Київський національний торговельно-економічний університет
Кафедра програмної інженерії та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА (ПРОЕКТ)

на тему:

“Розробка мобільного додатку визначення геопозиції клієнтів логістичної компанії”

Студента 2 курсу, 5 групи,
Спеціальності
121 «Інженерія програмного забезпечення»
Спеціалізації
«Інженерія програмного забезпечення»

Вікторов Владислав Віталійович

_____
підпис студента

Науковий керівник
Доктор технічних наук
Професор

Криворучко Олена Володимирівна

_____
підпис керівника

Гарант освітньої програми
Доктор технічних наук
Професор

Криворучко Олена Володимирівна

_____
підпис керівника

КИЇВ – 2019

ЗМІСТ

ВСТУП

Ринок мобільних додатків майже щодня поповнюється новими розробками. Корисні додатки мають великий попит і це дозволяє їхнім розробникам заробити. Для розробки мобільних додатків для iOS використовуються так звані нативні мови програмування. Для iOS це Swift. Ця мова доволі зручна, і має широкий спектр механізмів, патернів, методів і бібліотек для написання додатків. Звичайно швидкість написання додатків залежить від його складності.

Цільова платформа (iOS,) має значний вплив на мову розробки, яка буде використовуватися. Наприклад, можна розробляти рідні додатки для платформи або використовувати сторонній інструмент для оптимізації 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:

- витрат на живлення акумуляторної батареї мобільного пристрою;
- обмеження на кількість даних, що передаються через мережу Інтернет;
- зменшення швидкості передачі пакетів в мобільному Інтернет з'єднанні;
- можливостей втрати пакетів при передачі в мобільному Інтернет з'єднанні;
- безпеки даних користувача;
- значної фрагментації версій операційних систем та фреймворків;
- великої різноманітність розмірів екранів мобільних пристроїв;
- обмеженя запитів до системи визначення місцезнаходження абонента;
- обмеження на розмір файлу додатку;
- порівняно невеликого ліміту оперативної пам'яті мобільного пристрою;

- порівняно обмеженої швидкість передачі даних в мобільному Інтернет з'єднанні.

Дана тема є **актуальною,** тому що розробка мобільних додатків відіграє все більш важливу роль для організацій, яким необхідно спілкуватися зі співробітниками або клієнтами за допомогою вбудованих додатків. 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). Сьогодні фахівцями в області інформаційних технологій розробляються мобільні додатки, які дозволяють вирішувати безліч завдань, наприклад, створення 3D-анімації. Деякі служать для того щоб встановлювати з'єднання з мережею. Інші допомагають оптимізувати маршрут. Треті призначені для тих, хто шукає найвигідніші магазини. Є й такі, за допомогою яких можна замовити їжу додому. В основу кожної з таких програм лягли певні утиліти, що в результаті дозволяє швидко вирішувати поставлену задачу, економити час і досягати максимально комфортного рівня життя.

Всі мобільні додатки умовно можна поділити на програми для робочих цілей і на розважальні програми. Перші дозволяють бізнесменам і офісним працівникам контролювати бізнес-процеси, складати аналітичну звітність, виконувати такі завдання, як розробка дизайну фірмового стилю.

Протягом останніх років показник, що характеризує рівень попиту на мобільні пристрої, постійно зростає. Така статистика дозволяє зробити висновок про те, що розробка мобільних додатків актуальна і доцільна. Отже, тільки корисна розробка отримає гідне визнання з боку користувачів. Не винятком є предметна область власного стану здоров’я людини, і відповідно визначено об’єкт та предмет дослідження.

Об’єкт дослідження: геопозиція клієнтів логістичної компанії.

Предмет дослідження: процесс створення мобільного додатку за допомогою мови програмування Swift.

Метою випускної кваліфікаційної роботи (проекту) є розробка мобільних застосувань для операційної системи iOS за допомогою мови програмування Swift.

Відповідно до мети роботи поставлена низка задач:

- обгрунтування вибору мови програмування та середовища розробки;
- дослідження існуючих технологій та рішень для створення мобільних застосувань для ОС iOS;
- обгрунтування вибору певної технології, використовуючи iOS SDK;
- аналіз розробленого програмного продукту;
- оцінка використаних та досліджених підходів;
- визначення переваг та недоліків створеного програмного продукту;

Метод і технології розробки: В роботі використовуються сучасні методи об'єктно орієнтованого програмування, аналіз операційної системи iOS, програмування мовою Swift, розробка інтерфейсу користувача, використовуючі сучасні фреймворки та методи розробки

Наукова новизна отриманих результатів полягає в проведенні аналізу та виявленні недоліків традиційного підходу до розробки мобільних додатків для операційної iOS, а також у створенні мобільного додатку для визначення геопозиції клієнтів логістичної компанії

Результат роботи: створення мобільного додатку з інтуітивно зрозумілим інтерфейсом засобами IDE Xcone та фремворкуUiKit для зручного використання користувачем.

РОЗДІЛ 1.
ТЕХНОЛОГІЇ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ ДЛЯ МОБІЛЬНОЇ ОПЕРАЦІЙНОЇ СИСТЕМИ IOS

1.1 Мобільні додатки та операційна система iOS

Мобільний додаток – це програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях. Багато мобільних додатків вже установлені на самому пристрої або можуть бути завантажені на нього з додатків онлайн-магазинів, таких як App Store, BlackBerry App World та інших.

Раніше мобільні додатки використовувалися для швидкої перевірки електронної пошти, але їх високий попит призвів до розширення їхніх призначень і в інших областях, таких як ігри для мобільних телефонів, GPS, спілкування, перегляд відео та користування інтернетом. iOS (відома як iPhone OS до червня 2010 року) — це власницька мобільна операційна система від Apple. Розроблена спочатку для iPhone, вона стала операційною системою також для iPod Touch, iPad і Apple TV. Apple не дозволяє роботу ОС на мобільних телефонах інших фірм. iOS є похідною від OS X, отже, є за своєю природою Unix-подібною операційною системою.

Користувацький інтерфейс iOS заснований на концепції прямої маніпуляції з використанням жестів Multi-Touch. Елементи інтерфейсу управління складаються з повзунків, перемикачів і кнопок. Він призначений для безпосереднього контакту користувача з екраном пристрою. Внутрішній акселерометр використовуються деякими програмами для реагування на

струшування пристрою, яке є також загальною командою скасування, або обертання пристрою у трьох вимірах, що є загальною командою перемикання між книжковим та альбомним режимами.

1.2 Специфікації та особливостіі 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. 1.3).

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.

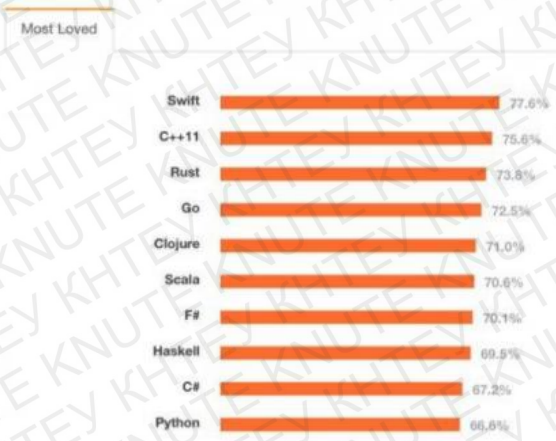

Рис. 1.1. Оцінки мов програмування

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.

1.3 Архітектура мобільної операційної системи 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. 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.

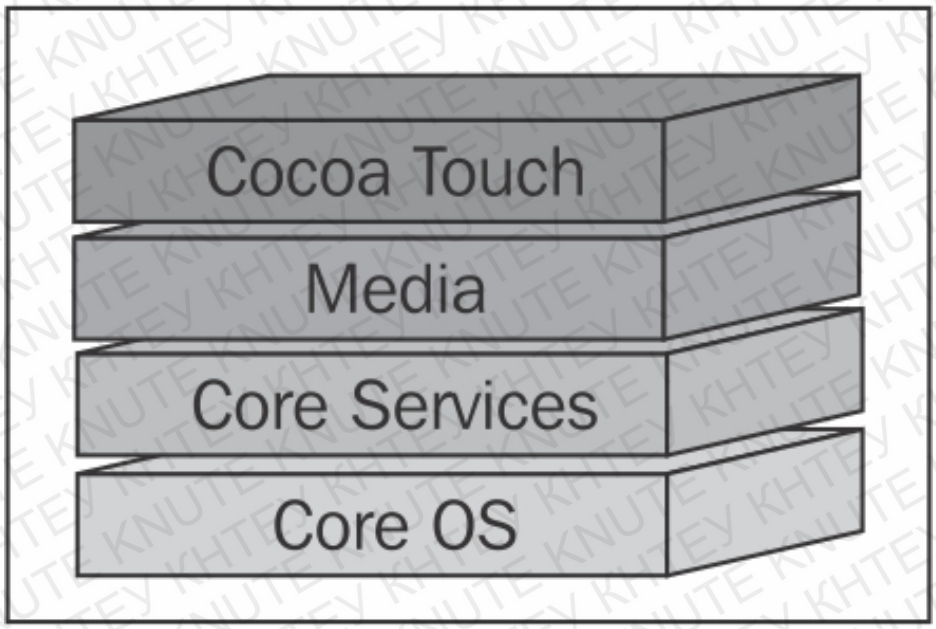

Рис. 1.2. Архітектура платформи iOS

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. 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.

Таблиця 1.1.

ОСНОВНІ ПЕРЕВАГИ ТА НЕДОЛІКИ ОПЕРАЦІЙНОЇ СИСТЕМИ IOS

Основні переваги iOS	**Основні недоліки iOS**
простота роботи	прив’язка до апаратного забезпечення одного виробника
висока стабільність	загальна «закритість» системи
безпечність	
відсутність вірусів	
відсутність фрагментованості	
регулярне оновлення	
зручність розробки додатків	
широкий вибір програм та ігор	

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

```
class Person {
    var name: String
    var age: Int

    init(name: String, age: Int) {
        self.name = name
        self.age = age
    }

    func getName() -> String {
        return "Your name is \(name)"
    }

}
```

Рис 1.3 Клас Person

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"

1.4 Опис основних інструментів розробки мобільного додатку для платформи 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 .

