МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Навчально-науковий інститут Інноваційно-освітніх технологій

Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

Савченко А.С.

"___" _____2020 р.

ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"МАГІСТРА"

ЗА СПЕЦІАЛІЗАЦІЄЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ (ЗА ГАЛУЗЯМИ)"

Тема: "iOS додаток для сервісу продажу автомобіля"

Виконавець: студент групи УС-201Мз Хамбір Владислав Русланович .

Керівник: к.т.н., доцент Райчев Ігор Едуардович .

Нормоконтролер: _____ Райчев І.Е.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Навчально-науковий інститут Іноваційно-освітніх технологій

Кафедра Комп'ютерних інформаційних технологій

Галузь знань, спеціальність, спеціалізація: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології (за галузями)"
(шифр, найменування)

ЗАТВЕРДЖУЮ

Завідувач кафедри

Савченко А.С.

" 20 " 11 2018 р.

ЗАВДАННЯ

на виконання дипломної роботи студента

Хамбіра Владислава Руслановича
(прізвище, ім'я, по батькові)

1.Тема роботи: "iOS додаток для сервісу продажу автомобіля".

Затверджена наказом ректора від "20" 11 2018 р. за № 2864/ст.

2. Термін виконання роботи: з 20.11.2018р. до 21 лютого 2019р.

3. Вихідні данні до роботи: мова програмування Swift. Електронний доступ. URL: https://ru.wikipedia.org/wiki/Swift_(язык_программирования), IDE Xcode. Електронний доступ. URL: https://uk.wikipedia.org/wiki/Xcode, Засіб управління щалежностями Cocoa-бібліотек CocoaPods. Електронний доступ. URL: https://habr.com/en/company/luxoft/blog/149631/, Прийоми об'єктно-орієнтованного проєктування. / Гамма Э. Хелм Р. Джонсон Р. Вліссідес Дж. [та ін.] ; Серія «Бібліотека програміста», СПб: Пітер, 2001. — 368 с. – Бібліогр.: с. 280–291.

4. Зміст пояснювальної записки: Огляд існуючих сервісів, засоби розробки, проєктування, реалізація, огляд функціоналу, тестування та публікація продукту.

5. Перелік обов'язкового ілюстративного матеріалу: 1) Децентралізовані системи контролю версій; 2) Схема роботи Git; 3) Двух-ланкова клієнт-серверна архітектура; 4) Діаграма взаємодії між собою компонентів шаблону MVC та MVVM; 5) Схема взаємодії між собою компонентів у шаблоні Observable; 6) Демонстрація залежності між класами; 7) Схема роботи Push-remote нотіфікацій; 8) Огляд екранів додатку.

6. Календарний план-графік

№ з/п	*Завдання*	*Термін виконання*	*Підпис керівника*
1	Огляд існуючих сервісів по продажу автомобілів	01.10.19 – 03.11.19	
2	Вибір та опис засобів розробки, IDE та мови програмування.	04.11. 19– 06.11.19	
3	Опис ключових моментів проєктування. Розгляд архітектури, та реактивного програмування.	07.11.19– 10.11.19	
4	Реалізція додатку та побудова основних сервісів для взаємодії з API та Push-повідомленнями.	11.11.19– 25.12.19	
5	Опис функціоналу додатка з його основними можливостями та ілюстрації роботи.	26.12.19– 31.12.19	
6	Ручне тестування додатку	03.01.20 – 09.01.20	
7	Публікування додатку у магазин додатків App Store.	10.01.20 – 26.01.20	
8	Створення доповіді та слайдів до неї	01.02.20 - 03.02.20	
9	Оформлення та друк пояснювальної записки дипломної роботи	02.02.19 – 02.02.19	

Студент-дипломник Хамбір Владислав Русланович .

Керівник дипломної роботи Райчев Ігор Едуардович .

7. Консультація з окремого(мих) розділу(ів) роботи:

Назва розділу	Консультант (посада, П.І.Б.)	Дата, підпис	
		Завдання видав	Завдання прийняв

8. Дата видачі завдання _____

Керівник дипломної роботи _____ Райчев І.Е..
(підпис)

Завдання прийняв до виконання _____
(підпис випускника) (ПІБ)

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту роботи «iOS додаток для сервісу продажу автомобіля», викладена на 79 сторінках, 49 рисунків, та 13 джерел. Основний текст роботи викладено на 10-77 сторінках.

Ключові слова: iOS, Xcode, RxSwift, Swift, Git, MVVM, Push-повідомлення.

Об'єкт дослідження – програмне забезпечення для продажу автомобіля.

Предмет дослідження – технології для реалізації iOS додатку.

Мета роботи – розробка мобільного iOS додатку з необхідним функціоналом. Публікація додатку у магазині додатків App Store.

Результат магістерскої роботи – реалізовано iOS додаток для сервісу по продажу автомобіля, проведено ручне тестування у результаті якого були виправлені усі помилки. Додаток було опубліковано у магазині додатків App Store.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API - Application Programming Interface

IDE - Integrated development environment

HTTP - HyperText Transfer Protocol

Rx - Reactive Extension

UI - User Interface

ПОП - Протоколо-орієнтоване програмування

Git - Global Information Tracker

ВСТУП

Ми живемо в ері інтернет-сервісів. Досить багато чого раніше було доступно користувачам у режимі “Офлайн”, тобто сервіси існували і раніше, але зараз вони активно перебираються у вимір інтернету.

Користуватися сервісами, використовуючи інтернет-сайти, мобільні додатки досить зручно. Це дозволяє нам не витрачати зайвий час, швидше отримувати потрібний нам результат. Користувачі замовляють їжу, оформлюють документи, записуються до лікаря, знімають автомобіль чи викликають таксі онлайн і це далеко не повний перелік усіх можливих сценаріїв інтернет-сервісів.

Існують країни, в яких подібна практика тільки набуває оборотів. Наприклад, Кувейт. У цій країні активно з’являються онлайн-сервіси на кшталт тих, що перераховані вище, але до сих пір не існує жодного сервісу, який би дозволив продавати свій автомобіль онлайн. Зараз для продажу свого авто, житель Кувейту мусить витратити біля 7-10 днів для продажу свого автомобіля на реальному “офлайн” ринку. Він має пригнати свій автомобіль, купити місце на місцевому базарі, найняти людину, яка буде чекати поки з’явиться зацікавлений покупець та тільки потім вони зможуть почати переговори.

Онлайн-сервіс “Carma” змінить цю ситуацію та дозволить продавати свій автомобіль онлайн, без потреби у фізичному розташуванні авто на базарах.

Користувач цього сервісу зможе додати свій автомобіль у лічені хвилини, вказавши мінімальний набір даних про сам автомобіль. Для продавців буде створено iOS додаток, а для людей, що зацікавлені у покупці цих автомобілів буде можливість переглядати усі автомобілі також онлайн, використовуючи web-сайт. iOS додаток Carma першої версії матиме мінімальний набір функціоналу, потрібний для

продажу авто, а з розвитком сервісу та наявністю певної статистики по продажах стане можливим вбудувати у сервіс функціонал оцінки автомобіля онлайн, що значно підвищить зацікавленість клієнтів у користуванні цього сервісу.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ СЕРВІСІВ ТА ЗАСОБІВ РОЗРОБКИ

1.1. Існуючі сервіси

Для розробки нового додатку по продажу автомобіля буде розглянуто існуючі схожі сервіси. Проаналізувавши їх, можна виділити основні переваги та недоліки і використати ці дані для створення нового подібного сервісу.

РСТ

Досить популярний майданчик, на якому можна онлайн купити авто в Україні. Тут також в основному представлені б/у машини, але можна придбати і зовсім нові авто за дуже цікавими цінами. Варто звернути увагу на те, що на цьому інтернет ресурсі немає можливості придбати запчастини.

На сайті РСТ автолюбителі мають можливість безкоштовно розмістити оголошення про продаж/обмін авто або скористатися зручним пошуком за допомогою фільтрів пошуку для покупки машини. Також ресурс дозволяє ознайомитися з пропозиціями спеціалізованих автосалонів.

Авторія

Авторія - найпопулярніший онлайн авторинок в Україні. Його основна спеціалізація - продаж б/у автомобілів. Тут ви також можете знайти запчастини.

Також, на сайті розміщується корисна для водіїв інформація з продажу та купівлі автомобілів, є ломбард, можливість придбати авто для розбирання на запчастини та інше. Для запобігання випадкам шахрайства на сайті обов'язкова

реєстрація за допомогою SMS. Щодня на сайті розміщується до 1,5 тисячі нових авто-оголошень з різними комерційними пропозиціями.

Інфокар

«Автобазар.Інфокар» - в основному спеціалізується на б/у автомобілях. На сьогодні український авторинок InfoCar.ua допомагає продати:

- Старі чи нові легкові авто
- Вантажні автомобілі
- Автобуси і мікроавтобуси
- Водний транспорт
- Мотоцикли, мопеди, моторолери
- Причепи та будинки-дачі
- Спецтехніку

Autoua

Перший автоклуб Autoua.net - найбільше і популярне автомобільне співтовариство в Україні. На сторінках порталу автолюбителі діляться досвідом, вирішують свої проблеми, обговорюють і вибирають, купують і продають машини. Читають новини і тест-драйви.

У розділі Маркет знаходиться каталог, що включає всі нові пасажирські автомобілі, представлені в салонах офіційних дилерів. З цінами,технічними характеристиками, комплектаціями, фотографіями і посиланнями на автосалони.

Майстер підбору допоможе вибрати автомобілі відповідно до заданих параметрів, включаючи ціну, тип кузова, об'єм двигуна, потужність, тип коробки передач і так далі. Там же можна ознайомитися з відгуками власників, які вже встигли дізнатися про цю машину все хороше і погане. Якщо ж потрібно купити

автомобіль з пробігом, то на допомогу прийде розділ Базар, що включають понад сто тисяч оголошень про продаж авто від відвідувачів Автоуа і сайту Автобазар.

Автомобільні новини, тест-драйви, статті та блоги на Autoua.net допоможуть не заблукати в автомобільному світі, даючи інформацію про новинки авто, достоїнства і недоліки продаються моделей, а також юридичних нюансах покупки і вибору нових машин. У блогах кожен може висловити свою думку на автомобільну тему, розповісти про подорожі на авто.

АвтоПортал

АвтоПортал - це стартова сторінка автолюбителя в Україні. Проект висвітлює всі напрямки автомобільного ринку - нові авто, оголошення про продаж б/у автомобілей, новини, статті, тест-драйви та унікальні сервіси для автолюбителів. Сайт наповнений перевіренною і ретельно відібраною інформацією, авторськими матеріалами. Вибрати і купити новий автомобіль на автопортали можна досить просто і швидко - завдяки зручному підбору авто за параметрами, сервісу порівняння автомобілів і інтуїтивно простию навігацією. Кожна модель автомобіля в каталозі має інформацію про комплектації, технічні характеристики, інформацію про конкурентів, автосалонах, акції, СТО і велику фото галерею інтер'єру і екстер'єру автомобіля. Власники залишають свої відгуки, що істотно допомагає у виборі автомобіля.

Старі автомобілі мають окремий розділ, в якому щодня додається більше сотні нових оголошень від продавців. Розділ має просту структуру і зручний пошук. Подача оголошення максимально спрощена і займає всього кілька хвилин.

OLX

Звичайно не можна обійти стороною топового лідера серед дощок оголошень - OLX.

Зручний інтерфейс дозволяє швидко знайти авто або запчастини, відповідні персональним запитам покупця. Також автомобілісти без зусиль зможуть розмістити оголошення про продаж будь-яких автомобільних товарів.

АВТОСАЙТ

Автобазар АВТОСАЙТ є однією з найпопулярніших інтернет майданчиків автопродажів в Україні. Щодня цей сайт відвідують десятки тисяч потенційних покупців б/у авто.

Автобазар складається з таких розділів:

- легкові авто
- мототехніка
- вантажні автомобілі
- автобуси і мікроавтобуси
- спецтехніка - будівельна, дорожня і ін.
- водний транспорт.

На сайт кожен день додаються сотні нових оголошень з усієї України. На автобазарі АВТОСАЙТ можна швидко знайти оголошення про продаж б / у авто в вашому місті.

AVITO.RU

AVITO.RU - найпопулярніший і найбільший сайт приватних оголошень в Росії (а також 3-й за розміром у світі і найбільший в Європі). Крім того, як показує статистика, саме тут розміщено найбільшу кількість оголошень про продаж автомобілів.

Досить зручна подача оголошення (за бажанням вам також дається можливість вказати найрізноманітніший вид опцій в продається вами автомобілі). Можна подати оголошення з мобільного телефону, встановивши відповідне мобільний додаток. Є можливість захистити номер свого телефону від спаму. Ліміт безкоштовних оголошень: 1 оголошення в місяць.

Інтерфейс вибору категорії шуканих вами автомобілів простий і досить функціональний (наявного набору необхідних опцій цілком вистачає). Немає можливості вибору відразу декількох моделей автомобілів (різних марок) і декількох міст і регіонів, але є можливість вибору певних районів міста (що може бути актуальним для великих міст).

Дуже велика популярність Avito в сукупності з його простотою і слабким адмініструванням - це в той же час і його проблема, на сайті дуже багато шахраїв і фейковий оголошень.

AUTO.RU

AUTO.RU - один з найстаріших автомобільних інтернет-порталів з уже майже 25-річною історією, але який не стоїть на місці, а постійно розвивається. Про це вказує і сам інтерфейс сайту (на мій погляд один з найбільш зручних), і наявність величезної кількості додаткових сервісів та інструментів для зручності покупки / продажу авто. Підсумкова оцінка - 89 з 100 балів. На нашу думку, найкращий сайт з продажу та купівлі автомобілів.

Треба відзначити, що в 2014 році портал AUTO.RU приєднався до компанії "Яндекс" і став не тільки його офіційним партнером, але і його тематичним підкаталогом на тему "АВТО" (тепер по посиланню auto.yandex.ru знаходиться портал auto.ru) . Природно все це істотно впливає на популярність як самого порталу, так і на що розміщуються на ньому оголошень, так як самі знаєте який пошуковик "рулить" в Росії. Саме приєднання до Яндексу остаточно зробило AUTO.RU найбільшим в рунеті агрегатором оголошень про продаж автомобілів.

Теперь не потрібна реєстрація - досить підтвердити номер свого телефону. Або ж, якщо у вас є акаунт в Яндексі, то ви легко за допомогою нього можете зайти або зареєструватися на порталі.

Дуже зручна покрокова подача оголошення - за бажанням вам дається можливість вказати найрізноманітніші характеристики і абсолютно будь-які опції продаваного вами автомобіля. Ви навіть можете прикріпити відео-огляд на свій автомобіль, розміщений в Youtube.

Можна подати оголошення з мобільного телефону, встановивши відповідне мобільний додаток auto.ru. Причому можливості там практично такі ж, як і на самому сайті. Є можливість захистити свій контактний телефон від всілякого спаму.

Перед подачею оголошення про продаж свого автомобіля, ви можете оцінити його ринкову вартість за допомогою сервісу "Оцінка авто", який на основі аналізу

декількох мільйонів оголошень і двох мільйонів проданих автомобілів, а також поточну ситуацію на ринку авто, допоможе вам визначитися з ціною на продаваний вами автомобіль. Ліміт безкоштовних оголошень: 1 оголошення протягом 6 місяців.

Пошук оголошень:

- Зручний інтерфейс вибору категорії шуканих вами автомобілів (аж до конкретної моделі і комплектації), а також інтервалу пробігу і інтервалу цін.
- Можливість вибору відразу декількох моделей автомобілів (навіть різних марок).
- Можливість вибору кількох міст або регіонів.
- Можливість пошуку нового автомобіля і кращої пропозиції у перевірених дилерів.

Крім того в спеціальному блоці відображається вся інформація про можливі знижки.

Багато оголошення містять звіт про перевірку автомобіля по VIN, який допоможе вам у великій мірі убезпечити себе від шахраїв.

Якщо Вас автомобіль виставляється не в перший раз, то вам нададуть всю історію розміщення цього авто на auto.ru.

При виборі певної моделі автомобіля, в додаткові сервіси "Відгуки" і "Тест-драйви" можна ознайомитись з відгуками по обраної моделі, за якими ви завжди можете хоч в якійсь мірі оцінити надійність і познайомиться з особливостями і тонкощами шуканого вами авто.

За допомогою додаткового сервісу "Оцінка авто" ви завжди можете оцінити приблизну ринкову вартість продаваного або купується вами автомобіля.

За допомогою розділів "Відгуки" і "Тест-драйви" ви завжди можете оцінити надійність і ходові якості шуканого вами авто і познайомиться з іншими його особливостями і тонкощами.

Для кожної моделі автомобіля є підрозділ "Статистика цін", за допомогою якого можна побачити графік динамічного зміни ціни за останній час, а також побачити "Як дешевшає шукане авто з віком".

Є сервіс пошуку нових і б / у автозапчастин, а також шин і дисків.

На порталі також присутні власні тематичні форуми і новинний журнал, але вони не так розвинені, як у деяких конкурентів.

До переваг auto.ru можна віднести і те, що вони постійно намагаються боротися з різного роду шахраями і фейковий оголошеннями, а також можливість підготовки документів купівлі-продажу авто в один клік.

1.2. Інтегроване середовище розробки

Для розробки додатку було обране інтегроване середовище розробки Xcode. Це IDE було розроблено компанією Apple. Дозволяє створювати програмне забезпечення для багатьох платформ, таких як: iOS, tvOS, watchOS, macOS та інші. Також у пакет входить набір бібліотек Cocoa та підтримка розробки на таких мовах як Swift та Objective-C.

За допомогою Xcode, розробник може програмувати логіку програми, створювати UI інтерфейс, писати Unit та UI тести, публікувати додатки у магазин додатків Apple - AppStore.

1.3. Мова програмування

Swift - відносно нова мова програмування, перше версія якої була представлена компанією Apple у 2014 році. Swift є компільованою мовою програмування, яка використовується в першу чергу iOS та macOS розробниками. Swift працює з фреймворками Cocoa та Cocoa Touch.

Мова Swift розроблялася як легка для читання та зрозуміла для програмістів. Також один з величезних плюсів цієї мови - вона строго типізована, що робить її більш стійкою до помилок програмістів, бо багато помилок можливо виявити на етапі компілювання програми. Більшість сторонніх бібліотек використовують саме цю мову програмування, але код написаний на Swift такод може працювати і з

кодом Objective-C у рамках одного проєкту. Swift дозволяє легко описувати об’єктно-орієнтовані конструкції з яких буде створено більша частина проєкту.

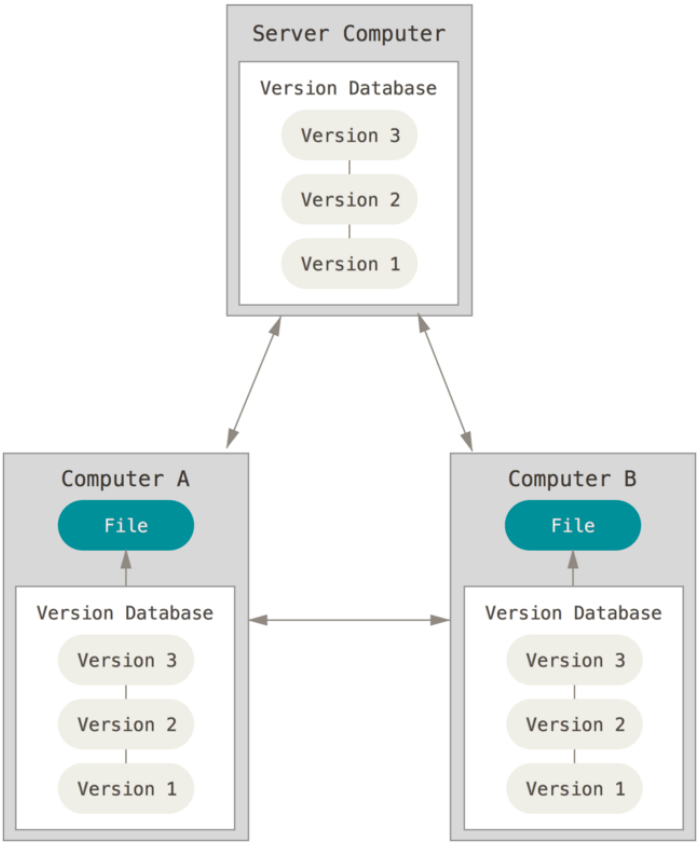
Виходячи з усіх вище перерахованих плюсів, для розробки проєкта було обрано

мову програмування Swift версії 5.0.

1.4. Система керування версіями

Система керування версіями - це система, що записує зміни у файл або набір файлів протягом деякого часу так, що користувач зможе повернутися до певної версії пізніше.

Для нашого проєкту буде використовуватися децентралізована система контролю версій Git (Рис. 1.4.1). Завдяки децентралізації, клієнти не просто отримують останній знімок файлів репозитирія, а вони є повною копією сховища разом з усією його історією. Таким чином, якщо перестає працювати будь-який сервер, через який співпрацюють розробники, будь-який з клієнтських репозиторіїв може бути скопійований назад до серверу, щоб відновити його. Кожна копія дійсно



є повною резервною копією всіх даних.

Рис. 1.4.1. Децентралізовані системи контролю версій.

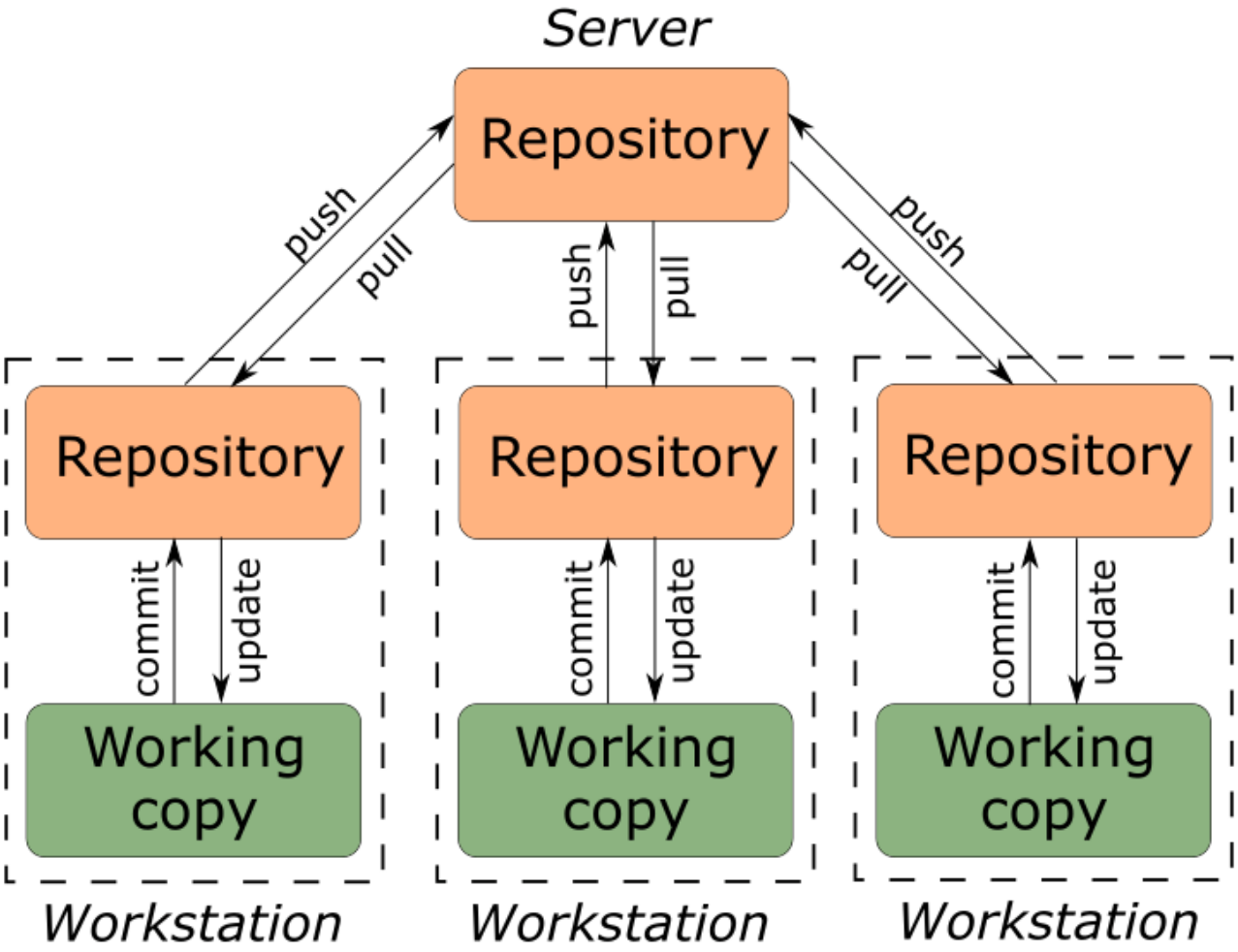
Для більшої стабільності та ускорення синхронізації різних версій проєкта локальний репозиторій зберігають онлайн у спеціальних сервісах: Github, Gitlab, Bitbucket.

