у системі;
- UserEventAgent — кількість процентів навантаженности головного процесору, які займає поточний сеанс користувача;
- notifyd — кількість активних повідомлень операційної системи;
- security — кількість діючих протоколів безпеки;

- powerd — кількість елементів, що отримують живлення від акумуляторної батареї пристрою;
- syslogd — кількість записів виконання системних команд найнижчого рівня;
- opendirectoryd — загальна кількість відкритих директорій пристрою.

2. 9. Розробка та побудова додатку під керуванням операційної системи Android

Розробка додатку почалась із створення робочого шаблону, який буде використаний далі (рис 3.5).

Рис. 3.5. Початковий шаблон додатку

На наступномі етапі була спроектована та розроблена частина додатку, що відповідає за представлення інформації о використанні існуючих ресурсів смартфону у реальному часі (рис 3.6). Показники змінюються, якщо користувач заповнює чи звільняє оперативну пам’ять пристрою, однак цей процес важно продемонструвати при тестовій експлуатації.

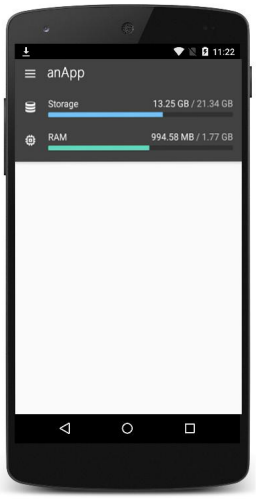


Рис. 3.6. Вид відображення рівня завантаженності мобільного пристрою

Наступним кроком є доповнення отриманного результату шляхом збільшення числа показників, що відповідають за основні елементи смартфону. Отриманий результат представлений на рисунку 3.7.

Рис. 3.7. Фінальний результат розробленого мобільного додатку під керуванням мобільної операційної системи Android

Опис відображених показників під час виконання додатку.

- Storage ― відображення кількості свободного фізичного місця, що залишилося на присрої;
- RAM ― відображення кількості зайнятої та вільної оперативної пам’яті пристрою;
- SD card storage ― відображення кількості свободного фізичного місця, яке залишилося на окремій карті пам’яті, що встановлена на пристрої;
- System cache ― відображення зайнятого та свободного фізичного місця, яке виділено під збереження системних файлів операційної системи за час цієї сесії;
- Processor ― відображеня проценту навантаження головного процесору пристрою у реальному часі.

2. 10. Аналіз отриманих результатів

Поставлена дослідженням ціль вимагала реалізації нетипової задачі ― доступу до інформації о ресурсах мобільних пристроїв у реальному часі. І хоча такий тип задач реалізован великою кількістю розробників мобільних додатків, на загальному плані, задачі такого типу зустрічається досить рідко ― приблизно 10% від загальної кількості додатків.

Під час розробки даних мобільних додатків, на практиці було відображено реальна різниця у підходах до проектування та розробки мобільних додатків під куруванням операційних систем Android та iOS. Як було зазначе у першому розділі дипломної роботи, в основі системи Android лежить ядро Linux і філософія відкритого ПЗ. Це означає, що є можливість всім бажаючим взяти рішення від Google за основу і доопрацювати відповідно до своїх уявлень. Таким чином, вся інформація, яку може предоставити мобільний телефон, лежить майже на поверхні. Досить лише мати права адміністратору або розробника, і операційна система Android видасть вам усі пожливі дані при першій вимозі.

Більш за те, не тільки відобразить, а ще й надасть можливість користувачу вручну змінювати найголовніші показники та елементи операційної системи, їх поведінку та навантаження. Наприклад, маючи всі права доступу користувач має змогу змінити тактову частоту головного процесору. При цьому, користувачу не будуть протиставити ніяких обмежень, тобто він має змогу змінювати практично будь-які показники від негативно-критичних до позитивно-критичних.

Керуючись такою особливістю операційної системи Android, більшість розробників мало звертають увагу на оптимізаційні заходи на цій системі, тому що у користувача є змогу користуватися мобільними додатками, які мають приблизні повноваження до вказаних вище. Вони мають інтуітивно зрозумілий інтерфейс користувача, що надає змогу більшості користувачів мобільних пристроїв на системі Android змінювати основні показники своєї операційної системи одним натисканням на потрібну кнопу.

У цей час, ситуація з мобільними пристроями під керуванням опработающегоераційної системи iOS зовсім інша. iOS ─ це закрита система. Тобто, при розробці мобільного додатку для цієї системи, розробник не має можливостей вказувати операційній системі, що вона повинна при натисканні цієї кнопки очистити оперативну пам’ять пристрою, або виділити більше оперативної пам’яті для вказаного працюючого додатку і т.д. Все що можливо домогтися від iOS розробнику та, у майбутньому, користувачу це моніторинг, тобто відображення показників головних елементів операційної системи, без змоги якимось чином впливати на них.

Через це, для операційної системи iOS майже відсутні додатки такого типу, що був розроблений. В них немає необхідності. Це не значить, що для мобільних додатків під керуванням iOS немає оптимізаційних методів, вони є. Різниця між виконанням оптимізації у iOS від Android лежить у тому, що для додатку на iOS виконати оптимізацію можливо лишень у етапі створення та розробки мобільного додатку. В Android оптимізацію можливо виконати й після встановлення бажаного мобільного додатку шляхом, що описаний вище, що не є

правильним підходом до розробки та підтримання додатку, але який використовують не дуже кваліфіковані або дуже економні розробники.

Такий підхід до захисту в операційнії системі iOS обумовлений бажанням збереження її високої надійності. Розробникам залишається тільки підлаштовуватися під такі вимоги, а майбутнім покупцям вибирати, що їх влаштовує більше ─ свобода дій і варіативність налаштувань, або надійність, швидкість роботи та довготривала підтримка старих версій.

2. 11. Процесс розробки модернізованого методу оптимізації мобільного додатку під керівництвом операційної системи iOS

Проаналізувавши та виконавши тестування означених у другому розділі дипломної роботи методів оптимізації додатків під керівництвом операційної системи iOS дійшов висновку, що оптимально найрезультативнішим методом оптимізації виявився: оптимізація ресурсів та оптимізація часу запуску. Однак, при розробці я враховувах зменшення кількості часу, який необхідно затрачувати розробнику, при використанні цих методів оптимізації порознь та підвищення ефективності оптимізації.

Модернізація методу оптимізації часу запуску.

При описанні методу оптимізації часу запуску у другій частині дипломної роботи, було відзначено, що найбільш повільним елементом серед компільованих файлів є файли стандартних біблиотек, які підгружаються разом із запуском додатку по черзі. Оптимізація цим методом полягає у примусовому відключенні більшості бібліотек за непотрібністю. Але метод описує такий тип оптимізації лишень для бібліотек, що створені на мові Swift.

Модернізація цього методу оптимізації полягає у тому, що я відстежив які саме біблиотеки та статичні файли, що створені на мові Objective-C включені у етап загрузки разом із стартом додатку. Виявилось, що деякі стандартні бібліотеки, які створені на мові Objective-C під час завантаження додатку, тягнуть за собою щей й додаткові бібліотеки, які створені на мові Swift та Objective-C. Відключи їх завантаження разом із додатком було неможливо, тому

був використанний спосіб так званого “уповільненого” завантаження. Цей спосіб не виключає завантаження непотрібних бібліотек разом із стартом додатку, однак, використовуючи спеціальну команду dlopen, що знаходиться в API dlfcn. Команда виглядає наступним чином:

void *handle = dlopen(pathStr, RTLD_LAZY);

При завантаженні динамічної біблиотеки через dlopen, вона приймає першим параметром шлях до бібліотеки (яка знаходиться в розділі Frameworks розробленого додатку), dlopen повертає декриптор, який у майбутньому буде використовуватися у функції dlsym, яка завантажує окремі символи. Таким чином, завантаження вказаної бібліотеки буде виконуватися не цілим файлом, як відбувається зазвичай, а посимвольно, що і мається на увазі під терміном “лінива” або “уповільнена” завантаження. Більш того, використовуючи такий спосіб, можливо завантажувати не лише бібліотеки, а й цілий код додатку, розбивши його на модулі та завантажуючи їх “ліниво”. Кінцевий результат показав, що при такій оптимізації час завантаження додатку скоротився на 30 мілісекунд.

Ітогові показники завантаження додатку з оригінальним методом оптимізації часу запуску, та з використанням модифікованого методу оптимізації представлені на рисунках 3.8 та 3.9.

