math_2023_042

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Факультет математики та інформатики
кафедра математичного моделювання

Розробка сервісу P2P доставки з використанням Django та Swift

Кваліфікаційна робота

Рівень вищої освіти – другий (магістерський)

Виконав:
студент 6 курсу, 607 групи
Цуркан Дмитро Євгенович
***Керівник**:*
асистент, к.т.н, Шкільнюк Д.В.

До захисту допущено
на засіданні кафедри
протокол № 9 від 5 грудня 2023 р.
Зав. кафедрою _____ проф. Черевко І.М.

Чернівці – 2023

Анотація

В дипломній роботі розглядається питання розробки сервісу з доставки товарів з використанням Django та Swift.

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів наукових досліджень інших авторів мають посилання на відповідне джерело.

_____ Д.Є. Цуркан
(підпис)

Annotation

The thesis deals with the development of a service for the delivery of goods using Django and Swift.

The qualification work contains the results of my own research. The use of ideas, results, and texts of scientific research of other authors have a link to the appropriate source.

_____ Д.Є. Цуркан
(підпис)

Зміст

Вступ

Сучасний світ переживає різкі зміни у сфері електронної комунікації та технологічних інновацій, які впливають на усі аспекти нашого повсякденного життя. Дані зміни суттєво впливають на способи, якими ми здійснюємо різні аспекти наших потреб, зокрема, доставку товарів та послуг.

Сервіси P2P (Peer-to-Peer) доставки, що базуються на мобільних додатках та веб-платформах, стали важливою складовою глобальної економіки та споживчої культури. Такі сервіси дозволяють індивідуальним користувачам здійснювати взаємодію з іншими користувачами, які можуть надавати послуги доставки товарів. Дана концепція відкриває нові можливості для зменшення часу та витрат, пов'язаних з доставкою, і сприяє розвитку ефективних та інноваційних методів отримання товарів та послуг.

В рамках даної дипломної роботи досліджується розробка сервісу P2P доставки, який базується на двох ключових технологіях: Django [1] та Swift [2]. Django [1] є популярним веб-фреймворком, побудованим на мові програмування Python [6], який надає зручні інструменти для розробки веб-додатків та управління базами даних. Swift [2], з іншого боку, є мовою програмування, розробленою Apple, і широко використовується для створення мобільних додатків для платформ iOS та macOS [4].

Дипломна робота присвячена докладному аналізу, проектуванню та розробці P2P сервісу доставки, який об'єднує функціональність веб-додатка та мобільного додатка. Розробка такого сервісу вимагає глибокого розуміння інформаційних технологій, включаючи роботу з базами даних, створення користувацького інтерфейсу та розробку серверної логіки.

Основні цілі цієї дипломної роботи включають:

1. Аналіз існуючих сервісів P2P доставки та їхніх функціональних особливостей.
2. Проектування архітектури сервісу P2P доставки на основі Django [1] та Swift [2].
3. Розробка веб-додатка та мобільного додатка, включаючи створення користувацького інтерфейсу та серверної логіки.
4. Тестування та оцінка продуктивності розробленого сервісу.
5. Підсумковий аналіз результатів та можливостей подальшого розвитку.

Дана дипломна робота є актуальною, оскільки вона спрямована на розробку практичного та інноваційного продукту, який може полегшити процес доставки товарів та послуг для користувачів. Результати цього дослідження можуть бути використані, як основа для подальших досліджень у галузі розвитку P2P доставки та технологічних рішень.

Загальна мета цієї дипломної роботи - надати інноваційний внесок у галузь P2P доставки, використовуючи потужність технологій Django [1] та Swift [2].

Розділ 1. Аналіз існуючих сервісів P2P доставки та їхніх функціональних особливостей.

1.1 Огляд існуючих рішень.

Сучасні сервіси P2P доставки стали невід'ємною частиною економіки та споживчої культури. Такі платформи дозволяють індивідуальним користувачам надавати послуги доставки товарів та послуг і взаємодіяти один з одним для задоволення своїх потреб. У цьому розділі проведено аналіз кількох існуючих сервісів P2P доставки та розглянуті їхні функціональні особливості.

Uber Eats:

- *Основна мета*: Uber Eats дозволяє користувачам замовляти їжу з ресторанів та отримувати її доставку до дверей.
- *Мобільний додаток*: Мобільний додаток доступний для платформ Android та iOS [4].
- *Вибір ресторанів*: Користувачі можуть переглядати ресторани в окрузі, переглядати меню та робити замовлення.
- *Доставка*: Доставка здійснюється кур'єрами, які працюють на платформі Uber Eats.
- *Оплата*: Користувачі можуть сплачувати за послуги через додаток за допомогою кредитних карт або інших методів оплати.

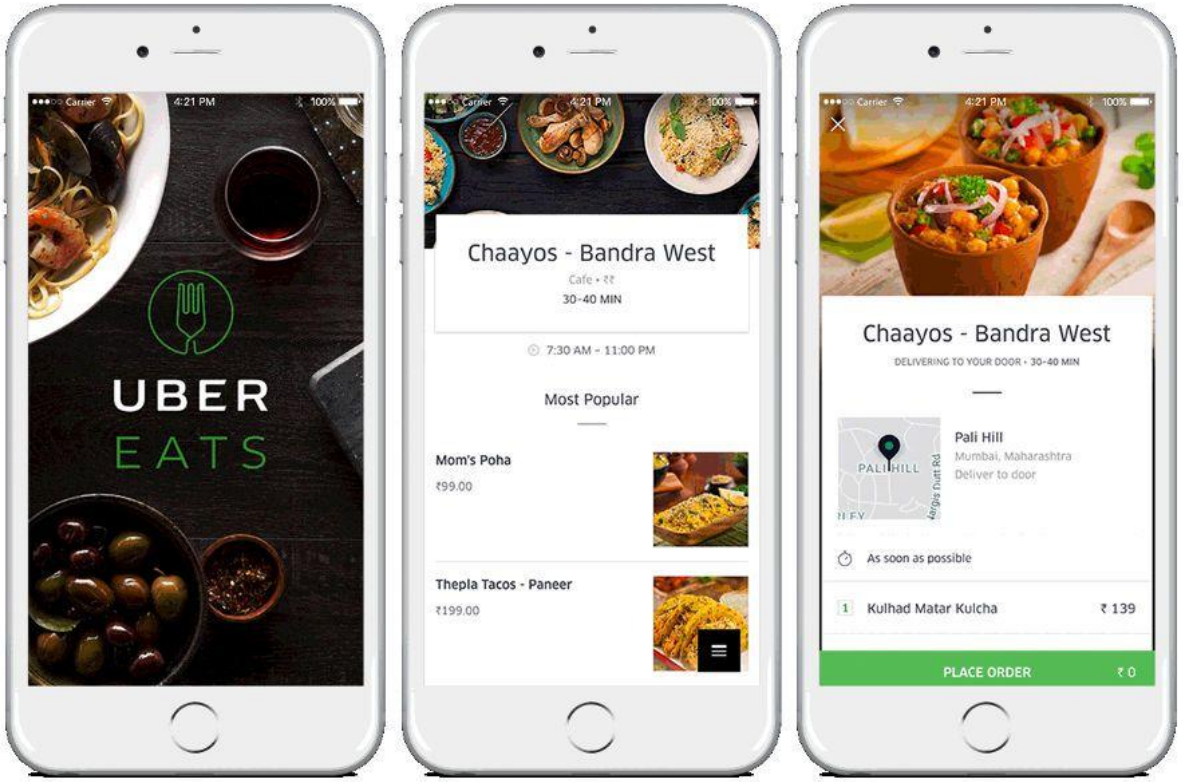


Рисунок 1.1 – Uber Eats iOS додаток

Airbnb:

- *Основна мета*: Airbnb дозволяє користувачам знаходити та замовляти житло для короткострокового проживання.
- *Веб-платформа та мобільний додаток*: Airbnb надає доступ до платформи, як через веб-сайт, так і через мобільні додатки на Android та iOS [4].
- *Пошук житла*: Користувачі можуть вказувати параметри пошуку, такі як місцезнаходження, дати проживання та тип житла.
- *Бронювання та оплата*: Користувачі можуть бронювати житло та сплачувати за нього через платформу Airbnb.

Рисунок 1.2 – Airbnb iOS додаток

Postmates:

- *Основна мета*: Postmates надає можливість замовити різноманітні товари та їжу для доставки до дверей.
- *Мобільний додаток*: Postmates працює на Android та iOS.
- *Замовлення товарів*: Користувачі можуть замовляти їжу, продукти, алкоголь, ліки та інші товари з різних закладів.
- *Доставка*: Доставка здійснюється кур'єрами, які можуть використовувати власний транспорт.
- *Оплата*: Користувачі оплачують товари та послуги через додаток.