З приходом СКВ працювати стало набагато простіше:

- можна скасовувати зміни;
- можна швидко і безболісно відновлювати пошкоджені файли;

- можна визначити, хто з команди писав певний блок коду;
- можна стежити за процесом, навіть якщо в один і той же момент над одним модулем працює кілька розробників або навіть команд по всьому світу.

Використання Git допомагає співробітникам завжди знати, що і коли сталося з файлом. У Git у кожного розробника є своя власна повноцінна робоча площадка, на якій і ведеться вся робота. Періодично (зазвичай в кінці робочого дня або після виконання завдання) проводиться синхронізація з віддаленим головним репозиторієм (Рис. 1.4.2.). Для синхронізації з репозиторієм буде використовуватися



клієнт системи керування версіями SourceTree, який спрощує взаємодію з репозиторіями Git.

Рис. 1.4.2. Схема роботи Git

1.5. Менеджер залежностей

Менеджери залежностей проєкту - керують залежностями в розрізі конкретного проекту. Тобто, в їх завдання входить опис залежностей, завантаження та оновлення їх коду.

На проєкті Carma буде використовуватися менеджер залежностей - CocoaPods. CocoaPods - це засіб управління залежностями Cocoa-бібліотек, які розробники використовують у своїх iOS та macOS проєктах.

iOS-розробники часто використовують напрацювання інших розробників (3rd party розробників). І як правило, бібліотеки поставляються з вихідним кодом. І звичайно ми додаємо вихідний код бібліотек в наш проект, одного разу взявши його зі сховищ розробників. Тобто замість скачування коду зі сховищ бібліотеки і копіювання в папку нашого проєкту ми можемо надати можливість CocoaPods зробити все за нас. У такого підходу є кілька недоліків:

- Іноді буває складно стежити за версіями бібліотек та їх зв'язками між собою;
- Немає одного загального місця, де можна переглянути перелік усіх доступних бібліотек. Безумовно, github - одне з великих притулків opensource-проєктів, але не єдине;
- Необхідно завжди пам'ятати про те, що вихідних код таких бібліотек потрібно оновлювати.

Менеджер управління залежностями CocoaPods допоможе вирішити усі перелікові проблеми. CocoaPods розбереться з залежностями між бібліотеками, які ви використовуєте, завантажить їх, створить і буде підітримувати структуру проєкту в належному вигляді.

Звісно є альтернатива CocoaPods - Carthage та Swift Package Manager, але обидва менеджери працюють не з усіма сторонніми бібліотеками. І тому було вирішено використовувати для проєкту Carma саме CocoaPods.

ВИСНОВКИ ДО РОЗДІЛУ 1

1. Більшість з перерахованих сервісів не мають мобільного додатку, через що у них є можливість зробити процес створення оголошень чи пошуку автомобілів більш обширним та детальним. Так як Carma для продавців буде функціонувати тільки через мобільний додаток, перелік кроків для створення оголошень має бути мінімальним.

2. Кожен з перерахованих сервісів має свої додаткові переваги, які створені для зацікавлення користувачів: новини, блоги та інше. Продавці автомобілів у Carma не будуть вказувати ціну на автомобіль, усі ціни будуть вказуватися покупцями, щось накшталт аукціону, але продавець може обрати будь-якого покупця для продажу свого автомобілю. Також ми надамо можливість користувачу оцінити його автомобіль онлайн. Оцінка буде здійнюватися на основі вже проданних автомобілів, саме тому цей функціонал стане доступний у версії 2.0.

3. Для розробки iOS додатку використовують інтегроване середовище розробки Xcode, яке дозволяє створювати UI інтерфейс, писати Unit та UI тести, публікувати додатки у магазин додатків Apple - AppStore.

4. Найактуальнішою мовою програмування для розробки iOS додатку на сьогодні є Swift 5. Swift є строго типізованною мовою програмування, яку зараз використовують для розробки більшості сторонніх бібліотек.

5. Для більш гнучкої та безпечної розробки слід використовувати систему керування версіями. Саме СКР зберігає зміни коду програму, що дозволяє повернутися до певної версії пізніше програмного коду у будь-який час.

6. У розділі розглянуті проблеми ручного управління залежностями, тому для управління залежностями на проєкті буде використовуватися менеджер управління залежностями CocoaPods.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ДОДАТКУ

2.1. Архітектура

Додаток Carma будується на основі двухланкової архітектури “Клієнт-сервер”. Архітектура “Клієнт-сервер" визначає загальні принципи організації взаємодії в мережі, де є сервери, вузли-постачальники деяких специфічних функцій (сервісів) і клієнти, споживачі цих функцій. Двухланковою вона називається через необхідність розподілу базових компонентів між двома вузлами:

клієнтом і сервером (рис. 2.1.1.)

Рис. 2.1.1. Двухланкова клієнт-серверна архітектура

2.2. Шаблон проєктування додатка

Існує багато шаблонів проєктування архітектури додатка, але найпопулярнішою серед iOS розробників є шаблон проектування MVC. Саме цей шаблон Apple використовує у своїй Core Cocoa бібліотеці й рекомендує цей шаблон для використання усіма розробниками, які створюють програмний продукт для екосистеми Apple.

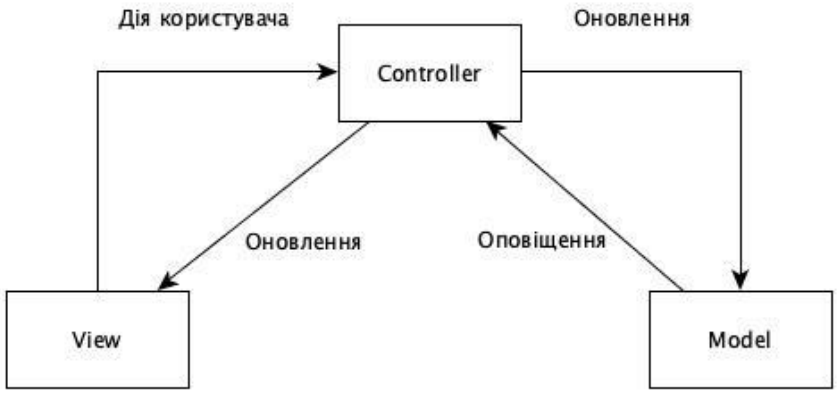
Розглянемо ключові особливості MVC, переваги та недоліки. MVC (модель, преставлення, контролер) - архітектурний шаблон, що передбачає поділ системи

на три взаємопов’язані частини: модель даних, вигляд або представлення та модуль керування.

Model об'єкти інкапсулюють дані, специфічні для програми та визначають логіку та обчислення, які маніпулюють та обробляють ці дані. Наприклад, Model об'єкт може представляти персонажа в грі або контакту в адресній книзі.

View об'єкт - це об'єкт у програмі, який користувачі можуть бачити. View вміє малювати себе і може реагувати на дії користувача. Основна мета об'єктів перегляду - відображення даних із моделей об'єктів програми.

Controller об’єкт виступає посередником між одним або кількома об’єктами View та одним або кількома об'єктами його моделі (Model). Об'єкти контролера, таким чином, є каналом, через який об'єкти View дізнаються про зміни в Model об'єктах і навпаки. Об'єкти контролера також можуть керувати життєвими циклами



інших об’єктів.

Рис. 2.2.1. Діаграма взаємодії між компонентами шаблону MVC

Як бачимо з діаграми взаємодії компонентів шаблону MVC (Рис. 2.2.1), View та Model не мають прямого зв’язку між собою. Саме це і є основним правилом шаблону MVC.

Загалом MVC досить непоганий шаблон проектування, але має свої недоліки:

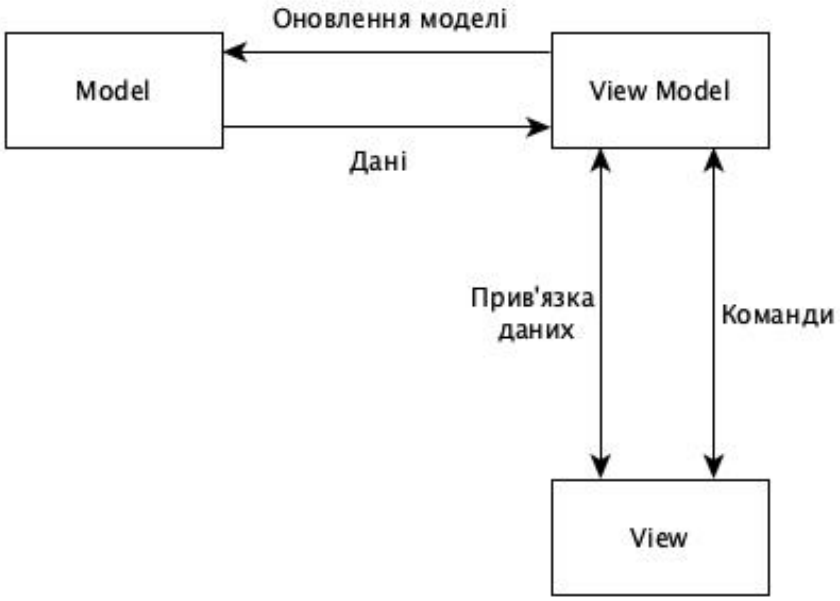
- Важко писати Unit тести. Так як уся логіка зусереджена в об’єкті Contoller, то тестувати зазвичай потрібно саме його, але цей об’єкт має дуже тісний зв’язок з View та Model. Якщо створити Mock для Model легко, то працювати з Mock View доволі важко.
- Слабке розділення відповідальності. З шаблоном MVC доволі легко порушити один з принципів SOLID, а саме - принцип єдиної відповідальності (Single Responsibility Principle). Об’єкт Contoller має дуже багато відповідальності. Він має обробляти дії користувача, слідкувати за змінами у Model, оновлювати Model та View, а також має працювати зі сторонніми сервісами, наприклад сервіс геолокації, камери, тощо. Також у клієнт-серверних додатках Controller працює з API серверу, відправляючи запити та обробляючи відповіді.

Через ці значні недоліки, були зроблені висновки, що MVC підходить для невеликих проєктів та пагано маштабується. Тому для нашого проєкту буде обрано інший, більш гнучкий та добре підходящий для великих проєктів шаблон - MVVM.

Шаблон MVVM (Model-View-ViewModel) дозволяє відокремити логіку додатки від візуальної частини (View). MVVM є архітектурним шаблоном, тобто він задає загальну архітектуру програми.

Даний патерн був представлений Джоном Госсманом (John Gossman) в 2005 році. MVVM застосовується в різних технологіях, в тому числі при розробці під Android, iOS та інші.

MVVM складається з трьох компонентів: моделі (Model), моделі



представлення (ViewModel) і представлення (View) (Рис. 2.2.2).

Рис. 2.2.2. Діаграма взаємодії між компонентами шаблону MVVM

Модель описує використовувані в додатку дані. Моделі можуть містити логіку, безпосередньо пов'язану цими даними, наприклад, логіку валідації властивостей моделі. У той же час модель не повинна містити ніякої логіки, пов'язаної з відображенням даних і взаємодією з візуальними елементами управління.

View або представлення визначає візуальний інтерфейс, через який користувач взаємодіє з додатком.

Хоча View (клас UIView) у UIKit може містити як інтерфейс, так і прив'язаний до нього код, проте код Swift не повинен містити логіки, тільки код анімацій, біндінгу, ініціалізацію View. Вся ж основна логіка додатки виноситься в компонент ViewModel.

ViewModel або модель представлення пов'язує Model і View через механізм прив'язки даних. Якщо в моделі змінюються значення властивостей, йде зміна відображуваних даних в View, хоча безпосередньо Model і View не пов’язані. ViewModel також містить логіку по отриманню даних з моделі, які потім

передаються у View. І також VewModel визначає логіку по оновленню даних в моделі.

View взаємодіє з ViewModel за допомогою команд. Наприклад, користувач хоче зберегти введені в текстове поле дані. Він натискає на кнопку і тим самим відправляє команду у ViewModel. А ViewModel вже отримує передані дані і відповідно до них оновлює модель.

Підсумком застосування шаблону MVVM є функціональний розподіл програми на три компонента, які простіше розробляти і тестувати, а також в подальшому модифікувати і підтримувати.

Для багатьох випадків трьох компонентів шаблону MVVM достатньо, але для більш складних програмних систем є сенц виділити ще декілька додаткових.

Наприклад, ми маємо розглянути питання:

- Де має міститися код роботи з зовнішнім API?
- Де має міститися код взаємодії з обладнанням (камера, геолокація, bluetooth та інші)?
- Де має місце взаємодія з локальною базою даних?

Для усіх подібних випадків можна створювати окремі сервіси (APIService, GeolocationService, CameraService та інші).

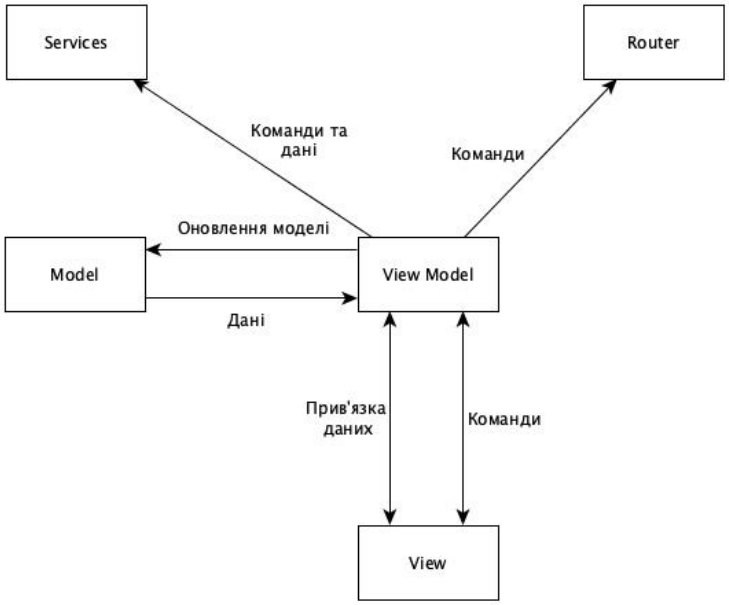
Також у кожному додатку є таке поняття як “Роутінг”. Роутинг потрібен для переходів з одного єкрану на інший, а також саме під час переходу створюється нова View та ViewModel, які в подальшому будуть зв’язані між собою. Було вирішено винести Router як окремий компонент, який буде відповідати за показ та створення екранів.

У цьому випадку наш шаблон MVVM буде мати такий вигляд (Рис. 2.2.3.)

Рис. 2.2.3. Діаграма взаємодії між компонентами модифікованного шаблону MVVM

2.3. Реактивне програмування. RxSwift.

Для MVVM дуже важливим є зв’язування даних між компонентом Model та



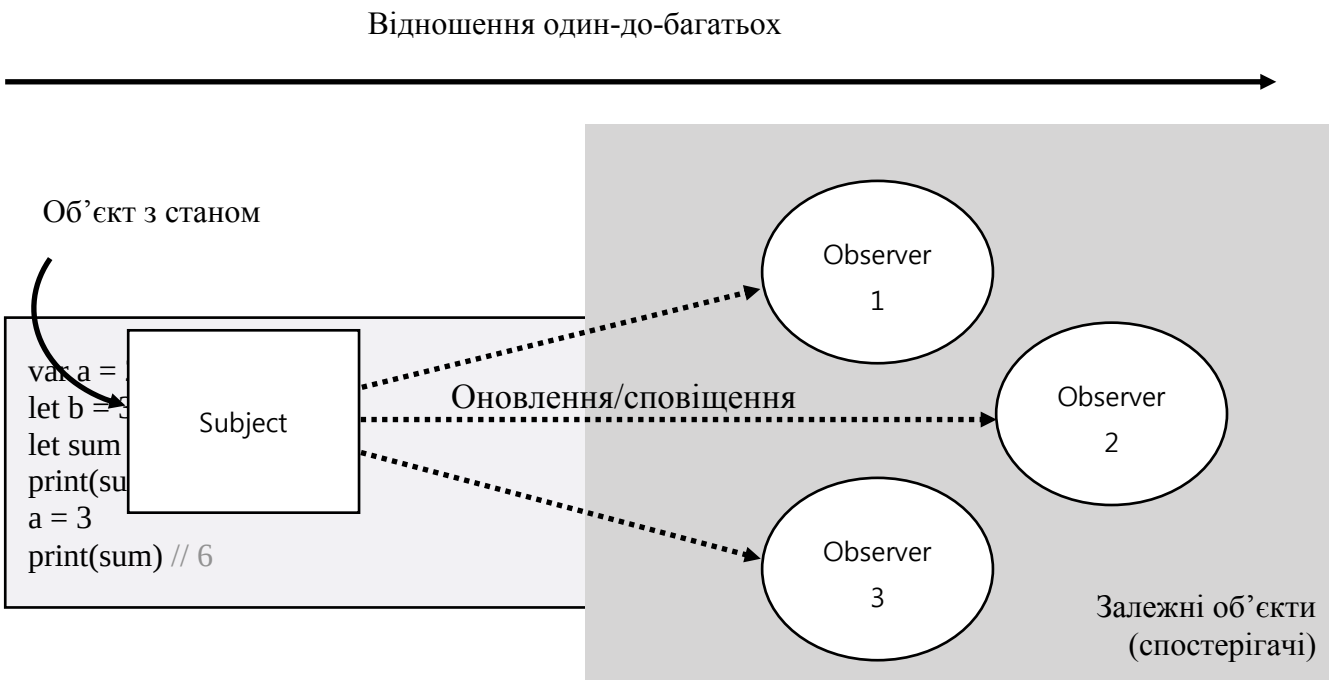
ViewModel. Для реалізування цієї задачі було вирішено використовувати реактивне програмування.

Реактивність - це здатність реагувати на будь-які зміни. В цьому контексті зміни - це зміни даних.

```
var a = 2
let b = 3
let sum = a + b
print(sum) // 5
a = 3
print(sum) // 5
```

Розглянемо приклад:

Значення не змінилось, бо дані необхідно перераховувати. Даний приклад демонструє звичну нам імперативну парадигму програмування. На відміну від



імперативного підходу, реактивний підхід будується на push стратегії поширення змін. Push стратегія має на увазі, що в разі зміни даних ці самі зміни будуть "проштовхуватися", і залежні від них дані будуть автоматично оновлюватися. Ось як би поводився наш приклад, якби застосовувалася push стратегія:

Даний приклад показує реактивний підхід. Варто зазначити, що цей приклад не має нічого спільного з реальністю, я його привів лише з метою показати різницю в підходах. Реактивний код в реальних додатках буде виглядати зовсім інакше.

Принцип роботи реактивного коду досить легко можна пояснити, використовуючи шаблон ООП - Observer.

Observer - визначає зв’язок "один-до-багатьох" між об'єктами таким чином, що при зміні стану одного об'єкта відбувається автоматичне оповіщення та оновлення всіх залежних об’єктів (Рис. 2.3.1.).

Рис. 2.3.1. Схема взаємодії компонентів в шаблоні Observer.

Для багатьох мов програмування було створено ReactiveX (Reactive Extension) бібліотеки, у тому числі й RxSwift. Саме RxSwift ми будемо використовувати у нашому проєкті для біндінгу між View та ViewModel.

RxSwift - це бібліотека для роботи з асинхронним кодом, яка представляє події в вигляді потоків подій з можливістю підписатися на них, а також дозволяє

застосовувати до них підходи функціонального програмування, що значною мірою знижує зі складними послідовностями асинхронних подій.

Причиною виникнення подібних інструментів є те, що в процесі еволюції iOS SDK представив нам широкий вибір інструментів для роботи з асинхронними подіями: GCD, KVO, Notification Center, шаблон делегування. Проблема полягає в тому, що кожен з цих інструментів вимагає індивідуального підходу і певної кодової бази навколо них, при цьому їх поєднання може виявитися не надто елегантним. API, описуваний проектом ReactiveX - це спроба зробити універсальний мову для роботи з асинхронними подіями. Бібліотеки, які реалізують цей підхід, існують для великої кількості мов (RxJava, RxJS, RxKotlin, RxPHP, RxCpp), так що не можна сказати, що цей світ обмежується тільки мобільного розробкою.

Глобально компоненти коду на Rx можна розділити на 3 основних види: observables, operators і schedulers.

Клас Observable <T> є основним у RxSwift, його призначення полягає в тому, щоб дозволити одним класам підписатися на послідовності подій, що містять дані типу T, які транслюються іншими класами.

Якщо заглянути у протокол ObservableType, то там можна виявити єдиний метод subscribe, який підписує Observer на прийом подій з цієї послідовності.

Події являють собою простий enum (Event), і бувають трьох видів:

- next - служить для передачі наступного значення Observer;
- completed - говорить про успішне завершення послідовності. Це означає, що Observable успішно завершив свій життєвий цикл і більше не буде транслювати події;
- error - говорить про те, що виконання Observable завершилося з помилкою і інших подій транслюватися не буде.

Ще за своєю природою Observables можуть бути кінцевими (наприклад, скачування файлу) і нескінченними (наприклад, відстеження зміни орієнтації пристрою).

Реалізація класу Observable, містить певну кількість допоміжних методів, які можуть виробляти різні операції над елементами послідовності. Так як вони не мають побічних ефектів, то їх можна комбінувати разом, що дозволяє будувати в декларативному стилі досить складну логіку. Є сайт RxMarbles, який візуалізує роботу цих операторів, рекомендую ознайомитися.

Schedulers є в деякому роді аналог DispatchQueues в світі Rx. Вони абстрагують виконання завдань в різних потоках і чергах, дозволяючи домогтися оптимальної продуктивності. Це не найпоширеніша завдання, тому не варто загострювати увагу на цій функціональності на першому етапі.

RxSwift реалізує базове API, описане в специфікації проекту ReactiveX. Підтримка Rx компонентами iOS SDK, реалізована в модулі RxCocoa. Так ми з коробки маємо можливість підписатися на такі дії, як натискання кнопки, перемикання світчей і тому подібне.

2.4. Діаграма прецедентів

Діаграма прецедентів — в UML, діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.

Основні елементи діаграми - учасник (actor) і прецедент (варіант).

Учасник - це безліч логічно пов'язаних ролей, виконуваних при взаємодії з прецедентами або сутностями (система, підсистема або клас). Учасником може бути людина або інша система, підсистема або клас, які представляють щось поза суті. Графічно учасник зображується "чоловічком".

Прецедент (use case) - опис безлічі послідовних подій (включаючи варіанти), що виконуються системою, які призводять до спостережуваного учасником результату. Прецедент являє поведінку суті, описуючи взаємодію між учасниками і системою. Прецедент не показує, "як" досягається певний результат, а тільки "що" саме виконується. Прецеденти позначаються дуже простим чином - у вигляді еліпса, всередині якого вказано його назву.

