НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

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

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»

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

"___" _____ 2016 р.

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

зі спеціальності 7.05010102, 8.05010102 Інформаційні технології проектування
7.050103, 8.05010103 Системне проектування
(код та назва спеціальності)

на тему: «Методи та інструментальні засоби побудови додатків для Apple Watch / iPhone. Auto Layout як засіб побудови графічних та мультимедійних інтерфейсів для WatchOS/iOS його аналіз та практичне використання»

Виконав: студент IV курсу, групи ДА-21
(шифр групи)

Осадчий Дмитро Юрійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник к.т.н., доцент Цурін О.П. ____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ проф. Семенченко Н.В. _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

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

Нормоконтроль ст.. викладач Британов О.А. ____
(підпис)

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

Київ – 2016 року

Національний технічний університет України
«Київський політехнічний інститут»

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

Кафедра Системного проектування
(повна назва)

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

Спеціальність 7.050102, 8.050102 Інформаційні технології проектування
7.050103, 8.050103 Системне проектування
(код і назва)

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

«___»_____2016 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Осадчий Дмито Юрійович
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) «Методи та інструментальні засоби побудови додатків для Apple Watch / iPhone. Auto Layout як засіб побудови графічних та мультимедійних інтерфейсів для WatchOS/iOS його аналіз та практичне використання»
керівник проекту (роботи) Цурін О.П., доцент к.т.н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «__» _____ 2016 р. №_____

2. Строк подання студентом проекту (роботи) _____

3. Вихідні дані до проекту (роботи)

Розглянути і проаналізувати мобільні додатки для операційної системи iOS. Розглянути технологію Auto-Layout, та відобразити аспекти її роботи.Розробити додаток який буде використовувати технологію Auto-Layout, а також передачу даних між розумним годиником і додатком. Для реалізації використовувати інструмент Xcode.
Мобільний додаток на вході буде отримувати дані з розумного годиника Apple Watch та відображати їх на єкрані.
Передбачена можливість збереження даних на пристрої, після виходу із додатку.
Використання CoreData для збереження даних на пристрої.

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

1. Вступ, огляд Apple Watch, оперційні системи iOS та WatchOS.
2. Аналіз та порівняння мов програмування objective c та swift.
3. Auto-layout як основний інструмент для створення графічного і мультимедійного інтерфейсу.
4. Анатомія Constraints в Auto-Layout.
5. Програмна реалізація та тестування додатку.
6. Функціонально-вартісний аналіз програмного продукту

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів тощо)

1. Схема роботы пари iPhone-Apple Watch – плакат.
2. Ілюстрація можливостей інтерфейсу Apple Watch – плакат.
3. Ілюстрація роботи створеного додатку – плакат.

6. Консультанти розділів проекту (роботи)*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	Семенченко Н.В. проф.док.ек.н		

7. Дата видачі завдання 01.02.2016

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2016	
2	Збір інформації	15.02.2016	
3	Вивчення варіантів реалізації та вибір архітектури для розробки	28.02.2016	
4	Розробка алгоритму роботи та проектування семантичного графу	10.03.2016	
5	Розробка плану тестування	15.03.2016	
6	Розробка семантичного веб-додатку	25.04.2016	
7	Тестування моделі	30.04.2016	
8	Оформлення дипломної роботи	31.05.2016	
9	Отримання допуску до захисту та подача роботи в ДЕК	08.06.2016	

Студент _____ Осадчий Д.Ю..
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ Цурін О.П.
(підпис) (ініціали, прізвище)

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

АНОТАЦІЯ

бакалаврської дипломної роботи Осадчого Дмитра Юрійовича

на тему «Методи та інструментальні засоби побудови додатків для Apple Watch / iPhone. Auto Layout як засіб побудови графічних та мультимедійних інтерфейсів для WatchOS/iOS його аналіз та практичне використання»

Дана дипломна робота присвячена вивченню можливостей побудови додатків для розумних годинників Apple Watch та взаємодії їх з iPhohe. В рамках дипломної роботи був проведений аналіз існуючих методів:

1. Операційні системи iOS та WatchOS;
2. Мова програмування.
3. Інструмент Auto-Layout для створення інтерфейсів

Результатом роботи є створення програмного продукту для Apple Watch, що являє собою фітнес додаток для користувачів, які займаються бігом. Також було досліджено системи та засоби зчитування таких характеристик користувача, як пульс, кількість витрачених калорій та географічні координати.

Результати роботи доповідались на конференціЇ SAIT 2016.

Загальний обсяг роботи: 94 сторінок, 56 рисунків, 10 таблиць, 13 посилань.

Ключові слова: Apple Watch, iPhone, Auto-Layout, WatchOS, iOS

АНОТАЦИЯ

бакалаврской дипломной работы Осадчего Дмитрий Юрьевича

на тему: «Методы та инструменты построения приложение для Apple Watch / iPhone. Auto Layout как метод построения графических и мультимедийных интерфейсов в WatchOS/iOS, его анализ и практическое использование »

Данная дипломная работа посвящена изучению возможностей построения приложений для умных часов Apple Watch и взаимодействия их с iPhohe. В рамках дипломной работы был проведен анализ существующих методов:

1. Операціонная система iOS и WatchOS;
2. Язык программирования.
3. Инструмент Auto-Layout для создания интерфейсов.

Результатом работы является создание программного продукта для Apple Watch, который представляет собой фитнес приложение для пользователей, занимающихся бегом. Также были исследованы системы и средства считывания таких характеристик пользователя, как пульс, количество потраченных калорий и географические координаты.

Результаты работы докладывалась на конференции SAIT 2016.

Общий объем работы 94 страниц, 56 рисунка, 10 таблиц, 13 библиографических наименований.

Ключевые слова: Apple Watch, iPhone, WatchOS, iOS, Auto-Layout.

ANNOTATION

on Dmytro Osadchyy bachelor's degree thesis: "Auto-Layout as an instrument for building multimedia and graphical interfaces in WatchOS/iOS, analise and practical usage"

This thesis is devoted to the study of possibilities to build applications for smart watches Apple Watch, and their interaction with iPhohe. analysis of existing methods was carried out in the framework of the thesis:

1. Operation systems iOS and WatchOS;
2. Programming Language.
3. User Experience Interface Auto-Layout

The work is to create software for Apple Watch, which is a fitness app for people who want to run. Also, the system and the means for reading such user characteristics have been studied as heart rate, calories burned, and geographic coordinates.

The results of work were reported at the 2016 conference of the SAIT.

The total amount of work: 94 pages, 56 pictures, 10 tables, 13 bibliographical items.

Keywords: Apple Watch, iPhone, WatchOS, iOS, Auto-Layout.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ	програмне забезпечення
GUI	графічний інтерфейс користувача
IDE	Integrated development environment,
IT	Інформаційні технології
ОС	Операційна система
БД	База даних
API	Application Programming Interface
URL	Uniform Resource Locator
CT	Cocoa Touch
HK	Health Kit
MVC	Model-View-Controller

ВСТУП

Світ технологій розвивається безперервно і динамічно. З кожним роком з'являються нові концепції та їх реалізації, а існуючі розширюють свої функціональні характеристики і можливості. Не нова і мета цієї спіралі прогресу - зробити умови життя людини якомога комфортнішими і продуктивними. Однією з причин такого бурхливого зростання є широке поширення Internet і значне збільшення швидкості передачі даних. Як зазначають в Google, до 2008 року був «Інтернет людей», тепер настав «Інтернет речей», так як пристроїв, підключених до світової мережі стало більше ніж жителів планети. Завдяки цьому взаємодію їх з людиною стало можливо винести на зовсім інший рівень.

Розумний годинник – це нове слово у світі високотехнологічних гаджетів. Будучи як би продовженням Вашого смартфону вони можуть допомогти у самих несподіваних ситуаціях – скинути небажаний виклик, подивитися прогноз погоди, дізнатися температуру тіла та багато іншого. Дуже зручним є використання таких гаджетів у якості GPS-навігатора Сучасна розробка мобільних застосувань для ОС Android передбачає використання Java в якості основної мови програмування.

Одним з основних гравців у цій сфері є компанія Apple з її рішенням - Apple Watch. У першу чергу, Apple Watch це пристрій для контролю здоров’я та активності людини. Я думаю, хто любить носити годинник, не відмовилися б від них. Великим плюсом для зайнятих людей є можливість переглядати повідомлення. Годинник відображає нові sms і пошту, повідомлення із соціальних мереж, оповіщення від новинних програм. Усе це виводиться на екран за мить після появи на iPhone.

На даний момент іде активна розробка і вдосконалення технології «Розумний» годинник. Незважаючи на це, процеси стандартизації та глобалізації почалися порівняно недавно й на даний час існує безліч однотипних рішень, але з різною реалізацією, зав'язаною на виробників та їх технології. Завдяки зацікавленості основних гравців ринку в розвитку системи

«Розумний» годинник та інтеграції в неї своїх систем та сервісів, з'явилися зрушення у бік популяризації технології та спільна зацікавленість у ній як покупців, так і виробників.

Думки про розширення функціональності годинника з’являлися у виробників електроніки дуже давно. Прототипом сучасних Smart-годинників був перший електронний годинник, такий як Pulsar 1972 року випуску. У 80-х -90-х роках функціонал таких пристроїв було розширено. Деякі пам’ятають популярний на території колишнього СРСР годинник Montana з секундомером та будильником. У 2000 році компанія IBM презентувала Linux Watch. Після цього починаються сміливі експерименти у цій області - Smart Personal Object Technology, Fossil Wrist PDA та багато іншого. З вихідом операційної системи Андроїд - потужної, полнофункціональної та легковісної ОС, використання її на Smart Watch стало справою часу. Перший годинник з Андроїд на борту - WIMM One вийшов у 2011 році. А вже у 2012 Sony випускє у продаж Sony Smart Watch, які користуються популярністю й зараз. За останні два роки багато різних компаній спробували себе у розробці нових розумних годинників. Всесвітню відомість отримали – Pebble, Cokoo Watch, Samsung Galaxy Gear та багато інших.

Мета даної роботи - провести аналіз методів та принципів побудови додатків для розумного годинника Apple Watch. Вивчити концепції та принципи побудови додатків, їх переваги та недоліки, а також проектування власного додатку на основі досліджених даних.

1. ОГЛЯД APPLE WATCH, ОПЕРЦІЙНІ СИСТЕМИ IOS ТА WATCHOS

1.1. Apple Watch

Apple Watch — наручний годинник із додатковими функціями (розумний годинник) створений корпорацією Apple. Годинник оснащений всіма необхідними датчиками для спорту і здоров'я, які розташовані в нижній частині і взаємодіють безпосередньо з вашим тілом. Це робить Apple Watch незамінним помічником для спортсменів, який позбавляє сенсу існування цілу галузь фітнес-браслетів. Годинники підтримують роботу з цифровим асистентом Siri і можуть взаємодіяти з iPhone через 802.11b / g і Bluetooth 4.0. До речі, підтримуються всі моделі смартфона, починаючи з iPhone 5. Дисплей годинника є сенсорним і розпізнає не лише дотики, але і так звані «посилені торкання» (Force Touch), яким присвоєно різні завдання. Бездротові модулі Apple Watch також дозволяють здійснювати транзакції через нову платіжну систему Apple Pay і використовувати гаджет для дистанційного керування Apple TV.

Apple Watch має датчик натискання на екран і новий жест - посилений дотик. Крім цього, користувач годинника може налаштувати циферблат. Свайпи по циферблату дозволяють швидко отримувати доступ до різних функцій. Доступ до додатків реалізований неймовірно просто - десятки «кульок» чекають своєї черги прямо на екрані.

1.1.1. Історія технології Apple Watch

Коли в Apple лише почали розробку смарт-годин, керівники компанії заговорили про найсучаснішому гаджет для контролю над станом здоров'я. Він повинен був заміряти тиск, серцевий ритм, рівень стресу і робити ще багато чого корисного. Це підтверджували і ті, хто, так або інакше, мав відношення до даного проекту.

Розумні годинник Apple Watch були представлені 9 вересня 2014 року,

вони здатні повноцінно взаємодіяти з iPhone і iPad, а також облаждают широким набором функцій, пов'язаних з охороною здоров'я - вимірювати пульс, кількість пройдених за день кроків і так далі. У продаж надійшли 24 квітня 2015 року у 9 країнах (США, Канада, Великобританія, Австралія, Франція, Німеччина, Гонконг, Китай, Японія). Але до 18 червня 2015 року годинники не були доступні у роздрібній торгівлі в магазинах, їх можна було замовити тільки за попереднім замовленням, зробленим через інтернет-магазин. Ціна становить від 349 дол. США.

1.1.2. Принцип роботи

Дисплей Apple Watch вимкнений, поки ви на нього не дивіться. Як тільки ви повернете руку до очей, екран активується (на відміну від деяких моделей на базі Android, апарат Apple відразу ж знову піде в режим очікування, як тільки ви відвернеться руку з годинником). Якщо потрібно негайно «згасити» дисплей, досить накрити його долонею.

Набір SMS - за допомогою голосової диктування (російський розпізнається), нормальна клавіатура не вміщується на маленькому екранчику. Можна використовувати готовий шаблон.

Вбудований фітнес-трекер (аналог Fitbit) - вимір пульсу, підрахунок кроків і калорій. Taptic Engine дозволяє відправити іншим власникам годин від Apple набір «торкань» або своє серцебиття.

Помічник Siri (викликається натисканням і утриманням коліщатка).

Поштовий клієнт з обмеженою функціональністю (текст деяких листів він показує повністю, а при спробі прочитати інші радить відкрити iPhone і запустити відповідну програму там). Відповісти на лист за допомогою годинника можна, можна лише видалити його, відзначити як непрочитане або встановити прапорець. Браузера в Apple Watch немає, номеронабирателя теж.

В Apple Watch можна завантажувати до 2 ГБ музики. Apple Watch не призначені для ігрового використання. У січні 2015 року низка компаній, включаючи TapSense і InMarket, оголосили про плани по запуску на дисплеях

годин гіперлокальной реклами.

При вмиканні годинник відразу просять приєднатися до свого «родича», телефону iPhone - просять піднести камеру смартфона до зображення на екрані Apple Watch. Потім на пристрій переносяться всі додатки, встановлені на iPhone і адаптовані для годин. Підключення двох пристроїв йде через Bluetooth. Крім того, Apple Watch можуть обмінюватися даними за допомогою Wi-Fi, якщо обидва девайса підключені до однієї бездротової мережі.

1.1.3. Основні характеристики системи

Apple Watch працюють на операційній системі Watch OS. Годинники мають поворотне коліщатко для прокрутки або збільшення. Натискання на нього дозволяє повернутися до домашнього екрану. Дисплей має розмір 38x38 або 42×42 мм, і здатний розрізняти натискання і дотик.[3] Апарат не має роз'ємів, підзарядка батареї відбувається за допомогою індуктивного адаптера з магнітною фіксацією. На нижній стороні годинників можуть бути розташовані світлодіоди і фотодіоди для вимірювання пульсу. Також є в наявності інтерфейс NFC, який дозволить робити безконтактну оплату через систему Apple Pay. Девайс доступний у трьох «колекціях»: Apple Watch Sport, Apple Watch та Apple Watch Edition (виконаний із жовтого і рожевого золота).

Великим плюсом для зайнятих людей буде і можливість переглядати повідомлення. Годинник показує нові sms і пошту, повідомлення із соціальних мереж, оповіщення від новинних програм. Усе це виводиться на екран за мить після появи на iPhone. Очевидно, що обробляти пошту і повідомлення значно швидше на екрані смартфона. Але часом навалюється так багато всього, що діставати щоразу iPhone просто незручно. Лише один погляд на годинник дає можливість зрозуміти: смартфон завібрував у кишені через важливий електронний лист чи знову через коментар на Facebook, вивчення якого можна відкласти.

1.2. Операційна система Watch OS

WatchOS — операційна система від корпорації Apple, що була спеціально розроблена для Apple Watch. Операційна система підтримує сторонні додатки, має функцію Siri, мапи, календар, нагадування, блокнот, і власне годинник. WatchOS розроблена виключно під апарат Apple Watch і не пристосована для роботи на будь-якому іншому пристрої. Вперше випущена 24 квітня 2015 року. watchOS підтримує застосунки, що розроблені за допомогою WatchKit. Головне меню системи виконане у вигляді Carousel, що надає зручний доступ до любого застосунку.

1.2.1. Загальна інформація про Watch OS

Ваш додаток на годинник та розширення на iPhone WatchKit працюють в тандемі для повноцінної роботи інтерфейсів. Коли ваш додаток на годиннику, watchOS завантажує призначений для користувача інтерфейс з програми Watch на мобільному додатку. Він також запускає свій додатковий WatchKit, який управляє вмістом інтерфейсу. Для додатку Watch і здійсненних повідомлень, розширення WatchKit обробляє взаємодії користувача з інтерфейсом. Рисунок 1.1 показує взаємозв'язок між додатком Watch, розширенням WatchKit і додатком IOS.

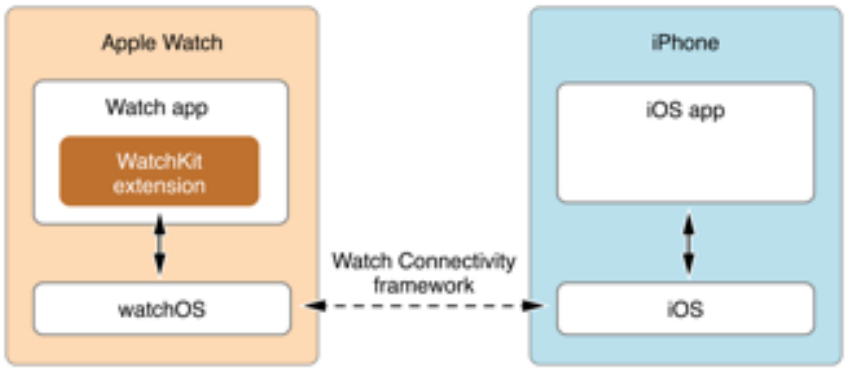

Рисунок 1.1 - Взаємодія додатків iOS та WatchOS

Як додаток IOS, програма “Годинник” складається з одної або більше

сцен, де кожна сцена представляє повний екран контенту. Кожна сцена є результатом керування одного об'єкта контролера інтерфейсу в розширенні WatchKit. Контролер інтерфейсу є екземпляром WKInterfaceController класу. Контролер інтерфейсу в watchOS служить для тієї ж мети, що і контролер уявлення в iOS. Він управляє контентом на екрані і реагує на дії користувача з цим вмістом. На відміну від контролера View, контролер інтерфейсу не керує фактичним видом інтерфейсу. Ці View управляються для вас watchOS.

Головний інтерфейс програми, як правило, містить кілька сцен, на кожній з них відображаються різні типи інформації. Тільки одна сцена відображається на екрані, додаток представляє нові сцени у відповідь на дії користувача. Наігаційний контролер визначає яким чином представлені сцени. Додатки можуть відображати сцени модально. Для отримання інформації про те , як представити нові сцени, см навігаційний інтерфейс .

1.2.2. Архітектурні аспекти

Для того щоб запустити додаток, необхідно обрати його з поміж інших додатків. Запуск додатку призводе до початку роботи життєвого циклу додатку. Є можливість запуску додатку у момент натходження сповіщення чи сигналу з годинника. Під час запуску watchOS автоматичо грузить storyBoard. Після того як сцена завантажена, watchOS дає запит на створення відповідного контролеру взаємодії, яким ми користуємось для підготовки сцени до роботи з користувачем. Розглянемо більш детально процес запуску додатку на інформаційному рисунку, який був опублікований компанію Apple на офіційній сторінці, присвяченій детальному опису роботи додатку на прикладному рівні.

На рисунку 1.2 можна побачити що після запуску додатку відбуважться завантаження інтерфейс контролеру. А вже після, іде ініціалізація UI і безпосередньо відображення сцени на дисплеї Apple Watch.