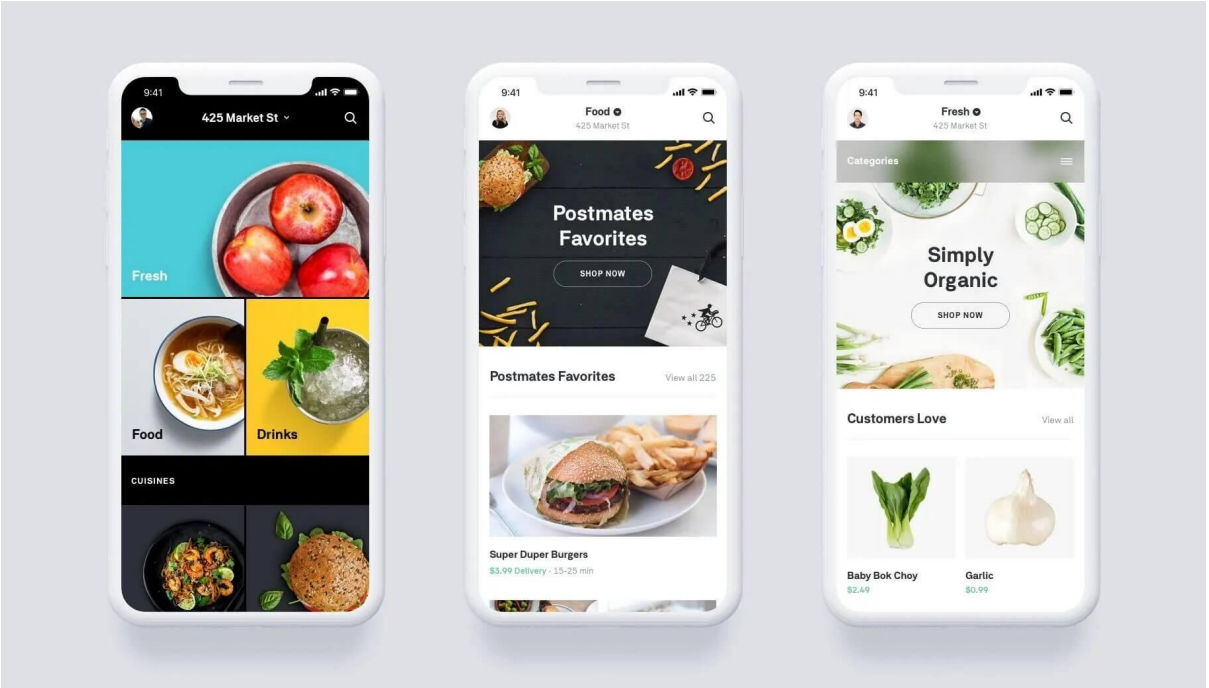


Рисунок 1.3 – Postmates iOS додаток

TaskRabbit:

- *Основна мета*: TaskRabbit дозволяє користувачам знайти людей, які готові виконати різні завдання або послуги.
- *Мобільний додаток*: TaskRabbit має мобільні додатки для Android і iOS [4].
- *Замовлення послуг*: Користувачі можуть створювати завдання, описувати їх та вказувати бюджет.
- *Вибір виконавця*: Інші користувачі (виконавці) можуть пропонувати свої послуги та ціни.
- *Оплата та рейтинг*: Оплата відбувається через платформу, а користувачі залишають відгуки та оцінки виконавцям.

Рисунок 1.4 – TaskRabbit iOS додаток

Такі приклади існуючих сервісів P2P доставки вказують на те, що ця галузь дозволяє задовольняти різні потреби користувачів через різноманітні функціональні можливості. При розробці власного сервісу P2P доставки на базі Django [1] та Swift [2], повинні бути враховані найкращі практики та функціональні особливості, які були вже впроваджені в цьому сегменті ринку.

1.2 Аналіз та постановка задачі на основі результатів досліджень.

На підставі аналізу існуючих сервісів P2P доставки та їхніх функціональних особливостей, можна виокремити кілька ключових аспектів, які варто враховувати при розробці власного сервісу. Зазначені аспекти надають базу для постановки завдань і визначення цілей такого проекту.

На підставі аналізу існуючих сервісів P2P доставки та їхніх функціональних особливостей, можна виокремити кілька ключових аспектів, які варто враховувати при розробці власного сервісу. Зазначені аспекти надають базу для постановки завдань і визначення цілей такого проекту.

1. **Мобільні платформи**: Багато існуючих сервісів P2P доставки успішно використовують мобільні додатки для спрощення процесу замовлення та взаємодії з користувачами. Отже, для даного проекту важливо розробити мобільний додаток для платформ Android і iOS [4], який надасть зручний інтерфейс для користувачів.
2. **Замовлення та доставка**: Система замовлення та доставки є ядром сервісів P2P доставки. Потрібно розробити ефективну систему замовлення товарів або послуг та організації доставки до клієнтів. Це включає в себе вибір товарів, встановлення цін, пошук вільних кур'єрів, визначення маршрутів і відстеження доставки.
3. **Платіжна система**: Одні з найважливіших функціональних особливостей - це забезпечення безпеки та зручності платежів. Потрібно розробити систему оплати, яка дозволить користувачам зручно та безпечно сплачувати за замовлення через додаток. Така система повинна підтримувати різні методи оплати, мати можливість блокування коштів в системі та постоплати після підтвердження замовлення.
4. **Користувацький інтерфейс**: Важливо створити зручний та привабливий користувацький інтерфейс для мобільних додатків, який дозволить користувачам легко здійснювати замовлення та взаємодіяти з сервісом. Це включає в себе розробку інтуїтивного дизайну, зручних елементів управління та можливість відстеження статусу замовлення.

5. **Відгуки та рейтинги**: Багато існуючих сервісів використовують системи відгуків та рейтингів для оцінки якості послуг та відвідуваних місць. В даному сервісі також має бути врахувано можливість для користувачів залишати відгуки та ставити оцінки товарів, послуг та кур'єрам.

На основі вищезазначених аспектів та функціональних особливостей існуючих сервісів P2P доставки, формулюються задачі та цілі для такого проекту:

1. **Розробка мобільного додатку**: Створення мобільного додатку для платформ Android і iOS, який надасть можливість користувачам замовляти товари або послуги та взаємодіяти з сервісом.
2. **Реалізація системи замовлення та доставки**: Створення системи, яка дозволить користувачам робити замовлення та організовувати доставку до місця призначення.
3. **Розробка платіжної системи**: Створення безпечної та зручної системи оплати для користувачів, яка підтримує різні методи оплати.
4. **Створення користувацького інтерфейсу**: Розробка інтуїтивного та привабливого користувацького інтерфейсу для мобільних додатків.
5. **Впровадження системи відгуків та рейтингів**: Реалізація системи для користувачів, яка дозволить залишати відгуки та ставити оцінки послугам та кур'єрам.

   Зазначені задачі та цілі становлять основу проекту розробки сервісу P2P доставки з використанням Django [1] та Swift [2]. Вони враховують найкращі практики та функціональні особливості, визначені під час аналізу існуючих сервісів і спрямовані на створення високоефективного та конкурентоспроможного продукту.

Можливості авторизованого користувача-кур'єра:

- sign up, sign in, forgot/reset password функціональності;
- вихід з профілю (logout);
- редагування облікового запису;
- перегляд всіх доступних замовлень;
- перегляд деталей замовлення;
- перегляд нотифікацій;
- добавлення платіжних даних;
- верифікація профілю;
- можливість відгукнутись на замовлення;
- залишити відкуг замовнику.

Можливості авторизованого користувача-замовника:

- sign up, sign in, forgot/reset password функціональності;
- вихід з профілю (logout);
- редагування облікового запису;перегляд нотифікацій;
- добавлення платіжних даних;
- верифікація профілю;
- перегляд усіх своїх замовлень;
- створення нових замовлень;
- підтвердження вибраного кур'єра;
- залишити відгук кур'єру.

1.3 Висновки до розділу “Аналіз існуючих сервісів P2P доставки та їхніх функціональних особливостей”.

Після ретельного аналізу різноманітних сервісів P2P доставки та їхніх функціональних особливостей можна зробити кілька важливих висновків, які слугуватимуть фундаментом для розробки нашого власного сервісу на базі Django [1] та Swift [2].