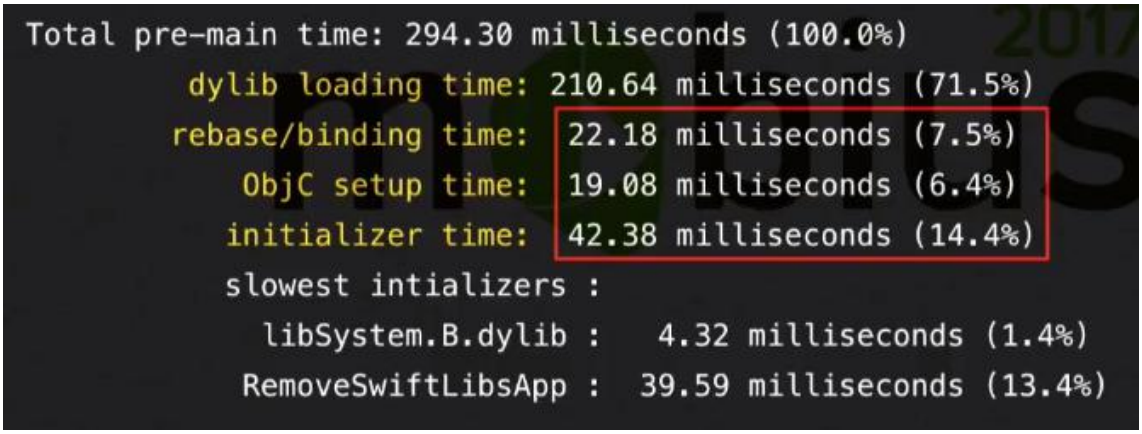


Рис. 3.8. Результат виконання оригінального методу оптимізації часу запуску

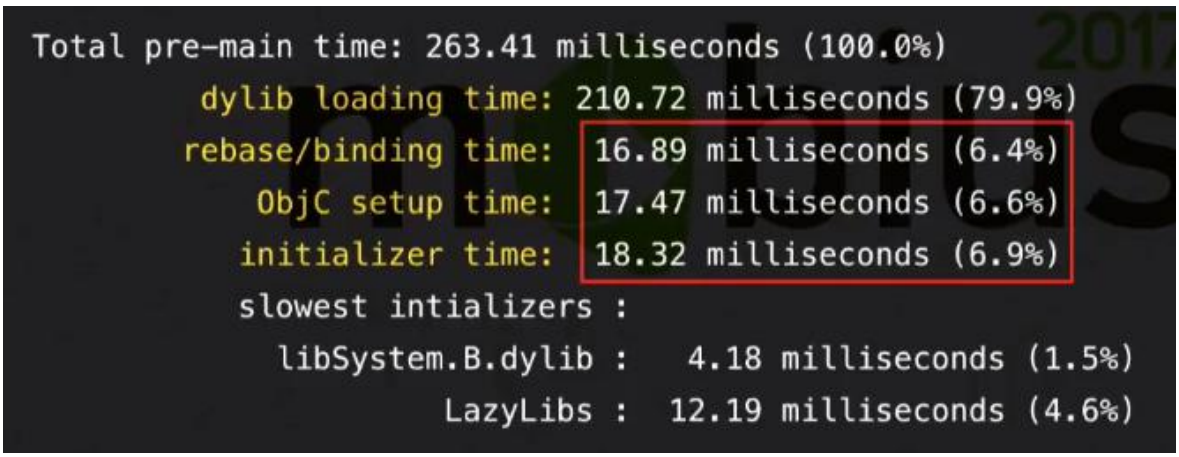


Рис. 3.9. Результат виконання модифікованого методу оптимізації часу запуску

Підсумовуючи вище сказане, модернізований метод оптимізації виправдав себе і має право на існування.

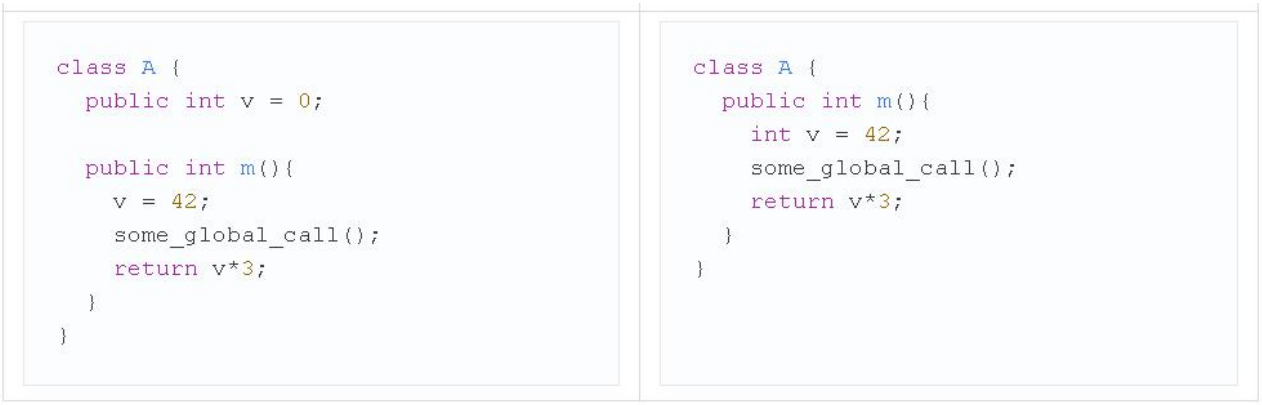
2. 12. Процесс розробки модернізованого методу оптимізації мобільного додатку під керівництвом операційної системи Android

Проаналізувавши та виконавши тестування роботи розробленого додатку під керівництвом операційної системи Android дійшов висновку, що оптимально найрезультативнішим методом оптимізації виявився: оптимізація апаратного прискорення.

Модернізація методу оптимізації апаратного прискорення. Треба відзначити, що цей метод оптимізації мобільних додатків й так є досить ефективним. Тому, його модернізація полягає в оптимізації написанного коду додатку.

Розроблено декілька кроків по оптимізації написання або редагування вже написанного програмного коду додатку.

Перший крок починається з того, що потрібно використовувати локальні змінні замість загальнодоступних полів класу. Таким чином, обмежуючи область видимості змінних, поліпшується читаність коду і зменшується число потенційних помилок, але і робить його більш оптимізованим. Приклад виконання першого кроку оптимізації представлен на рисунку 3.10.



```
class A {
  public int v = 0;

  public int m(){
    v = 42;
    some_global_call();
    return v*3;
  }
}
```

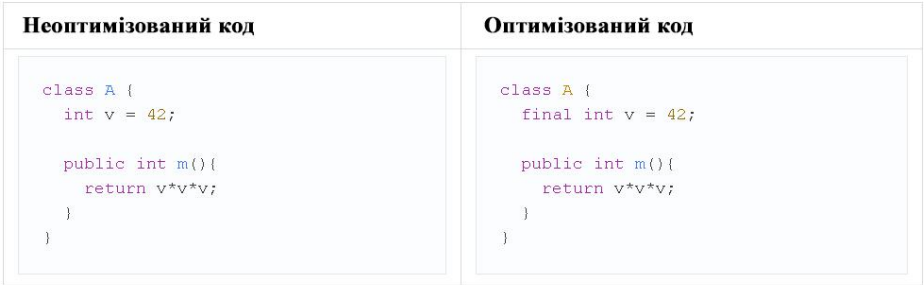
```
class A {
  public int m(){
    int v = 42;
    some_global_call();
    return v*3;
  }
}
```

Рис. 3.10. Представлення різниці між неоптимізованим та оптимізованим програмним кодом при виконанні першого кроку оптимізації

Другий крок полягає у використанні ключового слова final для того, щоб вказати компілятору те, що значення конкретного поля — константа.

Ключове слово final можна використовувати для того, щоб захистити код від випадкової зміни змінних, які повинні бути константами. Однак воно дозволяє поліпшити продуктивність, так як підказує компілятору, що перед ним саме константа.

У фрагменті неоптимізованого коду (рис 3.11), значення v * v * v має обчислюватися під час виконання програми, так як значення v може змінитися. В оптимізованому варіанті використання ключового слова final при оголошенні змінної і присвоєння їй значення, говорить компілятору про те, що значення змінної мінятися не буде. Таким чином, обчислення значення можна зробити на етапі компіляції і в вихідний код буде додано фінального значення, а не команди для його обчислення під час виконання програми.



Неоптимізований код

```
class A {
  int v = 42;

  public int m(){
    return v*v*v;
  }
}
```

Оптимізований код

```
class A {
  final int v = 42;

  public int m(){
    return v*v*v;
  }
}
```

Рис. 3.11. Представлення різниці між неоптимізованим та оптимізованим програмним кодом при виконанні другого кроку оптимізації

Третій крок полягає у використанні ключове слово final при оголошенні класів і методів.

Так як будь-який метод в Java може виявитися поліморфними, оголошення методу або класу з ключовим словом final вказує компілятору на те, що метод не перевизначений ні в одному з підкласів.

У неоптимізованими варіанті коду (рис 3.12) перед викликом функції m () потрібно провести її дозвіл.

У оптимизированном коді (рис 3.12), через використання при оголошенні методу m () ключового слова final, компілятор знає, яка саме версія методу буде викликана. Тому він може уникнути пошуку методу і замінити виклик методу m () його вмістом, вмонтувавши його в потрібне місце програми. В результаті отримуємо збільшення продуктивності.

Неоптимізований код

```
class A {
  public int m(){
    return 42;
  }
  public int f(){
    int sum = 0;
    for (int i = 0; i < 1000; i++)
      sum += m(); // m необходимо разрешить пе
ред вызовом
    return sum;
  }
}
```

Оптимізований код

```
class A {
  public final int m(){
    return 42;
  }
  public int f(){
    int sum = 0;
    for (int i = 0; i < 10
00; i++)
      sum += m();
    return sum;
  }
}
```

Рис. 3.11. Представлення різниці між неоптимізованим та оптимізованим програмним кодом при виконанні третього кроку оптимізації

Четвертий крок містить у собі ціль використовувати стандартні бібліотеки замість реалізації тієї ж функціональності у власному коді.

Стандартні бібліотеки Java серйозно оптимізовані. Якщо використовувати всюди, де це можливо, внутрішні механізми Java, це дозволить досягти

оптимальної роботи. Стандартні рішення можуть працювати значно швидше, ніж «самописние» реалізації. Спроба уникнути додаткового навантаження на систему за рахунок відмови від виклику стандартної бібліотечної функції може, насправді, погіршити продуктивність.

У неоптимізованими варіанті коду показана спроба уникнути виклику стандартної функції Math.abs () за рахунок власної реалізації алгоритму отримання абсолютного значення числа (рис 3.12). Однак, код, в якому викликається бібліотечна функція, працює швидше за рахунок того, що виклик замінюється оптимізованої внутрішньої реалізацією під час компіляції.