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.

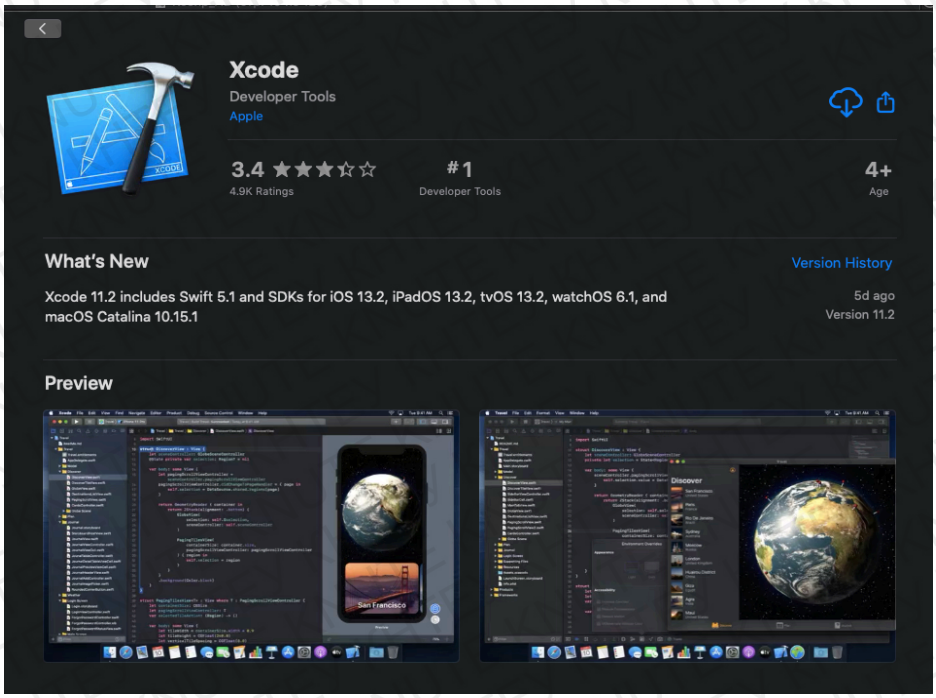

Рис. 1.3. Екран перегляду Xcode у офіційному сервісі поширення програмного забезпечення для операційних систем iOS та macOS

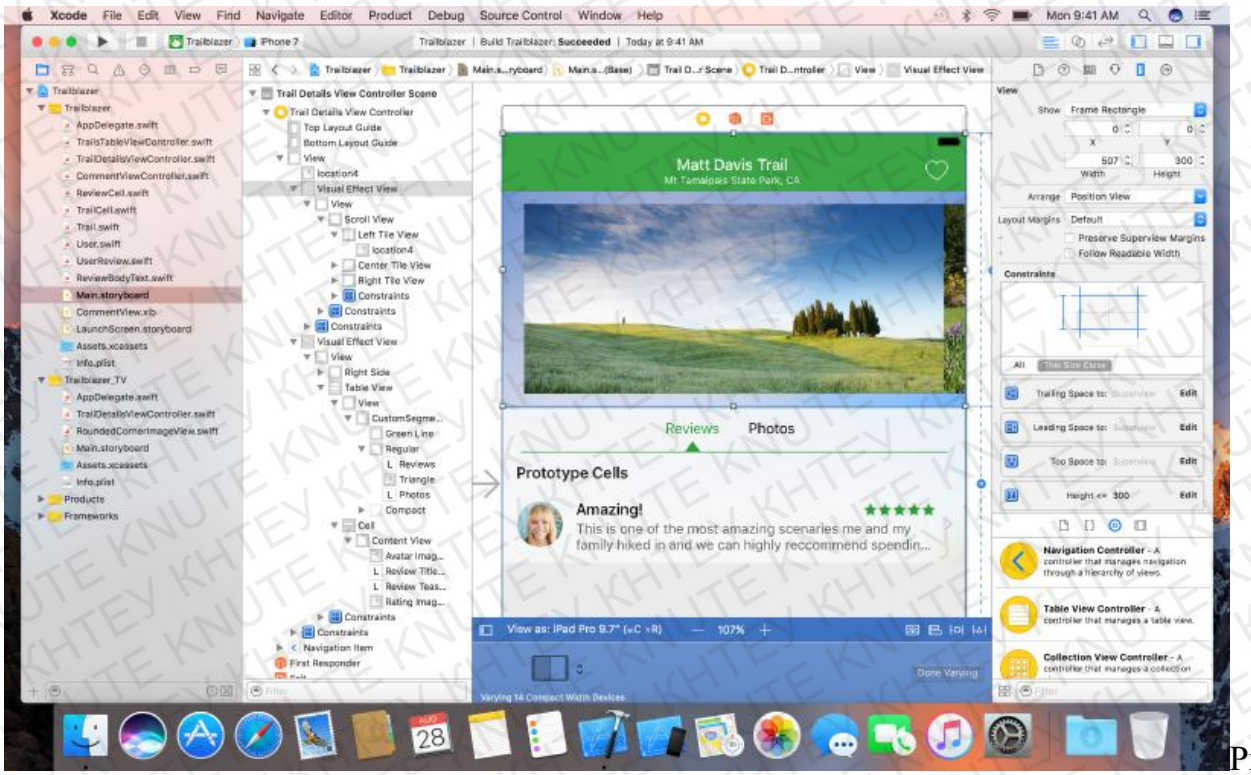

Рис. 1.4. Головний екран Xcode IDE 9.0

Середовище розробки Xcode - це пакет інструментів для розробки додатків під Mac OS X і iPhone OS, розроблений Apple. Остання версія Xcode11.2, . 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 в Xcode IDE.

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 можливості, 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 Apple Script для автоматизації процесу складання;
- підтримка налагоджувальної інформації в форматах DWARF і Stabs (налагоджувальна інформація для всіх проектів за замовчуванням генерується в форматі DWARF).

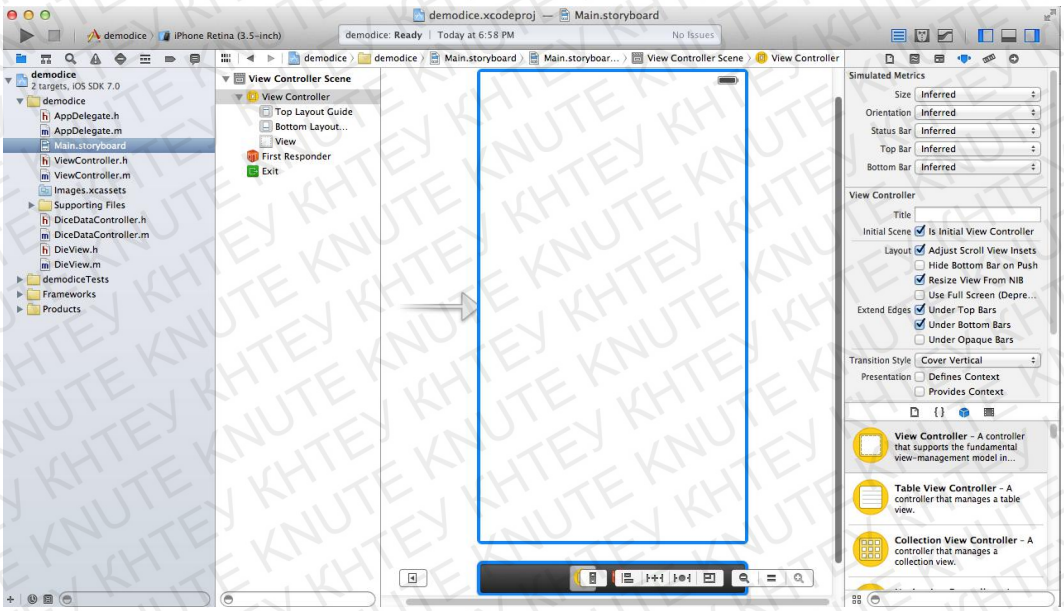

Рис. 1.5 Interface Builder

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:

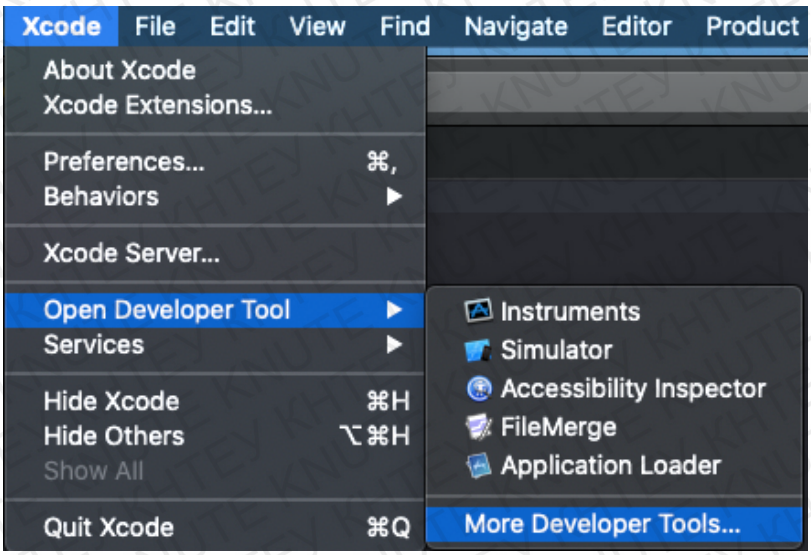

Рис. 1.6. Різноманітні Developer Tools в Xcode IDE 11.2

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.

1.5 Висновоки до розділу 1

Тема розробки мобільних застосувань для платформи iOS є досить цікавою і представляє собою широке поле для дослідження в галузі розробки мобільного програмного забезпечення. 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 платформу iOS. Також було доведено і обгрунтовано правильність вибору засобів для проектування. На основі проведеного аналізу встановлено, що найкращим середовищем розробки є Xcode, а мовою програмування – Swift.