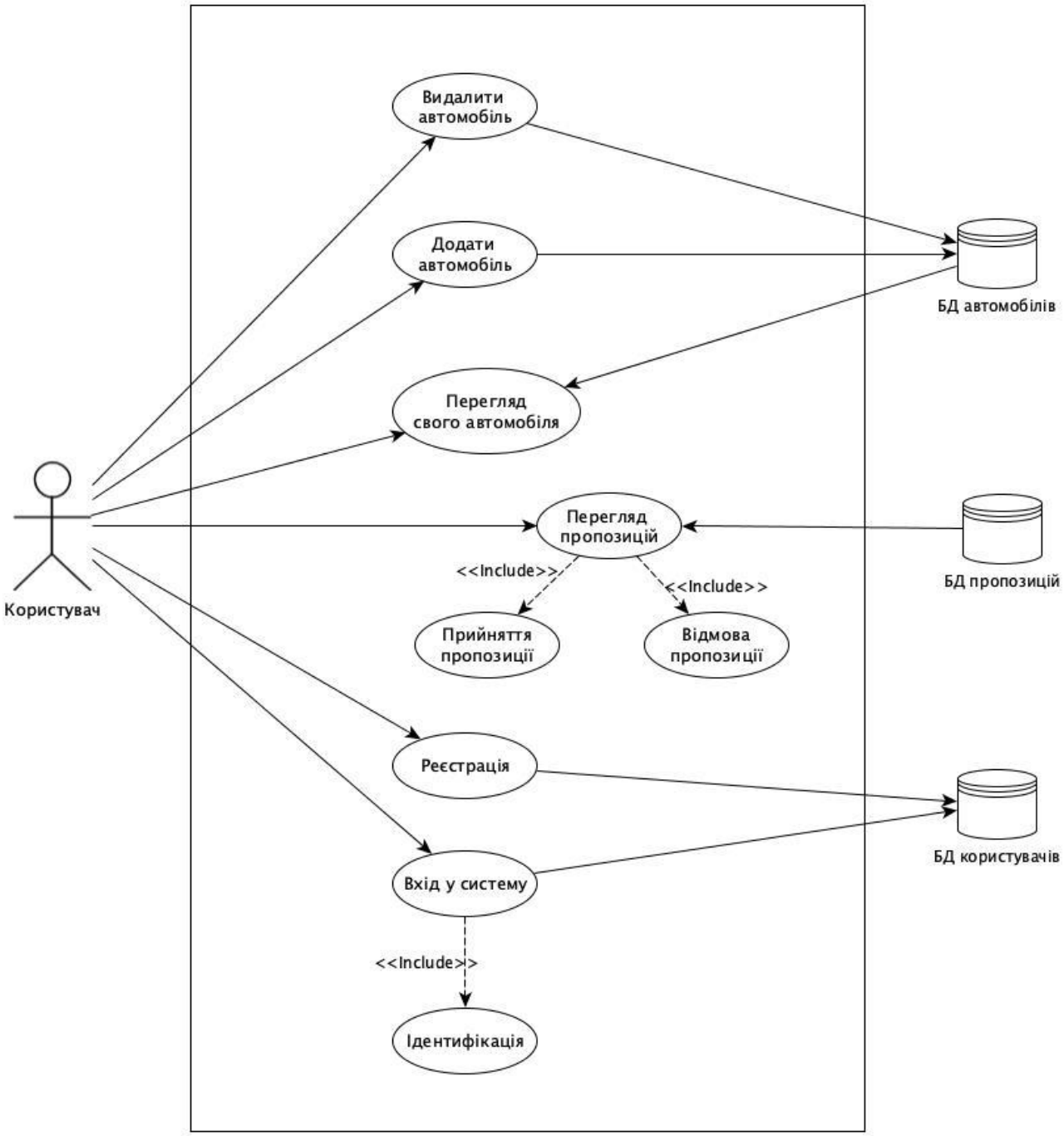


Рис. 2.4.1. Use-case діаграма додатку Carma

2.5. Реалізація

Додаток Carma використовує архітектурний шаблон MVVM, розширений сервісами та роутінгом. Усі ViewModel-і використовують потрібні їм сервіси, які передаються у конструктор ViewModel-і.

Наприклад ось як виглядає конструктор *CarModelsViewModel*, який

```
init(context: Context<MyCar>, carService: CarServiceType, localizer: LocalizerType)
```

використовується для відображення переліку моделей автомобілів:

Тут ми бачемо, що ViewModel буде використовувати *CarServiceType*, який інкапсулює у собі усі необхідні методи для роботи з API серверу. У конструкторах ми використовуємо протокол *CarServiceType*, а не конкретну реалізацію *CarService.*

Далі буде розглянуто:

- що таке протоколи та POP;
- як використовується кодо-генерація у проєкті;
- як саме відбувається підстановка реальної реалізації замість протоколу та що таке Dip;
- що таке сервіси і які сервіси використовує Carma;
- взаємодію з сервером та які допоміжні бібліотеки використовуються у Carma;
- механізм авторизації;
- Push-повідомлення;

2.6. Протоколо-орієнтоване програмування (POP)

На WWDC 2015 компанія Apple оголосила, що Swift є першим протоколо-орієнтованою мовою програмування.

Протоколо-орієнтоване програмування - нова парадигма програмування, яка з’явилась у Swift версії 2.0. У протоколо-орієнтованому підході ми починаємо проектувати нашу систему, визначаючи протоколи. Ми спираємось на нові поняття: розширення протоколу, успадкування протоколу та композицію протоколів.

У Swift типи значень переважні перед класами. Однак, об'єктно-орієнтовані концепції не працюють добре зі структурами та enum-ами. Структура чи enum не може успадковувати іншу структуру чи enum. Таким чином, успадковування - одна з

основних об'єктно-орієнтованих концепцій - не може бути застосована до типів значень (структури та enum-и). З іншого боку, типи значень можуть реалізовувати протоколи, навіть декілька протоколів. Таким чином, з POP вирішує усі наявні проблеми для типів значень у Swift.

Однією з потужних функцій Swift є його здатність розширювати протоколи. Протоколно-орієнтоване програмування використовує цю функцію і спонукає вас створювати архітектуру свого застосування, щоб в першу чергу створити один або кілька протоколів, а не переходити безпосередньо до конкретних типів.

Протоколи дозволяють задати певний інтерфейс, з яким повинен задовольняти клас, що підтримує цей протокол. Протоколи не можуть містити реалізацію методів, тому ми створюємо клас, стуктуру чи enum, який і буде реалізовувати методи та властовості, які перараховані у протоколі.

Приклад протоколу, який описує можливості сервісу, який використовується для

```
protocol PermissionServiceType: AutoMockable {
    func request(_ permissionType: PermissionType) -> Driver<MediaPermissionStatus>
}
```

запитів доступу до певних налаштувань користувача:

Цей протокол має один метод - *request,* який повертає статус дозволу до медіа-налаштувань користувача. Також цей протокол відповідає протоколу *AutoMockable,* який використовується для кодо-генерації.

2.7. Кодо-генерація

У підрозділі “Протоколо-орієнтоване програмування (POP)” було згадано про протокол *AutoMockable*, який використовується для кодо-генерації. Для кодо-генерації на проєкті Carma використовується Sourcery.

Sourcery - це генератор коду для мови Swift, побудований на базі власного SourceKit від Apple. Він розширює мовні абстракції, що дозволяє автоматично

генерувати шаблоний код. Він використовується у понад 30 000 проєктів як на iOS, так і на macOS.

```
class PermissionServiceTypeMock: PermissionServiceType {
    public var requestCallsCount = 0
    public var requestCalled: Bool {
        return requestCallsCount > 0
    }
    public var requestReceivedPermissionType: PermissionType?
    public var requestReturnValue: Driver<MediaPermissionStatus>!
    public var requestClosure: ((PermissionType) -> Driver<MediaPermissionStatus>)?

    public func request(_ permissionType: PermissionType) -> Driver<MediaPermissionStatus> {
        requestCallsCount += 1
        requestReceivedPermissionType = permissionType
        return requestClosure.map({ $0(permissionType) }) ?? requestReturnValue
    }
   public init() {}
}
```

Sourcery використовується у Carma для генерування коду створення:

- моків - об’єкт, службовець для цілей тестування і ведучий себе так само, як реальний об'єкт, але при цьому не є “справжнім”;
- генерування локалізованих стрічок для усіх підтримуваних мов;
- генерування enum-у з переліком усіх картинок, що завантажені у проєкт для їх використання з коду.

2.8. Впровадження залежностей

Залежність або залежне - означає покладатися на щось. Коли клас А використовує деяку функціональність з класу В, тоді клас А залежить від класу В (Рис. 2.8.1).

В Swift, перш ніж ми зможемо використовувати методи інших класів, нам необхідно для початку створити екземпляри цього класу (тобто клас А повинен створити екземпляр класу В).

Таким чином, передаючи завдання створення об’єкта іншому спеціальному об’єкту і пряме використання цієї залежності називається впровадженням залежностей. Впровадження залежностей слід сприймати як посередника у нашому коді, який виконує всю роботу по створенню об’єктів. Головною ідеєю впровадженням залежностей є те, що об’єкти залежать не від конкретної реалізації, а від інтерфейсів, або у випадку Swift - протоколів.

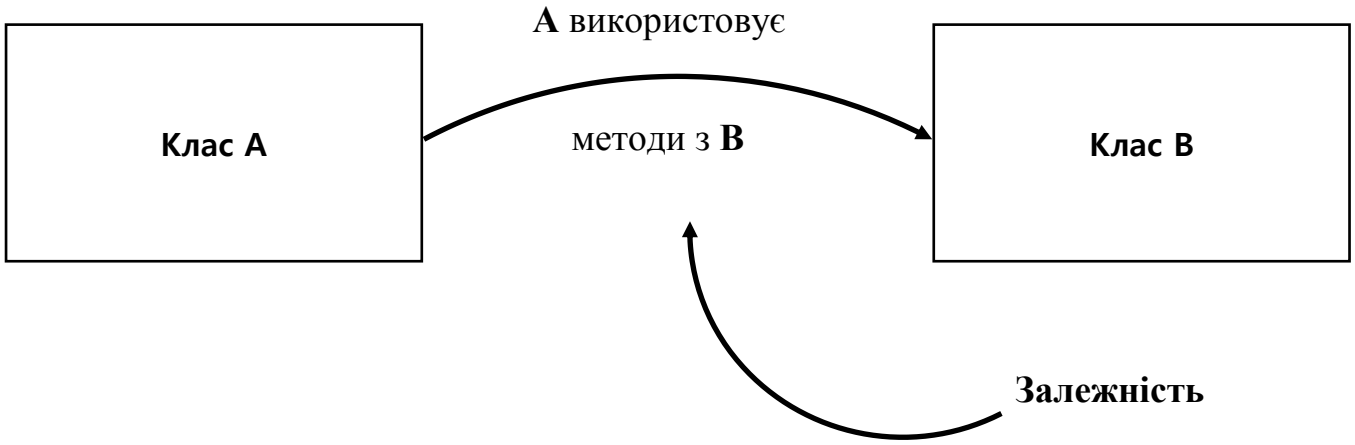


Рис. 2.8.1. Демонстрація залежності між класами

Переваги використання впровадження залежностей:

- Допомагає в модульному тестуванні
- Кількість шаблонного коду скорочується, оскільки ініціалізація залежностей виконується компонентом інжектора;
- Розширення програми стає ще простіше;
- Допомагає зменшити зв'язність коду, що важливо при розробці додатків.

До недоліків використання впровадження залежностей можна віднести: можливі помилки з процесу компіляції переходять у процес виконання програми.

Місце, де створюються конкретні екземпляри класів називають Dependency Injection Container. У проєкті Carma використовується стороння бібліотека Dip. Dip - це контейнер, який використовується для реєстрації та використування залежностей.

```
let container = DependencyContainer ()
```

Для використання Dip у проєкті, ми маємо створити контейнер:

```
container.register { MyService () as Service }
```

та зареєструвати свої залежності, пов'язуючи протокол або тип з фабрикою:

Потім для отримання екземпляру протоколу або певного типу з використанням

```
container.resolve() as Service
```

контейнеру ми маємо викликати:

```
func registerServices() {
        register(.singleton) { try PushNotificationsHandler(service: self.resolve(), sessionStorage:
self.resolve()) as PushNotificationsHandlerType }
        register { try FirebaseService(environment: self.resolve()) as FirebaseServiceType }
        register { PermissionService() as PermissionServiceType }
        register { try GalleryImageProvidingService(permissionService: self.resolve()) as
GalleryImageProvidingServiceType }
        register { try CameraImageProvidingService(permissionService: self.resolve()) as
CameraImageProvidingServiceType }
        register { try ImageProvidingService(galleryService: self.resolve(), cameraService:
self.resolve(), router: self.resolve(), localizer: self.resolve()) as ImageProvidingServiceType }
    }
```

Приклад реєстрування сервісів у проєкті Carma:

2.9. Сервіси

Сервіси - це об'єкти, які виконують певні операції або дії. Коли процес ускладнюється, його стає все складніше покривати тестами, або він взаємодіє більш ніж з одним типом моделі. Сервіс може допомогти зробити код чистіше.

Мета сервісу - ізолювати операції, відповідно до таких принципів:

- Строгість на вході і виході. Сервіс створюється, для роботи з дуже специфічними процесами, так що можна пожертвувати принципом надійності на користь створення дуже вузькоспеціалізованого інструменту.
- Ретельна документація. Цей модуль буде використовуватися в абсолютно несподіваних місцях, тому дуже важливо добре задокументувати його використання.

Завдяки тому, що ми виносимо певну логіку в єдиний модуль, ми централізували всі можливі зміни процесу, які можуть статися у майбутньому. Навіть якщо логіка, описана в сервісному об'єкті, стає все більш складною, тести все так же залишаються сфокусованими на одній операції, запобігаючи появі величезних файлів і громіздкою підготовки оточення.

Сервіс може бути дуже корисним інструментом для вичищена і рефакторінга коду. Ізольований код допомагає зробити логіку програми більш простою, акуратною, зручною для покриття тестами і, врешті-решт, забезпечать більш легку підтримку коду.

У Carma використовуються такі сервіси:

- PermissionService - сервіс, який використовується для запитів доступу до певних налаштувань користувача. Наприклад доступ до камери, нотіфікацій, тощо.
- FirebaseService - сервіс-обгортка для більш зручного використання Firebase у даному проєкті.
- Debugger - сервіс-логгер, який використовується для ведення логів створення та знищення єкранів.
- PushNotificationsHandler - сервіс у якому описана логіка роботи з Push Notifications.
- ImageProvidingService - сервіс, який використовується для імпорту фотографій з галереї або з фотокамери.

- Велика низка API сервісів, які створені для більш зручного спілкування з сервером.

```
public protocol PermissionServiceType: AutoMockable {
   func request(_ permissionType: PermissionType) -> Driver<MediaPermissionStatus>
}

public final class PermissionService: PermissionServiceType {
   public func request(_ permissionType: PermissionType) -> Driver<MediaPermissionStatus> {
      return Observable.create { observer in
         switch permissionType {
         case .camera:
            switch AVCaptureDevice.authorizationStatus(for: .video) {
            case .notDetermined:
               AVCaptureDevice.requestAccess(for: AVMediaType.video) { [observer] in
                  $0 ? observer.onNext(.authorized) : observer.onNext(.denied)
                  observer.onCompleted()
               }
            case .restricted: observer.onNext(.restricted); observer.onCompleted()
            case .denied: observer.onNext(.denied); observer.onCompleted()
            case .authorized: observer.onNext(.authorized); observer.onCompleted()
            @unknown default: observer.onNext(.unknown); observer.onCompleted()
            }
         case .photoLibrary:
            switch PHPhotoLibrary.authorizationStatus() {
            case .notDetermined:
               PHPhotoLibrary.requestAuthorization { [observer] in
                  $0 == .authorized ? observer.onNext(.authorized) : observer.onNext(.denied)
                  observer.onCompleted()
              }

             case .restricted: observer.onNext(.restricted); observer.onCompleted()
            case .denied: observer.onNext(.denied); observer.onCompleted()
            case .authorized: observer.onNext(.authorized); observer.onCompleted()
            @unknown default: observer.onNext(.unknown); observer.onCompleted()
            }
         }
         return Disposables.create()
      }.asDriver(onErrorDriveWith: .empty())
   }
}
```

Розглянемо деякі з цих сервісів:

Як бачимо, для сервісу *PermissionService* був створений протокол *PermissionServiceType*, який буде використовуватися у конструкторах інших компонентів додатка. І так як *PermissionServiceType* відповідає протоколу *AutoMockable*, то для цього протоколу генератор коду сгенерує відповідний Mock-об’єкт, який у подальшому буде використовуватися для написання Unit тестів.

```
public protocol FirebaseServiceType: AutoMockable {
   var configureTrigger: PublishRelay<Void> { get }
}
public class FirebaseService: FirebaseServiceType, ReactiveCompatible {
   public let configureTrigger = PublishRelay<Void>()
   init(environment: AppEnvironmentType) {
      configureTrigger.bind(onNext: {
         guard    let file = Bundle.main.path(forResource: environment.googleServiceInfo, ofType:
"plist"),
               let options = FirebaseOptions(contentsOfFile: file) else { fatalError("Can't create
firebas options") }
         FirebaseApp.configure(options: options)
      }).disposed(by: rx.disposeBag)
   }
}
```

Аналогічним чином реалізується і *FirebaseService:*

FirebaseService - сервіс-обгортка для більш зручного використання Firebase у даному проєкті. Firebase - це платформи розробки мобільних та веб-застосунків. Firebase розвивається з 2011 року компанією Firebase Inc., яку придбав Google у 2014 р.

У нашому проєкті використовується Firebase Crashlytics. Firebase Crashlytics - це репортер крашів у реальному часі, який допомагає відстежувати, визначати пріоритети та виправляти проблеми зі стабільністю, які погіршують якість програми. Crashlytics економить ваш час на усунення несправностей шляхом інтелектуального групування збоїв та виділення обставин, що призводять до них.

За допомогою цього сервісу ми зможемо дізнатисяс, чи впливає певна проблема додатку на багатьох користувачів, отримувати сповіщення, коли проблема несподівано зростає і навіть з’ясувати, які рядки коду викликають збої.

2.10. Взаємодія з API серверу

Головна роль в механізмі доступу до даних в проекті віддана класу *TRON*. Клас *TRON* взятий зі сторонньої бібліотеки - *TRON*. Ця бібліотека це легкий мережевий шар абстракції, побудований над *Alamofire*. З його допомогою можна

значно спростити взаємодію з веб-сервісами. *Alamofire* у свою чергу - мережева бібліотека HTTP, написана на Swift.

Клас *TRON* реалізує шаблон «Singleton», що приховує за собою налаштування підключення до API сервера, методи запитів по ключам API, налаштування відображення відповідей сервера з даними в об’єкти. У той же час надає єдиний інтерфейс роботи з даними у проєкті: досить викликати метод для отримання певних даних, а внутрішня логіка зробить запит до сервера і відобразить відповідь в набір об'єктів.

```
protocol SessionServiceType: AutoMockable {
    func post(email: String, password: String) -> Response<Session, SessionServiceError>
    func logOut() -> Response<Void, String>
}
```

Приклад API сервісу для механізму авторизації:

Це протокол API сервісу для отримання сессії, але навіть по ньому можна виявити у чому саме буде полягати суть цього сервісу.

Метод *logOut* використовується для видалення сессії. Для цього створюється запит

```
func logOut() -> Response<Void, String> {
   let request: Request<Void, String> = tron.request("session",
                                                     responseSerializer: VoidSerializationFactory())
   request.method = .delete
   return request.asResult()
}
```

“session" з HTTP методом delete.

Метод *post* з параметрами *email* та *password* використовується для створення сессії. У відповідь ми очікуємо JSON файл, який можна буде розпарсити та отримати або валидну модель сессії, або помилку.

Як бачимо з реалізації метода по створенню сессії, у ньому відбувається локальна валідація вхідних параметрів, що дозволяє перевірити правельність параметрів до

```
func post(email: String, password: String) -> Response<Session, SessionServiceError> {
        let errors: [SessionServiceError: [String]] = [:]
            <| (email.isEmpty, .email, localizer.localize(.emailError))
            <| (password.isEmpty, .password, localizer.localize(.passwordError))
        guard errors.isEmpty else { return .just(.failure(.init(errors))) }

        let request: Request<Session, SessionServiceError> = tron.swiftyJSON.request("session")
        request.method = .post
        request.parameters["email"] = email
        request.parameters["password"] = password
        return request.asResult()
  }
```

відправки їх на сервер. У разі, якщо параметри є недійсні, метод відразу поверне типізовану помилку *SessionServiceError*.

Отримана сессія буде екземпляром классу *Session* та буде збереженний у сховище Keychain. Сховище Keychain зберігає ваші паролі та інформацію облікового запису та зменшує кількість паролів, які ви повинні запам’ятати.

```
open class UserDefaultsSessionStorage: SessionStorageType, ReactiveCompatible {
    public let session: BehaviorRelay<Session?>

    public init(sessionKey: String = "session", keychainStorage: KeychainStorageType) {
        self.session = BehaviorRelay(value: nil)
        if let currentToken = keychainStorage.string(forKey: "token" + sessionKey) {
            session.accept(try? Session(token: currentToken))
        }

        self.session.asDriver().drive(onNext: { session in
            print("☐ Token in storage \(session?.token ?? "nil")")
            keychainStorage.set(session?.token, forKey: "token" + sessionKey)
        }).disposed(by: rx.disposeBag)
    }
}
```

Доступ до сессії, яка збережена у Keychain буде здійснюватися через клас *UserDefaultsSessionStorage*, який реалізує шаблон Singletone, доступний всіх класах проекту, існує протягом всієї сесії користувача:

2.11. Push-повідомлення

Основну роботу механізму push-повідомлень виконує фонова служба повідомлень операційної системи. Вона підключається до Apple Push Notification Server, отримує унікальний ключ пристрою. Цей ключ необхідно відправити на сервер, який буде відправляти повідомлення (Основний сервер).

Як тільки фонова служба повідомлень операційної системи отримує ключ, викликається метод делегату *AppDelegate*далі цей ключ передається у

```
func application(_ application: UIApplication,
didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {
        pushHandler.deviceTokenTrigger.accept(deviceToken)
    }
```

PushNotificationsHandler:

Протокол *PushNotificationsHandlerType:*

Для відправки ключа на основний сервер, *PushNotificationsHandler* використовує

```
public protocol PushTokenServiceType: AutoMockable {
    func register(token: String) -> Response<Void, String>
}
extension API {
    public class PushTokenService: API, PushTokenServiceType {
        public func register(token: String) -> Response<Void, String>  {
            let request: Request<Void, String> = tron.request("session", responseSerializer:
VoidSerializationFactory())
            request.method = .patch
            request.parameters["session[device_token]"] = token
            return request.asResult()
        }
    }
}
```

PushTokenService:

У момент відправки повідомлень, сервер формує список ключів пристроїв і відправляє його разом з текстом push-повідомлення на сервер повідомлень Apple. Сервер повідомлень в свою чергу розсилає повідомлення на пристрої (Рис. 2.11.1).

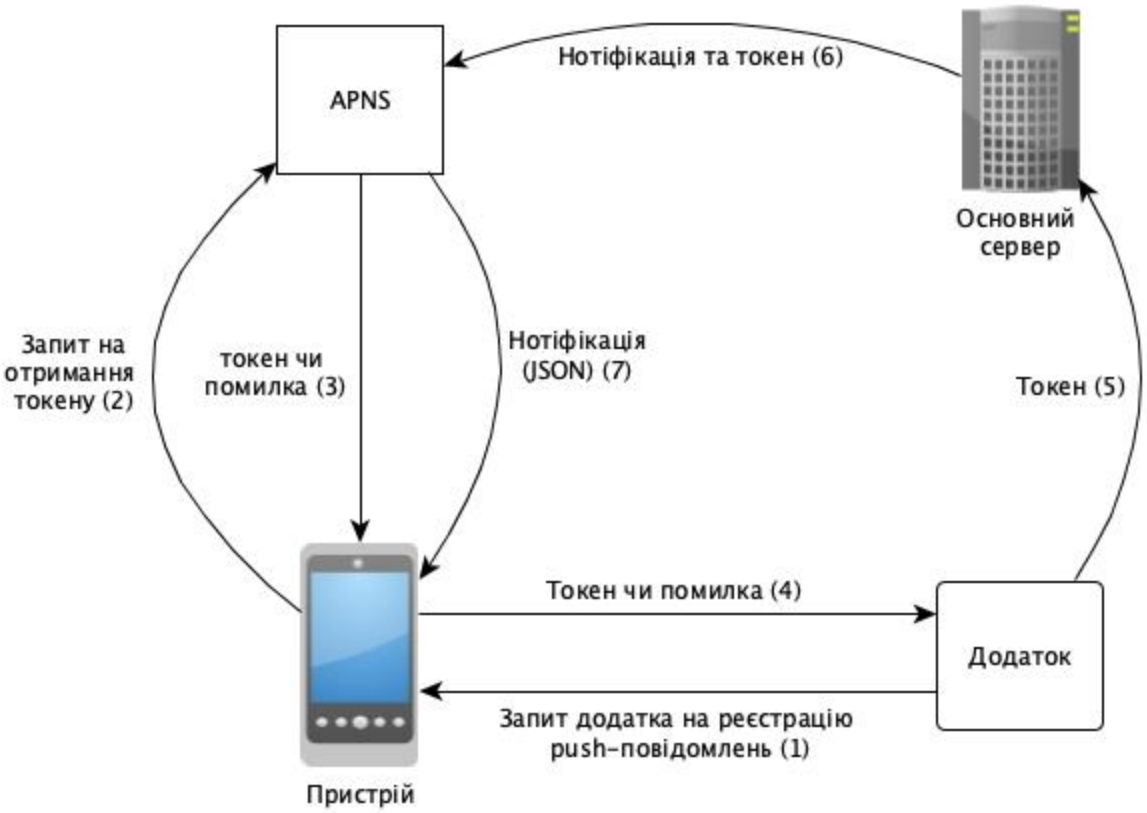


Рис. 2.11.1. Схема роботи Push-Remote нотіфікацій

У класі *AppDelegate* є можливість перевантажити метод, який викликається делегатом додатка після натискання на повідомлення в системній панелі повідомлення. Саме тут *PushNotificationsHandler* отримує всю необхідну інформацію для обробки типу повідомлення і подальшого роутингу.