Неоптимізований код

```
class A {
  public static final int abs(int a)
{
    int b;
    if (a < 0)
      b = a;
    else
      b = -a;
    return b;
  }
}
```

Оптимізований код

```
class A {
  public static final int abs (int a)
{
    return Math.abs(a);
  }
}
```

Рис. 3.12. Представлення різниці між неоптимізованим та оптимізованим програмним кодом при виконанні четвертого кроку оптимізації

Тестування розроблених вище кроків оптимізації. Для тестування був вибраний розроблений додаток під керівництвом операційної системи Android 6.0. Були зроблені заміри часу від старту до робочої готовності розробленого додатку та його копії. В оригінальному додатку був виконан розроблений метод оптимізації додатку, що складається з 4х кроків. Копія версія розробленого додатку залишилась в оригінальному стані, в якому вона була після закінчення розробки. Представлені результати тестування представленні у таблиці 1.

Таблиця 1 — Порівняння швидкості виконання оптимізаційного методу

№	Оптимізовано, мс	Не оптимізовано, мс
1	25	193
2	21	203
3	30	220
4	25	175
5	23	184
6	28	177
7	30	186
8	27	191
9	34	212
10	27	174
Середнє	27	191.5

В результаті виявилося, що оптимізований додаток виконується, в середньому, в 7 разів швидше неоптимізованого.

Висновок

На прикладі розробки нових методик оптимізації мобільних додатків для операційних систем Android та iOS були продемонстровані наступні моменти:

- В основі створення оптимізаційних методик лежить принцип створення початково оптимізованих додатків;
- Використання розроблених методів оптимізації мобільних додатків зарекомендовало себе, як робочі варіанти які можливо застосовувати у реальних проектах, що знаходяться у розробці;
- На практиці було підтвердженно, що виконання оптимізаційних дії над розробленими додатками під керуванням мобільних операційних систем Android та iOS важче саме на iOS. Це виражається як в закритості самої операційної ситеми, так і в неповних правах доступу до головних елементів операційної системи з точки зору розробника;
- Розроблені методи оптимізації мобільних додатків мають невеликий рівень входження для їх успішного використання розробникам низького та середнього рівня. Таким чином, отримані методи оптимізації мобільних додатків можуть використовуватися великою кількістю стороніх розробників.

ВИСНОВКИ

Ні для кого не секрет, що XXI століття — це століття інформаційних технологій. IT міцно увійшли в наше життя: вони оточують нас всюди. Як правило, людина, натискає педаль газу в своєму автомобілі, може навіть не здогадуватися, що за його рухами стежать сотні датчиків і мікропроцесорів, покликаних полегшити йому життя, а часто і врятувати її. В сучасні машини виробники вбудовують все більш і більш хитромудрі функції. Зважаючи на це з'явилася необхідність зручної взаємодії користувача і всієї навколишньої інфраструктури. З'явилася необхідність у використанні якогось пульта управління електронікою. Інженери Apple, а потім і Google, знайшли рішення. Вони створили операційну систему для телефонів, що дозволяє з легкістю розробляти свої додатки, починаючи від можливості читати електронні книги з екрану мобільних телефонів і закінчуючи управлінням побутовою технікою в «розумному будинку».

Тема розробки мобільних застосувань для платформи Android та iOS є досить цікавою і представляє собою широке поле для дослідження в галузі розробки мобільного програмного забезпечення та методів його оптимізації. Актуальність теми підкреслюється широким спектром можливостей для втілення ідей у вигляді мобільного додатку. В даному розділі було проаналізовані основні інструменти для розробки мобільного програмного забезпечення під платформи Android та iOS. Також було доведено і обгрунтовано правильність вибору засобів для проектування. На основі проведеного аналізу встановлено, що найкращим середовищем розробки є Android Studio та Xcode, а мовами програмування — Java та Swift.

Оптимізація це, з однієї сторони, інструментарій та технологія його використання, що дозволяє навіть розробникам середньої кваліфікації долучитися до виконання оптимізаційних процессів над своїми проектами, а з іншої пошаговий метод вдосконалення фінальних результатів роботи

розробленого мобільного додатку, що отримується тільки через певну кількість часу та спроб.

В останній час, завдяки інтеграції все більшої кількості інструментів для оптимізації мобільних додатків у середи розробки, а також численним функціям та можливостям, які вони предоставляють, цей інструментарій був вибран в якості основи майбутньої розробки.

У час оптимізаційні процеси виконуються вже після того, як мобільний додаток розроблений, а в інших випадках, випущений в реліз. Таким чином, сторонні розробники економлять на процесі розробки мобільних додатків, відсуваючи такий важливий момент у розробці мобільного дадтку, як їх оптимізація на інший раз.

На прикладі розробки нових методик оптимізації мобільних додатків для операційних систем Android та iOS були продемонстровані наступні моменти:

1. В основі створення оптимізаційних методик лежить принцип створення початково оптимізованих додатків;
2. Використання розроблених методів оптимізації мобільних додатків зарекомендовало себе, як робочі варіанти які можливо застосовувати у реальних проектах, що знаходяться у розробці;
3. На практиці було підтвердженно, що виконання оптимізаційних дії над розробленими додатками під керуванням мобільних операційних систем Android та iOS важче саме на iOS. Це виражається як в закритості самої операційної ситеми, так і в неповних правах доступу до головних елементів операційної системи з точки зору розробника;
4. Розроблені методи оптимізації мобільних додатків мають невеликий рівень входження для їх успішного використання розробникам низького та середнього рівня. Таким чином, отримані методи оптимізації мобільних додатків можуть використовуватися великою кількістю стороніх розробників.

В основі створення модифікованих методів оптимізації мобільних додатків полягає у зменщенні кількості тих незручних моментів, з якими може зіткнутися

користувач під час використання придбаного мобільного пристрою. Розгялнуті існуючи методи оптимізації мобільних додатків мають описані вище гідності та недоліки, одна як і будь-який інший результат, що був досягнут, може бути покращений.

Оптимізація часу запуску мобільних додатків під керуванням операційної системи iOS, яка була модифікована шляхом використання ленивого завантаження статичних бібліотек при старті додатку, зображує, що навіть такі невеликі рішення, виконують істотний ефект на загальну картину в цілому. Що зайвий раз показує, те що займатися оптимізацією може практично будь-який розробник в більшості ситуаций та виконувати її для будь-якого додатку, який розробляється або вже розроблений.

ПЕРЕЛІК ПОСИЛАНЬ

1. Меднікс З., Дорнін Л. Програмування під Android [Електрон. ресурс] / Спосіб доступу: URL: http://mikrotik.kpi.ua/index.php/courses-list/android/39-create-your-first-app-for-android. – Загол. з екрана.

2. Офіційний сайт AppCode [Електрон. ресурс] / Спосіб доступу: URL: https://www.jetbrains.com/objc/. – Загол. з екрана.

3. Офіційна сторінка Xcode [Електрон. ресурс] / Спосіб доступу: URL: https://developer.apple.com/xcode/. – Загол. з екрана.

4. Офіційна сайт Stack Overflow [Електрон. ресурс] / Спосіб доступу:URL:http://stackoverflow.com/research/developer-survey-2015#tech-super. – Загол. з екрана.

5. ANON The Swift Programming Language (Swift 2.1) / ANON – Cupertino: Apple Inc., 2014. – 528 p.

6. Офіційна документація мови програмування Swift. [Електрон. ресурс] / Спосіб доступу: URL: www.developer.apple.com/library/documentation/. – Загол. з екрана.

7. Vandad Nahavandipoor iOS 8 Swift Programming Cookbook / Vandad Nahavandipoor – Boston: O'Reilly Media., 2014. – 902 p.

8. Innovations in Science and Technology: the XVI All-Ukrainian R&D Students Conference Proceeding, (Kyiv, April 18, 2016) / National Technical University of Ukraine “Kyiv Polytechnic Institute”. – Part II. – Kyiv, 2016. – 116 p.

9. Яловий Г.К. Економіка та організація виробництва / Яловий Г.К., Пашін В.П., Сичов В.С. – К.: "Політехніка", 2004. – 80 с.

10. Camp O., Filipe J.B.L., Hammoudi S., Piattini M. Enterprise Information Systems V. – Kluwer, 2005. –339 p.

11. Богданюк В.Є. Методичні вказівки до виконання організаційно-економічного розділу дипломних проектів / Богданюк В.Є., Березовський К.В., Пашін В.П. – К.: НТУУ "КПІ", 1999. – 66 с.

12. Lischner R. Delphi in a Nutshell. O'Reilly Media, 2000. – 576 p.

13. Vogel O. Software Architecture: A Comprehensive Framework and Guide for Practitioners. – Springer, 2011. – 550 p.

14. Офіційний сайт Android Studio [Електрон. ресурс] / Спосіб доступу: URL: https://developer.android.com/studio/index.html. – Загол. з екрана.

15. Duggan D. Enterprise Software Architecture and Design: Entities, Services, and Resources. – Wiley, 2012. – 512 p

Evans Eric. Domain–Driven Design: Tackling Complexity in the Heart of Software. – Addison Wesley, 2003. – 560 p.

