Київський національний торговельно-економічний університет

Кафедра програмної інженерії та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка мобільного додатку системи iOS для логістичного підприємства»

Студента 2м курсу, 5 групи, спеціальності 121«Інженерія програмного забезпечення», спеціалізації «Інженерія програмного забезпечення»

_____
підпис студента

Бердар Костянтин Сергійович

Науковий керівник
Доктор технічних наук
Професор

_____
підпис керівника

Криворучко Олена Володимирівна

Гарант освітньої програми
Доктор технічних наук
Професор

_____
підпис гаранта

Криворучко Олена Володимирівна

КИЇВ – 2019

Київський національний торговельно-економічний університет

Факультет ФОАІС Кафедра «Програмної інженерії та кібербезпеки»

Спеціальність 121 «Інженерія програмного забезпечення»

Спеціалізація «Інженерія програмного забезпечення»

Затверджую

Зав. Кафедри Криворучко

Олена Володимирівна

"__"_____2019 р.

Завдання

на випускну кваліфікаційну роботу студентові

Бердару Костянтину Сергійовичу

(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи "Розробка мобільного додатку системи iOS для логістичного підприємства"

Розробка та затвердження завдання на проект магістра (Наказ №185 від 07.11.18)

2. Строк здачі студентом закінченого роботи (проекту)_____

3. Цільова установка та вихідні дані до роботи (проекту)_____

Мета роботи - розробка мобільного застосування під операційною системою iOS для логістичного підприємства за допомогою мови програмування Swift та програмного середовища Xcode. В ході роботи визначити найбільш корисні та швидкі методи створення додатків.

Об'єкт дослідження - Додаток логістичного підприємства.

Предмет дослідження - процеси програмування додатку для замовлень їжі додому.

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ОСНОВНІ ПОНЯТТЯ СТВОРЕННЯ ПЗ ДЛЯ ЛОГІСТИЧНИХ ПІДПРИЄМСТВ

1.1 Огляд ПЗ в логістиці

1.2 Огляд зарубіжного досвіду

1.3 Варіації ПЗ в сфері логістики

1.4 Висновки до Розділу 1

РОЗДІЛ 2. ЗАСОБИ РОЗРОБКИ

2.1 Обгрунтування вибору інструментів для розробки

2.2 Середовище розробки Xcode

2.3 Огляд обраних мов програмування

2.3.1 Swift

2.3.2 Python

2.4 Висновки до Розділу 2

РОЗДІЛ 3.РОЗРОБКА ДОДАТКУ

3.1 Проектування майбутнього додатку

3.2 Дизайн додатку

3.3 Розробка додатку

3.4 Висновки до Розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

ДОДАТКИ

6. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*	*25.09.2018*	
2.	*Розробка та затвердження завдання на проект магістра (Наказ №185 від 07.11.18)*	*07.11.2018*	
3.	*Вступ та перелік літературних джерел*	*28.02.2019*	
4.	*Розробка технічного завдання*	*22.03.2019*	
5.	*Розділ 1.* (*Назва розділу 1*)	*18.04.2019*	
6.	*Розділ 2.* (*Назва розділу 2*)	*24.05.2019*	
7.	*Розділ 3.* (*Назва розділу 3*)	*21.06.2019*	
8.	*Розділ 4.* (*Назва розділу 4, при необхідності, можливе об'єднання розділів 3 ти 4*)	*20.09.2019*	
9.	*Розробка програми та методики тестування*	*18.10.2019*	
10.	*Написання наукової статті*	*27.09.2019*	
11.	*Керівництво користувача*	*23.10.2019*	
12.	*Висновки та пропозиції*	*01.11.2019*	
13.	*Здача випускної кваліфікаційної роботи на кафедру (перша перевірка)*	*13.11.2019*	
14.	*Підготовка автореферату та презентації доповіді*	*14.11.2019*	
15.	*Попередній захист випускної кваліфікаційної роботи*	*14.11.2019 – 15.11.2019*	
16.	*Здача зброшурованої випускної кваліфікаційної роботи*	*25.11.2019*	
17.	*Зовнішнє рецензування випускної кваліфікаційної роботи*	*26.11.2019*	
18.	*Підготовка до публічного захисту випускної кваліфікаційної роботи*	*02.12.2019-04.12.2019*	

7. Дата видачі завдання **«26» вересня 2018 р.**

8. Науковий керівник випускної кваліфікаційної роботи

Криворучко О.В._____

(прізвище, ініціали, підпис)

9. Гарант освітньої програми Криворучко О.В._____

(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Бердар К.С._____

(прізвище, ініціали, підпис)

11. Відгук керівника випускної кваліфікаційної роботи

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Науковий керівник випускної кваліфікаційної роботи_____

(підпис, дата)

Відмітка про попередній захист Криворучко О.В._____

(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента _____Бердар К.С.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____ Криворучко О.В.

(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О.В.

(підпис, прізвище, ініціали)

11. Відгук керівника випускної кваліфікаційної роботи

Науковий керівник випускної кваліфікаційної роботи_____

(підпис, дата)

Відмітка про попередній захист Криворучко О.В._____

(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента _____Бердар К.С.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____ Криворучко О.В.

(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О.В.

(підпис, прізвище, ініціали)

11. Відгук керівника випускної кваліфікаційної роботи

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Науковий керівник випускної кваліфікаційної роботи_____

(підпис, дата)

Відмітка про попередній захист Криворучко О.В._____

(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента _____Бердар К.С.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____ Криворучко О.В.

(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О.В.

(підпис, прізвище, ініціали)

Анотвція

Величезною актуальністю користується така послуга як доставка їжі, тим паче якщо створена з використанням мобільних додатків, а в нашому випадку під ОС iOS. Все це стало актуальним разом з появою мобільного Інтернету. Сьогодні завжди є можливість підключитися до мережі за допомогою телефону, планшета або іншого пристрою. Але відразу ж варто відзначити, що без спеціальних додатків навряд чи б була досягнута необхідна ефективність.

У випускній кваліфікаційній роботі розглядається розробка мобільного застосування під операційною системою iOS для замовлення їжі додому, за допомогою мови програмування Swift у програмному середовищі Xcode й найпопулярнішого серверного веб-фреймворку Django написаного на Python.

Annotation

Such service as delivery of food, especially if created with the use of mobile applications, and in our case under iOS, is of great relevance. All this became relevant with the advent of the mobile Internet. Today, it's always possible to connect to the network with your phone, tablet, or other device. But it should be noted right away that without the special add-ons, the required performance is unlikely to have been achieved.

The final qualification examines the development of a mobile application under the iOS operating system for ordering food home, using the Swift programming language in Xcode and the most popular server web framework Django written in Python.

Київський національний торговельно-економічний університет

iOS застосування для сервісу доставки їжі

ТЕХНІЧНЕ ЗАВДАННЯ

на _ аркушах

ТЗ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: додаток системи iOS для логістичного підприємства.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на ВКР, затверджене кафедрою програмної інженерії та кібербезпеки КНТЕУ.

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

3.1. Призначення додатку.

Основним призначенням є створення додатку для підприємства, що дає можливість користувачам замовляти їжу, курьерам розносити замовлення, а ресторанам контролювати процес.

3.2. Мета інформаційної системи.

Метою є створення клієнт-серверної інформаційної системи з можливістю використання певного функціоналу.

3.3. Цільова аудиторія додатку.

Цільова аудиторія інформаційної системи представлена наступними групами користувачів:

- Представники служб доставки
- Власники ресторанів
- Всі решта хто бажає замовити їжу

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Додаток повинен забезпечувати такі основні функції:

1) можливість динамічного оновлення;

2) варіації вибору роботи додатку під керівництвом клієнта чи кур’ера

3) клієнт має без проблем замовити їжу, та легко її оплатити.

4) кур’єр вчасно переходить до оброблення замовлення.

5) менеджер ресторану має підтверджувати через сайт готовність замовлення.

6) клієнт повинен мати можливість оцінки роботи ресторану.

Розробку виконано в програмному середовищі XCode за архітектурним шаблоном mvc. Та клієнт серверною частиною за домопогою фреймворку Django написанім на мові Python.

Додаткові вимоги:

1) наявність динамічного меню;

2) наявність анімованих кнопок;

4) наявність особистих кабінетів та їх редагування;

5) дизайн сторінок з використанням в якості базових фіолетового та білого кольорів.

5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання роботи повинна бути розроблена наступна документація:

1) програма та методика тестування;

2) керівництво користувача(клієнт);

2) керівництво користувача(кур'єр);

6. ЕТАПИ ПРОЕКТУВАННЯ

Вивчення літератури за тематикою роботи.

Розробки та узгодження технічного завдання.

Розробки структури додатку.

Розробки дизайну сторінок та графічних елементів.

Програмна реалізація додатку.

Тестування тестування.

Підготовка матеріалів текстової частини роботи.

Підготовка матеріалів графічної частини роботи.

Оформлення технічної документації роботи.

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ООП - об'єктно-орієнтоване програмування.

ПЗ - програмне забезпечення.

MVC (Model, View, Controller) - архітектурне типове рішення для організації програми.

UI (User Interface) - інтерфейс користувача.

GCC (GNU Compiler Collection) - набір компіляторів для різних мов програмування.

CMS - система керування вмістом.

LLVM - Low Level Virtual Machine.

ФС – Файлова система.

БД - База даних.

СКБД – Система керування базами даних .

МД – Модель даних.

ЕОР - Електронний освітній ресурс .

ВНЗ - Вищий навчальний заклад.

HTML - Мова розмітки гіпертекстових документів.

CSS - Каскадні таблиці стилів.

TIS - торгові Інтернет-системи.

WWDC (Apple Worldwide Developers Conference) - щорічна конференція розробників для платформи Macintosh

HTTP - протокол передачі гіпертекстовій інформації.

API - інтерфейсом прикладного програмування.

SDK - Software Development Kit

XML - розширювана мова розмітки.

PHP – гіпертекстовий препроцесор

					КНТЕУ 121– 05-03.МР	
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*	Розробка мобільного додатку системи ios для логістичного підприємства	*Стадія* / *Аркуш* / *Аркушів*
Зав. каф.		Криворучко О.В.				ПС
Керівник		Криворучко О.В.				
Гарант		Криворучко О.В.			Перелік умовних скорочень	Факультет обліку, аудиту та інформаційних систем, 2м курс, 5 група
Розроб		Бердар К.С.				

ВСТУП

З розвитком і поширенням мережі Інтернет онлайн-додатки стали більш інтерактивними, масштабованими і доступними звичайним користувачам. І тому треба рухатись назустріч майбутньому й виконувати певні задачі для його наближення.

Основа ідеї це - прагнення до своєчасного забезпечення реалізації потреб інтернет користувачів у замовленні товару(продуктів) крім того, створена система об'єднує користувачів і працівників. Дане середовище добре сприяє їх взаємодії. У даних системах є можливості як для рядового користувача, так і для адміністратора, додаток має дружній інтерфейс, передбачено подальше поліпшення і розвиток.

Актуальністю є те, що сучасна людина робить все для того щоб досягти максимального комфорту. Сьогодні одним з бажань більшості людей є вихід в Інтернет. Причому вони завжди хочуть залишатися онлайн. Саме тому величезною актуальністю користується така послуга як доставка, що створена при розробці мобільних додатків під ОС iOS.

Відразу ж варто відзначити, що без спеціальних додатків навряд чи б була досягнута необхідна ефективність.

Сьогодні фахівцями в області інформаційних технологій розробляються мобільні додатки, які дозволяють вирішувати безліч завдань, як в нашому випадку додаток для логістичної компанії, що покращить якість роботи спеціалістів і системи вцілому. Деякі служать для того щоб встановлювати з'єднання з мережею. Інші допомагають оптимізувати маршрут. Треті призначені для тих, хто шукає найвигідніші магазини. В нашому випадку такі що допоможуть замовити їжу додому.

					КНТЕУ 121– 05-03.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*	Розробка мобільного додатку системи ios для логістичного підприемства	*Стадія*	*Аркуш*	*Аркушів*
Зав. каф.		Криворучко О.В.				В		
Керівник		Криворучко О.В						
Гарант		Криворучко О.В			Вступ	Факультет обліку, аудиту та інформаційних систем, 2м курс, 5 група		
Розроб		Бердар К.С.						

В основу кожної з таких програм лягли певні утиліти, що в результаті дозволяє швидко вирішувати поставлену задачу, економити час і досягати максимально комфортного рівня життя. Більшою популярністю користується спеціалізоване програмне забезпечення, наприклад, розробка додатку для логістичного підприємства, який необхідний компаніям, що працюють в напрямках логістики.

Також саме на таких програмах можна робити непогані гроші, адже сучасні компанії не шкодують інвестицій в продукти, які могли б в будь-якій мірі оптимізувати або спростити наявні бізнес-процеси. Протягом останніх років показник, що характеризує рівень попиту на мобільні пристрої, постійно зростає. Така статистика дозволяє зробити висновок про те, що розробка мобільних додатків актуальна і доцільна.

Об'єктом дослідження виступає програмне середовище Xcode та мова розробки додатків під систему iOS Swift.

Предметом дослідження є вивчення актуальності данного продукту й створення найконкурентнішого додатку в сфері послуг доставки.

Основною метою данної роботи є розробка мобільного застосування для операційної системи iOS за допомогою мови програмування Swift.

Науковою новизною є впровадження найновіших можливостей програмування на мові Swift 4. Адже з кожним роком вона набуває більш масштабних оновлень як у плані зручності для програміста так і у швидкодії продуктів створених на її основі.

Методи і засоби програмування

Postman - це потужний набір інструментів тестування API, який став необхідним для багатьох розробників. Swift - це нова мова програмування для розробки iOS і OS X додатків. Мова програмування Python 3 - це потужний інструмент для створення програм найрізноманітнішого призначення, доступний навіть для новачків.

РОЗДІЛ 1. ОСНОВНІ ПОНЯТТЯ СТВОРЕННЯ ПЗ ДЛЯ ЛОГІСТИЧНИХ ПІДПРИЄМСТВ

1.1 Огляд ПЗ в логістиці

На сьогодні є досить незначна кількість програм, доповнень та додатків на телефон для логістичних підприємств, що в свою чергу дає нам можливість запустити власну програму без хвилювань в тому, що вона буде не конкурентна, а тим паче може навіть зайняти перше місце як вибір користувачів.

Відмічається позитивна тенденція розширення спектра логістичної діяльності з надання додаткових послуг в сфері громадського харчування. Доставка виробів (страв) на будинок розглядається як форма продажу товарів і як послуга.

Ключові слова: логістика; організація, управління; доставка; ЄАІ-центр; громадське харчування; виріб; страва; послуга; споживач.

Значна кількість ПЗ данно сфери випадає на транспортні компанії що займаються великомасштабними перевезеннями товару на фурах.

Наприклад як у подукту Maxoptra рис1.1 - Сервіс для управління логістикою міської доставки. Автоматичне планування маршрутів з урахуванням тимчасових вікон, пробок, об'ємно-масових характеристик вантажу, вимог до перевезення, оснащеності транспортного засобу, графіків роботи водіїв і кур'єрів. Мобільний додаток для водія і онлайн-сервіс для диспетчера - який величезна кількість новостворених компаній намагається скопіювати. Який вигляд має програма Maxopra зображено на рисунку 1.1.

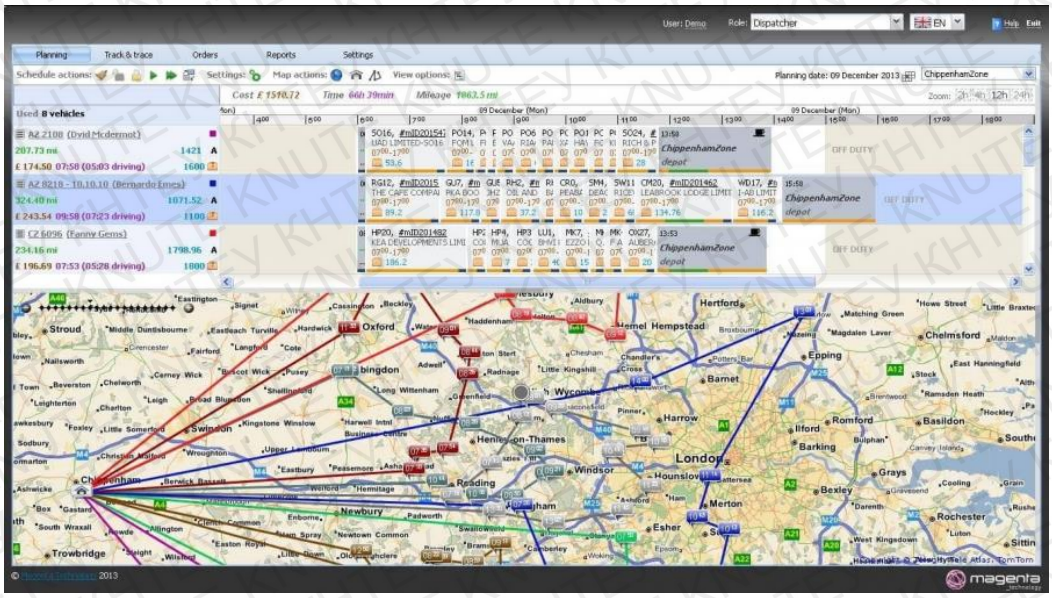


Рисунок 1.1. Інтерфейс програми Maxopra

В нашому випадку є можливість зайняття певного бізнесового пласту с використанням власного програмного продутку в сегменті логістики громадського харчування.