Перед тим як ініціалізувати UI, контролер повинен бути активований, для цього іде виклик awakeWithContent. Використовуйте willActive тільки для відображення новостворених або оновленної інформації.

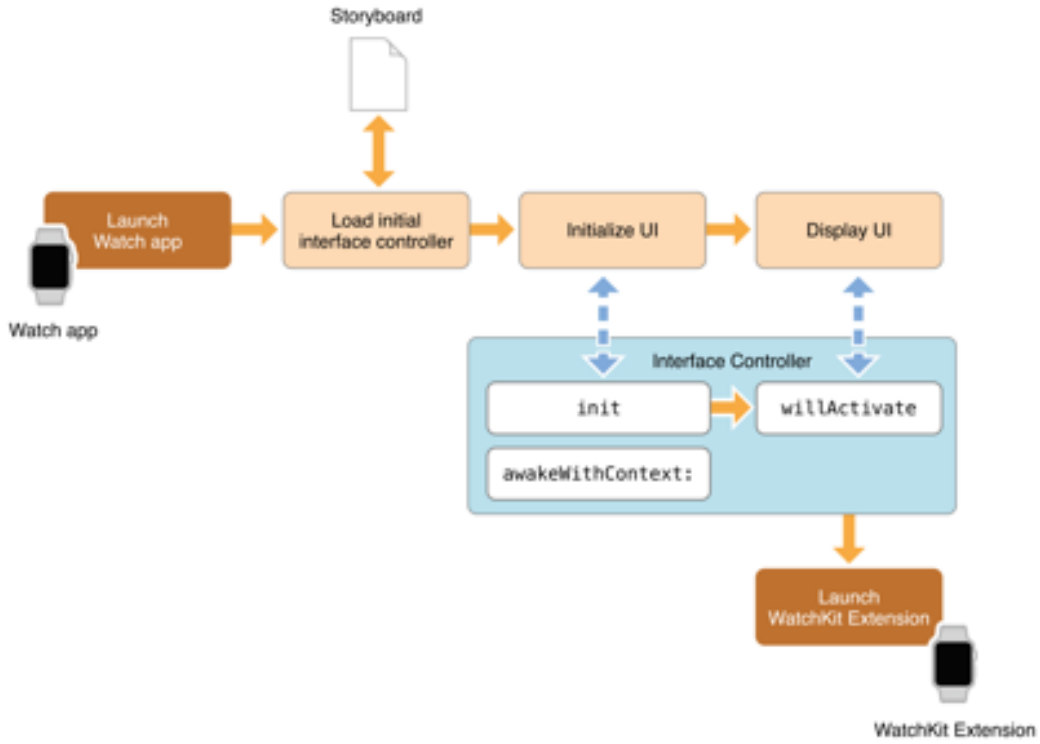

Рисунок 1.2 - Ініціалізація додатку

Ми розглянули основні єтапи життєвого циклу. Також зверніть увагу на стани додатку. В залежності від вихідних і вхідних умов ці стани відповідно змінюються.

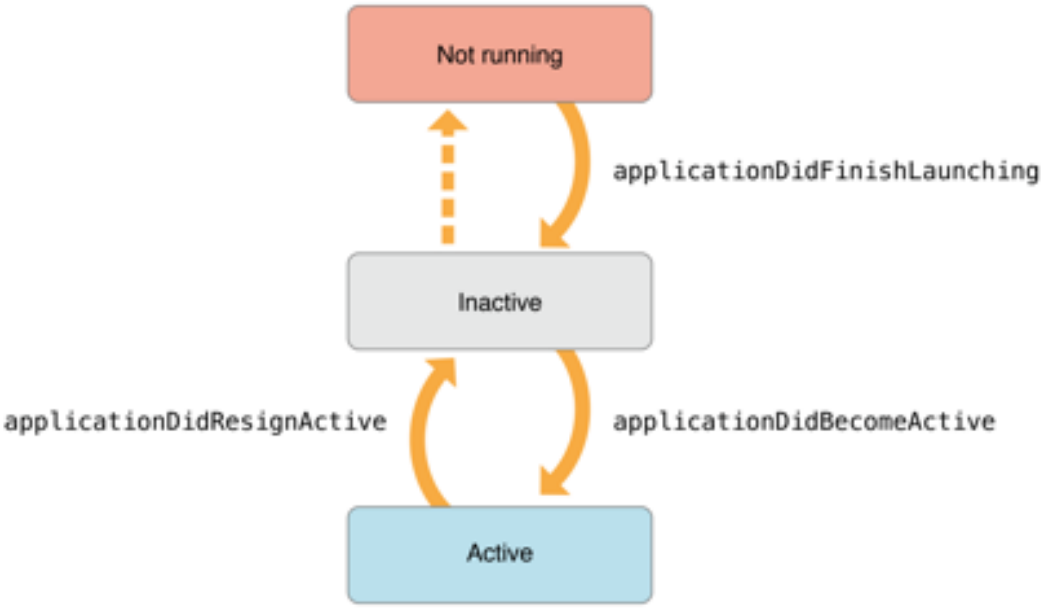

Рисунок 1.3 - Демонстрація станів

1.3. Операційна система iOS

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.

Рисунок 1.4 - Вигляд першої iPhoneOS 1.0

Внутрішній акселерометр використовуються деякими програмами для реагування на струшування пристрою, яке є також загальною командою

скасування, або обертати пристрій у трьох вимірах, що є загальною командою перемикання між книжковим та альбомним режимами.

Станом на 31 травня 2011 року інтернет-магазин App Store містить понад 500 тисяч застосунків для iOS, які були завантажені понад 15 мільярдів разів.

Рисунок 1.5 - Головний єкран iOS

Станом на червень 2016 року, iOS становив 23,4 % ринку операційних систем для смартфонів, другий після Android.

1.3.1.Історія

В якості операційної системи iOS була представлена з iPhone на

Macworld Conference & Expo 9 січня 2007 року і випущена в червні того ж року. Спершу, Apple не вказувала її ім'я, просто заявивши, що "iPhone використовує OS X". Спочатку, сторонні програми не підтримувалися.

Стів Джобс заявив, що розробники можуть створювати веб-програми, що „будуть вести себе, як рідні програми на iPhone“. 17 жовтня 2007 року Apple оголосила, що рідний SDK знаходиться в стадії розробки, і що вони планують поставити його „в руки розробників у лютому“. 6 березня 2008 року Apple випустила першу бета-версію, а також нове ім'я для операційної системи: iPhone OS.

Продажі мобільних пристроїв Apple викликали інтерес до SDK. Apple також продала більше одного мільйона iPhones під час курортного сезону 2007. У червні 2010 року, Apple перейменувала iPhone OS на iOS. Назва iOS користувалася компанією Cisco вже більше десяти років на маршрутизаторах Cisco. Для того, щоб уникнути будь-якого потенційного позову, Apple ліцензувала торгову марку iOS у Cisco.

1.3.2.Основні можливості iOS

- Фотографії — пошук по місцю і часу, а також нові можливості редагування фотографій.
- Повідомлення — можливість відправляти аудіозаписи і карту з місцезнаходженням в діалог. Тепер можна швидше переслати щойно зняті відеозаписи та фото, а також встановлювати функцію «Не турбувати» на потрібні діалоги.
- Quicktype — передбачає можливі за змістом слова, засновані на розмові, під час друкування пропозицій.
- Family Sharing — можна позначати до шести контактів як членів сім'ї і швидко ділитися з ними фотографіями, покупками додатків і музики, місцеположенням та іншим.
- iCloud Drive — можливість зберігати в хмарі будь-які види файлів з подальшим їх редагуванням на різних пристроях.
- HealthKit — організація відомостей про здоров'я в одному додатку.

- Spotlight — пошук став більш глобальним і тепер можна шукати різну інформацію і за межами телефону / планшета.
- Віджети- тепер в Центр сповіщеннях можна встановлювати віджети від сторонніх розробників.
- Сторонні клавіатури — вперше компанія Apple дозволила стороннім розробникам створювати альтернативні клавіатури, які зможуть замінити стандартну клавіатуру.

1.4. Висновки до розділу 1

Apple Watch являє собою потужний пристрій який ставить собі на меті якісний і простий у використанні інтерфейс. Компанія Apple суворо диктує правила яких розробники повинні дотримуватись при розробці графічного інтерфейсу для розумних годиників.

Постійна зв’язка з телефонм дає можливість синхронізувати дані майже миттєво з системою. Дуже корисними виявили собе додатки камера. За допогою котрої можливо зробити фото знімки не торкаючись телефона.

Операційна системи WatchOS розвивається швидкими темпами. Вона являє собою приклад операційної системи, яка впер за все націлена на користувача. Всі додатки розроблені відповідно до потреб людей. Кожна деталь бездоганно протестована і працює.

Завдяки жорсткій зв’язці операційних систем ми маємо можливість використовувати iPhone навіть без користування ним. СМС які надходять на телефон можна переглядати на годинику, також є модливість відповісти у вигляді тексту або смайлику.

2. АНАЛІЗ ТА ПОРІВНЯННЯ МОВ ПРОГРАМУВАННЯ OBJECTIVE C ТА SWIFT

2.1. Objective C

Рисунок 2.1 - Логотип мови 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 — 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 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.

2.2. Swift

Рисунок 2.2 - Логотип мови Swift

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. Основним застосуванням Swift є розробка користувацьких застосунків для MacOS X і Apple iOS з використанням тулкіта Cocoa і Cocoa Touch. При цьому Swift надає об'єктну модель, сумісну з Objective-C. Сирцевий код мовою Swift може змішуватися з кодом на C і Objective-C в одному проекті.

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.

2.3. Порівняльна характеристика Objective C та Swift

Мови програмування не вмирають швидко, а студії розробники, які чіпляються за згасаючі парадигми, вмирають. Якщо ви розробляєте програми для мобільних пристроїв і не вивчали Swift, то знайте: Swift не тільки витісняє Objective-C, коли мова йде про розробки додатків під OSX, iOS і WatchOS, але і в недалекому майбутньому замінить C для внутрішнього програмування, для Apple платформ .

Завдяки кільком ключовим особливостям, Swift має потенціал стати єдиною мовою програмування, для створення захоплюючих, гнучких, споживацьки-оріентрірованних 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 потрібно розкрити. Можна сказати, що Swift «призначений для зльоту від “hello, world” до цілої операційної системи.

Таблиця 2.1 - Порівняння мов програмування

	Swift	**Objective-C**
Спеціальні символи	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.	Для диференціації ключових слів і типів від C типів, Objective-C вводить нові ключові слова, використовуючи символ @.
Наявність крапки з комою в кінці строки	Swift переглядає загальноприйняті умови успадкування. Не повинні ставиити крапку з комою.	Обов’язково прийнятий стандарт.
Дужки після if/else умови	Ні	Так
Читабельність коду	Читабельність коду в Swift полегшує роботу для програмістів JavaScript, Java, Python, C #, C ++	Складна дот сприйняття
Вимога двох файлів .h .m	Свіфт поєднує в собі заголовок Objective-C (.h) і файли реалізації (.m) в одному файлі коду (.swift)	Обов’язкова. У Objective-C ви повинні вручну синхронізувати імена методів і коментарі між файлами.
Витрата часу на написання коду	Низька	Велика

Таблиця 2.1 (закінчення) - Порівняння мов програмування

	Swift	**Objective-C**
Спосіб обробки nil (NULL)	Опціональні типи дають можливість існування в Swift коді nil опционального значення, що говорить про можливість створення помилки компілятора при написанні поганого коду.	У Objective-C нічого не трапиться якщо ви спробуєте викликати метод зі змінною покажчика nil (неініціалізованих). Вираження рядок коду стає нездійсненними і може здатися, що вираз не впаде, але насправді буде повно багів.
Визначення типу	Так, у випадку опціональних типів	Визначення є обов’язковим, у випадку NULL можуливе вінікнення помилок
ARC - Автоматичний підрахунок ссилок	ARC обробляє всі управління пам'яттю під час компіляції, і витрати, які пішли б на управління пам'яттю, тепер можуть бути сфокусовані на ключевій логіці	У Objective-C, ARC підтримується всередині Cocoa API і об'єктно-орієнтованого коду; але він не доступний для C коду і API
Робота ARC рівні	На процесуально і на об'єктно-орієнтованому коді	На рівні Cocoa API
Робота зі строками	Swift приймає особливості сучасної мови, наприклад додавання 2-х рядків разом з оператором «+»	У Objective-C, працюючи з текстовими рядками, вам доводиться бути багатослівним, і щоб об'єднати дві частини інформації, буде потрібно безліч кроків.
Система типів	Визначення типу автоматично	Строготипізована мова
Спеціальні маркери рядків для ввиведення символів	Підтримка інтерполяції рядків	(% S,% d,% @)

2.4. Порівняння швидкості роботи

Таблиця 2.2 - Перемішування 1000000 інтових обєктів

Версія	**Objective-C**	**Swift**	**Device (iOS Version)**
Swift 1.1	0.296 seconds 1% STDEV	1.616 seconds 2% STDEV	iPhone 6 (8.3)
Swift 1.2	0.262 seconds 1% STDEV	0.523 seconds 1% STDEV	iPhone 6 (8.3)
Swift 2.0	0.271 seconds 2% STDEV	0.285 seconds 2% STDEV	iPhone 6 (8.3)
Swift 2.1	0.305 seconds 1% STDEV	0.336 seconds 5% STDEV	iPhone 6 (9.2)
Swift 2.2 Snapshot	0.256 seconds 1% STDEV	0.284 seconds 7% STDEV	iPhone 6 (9.2)

Таблиця 2.3 - Цикл видалення 1000000 елементів з масиву Int

Версія	Objective-C	Bridgeable Swift	Swift	Device (iOS Version)
Swift 1.1	0.010 seconds 38% STDEV	-	0.003 seconds 49% STDEV	iPhone 6 (8.3)
Swift 1.2	0.005 seconds 10% STDEV	-	0.001 seconds 20% STDEV	iPhone 6 (8.3)
Swift 2.0	0.005 seconds 23% STDEV	-	0.001 seconds 10% STDEV	iPhone 6 (8.3)
Swift 2.1	0.005 seconds 35% STDEV	-	0.001 seconds 9% STDEV	iPhone 6 (9.2)
Swift 2.2 Snapshot	0.005 seconds 24% STDEV	-	0.000 seconds 106% STDEV	iPhone 6 (9.2)

Розглянемо тепер стандартні операції додавання, видалення та вставку елементів у масив.

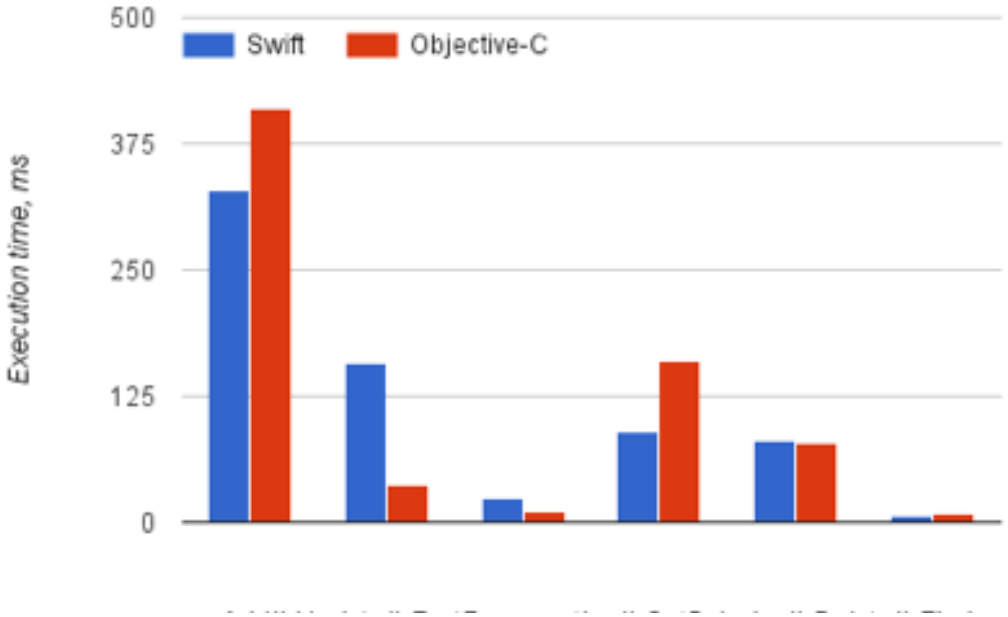

Рисунок 2.3 - Порівняльний графік стандартних операцій. Array/NSArray

- Операції додавання, оновлення та читання в масиві Swift набагато швидше, ніж в NSArray.
- Свіфт не швидший, ніж Objective-C.
- Існує аномалія масиву "видалити індексні" Результати: Видалення даних в Swift в 10 разів повільніше, ніж в Objective-C.

В Swift, всі класи створюються під час компіляції. Методи не можуть бути додані на льоту і всі типи відомі до часу виконання. Так як все відомо заздалегідь, компілятор може оптимізувати код без будь-яких проблем.

Objective-C, з іншого боку, не може оптимізувати так само ефективно, так як всі динамічні мови працюють повільніше, ніж статичний.

Причина в тому, статична перевірка типів в Swift і динамічний характер Objective-C.