ДОДАТОК А

Текст програми

Розроблений додаток під керуванням операційної системи iOS.

```
// !$*UTF8*$!
{
	archiveVersion = 1;
	classes = {
	};
	objectVersion = 46;
	objects = {

/* Begin PBXBuildFile section */
		5C80AA7C1AAB5DF300F21FFA /* main.m in Sources */ = {isa = PBXBuildFile; fileRef = 5C80AA7B1AAB5DF300F21FFA /* main.m */; };
		5C80AA7F1AAB5DF300F21FFA /* AppDelegate.m in Sources */ = {isa = PBXBuildFile; fileRef = 5C80AA7E1AAB5DF300F21FFA /* AppDelegate.m */; };
		5C80AA851AAB5DF300F21FFA /* Main.storyboard in Resources */ = {isa = PBXBuildFile; fileRef = 5C80AA831AAB5DF300F21FFA /* Main.storyboard */; };
		5C80AA871AAB5DF300F21FFA /* Images.xcassets in Resources */ = {isa = PBXBuildFile; fileRef = 5C80AA861AAB5DF300F21FFA /* Images.xcassets */; };
		5C80AA8A1AAB5DF300F21FFA /* LaunchScreen.xib in Resources */ = {isa = PBXBuildFile; fileRef = 5C80AA881AAB5DF300F21FFA /* LaunchScreen.xib */; };
		5C80AAA11AAB717100F21FFA /* ARTCRoomViewController.m in Sources */ = {isa = PBXBuildFile; fileRef = 5C80AAA01AAB717100F21FFA /* ARTCRoomViewController.m */; };
		5C80AAA61AAB71C700F21FFA /* ARTCVideoChatViewController.m in Sources */ = {isa = PBXBuildFile; fileRef = 5C80AAA51AAB71C700F21FFA /* ARTCVideoChatViewController.m */; };
		5C80AAAA1AAB788A00F21FFA /* ARTCRoomTextInputViewCell.m in Sources */ = {isa = PBXBuildFile; fileRef = 5C80AAA91AAB788A00F21FFA /* ARTCRoomTextInputViewCell.m */; };
		65B60C96FF584964C27562A6 /* libPods.a in Frameworks */ = {isa = PBXBuildFile; fileRef = EDADA2C0C3A5199BC24EF352 /* libPods.a */; };
/* End PBXBuildFile section */

/* Begin PBXFileReference section */
		5C80AA761AAB5DF300F21FFA /* AppRTC.app */ = {isa = PBXFileReference; explicitFileType = wrapper.application; includeInIndex = 0; path = AppRTC.app; sourceTree = BUILT_PRODUCTS_DIR; };
		5C80AA7A1AAB5DF300F21FFA /* Info.plist */ = {isa = PBXFileReference; lastKnownFileType = text.plist.xml; path = Info.plist; sourceTree = "<group>"; };
		5C80AA7B1AAB5DF300F21FFA /* main.m */ = {isa = PBXFileReference; lastKnownFileType = sourcecode.c.objc; path = main.m; sourceTree = "<group>"; };
		5C80AA7D1AAB5DF300F21FFA /* AppDelegate.h */ = {isa = PBXFileReference; lastKnownFileType = sourcecode.c.h; path = AppDelegate.h; sourceTree = "<group>"; };
		5C80AA7E1AAB5DF300F21FFA /* AppDelegate.m */ = {isa = PBXFileReference; lastKnownFileType = sourcecode.c.objc; path = AppDelegate.m; sourceTree = "<group>"; };
```

```
		5C80AA841AAB5DF300F21FFA /* Base */ = {isa = PBXFileReference; lastKnownFileType = file.storyboard; name = Base; path = Base.lproj/Main.storyboard; sourceTree = "<group>"; };
		5C80AA861AAB5DF300F21FFA /* Images.xcassets */ = {isa = PBXFileReference; lastKnownFileType = folder.assetcatalog; path = Images.xcassets; sourceTree = "<group>"; };
		5C80AA891AAB5DF300F21FFA /* Base */ = {isa = PBXFileReference; lastKnownFileType = file.xib; name = Base; path = Base.lproj/LaunchScreen.xib; sourceTree = "<group>"; };
		5C80AA9F1AAB717100F21FFA /* ARTCRoomViewController.h */ = {isa = PBXFileReference; fileEncoding = 4; lastKnownFileType = sourcecode.c.h; path = ARTCRoomViewController.h; sourceTree = "<group>"; };
		5C80AAA01AAB717100F21FFA /* ARTCRoomViewController.m */ = {isa = PBXFileReference; fileEncoding = 4; lastKnownFileType = sourcecode.c.objc; path = ARTCRoomViewController.m; sourceTree = "<group>"; };
		5C80AAA41AAB71C700F21FFA /* ARTCVideoChatViewController.h */ = {isa = PBXFileReference; fileEncoding = 4; lastKnownFileType = sourcecode.c.h; path = ARTCVideoChatViewController.h; sourceTree = "<group>"; };
		5C80AAA51AAB71C700F21FFA /* ARTCVideoChatViewController.m */ = {isa = PBXFileReference; fileEncoding = 4; lastKnownFileType = sourcecode.c.objc; path = ARTCVideoChatViewController.m; sourceTree = "<group>"; };
		5C80AAA81AAB788A00F21FFA /* ARTCRoomTextInputViewCell.h */ = {isa = PBXFileReference; fileEncoding = 4; lastKnownFileType = sourcecode.c.h; path = ARTCRoomTextInputViewCell.h; sourceTree = "<group>"; };
		5C80AAA91AAB788A00F21FFA /* ARTCRoomTextInputViewCell.m */ = {isa = PBXFileReference; fileEncoding = 4; lastKnownFileType = sourcecode.c.objc; path = ARTCRoomTextInputViewCell.m; sourceTree = "<group>"; };
		A91A157833594B50E74E767F /* Pods.debug.xcconfig */ = {isa = PBXFileReference; includeInIndex = 1; lastKnownFileType = text.xcconfig; name = Pods.debug.xcconfig; path = "Pods/Target Support Files/Pods/Pods.debug.xcconfig"; sourceTree = "<group>"; };
		B6DABCBD6F43FEC6530D36B2 /* Pods.release.xcconfig */ = {isa = PBXFileReference; includeInIndex = 1; lastKnownFileType = text.xcconfig; name = Pods.release.xcconfig; path = "Pods/Target Support Files/Pods/Pods.release.xcconfig"; sourceTree = "<group>"; };
		EDADA2C0C3A5199BC24EF352 /* libPods.a */ = {isa = PBXFileReference; explicitFileType = archive.ar; includeInIndex = 0; path = libPods.a; sourceTree = BUILT_PRODUCTS_DIR; };
/* End PBXFileReference section */

/* Begin PBXFrameworksBuildPhase section */
		5C80AA731AAB5DF300F21FFA /* Frameworks */ = {
			isa = PBXFrameworksBuildPhase;
			buildActionMask = 2147483647;
			files = (
				65B60C96FF584964C27562A6 /* libPods.a in Frameworks */,
			);
			runOnlyForDeploymentPostprocessing = 0;
		};
/* End PBXFrameworksBuildPhase section */

/* Begin PBXGroup section */
```

```
1148D91C97037D5C1CF3115B /* Frameworks */ = {
	isa = PBXGroup;
	children = (
		EDADA2C0C3A5199BC24EF352 /* libPods.a */,
	);
	name = Frameworks;
	sourceTree = "<group>";
};
4DD24E6966F906824AE26A91 /* Pods */ = {
	isa = PBXGroup;
	children = (
		A91A157833594B50E74E767F /* Pods.debug.xcconfig */,
		B6DABCBD6F43FEC6530D36B2 /* Pods.release.xcconfig */,
	);
	name = Pods;
	sourceTree = "<group>";
};
5C80AA6D1AAB5DF300F21FFA = {
	isa = PBXGroup;
	children = (
		5C80AA781AAB5DF300F21FFA /* AppRTC */,
		5C80AA771AAB5DF300F21FFA /* Products */,
		4DD24E6966F906824AE26A91 /* Pods */,
		1148D91C97037D5C1CF3115B /* Frameworks */,
	);
	sourceTree = "<group>";
};
5C80AA771AAB5DF300F21FFA /* Products */ = {
	isa = PBXGroup;
	children = (
		5C80AA761AAB5DF300F21FFA /* AppRTC.app */,
	);
	name = Products;
	sourceTree = "<group>";
};
5C80AA781AAB5DF300F21FFA /* AppRTC */ = {
	isa = PBXGroup;
	children = (
		5C80AAA21AAB718200F21FFA /* App */,
		5C80AAA31AAB718700F21FFA /* ViewControllers */,
		5C80AAA71AAB786200F21FFA /* Views */,
		5C80AA791AAB5DF300F21FFA /* Supporting Files */,
	);
	path = AppRTC;
	sourceTree = "<group>";
};
5C80AA791AAB5DF300F21FFA /* Supporting Files */ = {
	isa = PBXGroup;
	children = (
		5C80AA7A1AAB5DF300F21FFA /* Info.plist */,
		5C80AA7B1AAB5DF300F21FFA /* main.m */,
	);
```

```
                name = "Supporting Files";
                sourceTree = "<group>";
            };
            5C80AAA21AAB718200F21FFA /* App */ = {
                isa = PBXGroup;
                children = (
                    5C80AA7D1AAB5DF300F21FFA /* AppDelegate.h */,
                    5C80AA7E1AAB5DF300F21FFA /* AppDelegate.m */,
                    5C80AA831AAB5DF300F21FFA /* Main.storyboard */,
                    5C80AA861AAB5DF300F21FFA /* Images.xcassets */,
                    5C80AA881AAB5DF300F21FFA /* LaunchScreen.xib */,
                );
                name = App;
                sourceTree = "<group>";
            };
            5C80AAA31AAB718700F21FFA /* ViewControllers */ = {
                isa = PBXGroup;
                children = (
                    5C80AA9F1AAB717100F21FFA /* ARTCRoomViewController.h */,
                    5C80AAA01AAB717100F21FFA /* ARTCRoomViewController.m */,
                    5C80AAA41AAB71C700F21FFA /* ARTCVideoChatViewController.h */,
                    5C80AAA51AAB71C700F21FFA /* ARTCVideoChatViewController.m */,
                );
                name = ViewControllers;
                sourceTree = "<group>";
            };
            5C80AAA71AAB786200F21FFA /* Views */ = {
                isa = PBXGroup;
                children = (
                    5C80AAA81AAB788A00F21FFA /* ARTCRoomTextInputViewCell.h */,
                    5C80AAA91AAB788A00F21FFA /* ARTCRoomTextInputViewCell.m */,
                );
                name = Views;
                sourceTree = "<group>";
            };
/* End PBXGroup section */

/* Begin PBXNativeTarget section */
            5C80AA751AAB5DF300F21FFA /* AppRTC */ = {
                isa = PBXNativeTarget;
                buildConfigurationList = 5C80AA991AAB5DF300F21FFA /* Build configuration list for PBXNativeTarget "AppRTC" */;
                buildPhases = (
                    24ABD54F0D4EF6CABF005175 /* Check Pods Manifest.lock */,
                    5C80AA721AAB5DF300F21FFA /* Sources */,
```

```
                        5C80AA731AAB5DF300F21FFA /* Frameworks */,
                        5C80AA741AAB5DF300F21FFA /* Resources */,
                        50BFB4AA9193C934A869E7DE /* Copy Pods Resources */,
                        9C05033E66DB5BAA7FBF7700 /* Embed Pods Frameworks */,
                    );
                    buildRules = (
                    );
                    dependencies = (
                    );
                    name = AppRTC;
                    productName = AppRTC;
                    productReference = 5C80AA761AAB5DF300F21FFA /* AppRTC.app
*/;
                    productType = "com.apple.product-type.application";
                };
/* End PBXNativeTarget section */

/* Begin PBXProject section */
                5C80AA6E1AAB5DF300F21FFA /* Project object */ = {
                    isa = PBXProject;
                    attributes = {
                        LastUpgradeCheck = 0610;
                        ORGANIZATIONNAME = ISBX;
                        TargetAttributes = {
                            5C80AA751AAB5DF300F21FFA = {
                                CreatedOnToolsVersion = 6.1.1;
                            };
                        };
                    };
                    buildConfigurationList = 5C80AA711AAB5DF300F21FFA /* Build
configuration list for PBXProject "AppRTC" */;
                    compatibilityVersion = "Xcode 3.2";
                    developmentRegion = English;
                    hasScannedForEncodings = 0;
                    knownRegions = (
                        en,
                        Base,
                    );
                    mainGroup = 5C80AA6D1AAB5DF300F21FFA;
                    productRefGroup = 5C80AA771AAB5DF300F21FFA /* Products */;
                    projectDirPath = "";