РОЗДІЛ 2
ОГЛЯД ФРЕЙМВОРКІВ ДЛЯ РОЗРОБКИ МОБІЛЬНИХ ДОДАКТІВ ПЛАТФОРМИ IOS

2.1 Основні сучасні фреймворки для розробки мобільних додатків мобільної платформи iOS.

2.1.1 Фреймворк UIKit

UIKit – це платформа, яка визначає компоненти ядра iOS-додатка від ярликів і кнопок до табличних видів і контролерів навігації. У той час, як платформа Foundation визначає класи, протоколи і функції в розробці як під iOS, так і під OS X, платформа UIKit спрямована виключно на iOS-розробку. Це еквівалент середовища розробки Application Kit або AppKit для розробки OS X .

Як і Foundation, UIKit визначає класи, протоколи і функції, типи даних і константи. Він також додає додатковий функціонал в різні класи Foundation, начебто NSObject, NSString і NSValue за допомогою використання категорій Objective-C.

UIKit забезпечує ключову інфраструктуру для реалізації графічних, заснованих на подіях iOS-додатків. Кожне iOS-додаток використовує цю бібліотеку для реалізації наступних центральних можливостей :

- управління додатком;
- управління інтерфейсом користувача;
- підтримка графіки і вікон;
- підтримка багатозадачності;
- підтримка друку;
- підтримка обробки подій дотиків і жестів;
- об'єкти для відображення стандартних системних вікон і елементів управління;
- підтримка текстового і web-вмісту;
- підтримка «Вирізання», «Копіювання» та «Вставки»;
- підтримка анімації вмісту інтерфейсу користувача;
- інтеграція з іншими додатками системи за допомогою URL-схем;

- підтримка служби push-повідомлень Apple;
- підтримка додаткових можливостей для користувачів з обмеженими можливостями;
- планування і доставка локальних повідомлень;
- створення PDF-документів;
- підтримка особливих вікон введення інформації, які працюють як системна клавіатура;
- підтримка створення особливих текстових полів, які взаємодіють з системної клавіатурою.

Одним з ключових компонентів середовища розробки UIKit є клас UITableView. Він добре оптимізований для відображення нумерованих списків, як маленьких, так і великих. Table View можна редагувати і адаптувати під широкий діапазон сфер застосування, але основна ідея залишається незмінною - представлення нумерованого списку пунктів. Клас UITableView відповідає тільки за подання даних у вигляді списку рядків. Відображені дані управляються об'єктом dataSource в table view. Об'єктом dataSource може бути будь-який об'єкт, який відповідає протоколу UITableViewDataSource. Об'єкт-джерело даних в table view часто є оператором уявлення, який керує поданням table view.

Подібним чином, table view відповідає тільки за розпізнавання дотиків в table view. Він не відповідає за реагування на ці події. До того ж до об'єкта dataSource в table view, у table view також є делегований об'єкт. Коли table view розпізнає подію touch, він сповіщає делегований об'єкт про цю подію, і тоді вже в обов'язки делегованого об'єкта table view входить реагування на цю подію. Маючи об'єкт, який управляє даними, а також делегований об'єкт, який обробляє події взаємодії з користувачем, table view може зосередити ресурси на представленні даних. В результаті ми отримуємо дуже продуктивний UIKit- компонент з можливість повторного використання, який відмінно можна порівняти з MVC-патерном (Model-View-Controller). Клас UITableView успадковує параметри з UIView, і отже, відповідає за відображення даних програми.

Об'єкт data source трохи схожий, але не ідентичний делегованому об'єкту. Делегованого об'єкту передається контроль над призначеним для користувача інтерфейсом делегованих об'єктом. Об'єкт data source має контроль над даними. Table view запитує у об'єкта data source дані, які потрібно відобразити. Це має на увазі той факт, що об'єкт джерела даних також відповідальний за управління даними, які він передає в table view.

Клас UITableView успадковує параметри з UIScrollView, субкласса UIView, який надає підтримку відображення контенту, за розміром більшого, ніж вікно програми. Екземпляр UITableView складається з рядків, де кожен рядок містить одну клітинку - екземпляр UITableViewCell або субкласів. На відміну від UITableView в OS X, екземпляр NSTableView в UITableView має на одну колонку більше. Вкладені набори даних і ієрархії можна відобразити за допомогою комбінації table view і контролерів навігації (UINavigationController).

2.1.2 Фреймворк MapKit

Фреймворк MapKit надає прокручуваний інтерфейс з картою, який можна вбудувати в існуючу ієрархію вікон мобільного застосування. Ця технологія забезпечує підтримку анотування карти, тобто наносити коментарі на карту, виконує зворотне геокодування вибірок, щоб визначити placemarks для даного відображення координат.

MapKit дозволяє використовувати цю карту, щоб надати направлення або підсвітити пам'ятки. Додатки можуть програмно задавати атрибути карти або дозволити користувачеві управляти картою на свій розсуд. Також є можливість додавати примітки, картинки тощо до карти. У iOS 4.0, базове вікно з картою отримало підтримку переміщуваних анотацій і власних шарів . Переміщувані анотації дають можливість перепозиціонувати анотації як програмно, так і за допомогою маніпуляцій користувача, після того, як вони були вже додані на карту. Шари пропонують спосіб створення складних анотацій на карті, які складаються з більш ніж однієї точки. Наприклад, можна використовувати шари, щоб нанести

інформацію про маршрути автобусів, вибіркові карти, межі парків або інформацію поверх карти (наприклад, радіолокаційні дані).

2.2 Огляд напоширеніших методів оптимізації мобільних додатків під керуванням операційної системи iOS

2.2.1 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.

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.

2.2.2 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. 2.1. 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 Apple 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.

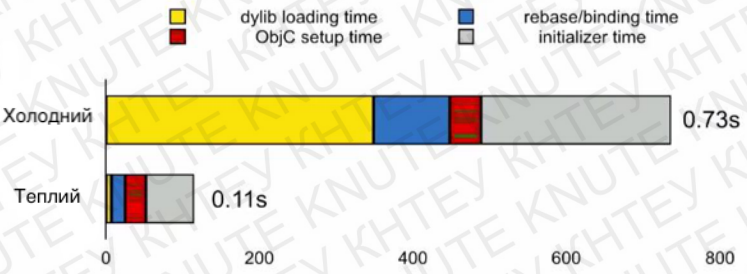

Рис. 2.2. Заміри часу запуска зіставних частин додатку при теплому та холодному старті

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.

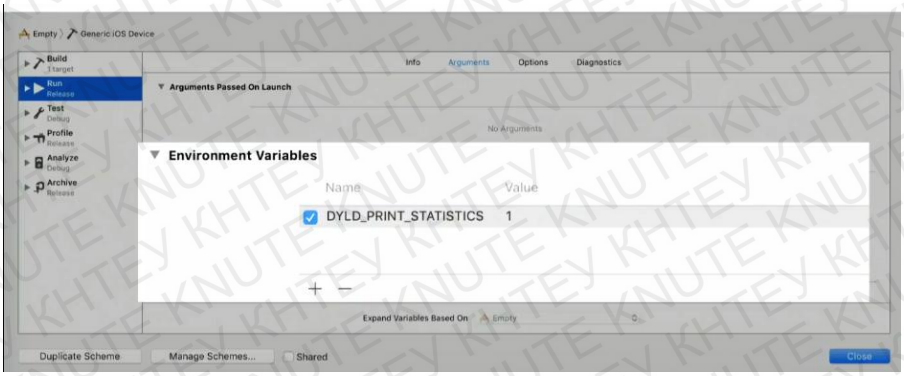

Рис. 2.3. Додавання змінної оточення

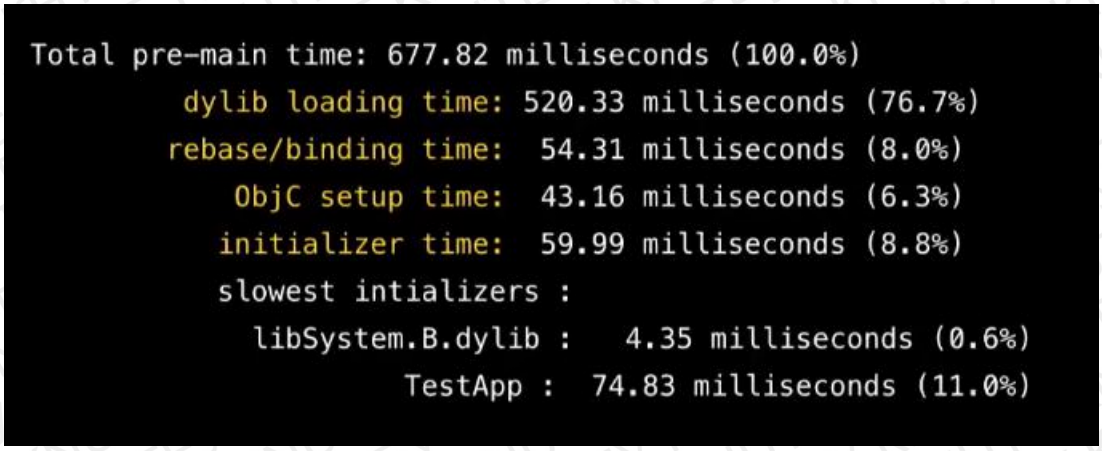

Рис. 2.4. Результат виконання додатку з включенною змінною оточення

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.