-

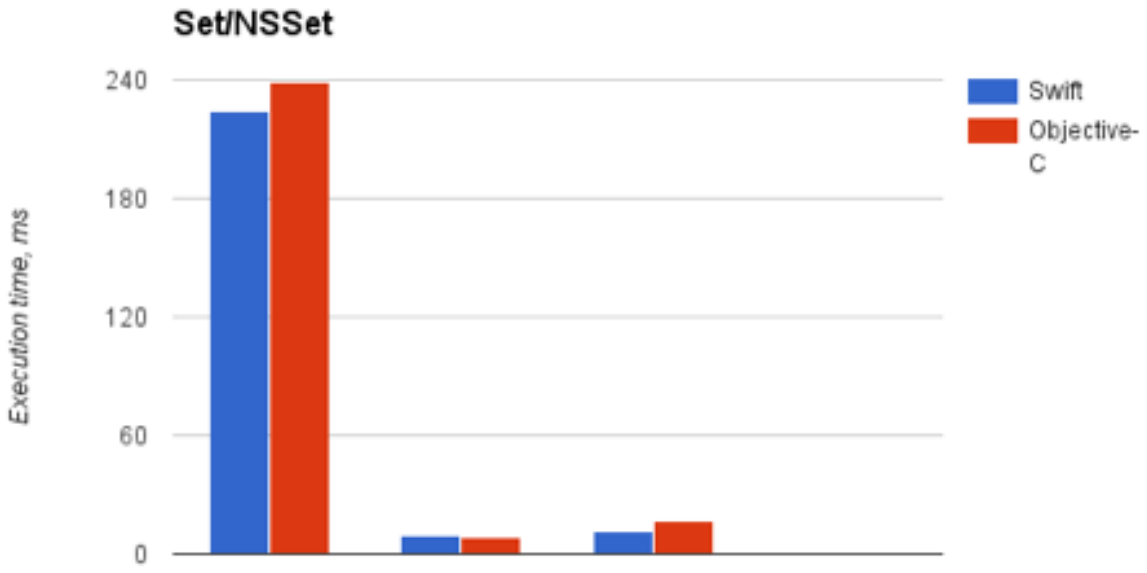

Рисунок 2.4 - Порівняльний графік стандартних операцій. Set/NSSet

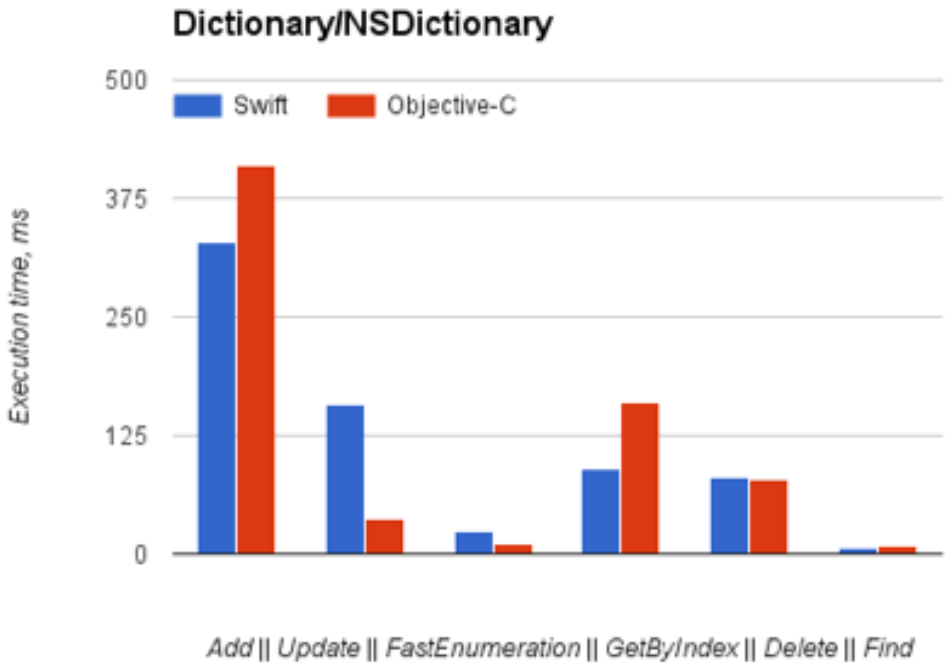

Рисунок 2.5 - Порівняльний графік стандартних операцій. Dictionary/ NSDictionary

	Swift	Objective-C
Class File	One file	Two files (example.h,example.m)
Declaring an int variable	var highScore: Int var playerName: String	int highScore; NSString *playerName;
Assigning to a variable	highScore = 1000 playerName = "Joe Chan"	highScore = 1000; playerName = @"Joe Chan";
Declaring and assigning constants	let skyColor = "Blue" let daysInYear = 365	NSString *const skyColor = @"Blue"; int const daysInYear = 365;
Defining a class	class CustomClass { }	**Header file (.h)** @interface CustomClass : NSObject @end **Implementation file (.m)** @implementation CustomClass @end
Creating a new class instance	var instance = CustomClass()	CustomClass *instance = [[CustomClass alloc] init]; CustomClass *secondInstance = [CustomClass new];

Method	func getOdometerReading() -> Int {
	return 50000
	}
		Header file (.h)
		- (int) getOdometerReading;
		
		Implementation file (.m)
		- (int) getOdometerReading
		{
		return 50000;
		}

Calling a method	var myCar = SpecificCar() myCar.getOdometerReading()	SpecificCar *myCar = [[SpecificCar alloc] init]; [myCar getOdometerReading];
Declaring a method with multiple parameters	func changeEngineOil(oil:Int, transmissionFluid:Int) { }	- (void)changeEngineOil:(int)oil transmissionFluid:(int)fluid { }
Calling a method with multiple parameters	myCar.changeEngineOil(10, transmissionFluid:10)	[myCar changeEngineOil:10 transmissionFluid:10];

Рисунок 2.6 - Порівняння сигнатури методів Swift та Objective-C

2.5. Висновки до розділу 2

Очевидно, що впровадження Swift помітно позначиться на всій програмної екосистемі Apple. Оскільки розробка додатків для iOS і OS X стає все простіше і легше, багатьом професіоналам в інших мовах і платформах захочеться спробувати свої сили на новому полі діяльності.

Це означає, що до мобільних і комп'ютерних платформ Apple буде залучено більше число розробників. Більше розробників - це більше додатків і більше вибору для споживача. А от питання про якісний рівень такого софта доведеться залишити відкритим. Поки що Swift занадто нова технологія і вона ще не освоєна навіть професійними програмістами для пристроїв Apple.

В кінцевому рахунку, Swift найбільш доступний повнофункціональний мова програмування, яка дозволить розробникам не тільки створювати додатки, але і програмувати вбудовані системи, типу нової низько споживаної Apple Watch, на багато років вперед.

3. AUTO-LAYOUT ЯК ОСНОВНИЙ ІНСТРУМЕНТ ДЛЯ СТВОРЕННЯ ГРАФІЧНОГО І МУЛЬТИМЕДІЙНОГО ІНТЕРФЕЙСУ

Основна мета розробника це створення програмного забезпечення, яке задовольняє замовника і кінцевого споживача. Додаток має бути написан якісно, оптимально і без помилок. Швидка робота і простота використання програми є запорукою тривалого користування вашим додатком клієнтами.

Зовнішній вигляд це основа і початок вашого застосування. Зробивши зрозумілий і інтуїтивний інтерфейс ви отримаєте позитивну реакцію від користувачів. І ваші користувачі винагородять вас за це.

Побудова додатків є складн і не тривіальним завданням. Це завдання ускладнюють такі чинники:

- Різні розміри дисплеїв.
- продуктивність телефонів
- конфігурація систем
- мультизадачність
- локалізація

Для вирішення однієї з таких задач, а саме "Різні розміри дисплеїв" існує інструмент, який інтегрований в середу розробки Xcode, Auto Layout. Якщо сказати в цілому, то Auto Layout це чорна коробка, яка приймає в себе величезну кількість вхідних даних, у вигляді Constraints (обмежень) і трансформує їх у рівняння, використовуючи лінійну алгебру отримуємо набір вікон. Це дасть зовнішній вигляд який ми визначили. Auto Layout дуже потужний і гнучкий інструмент для розробки графічних інтерфейсів

3.1. Основна інформація про Auto-Layout

Auto Layout використовується для розміщення вікон на робочому екрані з певними обмеженнями. Для прикладу, розглянемо тривіальну задачу, яка продемонструє проблему, яку вирішує Auto Layout.

Рисунок 3.1 - Розположення елемнту

Завдання 1: Розмістити кнопку на екрані, яка буде знаходитися внизу по центру екрана

Рішення без Auto Layout:

Розмістити кнопку на екрані з координатами точки такими, які б задовольняли нашій задачі, для прикладу якщо екран 50 на 100, то кнопку на позицію (x = 25, y = 90).

Проблема:

До iOS 5 існувало 2 основних розміри екранів 3.5 і 4 дюйма, що давало розробникам описувати розташування елементів і вікон вручну. Але з появою iOS 5 з'явилися різні розміри екранів 4.7, 5.5, 7, 9.1 і в майбутньому ще інших різних видів діагоналі. Опис розташування елемента для кожного екрану було б не раціональної завданням.

Рішення з Auto Layout:

Ми вказуємо правила (constraints) розташування наше кнопки, а саме:

- Кнопка повинна бути розташована по центру горизонту екрану щодо батьківського вікна

- Відступ від нижнього краю повинен бути 0 пунктів щодо батьківського вікна

Тепер опишемо ці правила як лінійне рівняння:

- Button.centerX = SuperView.centerX
- Button.bottom = SuperView.bottom - <padding>

І ці умови носять назву Constraints (обмеження).

Описавши правила за допомогою Constraints, в справу вступає Auto Layout який обчислює розташування вікон автоматично.

Ці 2 правила ми задаємо в якості параметрів для нашого Auto Layout елементів. Таким чином, не залежно від ширини екрану ми завжди будемо впевнені, що кнопка буде перебувати по горизонту в центрі, а також відступ від нижнього краю дорівнює нулю.

При зміні вмісту вашого додатку, новий зміст може зажадати нову розмітку. Це зазвичай відбувається в додатках, які відображають текст або зображення. Наприклад, новинні додатоки необхідно налаштувати так, що б в залежності від типів і обсягу статті, змінювалося кількість місце, яке потрібно для відображення на екрані. Точно так же, фото колаж повинен обробляти широкий діапазон розмірів зображення і пропорцій.

Цей приклад продемонстрував основну ідею Auto Layout, яка вирішує проблему різних розмірів екранів. Без прив'язки до числовим параметрам, ми делегуємо обов'язки системі, яка гарантує розміщення елементів згідно з заданими правилами.

3.2. Налаштування Auto-Layout для роботи

Для створення інтерфейсу ми повинні створити проект в якому ми будемо виконувати завдання:

Алгоритм дій:

1. Створимо новий проект проект в Xcode

Рисунок 3.2 - Стандартний екран Xcode

2. Меню створення проекту, вибираемо Create a new Xcode project

2.1.Виберемо Single View Application

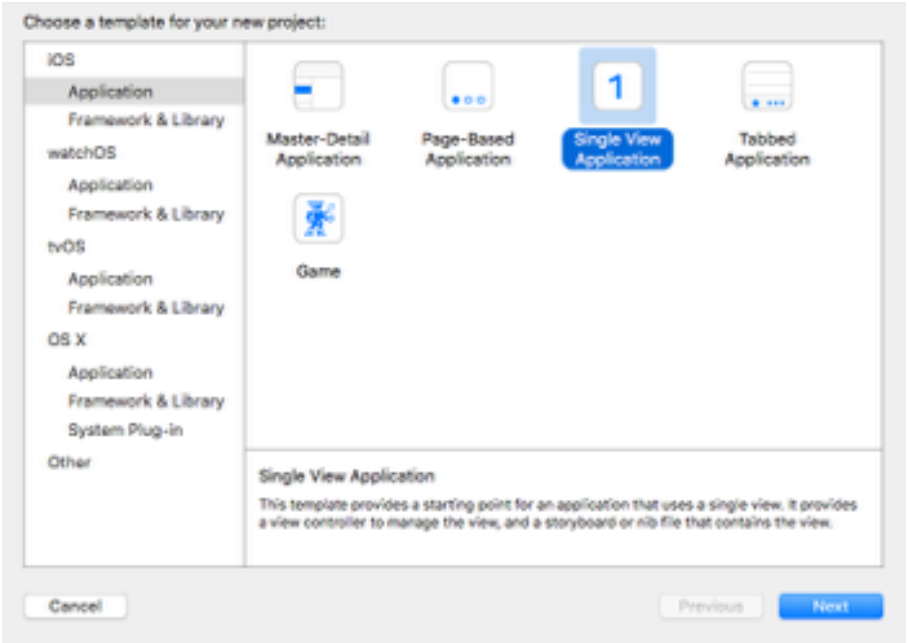

Рисунок 3.3 - Вибір проекту

2.2.Виберемо ім'я, унікальний ідентифікатор, і мова програми

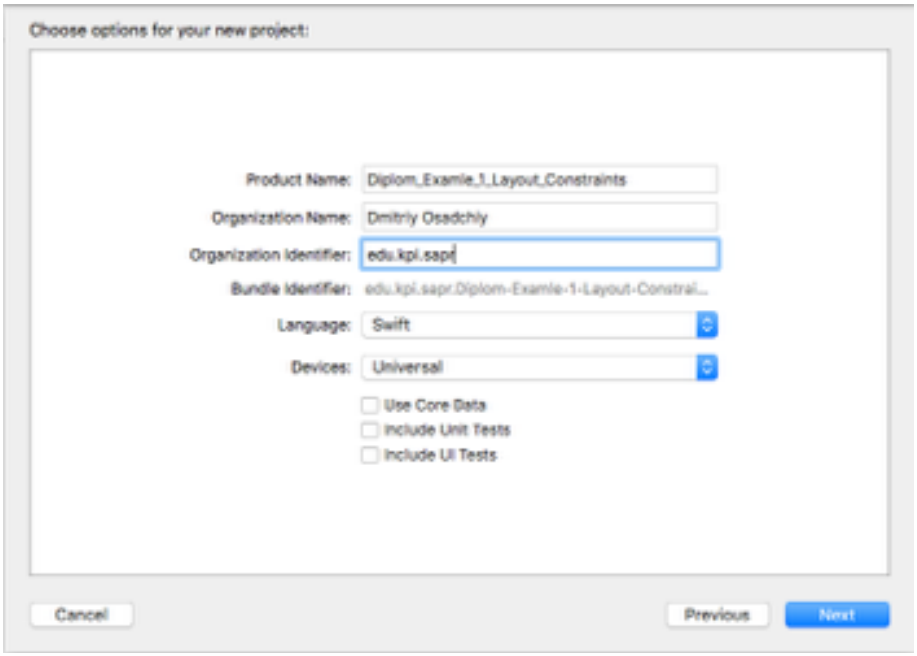

Рисунок 3.4 - Вибір унікального ідентифікатору

2.3.Збережемо проект в потрібну нам папку

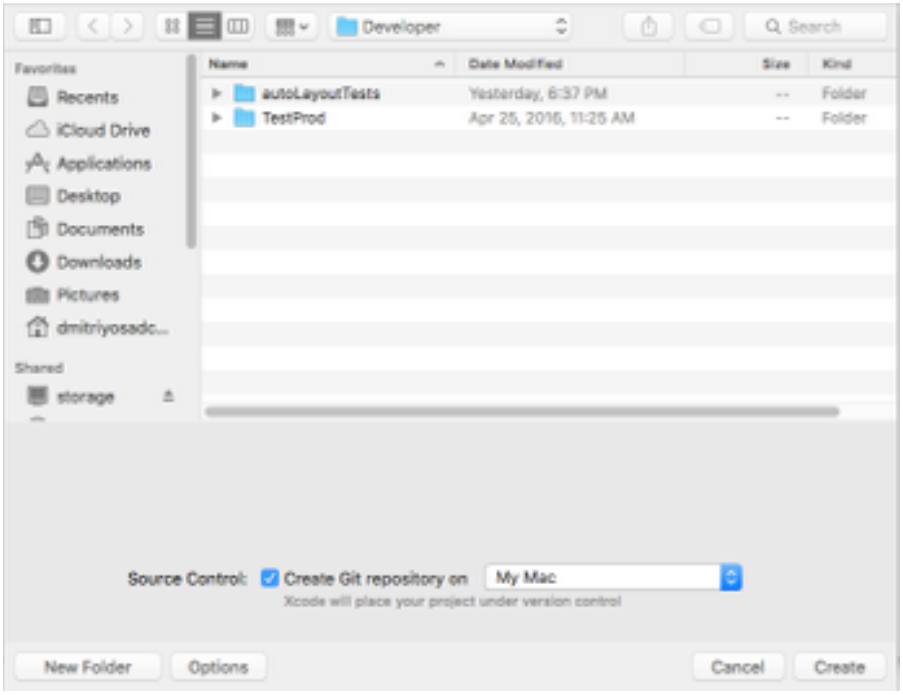

Рисунок 3.5 - Екран збереження проекту

3.3. Реалізація іструменту з допомогою GUI інтерфейсу

Для того щоб мати можливливість використовувати створення обмеження в програмі Xcode, ми створили проект в пункті 3.2

Наступні наші кроки для вирішення задачі будуть:

1. Оберемо вкладку Main.storyboard

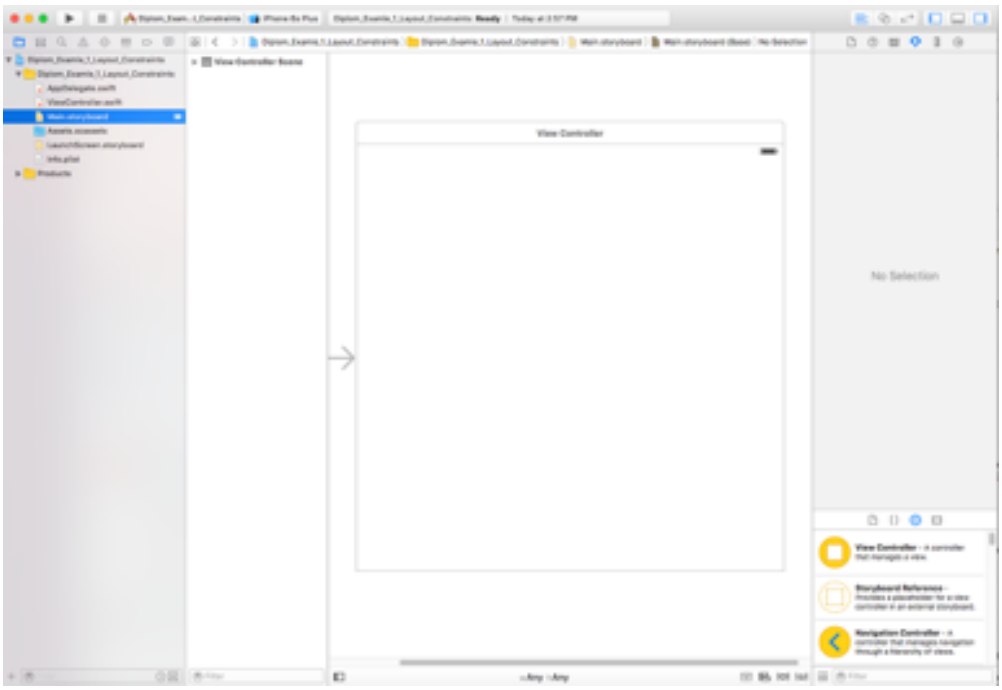

Рисунок 3.6 - Екран Interface Builder

Auto Layout використовується для розміщення вікон на робочому екрані з певними обмеженнями. Для прикладу, розглянемо тривіальну задачу, яка продемонструє проблему, яку вирішує Auto Layout.

2. Виконаємо Завдання 1 із вступу Auto Layout. Потрібно додати кнопку з певними правилами розташування.