Особливе місце логістика займає в організації доставки готової продукції і виробів (страв) додому.

Якщо торкатися нашої тематики то серед конкурентів є всього 2 продути з якими доведеться позмагатися, це UberEat та Glovo інтерфейс яких зображено на рисунку 1.2. та 1.3.

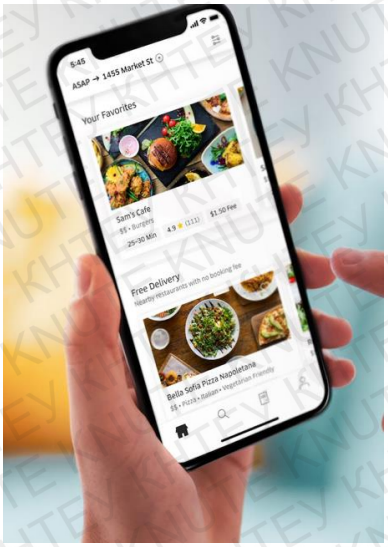


Рисунок 1.1. Зовнішній вигляд UberEats

Рисунок 1.1. Зовнішній вигляд Glovo

Можна зробити висновки, що Український ринок доставки їжі може вирости в декілька разів за найближчі роки. Ми знаходимося тільки на стартовій точці.

1.2 Огляд зарубіжного досвіду

В якості мотивації підприємств галузі громадського харчування до розробки та впровадження інновацій виступають наступні цілі: залучення клієнтури; зменшення витрат на виготовлення продукції із застосуванням більш досконалих технологій виробництва і збуту; підвищення якості продукції та забезпечення різноманітності послуг. Підприємницький підхід, новаторські рішення і реалізація інновацій можуть забезпечити успіх і конкурентоспроможність підприємств громадського харчування.

Ідея використання нових технологій в сфері надання послуг в громадському харчуванні давно хвилює зарубіжних підприємців. Традиція зустрічатися і харчуватися в кафе, барах і ресторанах широко поширена за кордоном. Гостра конкуренція між підприємствами громадського

харчування вимагає постійної уваги до з'являтимуться інновацій і, по можливості, їх якнайшвидшого впровадження в ресторанну діяльність. Удосконалення надання класичного набору послуг клієнтам старшого і середнього віку виявляється недостатнім. Інтерес власників підприємств громадського харчування розширився до винаходу різних методик і прийомів залучення молоді. Так зване «покоління Y», що не представляє свого існування без сучасних гаджетів, стає потенційною цільовою аудиторією.

Анкетування, проведене у Франції в 2014 р Національною асоціацією ресторанів, показало, що 63% респондентів користувалися новими технологіями при виборі ресторану або кафе, для ознайомлення з меню і підбору страв, для замовлення столика і обіду онлайн і електронної оплати послуг. Велика частка позитивних відповідей припадає на молодих людей від 18 до 34 років, широко використовують

технологічні нововведення: інтернет, скайп, чат, смс-повідомлення, соціальні мережі і всілякі мобільні додатки до телефонів. Це покоління користувачів інформаційних технологій, що відчуває постійний брак часу, потребує зручність, простоті і доступності послуг підприємств харчування.

Аналіз відповідей французької молоді показав наступне:

• 70% респондентів отримують необхідну інформацію про ресторанах за допомогою мобільного додатку;

• 56% здійснюють контакт з рестораном через соціальні мережі;

• 74% роблять замовлення онлайн через мобільний додаток;

• 88% шукають адреса та місцезнаходження кафе чи ресторану в мобільному додатку

Широке застосування технологічних інновацій, постійне оновлення спектра пропонованих послуг дозволяють підприємствам громадського харчування залучати нових клієнтів, особливо молодь.

Інтерактивні технології стрімко поширюються в сфері ресторанного бізнесу. У світовій практиці вже існують ресторани, відвідувачі яких можуть самостійно управляти інтер'єром приміщення і сервісом: столи представлені величезними планшетами, а підлогу, стіни і барна стійка є інтерактивними поверхнями, за допомогою яких можна «замовити» дизайн приміщення і навіть внутрішню атмосферу. Користуючись планшетом, клієнти отримують інформацію про кухню, без офіціанта самі вибирають страви, роблять замовлення, спостерігають за роботою кухаря, відправляють повідомлення, грають в комп'ютерні ігри і навіть можуть на свій смак оформити інтер'єр

Управлінські інновації. Спрямовані на поліпшення внутрішніх і зовнішніх зв'язків підприємства харчування з використанням інноваційного менеджменту. В даному випадку розробляється оригінальна маркетингова концепція, використовуються інноваційні маркетингові прийоми, проводяться PR-акції.

Нові технології успішно використовуються для оптимізації доставки продукції. Йдеться про геолокації, або відстеження, що дозволяє як професіоналам, так і клієнтам контролювати в режимі реального часу процес

доставки замовлення. Геолокація спочатку використовувалася тільки міжнародними перевізниками, проте широке використання смартфонів змінило ситуацію. В даний час все більше професійних додатків оснащуються системами, що дозволяють відслідковувати замовлення продуктів і страв від виходу з кухонь до прибуття до споживача. Сучасні гад-Жети дозволяють краще освоїти дану систему; відстеження стало доступно як рестораторам, так і їх клієнтам. Система дає можливість замовнику дізнатися в режимі реального часу, де знаходяться постачальники, оптимізувати їх поїздки і час доставки замовлення, скасувати замовлення або замінити доставщика, відстежуючи

переміщення по карті через планшет, комп'ютер або смартфон. Зв'язок підтримується через супутники і доставляється споживачеві, яким установлено на смартфон мобільний додаток

Гігантські британські фірми «Deliveroo» і «Just Eat» стали першими в системі відстеження доставок.

1.3 Дослідження доцільності створення ПЗ

Економічне значення громадського харчування визначається його місцем в регіональному відтворювальному процесі. Обслуговуючи процес суспільного виробництва, відновлюючи працездатність людини в процесі трудової діяльності, громадське харчування зайнято її відтворенням. Здійснюючи відтворення матеріальних благ і одночасне обслуговування різних галузей народного господарства і побуту населення, громадське харчування в силу своєї локальності сприяє розвитку продуктивних сил, займаючи, таким чином, одне з найважливіших місць в системі суспільного відтворення.

За останні двадцять років Українськими підприємствами накопичено значний досвід використання логістики в бізнес-процесах. У той же час логістика в сфері послуг, в тому числі в сфері громадського харчування, відноситься до малодослідженою області логістики, покликаної стосовно до даної сфери, з одного боку, сприяти зміцненню фінансової самостійності підприємств харчування, вдосконалення методів господарювання, з іншого - сталого функціонування. Тим часом діяльність підприємств громадського харчування сприяє вирішенню як економічних, так і соціальних завдань по задоволенню потреб населення в організації позадомашнього харчування і дозвілля. Для вирішення цих завдань необхідні нові підходи до розвитку прогресивних форм і методів обслуговування, до оцінки ефективності суб'єктів господарювання, зниження цін.

Логістика в сфері громадського харчування -це широкий діапазон діяльності, пов'язаний з рухом сировини від постачальника до початку виробничого процесу і готової продукції та напівфабрикатів від виходу з виробничої лінії до місця споживання відповідно до вимог споживача. Особливе місце логістика займає в організації доставки готової продукції і виробів (страв) додому.

1.4. Висновки та пропозиції до розділу 1

В першому розділі було розглянуто основні поняття щодо ПП в логістиці якою забезпечується досліджуваний інформаційний процес. Опрацьована наукова й технічна інформація об’єкту досліджень також з аналізом світового досвіду для найкращої можливості оцінки доцільності створення продукту. Досліджено Програмні продукти в вибраній тематиці та наведена доцільність створення даної роботи. Крім того було підведено під необхідність й актуальність впровадження інновацій в сфері логістики, проаналізовані наявні способи та засоби. Проведено аналіз наявних програмних засобів в сегменті, та визначено вектор роботи для найбільшого кпд при створенні додатку. Було прийнято рішення розробити програмне забезпечення, яке б не мало існуючих недоліків конкурентів та дозволяло оптимізувати процес замовлення їжі додому і вивести його на новий рівень.

РОЗДІЛ 2. ЗАСОБИ РОЗРОБКИ

2.1 Вибір інструментів для розробки

Програмування – основа основ комп'ютерної техніки. Корпорація Apple давно відома своїм умінням задавати тон розвитку індустрії на роки вперед, але з огляду на поступове сходження купертіновцев з ринку професійних рішень, надкушене яблуко асоціюється в першу чергу з споживчими товарами. Однак чергове дослідження громадської думки показує, що розробники ПЗ більш ніж задоволені свіжою пропозицією компанії – мовою програмування Swift. Згідно з інформацією ресурсу Stack Overflow, майже 80 відсотків професіоналів із задоволенням працювали або планують працювати з випущеним не так давно інструментом розробки від Apple (рис. 2.1). Дослідникам вдалося опитати 26 тисяч відвідувачів з більш ніж 150 країн світу. Близько третини постійно зайнятих в сегменті написання мобільного ПЗ респондентів працюють в основному на платформі iOS, а менше половини також займаються створенням додатків для конкуруючої ОС Android. 20 % опитаних не змогли визначитися, швидше за все, сюди входять фахівці, які регулярно розробляють програмне забезпечення для різних платформ.

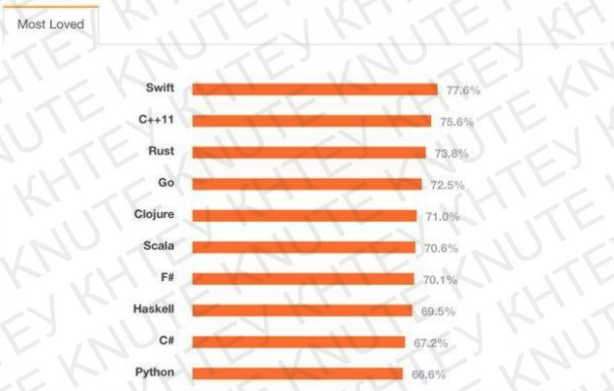


Рисунок 2.1 – Оцінки мов програмування

Swift – багатопарадигмова компільована мова програмування, розроблена

компанією Apple для того, щоб співіснувати з Objective C і бути стійкішою до помилкового коду. Swift-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 фреймворка Cocoa і Cocoa Touch. При цьому Swift надає об'єктну модель, сумісну з Objective-C. Вихідний код мовою Swift може змішуватися з кодом на C і Objective-C в одному проекті.

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.

Перед кожним розробником завжди постає питання: які саме інструменти вибрати для роботи. Головними критеріями стають простота у використанні та швидкість.

Порівнюючи Swift і Objective-C можна з впевненістю сказати, що Swift – це сучасні норми синтаксису, ефективне управління пам'яттю, висока швидкість роботи і інтерактивність. Swift – це майбутнє, яке вже не за горами.

Проаналізувавши особливості кожного з середовищ розробки, Xcode перемагає усіх вищепредставлених учасників, адже працювати в ньому – одне задоволення. Воно гнучке, швидке, потужне і здатне завжди прийти на допомогу. Xcode стає все краще, незважаючи на складні заходи, які вживаються Apple з метою утримання повного контролю над iOS додатками і пристроями. Робочий простір в Xcode тримає розробника зосередженим. Під час написання коду, в реальному часі можна побачити помилки компілятора або проблеми, а також повідомлення з докладним описом корисної інформації. Відладчик працює плавно, а симулятор – швидкий і орієнтований на користувача. Отже, Xcode перуевершує всі інші IDE.

Тому в дипломній роботі будуть використовуватися найновіші продукти від компанії Apple – середовище розробки Xcode і мова програмування Swift.

2.2 Середовище розробки Xcode

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

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

- систему управління проектом для визначення програмних продуктів;

-середовище для редагування коду, що включає можливості підсвічування синтаксису, автозаповнення коду та індексацію символів;

- просунуту програму перегляду і пошуку документації Apple;

- контекстно-залежний інспектор для перегляду інформації про виділений в коді символі;

- просунуту систему збирання проекту з перевіркою залежностей і перевіркою правил складання;

- компілятори GCC, що підтримують мови C, C ++, Objective-C, Objective-C ++ та інші;

- підтримка LLVM і Clang для мов C, C ++ і Objective-C;

- вбудоване налагодження вихідного коду з використанням GDB;

- розподілені обчислення, що дозволяють поширювати великі проекти через кілька мережевих пристроїв;

- інтелектуальна компіляція, яка прискорює час компіляції одного файлу;

- просунуті можливості по налагодженню коду;

- просунуті засоби налагодження, що дозволяють робити глобальні зміни в коді без зміни його поведінки;

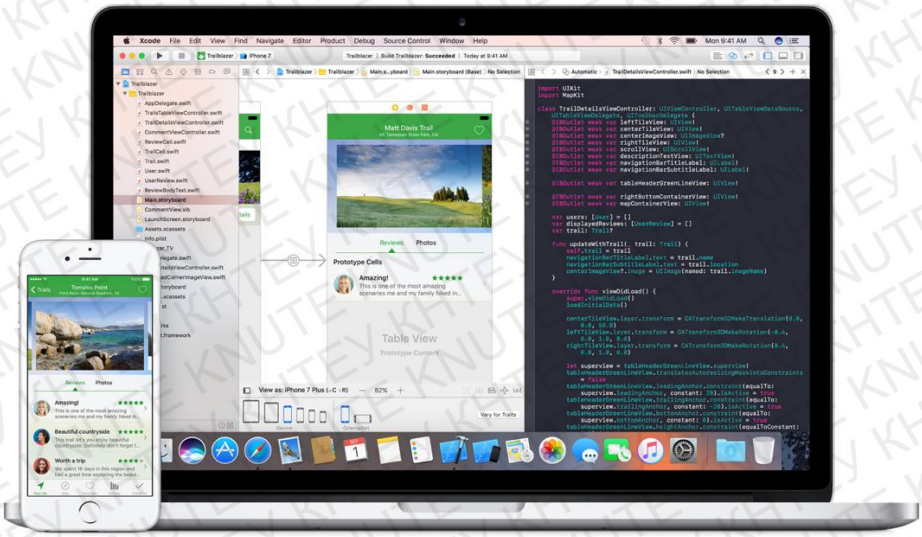


Рисунок 2.2 – Вигляд середовища та його тестування

Як виглядає процес розробки в Xcode можна побачити на (рис. 2.1).

Оскільки все так добре інтегровано, робочі процеси відчувають себе природними. Під час створення нового інтерфейсу редактор Assistant інтуїтивно представляє відповідний вихідний код на розділеній панелі вікон. Просто перетягніть мишу, щоб підключити елементи керування інтерфейсом до коду реалізації. Технології компілятора Apple LLVM аналізують ваш код, зберігаючи кожен символ, який ви бачите на відладчику LLDB, відповідно до редактора та компілятора. Під час друку той самий двигун постійно працює, знаходить помилки та пропонує виправлення для вашого коду.

Xcode навіть спілкується з веб-сайтом розробника Apple, тому ви можете ввімкнути такі послуги, як Game Center або Passbook, у своєму додатку одним натисканням кнопки. Коли ваш додаток готовий, Xcode зв’яже пакет і надішле ваш додаток у App Store.

Помічник редактора - Кнопка «Помічник» розділяє редактор Xcode на дві частини, з основного робочого документа зліва та правої області інтелектуального редактора помічника. Редактор-помічник автоматично виводить файли, які, як вважає Xcode, найбільш корисні для вас на основі роботи, яку ви виконуєте в основному редакторі. Наприклад, якщо ви

редагуєте MyClass.m в первинному редакторі, помічник автоматично покаже аналог MyClass.h.

Бар стрибків - Клацнувши на панелі переходів, розташованій у верхній частині кожної області редактора, ви можете швидко вибрати інформацію, яку потрібно переглянути для помічника редактора. Наприклад, під час редагування вихідного коду в первинному редакторі помічник може показувати заголовок колеги, підкласи або надкласи або пов'язані з ними тести.

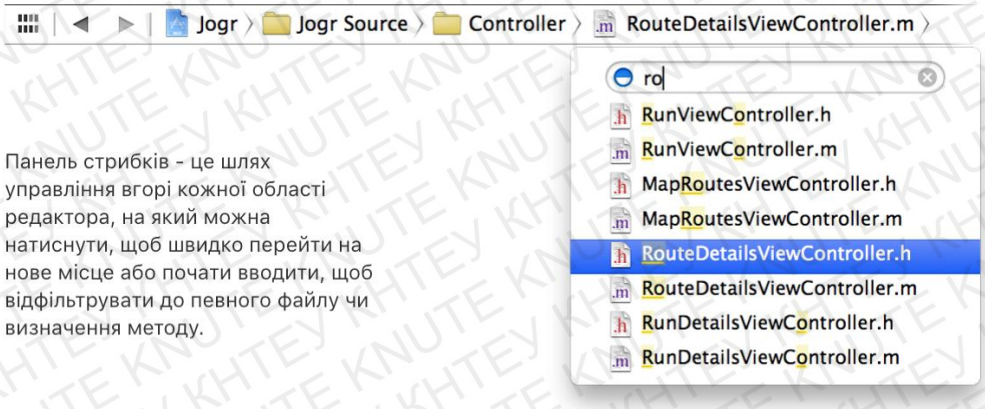


Рисунок 2.3 – Бар стрибків

Інтерфейс Builder - повністю інтегрований в Xcode IDE, полотно дизайнера Interface Builder спрощує прототип повного інтерфейсу користувача без написання коду. Прототип за лічені хвилини, а потім графічно підключіть свій інтерфейс до джерела в редакторі Xcode, виклавши вікна, кнопки та повзунки для створення функціонуючого інтерфейсу користувача Mac, iPhone або iPad. За допомогою редактора Assistant ви можете працювати над графічним дизайном поряд із вихідним кодом реалізації. Просте перетягування миші з елемента управління інтерфейсом на панель джерела створює зв’язок між кодом та інтерфейсом і навіть може створити заглушку коду для вас.