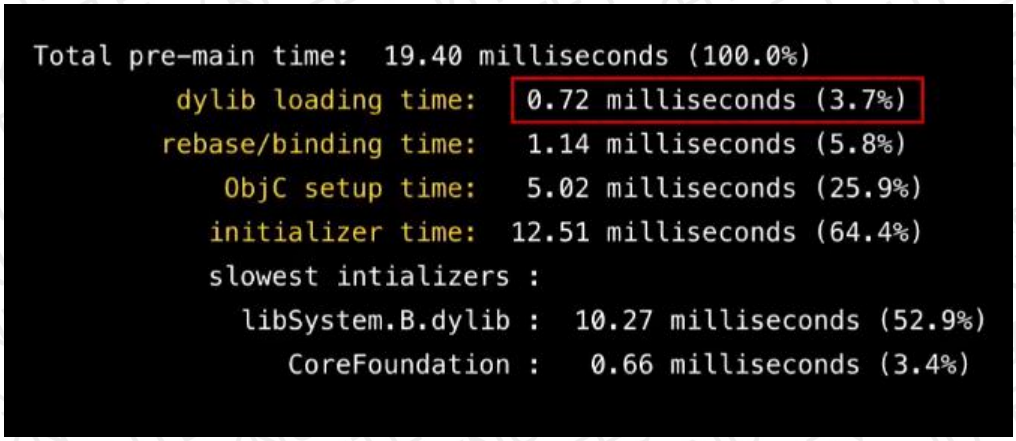

Рис. 2.5. Результат виконання додатку побудованого на мові Object-C

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

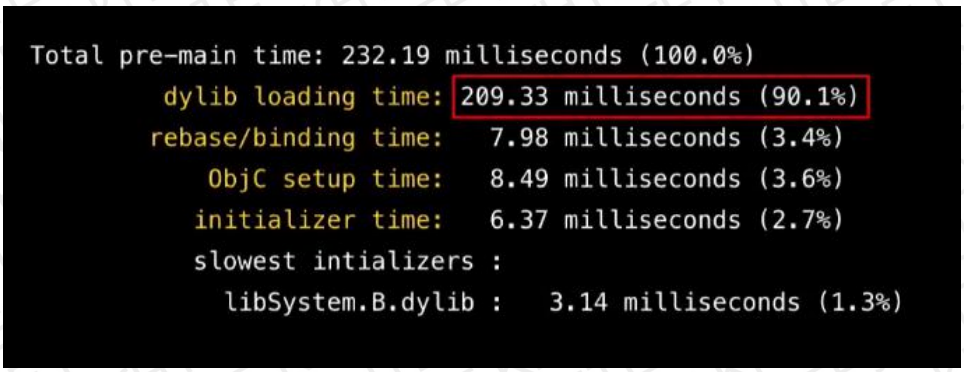

Рис. 2.6. Результат виконання додатку побудованого на мові Swift

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 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 6, і swift standard libraries просто стануть частиною системи, а додатки перестануть за собою ці бібліотеки тягати.

```
...
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. 2.7. Початковий набір подів

```
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. 2.8. 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 розробленого - 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.

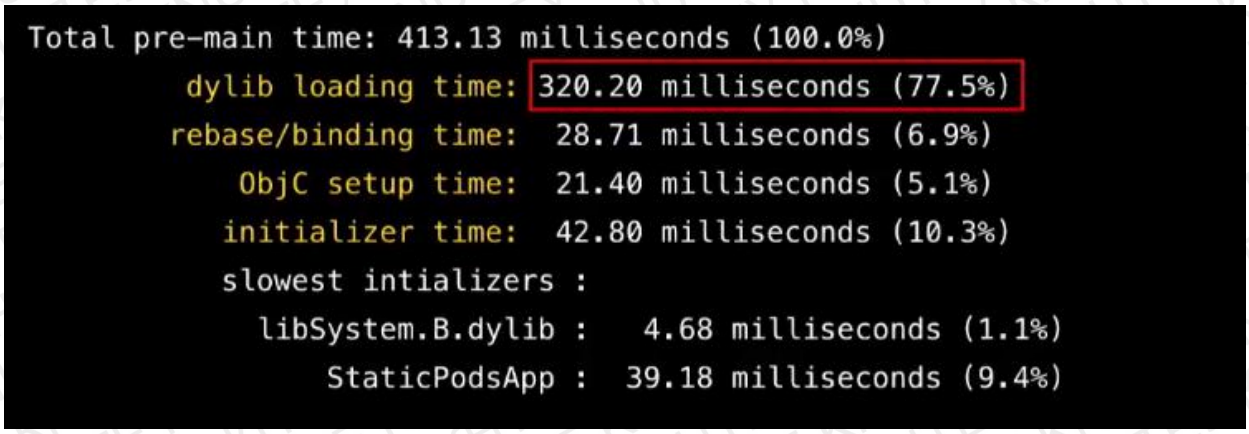

Рис. 2.9. Результат виконання додатку з плагіном для cocoa pods

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 перекривают його недоліки.

2.7 Висковоки до розділу 2

Розглянуто особливості архітектури платформи iOS та технологій розробки мобільних застосувань. Наведені основні фреймворки для вирішення поставленої задачі, обгрунтовано вибір даних технологій, адже ці інструменти найкраще підходять для розробки мобільних додатків, які в свою чергу повинні мати простий, лаконічний, привабливий користувацький інтерфейс, швидку і зручну навігацію, високу швидкодію. Проведений аналіз та оптимізовано ресурсоемність освновних модулів додатку та оптимізовано час запуску додатку у різних станах.

РОЗДІЛ 3.
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИЗНАЧЕННЯ ГЕОПОЗИЦІЇ КЛІЄНТІВ ЛОГІСТИЧНОЇ КОМПАНІЇ HYPERTRACK

3.1 Проектування інтерфейсу користувача

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).

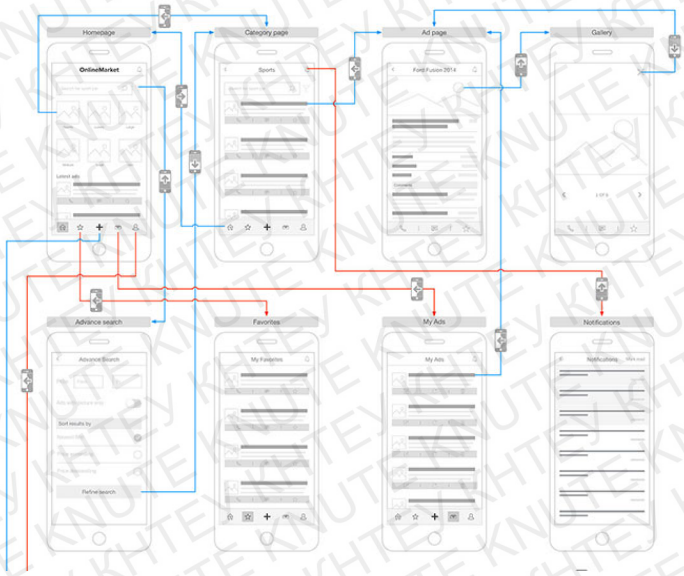

Рис. 3.1. Гайдлайн для мобільного додатку

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.

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 .

Отже, формулювання поетапної розробки виглядає наступним чином:

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

3.2 Оптимізація мобільного додатку для операційної системи iOS

Проаналізувавши та виконавши тестування означених у 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 враховував зменшення кількості часу, який необхідно затрачувати розробнику, при використанні цих методів оптимізації порознь та підвищення ефективності оптимізації.

При описанні методу оптимізації часу запуску 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 3.1 та 3.2.

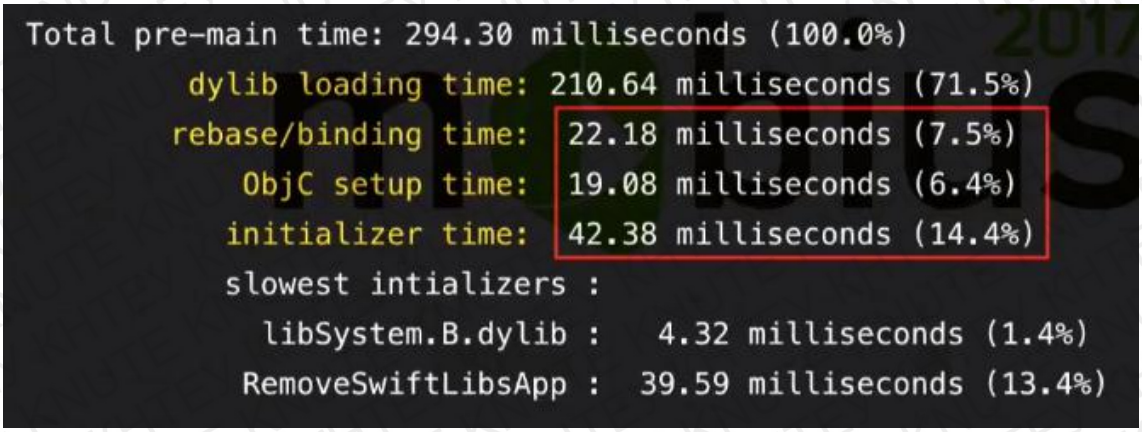

Рис. 3.1. Результат виконання оригінального методу оптимізації часу запуску

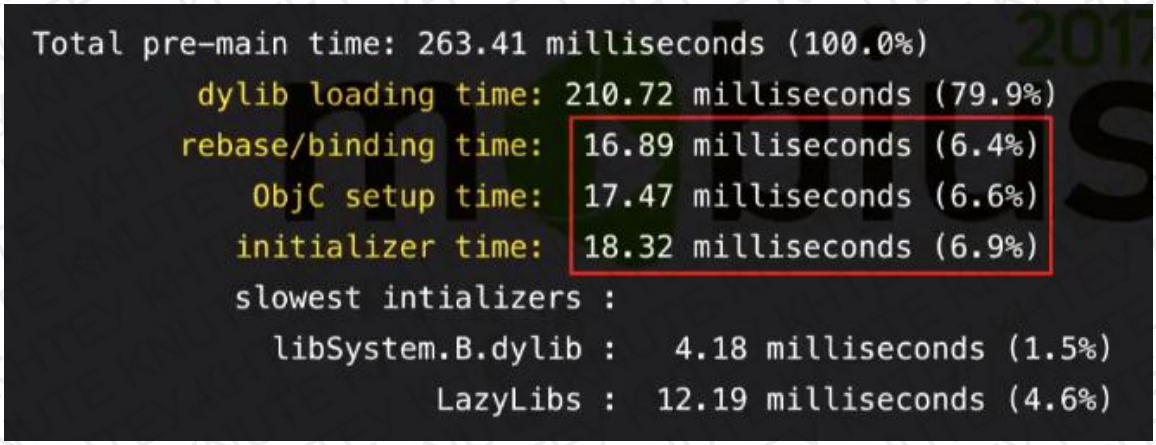

Рис. 3.2. Результат виконання модифікованого методу оптимізації часу запуску

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

3.3 Проектування та розробка мобільного додатку HyperTrack

Першочергово потрібно прозподілити файлову структуру для проетку. Побудова файлової структури проекту дозволить розподілити програмний код за його функціональним призначенням. Тобто потрібно визначити першочергові точки програмування та логіку їх розміщення.