1. **Мобільна спрямованість**: Сучасні користувачі все частіше використовують мобільні пристрої для здійснення замовлень та взаємодії з послугами доставки. Тому важливо розробити мобільний додаток для платформ Android і iOS [4], який надасть користувачам зручний доступ до сервісу.
2. **Система замовлення та доставки**: Процес замовлення та організації доставки є ключовими функціональними особливостями. Важливо створити ефективну систему, яка спрощує процес замовлення, забезпечує вибір товарів та послуг, пошук вільних кур'єрів та відстеження доставки.
3. **Платіжна система**: Безпека та зручність оплати є важливими чинниками для користувачів. Розробка системи оплати, яка підтримує різні методи оплати та гарантує безпеку транзакцій, є обов'язковою.
4. **Користувацький інтерфейс**: Інтуїтивний та привабливий користувацький інтерфейс дозволить користувачам легко взаємодіяти з сервісом та здійснювати замовлення без зайвих перешкод.
5. **Відгуки та рейтинги**: Відгуки та рейтинги є важливими для підвищення довіри користувачів до сервісу та оцінки якості послуг. Їх впровадження дозволить забезпечити якість та надійність сервісу.

Загальний висновок полягає в тому, що ринок P2P доставки надзвичайно конкурентний та швидкозмінний. Існуючі сервіси вже встановили високі стандарти щодо зручності, безпеки та якості обслуговування, тому новий проект повинен зосередитися на розвитку та вдосконаленні цих аспектів. Ретельний аналіз конкурентів повинен допомогти уникнути дублювання функцій та знайти нові можливості для вдосконалення такого сервісу.

Зазначені висновки допомогають краще розуміти вимоги користувачів та розвити сервіс, який відповідає сучасним стандартам та вимогам. Під час розробки даного проекту, планується акцентувати увагу на мобільному додатку, зручній системі замовлення та доставки, безпеці платежів, користувацькому інтерфейсі та системі відгуків та рейтингів, щоб надати користувачам найкращий можливий досвід використання сервісу P2P доставки.

Розділ 2. Опис інструментів та технологій розробки.

У цьому розділі розглядаються інструменти та технології, які були використані для розробки сервісу P2P доставки з використанням Django [1] та Swift [2]. Розділ поділений на підрозділи, які детально описують архітектуру, інструменти розробки, а також обґрунтовують вибір технологій та інструментів.

2.1 Вибір архітектури для серверної частини та мобільного додатку

При розробці серверної частини сервісу була обрана архітектура, заснована на фреймворку Django [1]. Django [1] надає потужні інструменти для швидкого створення серверних додатків та роботи з базами даних, що дозволить швидко розробити та підтримувати сервіс.

При розробці сервісу P2P доставки було необхідно вибрати підходящі архітектурні моделі для серверної частини і мобільного додатку. Для обох складових проекту було обрано архітектурну модель, відому як "MVC" (Model-View-Controller).

Модель-View-Controller (MVC) - це архітектурний шаблон, який розподіляє програмний код на три основні компоненти: модель (Model), представлення (View) та контролер (Controller). Кожен із цих компонентів відповідає за свої функції та спрощує розробку та підтримку програмного забезпечення.

2.2 Опис архітектури серверної частини (MVC для сервера)

Архітектура серверної частини нашого сервісу P2P доставки ґрунтується на моделі "Model-View-Controller" (MVC [11]). Дана модель розподіляє

функціональність на три основні компоненти, що спрощує організацію коду та взаємодію між ними.

1. **Модель (Model)**:
    - Модель включає у себе всі дані та бізнес-логіку, необхідну для функціонування серверної частини. В даному випадку, це може включати в себе дані про користувачів, замовлення, операції оплати, графік кур'єрів та інші об'єкти, які визначають функціональність сервісу.
    - Модель також відповідає за взаємодію з базою даних, де інформація зберігається і витягується. При використанні Django ORM (Object-Relational Mapping), можно описувати моделі даних та здійснювати запити до бази даних зручним та Python-подібним способом [6].
    - Модель визначає структуру даних і має методи для додавання, оновлення, видалення та читання інформації. Вона також містить логіку для валідації даних та забезпечення дотримання бізнес-правил.
2. **Представлення (View)**:
    - Представлення в серверній частині відповідають за обробку запитів та відображення даних. Вони приймають HTTP-запити від користувачів і відповідають на них, забезпечуючи доступ до функціональності сервісу.
    - Використовуючи Django REST framework [3], можно легко створювати API для взаємодії з мобільним додатком та іншими клієнтами. Представлення обробляють HTTP-запити, виконують відповідні операції в моделі та повертають відповіді у форматі JSON або інших стандартних форматах.

- Представлення визначають URL-шляхи та маршрутизують запити до відповідних представлень. Це дозволяє визначити, які операції користувачі можуть виконувати через API (наприклад, створення нового замовлення чи отримання списку активних кур'єрів).

3. **Контролер (Controller)**:
   - У контексті Django [1], контролери представлені URL-шляхами і фреймворком, які визначають маршрути та керують тим, які представлення відповідають на різні запити. Це робить систему управління запитами більш організованою та чіткою.
   - Контролери відповідають за обробку запитів від користувачів, викликаючи відповідні представлення та передаючи їм параметри. Це забезпечує зручну розподілену обробку запитів та допомагає підтримувати чіткість логіки додатку.

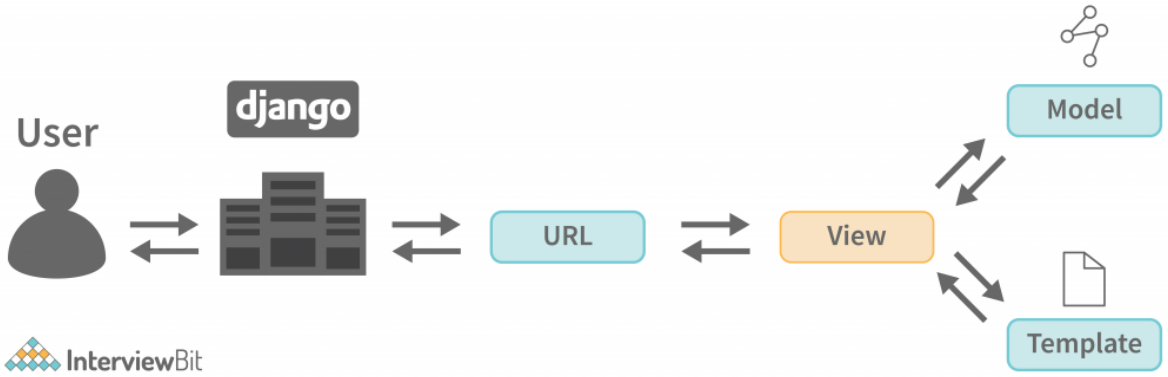


Рисунок 2.1.1 – Django MVC [11]

Використання архітектури "Model-View-Controller" для серверної частини серверу дозволяє розподілити відповідальності між компонентами, що спрощує розробку та підтримку серверної частини. Така модель дозволяє

створити добре організований та розширюваний код, який відповідає потребам сервісу P2P доставки.

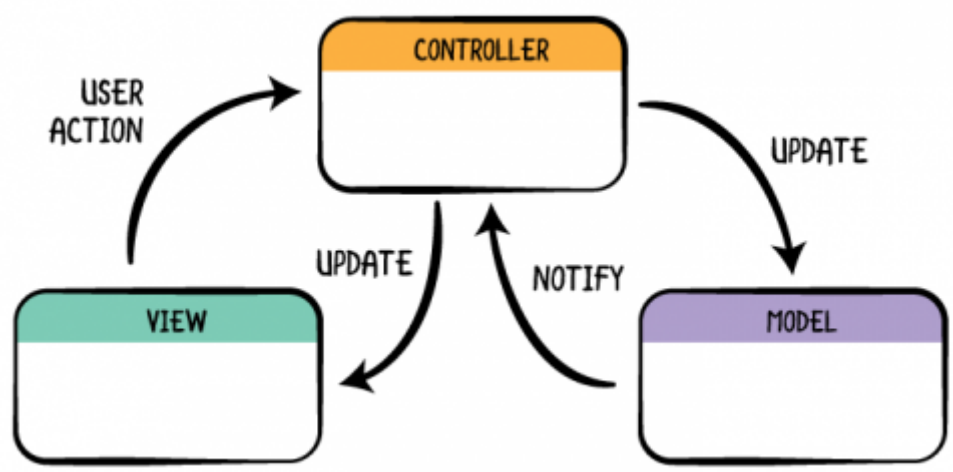
2.3 Опис архітектури мобільного додатку (MVC для мобільного додатку)

Архітектура мобільного додатку нашого сервісу P2P доставки базується на моделі "Model-View-Controller" (MVC [12]). Дана модель дозволяє розділити функціональність додатку на логічні компоненти та спрощує розробку, тестування та підтримку програмного забезпечення. Детальний розглянд кожного компоненту архітектури наведений нижче:

1. **Модель (Model)**:
    - Модель в мобільному додатку представляє собою дані та бізнес-логіку, необхідну для роботи додатку. Вона включає в себе дані про користувачів, замовлення, операції оплати та інші об'єкти, які відображають стан додатку.
    - Модель відповідає за роботу з цими даними, зокрема за зберігання, оновлення та витягування інформації. Вона також має бізнес-логіку для перевірки даних, обробки подій та виконання різних операцій.
    - Модель може включати в себе різноманітні інструменти для роботи з базою даних, мережевими запитами та обробки даних.
2. **Представлення (View)**:
    - Представлення в мобільному додатку відповідають за відображення інтерфейсу користувача. Вони містять всі компоненти і елементи управління, які користувач може бачити та взаємодіяти з ними.