У Carma використовуються push-пофідомлення для сповіщення користувача про нові пропозиції.

ВИСНОВКИ ДО РОЗДІЛУ 2

1. У розділі було розгянуто нову парадигму програмування - протоколо-орієнтоване програмування (ПОП). Саме цю парадигму буде використано під час розробки проєкту Carma.

2. Для створення mock-об’єктів, генерації стрічок буде використовуватися кодо-генерація від Sourcery. У розділі розглянуто основні можливості кодо-генерації.

3. Розглянуто основні сервіси, які використовуються у додатку Carma. Усі ці сервіси є об'єктами, які виконують певні операції або дії. Сервіс може допомогти зробити код чистіше та спрощує його тестування.

4. Взаємодія додатку з API серверу буде відбуватися з використанням сторонньої бібліотеки TRON, яка є мережевим шаром абстракції, побудований над Alamofire. Alamofire у свою чергу є мережевою бібліотекою HTTP.

5. Розгянуто основні сервіси по роботі з сессією користувача.

6. У розділі розглянуто роботу Push-повідомлень для додатку з переліком основних програмних компонентів та їх реалізацією.

7. У цьому розділі було розглянуто “Клієнт-серверну” архітектуру, яка буде використовуватися для додатку Carma.

8. Через недоліки MVC для великих проєктів, було вирішено використовувати архітектурний шаблон MVVM. У розділі було розглянуто ключові недоліки та переваги обох архітектурних шаблонів.

9. Для додатку Carma буде використовуватися розширений MVVM шаблон. Окрім Model, View та ViewModel також будуть використовуватися об’єкти Service (APIService, GeolocationService, CameraService) та Router, який буде відповідати за показ та створення екранів.

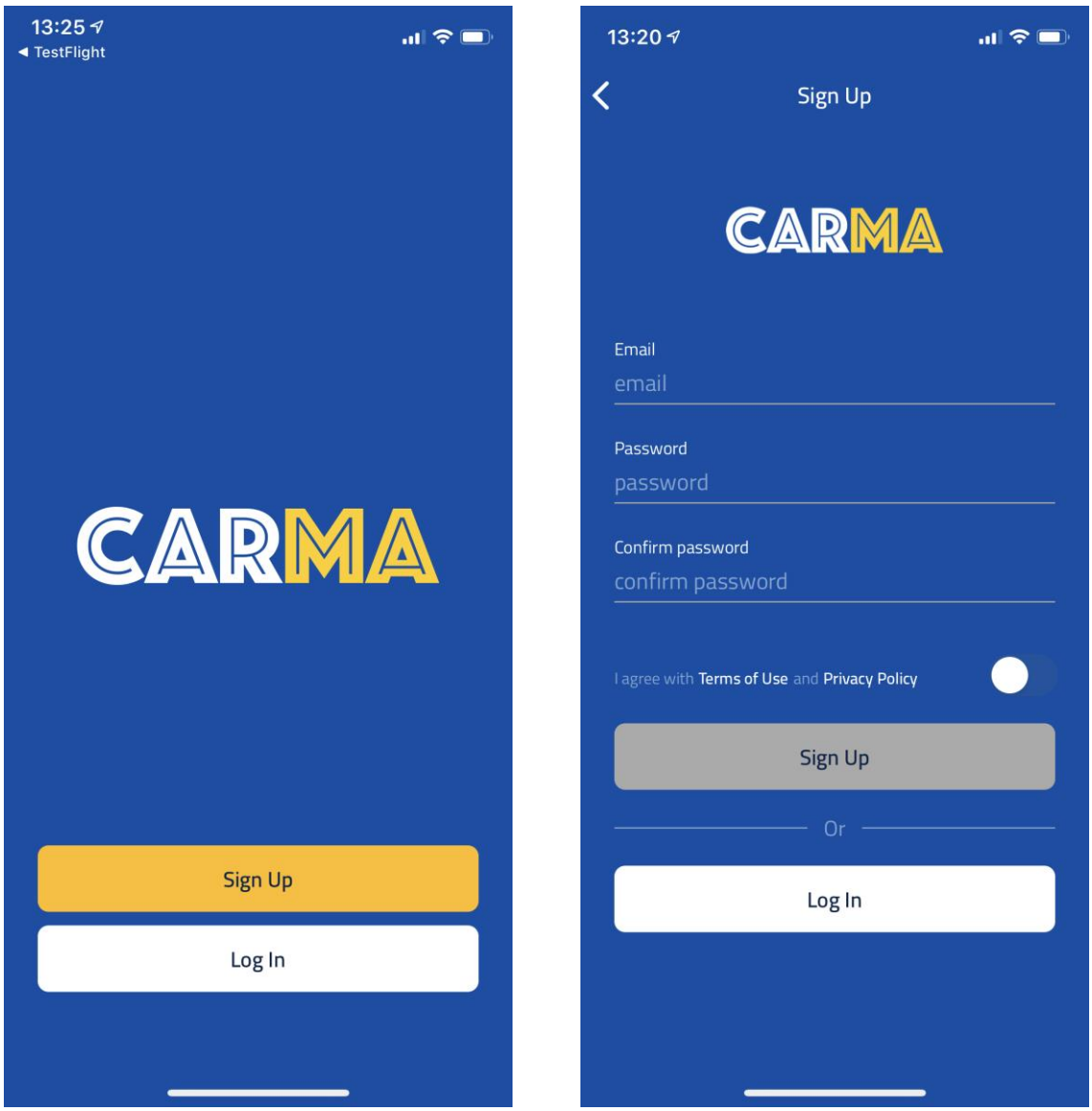
10. Для зв’язування View та ViewModel, буде використовуватися бібліотека RxSwift, що реалізує принципи реактивного програмування.

11. Для Carma була створена діаграма прецедентів, де показані основні можливості користувача.

РОЗДІЛ 3. ОГЛЯД ФУНКЦІОНАЛУ

3.1. Реєстрація користувача

Після першого запуску додатку, користувач бачить стартовий екран (Рис. 3.1.1) з якого він може перейти до реєстрації, або авторизуватися з існуючим



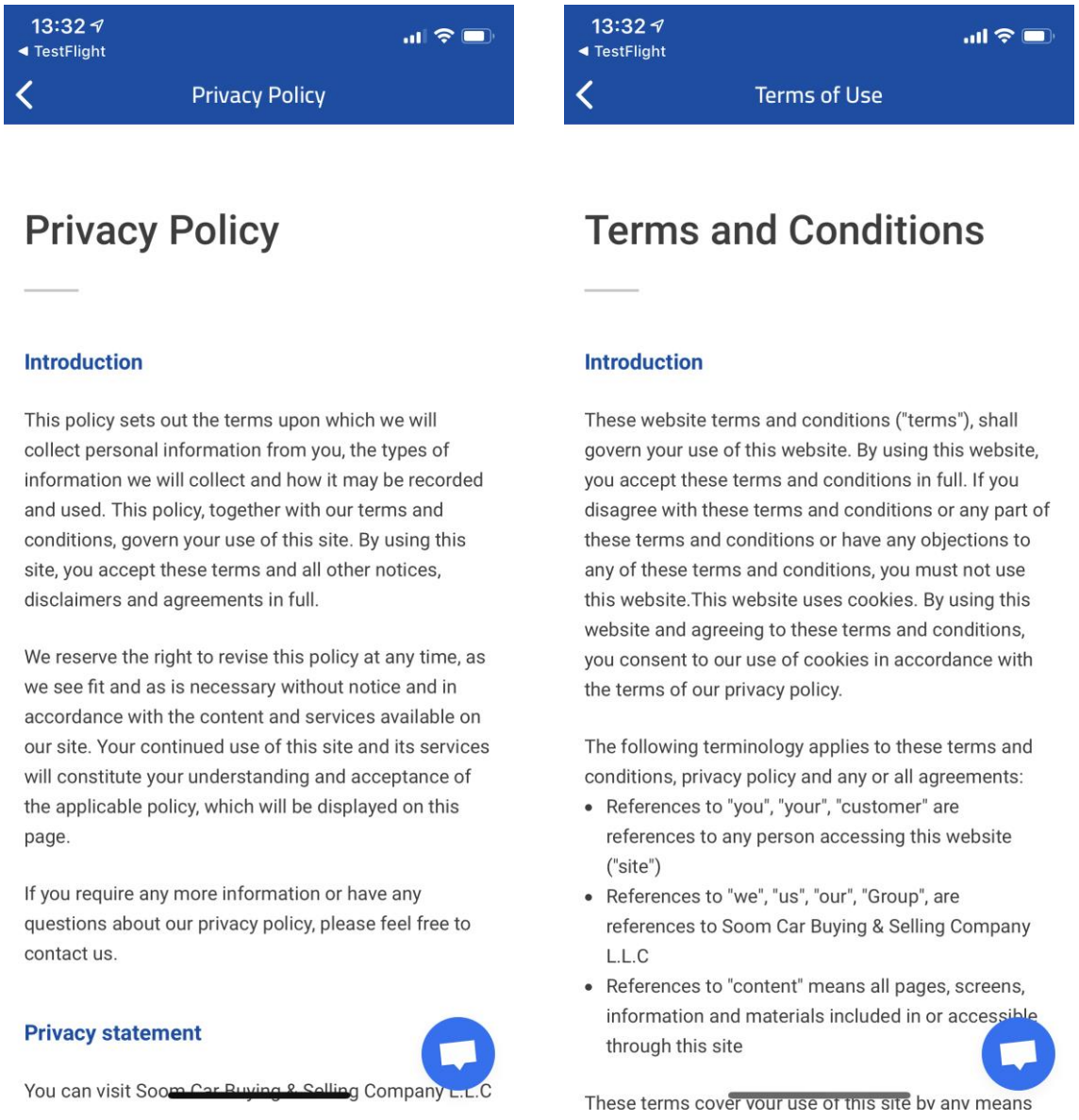
аккаунтом (Рис. 3.2.1).

Рис. 3.1.1. Демонстрація стартового екрану та екрану реєстрація

Для реєстрації користувача, додаток запитує наступні дані:

- Email користувача
- Пароль з його підтвердженням
- Згоду з умовами використання та політикою конфіденсійності.

Для реалізування екранів з умовами використання та політикою конфіденсійності було використано WebView, яка просто відображає контент web-сторінки (Рис. 3.1.2). Користувач може перейти на ці сторінки, просто натиснувши



на відповідний текст - “Terms of Use” або “Privacy Policy”.

Рис. 3.1.2. Демонстрація екрану з умовами використання та політикою конфіденсійності.

Користувач може підтвердити свою згоду з умовами використання та політикою конфіденсійності, змінивши положення відповідного перемикача, тільки у цьому випадку кнопка “Зареєструватися” стане активною.

Також на екрані реєстрації працює локальна валідація полів, що дозволить перевірити введені дані ще до відправки їх на сервер (рис. 3.1.3).

Рис. 3.1.3. Демонстрація відображення помилок при реєстрації

Після того, як усі дані будуть введені користувачем, можна продовжити

реєстрацію, натиснувши кнопку “Зареєструватися”.

Після натиснення кнопки “Зареєструватися”, система відправляє повідомлення на вказаний користувачем email з посиланням для підтвердження реєстрації. Додаток інформує користувача про це спеціальним повідомленням на екрані (Рис. 3.1.4.)

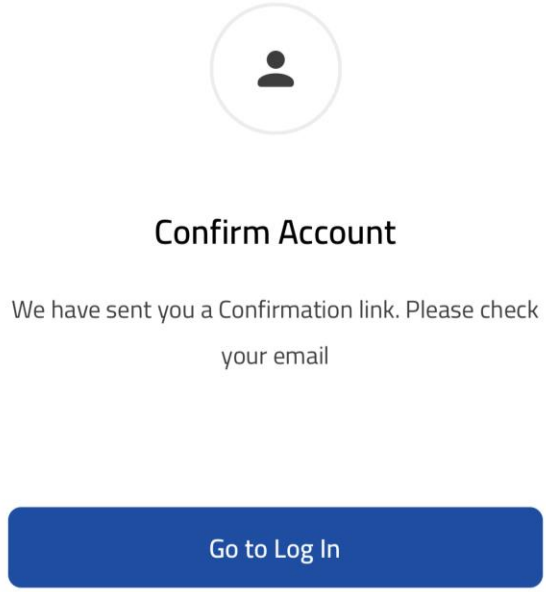


Рис. 3.1.4. Демонстрація екрану з повідомленням про перевірку емейлу

Натиснувши на посилання в емелі, який отримав користувач, відбувається підвердження реєстрації та відображення повідомлення про успішну реєстрацію на web-сторінці (Рис. 3.1.5.).

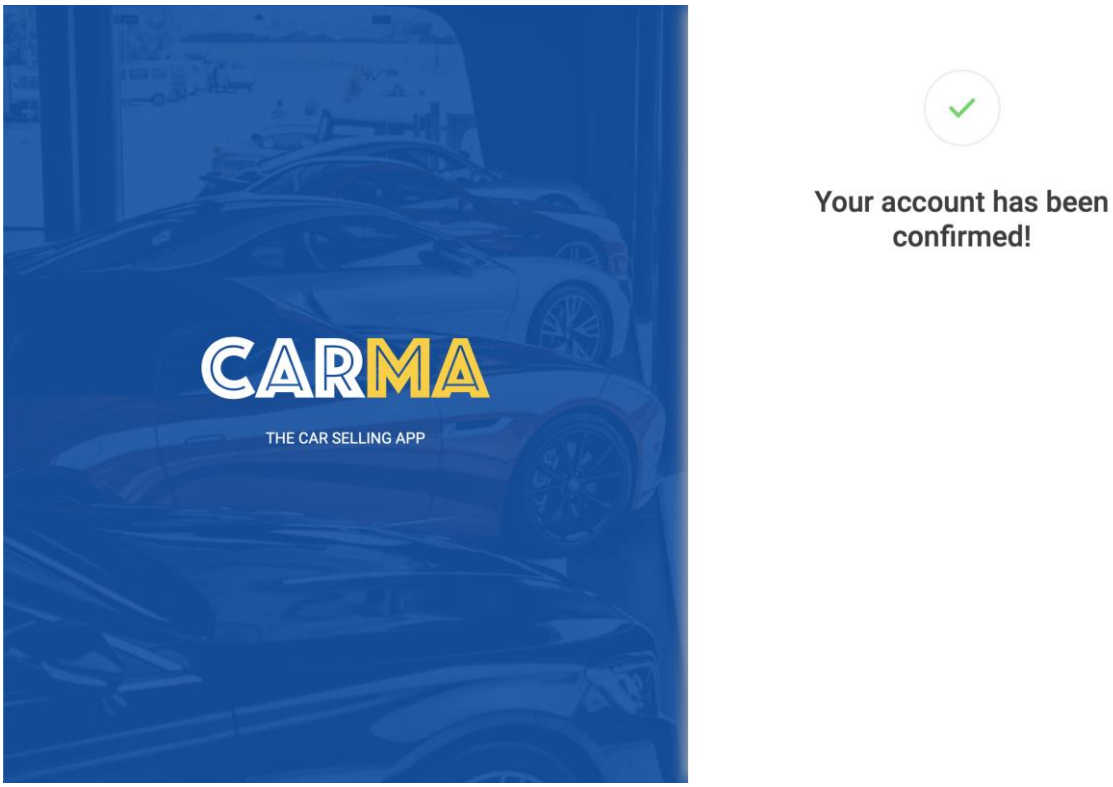
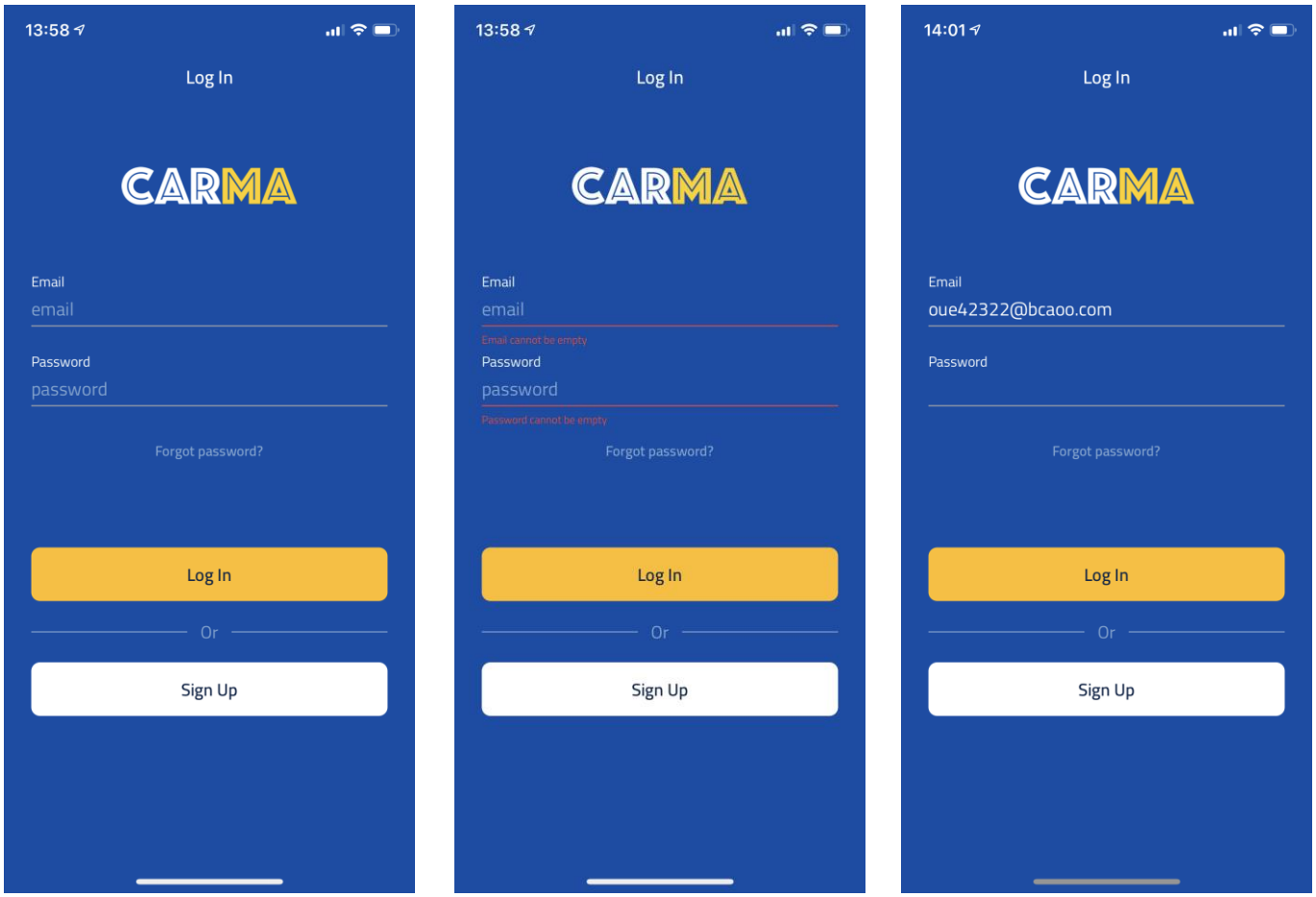


Рис. 3.1.5. Демонстрація web-сторінуи з повідомленням про успішну реєстрацію

На екрані з повідомленням про перевірку емейлу (Рис. 3.1.4.) користувач бачить кнопку “Перейти до входу”. По натисненню на цю кнопку, додаток відкриє екран, який використовується для авторизації користувача (Рис. 3.2.1.).

3.2. Авторизація користувача

На екрані авторизації користувач має ввести Email та пароль. На цьому екрані також працює локальна перевірка введених даних. Після вводу усіх необхідних даних, користувач має натиснути на кнопку “Увійти”, після чого система перевірить чи дійсно існує такий користувач та чи правильно введені дані для входу. У разі



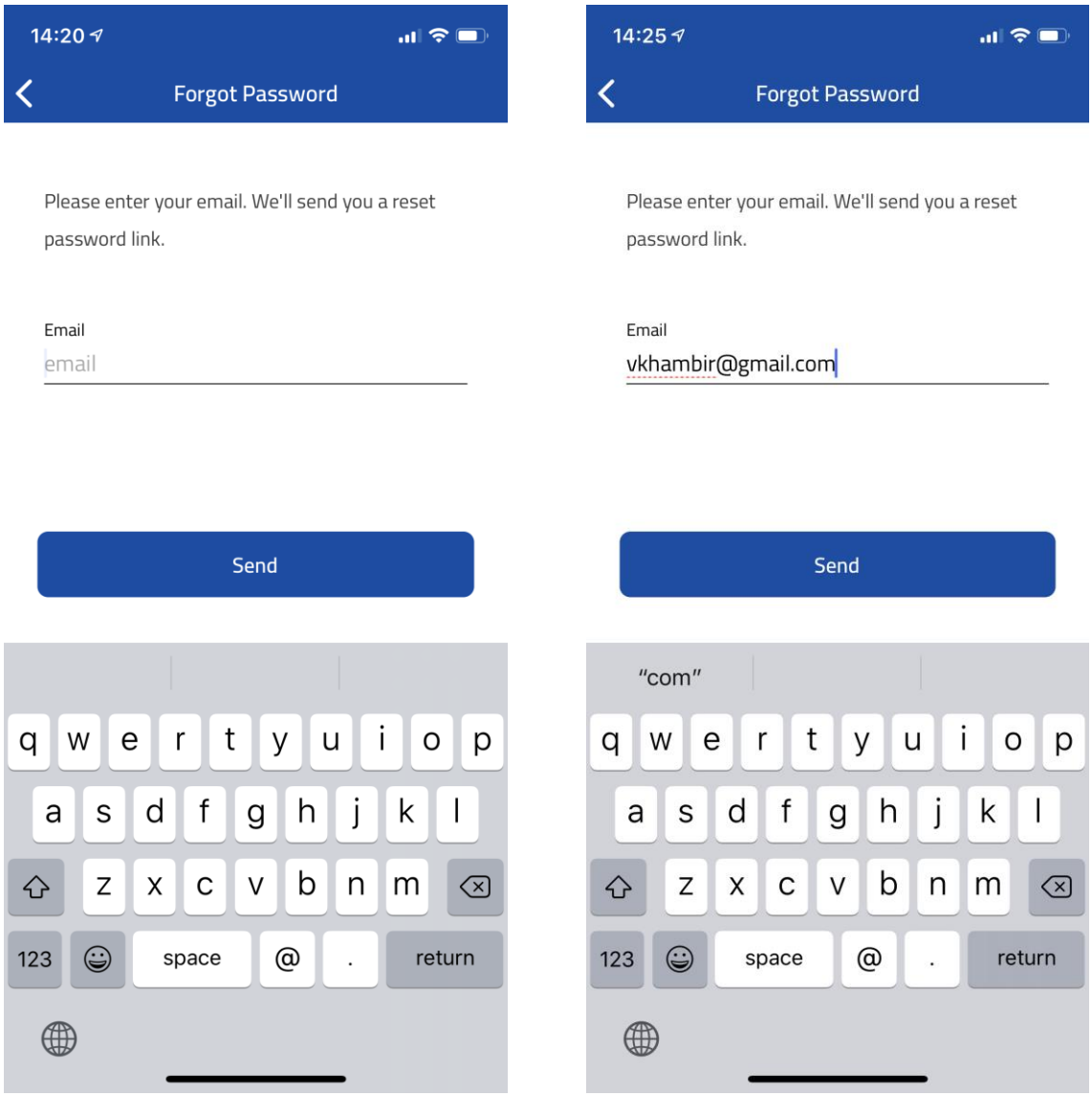
успіху усіх перевірок, користувач побачить головний екран додатку.

Рис. 3.2.1. Демонстрація екрану авторизації з його різними станами

Також з цього екрану користувач може перейти на екран реєстрації, натиснувши на кнопку “Реєстрація”, або у разі забутого паролю, користувач має змогу його відновити, натиснувши на кнопку “Забули пароль?”.