При побудові файлової структури проекту слід врахувати функціональний підхід, тобто створити файли які будуть відповідати за збереження окремих частин коду і їх підключення безпосередньо у процесі програмування. Це є першочерговим та невідьемним єтапом в створенні та проектуванні программного продукту.

Основні файли розструктуровано по папкам для більш зрозумілої логіки використання та групування. В папці проекту HyperTrack розміщуються безпсередньо файли додатку для платформи iOS, головний інтерфейс,та контролери роботи додатку.

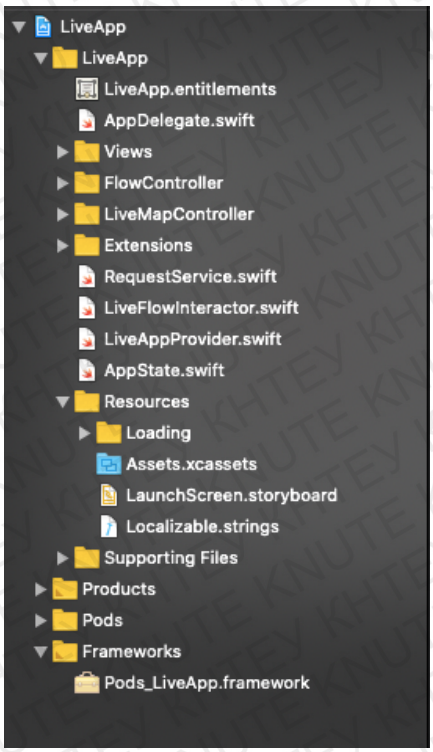

Рис.3.3 Структура файлів програмного продукту в середовищі розробки Xcode

Безпосередньо головні файли додатку розміщені в папці Live App в якій знаходится:

- Контроллер інтерфейсу
- Контроллер запуску
- Контроллер карт
- Контроллер датчиків заміру
- Контроллер авторизації
- Контроллер часу заміру

Відповідно до структури найбільш відповідальним стала схема роботи данних та структура файлу коду по суміщенню всіх функцій. Було прийняте рішення по змущеню всіх функцій та їх співпраця в одному файлі AppDelegate.swift. увібравши в себе основні функції та класси. Частина лістингу представлена нижче.

Для початку роботи основного фреймворку іпортується інформація із Pods_LiveApp.framework після чого задається фінкція відображення динамічного тексту на основному елементі інтерфейсу який відповідає за початок роботи добатку. Текст беспосередньо є динамічним та змінюється після початку або кінця роботи додатку

```
import UIKit
import HyperTrack
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
  var window: UIWindow?
  var appProvider: LiveAppProvider?
  func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?
    ) -> Bool {
    window = UIWindow(frame:UIScreen.main.bounds)
    window?.makeKeyAndVisible()
```

```
    window?.rootViewController = HTNavigationController()
    appProvider = LiveAppProvider()
    appProvider?.setUp()
    HyperTrack.registerForRemoteNotifications()
    return true
  }
```

Після чого викликається функція для відмалювання основного інтерфейсу программи для користувача та основних повідомлень та сповіщень про переміщення або зміни стану клієнта в реальному часі.

```
func application(
    _ application: UIApplication,
    didRegisterForRemoteNotificationsWithDeviceToken
    deviceToken: Data
    ) {
HyperTrack.didRegisterForRemoteNotificationsWithDeviceToken(deviceToken)
  }
  func application(
    _ application: UIApplication,
    didFailToRegisterForRemoteNotificationsWithError
    error: Error) {
    HyperTrack.didFailToRegisterForRemoteNotificationsWithError(error)
  }
  func application(
    _ application: UIApplication,
    didReceiveRemoteNotification userInfo: [AnyHashable: Any],
    fetchCompletionHandler completionHandler:
    @escaping (UIBackgroundFetchResult) -> Void
    ) {
```

```
        HyperTrack.didReceiveRemoteNotification(userInfo,
                                                fetchCompletionHandler: completionHandler)
    }
```

Також додано функцію для перезапуску попередніх функцій сповіщення так перезавантаження та повторної роботи відслідковування геопозиції при повільному чи відсутньому підключенні до мережі Інтернет.

```
func displayError(_ error: HyperTrackCriticalError, alertTitle: String) {
        guard let window = self.window,
            let viewController = window.rootViewController else { return }
        let alert = UIAlertController(
            title: NSLocalizedString(alertTitle, comment: ""),
            message: NSLocalizedString(error.errorMessage, comment: ""),
            preferredStyle: .alert)
        alert.addAction(
            UIAlertAction(title: NSLocalizedString("ALERT_OK_BUTTON", comment:
"""), style: .cancel, handler: nil)
        )
        viewController.present(alert, animated: true, completion: nil)
    }
}
extension Notification.Name {
    static let updatePKNotification = Notification.Name("HyperTrackUpdatePK")
    static let updateTrackingStateNotification =
Notification.Name("HyperTrackUpdateTrackingState")
    static let flowComplitedNotification =
Notification.Name("HyperTrackFlowComplited")
    static let getErrorNotification =
Notification.Name("HyperTrackGetErrorNotification")
}
```

В основний контролер додатку додано обробку та імпорт основних кітів та фреймворків які були розглянуті в першому розділі та додану обробку контроллерів запуску та завантаження основних модулів додатку.

```
import UIKit
import HyperTrack
import MapKit
import CoreMotion
import CoreLocation

class LiveMapViewController: UIViewController {
  private var requestService: RequestService!
  private var trackingLink: URL?
  private var isFirstStart: Bool = true
  private let tipsView: HTTipsView = {
  private let btCurrentUserPlace: HTButton = {
    let btn = HTButton(type:.custom)
    btn.clipsToBounds = true
    btn.setOnStateImage(
      normalImage: UIImage(named: "user_location_normal"),
      highlightedImage: UIImage(named: "user_location_selected"),
      disableImage: UIImage(named: "user_location_disable"))
    btn.currentState = .on
  btn.translatesAutoresizingMaskIntoConstraints = false
    btn.addTarget(
      self,
      action: #selector(moveToUserDotHendler),
      for: .touchUpInside)
    return btn
  }()
```

```
private let btChangeTrackingState: HTButton = {
    let btn = HTButton(type:.custom)
    btn.clipsToBounds = true
    btn.setOnStateImage(
        normalImage: UIImage(named: "tracking_on_normal"),
        highlightedImage: UIImage(named: "tracking_on_selected"),
        disableImage: UIImage(named: "tracking_on_disabled"))
    btn.setOffStateImage(
        normalImage: UIImage(named: "tracking_off_normal"),
        highlightedImage: UIImage(named: "tracking_off_selected"),
        disableImage: UIImage(named: "tracking_on_disabled"))
    btn.currentState = .off
    btn.translatesAutoresizingMaskIntoConstraints = false
    btn.addTarget(
        self,
        action: #selector(changeTrackingStateHendler(_:)),
        for: .touchUpInside)
    return btn
}()
private let btShareLiveLocation: UIButton = {
    let btn = UIButton.baseGreen
    btn.setTitle(String.localize(key: "MAP_BUTTON_TITLE"),
                 for: .normal)
    btn.addTarget(
        self,
        action: #selector(shareUserLocationHendler(_ :)),
        for: .touchUpInside)
```

Додано обробку завантаження при очікуванні данних від сервера для визначення зв’язку додатку з мережею інтернет

```
        getTrackingLinkFromServer { [weak self] link in
            guard let link = link, let self = self else { return }
            self.trackingLink = link
        }
        createUI()
    }
    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        reloadHyperTrackInitialization()
    }
    @objc private func createUI() {

        self.navigationController?.navigationBar.isHidden = true
        view.backgroundColor = .white
        let viewContainer = UIView()
        viewContainer.backgroundColor = .white
        viewContainer.translatesAutoresizingMaskIntoConstraints = false
        view.addSubview(map)
        view.addSubview(btChangeTrackingState)
        view.addSubview(viewContainer)
        viewContainer.addSubview(btShareLiveLocation)
        view.addSubview(btCurrentUserPlace)
        view.addSubview(notificationView)
        view.addSubview(tipsView)
        viewContainer.leftAnchor.constraint(equalTo:view.leftAnchor, constant:
0).isActive = true
        viewContainer.rightAnchor.constraint(equalTo:view.rightAnchor, constant:
0).isActive = true
        viewContainer.bottomAnchor.constraint(equalTo:view.bottomAnchor,
constant: 0).isActive = true
```

```
            viewContainer.heightAnchor.constraint(equalToConstant: 90).isActive =
true
```

Додаткові налаштування виконуються вже після запуску головного екрану користувача, для цього було додано окрему функцію:

```
        getTrackingLinkFromServer { [weak self] link in
            guard let link = link, let self = self else { return }
            self.trackingLink = link
        }
        createUI()
    }
  override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        reloadHyperTrackInitialization()
    }
    @objc private func createUI() {
        self.navigationController?.navigationBar.isHidden = true
        view.backgroundColor = .white
        let viewContainer = UIView()
        viewContainer.backgroundColor = .white
        viewContainer.translatesAutoresizingMaskIntoConstraints = false
        view.addSubview(map)
        view.addSubview(btChangeTrackingState)
        view.addSubview(viewContainer)
        viewContainer.addSubview(btShareLiveLocation)
        view.addSubview(btCurrentUserPlace)
        view.addSubview(notificationView)
        view.addSubview(tipsView)
        viewContainer.leftAnchor.constraint(equalTo:view.leftAnchor, constant:
0).isActive = true
```

```
            viewContainer.rightAnchor.constraint(equalTo:view.rightAnchor, constant:
0).isActive = true
            viewContainer.bottomAnchor.constraint(equalTo:view.bottomAnchor,
constant: 0).isActive = true
            viewContainer.heightAnchor.constraint(equalToConstant: 90).isActive =
true
```