- Представлення обробляють відображення даних і реагують на дії користувача, такі як натискання на кнопки, введення тексту і переходи між екранами. Вони також можуть відправляти запити до контролера для отримання даних чи виконання певних операцій.
- Представлення відповідають за створення і підтримку графічного інтерфейсу та дизайну додатку.

3. **Контролер (Controller)**:
   - Контролери в мобільному додатку відповідають за керування логікою додатку та взаємодію між моделлю та представленням.
   - Вони обробляють вхідні дані від користувача та відправляють відповідні запити до моделі для отримання даних та виконання операцій. Контролери також можуть відправляти оновлення даних до представлення для відображення змін.
   - Контролери визначають поведінку додатку та взаємодію з моделлю та представленням. Вони відповідають за логіку переходів між екранами та виконання бізнес-правил.



Архітектура "Model-View-Controller" (MVC [12]) для мобільного додатку дозволяє розподілити функціональність додатку на логічні компоненти та зробити код більш організованим і розширюваним. Вона спрощує розробку і підтримку додатку, робить його більш структурованим і допомагає команді розробників працювати над різними аспектами додатку паралельно.

2.4. Опис інструментів розробки та обґрунтування вибору технологій та інструментів

У процесі розробки сервісу P2P доставки використовуються різноманітні інструменти та технології для забезпечення функціональності та надійності додатку. Ось докладний огляд основних інструментів і їх застосувань:

1. **Django** [1]: Django - це потужний веб-фреймворк, побудований на мові програмування Python [6]. Він був використаний для розробки серверної частини додатку. Django [1] надає готовий набір інструментів для створення веб-додатків, включаючи систему аутентифікації, маршрутизацію, роботу з базою даних та багато інших корисних функцій.

   Django [1] був обраний для розробки серверної частини через його швидкість та потужність у створенні веб-серверів і RESTful API [8]. Django [1] також надає безпеку та можливості автентифікації, що були важливі для створення безпечного середовища для користувачів.

2. **Braintree та Stripe**: Braintree і Stripe - це платіжні шлюзи, які дозволяють користувачам безпечно та зручно оплачувати послуги та

товари через додаток. Вони інтегрованы для обробки транзакцій та забезпечення безпеки платежів.

Вибір Braintree та Stripe для обробки платежів обгрунтовується їхньою репутацією, безпекою та зручністю для користувачів. Перечислені платіжні системи надають можливість проводити транзакції з використанням різних методів оплати та забезпечують надійність операцій.

3. **Facebook SDK**: Facebook SDK дозволяє користувачам використовувати свої облікові записи на Facebook, Instagram для автентифікації в додатку. Така інтеграція спрощує реєстрацію та вхід користувачів, а також надає можливість обміну контентом на Facebook та Instagram.

   Facebook SDK був вибраний для спрощення автентифікації користувачів через їхні облікові записи на Facebook. В результаці процес реєстрації та входу для користувачів буде більш зручним і швидким.

4. **PassportEye і phonenumberslite**: PassportEye - це інструмент для розпізнавання даних з паспортів, що допомагає у верифікації користувачів та отриманні необхідних інформації. Phonenumberslite використовується для валідації та обробки номерів телефонів користувачів.
5. **Twilio**: Twilio надає інструменти для взаємодії з користувачами через SMS та голосовий зв'язок. Це особливо корисно для сповіщень та підтверджень, а також для зв'язку з користувачами через текстові повідомлення.

Twilio був обраний для забезпечення зручного і надійного зв'язку з користувачами через SMS та голосовий зв'язок.

6. **IGListKit**: IGListKit - це бібліотека, яка допомагає створити ефективний та реагуючий інтерфейс списків. Вона використовується для покращення продуктивності та взаємодії зі списками елементів у додатку.

   IGListKit сприяв покращенню продуктивності та реагуючості інтерфейсу користувача шляхом використання техніки віртуалізації списків для відображення динамічних даних.

7. **Google Maps**: Інтеграція з Google Maps дозволяє користувачам визначати маршрути та відстежувати доставку у режимі реального часу. Це надає користувачам можливість зручно планувати та відстежувати доставку замовлень.
8. **RxSwift**: RxSwift [10] - це реактивний фреймворк для мови програмування Swift [2]. Даний фреймворк допомагає спростити асинхронне програмування та реакцію на події, що робить додаток більш відгуклим і легким для розробки. Дана технологія робить код більш структурованим і легко зрозумілим.
9. **SendBird SDK**: SendBird SDK дозволяє впроваджувати функції спілкування та чату в додаток. Надає користувачам можливість спілкуватися та обмінюватися повідомленнями в додатку.
10. **Moya**: Moya - це мережевий абстракційний шар, який полегшує взаємодію з API та робить код більш структурованим і чистим. Він допомагає забезпечити відокремлення мережевого коду від основного додатку. Moya був вибраний для спрощення взаємодії з API та зробив код більш структурованим та чистим.

11. **Firebase**: Firebase використовується для аналізу даних користувачів, аналітики додатку, збереження файлів, пуш нотифікацій та інших важливих функцій. Firebase також допомагає взаємодіяти з базами даних та надає безпеку даних.

У процесі розробки сервісу P2P доставки використовується система управління базами даних (СУБД) PostgreSQL для збереження та управління даними на серверній стороні. PostgreSQL є однією з найпотужніших та надійних вільно розповсюджуваних СУБД з відкритим вихідним кодом, і має безліч переваг для розробників та підприємств:

1. **Надійність і стійкість**: PostgreSQL славиться своєю надійністю та стійкістю до відмов. Вона підтримує механізми транзакцій та контролю цілісності даних, що робить її ідеальним для додатків, які потребують надійної роботи.
2. **Розширюваність**: PostgreSQL може легко розширюватися, дозволяючи розробникам додавати нові функції та розширення за допомогою розширень та процедур бази даних.
3. **Підтримка геоданих**: PostgreSQL має вбудовану підтримку геоданих, що робить її ідеальним вибором для додатків, які вимагають обробки географічних даних, таких як визначення маршрутів та відстеження геолокації.
4. **Відкритий вихідний код**: PostgreSQL має відкритий вихідний код, що дозволяє розробникам адаптувати її до своїх потреб і вносити власні модифікації.
5. **Підтримка ACID-транзакцій**: PostgreSQL підтримує стандарт ACID (Atomicity, Consistency, Isolation, Durability) для гарантування цілісності даних та коректності транзакцій.

6. **Спільнота розробників і користувачів**: PostgreSQL має активну спільноту розробників і користувачів, що означає, що завжди є доступ до допомоги, документації та оновлень.
7. **Широкі можливості інтеграції**: PostgreSQL підтримує багато мов програмування та має багато розширень, що спрощує інтеграцію з іншими технологіями та сервісами.

Використання PostgreSQL дозволяє ефективно зберігати та керувати даними, які стосуються користувачів, замовлень, операцій оплати та багатьох інших аспектів нашого додатку. Велика продуктивність та надійність PostgreSQL роблять його ідеальним вибором для проектів, які потребують збереження та обробки великої кількості даних.

З використанням цих інструментів і технологій є можливість реалізувати різноманітні функції та забезпечити користувачам зручний та надійний сервіс P2P доставки. Вони допомогли впровадити платіжні системи, інтеграцію з соціальними медіа, відстежування геолокації та інші важливі можливості, що зробили наш додаток конкурентоспроможним і зручним для користувачів.

2.5. Висновок до розділу “Опис інструментів та технологій розробки”.

Обгрунтування та пояснення вибору конкретних інструментів та технологій у розділі "Опис інструментів та технологій розробки" свідчать про глибокий аналіз та обґрунтування рішень, прийнятих в ході розробки сервісу P2P доставки. Кожен інструмент був вибраний з урахуванням специфікації проекту та його цілей, з метою забезпечення надійності, безпеки та функціональності додатку.

Вибір Django [1] та PostgreSQL для серверної частини дозволяє створювати потужну та надійну основу для роботи додатку. Інтеграція Braintree та Stripe надає зручний спосіб для користувачів здійснювати платежі, забезпечуючи безпеку та зручність операцій. Використання Facebook SDK спрощує процес автентифікації користувачів, роблячи його більш зручним.

Інструменти PassportEye та phonenumberslite допомогають у верифікації користувачів та забезпечуються правильну обробку особистих даних. Twilio дозволяє забезпечити надійний зв'язок з користувачами через SMS та голосовий зв'язок.