                    projectRoot = "";
                    targets = (
                        5C80AA751AAB5DF300F21FFA /* AppRTC */,
                    );
                };
/* End PBXProject section */

/* Begin PBXResourcesBuildPhase section */
                5C80AA741AAB5DF300F21FFA /* Resources */ = {
                    isa = PBXResourcesBuildPhase;
                    buildActionMask = 2147483647;
```

```
                files = (
                        5C80AA851AAB5DF300F21FFA /* Main.storyboard in Resources */,
                        5C80AA8A1AAB5DF300F21FFA /* LaunchScreen.xib in Resources */,
                        5C80AA871AAB5DF300F21FFA /* Images.xcassets in Resources */,
                );
                runOnlyForDeploymentPostprocessing = 0;
        };
/* End PBXResourcesBuildPhase section */

/* Begin PBXShellScriptBuildPhase section */
        24ABD54F0D4EF6CABF005175 /* Check Pods Manifest.lock */ = {
                isa = PBXShellScriptBuildPhase;
                buildActionMask = 2147483647;
                files = (
                );
                inputPaths = (
                );
                name = "Check Pods Manifest.lock";
                outputPaths = (
                );
                runOnlyForDeploymentPostprocessing = 0;
                shellPath = /bin/sh;
                shellScript = "diff \"${PODS_ROOT}/../Podfile.lock\" \"${PODS_ROOT}/Manifest.lock\" > /dev/null\nif [[ $? != 0 ]] ; then\n    cat << EOM\nerror: The sandbox is not in sync with the Podfile.lock. Run 'pod install' or update your CocoaPods installation.\nEOM\n    exit 1\nfi\n";
                showEnvVarsInLog = 0;
        };
        50BFB4AA9193C934A869E7DE /* Copy Pods Resources */ = {
                isa = PBXShellScriptBuildPhase;
                buildActionMask = 2147483647;
                files = (
                );
                inputPaths = (
                );
                name = "Copy Pods Resources";
                outputPaths = (
                );
                runOnlyForDeploymentPostprocessing = 0;
                shellPath = /bin/sh;
                shellScript = "\"${SRCROOT}/Pods/Target Support Files/Pods/Pods-resources.sh\"\n";
                showEnvVarsInLog = 0;
        };
        9C05033E66DB5BAA7FBF7700 /* Embed Pods Frameworks */ = {
                isa = PBXShellScriptBuildPhase;
                buildActionMask = 2147483647;
                files = (
                );
```

```
                inputPaths = (
                );
                name = "Embed Pods Frameworks";
                outputPaths = (
                );
                runOnlyForDeploymentPostprocessing = 0;
                shellPath = /bin/sh;
                shellScript = "\"${SRCROOT}/Pods/Target Support Files/Pods/Pods-frameworks.sh\"\n";
                showEnvVarsInLog = 0;
        };
/* End PBXShellScriptBuildPhase section */

/* Begin PBXSourcesBuildPhase section */
        5C80AA721AAB5DF300F21FFA /* Sources */ = {
                isa = PBXSourcesBuildPhase;
                buildActionMask = 2147483647;
                files = (
                        5C80AAAA1AAB788A00F21FFA /* ARTCRoomTextInputViewCell.m in Sources */,
                        5C80AAA61AAB71C700F21FFA /* ARTCVideoChatViewController.m in Sources */,
                        5C80AAA11AAB717100F21FFA /* ARTCRoomViewController.m in Sources */,
                        5C80AA7F1AAB5DF300F21FFA /* AppDelegate.m in Sources */,
                        5C80AA7C1AAB5DF300F21FFA /* main.m in Sources */,
                );
                runOnlyForDeploymentPostprocessing = 0;
        };
/* End PBXSourcesBuildPhase section */

/* Begin PBXVariantGroup section */
        5C80AA831AAB5DF300F21FFA /* Main.storyboard */ = {
                isa = PBXVariantGroup;
                children = (
                        5C80AA841AAB5DF300F21FFA /* Base */,
                );
                name = Main.storyboard;
                sourceTree = "<group>";
        };
        5C80AA881AAB5DF300F21FFA /* LaunchScreen.xib */ = {
                isa = PBXVariantGroup;
                children = (
                        5C80AA891AAB5DF300F21FFA /* Base */,
                );
                name = LaunchScreen.xib;
                sourceTree = "<group>";
        };
/* End PBXVariantGroup section */

/* Begin XCBuildConfiguration section */
```

```
		5C80AA971AAB5DF300F21FFA /* Debug */ = {
			isa = XCBuildConfiguration;
			buildSettings = {
				ALWAYS_SEARCH_USER_PATHS = NO;
				CLANG_CXX_LANGUAGE_STANDARD = "gnu++0x";
				CLANG_CXX_LIBRARY = "libc++";
				CLANG_ENABLE_MODULES = YES;
				CLANG_ENABLE_OBJC_ARC = YES;
				CLANG_WARN_BOOL_CONVERSION = YES;
				CLANG_WARN_CONSTANT_CONVERSION = YES;
				CLANG_WARN_DIRECT_OBJC_ISA_USAGE = YES_ERROR;
				CLANG_WARN_EMPTY_BODY = YES;
				CLANG_WARN_ENUM_CONVERSION = YES;
				CLANG_WARN_INT_CONVERSION = YES;
				CLANG_WARN_OBJC_ROOT_CLASS = YES_ERROR;
				CLANG_WARN_UNREACHABLE_CODE = YES;
				CLANG_WARN__DUPLICATE_METHOD_MATCH = YES;
				"CODE_SIGN_IDENTITY[sdk=iphoneos*]" = "iPhone Developer";
				COPY_PHASE_STRIP = NO;
				ENABLE_BITCODE = NO;
				ENABLE_STRICT_OBJC_MSGSEND = YES;
				GCC_C_LANGUAGE_STANDARD = gnu99;
				GCC_DYNAMIC_NO_PIC = NO;
				GCC_OPTIMIZATION_LEVEL = 0;
				GCC_PREPROCESSOR_DEFINITIONS = (
					"DEBUG=1",
					"$(inherited)",
				);
				GCC_SYMBOLS_PRIVATE_EXTERN = NO;
				GCC_WARN_64_TO_32_BIT_CONVERSION = YES;
				GCC_WARN_ABOUT_RETURN_TYPE = YES_ERROR;
				GCC_WARN_UNDECLARED_SELECTOR = YES;
				GCC_WARN_UNINITIALIZED_AUTOS = YES_AGGRESSIVE;
				GCC_WARN_UNUSED_FUNCTION = YES;
				GCC_WARN_UNUSED_VARIABLE = YES;
				IPHONEOS_DEPLOYMENT_TARGET = 8.1;
				MTL_ENABLE_DEBUG_INFO = YES;
				ONLY_ACTIVE_ARCH = YES;
				SDKROOT = iphoneos;
				TARGETED_DEVICE_FAMILY = "1,2";
			};
			name = Debug;
		};
		5C80AA981AAB5DF300F21FFA /* Release */ = {
			isa = XCBuildConfiguration;
			buildSettings = {
				ALWAYS_SEARCH_USER_PATHS = NO;
				CLANG_CXX_LANGUAGE_STANDARD = "gnu++0x";
				CLANG_CXX_LIBRARY = "libc++";
```

```
                        CLANG_ENABLE_MODULES = YES;
                        CLANG_ENABLE_OBJC_ARC = YES;
                        CLANG_WARN_BOOL_CONVERSION = YES;
                        CLANG_WARN_CONSTANT_CONVERSION = YES;
                        CLANG_WARN_DIRECT_OBJC_ISA_USAGE                    =
YES_ERROR;
                        CLANG_WARN_EMPTY_BODY = YES;
                        CLANG_WARN_ENUM_CONVERSION = YES;
                        CLANG_WARN_INT_CONVERSION = YES;
                        CLANG_WARN_OBJC_ROOT_CLASS = YES_ERROR;
                        CLANG_WARN_UNREACHABLE_CODE = YES;
                        CLANG_WARN__DUPLICATE_METHOD_MATCH = YES;
                        "CODE_SIGN_IDENTITY[sdk=iphoneos*]"          =          "iPhone
Developer";
                        COPY_PHASE_STRIP = YES;
                        ENABLE_BITCODE = NO;
                        ENABLE_NS_ASSERTIONS = NO;
                        ENABLE_STRICT_OBJC_MSGSEND = YES;
                        GCC_C_LANGUAGE_STANDARD = gnu99;
                        GCC_WARN_64_TO_32_BIT_CONVERSION = YES;
                        GCC_WARN_ABOUT_RETURN_TYPE = YES_ERROR;
                        GCC_WARN_UNDECLARED_SELECTOR = YES;
                        GCC_WARN_UNINITIALIZED_AUTOS                        =
YES_AGGRESSIVE;
                        GCC_WARN_UNUSED_FUNCTION = YES;
                        GCC_WARN_UNUSED_VARIABLE = YES;
                        IPHONEOS_DEPLOYMENT_TARGET = 8.1;
                        MTL_ENABLE_DEBUG_INFO = NO;
                        SDKROOT = iphoneos;
                        TARGETED_DEVICE_FAMILY = "1,2";
                        VALIDATE_PRODUCT = YES;
                };
                name = Release;
        };
        5C80AA9A1AAB5DF300F21FFA /* Debug */ = {
                isa = XCBuildConfiguration;
                baseConfigurationReference    =    A91A157833594B50E74E767F    /*
Pods.debug.xcconfig */;
                buildSettings = {
                        ASSETCATALOG_COMPILER_APPICON_NAME = AppIcon;
                        INFOPLIST_FILE = AppRTC/Info.plist;
                        LD_RUNPATH_SEARCH_PATHS          =          "$(inherited)
@executable_path/Frameworks";
                        LIBRARY_SEARCH_PATHS = (
                                "$(inherited)",
                                "$(PROJECT_DIR)/Lib",
                        );
                        PRODUCT_NAME = "$(TARGET_NAME)";
                };
                name = Debug;
        };
        5C80AA9B1AAB5DF300F21FFA /* Release */ = {
```

```
                isa = XCBuildConfiguration;
                baseConfigurationReference = B6DABCBD6F43FEC6530D36B2 /*
Pods.release.xcconfig */;
                buildSettings = {
                        ASSETCATALOG_COMPILER_APPICON_NAME = AppIcon;
                        INFOPLIST_FILE = AppRTC/Info.plist;
                        LD_RUNPATH_SEARCH_PATHS = "$(inherited)
@executable_path/Frameworks";
                        LIBRARY_SEARCH_PATHS = (
                                "$(inherited)",
                                "$(PROJECT_DIR)/Lib",
                        );
                        PRODUCT_NAME = "$(TARGET_NAME)";
                };
                name = Release;
        };
/* End XCBuildConfiguration section */

/* Begin XCConfigurationList section */
        5C80AA711AAB5DF300F21FFA /* Build configuration list for PBXProject
"AppRTC" */ = {
                isa = XCConfigurationList;
                buildConfigurations = (
                        5C80AA971AAB5DF300F21FFA /* Debug */,
                        5C80AA981AAB5DF300F21FFA /* Release */,
                );
                defaultConfigurationIsVisible = 0;
                defaultConfigurationName = Release;
        };
        5C80AA991AAB5DF300F21FFA /* Build configuration list for
PBXNativeTarget "AppRTC" */ = {
                isa = XCConfigurationList;
                buildConfigurations = (
                        5C80AA9A1AAB5DF300F21FFA /* Debug */,
                        5C80AA9B1AAB5DF300F21FFA /* Release */,
                );
                defaultConfigurationIsVisible = 0;
                defaultConfigurationName = Release;
        };
/* End XCConfigurationList section */
    };
    rootObject = 5C80AA6E1AAB5DF300F21FFA /* Project object */;
}
```