2.1.Додамо кнопку. Перетягнувши її з нижнього правого меню, для полегшення завдання введемо "button" в пошук

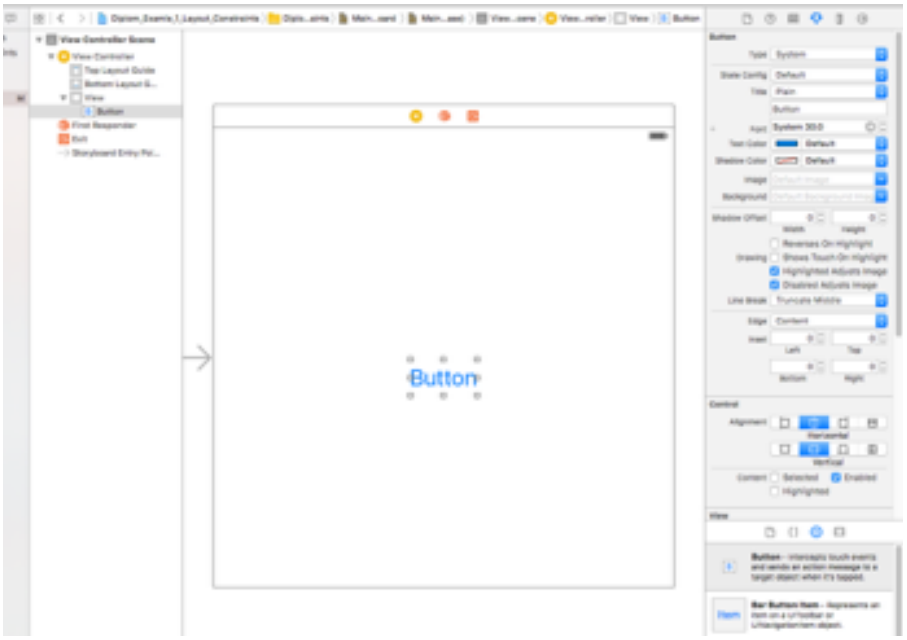

Рисунок 3.7 - Вибір елементу

3. Введемо перше правило, розміщення елемента в центрі по горизонту. Для цього натисніть на нижню іконку в робочому меню Interface Builder. Натисніть на осередок Horizontally in Container і Add Constraint внизу, як показано на зображенні 3.7

4. Введемо друге правило. Вкажіть що висота від нижнього краю буде дорівнювати нулю. Для цього натисніть на нижню іконку в робочому меню Interface Builder.Клікніте на нижню паличку у верхній частині вікна і натисніть Add Constraint, як показано на зображені 3.8

5. Тепер запустимо додаток в Preview на різних пристроях і переконаємося що Auto Layout працю, рисунок 3.9

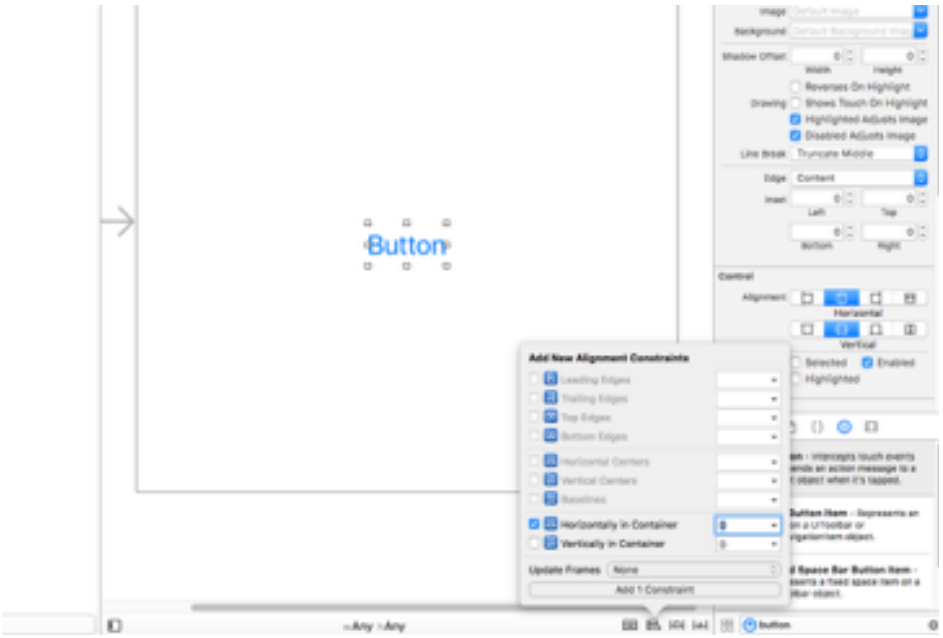

Рисунок 3.8 - Вибір Constraints

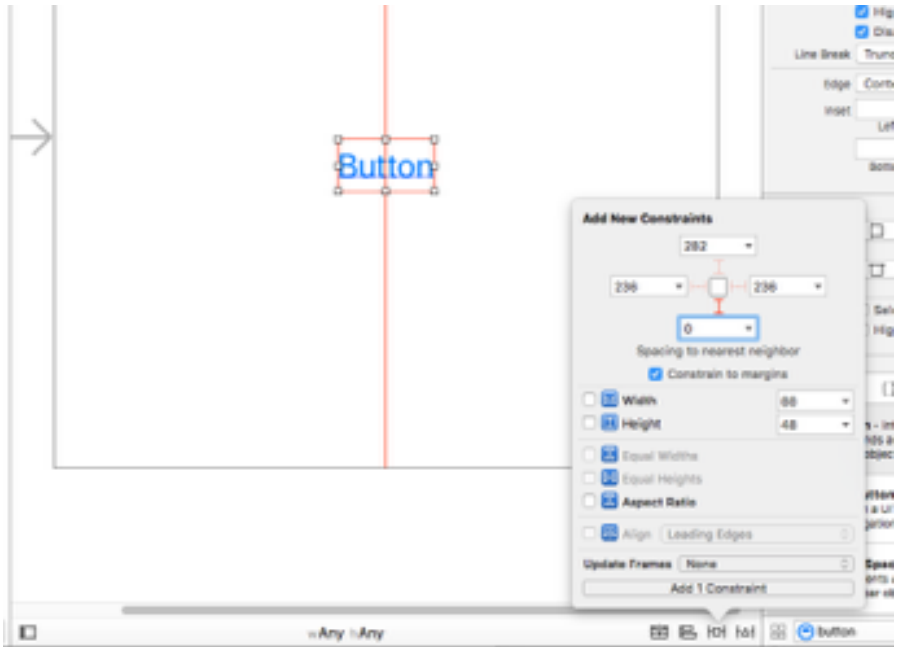

Рисунок 3.9 - Призначення Constraints

Рисунок 3.10 - Тестування інтерфейсу

Як ми бачимо все працює. Для достовірності роботи ми запустили додаток на різних пристроях (Рисунок 3.10)з різним розміром екрану

3.4. Программа реалізація інструменту без допомоги GUI

Вирішення задачі за допомогом программної реалізації, більш детальну інформацію ми отримаємо у розділі 3.7

```
//

import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }
    @IBOutlet weak var button: UIButton!


    override func viewWillAppear(animated: Bool) {
        super.viewWillAppear(animated)
        view.translatesAutoresizingMaskIntoConstraints = false

        let centerConstraint = NSLayoutConstraint(item: button, attribute: .CenterX, relatedBy: .Equal, toItem: view, attribute: .CenterX, multiplier: 1, constant: 0)
        let bottomConstraint = NSLayoutConstraint(item: bottomLayoutGuide, attribute: .Top, relatedBy: .Equal, toItem: button, attribute: .Bottom, multiplier: 1, constant: 0)
        view.addConstraints([centerConstraint, bottomConstraint])

    }
    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


}
```

Рисунок 3.11 - Приклад коду програмної реалізації

Запустимо програму і порівняємо результати з пунктом 3.4

Рисунок 3.12 - Приклад результату програмної реалізації

3.5. Розгляд джерел які впливають на зміну елементів

Підхід побудови інтерфейсу на основі обмежень, дозволяє будувати динамічні інтерфейси які реагують на внутрішні і зовнішні зміни.

Внутрішні зміни відбуваються коли розмір або форма батьківського View ієрархії змінилась. З кожною зміною ви повинні оновлювати розмітку вашої View ієрархії для використання вільного простору.

Джерела які можуть бути причиною внутрішніх змін:

- Користувач змінив розмір вікна.
- Відбувається переворот екрану.
- Ви хочете підтримувати різні розміри View.
- Активна запис або розмовна панель інструментів на верху екрану.

Внутрішні зміни відбуваються коли розміри View змінюються.

Джерела внутрішніх змін:

- Зміст View змінилося.
- Користувач змінив мову.
- Користувач змінив розмір шрифту в локальних настройках операційної системи.

При зміні вмісту вашого додатку, новий зміст може зажадати нову розмітку. Це зазвичай відбувається в додатках, які відображають текст або зображення. Наприклад, новинні додатоки необхідно налаштувати так, що б в залежності від типів і обсягу статті, змінювалося кількість місце, яке потрібно для відображення на екрані. Точно так же, фото колаж повинен обробляти широкий діапазон розмірів зображення і пропорцій.

Інтернаціоналізація є процесом який потрібн щоб ваш додаток був здатен адаптуватися до різних мов, регіонам і культурам. Схема інтернаціоналії додатку, потрібно приймати ці відмінності до уваги і правильно відображати

дані на всіх мовах, які підтримує додаток.

Інтернаціоналізація має три основні шляхи впливу на розмітку Constraints в Auto Layout. По-перше, коли ви переводите свій користувальницький інтерфейс на іншу мову, мітки вимагають різну кількість простору. Німецький, наприклад, як правило, вимагає значно більше місця, ніж англійську мову. Японська мова часто вимагає набагато менше місця.

По-друге, формат, який використовується для представлення дати і часу можуть змінюватися від регіону до регіону, навіть якщо мова не змінюється. Хоча ці зміни, як правило, більш тонкі, ніж зміни мови, призначений для користувача інтерфейс і раніше потребує адаптації до зміни розміру.

По-третє, зміна мови може вплинути не тільки на розмір тексту, але на організацію макета. Різні мови використовують різні напрямки компонування. Англійська, наприклад, використовує напрямок макета зліва направо, а арабська та іврит використовують з права наліво напрямок компонування. Загалом, порядок елементів призначеного для користувача інтерфейсу повинен відповідати напрямку компонування. Якщо кнопка знаходиться в правому нижньому кутку екрану на англійській мові, він повинен знаходитися в нижньому лівому кутку на арабській мові.

І, нарешті, якщо додаток підтримує IOS динамічний тип, користувач може змінити розмір шрифту, який використовується у вашому додатку. Це може змінити і висоту і ширину будь-яких текстових елементів в інтерфейсі. Якщо користувач змінює розмір шрифту, в той час як ваш додаток працює, шрифти і макет повинен адаптуватися до нових змін.

3.6. Побудова інтерфейсу без допомоги Constraints

Stack View забезпечують простий спосіб скористатися наявними можливостями Автокомпоновка без введення обмежень на складності. Один вид стека визначає рядок або стовпець елементів призначеного для користувача інтерфейсу. Вид стека організовує ці елементи на основі його властивостей.

- Вісь: (тільки UIStackView) визначає орієнтацію, вертикально або горизонтально.
- Орієнтація: (тільки NSStackView) визначає орієнтацію, вертикально або горизонтально.
- Розподіл: визначає розташування вікон уздовж осі.
- Вирівнювання: визначає розташування вікон перпендикулярно осі Stack View.
- Інтервал: визначає відстань між суміжними видами.

Щоб використовувати Stack View, в Interface Builder потрібно перетягнути вертикальну або горизонтальну Stack View на полотно. Потім розміщувати елементи всередині Stack View.

Якщо об'єкт має внутрішній зміст розмір, він з'являється в стеці в цьому розмірі. Якщо він не має характеристичну розмір контенту, Interface Builder забезпечує розмір за замовчуванням. Ви можете змінити розмір об'єкта, і Interface Builder додає обмеження для правильного розміщення.

Для подальшої тонкої настройки Auto Layout, можна змінити властивості Stack View, використовуючи Attributes Inspector. Наприклад, використовується інтервал 8 пунктів і "Заповнення Рівномірно" (Fill Equally) розподілу.

Stack View також визначає своє розташування на пріоритетах контенту розтягування і стиснення. Ви можете змінити їх за допомогою Attributes Inspector.

Крім того, ви можете вкладати Stack View всередині інших Stack View

для побудови більш складних макетів.

3.7. Анатомія Constraints в Auto-Layout

Розташування ієрархи різних View визначається як серія лінійних рівнянь. Кожен Constraint є одне рівняння. Мета цих рівнянь це одне і тільки одне рішення.

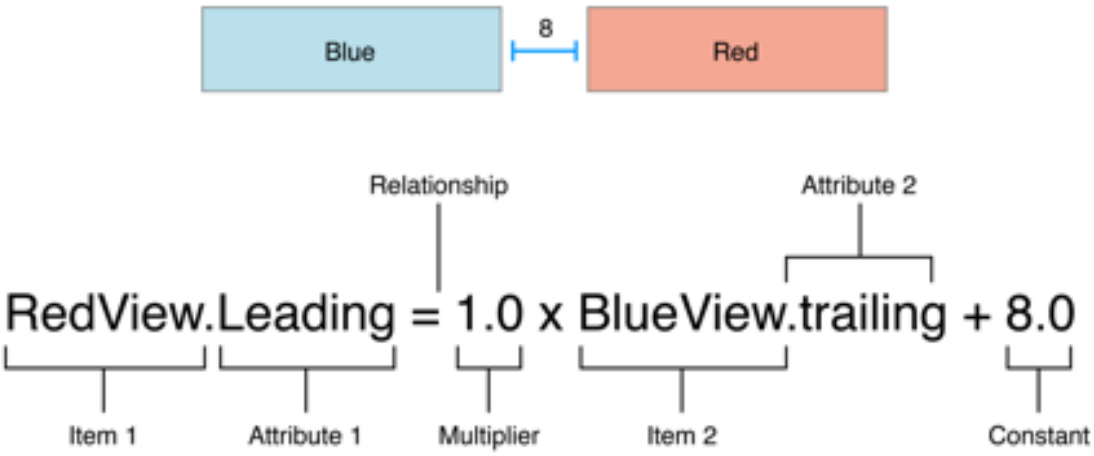

Рисунок 3.13 - Приклад рівняння

Constraint стверджує, що передній край червоного View повинен бути 8 точок після задньої кромки синього View.

Рівняння має ряд частин:

- Item 1. Перший елемент в рівнянні, в даному випадку, Червоний View.
- Attribute 1. Атрибут стримуватися на першому елементі, в даному випадку, переднім краєм червоного виду.
- Relationship. Відносини між лівою і правою сторін. Відносини можуть мати одне з трьох значень: одно, більше або дорівнює, або менше або дорівнює. У цьому випадку, ліва і права сторона рівні.
- Multiplier. Значення атрибута 2 множиться на це число з плаваючою комою. В цьому випадку множник дорівнює 1,0.
- Item 2. Другий пункт в рівнянні, в даному випадку, Синій View. На відміну від першого пункту, то це може бути залишено порожнім.
- Attribute 2. Атрибут на Item 2, в даному випадку, задній кромці

синього View.

- Constant. Константа з плаваючою точкою зміщення, в цьому випадку 8,0. Це значення додається до значення Attribute 2.

Constraints можуть також визначити зв'язок між двома різними атрибутами одного елемента, наприклад, встановивши співвідношення сторін між висотою і шириною розташований елемента. Можна також присвоїти постійні значення висоти або ширини розташований елемента.

3.7.1.Auto Layout Атрибути

В Auto Layout атрибути визначають функцію, яка може бути обмежена. Включає в себе чотири ребра (провідний, задній, верхній і нижній), а також висоту, ширину, а вертикальні і горизонтальні центри. Текстові елементи також мають один або кілька базових атрибутів.

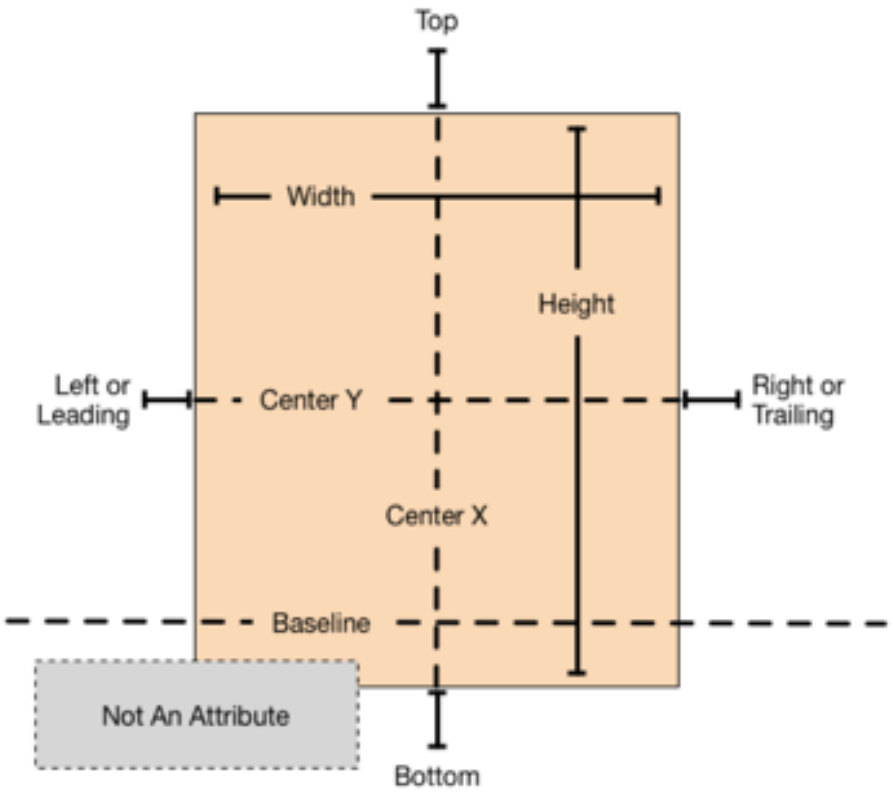

Рисунок 3.14 - Опис Constraints

Список атрибутів в мові Swift:

```
enum NSLayoutAttribute : Int {
```

```
        case Left
        case Right
        case Top
        case Bottom
        case Leading
        case Trailing
        case Width
        case Height
        case CenterX
        case CenterY
        case Baseline
        static var LastBaseline: NSLayoutAttribute { get }
        case FirstBaseline
        case LeftMargin
        case RightMargin
        case TopMargin
        case BottomMargin
        case LeadingMargin
        case TrailingMargin
        case CenterXWithinMargins
        case CenterYWithinMargins
        case NotAnAttribute
    }
```

3.7.2.Параметри рівнянь

Широкий діапазон параметрів і атрибутів доступні для цих рівнянь дозволяє створювати безліч різних типів Constraints. Ви можете визначити простір між View, поєднати краю View, визначити відносний розмір двох View, або навіть визначити пропорції. Однак, не всі атрибути сумісні.

Є два основних типи атрибутів. атрибути розміру (наприклад, висота і ширина) і атрибути розташування (наприклад, провідний, зліва і зверху). Атрибути розміру використовуються для визначення того, наскільки великий елемент, без будь-яких вказівок про його місцезнаходження. Атрибути

розташування використовуються для вказівки місця розташування будь-якого об'єкта по відношенню до чогось ще. Проте, вони не несуть ніяких вказівок на розмір елемента.

З урахуванням цих відмінностей на увазі, застосовуються такі правила:

- Ви не можете накласти Constraint на атрибут розміру для атрибута місцезнаходження.
- Ви не можете привласнити постійні значення атрибутів розташування.
- Ви не можете використовувати не поодинокий множник (значення, відмінне від 1,0) з атрибутами розташування.
- Для атрибутів розташування, ви не можете обмежити вертикальні атрибути горизонтальних атрибутів.
- Для атрибутів розташування, ви не можете обмежити початкові або кінцеві атрибути вліво або вправо атрибутів.

Наприклад, установка Top по елементу до постійного значення 20.0 не має сенсу без додаткового контексту. Ви завжди повинні визначити атрибути потрібного елемента, по відношенню до інших елементів, наприклад, 20,0 пункту нижче Top SuperView в. Проте, установка висоти по елементу 20.0 цілком допустимо.

```
// Установка постійної висоти
View.height = 0.0 * NotAnAttribute + 40.0

// Установка фіксованою дистанції між кнопками
Button_2.leading = 1.0 * Button_1.trailing + 8.0

// Вирівнювання ведучого краю двох кнопок
Button_1.leading = 1.0 * Button_2.leading + 0.0
```

```
// Установка обом кнопок однакової ширини
Button_1.width = 1.0 * Button_2.width + 0.0

// Установка View в центрі SuperView
View.centerX = 1.0 * Superview.centerX + 0.0
View.centerY = 1.0 * Superview.centerY + 0.0

// Установка константного співвідношення сторін для View
View.height = 2.0 * View.width + 0.0
```

3.7.3. Нерівність не є призначенням

Важливо відзначити, що рівняння, наведені в примітці представляють рівність, а не призначення.

Коли Auto Layout вирішує ці рівняння, це не просто привласнити значення справа наліво. Замість цього, він обчислює значення як для атрибута 1 і 2 атрибута, що робить відносини вірно. Це означає, що ми часто можемо вільно змінювати порядок елементів в рівнянні.

```
// Setting a fixed distance between two buttons
Button_1.trailing = 1.0 * Button_2.leading - 8.0

// Aligning the leading edge of two buttons
Button_2.leading = 1.0 * Button_1.leading + 0.0

// Give two buttons the same width
Button_2.width = 1.0 * Button.width + 0.0

// Center a view in its superview
Superview.centerX = 1.0 * View.centerX + 0.0
```

Superview.centerY = 1.0 * View.centerY + 0.0

// Give a view a constant aspect ratio
View.width = 0.5 * View.height + 0.0

3.7.4.Створення недвозначних інтерфейсів

При використанні Auto Layout, мета полягає в тому, щоб забезпечити ряд рівнянь, які мають одне і тільки одне з можливих рішень.