3.3. Відновлення паролю

У разі забутого паролю, користувач має змогу його відновити, натиснувши на кнопку “Забули пароль?” на екрані авторизації (Рис. 3.2.1.). Після натиснення на кнопку “Забули пароль?”, користувач буде перенаправленний на екран по

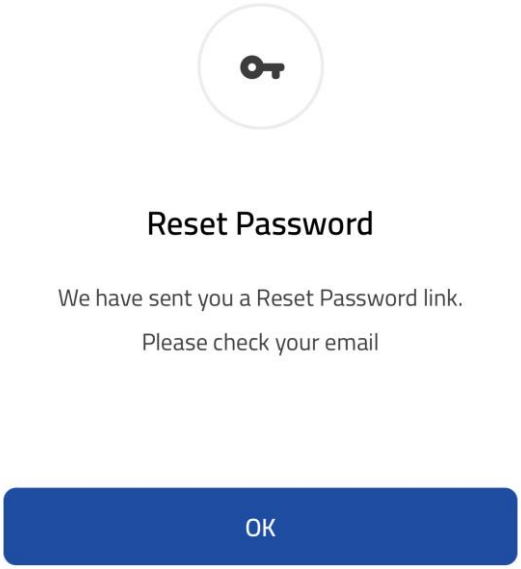


відновленню паролю.

На екрані по відновленню паролю, користувач має ввести свій email, та натиснути кнопку “Відправити” (Рис. 3.3.1.).

Рис. 3.3.1. Екран відновлення паролю

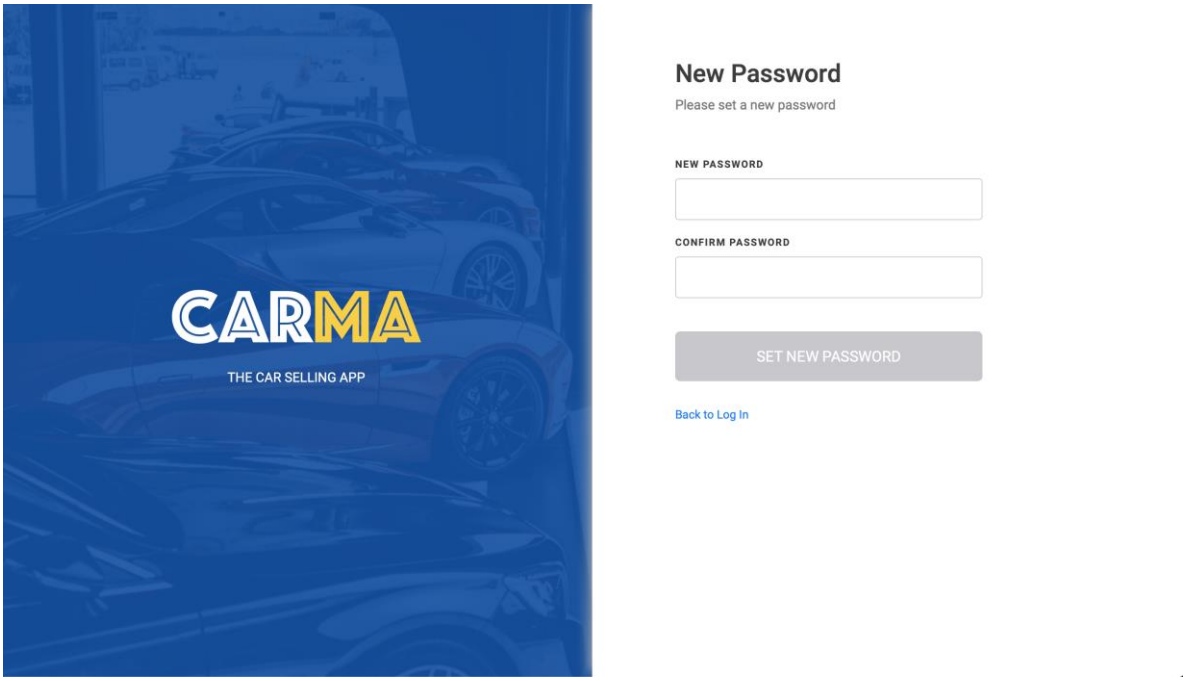
Після натиснення на кнопку “Відправити”, система відправляє лист з посиланням для відновлення паролю на вказаний користувачем емейл. Про це



користувача інформує екран з інформацією про відновлення паролю (Рис. 3.3.2.)

Рис. 3.3.2. Екран з інформацією про відновлення паролю

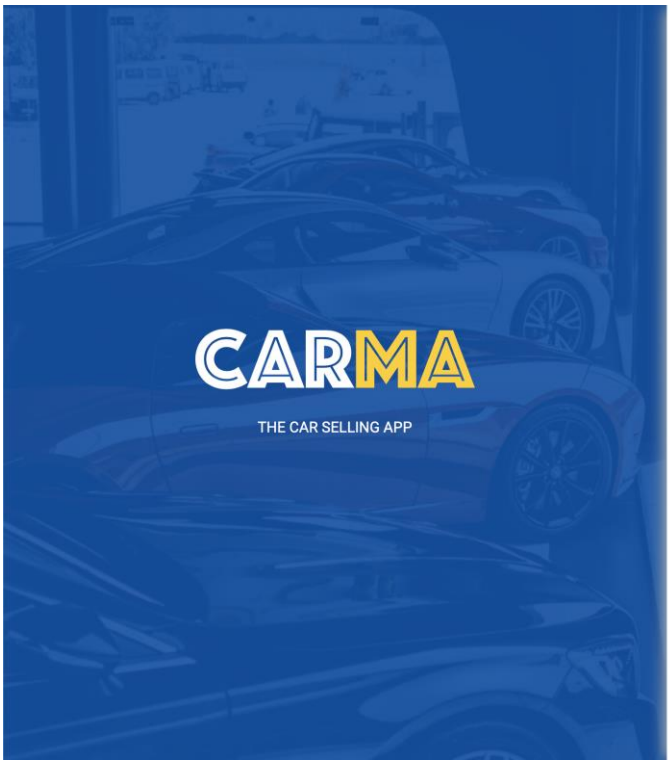
На емейл користувача відправляється лист з інструкціями по відновленню пароля. Коли користувач перейде за посиланням, указанним у листі, відкриється

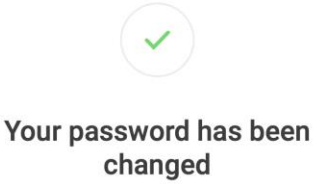


web-сторінка, на якій потрібно буде ввести новий пароль та підтвердити його.

Рис. 3.3.3. Web-сторінка для відновлення паролю користувача

Після вводу нового паролю та його підтвердження, кнопка “Встановити новий пароль” стане активною. Після натиснення на кнопку “Встановити новий пароль”, система спробує змінити старий пароль користувача на новий і у разі успіху, користувач буде повідомлений про це шляхом відображення web-сторінки з





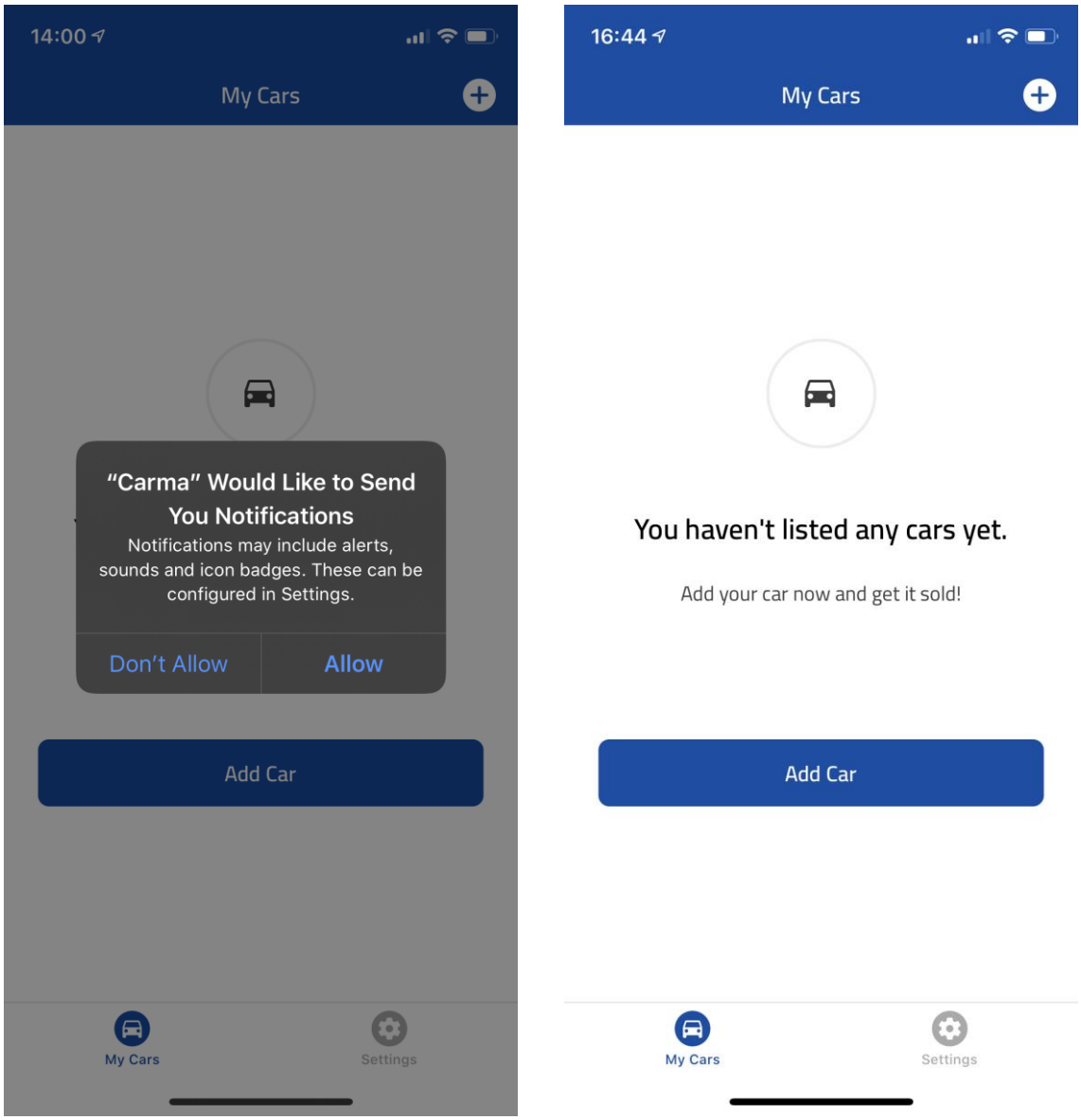
відповідним повідомленням (Рис. 3.3.4).

Рис. 3.3.4. Web-сторінка з повідомленням про успішну зміну паролю

Після успішної зміни паролю, користувач може натистути на кнопку “Ok” на екрані з інформацією про відновлення паролю (Рис. 3.3.2.) та перейти на екран авторизації (Рис. 3.2.1.) та увіти до свого аккаунту, використовуючи пароль, який щойно був встановленний.

3.4. Головний екран

Після успішної авторизації, користувач потрапляє на головний екран додатку (Рис. 3.4.1). Головний екран представляє собою Tab Bar Controller, який складається



з двох вкладок: “Мої автомобілі” та “Налаштування” (Рис. 3.5.1).

Рис. 3.4.1. Головний екран додатку

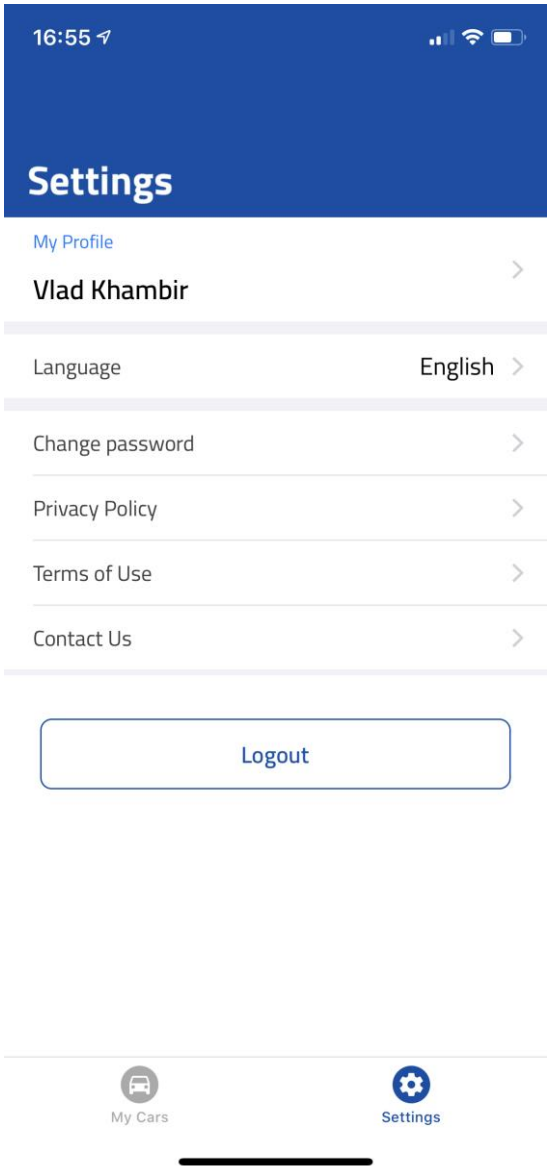
Якщо додатор був запущений вперше, то додаток запитає дозвіл у користувача на відправлення нотіфікацій. Дозвіл запитується через системний Alert, який має два

варіанти відповіді: дозволити та заборонити дозвіл на відправлення push-повідомлень.

3.5. Налаштування

Друга вкладка у додатку веде користувача на екран з налаштуваннями (Рис. 3.5.1). Додаток містить такі пункти налаштувань:

- Профиль - користувач може налаштовувати свій профіль, змінюючи в ньому ім’я та номер телефону.
- Мова - користувач може змінювати мову додатку на англійську та арабську.
- Зміна паролю - користувач може змінювати свій пароль.



- Вихід - через кнопку “Вихід” користувач може вийти з свого аккаунту.

Рис. 3.5.1. Екран налаштувань

Якщо користувач натисне на пункт зі своїм профілем, він побачить екран з деталями свого профілю (Рис. 3.5.2).

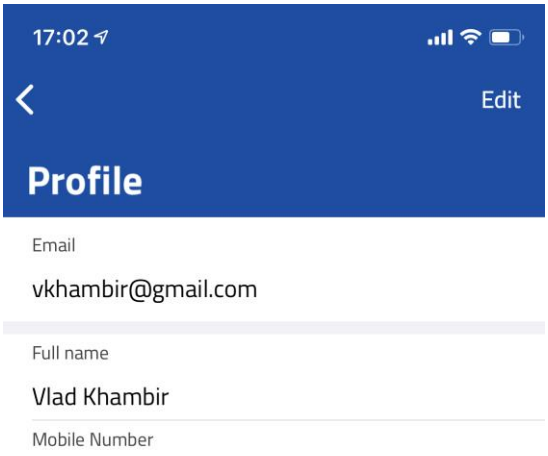
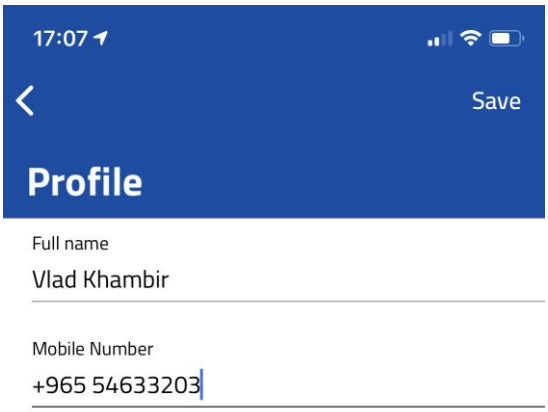


Рис. 3.5.2. Екран деталей профілю користувача

Серед деталей профілю можна побачити:

- Email користувача
- Повне ім’я користувача
- Номер телефону користувача

Усі вище перелічені дані, окрім Email, можна редагувати, настиснувши на кнопку “Редагувати”. По натисненню на цю кнопку, користувач перейде у режим

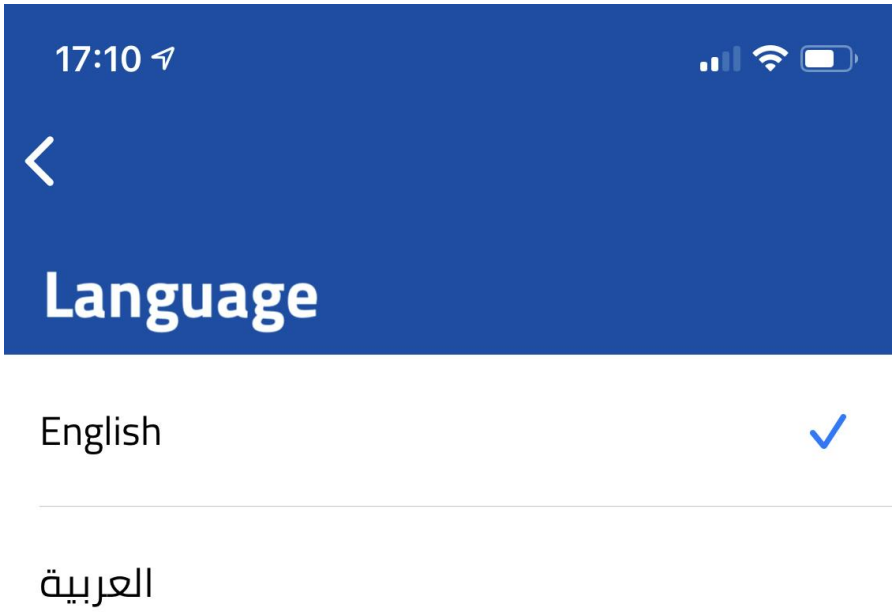


редагування своїх даних (Рис. 3.5.3).

Рис. 3.5.3. Екран редагування даних профіля користувача

На екрані редагування даних профіля користувач може змінити певні дані да зберегти їх, натиснувши на кнопку “Зберегти”. Після успішного збереження, користувач потрапить на екран з деталями свого профіля, який вже буде оновлено.

На екрані налаштувань, можна перейти до налаштувань мови додатку. Користувач побачить перелік мов, які підтримує додаток і може обрати зручну для



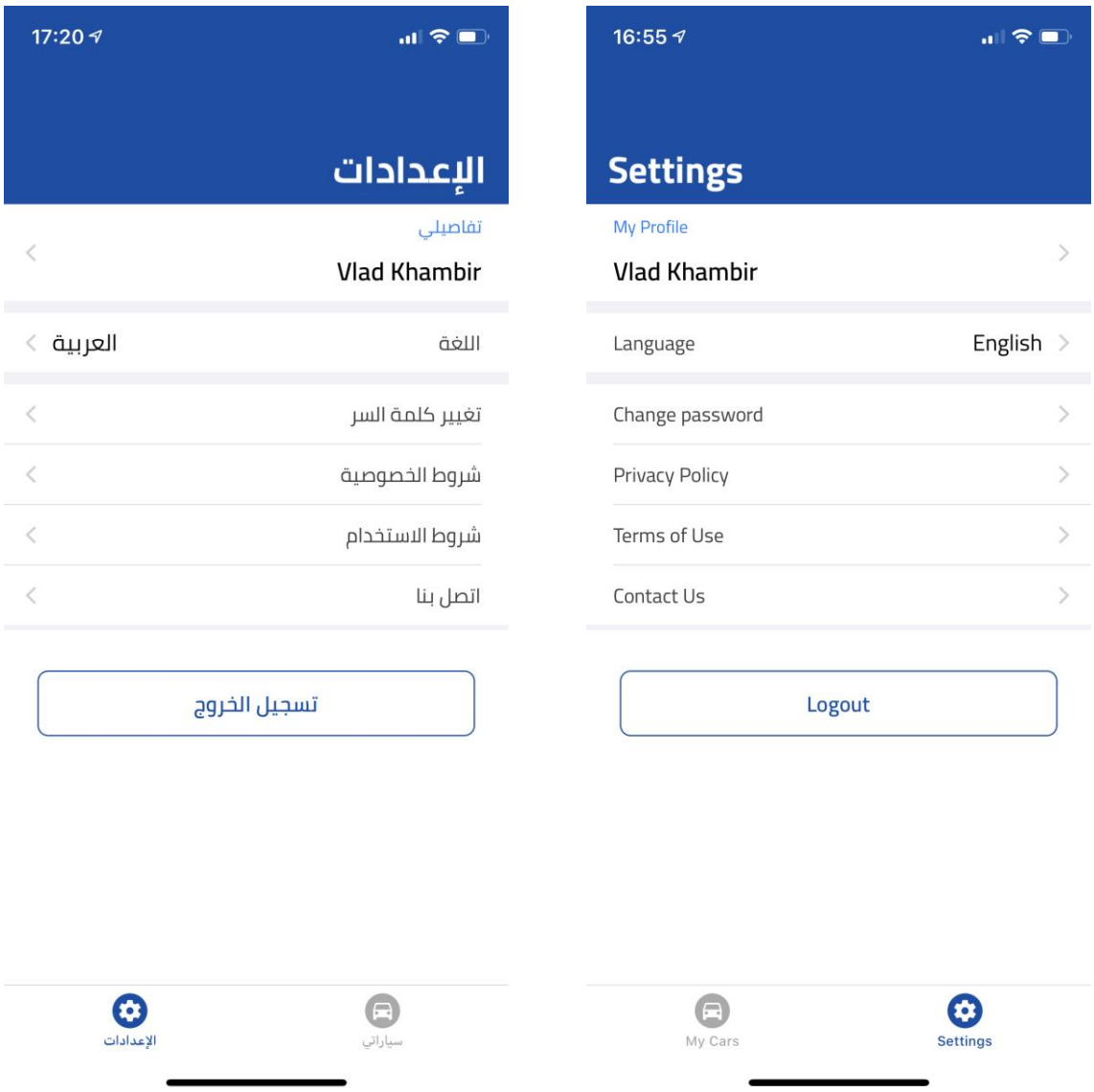
нього мову шляхом натиснення на бажану мову (Рис. 3.5.4).

Рис. 3.5.4. Екран налаштування мови додатку

Додаток підримує дві мови: англійську та арабську. Також додаток пидтримує два відповідних відображення інтерфейсу: RTL (Right to left) та LTR (Left to right), тобто при виборі арабскої мови, увесь UI буде перевернуто з права на ліво, включаючи анімації, методи вводу та інше, але залешаючи при цьому орігінальне відображення фотографій (Рис. 3.5.5.).

З екрану налаштувань користувач може змінити свій пароль. Для цього у меню налаштувань він має обрати пункт “Змінити пароль”. Після вибору цього пункту, додаток покаже користувачу екран для зміни паролю (Рис. 3.5.6.).

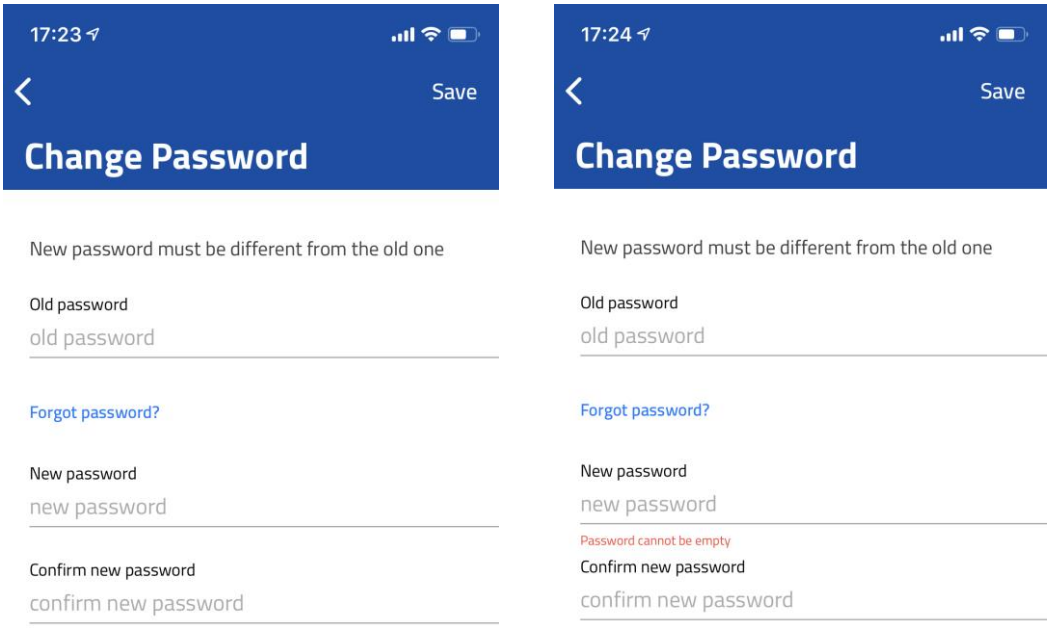
Рис. 3.5.5. Демонстрація RTL та LTR інтерфейсів



Для зміни паролю, на цьому екрані користувач має ввести свій старий пароль, новий та підтвердити новий пароль. Після цього натиснути на кнопку “Зберегти”. Усі введені дані валідуються локально та потім на стороні сервера.