Розроблений додаток під керуванням операційної системи Android.

```
package com.udinic.perfdemo.util;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.lang.reflect.Method;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.HashMap;
import java.util.List;
import java.util.Map.Entry;
import java.util.concurrent.CopyOnWriteArrayList;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.locks.ReentrantReadWriteLock;

import android.app.Activity;
import android.content.Context;
import android.content.pm.ApplicationInfo;
import android.os.Build;
import android.text.TextUtils;
import android.util.Log;
import android.view.View;
import android.view.ViewDebug;

/**
 * <p>This class can be used to enable the use of HierarchyViewer inside an
 * application. HierarchyViewer is an Android SDK tool that can be used
 * to inspect and debug the user interface of running applications. For
 * security reasons, HierarchyViewer does not work on production builds
 * (for instance phones bought in store.) By using this class, you can
 * make HierarchyViewer work on any device. You must be very careful
 * however to only enable HierarchyViewer when debugging your
 * application.</p>
 *
 * <p>To use this view server, your application must require the INTERNET
 * permission.</p>
 *
 * <p>The recommended way to use this API is to register activities when
 * they are created, and to unregister them when they get destroyed:</p>
 *
 * <pre>
 * public class MyActivity extends Activity {
 *     public void onCreate(Bundle savedInstanceState) {
 *         super.onCreate(savedInstanceState);
 *         // Set content view, etc.
 *         ViewServer.get(this).addWindow(this);
 *     }
 *
 *     public void onDestroy() {
 *         super.onDestroy();
 *         ViewServer.get(this).removeWindow(this);
 *     }
 *
```

```
*     public void onResume() {
*         super.onResume();
*         ViewServer.get(this).setFocusedWindow(this);
*     }
* }
* </pre>
*
* <p>
* In a similar fashion, you can use this API with an InputMethodService:
* </p>
*
* <pre>
* public class MyInputMethodService extends InputMethodService {
*     public void onCreate() {
*         super.onCreate();
*         View decorView = getWindow().getWindow().getDecorView();
*         String name = "MyInputMethodService";
*         ViewServer.get(this).addWindow(decorView, name);
*     }
*
*     public void onDestroy() {
*         super.onDestroy();
*         View decorView = getWindow().getWindow().getDecorView();
*         ViewServer.get(this).removeWindow(decorView);
*     }
*
*     public void onStartInput(EditorInfo attribute, boolean restarting) {
*         super.onStartInput(attribute, restarting);
*         View decorView = getWindow().getWindow().getDecorView();
*         ViewServer.get(this).setFocusedWindow(decorView);
*     }
* }
* </pre>
*/
public class ViewServer implements Runnable {
    /**
     * The default port used to start view servers.
     */
    private static final int VIEW_SERVER_DEFAULT_PORT = 4939;
    private static final int VIEW_SERVER_MAX_CONNECTIONS = 10;
    private static final String BUILD_TYPE_USER = "user";

    // Debug facility
    private static final String LOG_TAG = "ViewServer";

    private static final String VALUE_PROTOCOL_VERSION = "4";
    private static final String VALUE_SERVER_VERSION = "4";

    // Protocol commands
    // Returns the protocol version
    private static final String COMMAND_PROTOCOL_VERSION = "PROTOCOL";
    // Returns the server version
```

```
    private static final String COMMAND_SERVER_VERSION = "SERVER";
    // Lists all of the available windows in the system
    private static final String COMMAND_WINDOW_MANAGER_LIST = "LIST";
    // Keeps a connection open and notifies when the list of windows changes
    private static final String COMMAND_WINDOW_MANAGER_AUTOLIST =
"AUTOLIST";
    // Returns the focused window
    private static final String COMMAND_WINDOW_MANAGER_GET_FOCUS =
"GET_FOCUS";

    private ServerSocket mServer;
    private final int mPort;

    private Thread mThread;
    private ExecutorService mThreadPool;

    private final List<WindowListener> mListeners =
        new CopyOnWriteArrayList<ViewServer.WindowListener>();

    private final HashMap<View, String> mWindows = new HashMap<View, String>();
    private final ReentrantReadWriteLock mWindowsLock = new ReentrantReadWriteLock();

    private View mFocusedWindow;
    private final ReentrantReadWriteLock mFocusLock = new ReentrantReadWriteLock();

    private static ViewServer sServer;

    /**
     * Returns a unique instance of the ViewServer. This method should only be
     * called from the main thread of your application. The server will have
     * the same lifetime as your process.
     *
     * If your application does not have the <code>android:debuggable</code>
     * flag set in its manifest, the server returned by this method will
     * be a dummy object that does not do anything. This allows you to use
     * the same code in debug and release versions of your application.
     *
     * @param context A Context used to check whether the application is
     *                debuggable, this can be the application context
     */
    public static ViewServer get(Context context) {
        ApplicationInfo info = context.getApplicationInfo();
        if (BUILD_TYPE_USER.equals(Build.TYPE) &&
            (info.flags & ApplicationInfo.FLAG_DEBUGGABLE) != 0) {
            if (sServer == null) {
                sServer = new ViewServer(ViewServer.VIEW_SERVER_DEFAULT_PORT);
            }

            if (!sServer.isRunning()) {
                try {
                    sServer.start();
                } catch (IOException e) {
```

```
                Log.d(LOG_TAG, "Error:", e);
            }
        }
    } else {
        sServer = new NoopViewServer();
    }

    return sServer;
}

private ViewServer() {
    mPort = -1;
}

/**
 * Creates a new ViewServer associated with the specified window manager on the
 * specified local port. The server is not started by default.
 *
 * @param port The port for the server to listen to.
 *
 * @see #start()
 */
private ViewServer(int port) {
    mPort = port;
}

/**
 * Starts the server.
 *
 * @return True if the server was successfully created, or false if it already exists.
 * @throws IOException If the server cannot be created.
 *
 * @see #stop()
 * @see #isRunning()
 * @see WindowManagerService#startViewServer(int)
 */
public boolean start() throws IOException {
    if (mThread != null) {
        return false;
    }

    mThread = new Thread(this, "Local View Server [port=" + mPort + "]");
    mThreadPool                                                                    =
Executors.newFixedThreadPool(VIEW_SERVER_MAX_CONNECTIONS);
    mThread.start();

    return true;
}

/**
 * Stops the server.
 *
```

```
     * @return True if the server was stopped, false if an error occurred or if the
     *         server wasn't started.
     *
     * @see #start()
     * @see #isRunning()
     * @see WindowManagerService#stopViewServer()
     */
    public boolean stop() {
        if (mThread != null) {
            mThread.interrupt();
            if (mThreadPool != null) {
                try {
                    mThreadPool.shutdownNow();
                } catch (SecurityException e) {
                    Log.w(LOG_TAG, "Could not stop all view server threads");
                }
            }

            mThreadPool = null;
            mThread = null;

            try {
                mServer.close();
                mServer = null;
                return true;
            } catch (IOException e) {
                Log.w(LOG_TAG, "Could not close the view server");
            }
        }

        mWindowsLock.writeLock().lock();
        try {
            mWindows.clear();
        } finally {
            mWindowsLock.writeLock().unlock();
        }

        mFocusLock.writeLock().lock();
        try {
            mFocusedWindow = null;
        } finally {
            mFocusLock.writeLock().unlock();
        }

        return false;
    }

    /**
     * Indicates whether the server is currently running.
     *
     * @return True if the server is running, false otherwise.
     *
```

```
     * @see #start()
     * @see #stop()
     * @see WindowManagerService#isViewServerRunning()
     */
    public boolean isRunning() {
        return mThread != null && mThread.isAlive();
    }

    /**
     * Invoke this method to register a new view hierarchy.
     *
     * @param activity The activity whose view hierarchy/window to register
     *
     * @see #addWindow(View, String)
     * @see #removeWindow(Activity)
     */
    public void addWindow(Activity activity) {
        String name = activity.getTitle().toString();
        if (TextUtils.isEmpty(name)) {
            name = activity.getClass().getCanonicalName() +
                    "/0x" + System.identityHashCode(activity);
        } else {
            name += "(" + activity.getClass().getCanonicalName() + ")";
        }
        addWindow(activity.getWindow().getDecorView(), name);
    }

    /**
     * Invoke this method to unregister a view hierarchy.
     *
     * @param activity The activity whose view hierarchy/window to unregister
     *
     * @see #addWindow(Activity)
     * @see #removeWindow(View)
     */
    public void removeWindow(Activity activity) {
        removeWindow(activity.getWindow().getDecorView());
    }

    /**
     * Invoke this method to register a new view hierarchy.
     *
     * @param view A view that belongs to the view hierarchy/window to register
     * @name name The name of the view hierarchy/window to register
     *
     * @see #removeWindow(View)
     */
    public void addWindow(View view, String name) {
        mWindowsLock.writeLock().lock();
        try {
            mWindows.put(view.getRootView(), name);
        } finally {
```

```
        mWindowsLock.writeLock().unlock();
    }
    fireWindowsChangedEvent();
}

/**
 * Invoke this method to unregister a view hierarchy.
 *
 * @param view A view that belongs to the view hierarchy/window to unregister
 *
 * @see #addWindow(View, String)
 */
public void removeWindow(View view) {
    View rootView;
    mWindowsLock.writeLock().lock();
    try {
        rootView = view.getRootView();
        mWindows.remove(rootView);
    } finally {
        mWindowsLock.writeLock().unlock();
    }
    mFocusLock.writeLock().lock();
    try {
        if (mFocusedWindow == rootView) {
            mFocusedWindow = null;
        }
    } finally {
        mFocusLock.writeLock().unlock();
    }
    fireWindowsChangedEvent();
}

/**
 * Invoke this method to change the currently focused window.
 *
 * @param activity The activity whose view hierarchy/window hasfocus,
 *                 or null to remove focus
 */
public void setFocusedWindow(Activity activity) {
    setFocusedWindow(activity.getWindow().getDecorView());
}

/**
 * Invoke this method to change the currently focused window.
 *
 * @param view A view that belongs to the view hierarchy/window that has focus,
 *             or null to remove focus
 */
public void setFocusedWindow(View view) {
    mFocusLock.writeLock().lock();
    try {
        mFocusedWindow = view == null ? null : view.getRootView();
```

```
        } finally {
            mFocusLock.writeLock().unlock();
        }
        fireFocusChangedEvent();
    }

    /**
     * Main server loop.
     */
    public void run() {
        try {
            mServer = new ServerSocket(mPort, VIEW_SERVER_MAX_CONNECTIONS,
InetAddress.getLocalHost());
        } catch (Exception e) {
            Log.w(LOG_TAG, "Starting ServerSocket error: ", e);
        }

        while (mServer != null && Thread.currentThread() == mThread) {
            // Any uncaught exception will crash the system process
            try {
                Socket client = mServer.accept();
                if (mThreadPool != null) {
                    mThreadPool.submit(new ViewServerWorker(client));
                } else {
                    try {
                        client.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            } catch (Exception e) {