Для відокремлення посилання на координати додано окремий обьект та фукнція з фреймворку MapKit. Це дозволяє користувачу ділитися своїм місцезнаходженням з іншими користувачами чи безпосередньо з клієнтом чи з компанією

```
        @objc private func shareUserLocationHendler(_ sender: HTButton) {
            if let trackingLink = self.trackingLink {
                self.openShareActionSheet(link: trackingLink)
            } else {
                getTrackingLinkFromServer { [weak self] link in
                    guard let link = link, let self = self else { return }
                    self.trackingLink = link
                    self.openShareActionSheet(link: link)
                }
            }
        }
        private func openShareActionSheet(link: URL) {
            let linkString = "\(NSLocalizedString("MAP_LINK_SHARING", comment:
"")) \(link.absoluteString)"
            DispatchQueue.main.async {
                let actionSheet = UIActivityViewController(activityItems: [linkString],
                                                           applicationActivities: nil)
                actionSheet.excludedActivityTypes = [.saveToCameraRoll]
                self.present(actionSheet, animated: true, completion: nil)
```

```
        }
    }
    private func getTrackingLinkFromServer(completionHandler: @escaping (_
link: URL?) -> Void) {
        requestService.getSharedLink { link in
            guard let link = link else { return }
            completionHandler(link)
```

3.4 Висновки до розділу 3

Мобільний додаток для відслідковування геопозиції клієнтів логістичної компанії HyperTrack відповідає поставленим цілям з урахуванням всіх специфік та можливостей платформи iOS. Розроблений інтерфейс є максимально інформативним для користувача для максимально зручного використовування додатку в реальному часі. Проведено оптимізацію методу розробки мобільного додатку для найефективнішої роботи додатку та найефективнішого та найменш ресурсномісткого запуску додатку.

В основі створення модифікованих методів оптимізації мобільних додатків полягає у зменшенні кількості тих незручних моментів, з якими може зіткнутися користувач під час використання придбаного мобільного пристрою. Розглянуті існуючі методи оптимізації мобільних додатків мають описані вище гідності та недоліки, одна як і будь-який інший результат, який був досягнутий, може бути покращений.

Оптимізація часу запуску мобільного додатку під керуванням операційної системи iOS, яка модифікована шляхом використання лінивого завантаження статичних бібліотек при старті додатку, зображує, що навіть такі невеликі рішення, виконують істотний ефект на загальну картину в цілому.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Тема розробки мобільних застосувань під платформу iOS є досить цікавою і представляє широке поле для подальших досліджень в галузі розробки мобільного ПЗ. Специфіка даного сегмента полягає в тому, що розробка iOS-додатків повинна проводитися з урахуванням особливостей мобільних пристроїв: відмінностями інтерфейсу, іншим розміром екрану, сенсорним управлінням. Актуальність теми підкреслюється широким спектром можливостей для втілення ідей у вигляді мобільного додатку.

Розглянуто основні засоби розробки, доведено і обгрунтовано правильність вибору цих засобів для проектування. На основі проведеного аналізу встановлено, що найкращим середовищем розробки є Xcode, а мовою програмування – Swift. Адже вони найбільше підходять для розробки додатків під платформу iOS.

В результаті виконання даної дипломної роботи досліджено основні особливості мови програмування Swift, на її основі було розроблено мобільний додаток для операційної системи iOS. Розглянуто основні переваги Swift перед Objective-C такі як простота у використанні, більш простіший у використанні синтаксис, швидкість виконання додатків вища, що призводить до зменшення кількості часу на розробку. Swift доцільно використовувати у тих випадках, коли проект створюється з нуля, адже на даний момент в світі дуже багато додатків, які написані на Objective-C, і їх краще підтримувати саме на цій мові, а не переписувати на Swift.

Розглянуто та проаналізовано основні особливості архітектури платформи iOS та технології розробки мобільних застосувань, визначено основні переваги та недоліки даної платформи. Наведено основні фреймворки для вирішення поставленої задачі, обгрунтовано вибір даних технологій, адже ці інструменти найкраще підходять для розробки мобільних додатків, використовуючи саме Swift, які в свою чергу повинні мати простий, лаконічний, привабливий користувацький інтерфейс, швидку і зручну навігацію, високу швидкодію. Розглянуті фреймворки і технології відносяться до вищого рівня архітектури, тому вони забезпечують 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.

Результатом проведеної даної роботи є підтверджений відповідним аналізом висновок, що Swift найкраще підходить для розробки сучасного мобільного ПЗ під платформу iOS.

Проведено аналіз розробеного програмного продукуту та зроблено оцінку використаних досліджених підходів, визначено основні переваги та недоліки створеного рішення.

В цілому ідея розробки мобільного ПЗ є досить перспективною, адже протягом останніх років показник, що характеризує рівень попиту на мобільні пристрої, постійно зростає. Така статистика дозволяє зробити висновок про те, що розробка мобільних додатків актуальна і доцільна. Головне грамотно оцінити, для кого і навіщо створюється ПЗ. Тільки корисна розробка отримає гідне визнання з боку користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Білас О.Є. Якість програмного забезпечення та тестування : навч. посіб. / О.Є. Білас. – Львів : Львів. політехніка, 2011. – 216
2. Далека В.Д. Алгоритми та методи обчислень. Ч. 1. Алгоритми: Методичні вказівки до виконання лабораторних робіт./ В. Д. Далека, Н.М. Бондіна, Н.Ю. Любченко – Харків: НТУ «ХПІ». – 2015. – 70 с.
3. Табунщик Г.В. Проектування та моделювання програмного забезпечення сучасних інформаційних систем : навчальний посібник / Г.В. Табунщик, Т.І. Каплієнко, О.А. Петрова. – Запоріжжя : ЗНТУ, 2016. – 270 с
4. Бушуєв С.Д. Методологія управління бюджетними проектами: Посібник/С.Д. Бушуєв, С.В. Цюцюра, О.В. Криворучко та ін. – К. : КНУБА, 2016. – 196 с.
5. Петренко Н.О. Управління проектами навчальний посібник. / Н.О. Петренко, Л.О. Кустріч, М. О. Гоменюк. – К. : «Центр учбової літератури», 2015. – 244 с
6. .Jacko J.A. Human-Computer Interaction Handbook (3rd Edition). / Julie A. Jacko– CRCPress:Taylor& Francis Group 2012 – 1518p.
7. Jonathan Peppers. Xamarin Cross-platform Application Development Second Edition / Peppers J. – Birmingham : Packt, 2015.
8. Nilanchala Panigrahy. Xamarin Mobile Application Development for Android Second Edition / Panigrahy N. – Birmingham : Packt, 2015. – 296 с.
9. Charles Petzold. Creating Mobile Apps with Xamarin.Forms First Edition
10. Petzold C. – Redmond : Microsoft Press, 2016. – 1187 с.12.09.2019.

Інтернет-ресурси

11. Working with Xamarin.Forms – Режим доступу : https://developer.xamarin.com/guides/xamarin-forms/working-with/. – Дата доступу : 15.04.2019.
12. The Swift Programming Language https://swift.org/documentation/ переклад на українську - https://killobatt.gitbooks.io/-swift/content/1_language_guide/0_the_basics.html

13. Best Hybrid Mobile App UI Frameworks: HTML5, CSS and JS – Режим доступу : http://noeticforce.com/best-hybrid-mobile-app-ui-frameworks- html5- js-css/. – Дата доступу : 29.05.2019.
14. RoboVM documentation – Режим доступу: http://docs.robovm.com/. – Дата доступу : 03.04.2019.
15. Ionic documentation – Режим доступу: http://ionicframework.com/docs/. – Дата доступу : 03.04.2019.
16. About mono – Режим доступу: http://www.mono-project.com/docs/about-mono/. – Дата доступу : 05.05.2019.
17. Xamarin application fundamentals – Режим доступу : http://developer.xamarin.com/guides/crossplatform/application_fundamental s/. – Дата доступу : 20.05.2018.
18. Xamarin.Android application fundamentals – Режим доступу : https://developer.xamarin.com/guides/android/application_fundamentals/. – Дата доступу : 13.04.2016
19. Xamarin.iOS application fundamentals – Режим доступу : https://developer.xamarin.com/guides/ios/application_fundamentals/. Дата доступу : 13.04.2018.

ДОДАТКИ

Додаток А

Лістинг файлу AppDelegate.swift

```
import UIKit
import HyperTrack

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

  var window: UIWindow?
  var appProvider: LiveAppProvider?
  func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?
    ) -> Bool {

    window = UIWindow(frame:UIScreen.main.bounds)
    window?.makeKeyAndVisible()
    window?.rootViewController = HTNavigationController()
    appProvider = LiveAppProvider()
    appProvider?.setUp()

    HyperTrack.registerForRemoteNotifications()

    return true
  }

  func application(
    _ application: UIApplication,
    didRegisterForRemoteNotificationsWithDeviceToken
    deviceToken: Data
    ) {
    HyperTrack.didRegisterForRemoteNotificationsWithDeviceToken(deviceToken)
```

```
    }

    func application(
        _ application: UIApplication,
        didFailToRegisterForRemoteNotificationsWithError
        error: Error) {
        HyperTrack.didFailToRegisterForRemoteNotificationsWithError(error)
    }

    func application(
        _ application: UIApplication,
        didReceiveRemoteNotification userInfo: [AnyHashable: Any],
        fetchCompletionHandler completionHandler:
        @escaping (UIBackgroundFetchResult) -> Void
        ) {
        HyperTrack.didReceiveRemoteNotification(userInfo,
                                                fetchCompletionHandler: completionHandler)
    }

    /// For this simple example we display alerts with an ability to try to start tracking again.
    func displayError(_ error: HyperTrackCriticalError, alertTitle: String) {
        guard let window = self.window,
            let viewController = window.rootViewController else { return }
        let alert = UIAlertController(
            title: NSLocalizedString(alertTitle, comment: ""),
            message: NSLocalizedString(error.errorMessage, comment: ""),
            preferredStyle: .alert)
        alert.addAction(
            UIAlertAction(title: NSLocalizedString("ALERT_OK_BUTTON", comment: ""), style:
.cancel, handler: nil)
        )
        viewController.present(alert, animated: true, completion: nil)
    }
}
```

```
extension Notification.Name {
    static let updatePKNotification = Notification.Name("HyperTrackUpdatePK")
    static let updateTrackingStateNotification = Notification.Name("HyperTrackUpdateTrackingState")
    static let flowComplitedNotification Notification.Name("HyperTrackFlowComplited")
    static let getErrorNotification = Notification.Name("HyperTrackGetErrorNotification")
}
```