IGListKit сприяє покращенню продуктивності та реагуючості інтерфейсу користувача, що важливо для забезпечення задоволення від використання додатку. Інтеграція Google Maps надає можливість користувачам визначати маршрути та відстежувати доставку у реальному часі, що стало ключовим функціоналом даного сервісу доставки.

RxSwift [10] спрощує взаємодію з асинхронними операціями, забезпечуючи більш реактивний та продуктивний додаток. SendBirdSDK додає можливість спілкування та чату в додаток, роблячи його більш соціальним та інтерактивним.

Moya спрощує роботу з API, забезпечуючи чистий та структурований код. Stripe надав надійну систему оплати. Firebase допоміг у збереженні даних користувачів та аналітиці додатку.

PostgreSQL був вибраний для збереження даних через свою надійність та розширюваність, забезпечуючи ефективну роботу з великою кількістю даних.

Всі ці інструменти та технології обрані з метою забезпечення високоякісного та надійного продукту, що задовольняє потреби користувачів і відповідає завданням проекту. Вони сприяли успішному розвитку та впровадженню сервісу P2P доставки.

Розділ 3. Опис створення сервісу P2P доставки.

3.1. Створення нового iOS проекту на Swift та XCode.

По-перше, потрібно переконатись, що середовище розробки готове. Якщо не встановлено Xcode, потрібно встановіть його з офіційного AppStore. Для цього знадобиться Xcode версії 13 або новішої. Також необхідно переконатись що встановлено CocoaPods останніх версій.

Наступним кроком необхідно запустити Xcode, потім натисути "Створити новий проект Xcode" у вікні "Привітання з Xcode" або вибрати Файл > Створити > Проект. На сторінці, що з'явиться, потрібно вибрати цільову операційну систему або платформу і шаблон у розділі "Додаток". На наступних сторінках необхідно заповнити форми і вибрати опції для налаштування проекту.

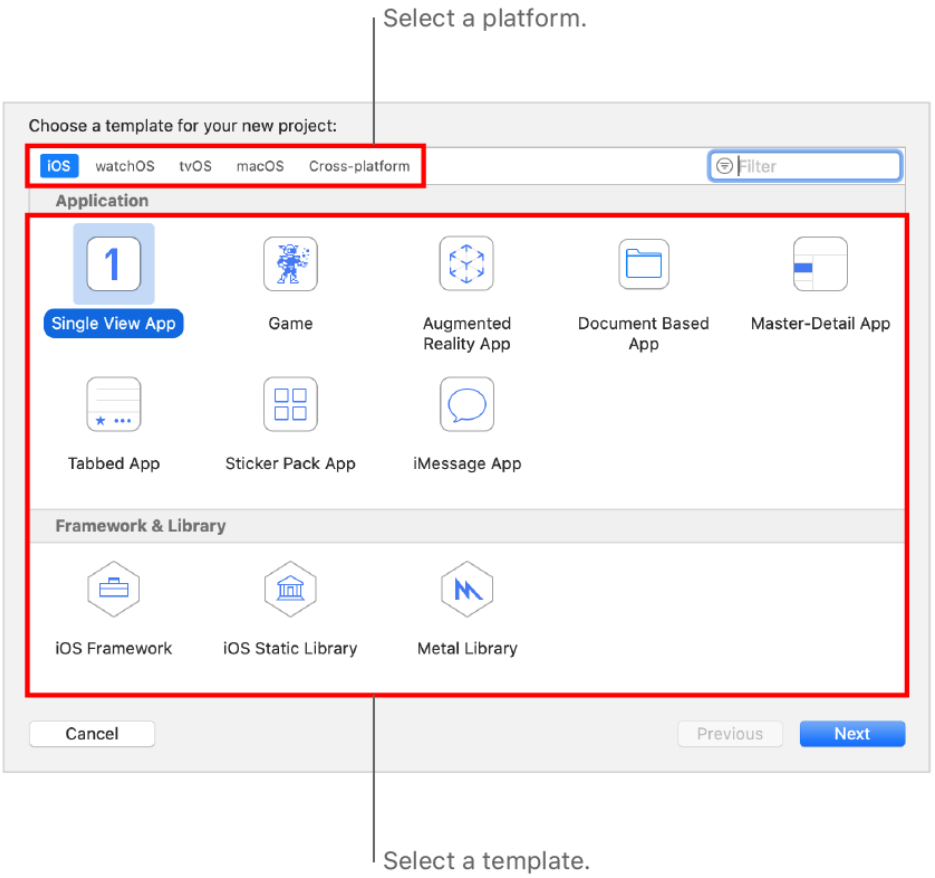


Рисунок 3.1.1 – Створення нового Xcode проекту.

Також необхідно вказати назву продукту та ідентифікатор організації, оскільки вони використовуються для створення ідентифікатора пакета, який ідентифікує програму в системі. Необхідно також ввести назву організації. Якщо немає е належності до організації, необхідно ввести своє ім'я.

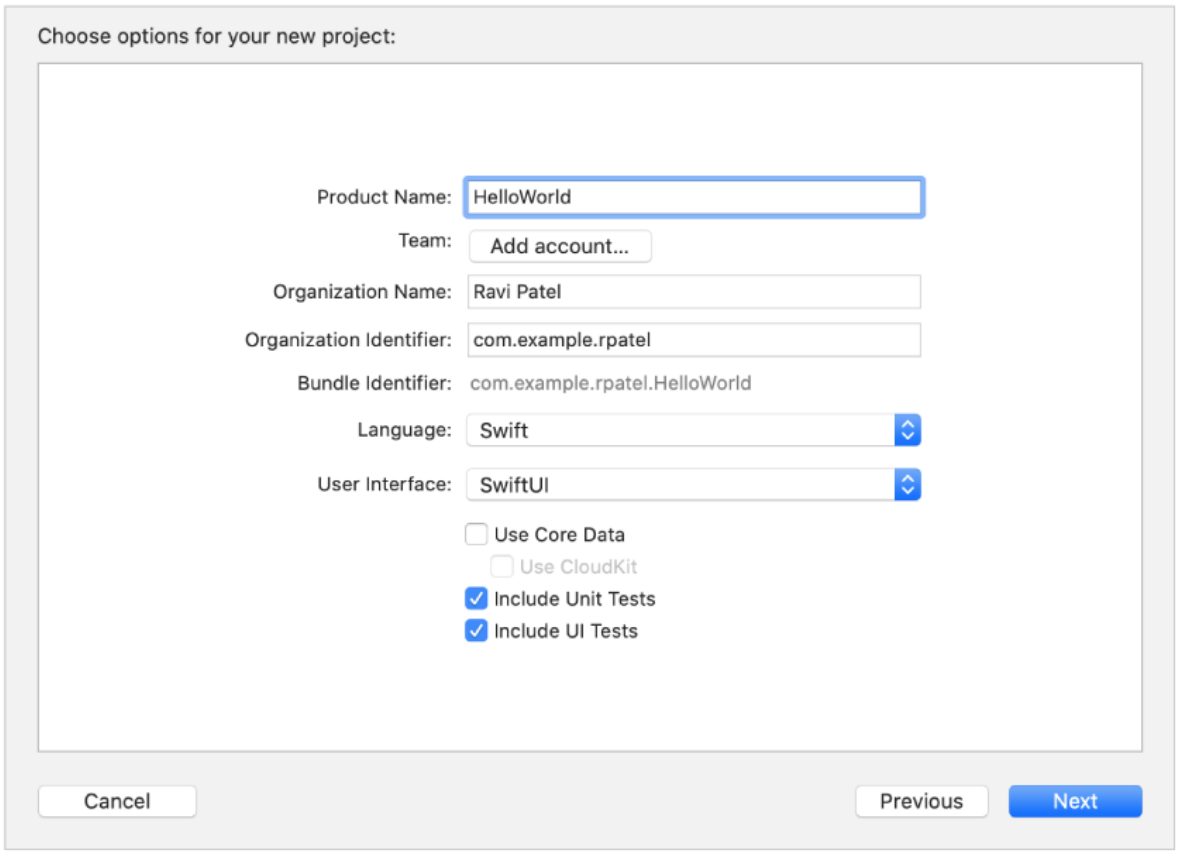


Рисунок 3.1.2 – Налаштування ідетифікатора Xcode проекту.

Коли створюється проект або відкривається існуючий проект, з'являється головне вікно, в якому відображаються файли і ресурси, необхідні для розробки додатку.

Є можливість отримати доступ до різних частин проекту за допомогою навігатора в головному вікні. Використовуючи навігатор проекту, щоб вибрати файли, які необхідно відредагувати в області редактора. Наприклад, якщо вибрати Swift-файл у навігаторі проекту, він відкриється у редакторі коду, де можно змінювати код і встановлювати точки зупинки.