                Log.w(LOG_TAG, "Connection error: ", e);
            }
        }
    }

    private static boolean writeValue(Socket client, String value) {
        boolean result;
        BufferedWriter out = null;
        try {
            OutputStream clientStream = client.getOutputStream();
            out = new BufferedWriter(new OutputStreamWriter(clientStream), 8 * 1024);
            out.write(value);
            out.write("\n");
            out.flush();
            result = true;
        } catch (Exception e) {
            result = false;
        } finally {
            if (out != null) {
                try {
                    out.close();
```

```
            } catch (IOException e) {
                result = false;
            }
        }
    }
    return result;
}

private void fireWindowsChangedEvent() {
    for (WindowListener listener : mListeners) {
        listener.windowsChanged();
    }
}

private void fireFocusChangedEvent() {
    for (WindowListener listener : mListeners) {
        listener.focusChanged();
    }
}

private void addWindowListener(WindowListener listener) {
    if (!mListeners.contains(listener)) {
        mListeners.add(listener);
    }
}

private void removeWindowListener(WindowListener listener) {
    mListeners.remove(listener);
}

private interface WindowListener {
    void windowsChanged();
    void focusChanged();
}

private static class UncloseableOutputStream extends OutputStream {
    private final OutputStream mStream;

    UncloseableOutputStream(OutputStream stream) {
        mStream = stream;
    }

    public void close() throws IOException {
        // Don't close the stream
    }

    public boolean equals(Object o) {
        return mStream.equals(o);
    }

    public void flush() throws IOException {
        mStream.flush();
```

```
    }

    public int hashCode() {
        return mStream.hashCode();
    }

    public String toString() {
        return mStream.toString();
    }

    public void write(byte[] buffer, int offset, int count)
        throws IOException {
        mStream.write(buffer, offset, count);
    }

    public void write(byte[] buffer) throws IOException {
        mStream.write(buffer);
    }

    public void write(int oneByte) throws IOException {
        mStream.write(oneByte);
    }
}

private static class NoopViewServer extends ViewServer {
    private NoopViewServer() {
    }

    @Override
    public boolean start() throws IOException {
        return false;
    }

    @Override
    public boolean stop() {
        return false;
    }

    @Override
    public boolean isRunning() {
        return false;
    }

    @Override
    public void addWindow(Activity activity) {
    }

    @Override
    public void removeWindow(Activity activity) {
    }

    @Override
```

```
    public void addWindow(View view, String name) {
    }

    @Override
    public void removeWindow(View view) {
    }

    @Override
    public void setFocusedWindow(Activity activity) {
    }

    @Override
    public void setFocusedWindow(View view) {
    }

    @Override
    public void run() {
    }
}

private class ViewServerWorker implements Runnable, WindowListener {
    private Socket mClient;
    private boolean mNeedWindowListUpdate;
    private boolean mNeedFocusedWindowUpdate;

    private final Object[] mLock = new Object[0];

    public ViewServerWorker(Socket client) {
        mClient = client;
        mNeedWindowListUpdate = false;
        mNeedFocusedWindowUpdate = false;
    }

    public void run() {
        BufferedReader in = null;
        try {
            in = new BufferedReader(new InputStreamReader(mClient.getInputStream()), 1024);

            final String request = in.readLine();

            String command;
            String parameters;

            int index = request.indexOf(' ');
            if (index == -1) {
                command = request;
                parameters = "";
            } else {
                command = request.substring(0, index);
                parameters = request.substring(index + 1);
            }
```

```
            boolean result;
            if (COMMAND_PROTOCOL_VERSION.equalsIgnoreCase(command)) {
               result = writeValue(mClient, VALUE_PROTOCOL_VERSION);
            } else if (COMMAND_SERVER_VERSION.equalsIgnoreCase(command)) {
               result = writeValue(mClient, VALUE_SERVER_VERSION);
            } else if (COMMAND_WINDOW_MANAGER_LIST.equalsIgnoreCase(command))
{
               result = listWindows(mClient);
            }                                                    else                                                    if
(COMMAND_WINDOW_MANAGER_GET_FOCUS.equalsIgnoreCase(command)) {
               result = getFocusedWindow(mClient);
            }                                                    else                                                    if
(COMMAND_WINDOW_MANAGER_AUTOLIST.equalsIgnoreCase(command)) {
               result = windowManagerAutolistLoop();
            } else {
               result = windowCommand(mClient, command, parameters);
            }

            if (!result) {
               Log.w(LOG_TAG, "An error occurred with the command: " + command);
            }
         } catch(IOException e) {
            Log.w(LOG_TAG, "Connection error: ", e);
         } finally {
            if (in != null) {
               try {
                  in.close();

               } catch (IOException e) {
                  e.printStackTrace();
               }
            }
            if (mClient != null) {
               try {
                  mClient.close();
               } catch (IOException e) {
                  e.printStackTrace();
               }
            }
         }
      }

      private boolean windowCommand(Socket client, String command, String parameters) {
         boolean success = true;
         BufferedWriter out = null;

         try {
            // Find the hash code of the window
            int index = parameters.indexOf(' ');
            if (index == -1) {
               index = parameters.length();
            }
```

```
            final String code = parameters.substring(0, index);
            int hashCode = (int) Long.parseLong(code, 16);

            // Extract the command's parameter after the window description
            if (index < parameters.length()) {
                parameters = parameters.substring(index + 1);
            } else {
                parameters = "";
            }

            final View window = findWindow(hashCode);
            if (window == null) {
                return false;
            }

            // call stuff
            final Method dispatch = ViewDebug.class.getDeclaredMethod("dispatchCommand",
                    View.class, String.class, String.class, OutputStream.class);
            dispatch.setAccessible(true);
            dispatch.invoke(null, window, command, parameters,
                    new UncloseableOutputStream(client.getOutputStream()));

            if (!client.isOutputShutdown()) {
                out = new BufferedWriter(new OutputStreamWriter(client.getOutputStream()));
                out.write("DONE\n");
                out.flush();
            }

        } catch (Exception e) {
            Log.w(LOG_TAG, "Could not send command " + command +
                    " with parameters " + parameters, e);
            success = false;
        } finally {
            if (out != null) {
                try {
                    out.close();
                } catch (IOException e) {
                    success = false;
                }
            }
        }

        return success;
    }

    private View findWindow(int hashCode) {
        if (hashCode == -1) {
            View window = null;
            mWindowsLock.readLock().lock();
            try {
                window = mFocusedWindow;
            } finally {
```

```
            mWindowsLock.readLock().unlock();
        }
        return window;
    }


    mWindowsLock.readLock().lock();
    try {
        for (Entry<View, String> entry : mWindows.entrySet()) {
            if (System.identityHashCode(entry.getKey()) == hashCode) {
                return entry.getKey();
            }
        }
    } finally {
        mWindowsLock.readLock().unlock();
    }

    return null;
}

private boolean listWindows(Socket client) {
    boolean result = true;
    BufferedWriter out = null;

    try {
        mWindowsLock.readLock().lock();

        OutputStream clientStream = client.getOutputStream();
        out = new BufferedWriter(new OutputStreamWriter(clientStream), 8 * 1024);

        for (Entry<View, String> entry : mWindows.entrySet()) {
            out.write(Integer.toHexString(System.identityHashCode(entry.getKey())));
            out.write(' ');
            out.append(entry.getValue());
            out.write('\n');
        }

        out.write("DONE.\n");
        out.flush();
    } catch (Exception e) {
        result = false;
    } finally {
        mWindowsLock.readLock().unlock();

        if (out != null) {
            try {
                out.close();
            } catch (IOException e) {
                result = false;
            }
        }
    }
```

```
    return result;
}

private boolean getFocusedWindow(Socket client) {
    boolean result = true;
    String focusName = null;

    BufferedWriter out = null;
    try {
        OutputStream clientStream = client.getOutputStream();
        out = new BufferedWriter(new OutputStreamWriter(clientStream), 8 * 1024);

        View focusedWindow = null;

        mFocusLock.readLock().lock();
        try {
            focusedWindow = mFocusedWindow;
        } finally {
            mFocusLock.readLock().unlock();
        }

        if (focusedWindow != null) {
            mWindowsLock.readLock().lock();
            try {
                focusName = mWindows.get(mFocusedWindow);
            } finally {
                mWindowsLock.readLock().unlock();
            }

            out.write(Integer.toHexString(System.identityHashCode(focusedWindow)));
            out.write(' ');
            out.append(focusName);
        }
        out.write('\n');
        out.flush();
    } catch (Exception e) {
        result = false;
    } finally {
        if (out != null) {
            try {
                out.close();
            } catch (IOException e) {
                result = false;
            }
        }
    }

    return result;
}

public void windowsChanged() {
```

```
    synchronized (mLock) {
        mNeedWindowListUpdate = true;
        mLock.notifyAll();
    }
}

public void focusChanged() {
    synchronized (mLock) {
        mNeedFocusedWindowUpdate = true;
        mLock.notifyAll();
    }
}

private boolean windowManagerAutolistLoop() {
    addWindowListener(this);
    BufferedWriter out = null;
    try {
        out = new BufferedWriter(new OutputStreamWriter(mClient.getOutputStream()));
        while (!Thread.interrupted()) {
            boolean needWindowListUpdate = false;
            boolean needFocusedWindowUpdate = false;
            synchronized (mLock) {
                while (!mNeedWindowListUpdate && !mNeedFocusedWindowUpdate) {
                    mLock.wait();
                }
                if (mNeedWindowListUpdate) {
                    mNeedWindowListUpdate = false;
                    needWindowListUpdate = true;
                }
                if (mNeedFocusedWindowUpdate) {
                    mNeedFocusedWindowUpdate = false;
                    needFocusedWindowUpdate = true;
                }
            }
            if (needWindowListUpdate) {
                out.write("LIST UPDATE\n");
                out.flush();
            }
            if (needFocusedWindowUpdate) {
                out.write("FOCUS UPDATE\n");
                out.flush();
            }
        }
    } catch (Exception e) {
        Log.w(LOG_TAG, "Connection error: ", e);
    } finally {
        if (out != null) {
            try {
                out.close();
            } catch (IOException e) {
                // Ignore
            }
```

```
            }
            removeWindowListener(this);
        }
        return true;
    }
  }
}
```