Загалом, Constraints повинні визначати розмір і положення кожного View. Припускаючи, що розмір батьківського View вже встановлено (наприклад, системний View в iOS). Недвозначний і здійсненний layout вимагає двох Constraints для View і для кожного дозволу (не рахуючи SuperView). Проте, у вас є широкий спектр варіантів, коли мова йде про вибір того, які обмеження ви хочете використовувати. Наприклад, наступні три макета все роблять недвозначні, здійсненні макети (тільки горизонтальні обмеження показані):

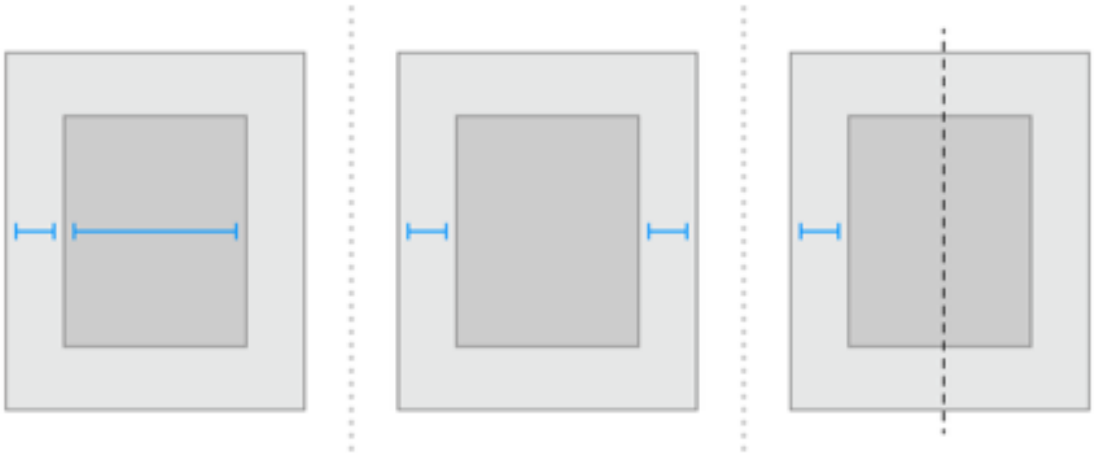

Рисунок 3.15 - Layout приклади

- Перший layout позначає передню кромку виду по відношенню до передньої кромці свого SuperView в. Це також дає вид на фіксовану ширину. Положення задньої кромки потім може бути обчислено на основі розміру SuperView і інших обмежень.

- Другий layout позначає передню кромку виду по відношенню до передньої кромці свого SuperView в. Це також обмежує виду по задньої

крайки щодо задньої кромки SuperView в. Ширина виду не може бути розрахована на основі розміру SuperView і інших обмежень.

- Третій layout позначає передню кромку виду по відношенню до передньої кромці свого SuperView в. Крім того, центр вирівнює вид і SuperView. Обидві позиції по ширині і задньої кромки краю може бути обчислена на основі розміру SuperView і інших обмежень.

Наступні рисунки 3.16 і 3.17 показують Views в портретному і ландшафтному вигляді:

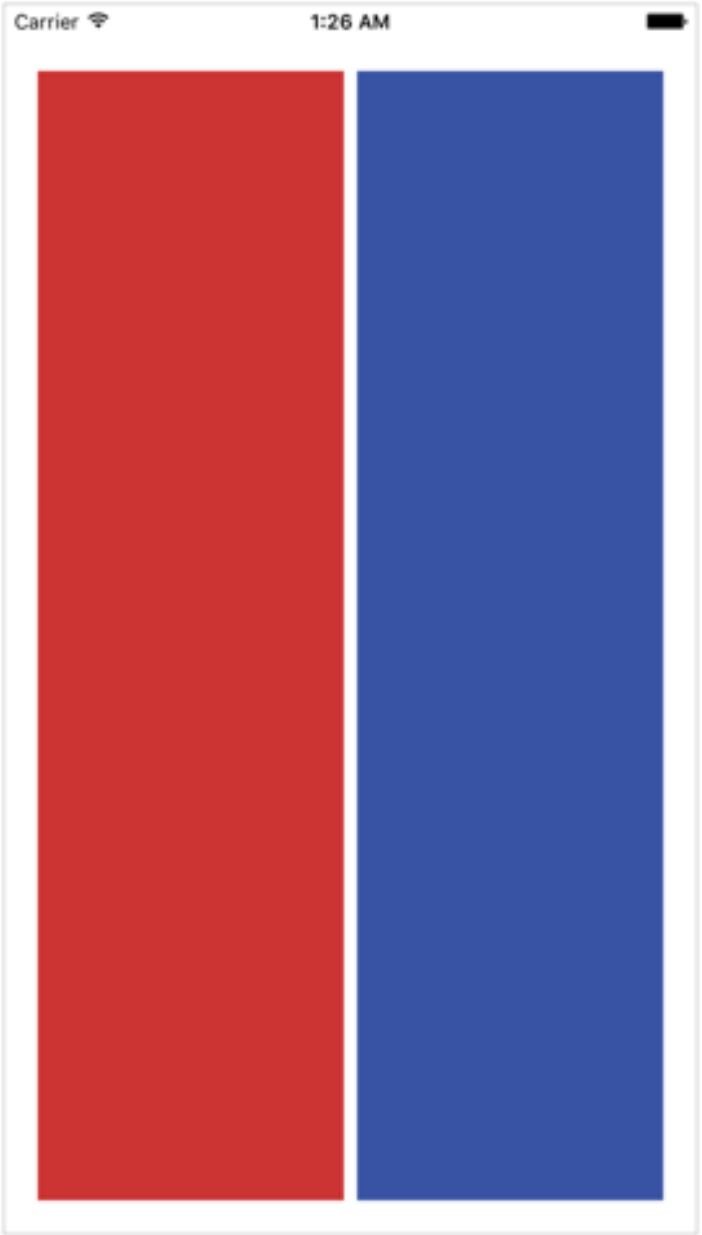

Рисунок 3.16 - Портретний вигляд

Рисунок 3.17 - Ландшафтний вигляд

Для даних Views, Constraints будуть виглядати наступним чином:

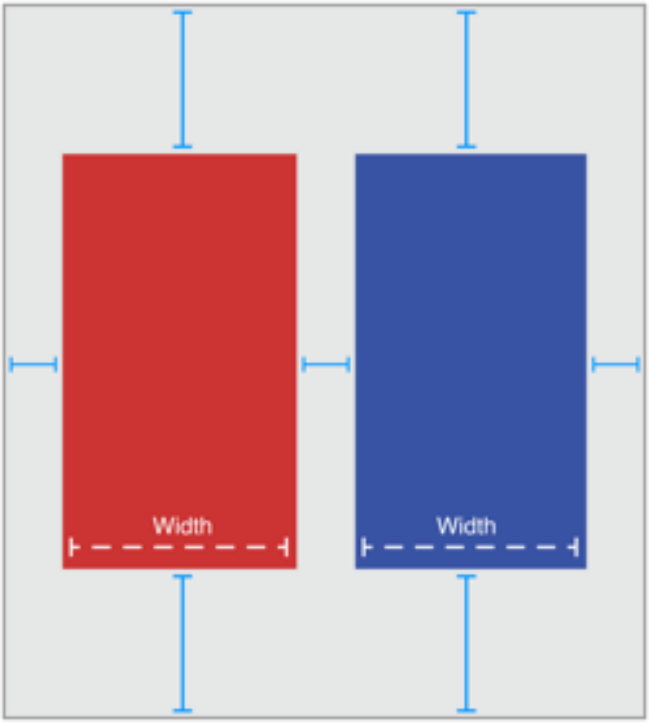

Рисунок 3.18 - Constraints на обох елементах

Рішення на зображенні 3.18 буде описуватися наступними Constraints

// Верткальние Constraints

Red.top = 1.0 * Superview.top + 20.0

Superview.bottom = 1.0 * Red.bottom + 20.0

Blue.top = 1.0 * Superview.top + 20.0

Superview.bottom = 1.0 * Blue.bottom + 20.0

// Горизонтальні Constraints

Red.leading = 1.0 * Superview.leading + 20.0

Blue.leading = 1.0 * Red.trailing + 8.0

Superview.trailing = 1.0 * Blue.trailing + 20.0

Red.width = 1.0 * Blue.width + 0.0

Дотримуючись правил, ця схема має дві точки зору, чотири горизонтальні обмеження і чотири вертикальних обмежень. Хоча це не є безпомилковим вирішенням, це швидкий ознака того, що ви на правильному шляху. Що ще більш важливо, обмежено однозначно повине визначити як розмір так і розташування обох Views, виробляючи недвозначний, виконуваний layout. Видаліть будь-який з цих обмежень, і рішення стає неповним, що привидітися до помилки. Додайте додаткові обмеження, і ви ризикуєте побачити конфлікт між двома Constraints.

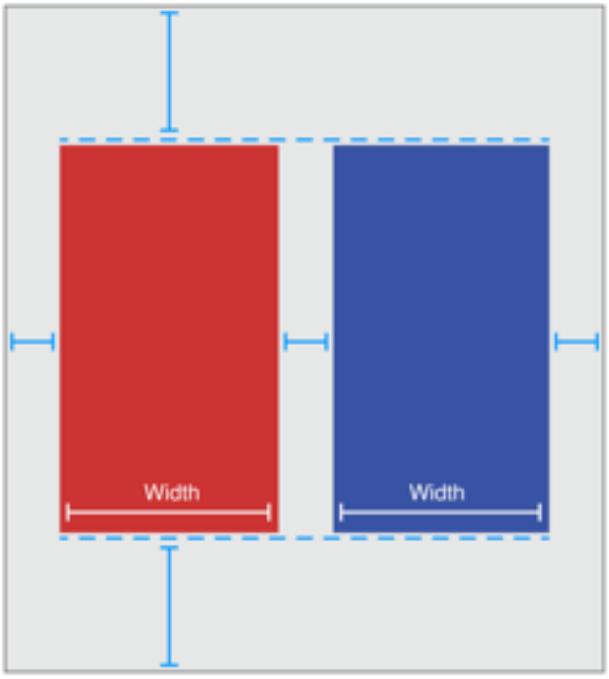

Рисунок 3.19 - Додаткові обмеження на елементи

Проте, це не є єдино можливим рішенням. Продемонструємо ще один підхід до вирішення завдання:

Замість того, щоб приколювати верхньої і нижні частини синій коробці, вирівнювати верхню частину синьої коробки з верхньою частиною червоною коробки. Точно так же, вирівнювати нижню частину синьої коробки з нижньою червоною коробкою. Обмеження наведені нижче.

```
// Верткальние Constraints
Red.top = 1.0 * Superview.top + 20.0
Superview.bottom = 1.0 * Red.bottom + 20.0
Red.top = 1.0 * Blue.top + 0.0
Red.bottom = 1.0 * Blue.bottom + 0.0

// Горизонтальні Constraints
Red.leading = 1.0 * Superview.leading + 20.0
Blue.leading = 1.0 * Red.trailing + 8.0
Superview.trailing = 1.0 * Blue.trailing + 20.0
Red.width = 1.0 * Blue.width + 0.0
```

3.7.5. Нерівність Constraints

До сих пір всі приклади показали Constraints як рівності, але це тільки частина історії. Обмеження можуть представляти нерівності. Зокрема, відносини Обмеженням може бути одно, більше або дорівнює, або менше або дорівнює.

До сих пір всі приклади показали Constraints як рівності, але це тільки частина історії. Обмеження можуть представляти нерівності. Зокрема, відносини Обмеженням може бути одно, більше або дорівнює, або менше або дорівнює. До сих пір всі приклади показали Constraints як рівності, але це тільки частина історії. Обмеження можуть представляти нерівності.

Зокрема, відносини Обмеженням може бути одно, більше або дорівнює,

або менше або дорівнює

Наприклад, ви можете використовувати обмеження для визначення мінімального або максимального розміру для виду (Лістинг 3.5).

Лістинг 3.5 Присвоєння мінімального і максимального розміру

```
// Установка мінімальної ширини
View.width> = 0,0 * NotAnAttribute + 40,0
// Установка максимальної ширини
View.width <= 0,0 * NotAnAttribute + 280,0
```

Як тільки ви почнете використовувати нерівності, два обмеження на View на розмір, як правило ламається. Ви завжди можете замінити одне відношення рівності, двома нерівностями. У лістингу 7-5-2, едині рівні відносини і пара нерівностей виробляють таку ж поведінку.

Лістинг 3.6 Замінюється єдине рівність двома нерівностями

```
// Єдине рівноправне ставлення
Blue.leading = 1,0 * 8,0 + Red.trailing
// Може бути замінено двома відносинами нерівності
Blue.leading> = 1,0 * 8,0 + Red.trailing
Blue.leading <= 1,0 * 8,0 + Red.trailing
```

Зворотне не завжди вірно, так як два нерівності не завжди еквівалентні одному відношенню рівності. Наприклад, нерівність в лістингу 7-5-1 обмежує діапазон можливих значень для подання по ширині, але самі по собі, вони не визначають ширину. Ви як і раніше повинні додати горизонтальні обмеження для визначення положення і розміру View в межах цього діапазону.

Замість того, щоб приколювати верхньоі і нижні частини синій коробці, вирівнювати верхню частину синьої коробки з верхньою частиною червоною коробки. Точно так же, вирівнювати нижню частину синьої коробки з нижньою

червоною коробкою. Обмеження наведені нижче

3.7.6.Власний розмір змісту

Деякі View мають власний розмір, враховуючи їх поточне утримання. Це визначається розміром їх змісту. Наприклад, внутрішній розмір вмісту кнопки є розмір його заголовка плюс невеликий відступ. Не всі види мають внутрішній розмір контенту. View які, які визначають власний розмір, мають ширину і висоту автоматично.

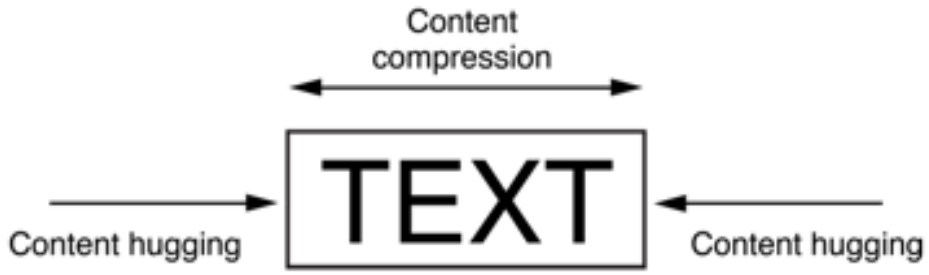

Рисунок 3.20 -Власний розмір

Власний розмір змісту заснований на поточному утриманні View. Внутрішній зміст розмір Текстового поля або кнопки на грунтується на кількості тексту, і використовуваного шрифту. Для інших View, внутрішня розмір контенту є ще більш складним. Наприклад, порожній вид зображення не мають внутрішній розміру контенту. Як тільки ви додаєте зображення, його внутрішній розмір, ставати еквівалентним розміру зображення.

Auto Layout представляє внутрішній розмір контенту, парою Constraint для кожного вимірювання. Стиснення контенту змушує View штовхати всередину вміст таким чином, щоб воно щільно облягала край і навпаки.

Ці обмеження визначаються за допомогою нерівності, показані в лістингу 3.5. Тут константи IntrinsicHeight і IntrinsicWidth представляють значення висоти і ширини від власного розміру вмісту виду.

Деякі приклади наведені в таблиці 3.1.

Таблиця 3.1 - Внутрішні розміри

View	Внутрішній розмір змісту.
UIView and NSView	Не має внутрішнього розміру.
Sliders	Визначає тільки ширину (iOS).
Labels, buttons, switches, and text fields	Визначають ширину і висоту.
Text views and image views	Наявність внутрішнього стан може варіюватися.

```
// Міцність на стискання
View.height> = 0,0 * NotAnAttribute + IntrinsicHeight
View.width> = 0,0 * NotAnAttribute + IntrinsicWidth

// Додержання розтягування
View.height <= 0,0 * NotAnAttribute + IntrinsicHeight
View.width <= 0,0 * NotAnAttribute + IntrinsicWidth
```

Кожне з цих обмежень може мати свій власний пріоритет. За замовчуванням View використовують 250 пріоритет для їх розтягування і 750 пріоритет для міцності при стисненні. Таким чином, легше розтягнути View, ніж стиснути його. Для більшості елементів управління, це бажане поведінку. Наприклад, ви можете спокійно розтягнути кнопку по ширині, так як на власний розмір контенту це ніяк не впливає. Проте, якщо ви зменшити його, контент може стати обрізання. Зверніть увагу, що Interface Builder іноді може змінити ці пріоритети, щоб допомогти запобігти небажані конфлікти між Constraints.

При розтягуванні серії View, щоб заповнити пробіл, якщо всі точки зору мають однаковий пріоритет змісту облягаючі, макет неоднозначний. Auto Layout не знає, який View повинен бути розтягнутий.

Типовим прикладом є пара етикетки і текстове поле. Як правило, ви хочете розтягнути і заповнити текстове поле додатковим простором, в той час

як мітка залишається на своєму власному розмірі вмісту. Щоб забезпечити це, переконайтеся, що горизонтальний пріоритет контенту розтягує текстове поле нижче, ніж лейбла.

3.8. Висновки до розділу 3

Після детально ознайомлення з інструментом для створення графічних і мультимедійних інтерфейсів Auto-Layout, ми маємо можливість впевненно сказати, що цей інструмент можна вважати невідємною частиною сучасного інструментарію Apple для операційнних систем iOS та WatchOS.

Auto-Layout дає нам можливість створювати інтерфейси за допомогою GUI в Interface Builder в программному продукті Apple, а також за допомогою коду.

На поведінку інструменту впливають різноманітні чинники, до яких відносяться як внутрішні зміни, так і зовнішні зміни. Різноманітні обмеження, які можливо накласти на позиціонування елементів, є ключовою можливістю для вирішення задач при проектуванні складних, багато-залежних, призначених для користувача інтерфейсів.

Анатомія Auto-Layout є складним елемнтом, який розробники Apple зуміли зробити зрозумілим для розробників.

Взагалом ця система є логічним кроком компанії, який призведе до збільшення кількості якісних продуктів у вигляді додатків для iPhone та Apple Watch.

4. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ДОДАТКУ

4.1. Опис додатку

Під час обговорення ідеї програми для дипломною роботи, наша команда знайшла рішення, яке б задовільняла всі сторони. Було вирішено створити мобільний додаток для iPhone, а також для Apple Watch.

Додаток був розроблений для користувачів, які займаються бігом. Він буде давати змогу користувачеві аналізувати свою показники після бігу, а саме:

- Перегляд пульсу
- Кількість затрачених калогій
- Геолокаційні дані положення

Усього система буде складатись з 2 додатків.

1. Додаток для Apple Watch

Основі функцій:

- старт програми
- початок тренування
- зупинка тренування
- збереження даних під час бігу
- вимір пульсу
- зчитування кількості калорій на кожному проміжку фіксації пульсу
- зчитування гео-локаційних даних на кожному проміжку
- передача даних на iPhone
- Інтеграція з HealthKit
- зупинка тренування
- вимкнення програми

2. Додаток для iPhone

Основні функцій:

- запуск программ
- збереження даних
- відображення всіх тренуваннь

- вибір тренування
- перегляд карти на якій зображені точки
- вибір точки і перегляд інформації пульс і час коли буи зроблений замір
- вихід з перегляду тренування
- видалення тренування
- синхронізація з годиником

Розглянемо основні єкрани додатків.

Apple Watch.