Рисунок 3.1.3 – Керування файлами в головному вікні Xcode

Відомості про вибраний файл також з'являються в області інспектора праворуч. В області інспекторів є можливість вибрати інспектор атрибутів для редагування властивостей файлу або елемента інтерфейсу користувача. Якщо є необхідність приховати інспектор, щоб звільнити місце для редактора, потрібно натиснути кнопку "Приховати або показати інспектори" у верхньому правому куті панелі інструментів.

Панель інструментів використовується для створення і запуску програми на симульованому або реальному пристрої. Для iOS-додатків [4] необхідно виберати цільовий додаток і симулятор або пристрій у меню призначення запуску на панелі інструментів, а потім натиснути кнопку "Запустити".

Щоб змінити властивості, введені під час створення проекту, виберіть назву проекту в навігаторі проекту, що з'явиться вгорі, після чого в області редагування відкриється редактор проекту. Більшість введених властивостей відображаються на панелі "Загальні" редактора проєкту.

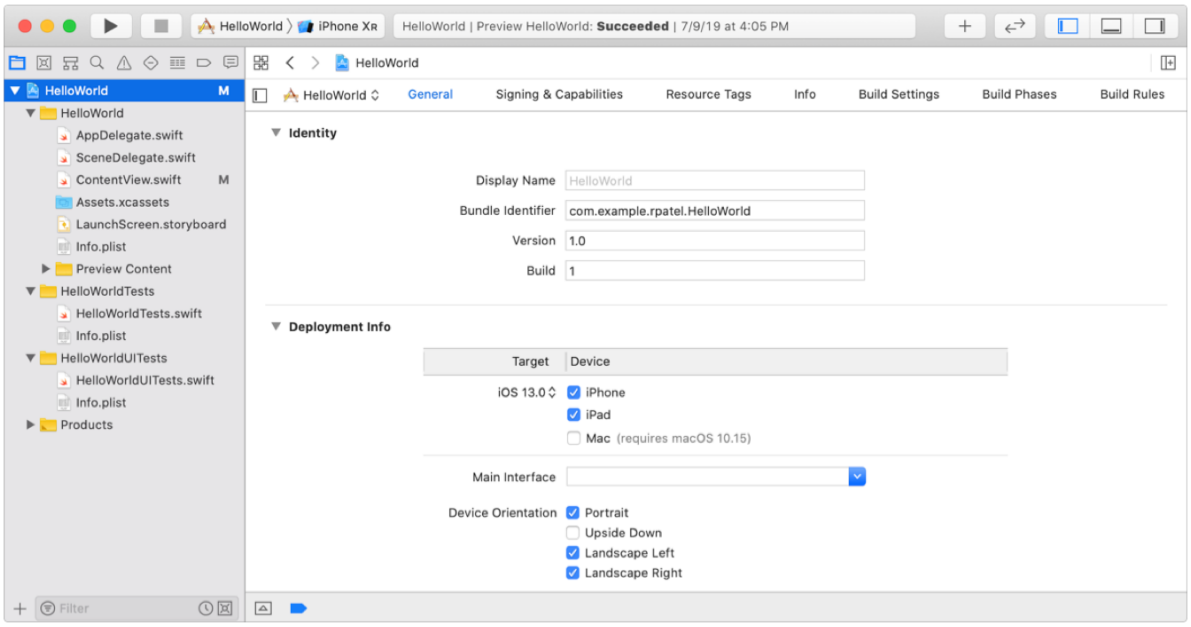


Рисунок 3.1.4 – Зміна налаштувань проекту Xcode

3.2. Створення нового бекенд-проекту на Django [1].

Необхідно переконайтеся, що встановлено Django [1] на комп'ютері. Якщо цього не робили, то можно встановити його за допомогою pip, командою:

```
pip install django
```

Рисунок 3.2.1 – Команда встановлення Django [1]

Далі потрібно відкрити термінал і перейти до папки, де буде міститись створений проект. Виконайте команду Django [1] для створення нового проекту, де "myproject" - це назва проекту:

```
django-admin startproject myproject
```

Рисунок 3.2.2 – Команда створення нового Django [1] проекту

Django [1] створить папку "myproject", в якій буде розміщена основна структура проекту. Вона буде містити файли та папки, необхідні для розробки вашого бекенду.

Django [1] використовує концепцію "додатків" для організації коду. Створити новий додаток командою можно командою:

Рисунок 3.2.3 – Команда створення нового додатку

Де "myapp" - це назва додатку.

Налаштування бази даних. В файлі settings.py проекту є можливість налаштувати параметри бази даних, яка буде використовуватись.

Створення моделей: бекенд буде включати моделі, які визначають структуру даних додатку. Можно створити моделі в файлі додатку models.py і використовувати міграції для створення таблиць в базі даних.

Створення URL-шляхів та видів: Визначте URL-шляхи та види для додатку, щоб обробляти HTTP-запити та відображати вміст на стороні сервера.

Розробка бізнес-логіки: Розробка бізнес-логіки додатку, включає обробку запитів, взаємодію з базою даних, автентифікацію та авторизацію користувачів.

Створення адмін-панелі: Django [1] надає готову адміністративну панель, яку можна налаштовувати для управління даними вашого додатку.

Запуск сервера відбувається командою:

Рисунок 3.2.4 – Команда запуску сервера

Результатом виконання команди буде URL, за яким можна буде звертатися до додатку.

Це загальний процес створення нового бекенд-проекту на Django [1]. Подальшими кроками можно розширювати та налаштовувати проект відповідно до потреб.

3.3. API Документація

Для створення API документації з використанням Swagger для Django [1], можно використовувати бібліотеку ***drf-yasg*** (Yet Another Swagger Generator). Ось кроки для налаштування Swagger документації для Django [1] проекту:

1. **Необхідно встановіть drf-yasg**:

   Встановлення бібліотеки drf-yasg за допомогою pip:

Рисунок 3.3.1 – Встановлення drf-yasg

2. **Додавання drf-yasg до INSTALLED_APPS**:

   В файлі налаштувань Django [1] (settings.py) необхідно додати drf-yasg до INSTALLED_APPS:

Рисунок 3.3.2 – Налаштування INSTALLED_APPS

3. **Налаштування Swagger для API**:

У файлі urls.py, необхідно визначите URL-шлях для Swagger документації. Потрібно вставите такий код:

```
from django.contrib import admin
from django.urls import path, re_path, include
from rest_framework import permissions
from drf_yasg.views import get_schema_view
from drf_yasg import openapi

schema_view = get_schema_view(
    openapi.Info(
    title="Your API",
    default_version='v1',
    description="Your API description",
    terms_of_service="https://www.yourapp.com/terms/",

contact=openapi.Contact(email="contact@yourapp.com"),
    license=openapi.License(name="Your License"),
    ),
    public=True,
    permission_classes=(permissions.AllowAny,),
)

urlpatterns = [
    path('admin/', admin.site.urls),
```

```
    path('api/', include('yourapp.urls')),  # Замість
'yourapp.urls' підставте свій URL
    re_path(r'^swagger(?P<format>\.json|\.yaml)$',
schema_view.without_ui(cache_timeout=0),
name='schema-json'),
    path('swagger/', schema_view.with_ui('swagger',
cache_timeout=0), name='schema-swagger-ui'),
]
```

Замість 'yourapp.urls' потрібно вставити URL проекту, де визначено маршрути для API.

4. **Запуск серверу Django** [1]:

Запустити сервер Django [1] можно, наприклад, виконавши команду:

Рисунок 3.3.3 – Запуск Django [1] сервера

5. **Перейхід до Swagger документації**:

   Для переходу необхідно відкрийте веб-браузер і перейти за адресою, де запущений сервер (зазвичай http://127.0.0.1:8000/swagger/). Там буде можливість побачите Swagger документацію для Django [1] API.

Тепер можно використовувати Swagger для опису та тестування Django [1] API. Також можно налаштовувати додаткові параметри та інформацію про API в налаштуваннях Swagger в коді.