Рисунок 2.4 – Інтерфейс Builder

Редактор версій дозволяє легко порівнювати дві версії файлу, переглядати журнали фіксування, перевіряти, хто змінив код, і навіть збільшувати масштаб назад через шкалу фіксації. Редактор версій розбиває панель, щоб показати дві різні версії одного файлу. Відмінності підкреслюються під час проходження часової шкали, що розділяє перегляди редактора. Xcode також може створити локальне сховище Git для нових проектів або перевірити розміщену Subversion або Git repo. Меню верхнього рівня управління джерелом дозволяє легко виконувати операції з філіалами та злиттями, ідеально підходить для розподілених команд.

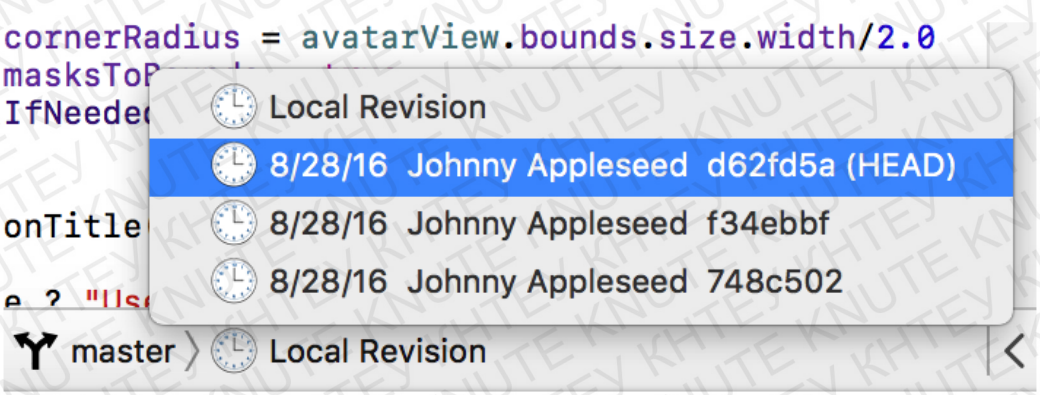


Рисунок 2.5 – Редактор версій та контроль над джерелами

Тестова розробка - це першокласний робочий процес у Xcode. Тестовий навігатор дозволяє надзвичайно легко перейти до будь-якого тесту у вашому проекті, виконати індивідуальний тест або виконати групу тестів. У редакторі Assistant є нові види тестів, які автоматично відстежують, які тести виконують код, який ви зараз редагуєте, постійно підтримуючи ваші тести та код синхронізованими.

Середовище Xcode може бути налаштовано так, щоб відповідати практично будь-якому робочому процесу, включаючи функції налаштування, такі як вкладки, поведінка та фрагменти.

Сніпети –це десятки попередньо налаштованих доповнень коду, такі як визначення нового класу чи методу, включені до бібліотеки фрагментів. Налаштувавши або додавши фрагменти, ви можете вставити часто введений код, набравши лише кілька символів.

Отримайте швидкий доступ до будь-яких файлів, якими користується ваш проект, за допомогою Open Quick (Command-Shift-O). Xcode негайно пропонує завершення пошуку, що дозволяє вибрати один і натисніть кнопку Return, щоб відкрити файл, або натисніть Option-Return, щоб відкрити його в редакторі Assistant.

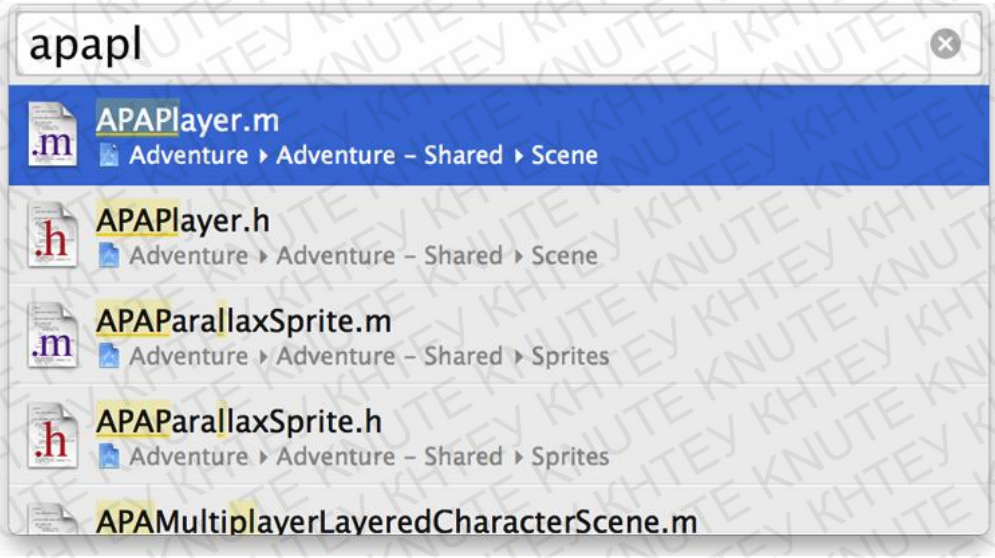


Рисунок 2.6 – Сніпети

2.3 Огляд обраних мов програмування

2.3.1 Swift

Swift - це потужна та інтуїтивно зрозуміла мова програмування для macOS, iOS, watchOS, tvOS та інших. Код Свіфта є безпечним по дизайну, але також виробляє програмне забезпечення, яке працює блискавично.

Swift - результат останнього дослідження мов програмування в поєднанні з десятилітнім досвідом створення платформ Apple. Названі параметри виражаються в чистому синтаксисі, що робить API в Swift ще простішими для читання та обслуговування. Ще краще, вам навіть не потрібно вводити напівколонки. Зазначені типи роблять код чистішим та менш схильним до помилок, тоді як модулі усувають заголовки та надають простори імен. Щоб найкраще підтримувати міжнародні мови та смайли, Strings не відповідають правилам Unicode і використовують кодування на основі UTF-8 для оптимізації продуктивності для найрізноманітніших випадків використання. Пам'ять управляється автоматично, використовуючи чіткий, детермінований підрахунок посилань, зводячи до мінімуму використання пам'яті без накладних витрат на збирання сміття.

У Swift є багато інших функцій, щоб зробити ваш код більш виразним:

- Потужні та прості у користуванні дженерики
- Розширення протоколу, які ще більше спрощують написання загального коду
- Функції першого класу та легкий синтаксис закриття
- Швидка і стисла ітерація в асортименті чи колекції
- Кортежі та кілька повернених значень
- Структури, що підтримують методи, розширення та протоколи
- Енуми можуть мати корисні навантаження та відповідність моделей підтримки
- Функціональні схеми програмування, наприклад, карта та фільтр

- Власне поводження з помилками за допомогою спроби / ловити / кидати

Swift *призначений для безпеки,* виключає цілі класи небезпечного коду. Змінні завжди ініціалізуються перед використанням, масиви та цілі числа перевіряються на переповнення, пам'ять автоматично керується та забезпечення ексклюзивного доступу до захисників пам'яті проти багатьох помилок програмування. Синтаксис налаштований так, щоб було легко визначити свій намір - наприклад, прості ключові слова з трьома символами визначають змінну (var) або константу (let). I Swift сильно використовує типи значень, особливо для часто використовуваних типів, таких як масиви та словники. Це означає, що коли ви робите копію чогось із цим типом, ви знаєте, що це не буде змінено в іншому місці.

Ще одна особливість безпеки полягає в тому, що об'єкти Swift за замовчуванням ніколи не можуть бути нульовими . Насправді компілятор Swift не дозволить вам зробити або використати об'єкт нуля з помилкою компіляції. Це робить код написання набагато більш чистим та безпечним, а також запобігає величезній категорії збоїв часу виконання програм. Однак бувають випадки, коли нуль дійсний і доцільний. У цих ситуаціях Swift має інноваційну функцію, відому як необов'язкові. Необов’язково може містити нуль , але синтаксис Swift змушує вас безпечно боротися з ним за допомогою ? синтаксис, щоб вказати компілятору, що ви розумієте поведінку, і буде безпечно поводитися з нею.

З самого раннього задуму Свіфт був побудований на швидкій основі. За допомогою неймовірно високоефективної технології компіляції LLVM код Swift перетворюється на оптимізований нативний код, який отримує максимальну користь від сучасного обладнання. Синтаксис і стандартна бібліотека також були налаштовані, щоб зробити найбільш очевидним способом написання вашого коду, а також найкращим чином,

незалежно від того, працює він у годиннику на зап'ясті або через кластер серверів.

Swift є спадкоємцем як мов C, так і Objective-C. Вона включає примітиви низького рівня, такі як типи, контроль потоку та оператори. Він також пропонує об'єктно-орієнтовані функції, такі як класи, протоколи та загальні засоби, що дає розробникам Cocoa та Cocoa Touch сенсорну ефективність та потужність, яку вони вимагають.

Свіфт може відкрити двері у світ кодування. Насправді, він був розроблений як перша програма програмування будь-кого, будь то ще в школі чи вивчаєш нові шляхи кар'єри. Для викладачів Apple створила безкоштовну навчальну програму, щоб навчати Свіфта як у класі, так і поза ним. Кодери, що вперше можуть завантажувати Swift Playgrounds - додаток для iPad, що робить роботу з кодом Swift інтерактивною та цікавою.

Прагнучі розробники додатків можуть отримати безкоштовні курси, щоб навчитися будувати свої перші програми в Xcode. Сьогодні Apple-магазини Apple по всьому світу проходять на сесіях Apple Coding & Apps, де ви можете отримати практичний досвід із кодом Swift.

З Swift 5 вам не потрібно змінювати жоден код Swift 4, щоб використовувати нову версію компілятора. Натомість ви можете почати використовувати новий компілятор і мігрувати у власному темпі, скориставшись новими можливостями Swift 5 , по одному модулю. І Swift 5 тепер представляє бінарну сумісність для додатків. Це означає, що більше не потрібно включати бібліотеки Swift у додатки, орієнтовані на поточні та майбутні випуски ОС, оскільки бібліотеки Swift будуть включені до кожного випуску ОС, що рухається вперед. Ваші програми використовуватимуть останню версію бібліотеки в ОС, і ваш код буде продовжувати працювати без перекомпіляції. Це не тільки спрощує розробку програми, але й зменшує розмір програми та час його запуску.

Swift розроблений відкрито на Swift.org, із вихідним кодом, трекером помилок, форумами та регулярними версіями розробки, доступними для всіх. Ця широка спільнота розробників, як всередині Apple, так і сотні сторонніх розробників працюють разом, щоб зробити Swift ще більш дивовижним. Існує ще більш широкий спектр блогів, подкастів, конференцій та зустрічей, де розробники спільноти діляться своїм досвідом, як реалізувати великий потенціал Свіфта.

Swift зменшує кількість коду, необхідного для повторюваних заяв і рядків (рис. 2.7). У Objective-C, працюючи з текстовими рядками, для того, щоб об'єднати дві частини інформації, потрібно зробити безліч кроків. Для додавання двох рядків Swift використовує оператор «+». Ця особливість відсутня у Objective-C. Така підтримка для об'єднаних символів і рядків має важливе значення для будь-якої мови програмування, яка використовує відображення тексту для користувача на екрані.

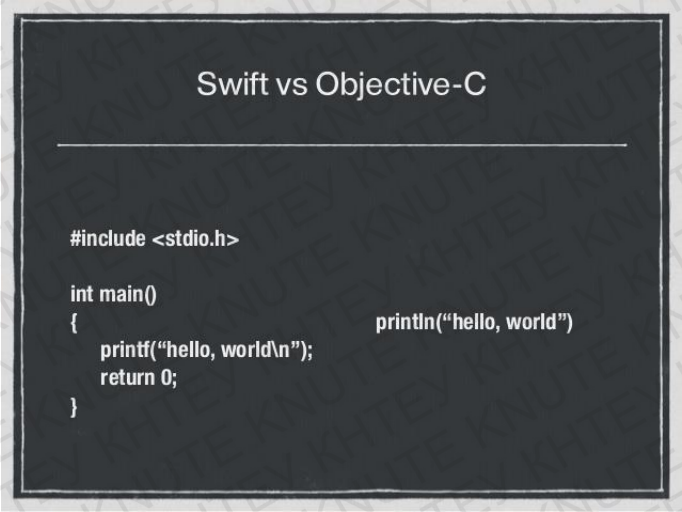


Рисунок 2.7 – Порівняння Swift і Objective-C

Система типів в Swift знижує складність в написанні коду, так як компілятор може з'ясовувати типи. Наприклад: Objective-C вимагає, щоб програмісти запам'ятовували спеціальні маркери рядків (%S, %d, %@) і використовували розділений комами список. Swift підтримує інтерполяцію рядків, що усуває необхідність запам'ятовування символів і дозволяє програмістам вставляти змінні, такі як назва ярлика або кнопки,

безпосередньо у вбудований рядок користувача. Тип виведення системи і рядки інтерполяції зменшують кількість помилок, які поширені в Objective-C. У Objective-C, якщо порушити порядок розташування або використовувати неправильний символ, мобільний додаток працювати не буде. Swift ж знову зменшує кількість написання коду вбудованій підтримці для обробки текстових рядків і даних.

• Interactive Playgrounds – спонукання до інтерактивного програмування: Одним із важливих нововведень Swift, в порівнянні з Objective-C, є можливість писати код і переглядати результати в реальному часі. Ми можемо внести деякі зміни в програмі і відразу побачити результат, так що нам не потрібно перекомпільовувати і перезапускати програму. Це має назву Interactive Playgrounds. Також тут можна використовувати Quick Look, який відображає графіку або список результатів, Timeline Assistant, який допомагає експериментувати з кодом UI (інтерфейсом користувача) або продивлятися повний цикл створення анімацій. Даний код можна перенести в основний проект (рис. 2.8).

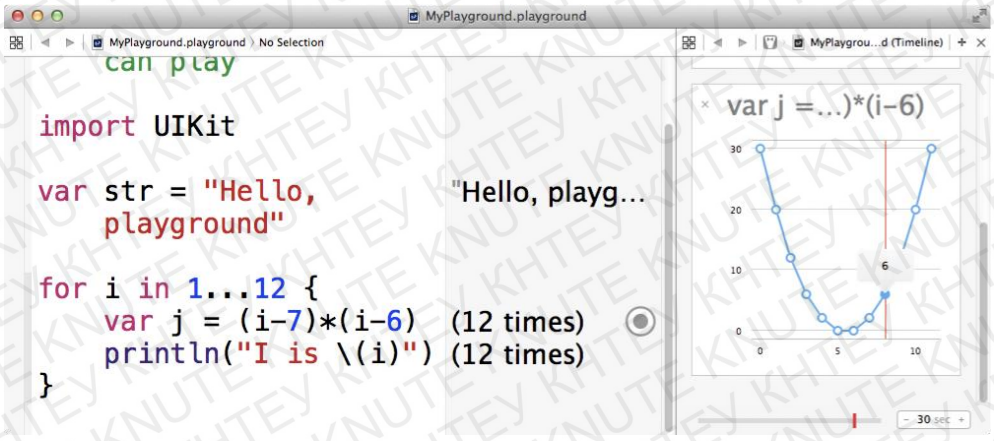


Рисунок 2.8 – Interactive Playgrounds

Apple додала вбудоване виконання коду під час його написання, щоб допомогти програмістам створювати шматки коду або писати алгоритми і зразу ж отримувати результат, що може поліпшити швидкість написання коду, тому що та модель, яка традиційно потрібна програмісту,

може бути замінена візуалізацією даних в пісочниці. Програмування – повторюваний процес, і кожен strain може бути зменшений або використаний на додаток до творчого процесу. Це зробить роботу програмістів більш продуктивною і звільнить їх від зайвої роботи, яку додавали програмістам традиційні компілятори.

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

2.3.2 Python

Python - це універсальний сучасний ЯП високого рівня, до переваг якого відносять високу продуктивність програмних рішень і структурований, добре читається код. Синтаксис Пітона максимально полегшений, що дозволяє вивчити його за порівняно короткий час. Ядро має дуже зручну структуру, а широкий перелік вбудованих бібліотек дозволяє застосовувати значний набір корисних функцій і можливостей. ЯП може використовуватися для написання прикладних програм, а також розробки WEB-сервісів.

Python може підтримувати широкий перелік стилів розробки додатків, в тому числі, дуже зручний для роботи з ООП і функціонального програмування.