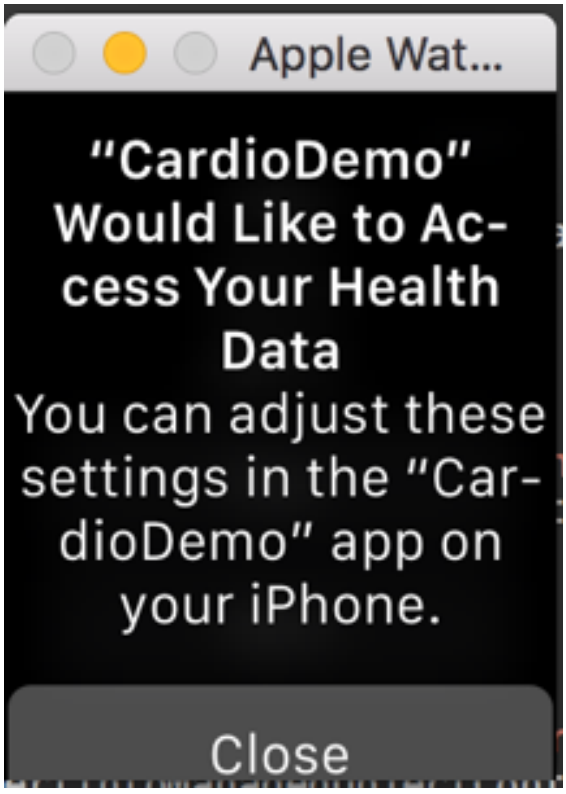

Рисунок 4.1 - Початковий екран

На рисунку 4.1 зображен запит для користувача на дозвіл користування Health Data на розумному годинику. Операційна систеса рабить такий запит для того щоб користувачі розуміли що саме буде використовувати додаток. У випадку якщо користвачвваже що додаток не повинен мати можливість отримувати дані, то він може натиснути Close.

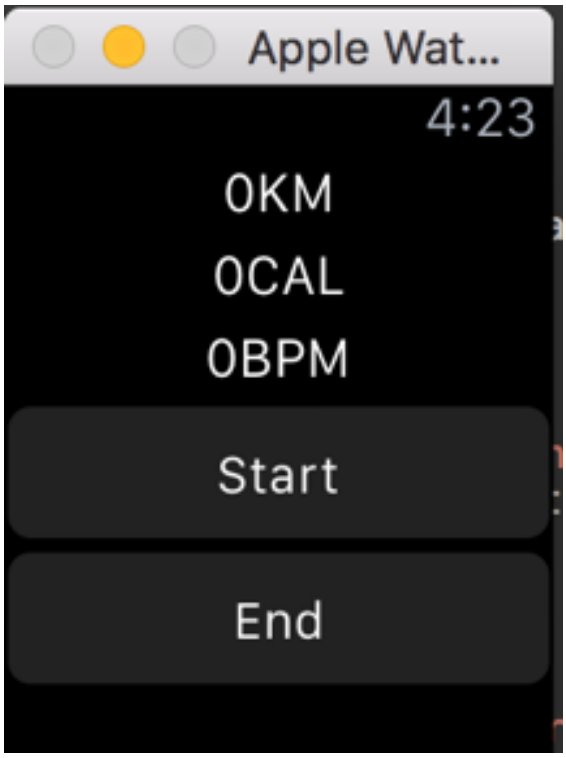

Рисунок 4.2 - Головний екран тренування

На рисунку 4.2 зображен головний екран додатку. На ньому користувач має змогу почати тренування та завершіті тренування. Таклж е можливість переглядати дані останього виміру.

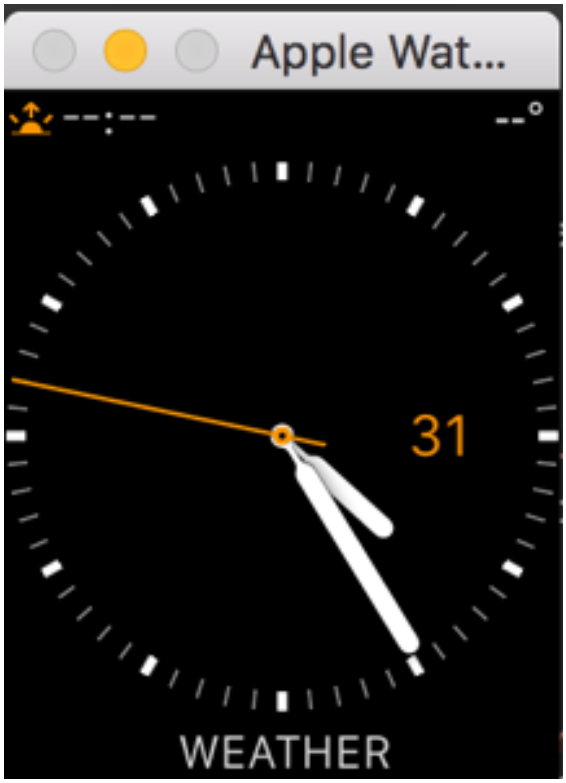

Рисунок 4.3 - Екран годиника

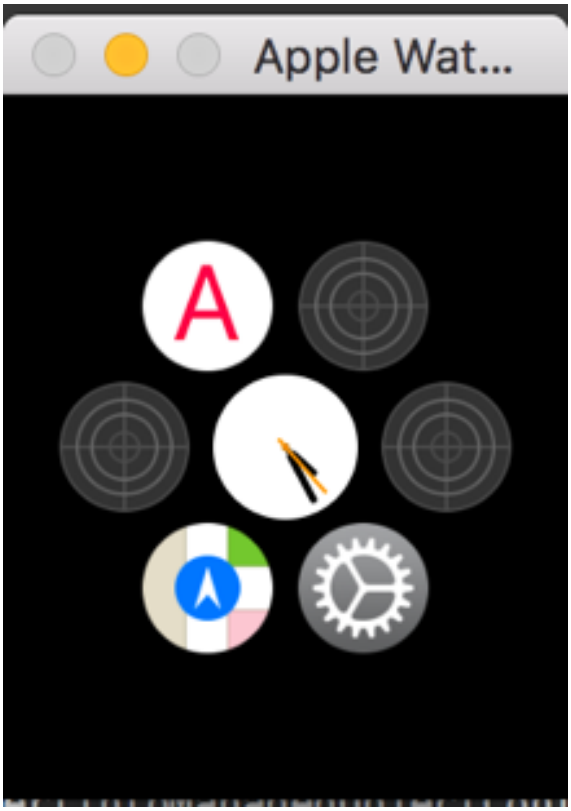

Рисунок 4.4 - Вигляд додатку у меню запуску додатків

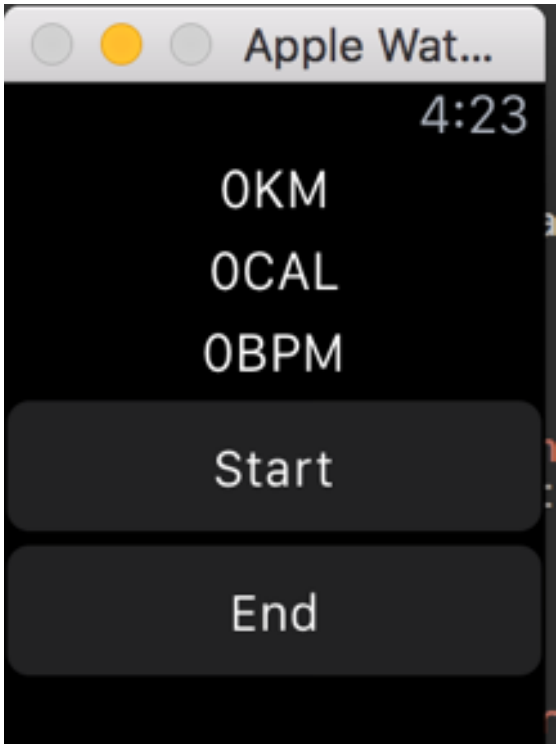

Рисунок 4.5 - Головний екран у робочому режимі з кнопокою Pause

iPhone додаток.

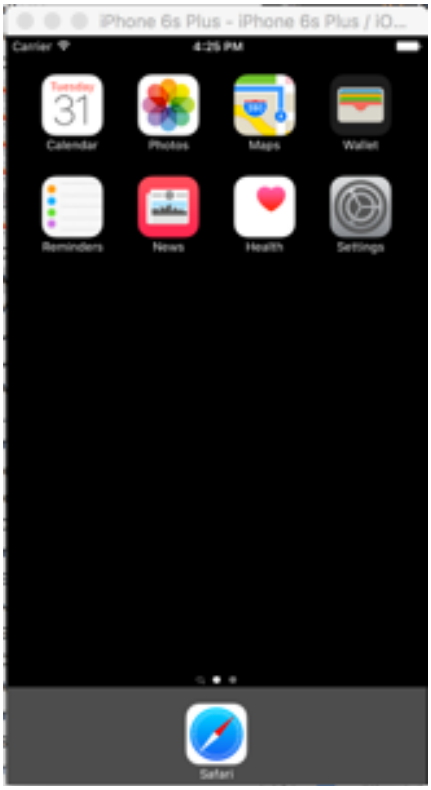

Рисунок 4.6 - Головне меню з вибором додатків в iOS

Рисунок 4.7 - Головне меню додатку з вибором тренування

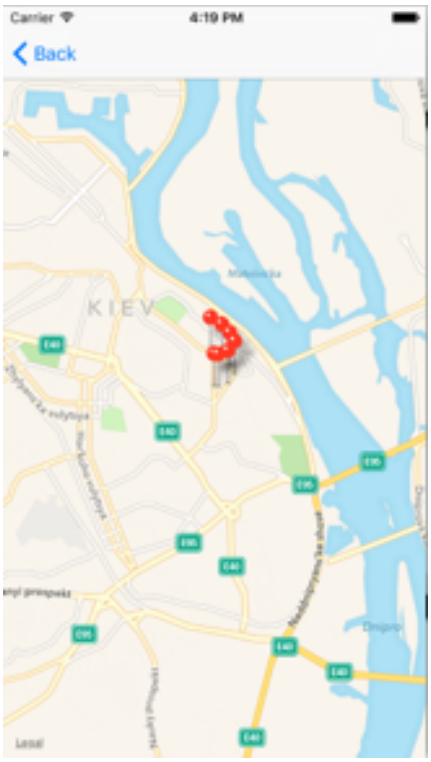

Рисунок 4.8 - Карта після вибору тренування

Рисунок 4.9 - Карта після збільшення

Рисунок 4.10 - Відображення інформації принатисканні на Pin

Рисунки нижче, дають загальне уявлення про інтерфейс програми. У головному вікні - відображають список готових продуктів. Натиснувши на один з них, програма відображає інформацію про те, як використовувати цей продукт, його категорію, скільки на це потрібно часу, які і в якій кількості необхідні тренування, а також кнопку яка відображає текстову інструкцію з початку програм (рисунок 4.7). Також в програмі є можливість конвертації значень між грамами, фунтами і унциями, а також таблицю тренувань яка відсортована за датою.

4.2. Cocoa Touch як програмний каркас для створення WatchOS та iOS додатків

Cocoa Touch - це фреймворк для створення додатків під iPhone, iPod touch, і iPad.

Бібліотека Cocoa Touch надає рівень абстракції для iOS (операційної системи iPhone, iPad і iPod touch). Cocoa Touch заснована на класах фреймворка Cocoa, використовуваного в Mac OS X, і, аналогічно їй, використовує мову Objective-C. Cocoa Touch слід шаблоном проектування Model-View-Controller.

Інструменти для розробки додатків з використанням Cocoa Touch включені в iOS SDK.

Як і у всіх середовищах додатків, Cocoa має два світи: світ runtime'a і світ розробки. У світі runtime'a, Cocoa додатки представляють призначений для користувача інтерфейс і тісну інтеграцію з іншими компонентами операційної системи в Mac OS X, наприклад, Finder і Dock.

Але в світі розробки Cocoa - інтегрований набір об'єктно-орієнтованих програмних компонентів-класів, які, власне, дозволяють творити ПО під Mac OS X і IOS. Вони дають можливість робити велику кількість речей, від user-interface'a до управління масивами даних.

При розробці Cocoa додатків, насправді, можна використовувати кілька мов програмування, але рідна мова - Objective-C, який є розширенням ANSI C, з деякими синтаксичними і семантичними особливостями (на основі Smalltalk) для підтримки ООП. Крім того, ваш код може викликати функції визначені в non-Cocoa інтерфейсі, такі як бібліотеки BSD в / usr / include. Ви навіть можете змішувати C ++ код з Cocoa кодом і посилатися на цей скомпільований чудо в вашому файлі,.

Найбільш важливі бібліотеки Cocoa упаковані в два основних framework'a для кожної платформи: AppKit для Mac OS X і UIKit для IOS. Як і всі framework'і, вони містять не тільки динамічно доступні бібліотеки (а іноді і кілька версій бібліотек, необхідні для забезпечення зворотної сумісності), але і файли заголовків, API документацію, і пов'язаних з ними ресурси. Framework -

це дуже важлива складова для будь-якого проекту під Mac або IOS.

Mac OS X підтримує до всього іншого ще багато корисних бібліотек, таких як: WebKit і Address Book frameworks, але про це пізніше.

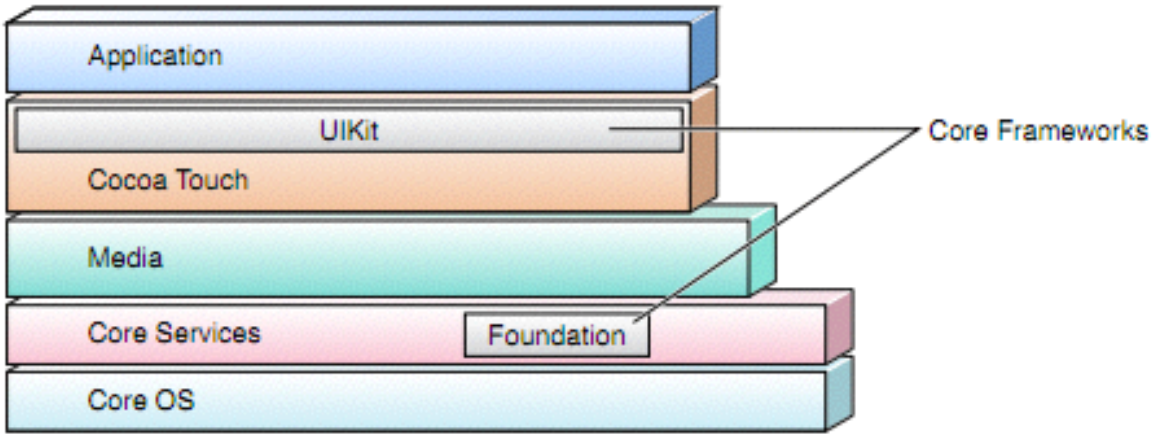

Рисунок 4.11 - Програмні шари iOS

Нижче наводиться короткий виклад деяких з framework'ов кожного шару:

- Core OS. Цей рівень містить ядро, файлову систему, мережевої інфраструктури, безпеку, управління живленням, а також ряд драйверів пристроїв. У ньому, до всього іншого, міститься бібліотека libSystem, яка підтримує POSIX / BSD 4.4 / C99 API специфікації і включає в себе системи на рівні API для багатьох сервісів.

- Core Services. Надає основні сервіси, такі як маніпуляції з рядками, управління колекціями та мережевим взаємодією, управління контактами, настройками. Вони також дають можливість користуватися апаратними особливостями будови (GPS, компас, акселерометр і гіроскоп).

- Приклади framework'a цього шару - Core Location, Core Motion, і System Configuration.

- Цей шар включає в себе як Foundation так і Core Foundation, які пропонують деякі типи даних, такі як рядки і колекції.

- Media. Забезпечує шару Cocoa Touch доступ до мультимедійних можливостей. Включає в себе Core Graphics, Core Text, OpenGLES,

- Core Animation, AVFoundation, Core Audio, і сервіси відтворення відео.

- Cocoa Touch. Займається безпосередньою підтримкою додатків. Містить такі компоненти як Game Kit, Map Kit і iAd.

Для того щоб інтегрувати Map Kit фреймворк в наш додаток для спортсменів які займаються бігом, нам знадобилось використвти Cocoa Touch. Рисунок 4.8.

Безпосередньо за увесь UI відповідає відповідає бібліотека UIKit за допомогою неї ми маємо можливість забезпечити створення графічного інтерфейсу та елементи керування. Він імплементує найнижчі класи які визначають базову поведінку елементів

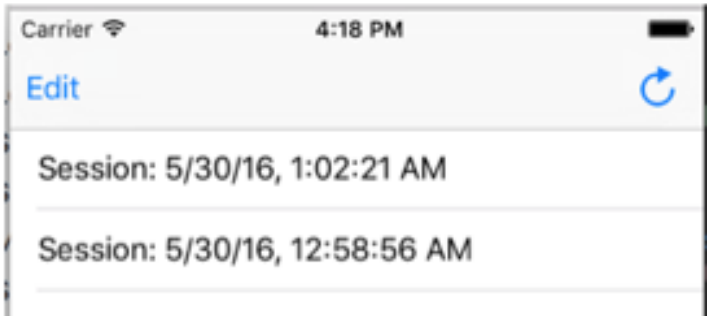

Рисунок 4.12 - Приклад елементів які забезпечує UIKit

На рисунку 4.12 ми можмо бачити фрагмент екрану iPhone під операційною системою iOS. На ньму усі елементи підконтрольні батьківскому фреймворку Cocoa Touch

4.3. Розробка структури додатку на базі MVC

Моде́ль-вигляд-контро́лер (або Модель-вид-контролер, англ. Model-view-controller, MVC) — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

Цей шаблон поділяє систему на три частини: модель даних, вигляд даних та керування. Застосовується для відокремлення даних (модель) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Бачення MVC компанією Apple, давайте подивимося на традиційну версію ми можемо розглянути на рисунку 4.13

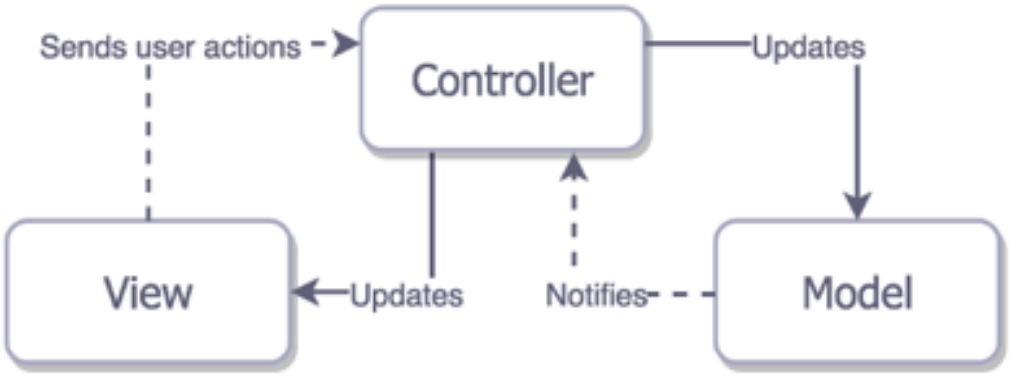

Рисунок 4.13 - Традиційне бачення MVC

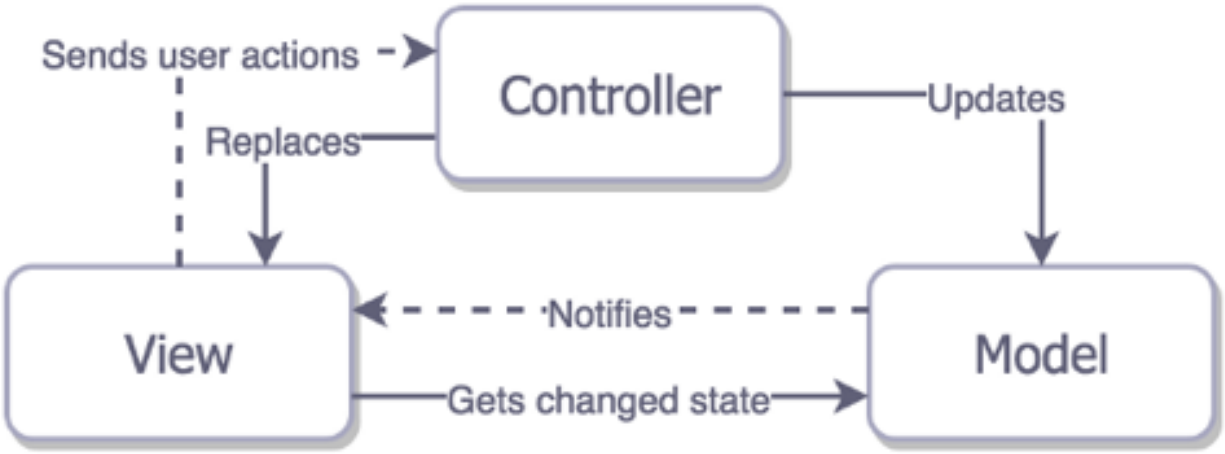

Рисунок 4.14 - Бачення MVC у компанії Apple

У традиційному MVC View не зберігає стану в собі. Controller просто «рендерить» View при змінах Model. Наприклад, веб-сторінка повністю перевантажується після того, як ви натиснете на посилання для переходу в інше місце. Хоча можна реалізувати традиційний MVC в середовищі iOS, це не має особливого сенсу через архітектурну проблеми: все три сутності тісно пов'язані, кожна сутність знає про двох інших. Це сильно знижує можливість повторного використання кожного з елементів. З цієї причини ми не будемо навіть намагатися написати приклад канонічного MVC.