Лістинг файлу HTButton.swift

```
import UIKit
enum ButtonState {
   case on
   case off
}
class HTButton: UIButton {
   private var backgroundImageONForStateHighlighted: UIImage?
   private var backgroundImageONForStateNormal: UIImage?
   private var backgroundImageONForStateDisable: UIImage?
   private var backgroundImageOFFForStateHighlighted: UIImage?
   private var backgroundImageOFFForStateNormal: UIImage?
   private var backgroundImageOFFForStateDisable: UIImage?
   var currentState: ButtonState = .off {
      didSet {
         changeBackgroundImage()
      }
   }
   func setOnStateImage(normalImage: UIImage?,
                        highlightedImage: UIImage?,
                        disableImage: UIImage?) {
      backgroundImageONForStateHighlighted = highlightedImage
      backgroundImageONForStateNormal = normalImage
      backgroundImageONForStateDisable = disableImage
   }
   func setOffStateImage(normalImage: UIImage?,
                         highlightedImage: UIImage?,
                         disableImage: UIImage?) {
      backgroundImageOFFForStateHighlighted = highlightedImage
      backgroundImageOFFForStateNormal = normalImage
      backgroundImageOFFForStateDisable = disableImage
```

```
    }
    private func changeBackgroundImage() {
        switch currentState {
        case .on:
            self.setBackgroundImage(
                backgroundImageONForStateNormal,
                for: .normal)
            self.setBackgroundImage(
                backgroundImageONForStateHighlighted,
                for: .highlighted)
            self.setBackgroundImage(
                backgroundImageONForStateDisable,
                for: .disabled)
        case .off:
            self.setBackgroundImage(
                backgroundImageOFFForStateNormal,
                for: .normal)
            self.setBackgroundImage(
                backgroundImageOFFForStateHighlighted,
                for: .highlighted)
            self.setBackgroundImage(
                backgroundImageOFFForStateDisable,
                for: .disabled)
        }
    }
    override func draw(_ rect: CGRect) {
        drawShadow()
    }
    func drawShadow() {
        self.layer.shadowColor = UIColor.black.cgColor
        self.layer.shadowOffset = CGSize(width: 0, height: 5)
        self.layer.shadowOpacity = 0.2
        self.layer.shadowRadius = 3
```

```
        self.layer.masksToBounds = false
    }
    func hide(_ animated: Bool) {
        self.tag = 0
        if animated {
            createAnimation(true)
        } else {
            self.alpha = 0
        }
    }
    func show(_ animated: Bool) {
        self.tag = 1
        if animated {
            createAnimation(false)
        } else {
            self.alpha = 1
        }
    }
    private func createAnimation(_ isHidde: Bool) {
        UIView.animate(withDuration: 0.3) {
            self.alpha = isHidde ? 0 : 1
        }
    }
}
```

Лістинг файлу WelcomeViewController.swift

```
import UIKit
fileprivate let welcomeStateKey = "WelcomeStateKey"
class WelcomeViewController: BaseFlowController {
  private var currentState: Bool = false {
    didSet {
      UserDefaults.standard.set(
        currentState, forKey: welcomeStateKey)
    }
  }
  private let titleLabel: UILabel = {
    let label = UILabel.titleLabel
    label.text = String.localize(key: "WELCOME_TITLE_1")
    return label
  }()
  private let image: UIImageView = {
    let image = UIImageView.baseImageView
    image.image = UIImage(named: "illustrations")
    return image
  }()
  private let stepOneLabel: UILabel = {
    let label = UILabel.baseLabel
    label.text = String.localize(key: "WELCOME_STEP_1")
    return label
  }()
  private let stepTwoLabel: UILabel = {
    let label = UILabel.baseLabel
    label.text = String.localize(key: "WELCOME_STEP_2")
    return label
  }()
  private let stepThreeLabel: UILabel = {
```

```
    let label = UILabel.baseLabel
    label.text = String.localize(key: "WELCOME_STEP_3")
    return label
}()
private let button: UIButton = {
    let btn = UIButton.baseGreen
    btn.setTitle(String.localize(
        key: "WELCOME_BUTTON_TITLE"), for: .normal)
    btn.addTarget(
        self,
        action: #selector(buttonClicked),
        for: .touchUpInside)
    return btn
}()
convenience init() {
    self.init(nibName:nil, bundle:nil)
    currentState = UserDefaults.standard.bool(
        forKey: welcomeStateKey)
    isAnimationNeeded = false
}
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view.
    createUI()
}
private func createUI() {
    self.navigationController?.navigationBar.isHidden = true
    view.backgroundColor = .white
    view.addSubview(titleLabel)
    view.addSubview(image)
    let viewLabelContainer = UIView()
    viewLabelContainer.translatesAutoresizingMaskIntoConstraints = false
    viewLabelContainer.addSubview(stepOneLabel)
```

```
viewLabelContainer.addSubview(stepTwoLabel)
viewLabelContainer.addSubview(stepThreeLabel)
view.addSubview(viewLabelContainer)
view.addSubview(button)
titleLabel.topAnchor.constraint(
   equalTo:view.topAnchor,
   constant: view.frame.height * 0.1256).isActive = true
titleLabel.leftAnchor.constraint(
   equalTo:view.leftAnchor,
   constant:37).isActive = true
titleLabel.rightAnchor.constraint(
   equalTo:view.rightAnchor,
   constant:-37).isActive = true
image.topAnchor.constraint(
   equalTo:titleLabel.bottomAnchor,
   constant:30).isActive = true
image.leftAnchor.constraint(
   equalTo:view.leftAnchor,
   constant:70).isActive = true
image.rightAnchor.constraint(
   equalTo:view.rightAnchor,
   constant:-70).isActive = true
image.heightAnchor.constraint(
   equalTo: image.widthAnchor,
   multiplier: 1.0/1.0).isActive = true
viewLabelContainer.topAnchor.constraint(
   equalTo:image.bottomAnchor,
   constant: 30).isActive = true
viewLabelContainer.centerXAnchor.constraint(
   equalTo: button.centerXAnchor).isActive = true
viewLabelContainer.heightAnchor.constraint(
   equalToConstant: 19).isActive = true
stepOneLabel.topAnchor.constraint(
```

```
    equalTo:viewLabelContainer.bottomAnchor,
    constant: 0).isActive = true
stepOneLabel.leftAnchor.constraint(
    equalTo:viewLabelContainer.leftAnchor,
    constant:0).isActive = true
stepOneLabel.rightAnchor.constraint(
    equalTo:viewLabelContainer.rightAnchor,
   constant:0).isActive = true
stepTwoLabel.topAnchor.constraint(
    equalTo:stepOneLabel.bottomAnchor,
    constant: 5).isActive = true
stepTwoLabel.leftAnchor.constraint(
    equalTo:viewLabelContainer.leftAnchor,
    constant:0).isActive = true
stepTwoLabel.rightAnchor.constraint(
    equalTo:viewLabelContainer.rightAnchor,
    constant:0).isActive = true

stepThreeLabel.topAnchor.constraint(
    equalTo:stepTwoLabel.bottomAnchor,
    constant: 5).isActive = true
stepThreeLabel.leftAnchor.constraint(
    equalTo:viewLabelContainer.leftAnchor,
    constant:0).isActive = true
stepThreeLabel.rightAnchor.constraint(
    equalTo:viewLabelContainer.rightAnchor,
    constant:0).isActive = true

button.bottomAnchor.constraint(
    equalTo:view.safeAreaLayoutGuide.bottomAnchor,
    constant: -28).isActive = true
button.leftAnchor.constraint(
    equalTo:view.leftAnchor,
```

```
            constant:28).isActive = true
        button.rightAnchor.constraint(
            equalTo:view.rightAnchor,
            constant:-28).isActive = true
        button.heightAnchor.constraint(
            equalToConstant: 40).isActive = true
    }
    @objc func buttonClicked() {
        currentState = true
        interactorDelegate?.haveFinishedFlow(
            sender: self)
    }
    override func isFlowCompleted() -> Bool {
        return currentState
    }
}
```