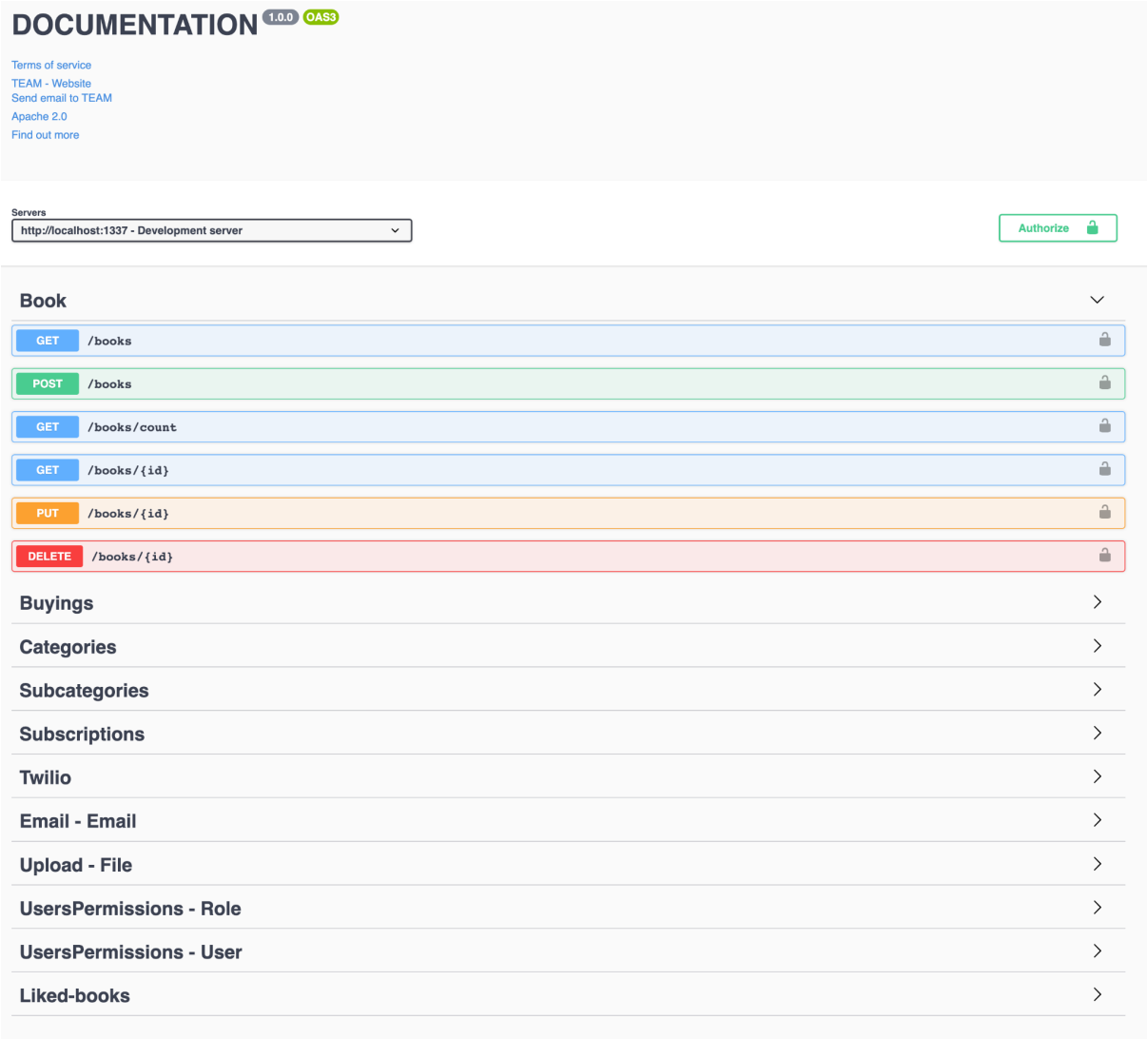


Рисунок 3.3.1 – Документація Swagger UI.

3.4. Функціонал сервісу.

3.4.1. Процес реєстрації/авторизації.

Kengu надає такі можливості:

1. Зареєструвати користувача.
2. Зайти в систему з логіном та паролем або через Facebook провайдера.
3. “Забули пароль” сторінка.

4. Налаштування профайлу користувача.