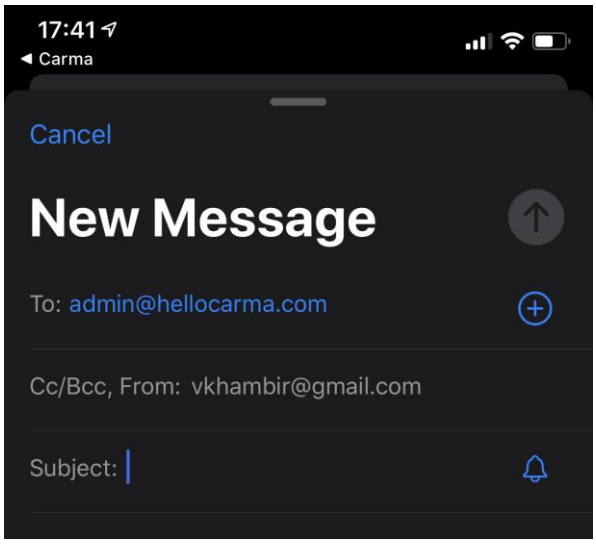
У разі, якщо користувач забув свій пароль, то з цього екрану він має змогу його відновити, натиснувши на кнопку “Забули пароль?”.

Рис. 3.5.6. Екран зміни паролю



Після успішної валідації та зміни паролю, система завершить усі існуючі сессії користувача і як наслідок, поточна сессія такочж завершиться і користувач побачить екран авторизації з повідомленням про те, що його пароль успішно змінено і він може авторизуватися ще раз, використовуючи вже новий пароль, який щойно був встановлений.

Рис. 3.5.7. Шаблон листа для андміністратора



Через екран налаштувань, користувач може зв’язатися з адміном, обравши пунк “Зв’язатися з нами”, після чого буде відкрито додаток Mail з готовим шаблоном написання повідомлення на адресу адміністратора (Рис. 3.5.7.).

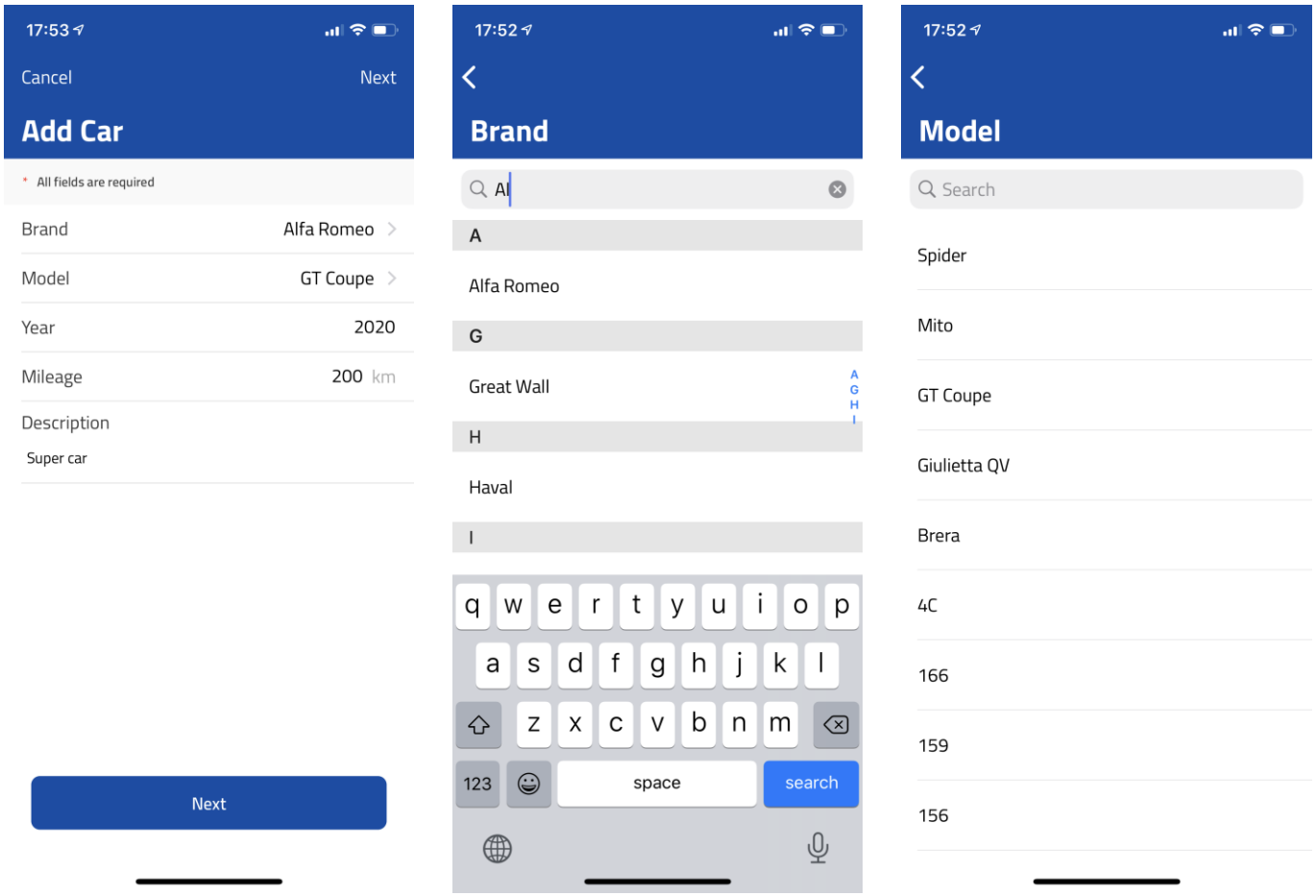
3.6. Додавання авто

Процес додавання авто складається з трьох частин:

1. Заповнення основних даних автомобіля: бренд, модель, рік випуску, пробіг та додатковий опис (Рис. 3.6.1).
2. Заповнення додаткових даних автомобіля: специфікація, наявність гарантії, сервіс, інформація про аварії та заміни деталей (Рис. 3.6.2)
3. Додавання фотографій автомобіля.

Рис. 3.6.1. Перший етап додавання автомобіля

На першому етапі додавання автомобіля користувачу потрібно обрати бренд автомобілю. Для цього додаток відкриє екран зі списком усіх доступних брендів. На



цьому екрані є пошук по імені бренду, завдяки чому користувач може швидко знайти потрібний йому варіант. Подібний екран був також створенний для пошуку потрібної моделі.

Після заповнення усіх параметрів, що доступні на першому кроці додавання авто, кнопка “Далі” стає активною і користувач може перейти до наступного кроку.

На другому етапі додавання автомобіля, користувач має зазначити специфікацію автомобіля, наявність гарантії та вказати інформація про аварії, якщо такі були. Після вводу усіх даних, користувач може переходити до останнього кроку додавання автомобіля, натиснувши на кнопку “Далі”

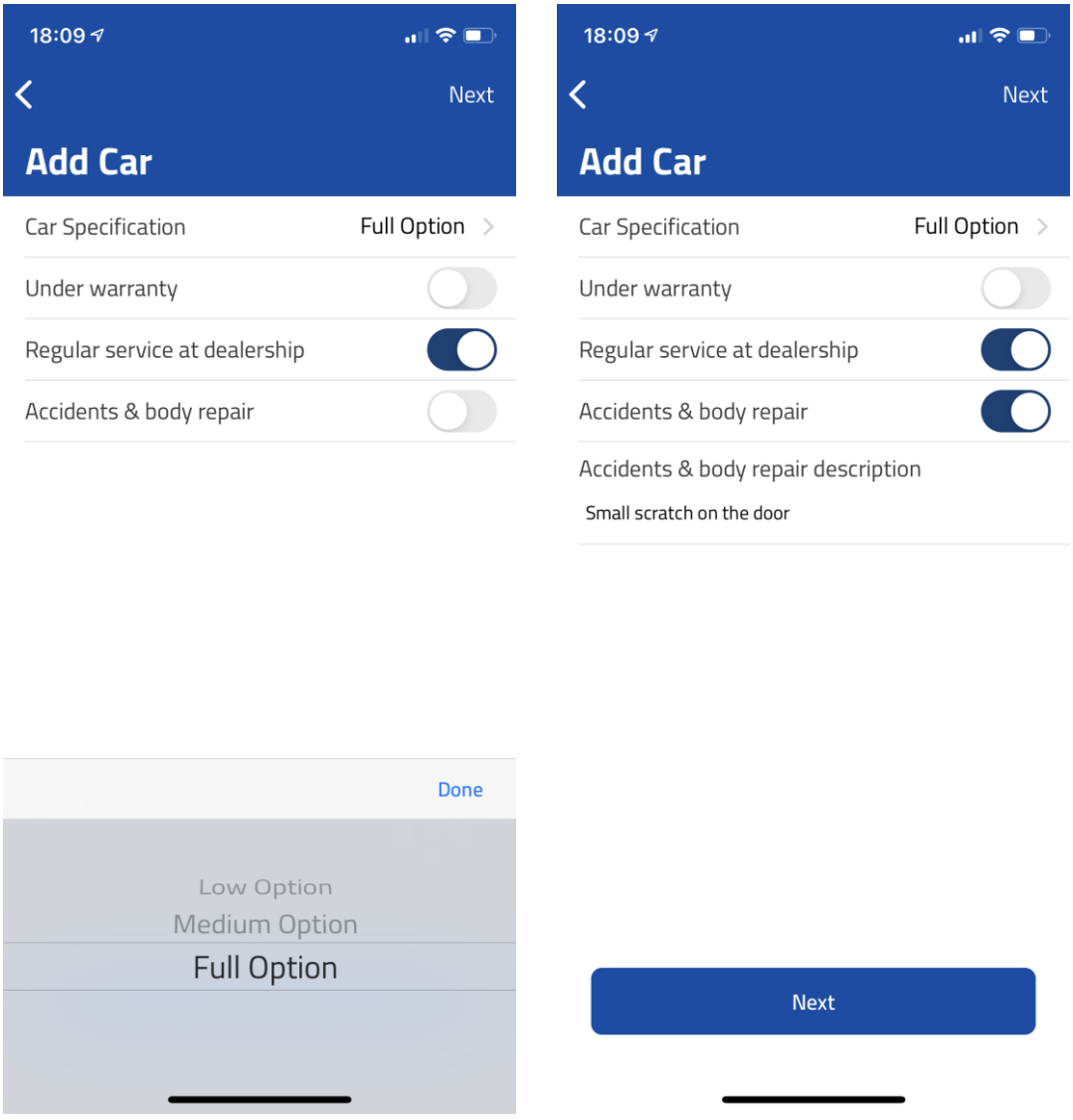


Рис. 3.6.2. Другий етап додавання автомобіля

Третій етап додавання автомобіля є останнім. На цьому етапі, користувач має додати фотографії свого авто. Користувач має додати мінімум одну фотографію та максимум шість.

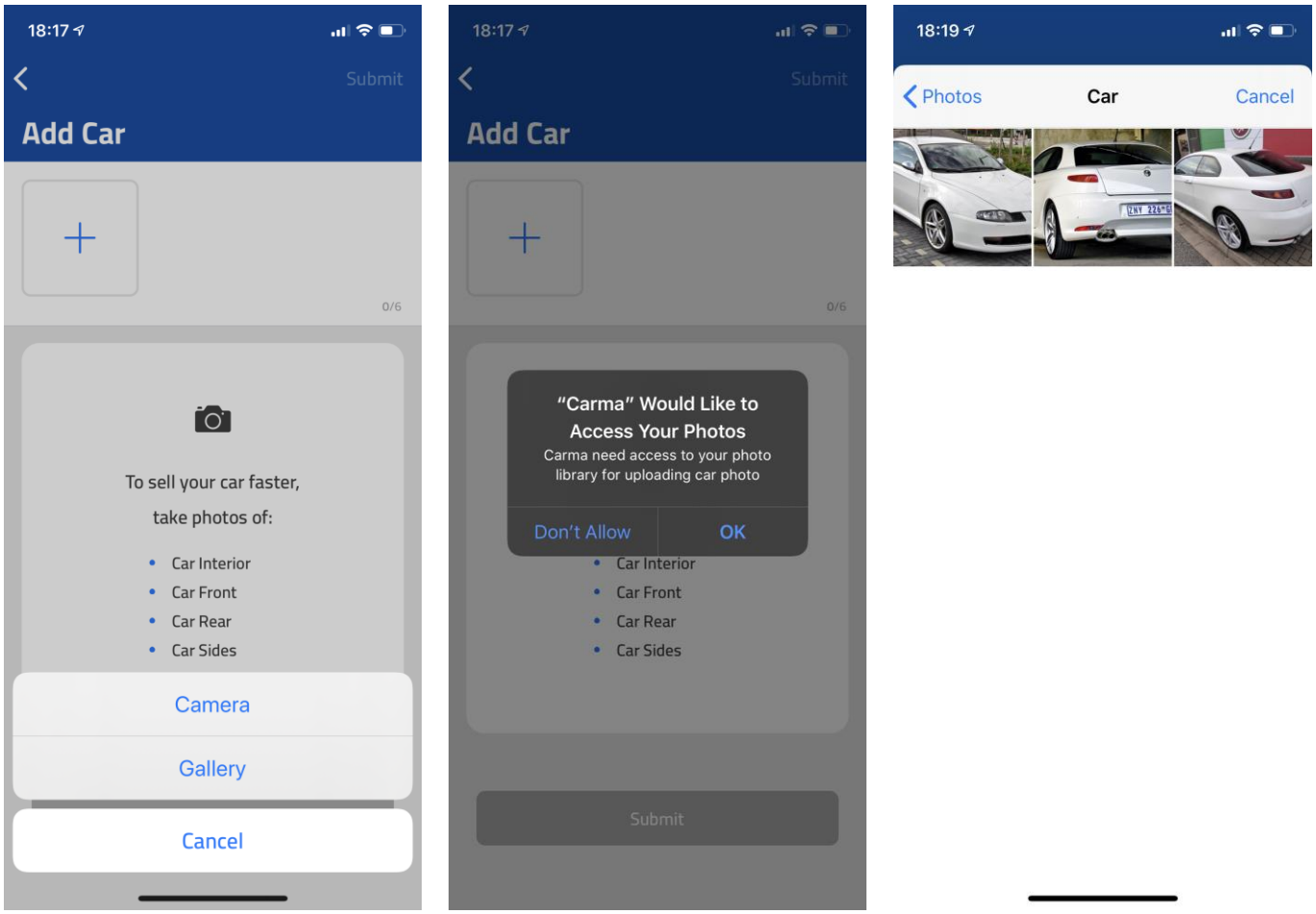


Рис. 3.6.3. Демонстрація додавання авто

Фотографії можна додавати, використовуючи камеру чи галерею. У обидвох випадках, iOS спитає користувача про дозвіл для доступу до камери чи галереї (Рис. 3.6.3.).

Після вибору фотографії, вона відправляється на завантаження. Усі додані фотографії можна видалити при потребі, натиснувши на кнопку “X”, що знаходиться над кожною фотографією.

На екрані останнього етапу додавання авто зазначена рекомендація щодо додавання фотографій та є кнопка “Відправити”, яка стає активною після завантаження фотографій (Рис. 3.6.4.).

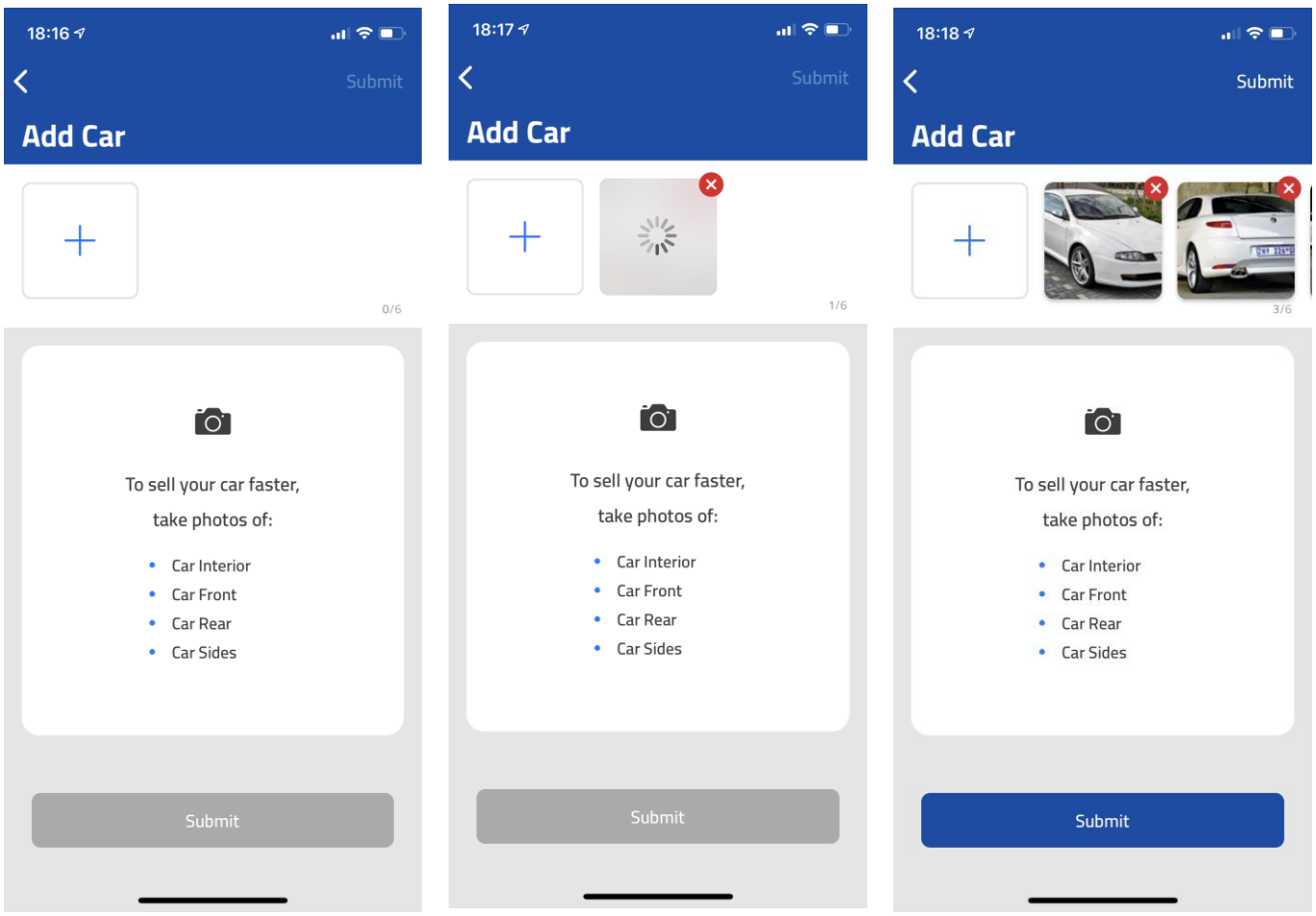
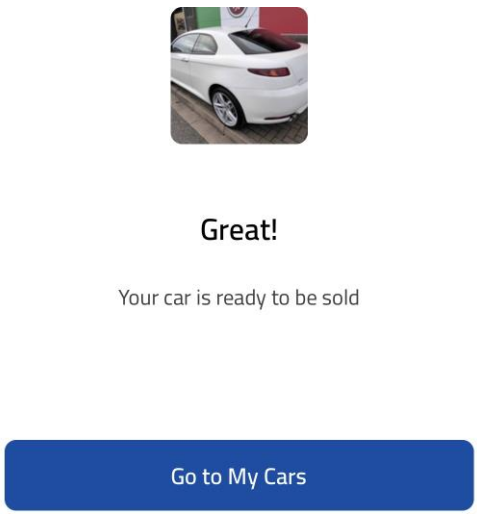


Рис. 3.6.4. Третій крок додавання автомобіля

Після натискання кнопки “Відправити” система додасть автомобіль користувача у базу автомобілів для продажу і його побачать усі діллери. Також користувач буде повідомлений про те, що його автомобіль успішно додано та він



готовий для продажу (Рис. 3.6.5).

Рис. 3.6.5. Екран з повідомленням про успішно додане авто

3.7. Перелік автомобілів

На першій вкладці додатку, користувач може переглядати перелік доданих автомобілів. Користувач може бачити базову інформацію про автомобіль, а саме:

- Марку та бренд авто
- Пробіг
- Короткий опис
- Головне фото авто та загалька кількість фотографій
- Кількість пропозицій

Власник авто може видалити свій автомобіль, використавши жест змахування по автомобілю, після чого стане доступна опція видалення. Для видалення автомобіля, користувачу необхідно підтвердити свої дії через додаткове вікно.

Також користувач бачить інформацію про те, що для додавання нового авто йому необхідно видалити теперішнє, або відмітити його як продане (Рис. 3.7.1.).

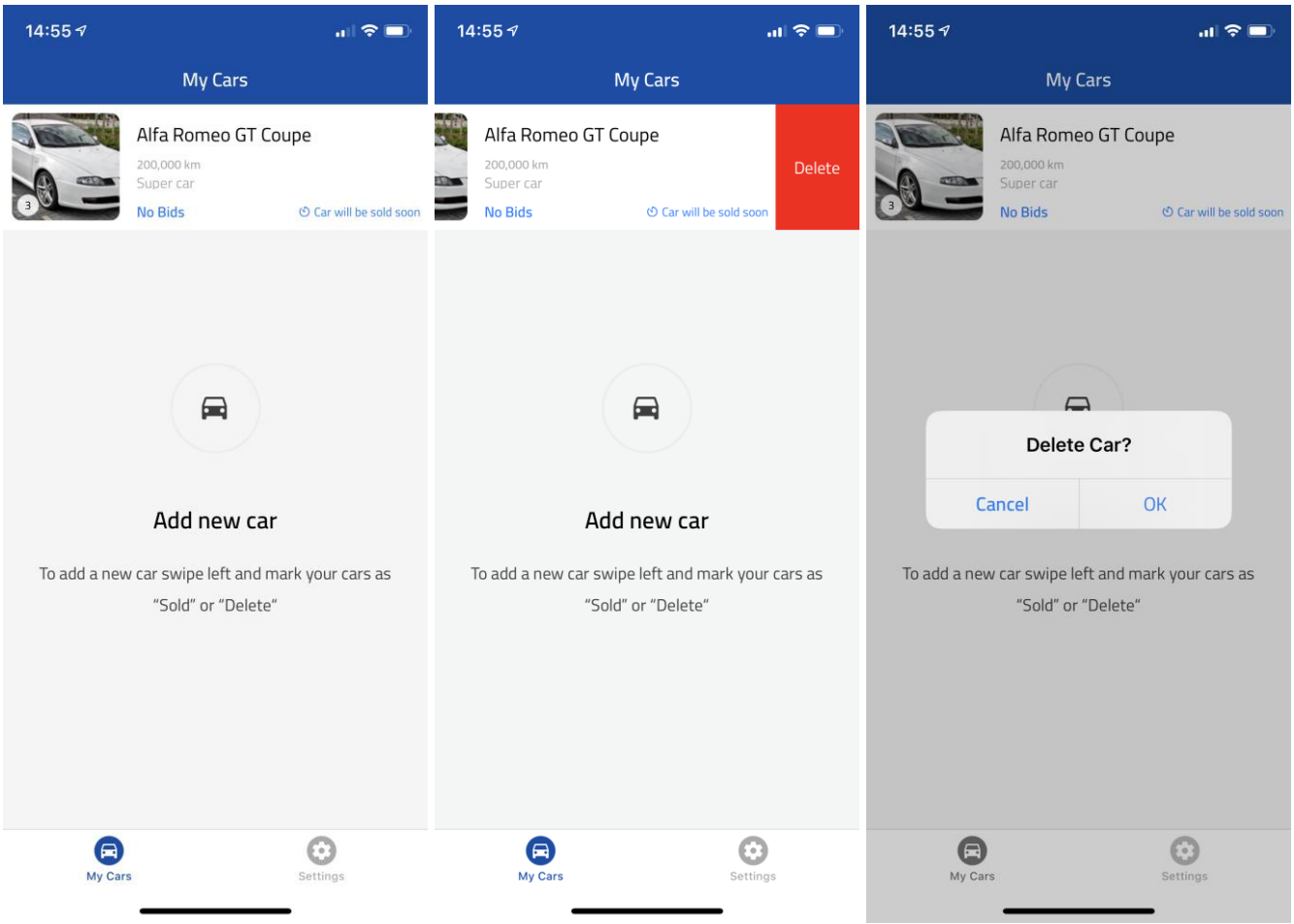
Рис. 3.7.1. Перелік автомобілів та можливість видалення

З цього екрану користувач може потрапити на екран з деталями автомобіля, для цього користувачу необхідно натиснути на автомобіль.