Один з найпопулярніших інтерпретаторів мови - CPython, написаний на Сі . Поширюється це середовище розробки безкоштовно по вільної ліцензії. Інтерпретатор підтримує більшість популярних платформ.

Пітон активно розвивається. Приблизно раз в 2 роки виходять оновлення. Важливою особливістю мови є відсутність таких стандартів кодування як ANSI, ISO і деяких інших, вони працюють завдяки інтерпретатора.

Пітон підтримує практично всі поширені операційні системи. Він може прекрасно працювати на кишенькових комп'ютерах, так і на великих серверах. У разі, якщо платформа значно застаріває, вона виключається з підтримки ядра. Наприклад, версії мови, починаючи від 2.6, вже не працюють з платформами Windows 95, 98 і ME. У разі необхідності можна скористатися більш старими версіями, відмовившись від застосування сучасних інструментів мови. І тоді додаток буде працювати в тому числі з цими ОС. Для старих версій періодично виходять патчі. Мова також може підтримувати роботу з віртуальною машиною Java. Мова має чітко структуроване семантичне ядро і досить простий синтаксис. Все, що пишеться на цій мові, завжди легко читаємо. У разі необхідності передати аргументи мову використовує функцію call-by-sharing.

Набір операторів в мові цілком стандартний. Зручна особливість синтаксису - це форматування тексту коду за допомогою розбивки їх на блоки за допомогою відступів, які створюють натисканням клавіш «Space» і «Tab». У синтаксисі відсутні фігурні або операторні дужки, що позначають початок і кінець блоку. Таке рішення помітно скорочує кількість рядків тіла програми і привчає програміста дотримуватися хороший стиль і акуратність при написанні коду.

У 2018 році в Пітоні були змінені деякі ключові терміни, але це скоріше спростило розуміння. А тому проблем у розробників при вивченні документації не виникає.

Синтаксис Python має на увазі обов'язкове визначення типу даних для змінних, констант, масивів, списків і т.д. Основні типи нічим не відрізняються від інших мов з жорстко заданою типизацией.

Найважливіші типи:

1. Числові: цілі, дробові, речові з плаваючою точкою, комплексні.
2. Логічні: тип для зберігання значень алгебри логіки - «істина» або «брехня».
3. Строкові: містять символи Юнікоду, в тому числі, html-код.
4. Списки - впорядковані масиви змінних.
5. Кортежі - масив упорядкованих констант, тобто значень, які не можуть змінюватися в процесі роботи.
6. Безлічі - масиви невпорядкованих даних.
7. Словники - спеціалізований масив, що складається з пари - «ключ» - «значення».
8. Байти, масиви байтів - зазначені області пам'яті для зберігання зображень (jpg, gif і т.д.), pdf-документів та інших файлів.

1.4. Висновки та пропозиції до розділу 1

Тема розробки мобільних застосувань для платформи iOS є досить цікавою і представляє собою широке поле для дослідження в галузі розробки мобільного програмного забезпечення. Актуальність теми підкреслюється широким спектром можливостей для втілення ідей у вигляді мобільного додатку для логістики. В даному розділі було проаналізовані основні інструменти для розробки мобільного програмного забезпечення під платформу iOS. Також було доведено і обгрунтовано правильність вибору засобів для проектування. Перед кожним розробником завжди постає питання: які саме інструменти вибрати для роботи. Головними критеріями стають простота у використанні та

швидкість. На основі проведеного аналізу встановлено, що найкращим середовищем розробки є Xcode, а мовою програмування – Swift.

РОЗДІЛ 3. РОЗРОБКА ДОДАТКУ

3.1 Проектування майбутнього додатку

Сьогодні у нас є багато варіантів архітектурних патернів проектування:

- **MVC;**
- **MVP;**
- **MVVM;**
- **VIPER.**

Перші три з них припускають призначення сутностей додатки в одну з 3 категорій:

Models - відповідальні за дані домену або шар доступу до даних, який маніпулює даними, наприклад,
клас **Person** або **PersonDataProvider** ;

Views - відповідальні за рівень представлення (**GUI**); для навколишнього середовища iOS це все, що починається з префікса **UI** ;

- **Controller / Presenter / ViewModel** - посередник між **Model** і **View** ; в цілому відповідає за зміни **Model** , реагуючи на дії користувача, виконані на **View** , і оновлює **View** , використовуючи зміни з **Model** .

Маючи розділені суті, ми можемо:

- краще розуміти їх;

повторно їх використовувати (в основному застосовується до **View** і **Model**);

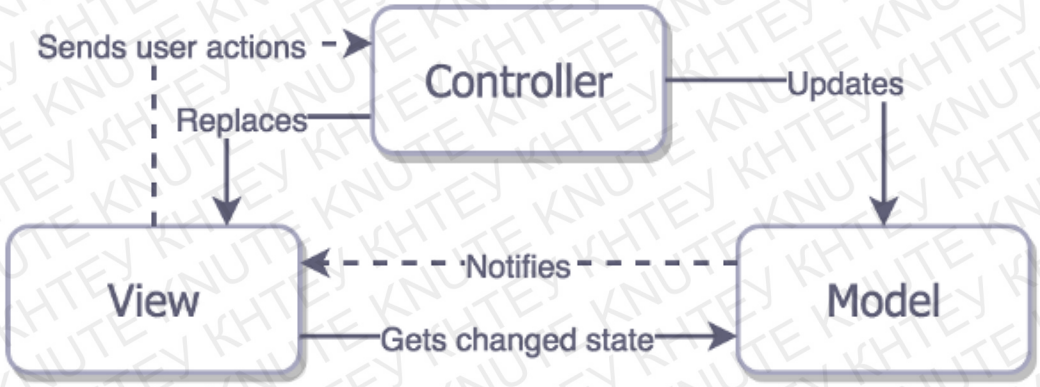
тестувати їх окремо один від одного.

Давайте почнемо з MV (X) патернів і пізніше повернемося до VIPER.

MVC

Як було раніше

Перш ніж обговорювати бачення MVC компанією Apple, давайте подивимося на [традиційну версію](#) .

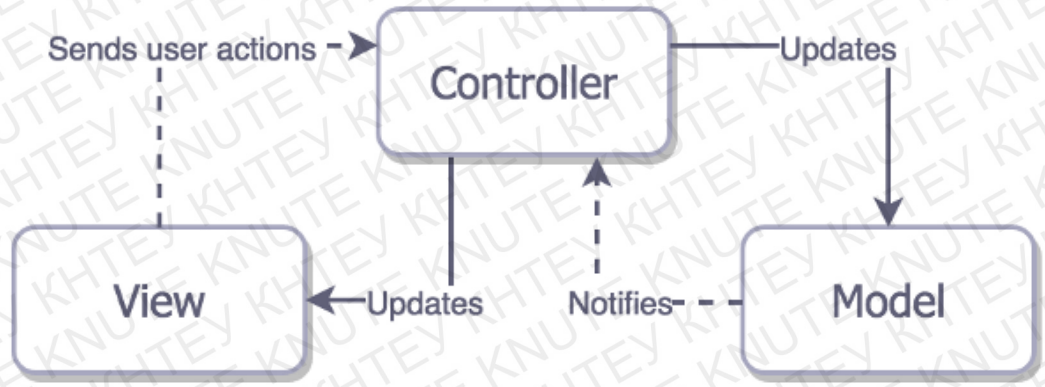


У традиційному MVC **View** не зберігає стану в собі. **Controller** просто «рендерить» **View** при змінах **Model** . Наприклад, веб-сторінка повністю перевантажується після того, як ви натиснете на посилання для переходу в інше місце. Хоча можна реалізувати традиційний MVC в середовищі iOS, це не має особливого сенсу через архітектурної 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.

MVC від Apple

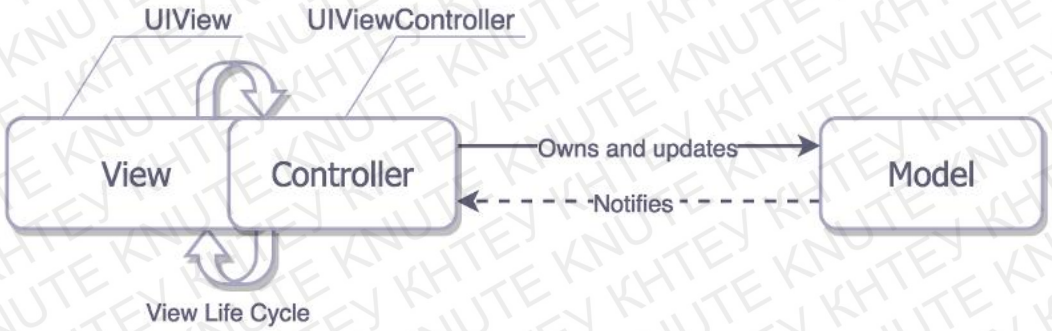
очікування



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?

реальність



Cocoa MVC заохочує вас писати **Massive** View Controller, тому що контролер настільки залучений в життєвий цикл **View** , що важко сказати, що він є окремою сутністю. Хоча у вас все ще є можливість відвантажити частину бізнес-логіки і перетворення даних в **Model** , коли справа доходить до відвантаження роботи у **View** , у вас не так багато варіантів. У більшості випадків вся відповідальність **View** полягає в тому, щоб відправити дії до контролера. У підсумку все закінчується тим, що View Controller стає делегатом і джерелом даних, а також місцем запуску і скасування серверних запитів і, в общем-то, всього чого завгодно.

Скільки разів ви бачили такий код:

```
var userCell = tableView.dequeueReusableCellWithIdentifier("identifier") as UserCell
```

```
userCell.configureWithUser(user)
```

View як осередок конфигурируется безпосередньо з **Model** . Таким чином порушуються принципи MVC, але такий код можна побачити дуже часто, і, як правило, люди не розуміють, що це неправильно. Якщо ви строго прямуєте MVC, то повинні налаштовувати осередок всередині контролера і не передавати **Model** під View, що збільшить **Controller** ще більше.

Cocoa MVC обгрунтовано розшифровують як Massive View Controller.

Проблема не очевидна, поки справа не доходить до [юніт-тестів](#) (сподіваюся, що в вашому проекті воно все ж доходить). Так як View Controller тісно пов'язана з **View** , її стає важко тестувати, і доводиться йти витонченим шляхом, замінюючи **View** [Mock-об'єктами](#)і імітуючи їх життєвий цикл, а також писати код View Controller таким чином, щоб бізнес-логіка була по максимуму відокремлена від коду view layout.

Давайте подивимося на простий приклад з «плейграунда»

```
import UIKi
struct Person { // Model
  let firstName: String
  let lastName: String
}
class GreetingViewController : UIViewController { // View + Controller
  var person: Person!
  let showGreetingButton = UIButton()
  let greetingLabel = UILabel()
  override func viewDidLoad() {
    super.viewDidLoad()
    self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
.TouchUpInside)
  }
  func didTapButton(button: UIButton) {
    let greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
    self.greetingLabel.text = greeting
      }
  // layout code goes here
}
// Assembling of MVC
let model = Person(firstName: "David", lastName: "Blaine")
let view = GreetingViewController()
```

view.person = model;

Збірка MVC може бути виконана в «презентує» View Controller.

Здається, це складно протестувати, чи не так? Ми можемо виділити генерацію вітання в новий клас *GreetingModel* і тестувати її окремо, але ми не можемо протестувати логіку уявлення (хоч в прикладі її не так багато) всередині *GreetingViewController* без виклику методів життєвого циклу **View** безпосередньо (*viewDidLoad, didTapButton*), що може привести до завантаження всіх UIView, і це погано для юніт-тестів.

Насправді, тестування UIViews на одному симуляторі (наприклад, iPhone 4S) не гарантує, що він буде працювати нормально на інших пристроях (наприклад, iPad), так що я рекомендую прибрати галочку *Host Application*з конфігурації таргета юніт-тестів і запускати їх на симуляорі, не включаючи сам додаток.

*Взаємодія між **View** і **Controller** насправді [не особливо піддається тестуванню за допомогою юніт-тестів]* .

Після всього сказаного може здатися що, Cocoa MVC є досить поганим вибором патерну. Але давайте оцінимо його з точки зору **ознак хорошою архітектури** , визначених на початку статті:

- **розподіл** : **View** і **Model** насправді розділені, але **View** і **Controller** тісно пов'язані;
- **тестованих** : через погане розподілу ви, ймовірно, будете тестувати тільки **Model** ;
- **простота використання** : найменшу кількість коду серед інших патернів. До того ж він виглядає зрозумілим, тому його легко може підтримувати навіть недосвідчений розробник.

Cocoa MVC - це розумний вибір, якщо ви не готові інвестувати багато часу в свою архітектуру і відчуваєте, що патерн з більш високою вартістю обслуговування вашому невеликому проекту або стартапу буде не по кишені.

Cocoa MVC є найкращим архітектурним патерном з точки зору швидкості розробки.

Core Data - це потужний і гнучкий фреймворк для зберігання і управління графом вашої моделі, який заслужено займає своє місце в арсеналі будь-якого iOS-розробника. Напевно ви, як мінімум, чули про це фреймворку, і не один раз, і якщо з якихось причин ви його ще не використовуєте, - то саме час почати це робити.

Так як гола теорія, як правило, досить нудна і погано засвоюється, розглядати роботу с Core Data ми будемо на практичному прикладі, створюючи додаток. Такі поширені приклади роботи з Core Data, як «Список справ» і їм подібні, на мій погляд, не дуже підходять, так як використовують всього одну сутність і не використовують взаємозв'язку, що є істотним спрощенням роботи з даними фреймворком. У даній статті ми розробимо програму, де буде використовуватися кілька сутностей і взаємозв'язків між ними.

Передбачається, що читач знайомий з основами розробки під iOS: знає Storyboard і розуміє MVC, вміє використовувати базові елементи управління. Я сам переключився на iOS недавно, тому, можливо, в статті є помилки, неточності або ігнорування *best practices*, Прохання за це сильно не штовхати, краще аргументовано ткнути носом, ніж допоможете мені і іншим початківцям iOS-розробникам. Я буду використовувати Xcode 7.3.1 і iOS 9.3.2, але все повинно працювати і в інших версіях.

Загальні відомості про Core Data

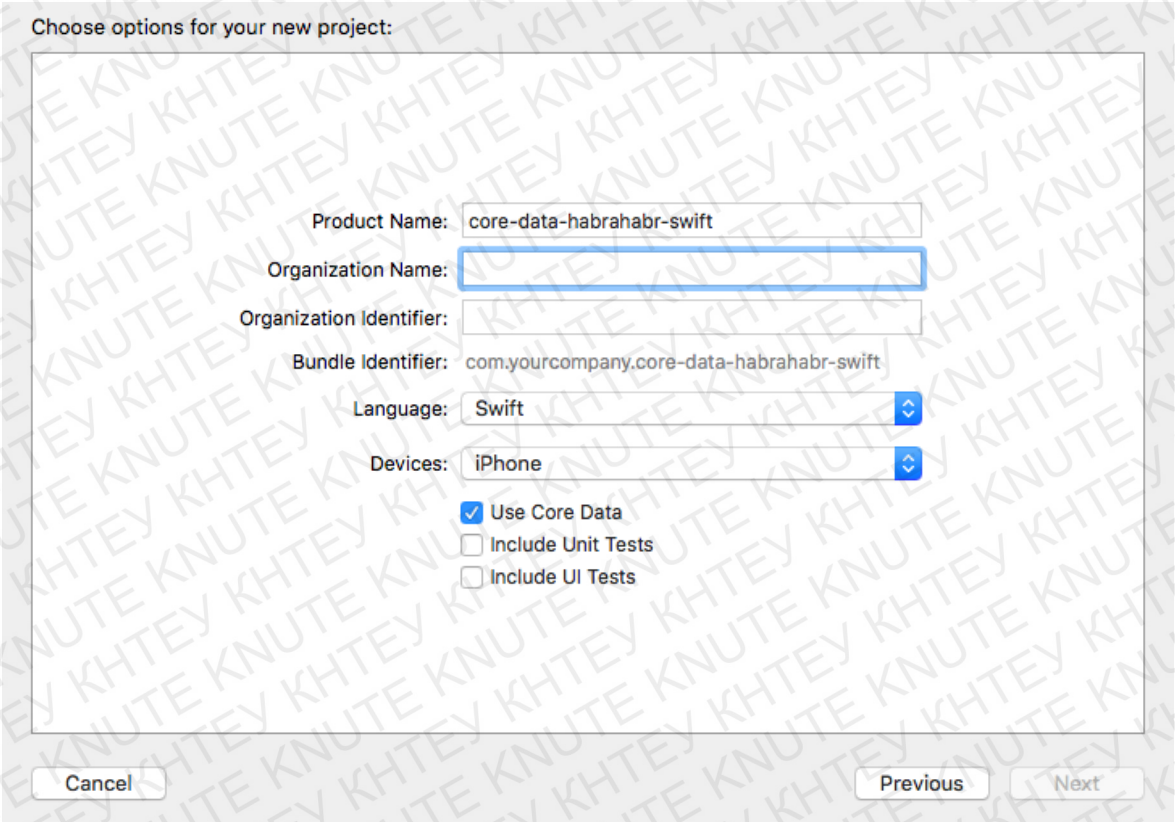
Як було сказано вище, Core Data - це фреймворк для зберігання і управління об'єктним графом вашої моделі даних. Звичайно, управляти і, тим більше, зберігати дані можна і без Core Data, але з цим фреймворком це набагато приємніше і зручніше.