LiveMapViewController.swift

```
import UIKit
import HyperTrack
import MapKit
import CoreMotion
import CoreLocation

class LiveMapViewController: UIViewController {
    private var requestService: RequestService!
    private var trackingLink: URL?
    private var isFirstStart: Bool = true
    private let tipsView: HTTipsView = {
        let view = HTTipsView()
        view.isHidden = true
        return view
    }()
    private let notificationView: HTNotificationView = {
        let view = HTNotificationView()
        return view
    }()
    @objc private let map: MKMapView = {
        let map = MKMapView()
        map.tintColor = UIColor("#00CE5B")
        map.translatesAutoresizingMaskIntoConstraints = false
        map.showsCompass = false
        return map
    }()
    private let btCurrentUserPlace: HTButton = {
        let btn = HTButton(type:.custom)
        btn.clipsToBounds = true
        btn.setOnStateImage(
            normalImage: UIImage(named: "user_location_normal"),
            highlightedImage: UIImage(named: "user_location_selected"),
```

```
            disableImage: UIImage(named: "user_location_disable"))
        btn.currentState = .on
        btn.translatesAutoresizingMaskIntoConstraints = false
        btn.addTarget(
            self,
            action: #selector(moveToUserDotHendler),
            for: .touchUpInside)
        return btn
    }()
    private let btChangeTrackingState: HTButton = {
        let btn = HTButton(type:.custom)
        btn.clipsToBounds = true
        btn.setOnStateImage(
            normalImage: UIImage(named: "tracking_on_normal"),
            highlightedImage: UIImage(named: "tracking_on_selected"),
            disableImage: UIImage(named: "tracking_on_disabled"))
        btn.setOffStateImage(
            normalImage: UIImage(named: "tracking_off_normal"),
            highlightedImage: UIImage(named: "tracking_off_selected"),
            disableImage: UIImage(named: "tracking_on_disabled"))
        btn.currentState = .off
        btn.translatesAutoresizingMaskIntoConstraints = false
        btn.addTarget(
            self,
            action: #selector(changeTrackingStateHendler(_:)),
            for: .touchUpInside)
        return btn
    }()
    private let btShareLiveLocation: UIButton = {
        let btn = UIButton.baseGreen
        btn.setTitle(String.localize(key: "MAP_BUTTON_TITLE"),
                     for: .normal)
        btn.addTarget(
```

```
            self,
            action: #selector(shareUserLocationHendler(_ :)),
            for: .touchUpInside)
        return btn
    }()

    convenience init(appState: AppState) {
        self.init(nibName: nil, bundle: nil)
        self.requestService = RequestService(appState)
        configureController()
    }

    private func configureController() {
        NotificationCenter.default.addObserver(
            self,
            selector: #selector(determineTrackingStateBehaviour),
            name: UIApplication.willEnterForegroundNotification,
            object: nil)
        NotificationCenter.default.addObserver(
            self,
            selector: #selector(determineTrackingStateBehaviour),
            name: NSNotification.Name.HyperTrackStartedTracking,
            object: nil)
        NotificationCenter.default.addObserver(
            self,
            selector: #selector(determineTrackingStateBehaviour),
            name: NSNotification.Name.HyperTrackStoppedTracking,
            object: nil)
        NotificationCenter.default.addObserver(
            self,
            selector: #selector(getHyperTrackError(_:)),
            name: Notification.Name.getErrorNotification,
            object: nil)
```

```
    NotificationCenter.default.addObserver(
      self,
      selector: #selector(hypertrackInitialized(_:)),
      name: Notification.Name.HyperTrackHasInitialized,
      object: nil)
    map.delegate = self
  }

  override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view.
    getTrackingLinkFromServer { [weak self] link in
      guard let link = link, let self = self else { return }
      self.trackingLink = link
    }
    createUI()
  }

  override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    reloadHyperTrackInitialization()
  }

  @objc private func createUI() {

    self.navigationController?.navigationBar.isHidden = true

    view.backgroundColor = .white
    let viewContainer = UIView()
    viewContainer.backgroundColor = .white
    viewContainer.translatesAutoresizingMaskIntoConstraints = false
    view.addSubview(map)
    view.addSubview(btChangeTrackingState)
```

```
view.addSubview(viewContainer)
viewContainer.addSubview(btShareLiveLocation)
view.addSubview(btCurrentUserPlace)
view.addSubview(notificationView)
view.addSubview(tipsView)

viewContainer.leftAnchor.constraint(equalTo:view.leftAnchor, constant: 0).isActive = true
viewContainer.rightAnchor.constraint(equalTo:view.rightAnchor, constant: 0).isActive = true
viewContainer.bottomAnchor.constraint(equalTo:view.bottomAnchor, constant: 0).isActive = true
viewContainer.heightAnchor.constraint(equalToConstant: 90).isActive = true

map.topAnchor.constraint(equalTo:view.topAnchor, constant: 0).isActive = true
map.leftAnchor.constraint(
   equalTo:view.leftAnchor,
   constant: 0).isActive = true
map.rightAnchor.constraint(
   equalTo:view.rightAnchor,
   constant: 0).isActive = true
map.bottomAnchor.constraint(
   equalTo:view.bottomAnchor,
   constant: 0).isActive = true

btChangeTrackingState.topAnchor.constraint(
   equalTo:view.safeAreaLayoutGuide.topAnchor,
   constant: 5).isActive = true
btChangeTrackingState.rightAnchor.constraint(
   equalTo:view.rightAnchor,
   constant: -15).isActive = true

tipsView.rightAnchor.constraint(
   equalTo:btChangeTrackingState.leftAnchor,
   constant: -10).isActive = true
tipsView.centerYAnchor.constraint(
```

```
        equalTo: btChangeTrackingState.centerYAnchor).isActive = true

    btCurrentUserPlace.rightAnchor.constraint(
        equalTo:viewContainer.rightAnchor,
        constant: -15).isActive = true
    btCurrentUserPlace.bottomAnchor.constraint(
        equalTo:viewContainer.topAnchor,
        constant: -17).isActive = true

    btShareLiveLocation.bottomAnchor.constraint(
        equalTo:viewContainer.bottomAnchor,
        constant: -28).isActive = true
    btShareLiveLocation.leftAnchor.constraint(
        equalTo:viewContainer.leftAnchor,
        constant:28).isActive = true
    btShareLiveLocation.rightAnchor.constraint(
        equalTo:viewContainer.rightAnchor,
        constant:-28).isActive = true
    btShareLiveLocation.heightAnchor.constraint(
        equalToConstant: 40).isActive = true

    notificationView.leftAnchor.constraint(
        equalTo:view.leftAnchor,
        constant: 0).isActive = true
    notificationView.rightAnchor.constraint(
        equalTo:view.rightAnchor,
        constant: 0).isActive = true
    notificationView.bottomAnchor.constraint(
        equalTo:view.bottomAnchor,
        constant: -90).isActive = true

    determineTrackingStateBehaviour()
}
```

```
@objc private func determineTrackingStateBehaviour() {
    if HyperTrack.isTracking {
        map.showsUserLocation = true
        btShareLiveLocation.isEnabled = true
        btChangeTrackingState.currentState = .on
        notificationView.hideNotificationView()
        tipsView.hideTipsView()
    } else {
        btCurrentUserPlace.hide(false)
        map.showsUserLocation = false
        btShareLiveLocation.isEnabled = false
        btChangeTrackingState.currentState = .off
    }
}

private func showCurrentUserPlaceButtonIfNeeded() {
    if HyperTrack.isTracking,
        !map.isUserLocationVisible {
        btCurrentUserPlace.show(true)
    } else {
        btCurrentUserPlace.hide(true)
    }
}

@objc private func changeTrackingStateHendler(_ sender: HTButton) {
    if HyperTrack.isTracking {
        HyperTrack.stopTracking()
        NotificationCenter.default.post(
            name: Notification.Name.updateTrackingStateNotification,
            object: nil,
            userInfo: [userTrackingStateKey: false])
    } else {
```

```
        HyperTrack.startTracking()
        NotificationCenter.default.post(
            name: Notification.Name.updateTrackingStateNotification,
            object: nil,
            userInfo: [userTrackingStateKey: true])
    }

    if let userCoordinate = self.map.userLocation.location,
        !map.visibleMapRect.contains(MKMapPoint(userCoordinate.coordinate)),
        !self.isFirstStart {
        self.btCurrentUserPlace.show(true)
    }
}

// MARK: Map

@objc private func moveToUserDotHendler() {
    btCurrentUserPlace.hide(true)
    map.setRegion(MKCoordinateRegion(
        center: map.userLocation.coordinate,
        latitudinalMeters: 400,
        longitudinalMeters: 400), animated: true)
}

// MARK: Tracking link

@objc private func shareUserLocationHendler(_ sender: HTButton) {
    if let trackingLink = self.trackingLink {
        self.openShareActionSheet(link: trackingLink)
    } else {
        getTrackingLinkFromServer { [weak self] link in
            guard let link = link, let self = self else { return }
            self.trackingLink = link
```

```
            self.openShareActionSheet(link: link)
        }
    }
}

private func openShareActionSheet(link: URL) {
    let linkString = "\(NSLocalizedString("MAP_LINK_SHARING", comment: "")) \(link.absoluteString)"
    DispatchQueue.main.async {
        let actionSheet = UIActivityViewController(activityItems: [linkString],
                                                   applicationActivities: nil)
        actionSheet.excludedActivityTypes = [.saveToCameraRoll]
        self.present(actionSheet, animated: true, completion: nil)
    }
}

private func getTrackingLinkFromServer(completionHandler: @escaping (_ link: URL?) -> Void) {
    requestService.getSharedLink { link in
        guard let link = link else { return }
        completionHandler(link)
    }
}

// MARK: Error hendler

@objc private func getHyperTrackError(_ notif: Notification) {
    determineTrackingStateBehaviour()
    guard let error = notif.userInfo?[errorKey] as? HyperTrackCriticalError else { return }
    switch error.type {
    case .criticalErrorPermissionDenied:
        notificationView.showNotificationView(errorText:
NSLocalizedString("PERMISSIONS_ERROR_MESSAGE", comment: ""))
    case .criticalErrorAuthorizationError,
     .criticalErrorGeneralError,
```

```
            .criticalErrorInvalidPublishableKey:
            let appDelegate = UIApplication.shared.delegate as! AppDelegate
            appDelegate.displayError(error, alertTitle: "Error")
        @unknown default:
            fatalError()
        }
        tipsView.hideTipsView()
    }

    @objc private func hypertrackInitialized(_ notif: Notification) {
        let appDelegate = UIApplication.shared.delegate as! AppDelegate
        if !HyperTrack.isTracking,
            let trackingState = appDelegate.appProvider?.appState.isTrackingStartedByUser,
            !trackingState {
            tipsView.showTipsView()
        } else {
            tipsView.hideTipsView()
        }
    }

    private func reloadHyperTrackInitialization() {
        let appDelegate = UIApplication.shared.delegate as! AppDelegate
        appDelegate.appProvider?.canSetupHyperTrack()
    }
}

extension LiveMapViewController: MKMapViewDelegate {
    func mapView(_ mapView: MKMapView,
                 regionDidChangeAnimated
        animated: Bool) {
        showCurrentUserPlaceButtonIfNeeded()
    }
```

```
    // First time change Camera&Zoom to current user location
    func mapView(_ mapView: MKMapView,
                 didUpdate
      userLocation: MKUserLocation) {
      if isFirstStart {
        isFirstStart = false
        moveToUserDotHendler()
      }
    }
}
```