ДОДАТОК Б

ВІДГУК

на дипломну роботу магістра на тему:

«Дослідження механізмів управління ресурсами в операційних системах Android та iOS для розробки мобільних додатків»

студента групи 121м-16-1 Козира Артема Володимировича

1. Ціль дипломної роботи магістра удосконалення методики розробки початково оптимізованих мобільних додатків.
2. Актуальність даної теми зумовлена відсутністю актуального методу розробки оптимізованих мобільних додатків, що мають оптимальну оптимізацію ще на етапі розробки мобільного додатку.
3. Тема дипломної роботи безпосередньо зв’язана з об’єктом діяльності магістра спеціальності 8.05010301 «Програмне забезпечення систем» направлення 6.050103 «Програмна інженерія» – створення, дослідження і реалізація моделей та програмних засобів.
4. Наукова новизна результатів, які очікуються, полягає у створенні нового архітектурного підходу до методів виконання оптимізації мобільних додатків та реалізація його у вигляді розробленого мобільного додатку для операційних систем Android та iOS.
5. Оригінальність технічних рішень при розробці програмного засобу полягають у створенні комбінованої методики виконання оптимізації мобільного додатку на мовах програмування Swift та Java, а також явної демонстрації різних варіантів оптимізації на двох різних операційних системах – Android та iOS.
6. Практична цінність дослідження полягає в розробці комплексного методу оптимізації мобільних додатків з врахуванням особливостей мобільних операційних систем Android та iOS.
7. Оформлення дипломної роботи магістра виконано на сучасному рівні та відповідає вимогам, які пред'являються до робіт даної кваліфікації. Ступінь самостійності виконання достатньо висока.
8. Дипломна робота магістра в цілому заслуговує оцінки «відмінно», а сам автор – присвоєння кваліфікації «інженер-програміст».

Керівник дипломної
роботи магістра, д.т.н.,
проф. кафедри ПЗКС В.М. Куваєв

ДОДАТОК В

РЕЦЕНЗІЯ

на дипломну роботу магістра на тему:

«Дослідження механізмів управління ресурсами в операційних системах Android та iOS для розробки мобільних додатків»

студента групи 121м-16-1 Козира Артема Володимировича

В даний час існує досить велика кількість користувачів телефонів та додатків до них під керівництвом операційних систем Android та iOS. З плином часу, деякі смартфони за своїми характеристиками вже не поступаються стаціонарним комп'ютерам, про що раньше можливо було тільки мріяти. Однак, така ситуація дозволила більшості розробників мобільних додатків орієнтуватися на найпотужниші смартфони, які є на ринку. При цьому, роль оптимізації все частіше почав відходити на другий або третій план. У цей час, процент таких смартфонів, що знаходиться у використанні користувачів невеликий.

Головна причина цього явища – відсутність єдиного механізму забезпечення оптимізації під певну операційну систему. Керівникам великих проектів спочатку доводиться витрачати значні матеріальні засоби для придбання і розробки мобільних додатків, їх супроводу, публікації в засобах поширення і т. д.

Наукова новизна складається в створенні нового методу до комплексної оптимізації мобільних додатків для різних типів мобільниз операційних систем та реалізації його у вигляді двох додатків, що наочно демонструють різницю у архітектурах двох операційних систем − Android та iOS.

Студент Козир А.В. достатньо добре розібрався в специфіці використання різноманітних інформаційних технологій: Xcode, Swift, Object-C, Java, Calabash та Systrace.

Використовувані технології розробки мобільних додатків безпосередньо пов'язані з об'єктом діяльності магістра напряму 6.050103 «Програмна інженерія».

Беручи до уваги вище викладене, можна зробити висновок, що дана робота цілком відповідає вимогам, що пред'являються до кваліфікаційних робіт рівня магістра.

З огляду на наукову новизну і ступінь опрацювання компонентів даної роботи, в цілому автор заслуговує оцінки «відмінно», а також присвоєння кваліфікації «інженер-програміст».

Професор кафедри
Автоматизації технологічних процесів
НМетАУ, д.т.н. В.І Головко