На мій погляд, важливо зрозуміти основні компоненти і принцип роботи Core Data відразу цілком. Тобто крива навчання передбачає поріг входу, трохи вище середнього, якщо так можна висловитися. Ключовими компонентами Core Data, які використовуються завжди, є такі:

- **managed object model** (керована об'єктна модель) - фактично це ваша модель (в парадигмі MVC), яка містить усі сутності, їх атрибути і взаємозв'язку;
- **managed object contexts** (контекст керованого об'єкта) - використовується для управління колекціями об'єктів моделі (в загальному випадку, може бути кілька контекстів);
- **persistent store coordinator** (координатор постійного сховища) - посередник між сховищем даних і контекстом, в яких ці дані використовуються, відповідає за зберігання даних і їх кешування

Звичайно, Core Data не відмежовує тільки цими компонентами (деякі інші ми розглянемо нижче), але ці три складають основу фреймворка і дуже важливо зрозуміти їх призначення і принцип роботи.

Давайте, продовжимо розгляд Core Data на прикладі.
Створіть новий проект на основі шаблону **Single View Application** і на сторінці вибору опцій нового проекту поставте прапорець **«Use Core Data»** .



При установці даного прапорця Xcode додасть в проект порожню модель даних і деяку

кількість програмного коду для роботи з Core Data. Зрозуміло, можна почати використовувати Core Data вже в існуючому проекті: в цьому випадку треба самостійно створити модель даних і написати відповідний програмний код.

За замовчуванням, Xcode додає код для роботи з Core Data в клас делегата додатки (**AppDelegate.swift**). Давайте розглянемо його більш детально, він починається з коментаря:

```
// MARK: - Core Data stack
```

Тут чотири змінні, всі вони не започатковано за допомогою замикання. Однак, перша з них, **applicationDocumentsDirectory**- просто допоміжний метод, який повертає директорію для зберігання даних. За замовчуванням, це **Document Directory**, можна змінити, але малоймовірно, що вам це дійсно треба. Реалізація проста і не повинна викликати труднощів для розуміння.

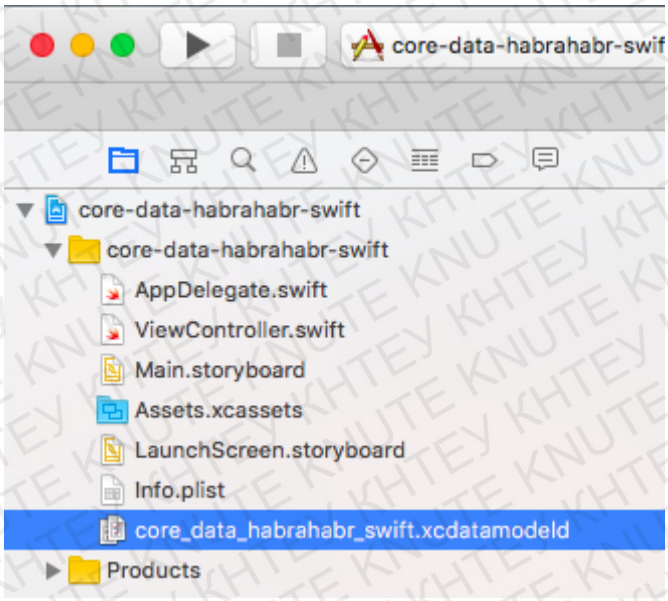
```
lazy var applicationDocumentsDirectory: NSURL = {
    let urls = NSFileManager.defaultManager().URLsForDirectory(.DocumentDirectory, inDomains: .UserDomainMask)
    return urls[urls.count-1]
}()
```

Наступне визначення - **managedObjectModel**- більш цікаво, так як має саме безпосереднє відношення до Core Data:

```
lazy var managedObjectModel: NSManagedObjectModel = {
    let modelURL = NSBundle.mainBundle().URLForResource("core_data_habrahabr_swift", withExtension: "momd")!
    return NSManagedObjectModel(contentsOfURL: modelURL)!
}()
```

Логіка програмного коду нехитра - отримуємо з збірки додатку якийсь файл з розширенням **momd**і створюємо на підставі його об'єктну модель даних. Залишилося з'ясувати, що це за файл такий. Подивіться на файли в **Навігаторі проекту (Project**

navigator) , там ви знайдете файл з розширенням **xdatamodel**- це наша модель даних Core Data (як з нею працювати ми розглянемо трохи пізніше), яка при компіляції проекту включається в файл-збірку додатки з розширенням **momd**.



Йдемо далі, - **persistentStoreCoordinator**- найбільш об'ємне визначення, але, незважаючи на кілька страхітливий вигляд, не варто його лякатися - більшу частину коду займає обробка винятків:

```
lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator = {
    let coordinator = NSPersistentStoreCoordinator(managedObjectModel:
self.managedObjectModel)
    let url =
self.applicationDocumentsDirectory.URLByAppendingPathComponent("SingleViewCoreData.sqli
te")
    var failureReason = "There was an error creating or loading the application's saved
data."
    do {
      try coordinator.addPersistentStoreWithType(NSSQLiteStoreType, configuration: nil,
URL: url, options: nil)
    } catch {
      var dict = [String: AnyObject]()
      dict[NSLocalizedDescriptionKey] = "Failed to initialize the application's saved data"
      dict[NSLocalizedFailureReasonErrorKey] = failureReason
```

```
            dict[NSUnderlyingErrorKey] = error as NSError
            let wrappedError = NSError(domain: "YOUR_ERROR_DOMAIN", code: 9999,
userInfo: dict)
            NSLog("Unresolved error \(wrappedError), \(wrappedError.userInfo)")
            abort()
        }
        return coordinator
    }()
```

Тут на основі об'єктної керованої моделі створюється координатор постійного сховища. Потім ми визначаємо, де саме повинні зберігатися дані. І на завершення підключаємо власне саме сховище (**coordinator.addPersistentStoreWithType**), передавши відповідного методу в якості параметрів тип сховища і його розташування. За замовчуванням використовується **SQLite** . У двох інших параметрах можуть передаватися додаткові параметри і опції, але на даному етапі нам це не треба, тому просто передамо **nil**.

Останнє визначення - **managedObjectContext**- впевнений, проблем з ним не буде:

```
    lazy var managedObjectContext: NSManagedObjectContext = {
        let coordinator = self.persistentStoreCoordinator
        var managedObjectContext = NSManagedObjectContext(concurrencyType:
.MainQueueConcurrencyType)
        managedObjectContext.persistentStoreCoordinator = coordinator
        return managedObjectContext
    }()
```

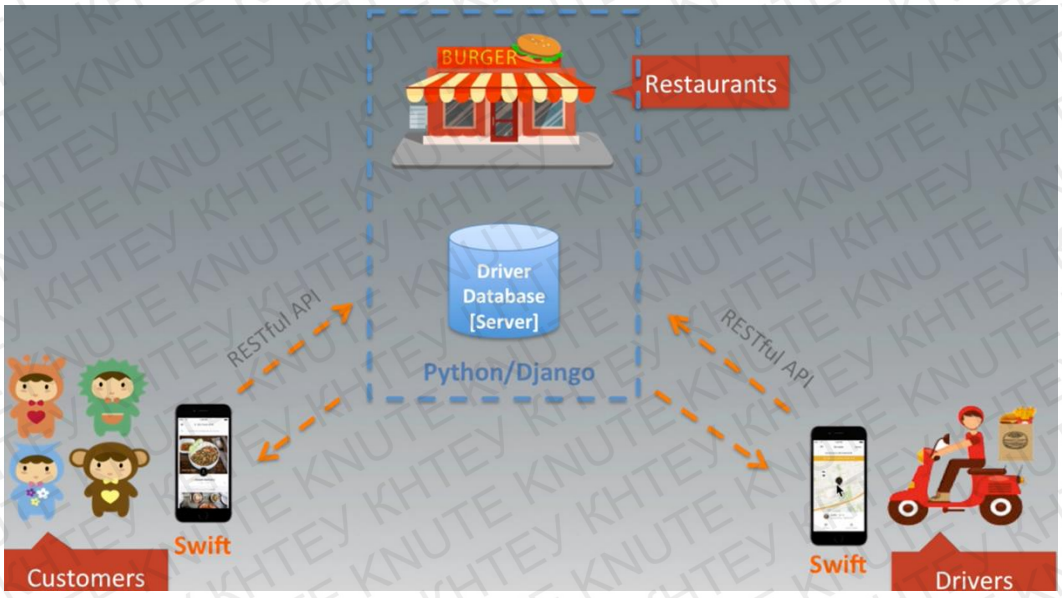
Тут ми створюємо новий контекст керованого об'єкта і присвоюємо йому посилання на наш координатор постійного сховища, за допомогою якого він і буде читати і писати необхідні нам дані. Деталь, яка заслуговує уваги - аргумент конструктора **NSManagedObjectContext**. У загальному випадку, може бути кілька робочих контекстів виконуваних в різних потоках (наприклад, один для інтерактивної роботи, інший - для фонової подгрузки даних). Передаючи в якості аргументу **MainQueueConcurrencyType**, ми вказуємо, що даний контекст повинен бути створений в основному потоці.

Також у нас тут є одна допоміжний функція для зручності збереження контексту. Сенс її

очевидний - запис даних відбувається тільки в тому випадком, якщо вони дійсно були змінені.

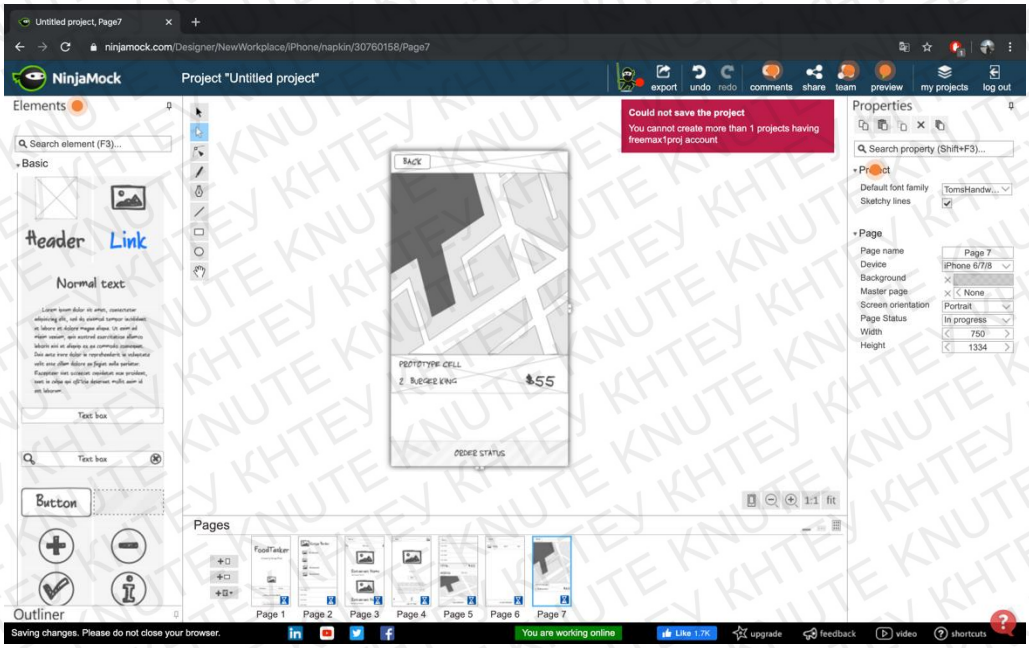
```
func saveContext () {
    if managedObjectContext.hasChanges {
        do {
            try managedObjectContext.save()
        } catch {
            let nserror = error as NSError
            NSLog("Unresolved error \(nserror), \(nserror.userInfo)")
            abort()
        }
    }
}
```

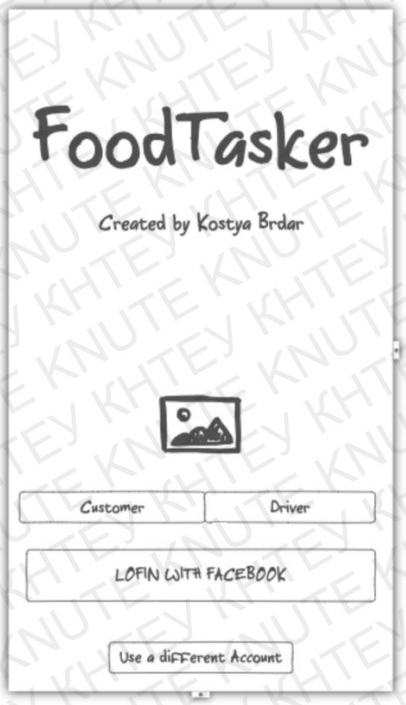
Тут важливо відзначити: вся робота з даними (створення, модифікація, видалення) завжди відбувається в рамках будь-якого контексту . Фактична запис в сховище буде виконана тільки при явному виклику функції збереження контексту.



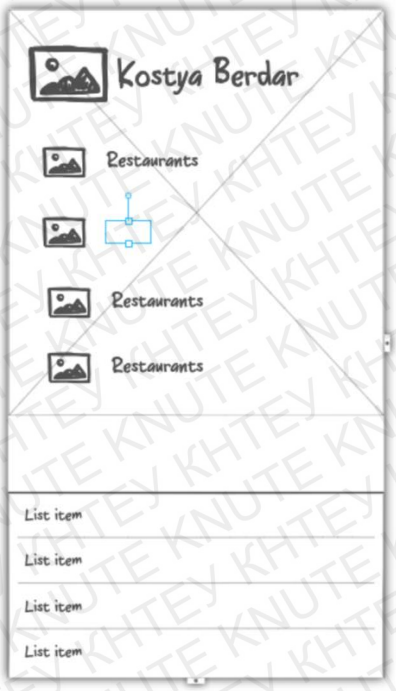
3.2 Дизайн додатку

Для створення вигляду додатку було використано сайт https://ninjamock.com/





Авторизация



Боковое Меню

Список ресторанов

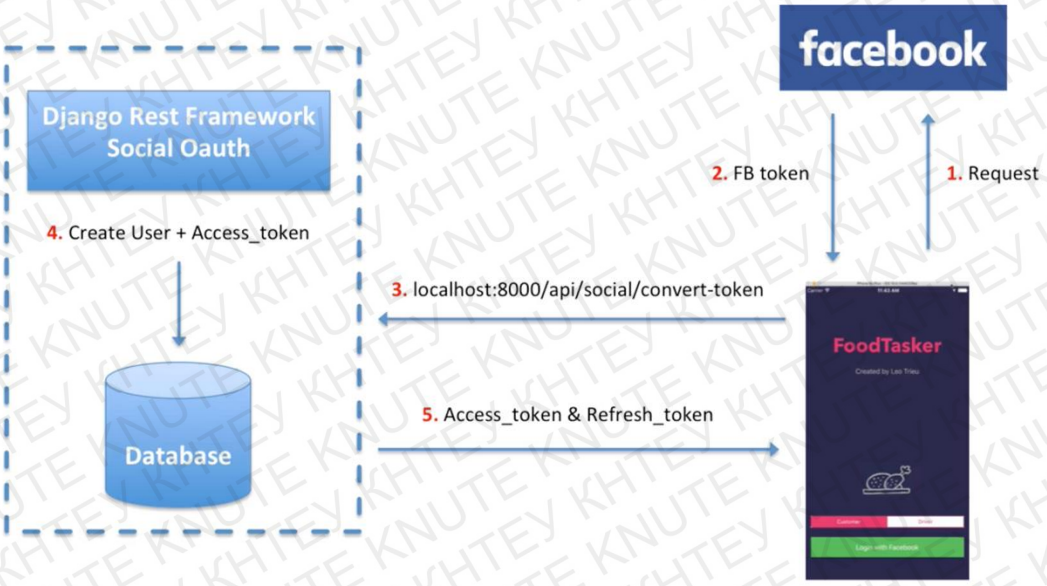
Замовлення

Дизайн окна заказов

Оплата заказа

Вигляд вікна єтапу замовлення

3.3 Розробка додатку



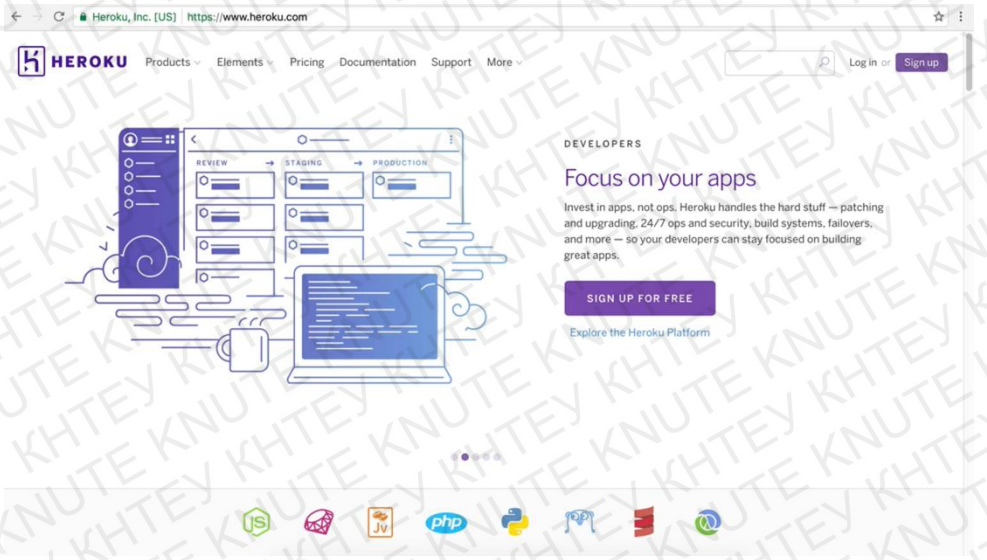
Авторизация у фейсбуці через токени

```
import UIKit
import FBSDKLoginKit
class LoginViewController: UIViewController {
@IBOutlet weak var bLogin: UIButton!
@IBOutlet weak var bLogout: UIButton!
@IBOutlet weak var switchUser: UISegmentedControl!
var fbLoginSuccess = false
var userType: String = USERTYPE_CUSTOMER
override func viewDidLoad() {
super.viewDidLoad()
if (AccessToken.current != nil) {
bLogout.isHidden = false
FBManager.getFBUserData(completionHandler: {
self.bLogin.setTitle("Continue as \(User.currentUser.email!)",
for: .normal)
//self.bLogin.sizeToFit()
})}}
```