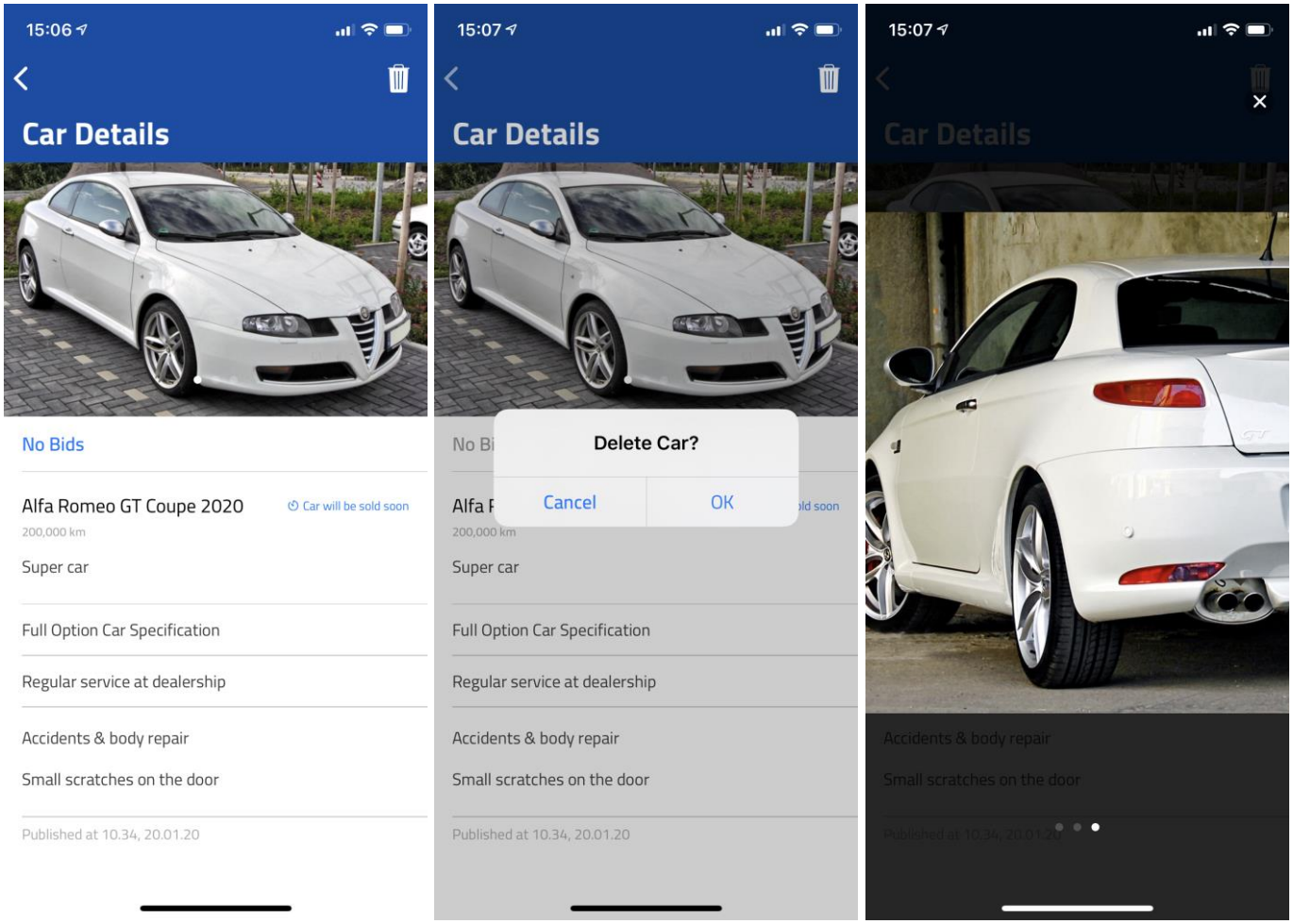
3.8. Деталі автомобіля

На екрані з деталями автомобіля користувач може переглянути усі деталі раніше доданного авто (Рис. 3.8.1.). Серед деталей можна побачити:

- Кількість пропозицій
- Фотографії авто
- Бренд та модель авто
- Опис авто



• Наявність гарантії



• Навність аварій та/або замін певних деталей

Рис. 3.8.1. Екран з деталями автомобіля

З екрану деталей автомобіля також можна його видалити, натиснувши відповідну кнопку для видалення, що знаходиться у барі навігації у правій частині. Так само як і на екрані з переліком автомобілів, для видалення авто потрібно підтвердити видалення у додаткованному вікні.

Фотографії автомобіля вміщенні у спеціальну галерею, що дозволяє переглядати усі фотографії, просто листаючи їх вправо та вліво. Галерея має “точки навігації”, які дозволяють побачити загальну кількість фотографій та зореєнтуватися на якій саме фотографії зараз знаходиться користувач. Під час завантаження фотографій, користувач бачить Loading spinner.

Для більш швиткого завантаження фотографій, кожне фото зберігається у трьох форматах:

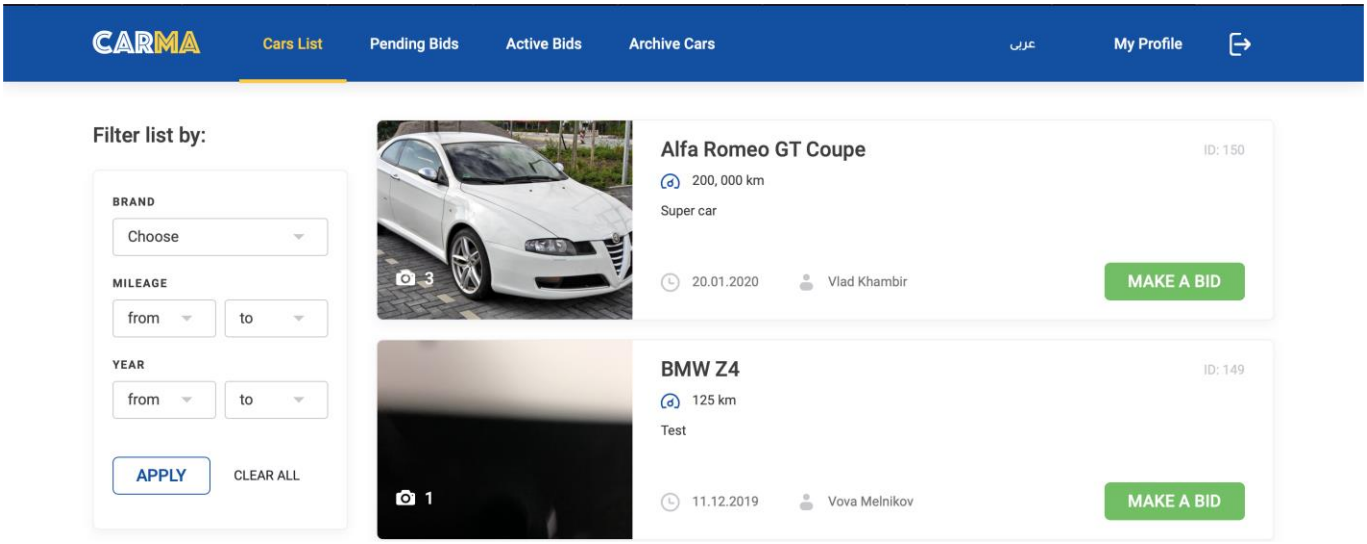
- Thumbnail - використовується для відображення фотографій у малому форматі, наприклад на екрані з переліком авто
- Medium - використовується для відображення фотографій у середньому форматі з достатньою якістю для більш детального ознайомлення, але цей формат не підходить для перегляду фотографій з можливістю збільшення
- Original - орігінальний розмір фотографії, що дозволяє користувачу збільшувати фото та роздивитися дрібні деталі

Спеціально для Original фотографій створена окрема галерея, на яку користувач може перейти, просто натиснувши на будь-яку з фотографій. На цій галереї також є “точки навігації”, а ще користувач має змогу використовувати жест “Подвійне натискання” та/або “Pinch to Zoom” для збільшення картинки, зо дозволить розглянути усі необхідні деталі авто. Закрити цю галерею користувач може, натиснувши відповідну кнопку “X”, що знаходиться у верхньому правому кутку галереї.

Якщо користувач має пропозиції від покупців, то він побачить кількість цих пропозицій на екрані деталей. Якщо є пропозиції, то користувач може перейти на екран перегляду пропозицій, натиснувши на поле з кількістю пропозицій. Якщо пропозицій немає, то перейти до їх перегляду неможливо і користувач буде бачити текст “Немає пропозицій” на відповідному полі на екрані деталей автомобіля.

3.9. Перелік пропозицій

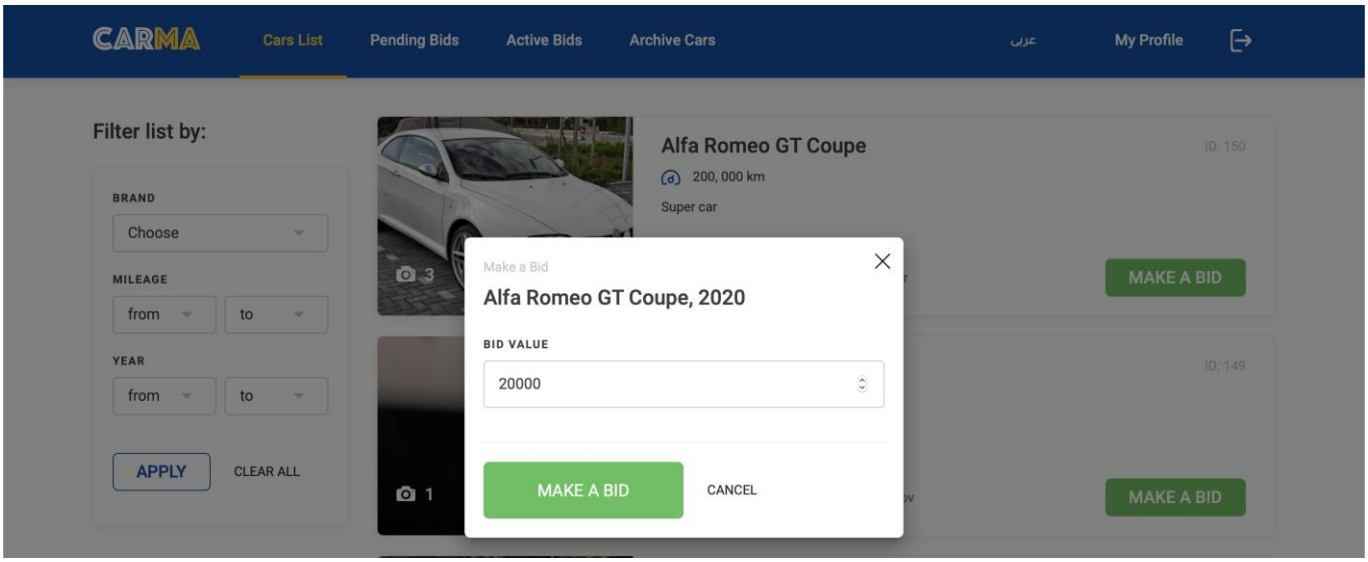
Покупці автомобілів мають свій Web-додаток, через який вони можуть



переглядати автомобілі, що знаходяться на продажі (Рис. 3.9.1).

Рис. 3.9.1. Web-додаток для покупців

Покупці мають змогу фільтрувати список автомобілів, переглядати їх деталі та

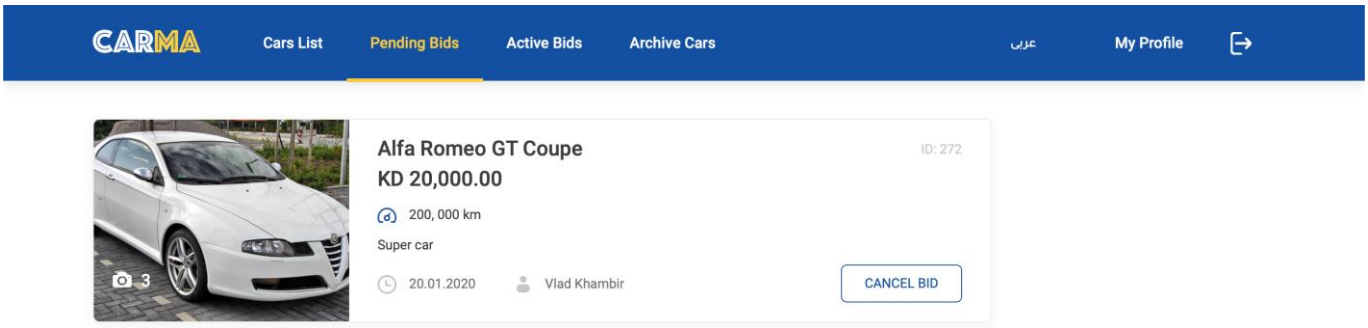


робити пропозиції (Рис. 3.9.2.).

Рис. 3.9.2 Екран створення пропозиції

Створена пропозиція відправляється до продавця. Про нову пропозицію продавець може дізнатися, завдяки Push-повідомленню, яке сповістить користувача про наявність нової пропозиції.

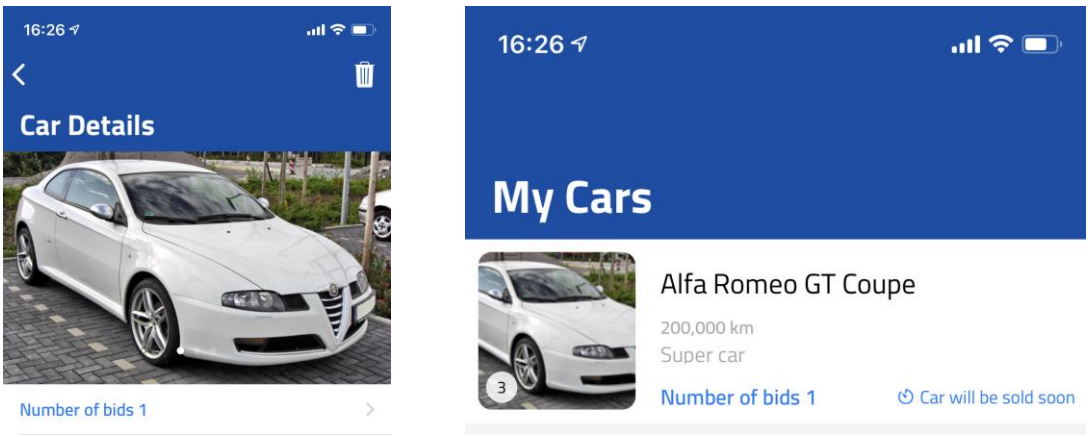
Після того як покупець зробив пропозицію, вибране авто не потрапляє більше у перелів автомобілів для покупки, а від тепер видображається у переліку зроблених



пропозицій (Рис. 3.9.3)

Рис. 3.9.3. Екран переліку зроблених пропозицій, що чекають підтвердження.

Отримана продавцем пропозиція потрапляє у перелік активних пропозицій, користувач може дізнатися про це через Push-повідомлення, або через UI, що вказує

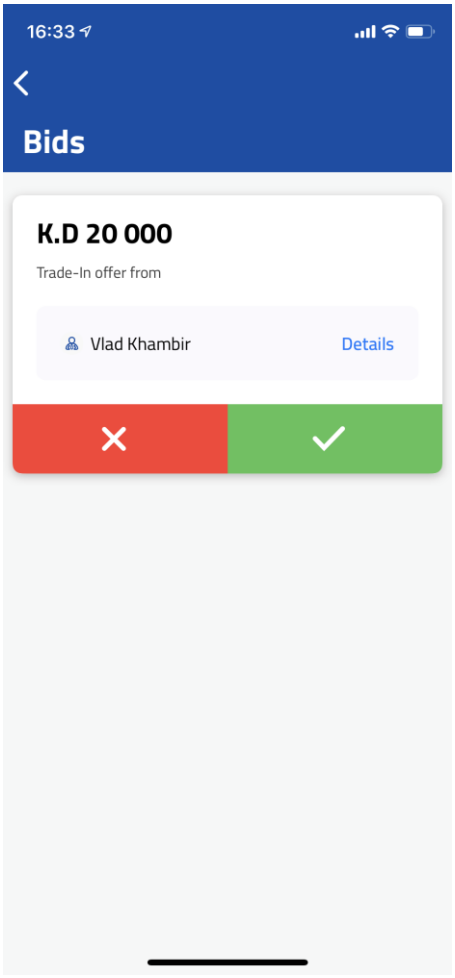


на те, що кількість попозицій - одна Рис. (3.9.4).

Рис. 3.9.4. Відображення кількості пропозицій

Усі пропозиції користувач може переглянути на відповідному екрані - “Перелік пропозицій” (Рис. 3.9.5). На цьому екрані кожна пропозиція

відображається як окрема карточка. На кожній карточці користувач може

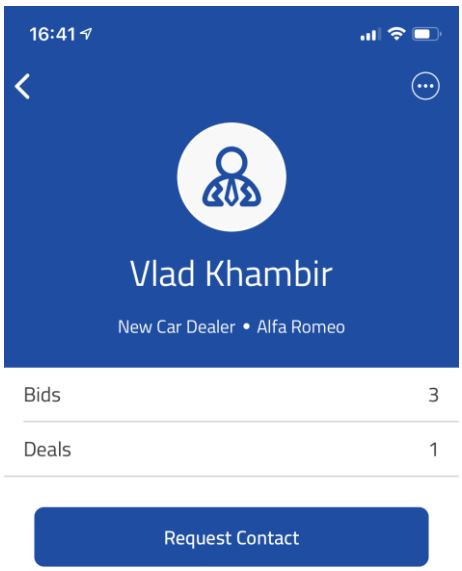


переглянути усю необхідну інформацію пропозиції, а також може прийняти цю пропозицію до розгляду, або відклонити її.

Серед відкритої інформації попозиції є: Ціна, ім’я та фотографія покупця. Користувач має змогу подивитися інформацію про продавця більш детально, натиснувши на кнопку “Деталі” у карточці пропозиції. Після чого користувача буде перенаправлено на екран з деталями покупця (Рис. 3.9.6.).

Рис. 3.9.5. Екран з переліком нових пропозицій

Серед деталей покупця можна побачити кількість завершених угод та загальну кількість пропозицій. Більш детальну інформацію можна отримати, натиснувши на кнопку “Запросити контакти”. У цьому випадку користувач підтверджує пропозицію, виявиши зацікавленість у контактах покупця. Отримані контакти



покупця, користувач може використовувати для зв’язку з покупцем та подальших домовленностей по продажу автомобіля.

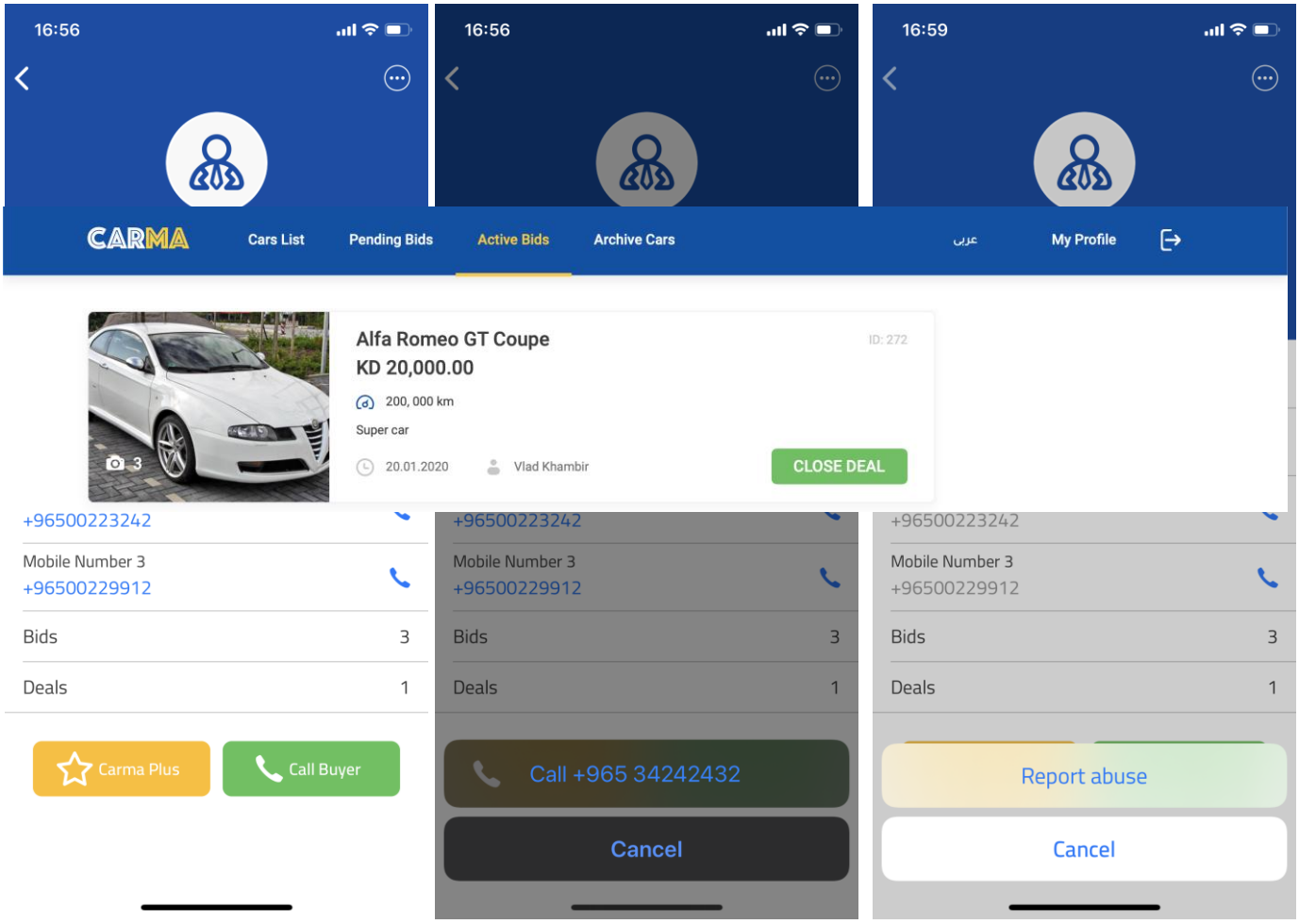
Серед відкритих контактів покупця, користувач може побачити декілька номерів телефонів та емейл покупця. По натисненню на будь-який номер телефону, користувач побачить системне повідомлення з варіантами “Зателефонувати” та “Відмінити”.

Рис. 3.9.6. Профіль покупця до запиту контактів

Також користувач може поскаржитися на покупця, наприклад, якщо користувач вказав невірний номер телефону чи емейл, або якщо пропозиція є занадто малою і відноситься до категорії спаму (Рис. 3.9.7.). Також причиною для скарження може бути будь-яка інша причина, яку користувач може описати на відповідному екрані. Усі скаргі на користувачів відправляються до адміністратора, який може приймати рішення по блокуванню тих чи інших користувачів системи Carma. Сессія заблокованого користувача буде видалена і як наслідок, спрацює автоматичний вихід з системи заблокованого покупця чи продавця. При спробі повторно увійти в свій акаунт, покупець чи продавець побачить відповідну помилку про те, що його було заблоковано і він може звернутися до адміністратора. Усі автомобілі, що додав заблокований продавець буде сховано від відображення у

системі і усі пропозиції, що були створені заблокованим покупцем також буде сховано від відображення у продавців.