Традиційний MVC здається непридатним до сучасної iOS розробці.

Розглянемо тепер рисунок 4.14 Controller є посередником між View і Model, отже, дві останніх не знають про існування один одного. Тому Controller важко повторно використовувати, але це, в принципі, нас влаштовує, так як ми повинні мати місце для тієї хитрої бізнес-логіки, яка не вписується в Model.

В теорії все виглядає дуже просто, але ви відчуваєте, що щось не так, чи не так? Ви напевно чули, що люди розшифровують MVC як Massive View Controller. Крім того, розвантаження ViewController стала важливою темою для iOS-розробників. Чому це відбувається, якщо в Apple просто взяли традиційний MVC і трохи його поліпшили?

4.4. Структура бази даних

Для того щоб зберігати отриммані дані під час роботи додатку, на основі попереднього аналізу, було прийнято рішення використати системний фреймворк Core Data.

Core Data - один з найбільш потужних фреймворків в iOS. Фундаментально, це спосіб створення Swift (Objective-C) об'єктів як об'єктів, "відображених" в SQL або XML базах даних. Це свого роду "місток" між об'єктно-орієнтованої "територією" і "територією" баз даних. За базу даних домінує SQLite.

Для такого "відображення", як і у всякій іншій базі даних, ми створюємо в Xcode Модель Даних (Data Model) і там працюємо з:

Сутностями (Entities) - в "світі" баз даних це таблиці, в яких будуть "відображатися" наші Swift об'єкти,

Атрибутами (Attributes) - це колонки в таблиці бази даних. У нашому об'єктно-орієнтованому "світі" це відповідає властивостям (properties) об'єктів.

Взаємозв'язками (Relationships) - це також властивості об'єктів, але вони є покажчиками на інші об'єкти в базі даних, або покажчиками на ряд інших об'єктів, це щось типу "joins" між таблицями в базі даних.

Запитами як властивостей (Fetch properties) - це "обчислюється" спосіб отримати покажчик на деякі інші властивості. Якщо Взаємозв'язки посилаються безпосередньо на кінцеві об'єкти, то Запити посилаються на об'єкти, вибрані зазначеним предикатом.

Як нам отримати доступ до всього цього в нашому Swift коді після того, як ми створили Модель Даних (Data Model)? Відповідь полягає в тому, що нам

потрібен контекст NSManagedObjectContext (для стислості надалі позначимо його MOC). Цей MOC в коді є "центральним простором" для створення об'єктів в базі даних, встановлення їх атрибутів і запитів до об'єктів. Все це ми будемо робити через MOC. Він же буде автоматично "відображати" наші дії в SQLite.

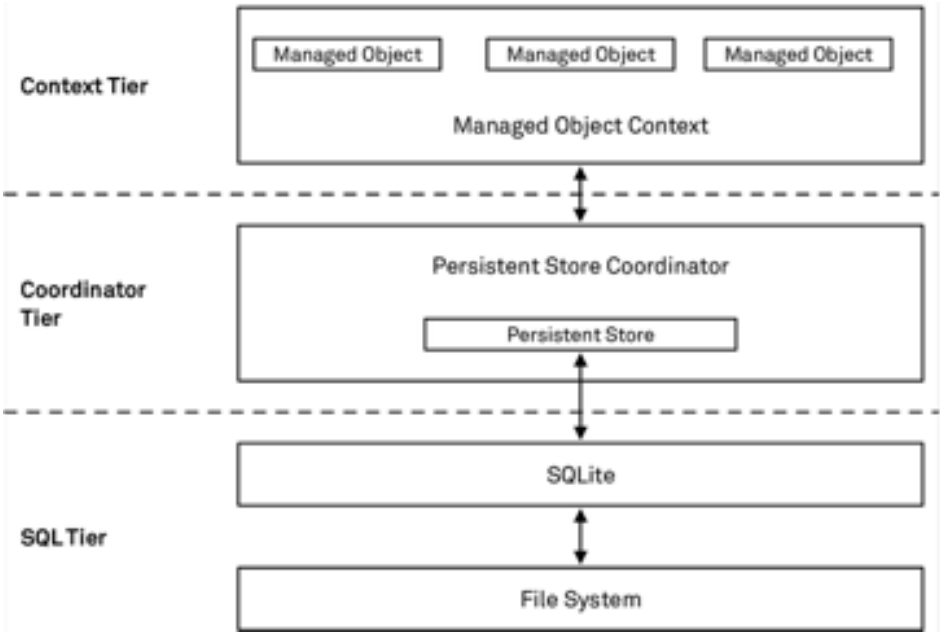

Рисунок 4.15 - Схематичне представлення архітектури Core Data

Звичайно ж Core Data не панацея і є інші варіанти зберігання даних, які можуть краще підійти при вирішенні певних завдань, але вже дуже добре і красиво Core Data вписується в Cocoa Touch. Більшість деталей по роботі зі сховищем даних Core Data приховує, дозволяючи вам сконцентруватися на тому, що дійсно робить ваше додаток, унікальним і зручним у використанні. Також в середовищі розробки xCode присутні зручні інструменти для роботи з фреймворком.

Не дивлячись на те, що Core Data може зберігати дані в реляційної базі даних на зразок SQLite, Core Data не є СУБД. По-правді Core Data в якості сховища може взагалі не використовувати реляційні бази даних. Core Data не є чимось на зразок Hibernate, хоча і надає деякі можливості ORM. Core Data швидше є оболонкою / фреймворком для роботи з даними, яка дозволяє

працювати з сутностями і їх зв'язками (відносинами до інших об'єктами), атрибутами, в тому вигляді, який нагадує роботи з об'єктним графом в звичайному об'єктно-орієнтованому програмуванні.

Для того щоб зберігати дані під час початку сесії, було створено дві моделі. Session - описує сесію тренування користувача. Location - описує дані, які додаток збирає під час того, як користувач почав біг.

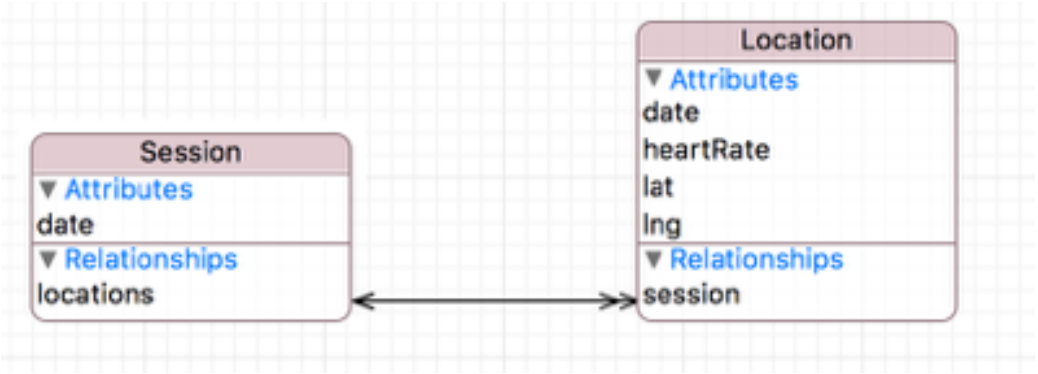

Рисунок 4.16 - Схема бази даних

Після того як користувач починає сесію, тобто “тренування”, створюється обєкт Session з поточною датою. Через кожні 5 секунд програма опитує систему на наявність нової партії обновлених статистичних даних від датчиків пульсу та локації, і зберігає їх в модель Location. Після того як користувач припиняє біг і всі збережені дані Locations додаються в масив locations мождели Session і зберігаються в локальне сховище. Тепер ці дані можуть бути використані для грфічного відображення або для подальшого аналізу.

Продумування схеми даних - найважливіший момент при роботі з Core Data. Виправлення помилки, допущеної на етапі проектування архітектури, може коштувати розробнику купу часу і нервів. Ідеальний варіант, якщо після виходу в бій модель не змінюється. У реальності, якщо нам не доведеться вдаватися до ручної міграції через Migration Manager і всі зміни проковтують Lightweight Migration. Приділяйи цьому етапу якомога більше часу і намагатися експериментувати з різними варіантами моделей.

4.5. Програмний інтерфейс додатку на iPhone та Apple Watch

Беручи до уваги результати попереднього аналізу з існуючих інструментів побудови графічного інтерфейсу, було прийнято рішення використовувати фреймворк AutoLayout.

У iOS 6 Apple представили чудову можливість для верстки UI для iOS-додатків - Auto Layout. Але ось що дивно, до цих пір дуже мало хто проекти використовують цю можливість. А адже це дуже сильний інструмент, якщо з розумом підійти до верстці UI, можна заощадити дуже багато часу на підстроювання елементів для 3,5 "і 4" екранів, портретно-ландшафтному розташуванні екрану і навіть на універсальної верстці для iPhone і iPad. І це все не рахуючи того, що скоро представлять iPhone 6 і ніхто досі точно не знає, яке там буде дозвіл і який екран. Краще б заздалегідь підстрахуватися.

В основному, тема Auto Layout досить проста, і вивчити її нескладно. Але особисто я зіткнувся з проблемою при розташуванні елементів в UIScrollView. Я витратив чимало часу і нервів на вивчення того, як же правильно розташувати елементи і вказати розмір контенту для того щоб ScrollView почав перегортувалися.

Рисунок 4.17 - Схема інтерфейсу Apple Watch додатку

AutoLayout використовується для побудови динамічних призначених для користувача інтерфейсів, масштабованих і адаптуються до різних форматів і дозволами екранів пристроїв, а також їх орієнтаціям. AutoLayout прийшов на зміну системі «пружин і розтяжок» застосовується 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, ми можемо створити зв'язок яка 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'язку і математично обчислює ідеальні позиції і розміри для всіх уявлень. Нам не потрібно більше встановлювати розміри уявлень вручну, задавати їх координати розташування - AutoLayout бере цю роботу на себе.

Створювати зв'язку можна як програмно, так і за допомогою Interface Builder.

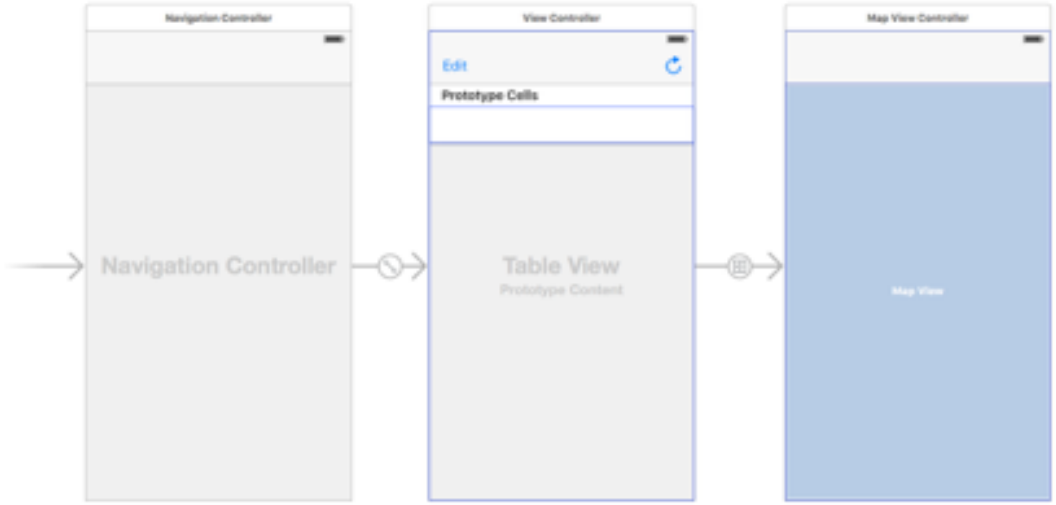

Рисунок 4.18 - Схема інтерфейсу iOS додатку

4.6. Тестування та результати роботи програми

Для тестування результатів роботи додатку було використано смартфон iPhone 6 та годинник Apple watch 42мм. У процесі розробки додатку у нас виникали певні труднощі, переважно — під час імплементації дизайну. Ось детальніше про деякі з них.

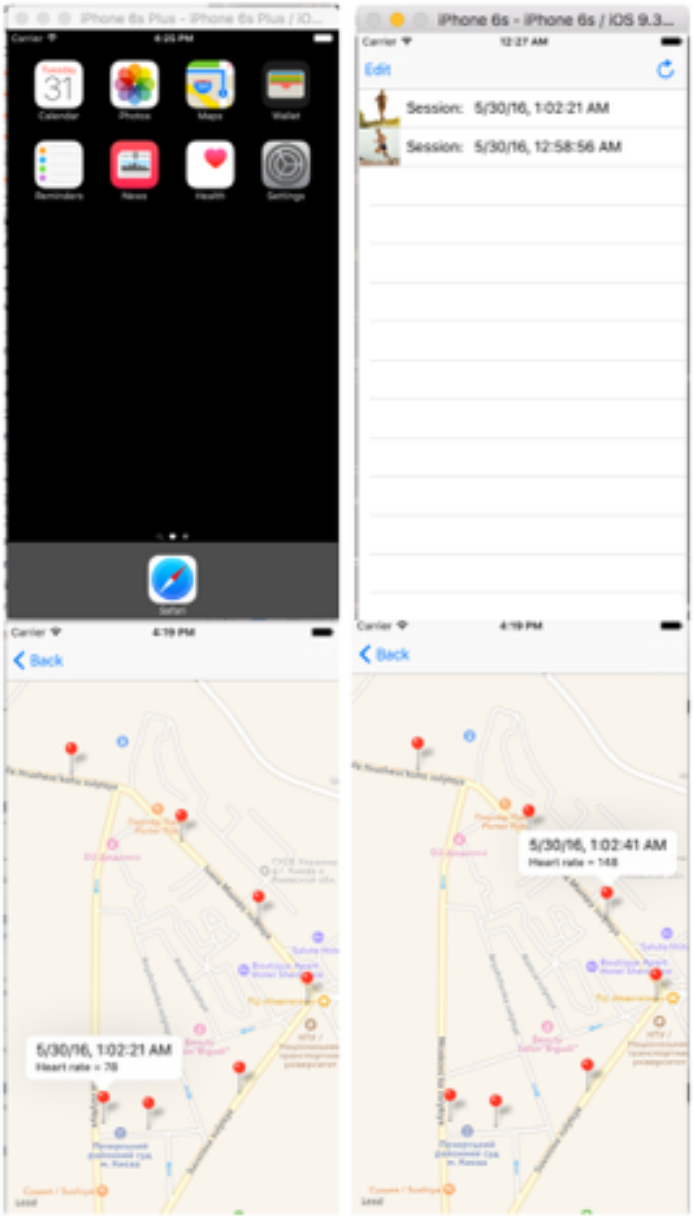

Рисунок 4.19 - Інтерфейс iOS додатку

У першу чергу, Apple Watch це пристрій для контролю здоров’я та активності людини.

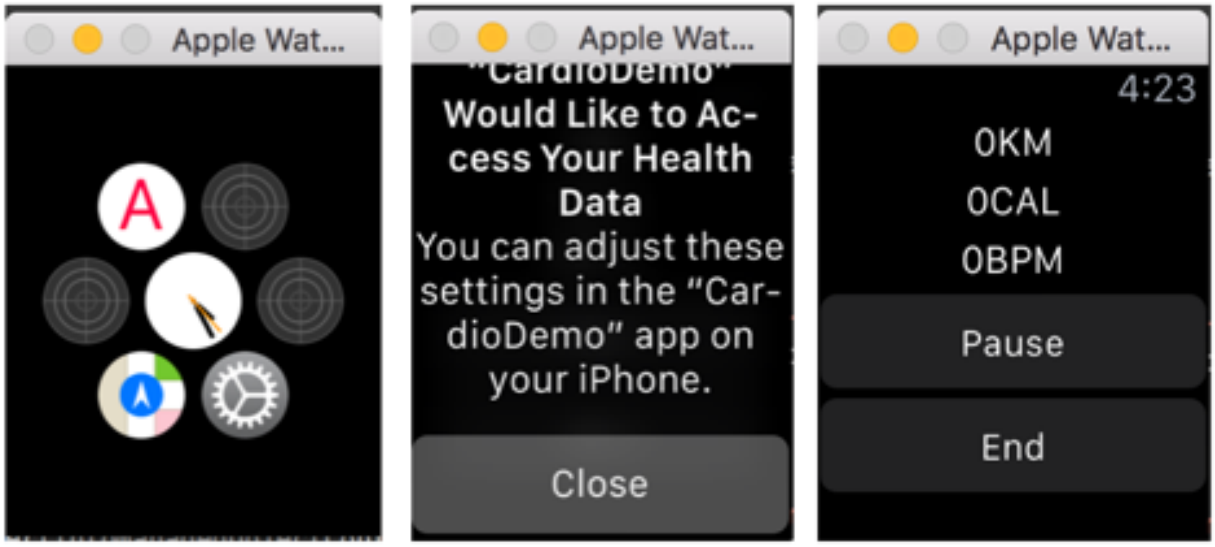

Рисунок 4.20 - Інтерфейс AppleWatch додатку

Тобто, як бачимо, розумний годинник Apple без iPhone —працює корректно. І хоча Apple згадував про WatchKit Apps — повнофункційні додатки, які, теоретично, повинні працювати без iPhone, поки що вони недоступні, і ніхто не знає, коли з'являться на ринку.

З іншого боку цей істотний недолік — як би парадоксально це не звучало — є великим плюсом. Зважаючи на те, що додаток годинника, по суті, працює на iPhone, програміст має у своєму розпорядженні увесь функціонал і дані з самого iPhone (щоправда, Apple вже попередив нас про ймовірні обмеження на енергозатратні операції). Сам же WatchKit — це такий собі «місток», що з'єднує код на iPhone з інтерфейсом на Apple Watch.

4.7. Висновки до розділу 4

Враховуючи описані особливості цільової операційної системи, для створення додатку, що відповідає новим тенденціям в плані дизайну інтерфейсу та функціональності, були виконані наступні кроки: комбіноване використання засобів статичної та динамічної розмітки Auto Layout, робота з

базою даних, використані стандартні методи зчитування даних пульсу і розвинута система захисту даних. Статично встановлюється розмітка компонентів, а динамічно - їх угруповання на екрані смартфона та годинника.

Продумування схеми даних - найважливіший момент при роботі з Core Data. Виправлення помилки, допущеної на етапі проектування архітектури, може коштувати розробнику купу часу і нервів. Ідеальний варіант, якщо після виходу в бій модель не змінюється. У реальності, якщо нам не доведеться вдаватися до ручної міграції через Migration Manager і всі зміни проковтують Lightweight Migration.

Способи взаємодії реалізовані з метою створення найкращого досвіду користувача. Навігація між частинами додатка виконується за допомогою використання шаблону Navigation від Apple. Результат відповідає уявленням про систему та узгоджений в технічній документації.

Взаємодія між девайсами відбувається за допомогою фреймворку Watch Connectivity і реалізована програмно через клас WCSession з пакету Foundation.

Домашній екран з іконками програм, і дока в нижній частині екрана, де користувачі можуть пов'язують найбільш часто використовувані програми, представляється щоразу, коли пристрій увімкнений або натискається кнопка Home. Екран має статус-бар у верхній частині екрану для відображення даних, таких як: час, рівень заряду батареї, сили сигналу, стан інтернет-з'єднання та блютузу тощо.