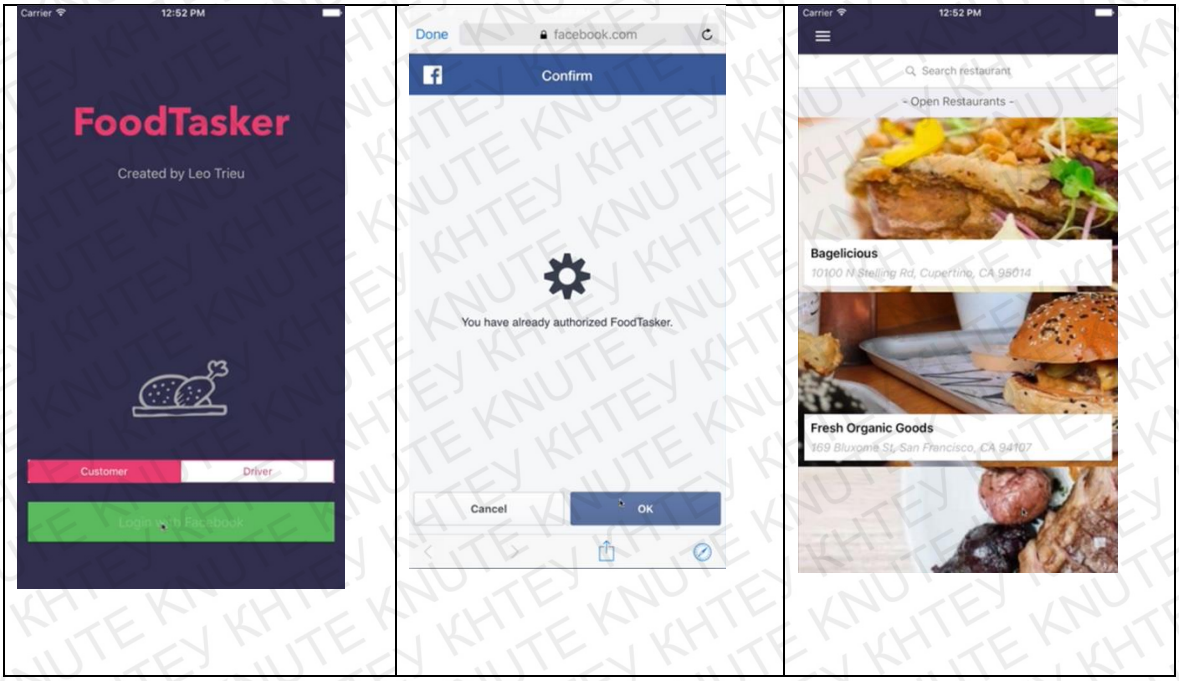
```
    override func viewDidAppear(_ animated: Bool) {
    userType = userType.capitalized
    if (AccessToken.current != nil && fbLoginSuccess == true) {
    performSegue(withIdentifier: "\(userType)View", sender: self)
    }}
    @IBAction func facebookLogout(_ sender: AnyObject) {
    APIManager.shared.logout { (error) in
    if error == nil {
    FBManager.shared.logOut()
    User.currentUser.resetInfo()
    self.bLogout.isHidden = true
    self.bLogin.setTitle("Login with Facebook", for: .normal)
    }}}
    @IBAction func facebookLogin(_ sender: AnyObject) {
    if (AccessToken.current != nil) {
    APIManager.shared.login(userType: userType, completionHandler:
{ (error) in
    if error == nil {
    self.fbLoginSuccess = true
    self.viewDidAppear(true)
    })} else {
    FBManager.shared.logIn(
    permissions: ["public_profile", "email"],
    from: self,
    handler: { (result, error) in
    if (error == nil) {
    FBManager.getFBUserData(completionHandler: {
    APIManager.shared.login(userType: self.userType, completionHandler:
{ (error) in
```

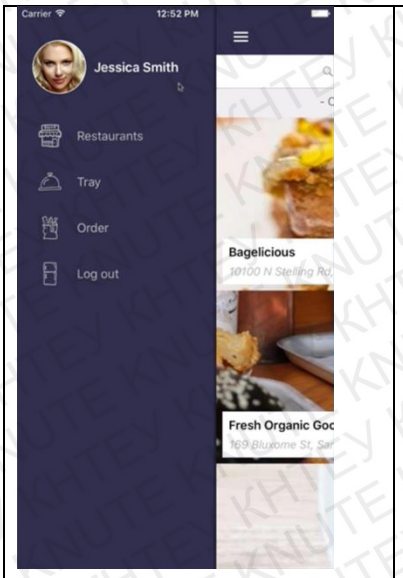
```
if error == nil {
self.fbLoginSuccess = true
self.viewDidAppear(true)
}
})})})})}
@IBAction func switchAccount(_ sender: AnyObject) {
let type = switchUser.selectedSegmentIndex
if type == 0 {
userType = USERTYPE_CUSTOMER
} else {
userType = USERTYPE_DRIVER
}}}
```

В основі додаток має реалізацію на клієнт-серверній архітектурі за допомогою локального хостинга та сайта керування та контролю доступу Herocu.com

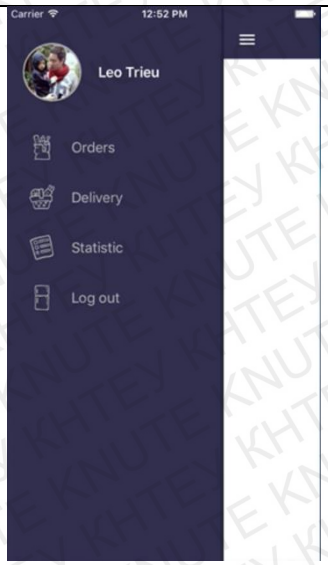


Вже по створеним макетам відкриваємо Xcode та починаємо створювать наші View.

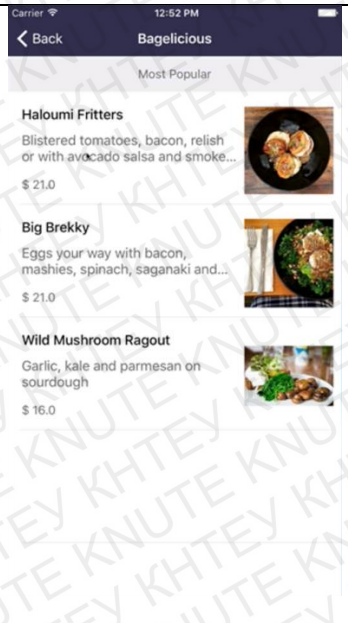



Carrier
12:52 PM
Jessica Smith
Restaurants
Tray
Order
Log out
Bagelicious
Fresh Organic Goc

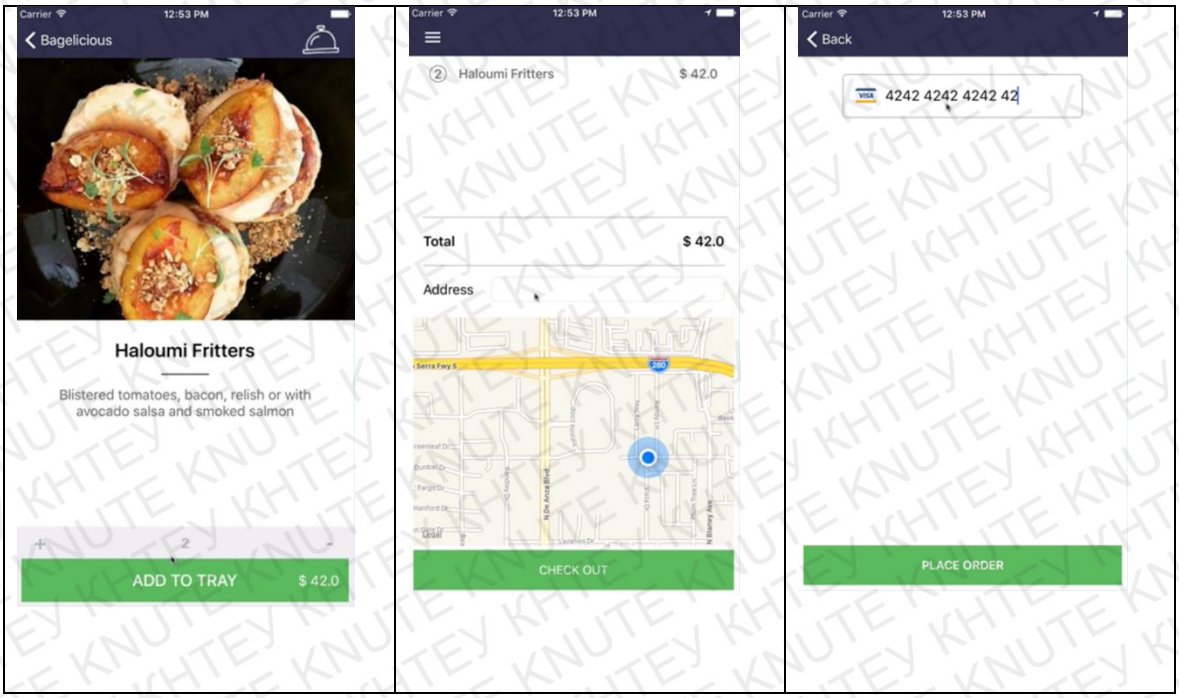

Carrier
12:52 PM
- Open Restaurants -
Bagelicious
10100 N Stelling Rd, Cupertino, CA 95014


Carrier
12:52 PM
Leo Trieu
Orders
Delivery
Statistic
Log out

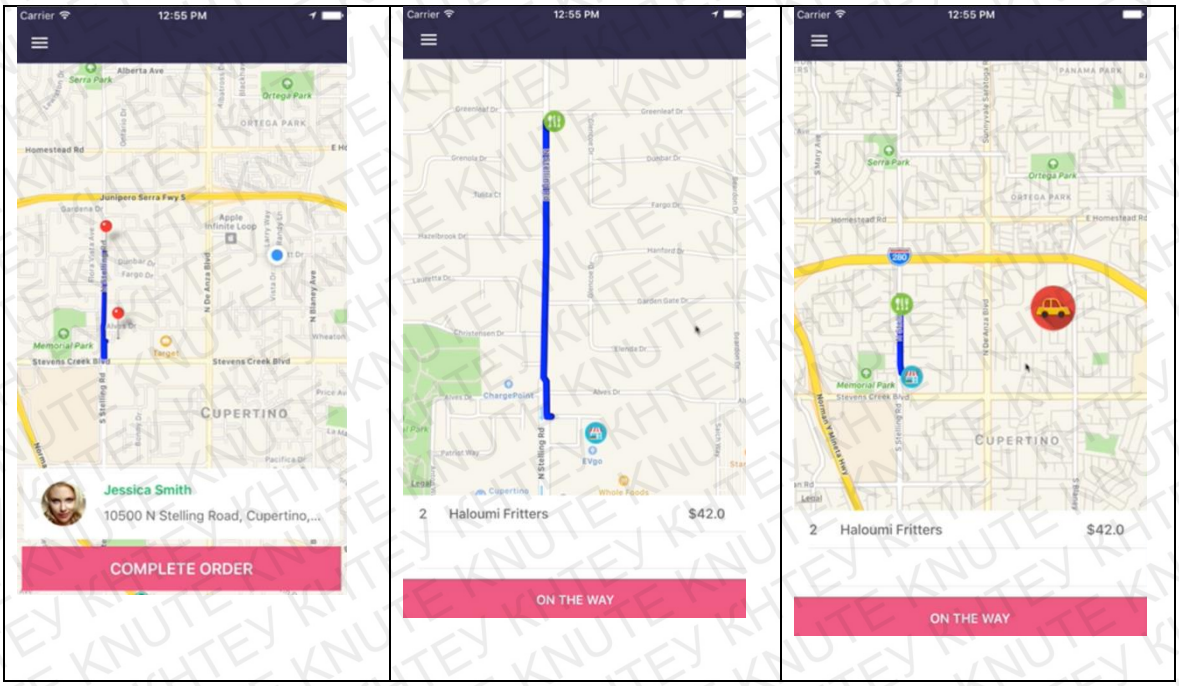

Carrier
12:52 PM
Back
Bagelicious
Most Popular
Haloumi Fritters
Blistered tomatoes, bacon, relish or with avocado salsa and smoke...
$ 21.0
Big Brekky
Eggs your way with bacon, mashies, spinach, saganaki and...
$ 21.0
Wild Mushroom Ragout
Garlic, kale and parmesan on sourdough
$ 16.0

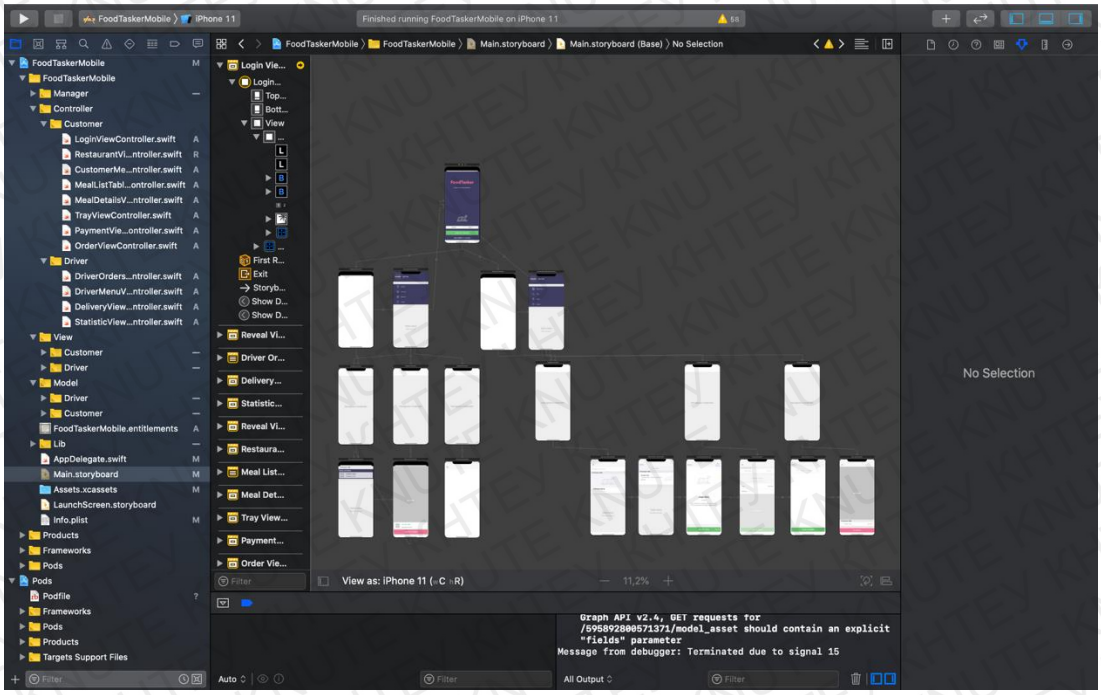

Carrier
12:52 PM
Back
Bagelicious
Most Popular
Haloumi Fritters
Blistered tomatoes, bacon, relish or with avocado salsa and smoke...
$ 21.0
Big Brekky
Eggs your way with bacon, mashies, spinach, saganaki and...
$ 21.0
Wild Mushroom Ragout
Garlic, kale and parmesan on sourdough
$ 16.0


Carrier
12:52 PM
Bagelicious
Haloumi Fritters
Blistered tomatoes, bacon, relish or with avocado salsa and smoked salmon
1
ADD TO TRAY
$ 21.0


Carrier
12:53 PM
Bagelicious
Haloumi Fritters
Blistered tomatoes, bacon, relish or with avocado salsa and smoked salmon
2
ADD TO TRAY
$ 42.0
Haloumi Fritters
$ 42.0
Total
$ 42.0
Address
CHECK OUT
Back
4242 4242 4242 42
PLACE ORDER