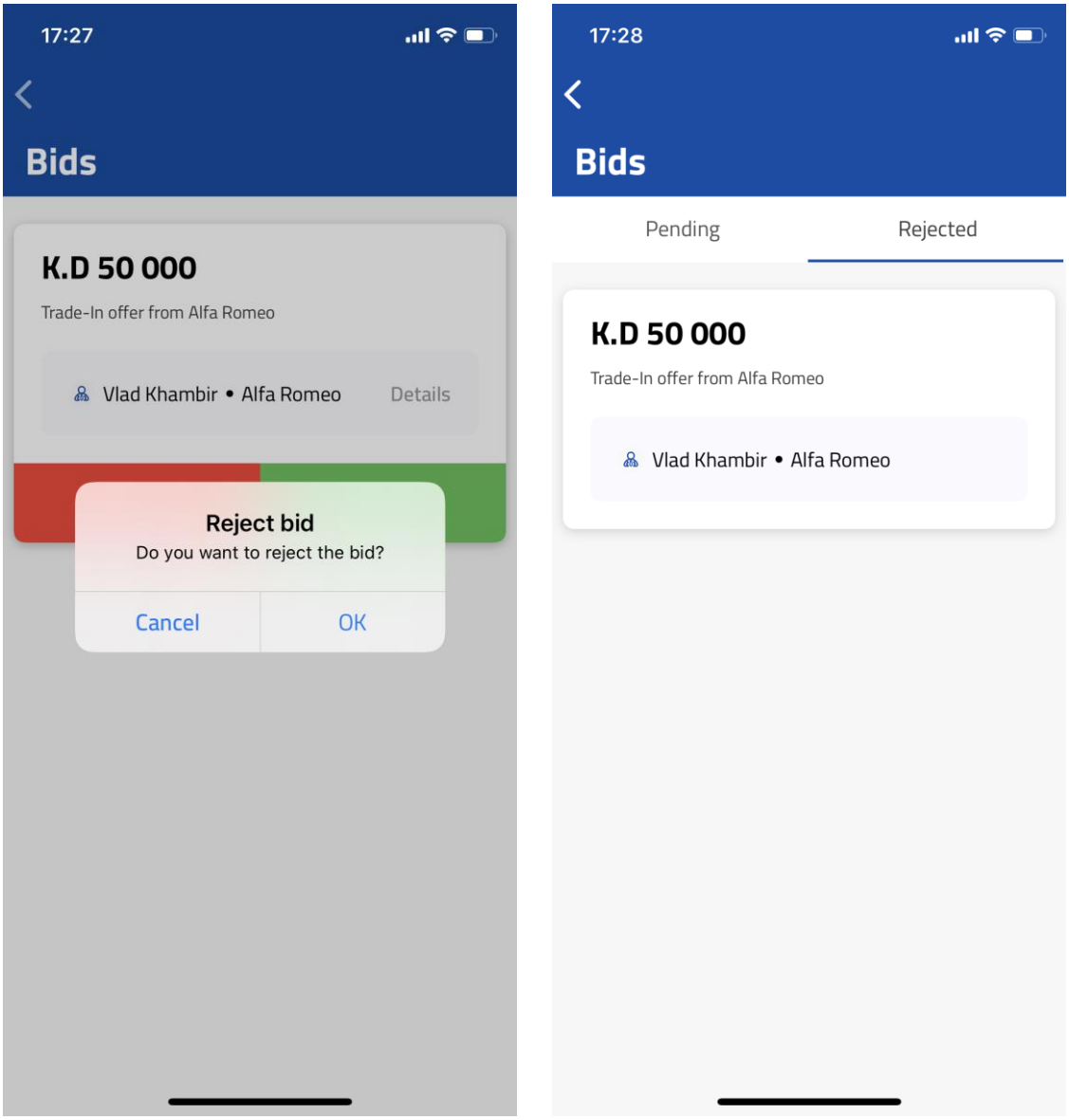
Рис. 3.9.7. Повний профіль покупця



Після того, як продавець підтверджує свою зацікавленість та підтверджує пропозицію, він отримає контакти покупця, а покупець у свою чергу отримує нотіфікацію та лист на пошту про те, що його пропозиція змінила статус на “Активна пропозиція” (Рис. 5.9.8.).

Рис. 3.9.8. Вкладка з активними пропозиціями у покупця

У разі, якщо користувач не захоче розгладати певну пропозицію, він має змогу відмовитися від пропозиції, натиснувши на відповідну кнопку. Після чого додаток запитає підтвердження користувача про відмову від пропозиції і у разі



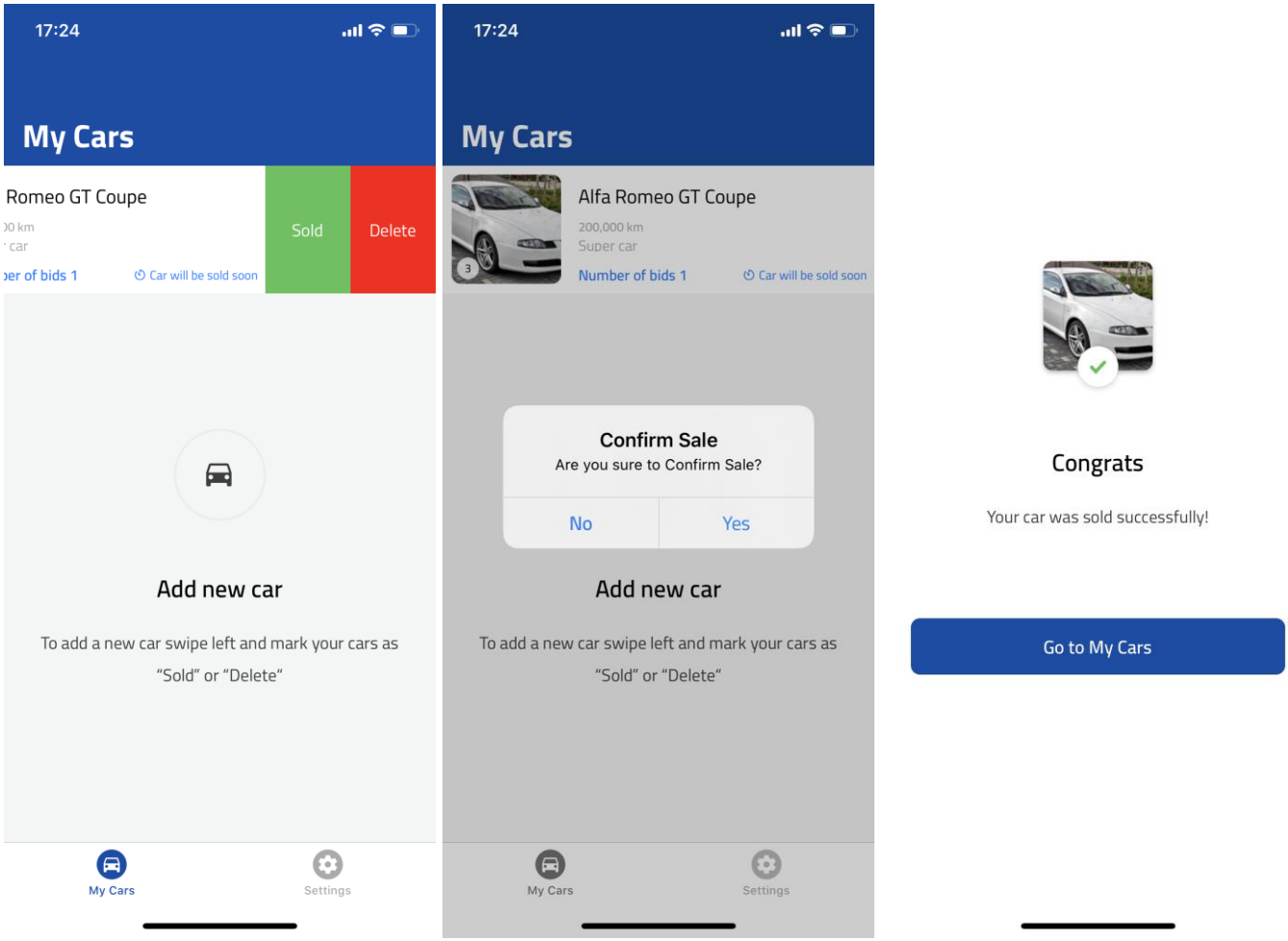
підтвердження, пропозиція буде відхилена Рис. (3.9.9).

Рис. 3.9.9. Відхилення пропозиції

Якщо користувач має хочаб одну відхилену пропозицію, то перелік пропозицій буде розбито на дві групи: “В очікуванні” та “Відхилені”. У группі відхилених пропозицій будуть відображатися відхилені пропозиції без можливості перегляду деталей покупця, що зробив цю пропозицію.

3.10. Процес продажу авто

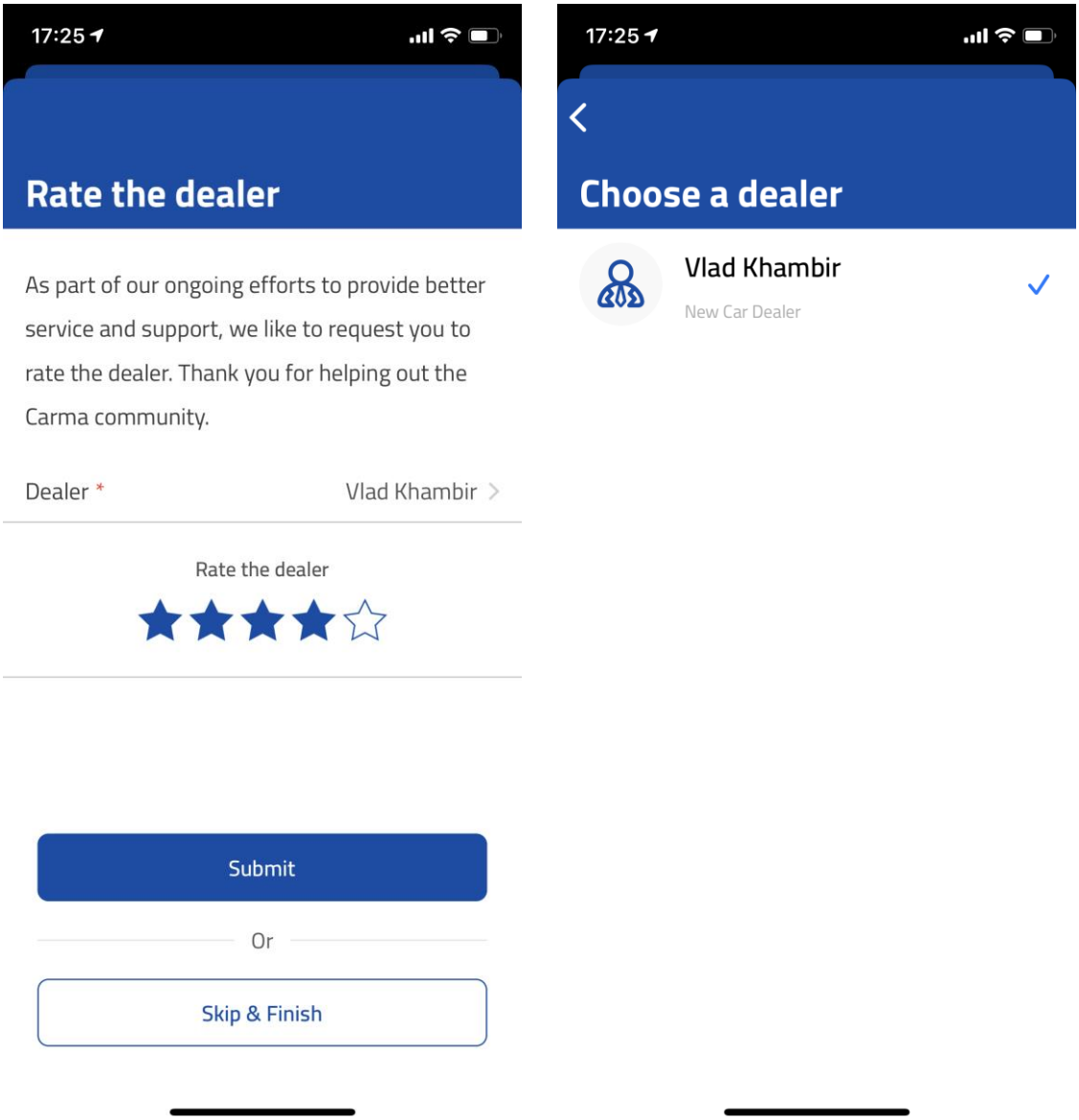
Користувач може відмітити автомобіль як проданний з екрану пропозицій, або з екрану автомобілей. Після натискання на кнопку “Продано”, додаток запитає підтвердження у користувача і у разі підтвердження, автомобіль будет відмічено як проданий та користувач побачить екран з повідомленням про успішний продаж автомобіля. З цього екрану користувач може повернутися на екран зі списком автомобілів, натиснувши на кнопку “Перейти до моїх авто” (Рис. 3.10.1). Якщо автомобіль був відміченний як проданий, то він зникне зі списку автомобілів для



продажі.

Рис. 3.10.1. Процес продажі авто

Після успішного продажу, додаток запитає у користувача кому саме був проданий автомобіль та як він оцінює покупця. Користувач може обрати одного з продавців, пропозицію котрого було прийнято у процесі продажі. Ця інформація є дуже корисною для статистики нашого сервісу, але якщо користувач не бажає ділитися даною інформацією, він має змогу відммовитися, натиснувши на кнопку



“Пропустити та продовжити” (Рис. 3.10.2).

Рис. 3.10.2. Екран оцінки покупця

ВИСНОВКИ ДО РОЗДІЛУ 3

1. У цьому розділі було розглянуто весь функціонал додатку, що включає у себе огляд реєстрації, авторизації, основних налаштувань, процесу додавання авто, розгляд пропозицій та процесу продажу авто з оцінкою покупця.

2. Користувач має змогу зареєструватися, використовуючи свій email.

3. Серед налаштувань доступними є: налаштовування мови додатка, змінення своїх персональних даних та відновлювання втраченного паролю.

4. Користувач має змогу додати автомобіль у три кроки, де заповнюється уся необхідна інформація для продажу та додаються фотографії авто.

5. Отримані пропозиції користувач може переглянути, прийняти, або відклонити.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА ПУБЛІКАЦІЯ ПРОДУКТУ

4.1. Тестування продукту

Тестування додатку - це важливий етап, на якому перевіряється поведінка додатки на великій кількості вхідних даних, включаючи невірні.

На тестування було відділено тиждень. Завданням займалася наша команда тестувальників. Використовувалось ручне (*manual*) тестування.

Ручне тестування (manual testing) - частина процесу тестування на етапі контролю якості в процесі розробки програмного забезпечення. Воно проводиться тестувальниками або звичайними користувачі шляхом моделювання можливих сценаріїв дії користувача.

Завдання тестувальника полягає в пошуку найбільшої кількості помилок. Він повинен добре знати яких найчастіше припускаються помилки і вміти знаходити їх за мінімально короткий період часу. Решта помилки, які не є типовими, виявляються тільки ретельно створеними наборами тестів.

Ручне тестування полягає у виконанні задокументованої процедури, де описана методика виконання тестування. Методика задає порядок тестів і для

кожного тесту - список значень параметрів, який подається на вхід зі списку результатів на виході.

Приклад фрагмента процедури:

- Подати на вхід різні дані;
- Запустити тестове виконання;
- Перевірити, чи відповідає отриманий результат таблиці.

У процесі тестування було виявлено ряд помилок у додатку, які були успішно виправлені, а також були внесені деякі зміни у UI.

4.2. Публікація продукту

Для публікації додатка в AppStore необхідно мати акаунт розробника і членство в програмі розробників Apple. Apple Developer Program - оплачувана раз на рік підписка, що дає вам доступ до Apple Developer Center і можливість публікувати ваші програми у App Store.

Для розробки і публікації додатка необхідно згенерувати відповідні сертифікати: сертифікат розробника і сертифікат дистриб'ютора. Членство в програмі розробників Apple потрібно для запиту, завантаження та використування сертифікатами для підписання, видані Apple.

Для розробників, які входять до команди, зареєстрованої як організація, ви також повинні бути власником облікового запису або адміністратором, щоб вимагати сертифікатів розповсюдження, які використовуються для подання програм у App Store.

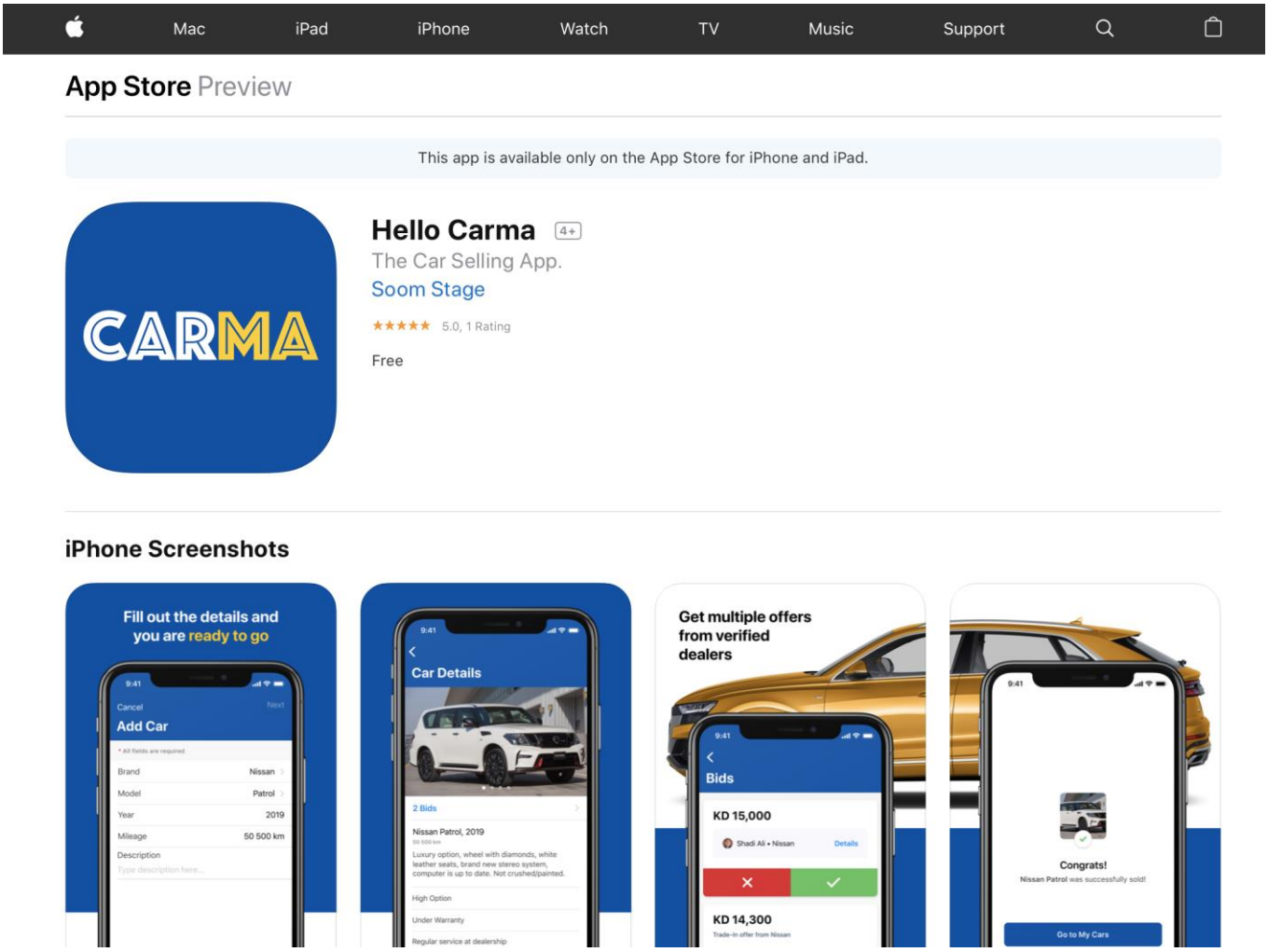
Сертифікати завантажуються в IDE Xcode, яка потім дозволяє завантажити білд додатка безпосередньо у центр управління додатками - iTunes Connect. У центрі управління додатками можна організувати альфа- і бета-тестування за допомогою сервісу TestFlight. TestFlight дозволяє запрошувати користувачів тестувати ваші

програми та збирати цінні відгуки, перш ніж випускати ваші додатки у App Store. Ви можете запросити до 10 000 тестувальників, використовуючи лише свою електронну адресу або поділившись загальнодоступним посиланням. Тестери можуть розпочати роботу, прийнявши ваше запрошення електронною поштою або перейшовши на загальнодоступне посилання. Щоб встановити додаток і надати зворотній зв'язок, тестери використовуватимуть додаток TestFlight для iPhone, iPad, iPod touch, Apple Watch та Apple TV. Завдяки TestFlight 2.3 на iOS 13 тестери можуть надсилати відгуки безпосередньо з вашого додатку, просто знімаючи знімок екрана.

Тестувальники також можуть надати додатковий контекст щодо збоїв у додатку одразу після його виникнення. Щоб переглянути цей відгук, треба перейти на сторінку програми TestFlight у додатку в App Store Connect та натиснути “Збій” або “Скріншоти” у розділі “Зворотній зв’язок". Відгуки тестерів на tvOS або більш ранніх версіях iOS будуть надіслані на електронну адресу, яка вказана в тестовій інформації.

Після тестування заявку на публікацію програми необхідно відправити на рецензування в компанію Apple. Білду надається версія, докладний опис і кілька скріншотів екрану. Розгляд займає в середньому 7-20 днів. Після закінчення розгляду, компанія залишає за собою право відмовити в публікації продукту, якщо він не відповідає правилам для публікації.

Білд додатку з версією 1.0.0 був завантажена в центр управління додатками і відправлено на розгляд. Через декілька днів додаток успішно пройшло рецензування



і було опубліковано.

Рис. 4.2.1. Публікація у магазині AppStore

ВИСНОВКИ ДО РОЗДІЛУ 4

1. Було проводено ручне тестування додатку, що дозволило виявити помилки, які у результаті були виправлені.

2. У розділі було розглянуто основні положення публікації додатку у магазин додатків AppStore.

3. У розділі було розглянуто можлівості сервісу App Store Connect та TestFlight, які використовуються для поширення додатку між користувачами та тестувальникамми.

4. Додаток пройшов рецензування від Apple та був успішно опублікований у магазині додатків AppStore.

ВИСНОВКИ

У ході виконання дипломного проєкту було створено iOS додаток-клієнт для сервісу по продажу автомобіля онлайн “Carma”.

У додатку користувач може з легкістю додати свій автомобіль на продажу, зазначивши мінімальний набір даних про цей автомобіль. Також було реалізовано підтримку RTL інтерфейсу, що значно полегшить використання додатка для жителів країни Кувейт.

Було проведено ручне тестування у ході якого були усунені усі найдені проблеми. Також покриття Unit-тестами проєкту сягає більше ніж 60% коду, що дозволить у подальшому безпечно проводити рефакторінг та додавати новий функціонал.

Додаток успішно опубліковано у магазині, запропонованому Apple - AppStore. На даний момент розглядається розробка наступної версії додатка з розширеним функціоналом, серед якого є оцінювач авто онлайн.

Під час виконання дипломного проєкту було розглянуто шлях до вибору архітектури, розроблено поліпшену версію MVVM архітектури, наведено приклади використання реактивного та протоколо-орієнтованного програмування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- Мова програмування Swift. Електронний доступ. URL: https://ru.wikipedia.org/wiki/Swift_(язык_программирования)

- Матеріал про SourceTree з Національної бібліотеки ім. Н.Є. Баумена. Електронний доступ. URL: https://ru.bmstu.wiki/SourceTree
- IDE Xcode. Електронний доступ. URL: https://uk.wikipedia.org/wiki/Xcode
- Вступ про систему контролю версій GIT. Електронний доступ. URL: https://git-scm.com/book/uk/v2/Вступ-Про-систему-контролю-версій
- Засіб управління щалежностями Cocoa-бібліотек CocoaPods. Електронний доступ. URL: https://habr.com/en/company/luxoft/blog/149631/
- Огляд від Apple MVC. Електронний доступ. URL: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html
- MVVM. Електронний доступ. URL: https://metanit.com/sharp/wpf/22.1.php
- RxSwift. Електронний доступ. URL: http://ovchinnikov.cc/writing/rxswift-intro/
- Діаграма прецедентів. Електронний доступ. URL: https://flexberry.github.io/ru/fd_use-case-diagram.html
- Протоколо-орієнтоване програмування. Електронний доступ. URL: https://www.hackingwithswift.com/example-code/language/what-is-protocol-oriented-programming
- Кодогенерація Sourcery. Електронний доступ. URL: https://github.com/krzysztofzablocki/Sourcery
- Crashlytics від Firebase. Електронний доступ. URL: https://uk.wikipedia.org/wiki/Firebase
- Прийоми об’єктно-орієнтованного проєктування. / Гамма Э. Хелм Р. Джонсон Р. Влиссидес Дж. [та ін.] ; Серія «Бібліотека програміста», СПб: Пітер, 2001. — 368 с. – Бібліогр.: с. 280–291