Програмне забезпечення мобільного додатку містить всі обумовлені в технічному завданні функції і знаходиться на ранній стадії відкритого тестування, щоб закрити ті помилки, які не виявили автоматичні тести. Весь код опублікований в публічному репозитарії в Github.

5. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, призначеного для виміру пульсу на розумних годиниках Apple Watch з відображенням на мобільному додатку на телефоні під керуванням операційної системи iOS. Інтерфейс користувача буде створений з використанням технології Auto-Layout.

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

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

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

Фактично цей метод працює за таким алгоритмом:

− визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

− для кожної функції визначаються повні річні витрати й кількість робочих часів.

− для кожної функції наоснові оцінок попереднього пункту визначається кількісна характеристика джерел витрат.

− після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

5.1. Постановка задачі техніко-економічного аналізу

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

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

− програмний продукт повинен функціонувати на розумних годиниках Apple Watch під керуванням WatchOS та додатку на iOS

− забезпечувати візуальний гарний інтерфейс та інтуітивну логіку додатку для користувача

− забезпечувати зручність і простоту взаємодії з користувачем;

− передбачати мінімальні витрати на впровадження програмного продукту.

5.1.1.Обгрунтування функцій програмного продукту

Головна функція F_0– розробка програмного продукту, який аналізує процес за вхідними даними та будує його модель для подальшого прогнозування. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір мови програмування;

F_2 – вибір оптимальної БД;

F_3 – інтерфейс користувача.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) мова програмування Swift

б) мова програмування ObjectiveC;

Функція F_2:

а) CoreData;

б) Realm.

Функція F_3:

а) інтерфейс користувача, створений за технологією Auto-Layout

б) інтерфейс користувача, створений без технології Auto-Layout.

5.1.2.Варіанти реалізації основних функцій

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

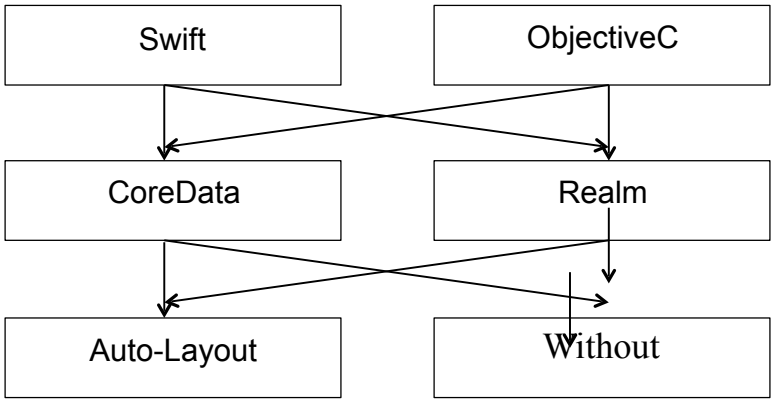

Рисунок 4.1 – Морфологічна карта

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

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	*А*	Займає менше коду, швидкість роботи, новий язик	Недостатньо бібліотек пряцює з цією мовою
	Б	Повна документація, більше людей працює з цим язиком	Низька швидкодія
F2	А	Працює з коробки, безкоштовна	Не має можлівість підтримувати великі розміри баз даних
	Б	Нова швидка БД, крос платформена, працює на багатьох OS	Висока вартість, нова технологія
F3	*А*	Під різні діагоналі єкранів	Складна технологія
	Б	Простота створення	Багато коду для різних типів діагоналей єкранів

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

Функція F1:

Вибір мови відіграє вжливу роль і обидва варіанти буде гарно порівнти а) та б).

Функція F2:

Оскільки ми не будемо мати великіх обсягів даних, а програма буде працювати тільки під ОС iOS та WatchOS то вибір БД буде CoreData а).

Функція F3:

Оскільки кількість пристроїв з різними розмірами єкраниів у ОС iOS багото то Auto-Layout потрібно використовувати а).

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1а – F2а – F3а
2. F1б – F2а – F3а

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

5.2. Обґрунтування системи параметрів ПП

5.2.1.Опис параметрів

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

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

−X1 – швидкодія мови програмування;

−X2 – об’єм програмного коду;

−X3 – час компіляції коду;

−X4 – популярність серед програмістіви.

X1: Відображає швидкодію операцій залежно від обраної мови програмування.

X2: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

X3: Відображає час, який витрачається на дії при компіляції програми.

X4: Показує індекс популярності серед програмістів

5.2.2.Кількісна оцінка параметрів

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

Таблиця 4.2 – Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	мс	70	150	200
Об’єм програмного коду	X2	кількість строк коду	350	310	150
Час компіляції коду	X3	мс	1000	800	600
Популярність серед програмісів	X4	рейтинг інтересу до язика	27	50	71

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

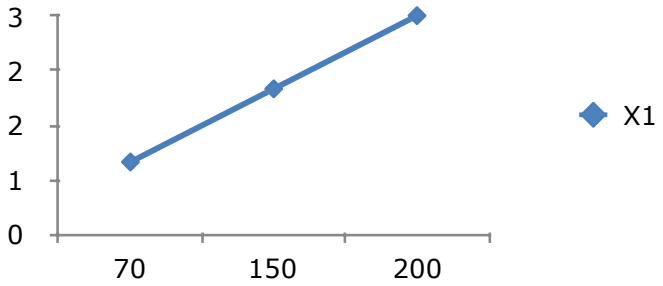

Рисунок 4.2 – X1, швидкодія мови програмування

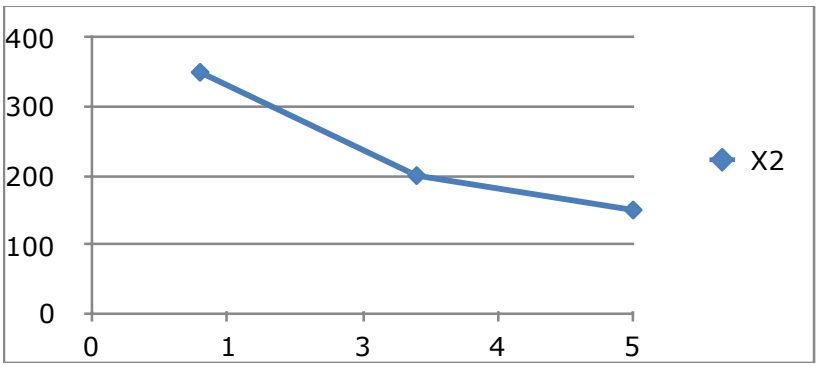

Рисунок 4.3 – X2, Об’єм програмного коду

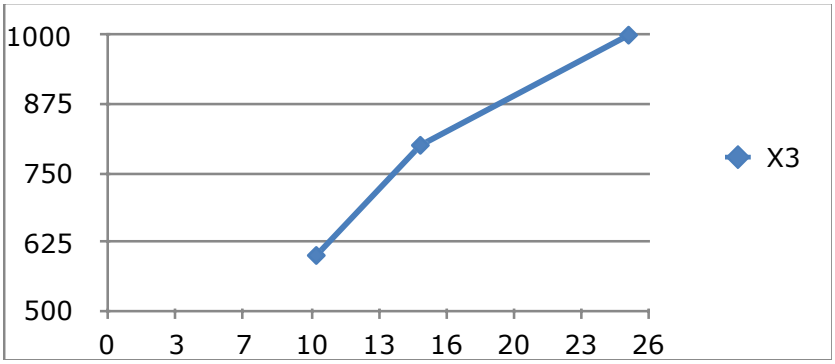

Рисунок 4.4 – X3, Час компіляції коду

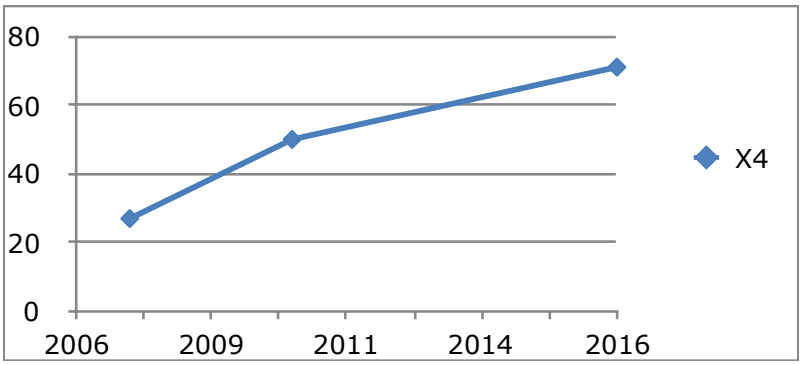

Рисунок 4.5 – X4, Популярність серед програмісів

5.2.3.Аналіз експертного оцінювання параметрів

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

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

−визначення рівня значимості параметра шляхом присвоєння різних рангів;

−перевірку придатності експертних оцінок для подальшого використання;

−визначення оцінки попарного пріоритету параметрів;

−обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	мс	2	3	3	3	3	2	4	20	5	25
X2	Об’єм програмного коду	кількість строк коду	4	2	2	5	4	4	3	24	1	1
X3	Час компіляції коду	мс	3	4	4	2	3	3	2	21	4	16
X4	Популярність серед програмісів	рейтинг інтересу до язика	5	5	5	5	5	5	5	35	-10	100
	Разом		14	14	14	15	15	14	14	100	0	142
	Порахуємо коефіцієнт узгодженості:	0.5795										

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0.58.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю4.4.

Таблиця4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	<	<	>	>	<	<	<	0,5
X1 і X3	>	>	>	<	“="	<	<	<	0,5
X1 і X4	>	>	>	>	>	>	>	>	1,5
X2 і X3	<	>	>	<	<	<	<	<	0,5
X2 і X4	>	>	>	“="	>	>	>	>	1,5
X3 і X4	>	>	>	>	>	>	>	>	1,5

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

$$a_{ij} = \begin{cases} 1{,}5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

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

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).

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

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри	Параметри				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4						
X1	1,0	0,5	0,5	1,5	3,5	0,219	22,25	0,216	100	0,215
X2	1,5	1,0	0,5	1,5	4,5	0,281	27,25	0,282	124,25	0,283
X3	1,5	1,5	1,0	1,5	5,5	0,344	34,25	0,347	156	0,348
X4	0,5	0,5	0,5	1,0	2,5	0,156	14,25	0,155	64,75	0,154
Всього:					16	1	98	1	445	1

5.3. Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо. Абсолютні значення параметрів *X2*(об'єм програмного коду) та *X1* (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3 (*Час компіляції коду) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 1000 мс або варіанту б) 800мс.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується як в таблиці 4.6.

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

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1)	А	150	3,6	0,215	0,774
F2(X2)	А	310	3,4	0,283	0,962
F3(X3,X4)	А	800	2,4	0,348	0,835
	Б	50	1	0,154	0,154

Визначаємо рівень якості кожного з варіантів:

$K_{K1} = 0,774 + 0,962 + 0,835 = 2,57$

$K_{K2} = 0,774 + 0,962 + 0,154 = 1,89$

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

5.4. Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка дизайну додатку;

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

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує накопичену інформацію та досвід працівника.

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

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

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

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

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1,

трудомісткість дорівнює: T_P =40 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$T_1 = 40 \cdot 1.7 \cdot 0.8 = 54.4$ людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм першої групи складності, степінь новизни Б), тобто T_P =20 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

$T_2 = 20 \cdot 0.9 \cdot 0.8 = 14.44$ людино-днів.

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

$T_I = (54.4 + 14.44 + 4.8 + 14.44) \cdot 8 = 704{,}64$ людино-годин;

$T_{II} = (54.4 + 14.44 + 6.91 + 14.44) \cdot 8 = 706.75$ людино-годин;

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь програміст з окладом 50000 грн., та дизайнер з окладом 45000. Визначимо зарплату за годину за формулою:

Сч = (50000 + 45000)/(22 * 8 * 2) = 135грн

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 135 \cdot 704{,}64 \cdot 1.2 = 114151.68$ грн.

II. $С_{ЗП} = 135 \cdot 706.75 \cdot 1.2 = 114493.5$ грн.

Відрахування на єдиний соціальний становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 114151.68*0.22 = 25113.36$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 114493.5*0.22 = 25188.57$ грн.

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

Так як одна ЕОМ обслуговує одного програміста з окладом 50000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12*50000*0.2 = 120000$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 120000*(1 + 0.2) = 144000$грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.3677 = 144000 * 0.22 = 31680$ грн

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

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

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

Витрати на ремонт та профілактику розраховуємо як:

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

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

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

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

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706.4*0.156*0.2*2.021 = 107.59$ грн.,

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

Накладні витрати розраховуємо за формулою:

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

Тоді, річні експлуатаційні витрати будуть:

$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$

$С_{М-Г} = С_{ЕКС} / Т_{ЕФ} = 185937.59/1706.4 = 108.96$ грн/час.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М-Г} = С_{ЕКС} / Т_{ЕФ} = 185937.59/1706.4 = 108.96$ грн/час.

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

$С_М = С_{М-Г} \cdot Т$

I. $С_М = 108.96 * 704.64 = 76777.57$грн.;

II. $С_М = 108.96 * 706.75 = 77007.48$грн.;

Накладні витрати складають 67% від заробітної плати:

$С_Н = С_{ЗП} \cdot 0,67$

I. $С_Н = 114151.68 * 0.67 = 76481.62$ грн.;

II. $С_Н = 114493.5 * 0.67 = 76710.64$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$

I. $С_{ПП} = 114151.68 + 31680 + 76777.57 + 76481.62 = 299259.98$ грн.;

II. $С_{ПП} = 114151.68 + 31680 + 76777.57 + 76481.62 = 299259.98$ грн.;

5.5. Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$К_{ТЕРj} = К_{Кj} / С_{Фj}$,

$К_{ТЕР1} = 3.48 / 299259.98 = 0.00001163$;

$К_{ТЕР2} = 2.49 / 300,061.24 = 0.0000083$;

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 0.1163 \cdot 10^{-45}$.

5.6. Висновки до розділу 5

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

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

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

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 0.1163 \cdot 10^{-4}$.

Цей варіант реалізації програмного продукту має такі параметри:

−мова програмування – Swift;

−БД CoreData;

−інтерфейс користувача, створений за допомогою нового інструменту Auto-Layout.

Даний варіант інструментів дає змогу створити оптимальни і потужний додаток на новітній мові програмування, з натвною швидкою базою даних для оброблення результатів, а також з використанням системи Auto-Layout, для вирішення проблеми різних діагоналей єкранів смартфонів.

ВИСНОВКИ

Світ технологій розвивається безперервно і динамічно. З кожним роком з'являються нові концепції та їх реалізації, а існуючі розширюють свої функціональні характеристики і можливості. Не нова і мета цієї спіралі прогресу - зробити умови життя людини якомога комфортнішими і продуктивними. Однією з причин такого бурхливого зростання є широке поширення Internet і значне збільшення швидкості передачі даних. Як зазначають в Google, до 2008 року був «Інтернет людей», тепер настав «Інтернет речей», так як пристроїв, підключених до світової мережі стало більше ніж жителів планети. Завдяки цьому взаємодію їх з людиною стало можливо винести на зовсім інший рівень.

Розумний годинник – це нове слово у світі високотехнологічних гаджетів. Будучи як би продовженням Вашого смартфону вони можуть допомогти у самих несподіваних ситуаціях – скинути небажаний виклик, подивитися прогноз погоди, дізнатися температуру тіла та багато іншого. Дуже зручним є використання таких гаджетів у якості GPS-навігатора.

Одним з основних гравців у цій сфері є компанія Apple з її рішенням - Apple Watch. У першу чергу, Apple Watch це пристрій для контролю здоров’я та активності людини. Я думаю, хто любить носити годинник, не відмовилися б від них. Великим плюсом для зайнятих людей буде і можливість переглядати повідомлення. Годинник показує нові sms і пошту, повідомлення із соціальних мереж, оповіщення від новинних програм. Усе це виводиться на екран за мить після появи на iPhone.

На даний момент іде активна розробка і вдосконалення технології «Розумний» годинник. Незважаючи на це, процеси стандартизації та глобалізації почалися порівняно недавно й на даний час існує безліч однотипних рішень, але з різною реалізацією, зав'язаною на виробників та їх технології. Завдяки зацікавленості основних гравців ринку в розвитку системи «Розумний» годинник та інтеграції в неї своїх систем та сервісів, з'явилися зрушення у бік популяризації технології та спільна зацікавленість у ній як

покупців, так і виробників.

З появою Apple Watch у багатьох програмістів виникає логічне бажання запрограмувати що-небудь для розумного годинника. А якщо вже програмувати, то краще щось корисне. Керуючись саме такою логікою, в даній роботі було спроектовано фітнес додаток для людей, що займаються бігом. Було досліджено системні можливості платформи і способи збору статистичних даних про користувача, щоб надати користувачам простий і зрозумілий інтерфейс моніторингу за станом свого здотовя під час тренування з фізичними навантаженнями.

Проведено аналіз та порівняння мов програмування Objective C та Swift, а також досліджено функціональні можливості системних фреймворків. Попри загальне враження, що Apple Watch пропонує широке поле для творчості, на жаль, поточні можливості розробки на емуляторі не збігаються із очікуваннями. Робити щось дійсно потрібне і функціональне для Apple Watch з поточною версією SDK недуже зручно. Тому для тих хто цікавиться даною розробкою рекомендовано мати справжній пристрій Apple Watch і бажано компюрер марки Apple.

У подальшому планується додати до системи і можливість експорту зібраних статистичних даних для подальшої обробки в інших системах. Наприклад відсилати результати тренувань своєму тренеру або лікарю.

ПЕРЕЛІК ПОСИЛАНЬ

1. Understanding Auto Layout:- Режим доступу: https:/developer.apple.com/library/ios/documentation/UserExperienceConceptual/AutolayoutPG. – Дата доступу: 13.02.2016.
2. Auto-Layout in iOS7 - Режим доступу: https:/www.raywenderlich.com/50317/beginning-auto-layout-tutorial-in-ios-7-part-1. – Дата доступу: 28.02.2016.
3. Erica Sadun, Mobile Programming Series: iOS Auto Layout Demystified, Second Edition / Erica Sadun. - Manning Publications, 2013. – 41 p.
4. Разработка мобильных приложений - Режим доступу: https://habrahabr.ru/company/mailru/blog/179113/. – Дата доступу: 14.02.2015.
5. Ray Wanderlich. Developing watchOS apps / Ray Wanderlich. - Mc Press, 2015. - 65 p.
6. Ive John. Design Patterns / Ive John. - Mc Press, 2015. – 165 p.
7. Brucker Peter. Scheduling Algorithms / Brucker Peter. - Springer-Verlag, 2001. - 365 p.
8. Brucker Peter. Complex scheduling Springer-Verlag / Brucker Peter, Knust Seve. Berlin press, 2006. - 210 p.
9. Design Patterns: Elements of Reusable Object-Oriented Software / [Erich Gamma, Richard Helm, Ralph Johnson et al.]. – Addison-Wesley Professional, 1994. – 395 p.
10. Luke Hohmann. Beyond Software Architecture: Creating and Sustaining Winning Solutions / Luke Hohmann. – Addison-Wesley Professional, 2003. – 352 p.
11. Осадчий Дмитро. Auto-Layout как способ построения графических и мультимедийных интерфейсов для приложений операционной системы iOS / Осадчий Дмитрий. // Системний аналіз та інформаційні технології : «САІТ-2016», 30 травня –2 червня 2016, Київ, Україна : матеріали. – К. : НТУУ «КПІ», 2016. – 113 с.
12. Is Swift Faster Than Objective-C. - Режим доступу: https://yalantis.com/blog/is-swift-faster-than-objective-c/. - Дата доступу: 28.05.2016.

13. Cocoa Touch [Електронний ресурс] - Режим доступу: https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Cocoa.html. - Дата доступу: 21.10.2015.