Hi, Lily Doan
Bagelicious
DASHBOARD
Orders
Meals
Transactions
Report
PROFILE
Account
Logout
Orders
Id
Order Details
Order Address
Customer
Driver
Total (USD)
Status
Action
45
Haloumi Fritters (21.0$) x 2 = 42.0$
10500 N Stelling Road, Cupertino, CA 95014
Jessica Smith
None
42.0
Cooking
Ready
39
Haloumi Fritters (21.0$) x 2 = 42.0$
Wild Mushroom Ragout (16.0$) x 2 = 32.0$
99 Burwood Hwy, Burwood East
Jessica Smith
Jason Gioran
74.0
Delivered
38
Wild Mushroom Ragout (16.0$) x 1 = 16.0$
Haloumi Fritters (21.0$) x 1 = 21.0$
Big Brekky (21.0$) x 2 = 42.0$
162 Barkers Road, Hawthorn
Jessica Smith
Leo Trieu
79.0
Delivered
36
Haloumi Fritters (21.0$) x 1 = 21.0$
Big Brekky (21.0$) x 3 = 63.0$
162 Barkers Road, Hawthorn
Jessica Smith
Jason Gioran
84.0
Delivered
35
Haloumi Fritters (21.0$) x 1 = 21.0$
99 Burwood Hwy, Burwood East
Jessica Smith
Jessica Smith
21.0
Delivered


Carrier
12:55 PM
Jessica Smith
10500 N Stelling Road, Cupertino,...
COMPLETE ORDER
2 Haloumi Fritters $42.0
ON THE WAY
2 Haloumi Fritters $42.0
ON THE WAY


No Selection


FoodTasker
Login with Facebook
USE A DIFFERENT ACCOUNT
Table View
Table View


Navigation Controller
Navigation Controller
Navigation Controller
Table View
COMPLETE ORDER


Table View
Navigation Controller
Navigation Controller
Navigation Controller
Burger King
ADD TO TRAY
PLACE ORDER


FoodTasker

3.4 Висновки до Розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Було розглянуто особливості архітектури платформи iOS та технологій розробки мобільних застосувань. Було наведено основні фреймворки для вирішення поставленої задачі, обгрунтовано вибір даних технологій, адже ці інструменти найкраще підходять для розробки мобільних додатків, які в свою чергу повинні мати простий, лаконічний, привабливий користувацький інтерфейс, швидку і зручну навігацію, високу швидкодію. Результати проведеного дослідженя дають підставу зробити висновки, що на теперішний час існують програми для багатьох логістичних компаній, та не всі вони мають необхідний інтерфейс для найбільш ефективної та злогодженої роботи. Вони дають можливість виконувати базовий набір функцій, та не всі з них навіть намагаються покращити роботу, багато з таких мають на меті ціль ‘Аби було що показати’ та згодом дана порожнеча заповниться хорошими програмами, що написали зацікавлені розробники. Проаналізовані всі недоліки на позитивні сторони. Таким чином можна зазначити, що розроблений додаток має великий потенціал для подальшого розвитку та вдосконалення в сфері логістики і може бути прикладом для кожного, хто в подальшому буде створювати щось подібне.

ДОДАТКИ

Додаток А

Лістинг контролеру профілю користувача

```
//
//  CustomerMenuTableViewController.swift
//  FoodTaskerMobile
//
//  Created by Kostya Berdar on 7/11/19.
//  Copyright © 2019 Kostya Berdar. All rights reserved.
//
import UIKit
class CustomerMenuTableViewController: UITableViewController {
  @IBOutlet weak var imgAvatar: UIImageView!
  @IBOutlet weak var lbName: UILabel!
override func viewDidLoad() {
    super.viewDidLoad()
lbName.text = User.currentUser.name
imgAvatar.image = try! UIImage(data: Data(contentsOf: URL(string:
User.currentUser.pictureURL!)!))
    imgAvatar.layer.cornerRadius = 70 / 2
    imgAvatar.layer.borderWidth = 1.0
    imgAvatar.layer.borderColor = UIColor.white.cgColor
    imgAvatar.clipsToBounds = true

    view.backgroundColor = UIColor(red: 0.19, green: 0.18, blue: 0.31,
alpha: 1.0)
  }
  override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
if segue.identifier == "CustomerLogout" {
APIManager.shared.logout(completionHandler: { (error) in
```

```
        if error == nil {
                        FBManager.shared.logOut()
                        User.currentUser.resetInfo()
                        // Re-render the LoginView once you completed your loggin out
process
                        let storyboard = UIStoryboard(name: "Main", bundle: nil)
                        let appController =
storyboard.instantiateViewController(withIdentifier: "MainController") as!
LoginViewController
        let appDelegate = UIApplication.shared.delegate as! AppDelegate
                        appDelegate.window!.rootViewController = appController
                    }
                })
        }
           }

        }
```

Лістинг контролера меню.

```
//
//  MealListTableViewController.swift
//  FoodTaskerMobile
//
//  Created by Kostya Berdar on 7/11/19.
//  Copyright © 2019 Kostya Berdar. All rights reserved.
//
import UIKit
class MealListTableViewController: UITableViewController {
   var restaurant: Restaurant?
   var meals = [Meal]()
   let activityIndicator = UIActivityIndicatorView()
   override func viewDidLoad() {
      super.viewDidLoad()
      if let restaurantName = restaurant?.name {
         self.navigationItem.title = restaurantName   }
loadMeals()
   }
func loadMeals() {
Helpers.showActivityIndicator(activityIndicator, view)
      if let restaurantId = restaurant?.id
         APIManager.shared.getMeals(restaurantId: restaurantId,
completionHandler: { (json) in
            if json != nil {
               self.meals = []
if let tempMeals = json["meals"].array {
                  for item in tempMeals {
                     let meal = Meal(json: item)
```

```
                    self.meals.append(meal)   }
                self.tableView.reloadData()
Helpers.hideActivityIndicator(self.activityIndicator)
                }   }  })  }    }
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "MealDetails" {
        let controller = segue.destination as! MealDetailsViewController
        controller.meal = meals[(tableView.indexPathForSelectedRow?.row)!]
        controller.restaurant = restaurant        }    }
    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1    }
    override func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
        return meals.count    }
    override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "MealCell", for:
indexPath) as! MealViewCell
        let meal = meals[indexPath.row]
        cell.lbMealName.text = meal.name
        cell.lbMealShortDescription.text = meal.short_description

        if let price = meal.price {
            cell.lbMealPrice.text = "$\(price)     }
        if let image = meal.image {
            Helpers.loadImage(cell.imgMealImage, "\(image)")    }
    return cell    }

    }
```

```
        Контролер віду ресторану
//
        //  RestaurantViewController.swift
        //  FoodTaskerMobile
        //
        //  Created by Kostya Berdar on 7/11/19.
        //  Copyright © 2019 Kostya Berdar. All rights reserved.
        //
        import UIKit
        class RestaurantViewController: UIViewController {
            @IBOutlet weak var menuBarButton: UIBarButtonItem!
            @IBOutlet weak var searchRestaurant: UISearchBar!
            @IBOutlet weak var tbvRestaurant: UITableView!
            var restaurants = [Restaurant]()
            var filteredRestaurants = [Restaurant]()
            let activityIndicator = UIActivityIndicatorView()
            override func viewDidLoad() {
                super.viewDidLoad()
                if self.revealViewController() != nil {
                    menuBarButton.target = self.revealViewController()
                    menuBarButton.action =
#selector(SWRevealViewController.revealToggle(_:))
        self.view.addGestureRecognizer(self.revealViewController().panGestureRecog
nizer())
                }
        loadRestaurants()
            }
        func loadRestaurants() {
        Helpers.showActivityIndicator(activityIndicator, view)
```

```
        APIManager.shared.getRestaurants { (json) in
        if json != nil {
        self.restaurants = []
        if let listRes = json["restaurants"].array {
                    for item in listRes {
                        let restaurant = Restaurant(json: item)
                        self.restaurants.append(restaurant)
                    }
        self.tbvRestaurant.reloadData()
                    Helpers.hideActivityIndicator(self.activityIndicator)
                }
            }
        }
    }
        override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "MealList" {
        let controller = segue.destination as! MealListTableViewController
            controller.restaurant =
restaurants[(tbvRestaurant.indexPathForSelectedRow?.row)!]
        }
    }
    }
        extension RestaurantViewController: UISearchBarDelegate {
        func searchBar(_ searchBar: UISearchBar, textDidChange searchText: String)
{
        filteredRestaurants = self.restaurants.filter({ (res: Restaurant) -> Bool in
        return res.name?.lowercased().range(of: searchText.lowercased()) != nil
        })
        self.tbvRestaurant.reloadData()
    }
```

```
    }

    extension RestaurantViewController: UITableViewDelegate,
UITableViewDataSource {
    func numberOfSections(in tableView: UITableView) -> Int {
        return 1
      }
    func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
    if searchRestaurant.text != "" {
        return self.filteredRestaurants.count
        }
    return self.restaurants.count
      }
    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
"RestaurantCell", for: indexPath) as! RestaurantViewCell
    let restaurant: Restaurant
    if searchRestaurant.text != "" {
        restaurant = filteredRestaurants[indexPath.row]
        } else {
        restaurant = restaurants[indexPath.row]
        }
        cell.lbRestaurantName.text = restaurant.name!
        cell.lbRestaurantAddress.text = restaurant.address!
    if let logoName = restaurant.logo {
        Helpers.loadImage(cell.imgRestaurantLogo, "\(logoName)")
        }
                                return cell    }}
```

Лістинг контролера деталей замовлення

```
//
//  MealDetailsViewController.swift
//  FoodTaskerMobile
//
//  Created by Kostya Berdar on 7/11/19.
//  Copyright © 2019 Kostya Berdar. All rights reserved.
//
import UIKit
class MealDetailsViewController: UIViewController {
   @IBOutlet weak var imgMeal: UIImageView!
   @IBOutlet weak var lbMealName: UILabel!
   @IBOutlet weak var lbMealShortDescription: UILabel!
@IBOutlet weak var lbTotal: UILabel!
   @IBOutlet weak var lbQty: UILabel!
var meal: Meal?
   var restaurant: Restaurant?
   var qty = 1
override func viewDidLoad() {
      super.viewDidLoad()
      loadMeal()
   }
func loadMeal() {
if let price = meal?.price {
         lbTotal.text = "$\(price)"
      }
lbMealName.text = meal?.name
      lbMealShortDescription.text = meal?.short_description
      if let imageUrl = meal?.image {
         Helpers.loadImage(imgMeal, "\(imageUrl)")
```

```
        }
    }
    @IBAction func addToTray(_ sender: AnyObject) {
    let image = UIImageView(frame: CGRect(x: 0, y: 0, width: 60, height: 40))
        image.image = UIImage(named: "button_chicken")
        image.center = CGPoint(x: self.view.frame.width/2, y:
self.view.frame.height-100)
        self.view.addSubview(image)
    UIView.animate(withDuration: 1.0,
                   delay: 0.0,
                   options: UIView.AnimationOptions.curveEaseOut,
                   animations: { image.center = CGPoint(x: self.view.frame.width -
40, y: 24) },
                   completion: { _ in
    image.removeFromSuperview()
            let trayItem = TrayItem(meal: self.meal!, qty: self.qty)
            guard let trayRestaurant = Tray.currentTray.restaurant, let
currentRestaurant = self.restaurant else {
                // If those requirements are not met
    Tray.currentTray.restaurant = self.restaurant
                Tray.currentTray.items.append(trayItem)
                return
            }
            let cancelAction = UIAlertAction(title: "Cancel", style: .cancel)
    // If ordering meal from the same restaurant
            if trayRestaurant.id == currentRestaurant.id {
    let inTray = Tray.currentTray.items.firstIndex(where: { (item) -> Bool in
    return item.meal.id! == trayItem.meal.id!
                })
```

```
                if let index = inTray {
        let alertView = UIAlertController(
                    title: "Add more?",
                    message: "Your tray already has this meal. Do you want to add
more?",
                    preferredStyle: .alert)
        let okAction = UIAlertAction(title: "Add more", style: .default, handler:
{ (action: UIAlertAction!) in
        Tray.currentTray.items[index].qty += self.qty
                })
        alertView.addAction(okAction)
                alertView.addAction(cancelAction)
                self.present(alertView, animated: true, completion: nil)
            } else {
                Tray.currentTray.items.append(trayItem)
            }
        }
            else {// If ordering meal from the another restaurant
        let alertView = UIAlertController(
                title: "Start new tray?",
                message: "You're ordering meal from another restaurant. Would
you like to clear the current tray?",
                preferredStyle: .alert)
        let okAction = UIAlertAction(title: "New Tray", style: .default, handler:
{ (action: UIAlertAction!) in
        Tray.currentTray.items = []
                Tray.currentTray.items.append(trayItem)
                Tray.currentTray.restaurant = self.restaurant
            })
```

```
            alertView.addAction(okAction)
            alertView.addAction(cancelAction)
self.present(alertView, animated: true, completion: nil)
        }
    })
  }
@IBAction func removeQty(_ sender: AnyObject) {
if qty >= 2 {
        qty -= 1
        lbQty.text = String(qty)
if let price = meal?.price {
            lbTotal.text = "$\(price * Float(qty))"
        }
    }
  }
@IBAction func addQty(_ sender: AnyObject) {
if qty < 99 {
        qty += 1
        lbQty.text = String(qty)
if let price = meal?.price {
            lbTotal.text = "$\(price * Float(qty))"
        }
    }
  }
}
```

Лістинг контролера подання замовлення

```
//
//  TrayViewController.swift
//  FoodTaskerMobile
//
```

```
//  Created by Kostya Berdar on 7/11/19.
//  Copyright © 2019 Kostya Berdar. All rights reserved.
/
import UIKit
import MapKit
import CoreLocation
class TrayViewController: UIViewController {
    @IBOutlet weak var menuBarButton: UIBarButtonItem!
    @IBOutlet weak var tbvMeals: UITableView!
    @IBOutlet weak var viewTotal: UIView!
    @IBOutlet weak var viewAddress: UIView!
    @IBOutlet weak var viewMap: UIView!
    @IBOutlet weak var lbTotal: UILabel!
    @IBOutlet weak var tbAddress: UITextField!
    @IBOutlet weak var map: MKMapView!
    @IBOutlet weak var bAddPayment: UIButton!
    var locationManager: CLLocationManager!
    override func viewDidLoad() {
        super.viewDidLoad()
    if self.revealViewController() != nil {
            menuBarButton.target = self.revealViewController()
            menuBarButton.action =
#selector(SWRevealViewController.revealToggle(_:))

self.view.addGestureRecognizer(self.revealViewController().panGestureRecognizer(
))
        }
    if Tray.currentTray.items.count == 0 {
            // Showing a message here
```

```
            let lbEmptyTray = UILabel(frame: CGRect(x: 0, y: 0, width:
self.view.frame.size.width, height: 40))
            lbEmptyTray.center = self.view.center
            lbEmptyTray.textAlignment = NSTextAlignment.center
            lbEmptyTray.text = "Your tray is empty. Please select meal."
        self.view.addSubview(lbEmptyTray)
        } else {
            // Display all of the UI controllers
        self.tbvMeals.isHidden = false
            self.viewTotal.isHidden = false
            self.viewAddress.isHidden = false
            self.viewMap.isHidden = false
            self.bAddPayment.isHidden = false
        loadMeals()
        }
        // Show current user's location
        if CLLocationManager.locationServicesEnabled() {
        locationManager = CLLocationManager()
            locationManager.delegate = self
            locationManager.desiredAccuracy = kCLLocationAccuracyBest
            locationManager.requestAlwaysAuthorization()
            locationManager.startUpdatingLocation()
        self.map.showsUserLocation = true
        }
    }
    func loadMeals() {
        self.tbvMeals.reloadData()
        self.lbTotal.text = "$\(Tray.currentTray.getTotal())"
    }
    @IBAction func addPayment(_ sender: AnyObject) {
```

```
            if tbAddress.text == "" {
                // Showing alert that this field is required.
        let alertController = UIAlertController(title: "No Address", message: "Address
is required", preferredStyle: .alert)
        let okAction = UIAlertAction(title: "OK", style: .default, handler: { (alert) in
                    self.tbAddress.becomeFirstResponder()
                })
        alertController.addAction(okAction)
                self.present(alertController, animated: true, completion: nil)
        } else {
        Tray.currentTray.address = tbAddress.text
                self.performSegue(withIdentifier: "AddPayment", sender: self)
            }
        }
        }
        extension TrayViewController: CLLocationManagerDelegate {
        func locationManager(_ manager: CLLocationManager, didUpdateLocations
locations: [CLLocation]) {
        let location = locations.last! as CLLocation
        let center = CLLocationCoordinate2D(
                latitude: location.coordinate.latitude,
                longitude: location.coordinate.longitude)
        let region = MKCoordinateRegion(center: center, span:
MKCoordinateSpan(latitudeDelta: 0.01, longitudeDelta: 0.01))
        self.map.setRegion(region, animated: true)
        }
        }
        extension TrayViewController: UITableViewDataSource,
UITableViewDelegate {
```

```
    func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }
    func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
        return Tray.currentTray.items.count
    }
    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "TrayItemCell",
for: indexPath) as! TrayViewCell
    let tray = Tray.currentTray.items[indexPath.row]
        cell.lbQty.text = "\(tray.qty)"
        cell.lbMealName.text = tray.meal.name
        cell.lbSubTotal.text = "$\(tray.meal.price! * Float(tray.qty))"
    return cell
    }
    }
    extension TrayViewController: UITextFieldDelegate {
    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
    let address = textField.text
        let geocoder = CLGeocoder()
        Tray.currentTray.address = address
    geocoder.geocodeAddressString(address!) { (placemarks, error) in
    if (error != nil) {
            print("Error: ", error)
        }
    if let placemark = placemarks?.first {
```

```
                let coordinates: CLLocationCoordinate2D =
placemark.location!.coordinate
                let region = MKCoordinateRegion(
                  center: coordinates,
                  span: MKCoordinateSpan(latitudeDelta: 0.01, longitudeDelta:
0.01)
                )
    self.map.setRegion(region, animated: true)
                self.locationManager.stopUpdatingLocation()
    // Create a pin
                let dropPin = MKPointAnnotation()
                dropPin.coordinate = coordinates
    self.map.addAnnotation(dropPin)
            }
        }
    return true
    }
    }
```

Лістинг контролера замовлень

```
//  OrderViewController.swift
//  FoodTaskerMobile
//
//  Created by Kostya Berdar on 7/11/19.
//  Copyright © 2019 Kostya Berdar. All rights reserved.
//
import UIKit
import SwiftyJSON
import MapKit
class OrderViewController: UIViewController {
   @IBOutlet weak var menuBarButton: UIBarButtonItem!
   @IBOutlet weak var tbvMeals: UITableView!
   @IBOutlet weak var map: MKMapView!
   @IBOutlet weak var lbStatus: UILabel!
var tray = [JSON]()
var destination: MKPlacemark?
   var source: MKPlacemark?
var driverPin: MKPointAnnotation!
   var timer = Timer()
override func viewDidLoad() {
      super.viewDidLoad()
if self.revealViewController() != nil {
         menuBarButton.target = self.revealViewController()
         menuBarButton.action =
#selector(SWRevealViewController.revealToggle(_:))
self.view.addGestureRecognizer(self.revealViewController().panGestureRecognizer(
))
      }
```

```
        getLatestOrder()
    }
    func getLatestOrder() {
    APIManager.shared.getLatestOrder { (json) in
    print(json)
    let order = json["order"]
    if order["status"] != nil {
    if let orderDetails = order["order_details"].array {
    self.lbStatus.text = order["status"].string!.uppercased()
                    self.tray = orderDetails
                    self.tbvMeals.reloadData()
                }
    let from = order["restaurant"]["address"].string!
                let to = order["address"].string!
    self.getLocation(from, "RES", { (sou) in
                    self.source = sou
    self.getLocation(to, "CUS", { (des) in
                        self.destination = des
                        self.getDirections()
                    })
                })
    if order["status"] != "Delivered" {
                    self.setTimer()
                }
            }

        }
    }
    func setTimer() {
        timer = Timer.scheduledTimer(
```

```
        timeInterval: 1,
        target: self,
        selector: #selector(getDriverLocation(_:)),
        userInfo: nil, repeats: true)
    }

    @objc func getDriverLocation(_ sender: AnyObject) {
      APIManager.shared.getDriverLocation { (json) in

        if let location = json["location"].string {

          self.lbStatus.text = "ON THE WAY"

          let split = location.components(separatedBy: ",")
          let lat = split[0]
          let lng = split[1]

          let coordinate = CLLocationCoordinate2D(latitude:
CLLocationDegrees(lat)!, longitude: CLLocationDegrees(lng)!)

          // Create pin annotation for Driver
          if self.driverPin != nil {
            self.driverPin.coordinate = coordinate
          } else {
            self.driverPin = MKPointAnnotation()
            self.driverPin.coordinate = coordinate
            self.driverPin.title = "DRI"
            self.map.addAnnotation(self.driverPin)
          }
```

```
                // Reset zoom rect to cover 3 locations
                self.autoZoom()
        } else {
                self.timer.invalidate()
            }
        }
    }
    func autoZoom() {
    var zoomRect = MKMapRect.null
        for annotation in self.map.annotations {
          let annotationPoint = MKMapPoint(annotation.coordinate)
          let pointRect = MKMapRect(x: annotationPoint.x, y: annotationPoint.y,
width: 0.1, height: 0.1)
          zoomRect = zoomRect.union(pointRect)
        }
    let insetWidth = -zoomRect.size.width * 0.2
        let insetHeight = -zoomRect.size.height * 0.2
        let insetRect = zoomRect.insetBy(dx: insetWidth, dy: insetHeight)
    self.map.setVisibleMapRect(insetRect, animated: true)
    }}
    extension OrderViewController: MKMapViewDelegate {
    // #1 - Delegate method of MKMapViewDelegate
    func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) -
> MKOverlayRenderer {
        let renderer = MKPolylineRenderer(overlay: overlay)
        renderer.strokeColor = UIColor.blue
        renderer.lineWidth = 5.0
    return renderer
    }
```

```
    // #2 - Convert an address string to a location on the map
    func getLocation(_ address: String,_ title: String,_ completionHandler:
@escaping (MKPlacemark) -> Void) {
    let geocoder = CLGeocoder()
    geocoder.geocodeAddressString(address) { (placemarks, error) in
    if (error != nil) {
            print("Error: ", error as Any)
        }
    if let placemark = placemarks?.first {
    let coordinates: CLLocationCoordinate2D = placemark.location!.coordinate
    // Create a pin
            let dropPin = MKPointAnnotation()
            dropPin.coordinate = coordinates
            dropPin.title = title
    self.map.addAnnotation(dropPin)
            completionHandler(MKPlacemark.init(placemark: placemark))
        }    }    }
    // #3 - Get direction and zoom to address
    func getDirections() {
    let request = MKDirections.Request()
        request.source = MKMapItem.init(placemark: source!)
        request.destination = MKMapItem.init(placemark: destination!)
        request.requestsAlternateRoutes = false
    let directions = MKDirections(request: request)
        directions.calculate { (response, error) in

          if error != nil {
            print("Error: ", error)
          } else {
            // Show route
```

```
                self.showRoute(response: response!)
            } }}
        // #4 - Show route between locations and make a visible zoom
          func showRoute(response: MKDirections.Response) {
        for route in response.routes {
            self.map.addOverlay(route.polyline, level:
MKOverlayLevel.aboveRoads)
        }   }
        // #5 - Customise pin point with Image
          func mapView(_ mapView: MKMapView, viewFor annotation:
MKAnnotation) -> MKAnnotationView? {
        let annotationIdentifier = "MyPin"
        var annotationView: MKAnnotationView?
            if let dequeueAnnotationView =
mapView.dequeueReusableAnnotationView(withIdentifier: annotationIdentifier) {
        annotationView = dequeueAnnotationView
            annotationView?.annotation = annotation
        } else {
        annotationView = MKAnnotationView(annotation: annotation, reuseIdentifier:
annotationIdentifier)
        }
        if let annotationView = annotationView, let name = annotation.title! {
            switch name {
            case "DRI":
              annotationView.canShowCallout = true
              annotationView.image = UIImage(named: "pin_car")
            case "RES":
              annotationView.canShowCallout = true
              annotationView.image = UIImage(named: "pin_restaurant")
            case "CUS":
```

```
                annotationView.canShowCallout = true
                annotationView.image = UIImage(named: "pin_customer")
            default:
                annotationView.canShowCallout = true
                annotationView.image = UIImage(named: "pin_car")
            }    }
        return annotationView
    }}
    extension OrderViewController: UITableViewDelegate,
UITableViewDataSource {
    func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }
    func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
        return tray.count
    }
    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
"OrderItemCell", for: indexPath) as! OrderViewCell
    let item = tray[indexPath.row]
        cell.lbQty.text = String(item["quantity"].int!)
        cell.lbMealName.text = item["meal"]["name"].string
        cell.lbSubTotal.text = "$\(String(item["sub_total"].float!))"
    return cell
    }
    }
```

Лістинг Авторизації

```
//
//  LoginViewController.swift
//  FoodTaskerMobile
//
//  Created by Kostya Berdar on 7/11/19.
//  Copyright © 2019 Kostya Berdar. All rights reserved.
//
import UIKit
import FBSDKLoginKit
class LoginViewController: UIViewController {
   @IBOutlet weak var bLogin: UIButton!
   @IBOutlet weak var bLogout: UIButton!
   @IBOutlet weak var switchUser: UISegmentedControl!
var fbLoginSuccess = false
   var userType: String = USERTYPE_CUSTOMER
override func viewDidLoad() {
      super.viewDidLoad()
      if (AccessToken.current != nil) {
bLogout.isHidden = false
         FBManager.getFBUserData(completionHandler: {
self.bLogin.setTitle("Continue as \(User.currentUser.email!)", for: .normal)
            // self.bLogin.sizeToFit()
         })
      }
   }

   override func viewDidAppear(_ animated: Bool) {
      userType = userType.capitalized

      if (AccessToken.current != nil && fbLoginSuccess == true) {
```

```
            performSegue(withIdentifier: "\(userType)View", sender: self)
        }
    }
@IBAction func facebookLogout(_ sender: AnyObject) {
APIManager.shared.logout { (error) in
            if error == nil {
                FBManager.shared.logOut()
                User.currentUser.resetInfo()
self.bLogout.isHidden = true
                self.bLogin.setTitle("Login with Facebook", for: .normal)
            }
        }
    }
    @IBAction func facebookLogin(_ sender: AnyObject) {
if (AccessToken.current != nil) {
APIManager.shared.login(userType: userType, completionHandler: { (error) in
            if error == nil {
                self.fbLoginSuccess = true
                self.viewDidAppear(true)
            }
        })

    } else {
FBManager.shared.logIn(
            permissions: ["public_profile", "email"],
            from: self,
            handler: { (result, error) in
if (error == nil) {
FBManager.getFBUserData(completionHandler: {
                    APIManager.shared.login(userType: self.userType,
```

```
completionHandler: { (error) in
                        if error == nil {
                            self.fbLoginSuccess = true
                            self.viewDidAppear(true)
                        }
                    })
                })
            }
        })
    }
}
@IBAction func switchAccount(_ sender: AnyObject) {
    let type = switchUser.selectedSegmentIndex

    if type == 0 {
        userType = USERTYPE_CUSTOMER
    } else {
        userType = USERTYPE_DRIVER
    }
}

}
```

Додаток Б

Програма та методика тестування.

Стек включає в себе чотири компоненти з відкритим вихідним кодом: React, TestApi, React і PythonTest. Серед цих технологій Xcdoe – включає бази даних, Swift - внутрішнє середовище виконання, PythonTest - внутрішня веб-інфраструктура, Swift – інтерфейсне середовище та Xcode – інструмент управління станом.

Клієнтська та серверна частина інформаційної системи розроблена різними мовами програмування. Серверна частина інформаційної системи розроблена мовою програмування XCode за допомогою фреймворку Testing5. Основний функціонал серверної частини:

- Створення серверу;
- Створення колекцій даних;
- Маршрутизація даних.

Для тестування серверної частини доцільно використовувати метод тестування модульне тестування. Модульне тестування, або юніт-тестування – це процес в програмуванні, що дозволяє перевірити на коректність окремі модулі вихідного коду програми. Метод полягає в тому, щоб створювати тести для кожної нетривіальної функції або методу. Це дозволяє досить швидко перевірити, чи не призвело чергову зміну коду до регресії, тобто до появи помилок у раніше тестованих місцях програми, а також полегшує виявлення і усунення таких помилок.

Засоби тестування серверної частини інформаційної системи.

npm install -g mocha

npm install –save-dev chai

npm install –save-dev chai-http

Після цього з’явиться відповідні пакети в клієнт-серверній інформаційній системі.

"scripts": {

```
    "test": "mocha --require babel-register tests/*.js --exit",
  "dev": "nodemon --exec babel-node --presets babel-preset-env ./server.js"
    "dependencies": {
      "bcryptjs": "^2.4.3",
      "body-parser": "^1.19.0",
      "classnames": "^2.2.6",
      "config": "^3.1.0",
      "express": "^4.17.1",
      "express-validator": "^5.3.1",
      "gravatar": "^1.8.0",
      "jsonwebtoken": "^8.5.1",
      "mongoose": "^5.5.12",
      "passport": "^0.4.0",
      "passport-jwt": "^4.0.0",
      "request": "^2.88.0",
      "validator": "^11.1.0"
    },
    "devDependencies": {
    "babel-cli": "^6.26.0",
    "babel-preset-env": "^1.7.0",
    "chai": "^4.1.2",
    "chai-http": "^4.0.0",
    "mocha": "^5.1.1",
    "nodemon": "^1.17.4"
  }
```

Після встановлення пакетів, створюється файл test.js в якому записується код для тестування серверної частини.

Лістинг тестування профілю користувача.

```
// Import the dependencies for testing
import chai from 'chai';
```

```
import chaiHttp from 'chai-http';
import app from './server';// Configure chai
chai.use(chaiHttp);
chai.should();describe("profiles", () => {
  describe("GET /api/profile/all", () => {
    // Test to get all profiles
    it("should get all profiles", (done) => {
      chai.request(app)
        .get('/api/profile/all')
        .end((err, res) => {
          res.should.have.status(200);
          res.body.should.be.a('object');
          done();
        });
     });// Test to get profile by user ID
    it("should get profile by user ID", (done) => {
      const id = USER_ID;
      chai.request(app)
        .get(`/api/profile/user/:user_id`)
        .end((err, res) => {
          res.should.have.status(200);
          res.body.should.be.a('object');
          done();
        });  });
    // Test to PUT Create NEW PROFILE
    it("should put a create new profile ", (done) => {
      const personalTable = {
                halfYear: '1'
        trainningForm: '2'
        faculty: 'OAIS'
```

```
                disciplinesName: 'Osnovi programuvannya'
                term: '2'
                course: '1'
                groupNumber: '3'
                secondTeacher: 'Ivanov'
                lectionsNumb: '65'
                labsNumb: '12'
                consultaionsNumb: '2'
                practicalNumb'10'
                ModularContNumb: '2'
                ExamsNumb: '2'
              }
            chai.request(app)
              .put(`/api/profile/experience`)
              .end((err, res) => {
                res.should.have.status(200);
                res.body.should.be.a('object');
                done();
              }); });});});
      // Test to failed DELETE user
it("should not delete a user", (done) => {
const id = null;
chai.request(app)
.delete(`/api/profile`)
.end((err, res) => {
res.should.have.status(404);
done();
});
});
```

Отже, після вводу коду для тестування інформаційної системи потрібно запустити тестування ввівши в термінал команду mocha.

Результат тестування серверної частини:

GET /api/profile/all 200 5ms - 103b

GET /api/profile/user/:user_id 200 9ms - 150b

PUT /api/profile/experience 200 200 23ms 196b

DELETE /api/profile 404 3ms 100b

Тестування клієнтської частини інформаційної системи.

Клієнтська частина інформаційної системи розроблена мовою програмування JavaScript за допомогою бібліотеки React.js.

Основний функціонал клієнтської частини:

- Створення серверу;
- Створення колекцій даних;
- Маршрутизація даних.

Для тестування серверної частини доцільно використовувати метод тестування функціонального тестування на базі компонентів.

Засоби тестування серверної частини інформаційної системи.

Sinon - це тестова бібліотека, яка дає нам змогу писати шпигунів, заглушок та глузувань.

заснована на DOM Testing Library, додаючи API для роботи з компонентами React.

react-testing-library

Jest - це бібліотека JavaScript для створення, запуску та структурування тестів. Jest поширюється у вигляді пакету NPM.

Enzyme - бібліотека тестування JavaScript, побудована та підтримується Airbnb.

Встановлення засобів тестування клієнтської частини.

Для того щоб встановити засоби тестування потрібно, щоб на комп’ютері був встановлений Xcode Після встановлення Xcode потрібно, використовуючи

менеджер пакетів npm, встановити засоби тестування прописавши відповідну команду:

npm install –save-dev babel-jest babel-preset-stage-0 enzyme jest-cli react-addons-test-utils react-test-renderer redux-mock-store sinon

Після цього з’явиться відповідні пакети в клієнт-серверній інформаційній системі.

Після встановлення пакетів, створюється файл TextInput.test.js в якому записується код для тестування серверної частини.

Лістинг тестування профілю користувача.

```
import React from 'react';
import TestUtils from 'react-addons-test-utils';
import TextInput from '../ TextInput;
describe(TextInput, () => {
 it('wraps content in a div with .notificationsFrame class', () => {
  const wrapper = TestUtils.renderIntoDocument(<TextInput />);
  const node = TestUtils .findRenderedDOMComponentWithClass(wrapper, 'notificationsFrame');
 });
})
```

Лістинг тестування навігації користувача.

```
import Foundation
import SwiftyJSON

class User {

  var name: String?
  var email: String?
  var pictureURL: String?

  static let currentUser = User()
```

```
    func setInfo(json: JSON) {
        self.name = json["name"].string
        self.email = json["email"].string

        let image = json["picture"].dictionary
        let imageData = image?["data"]?.dictionary
        self.pictureURL = imageData?["url"]?.string
    }

    func resetInfo() {
        self.name = nil
        self.email = nil
        self.pictureURL = nil
    }

}
```

Результат тестування клієнтської частини:

Test Suites: 4 passed, 4 total

Test: 3 passed, 3 total

Time: 1.299s, estimated 2s