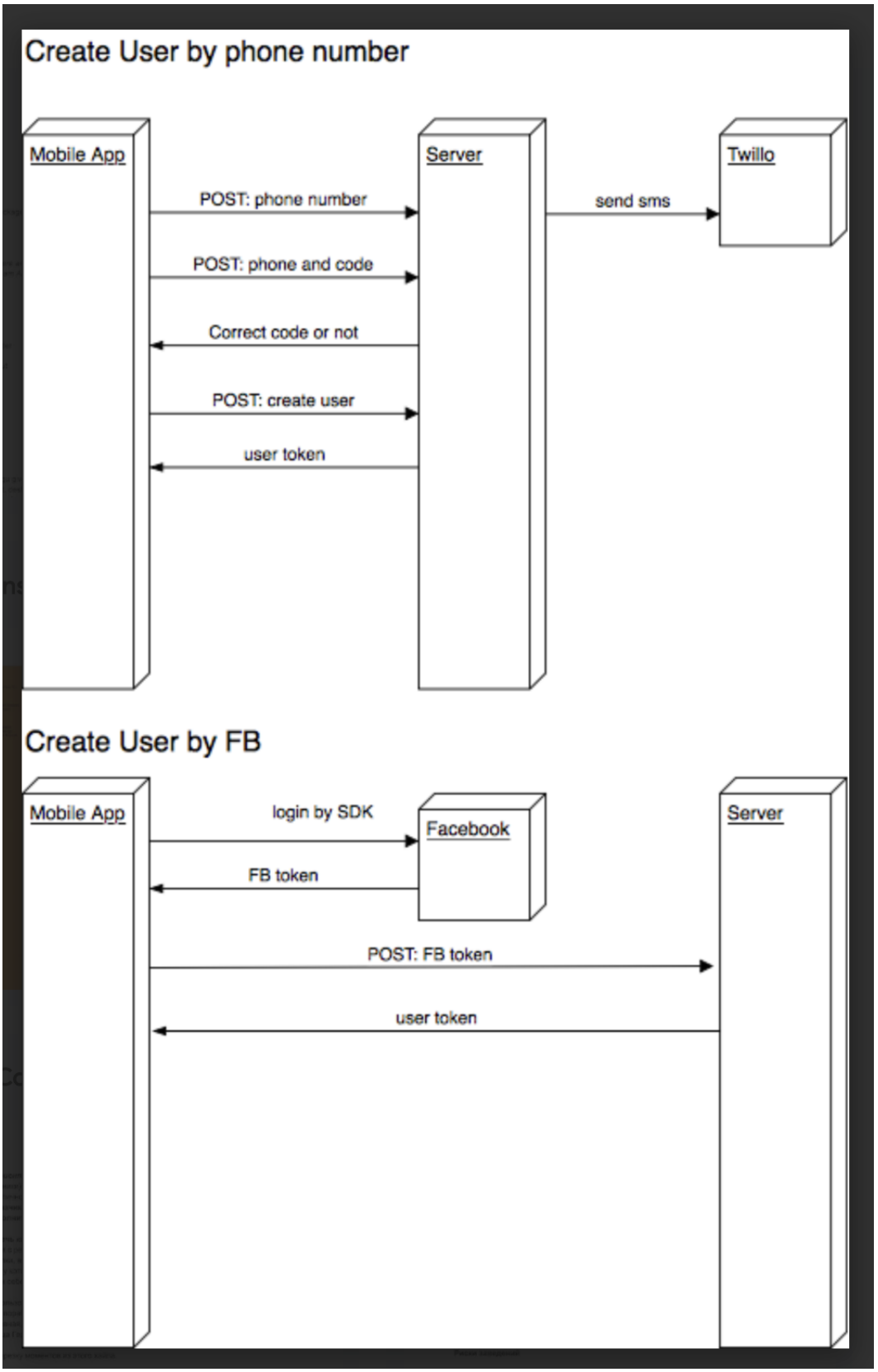
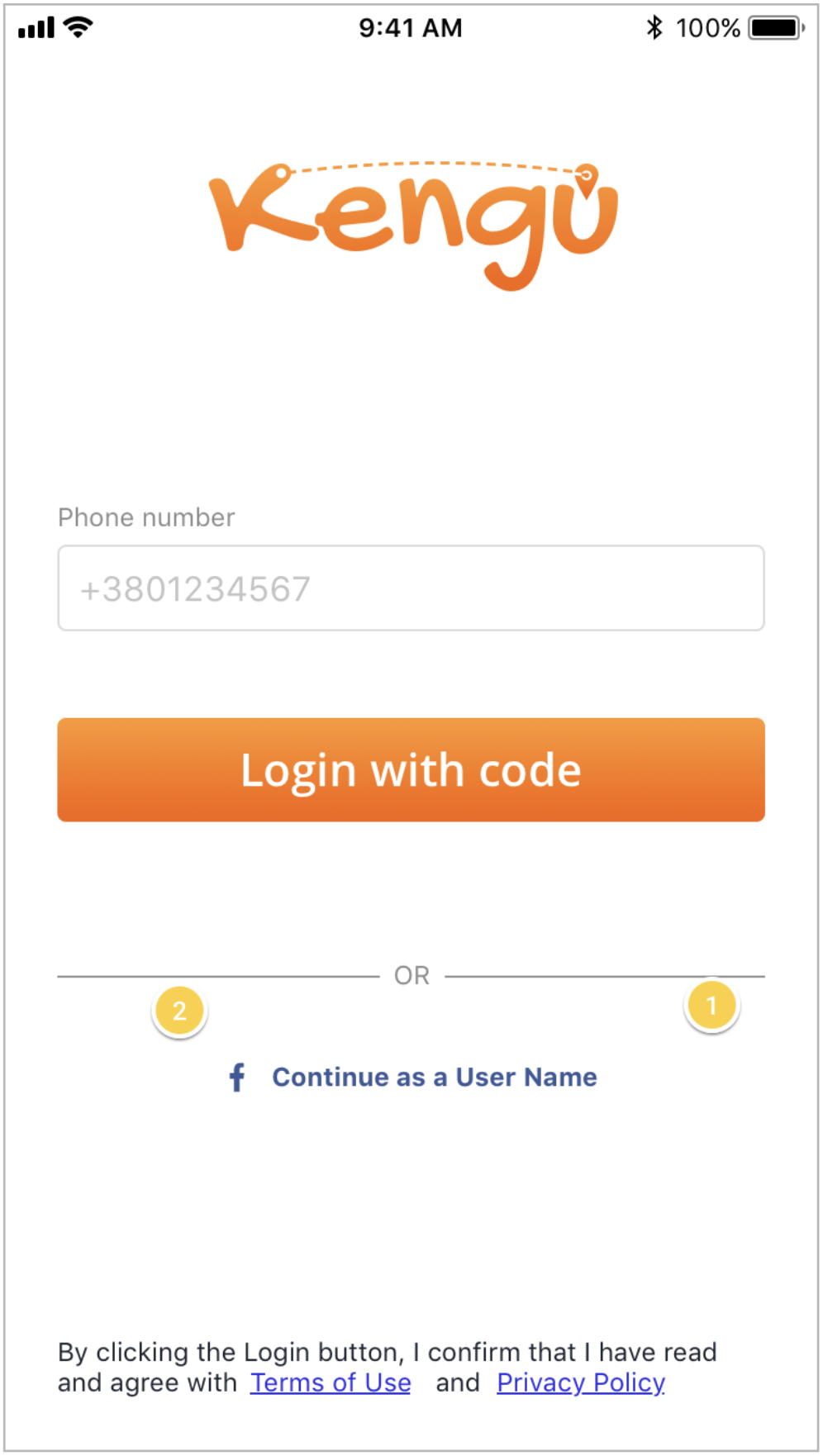
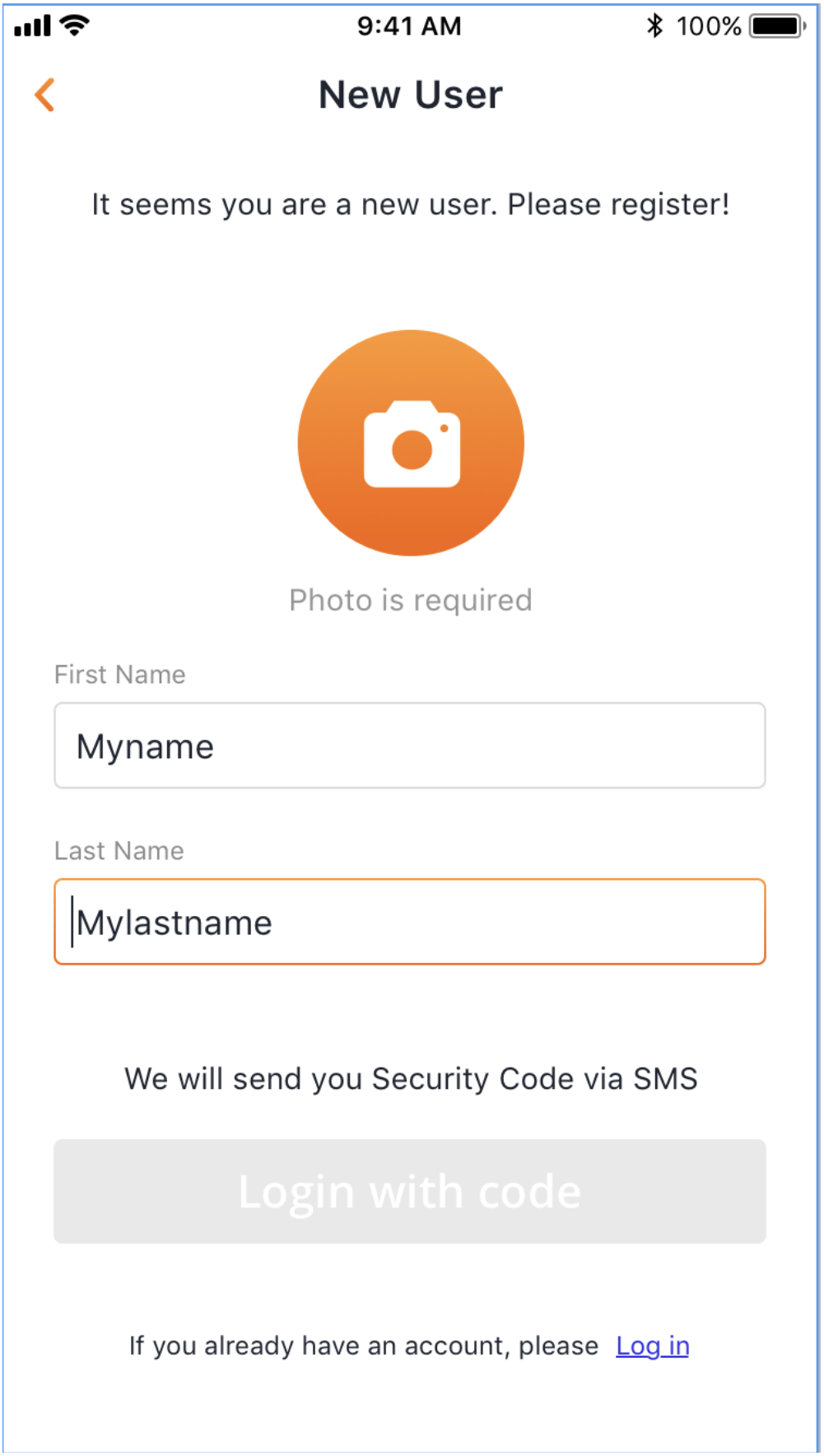


Рисунок 3.4.1.1 – Створення користувача в системі



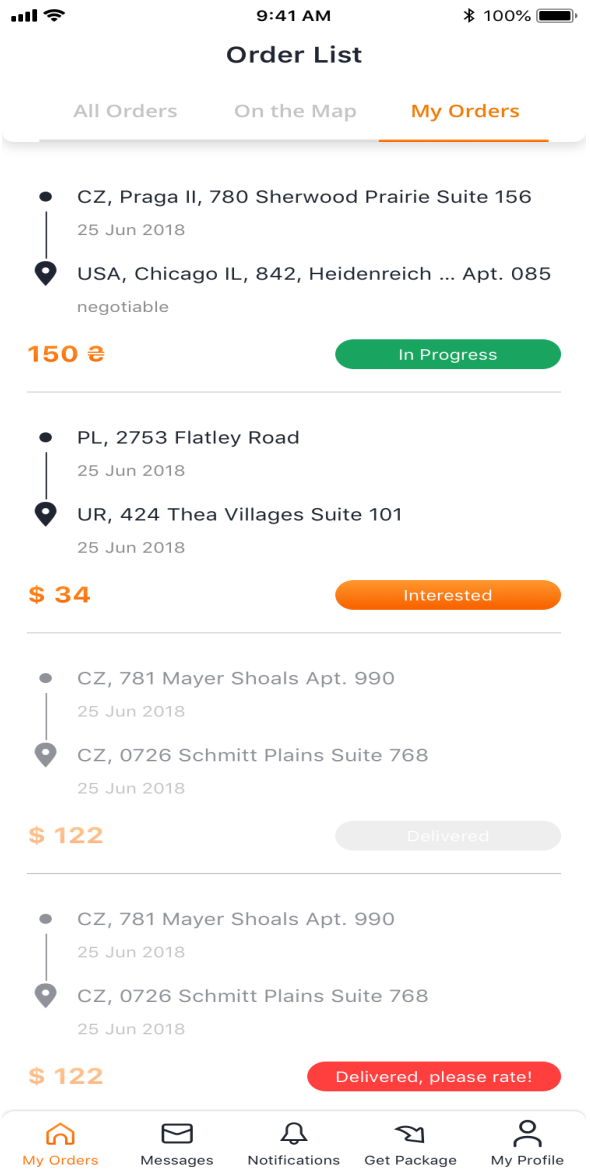
3.4.1.2 – UI Реєстрації користувача в системі



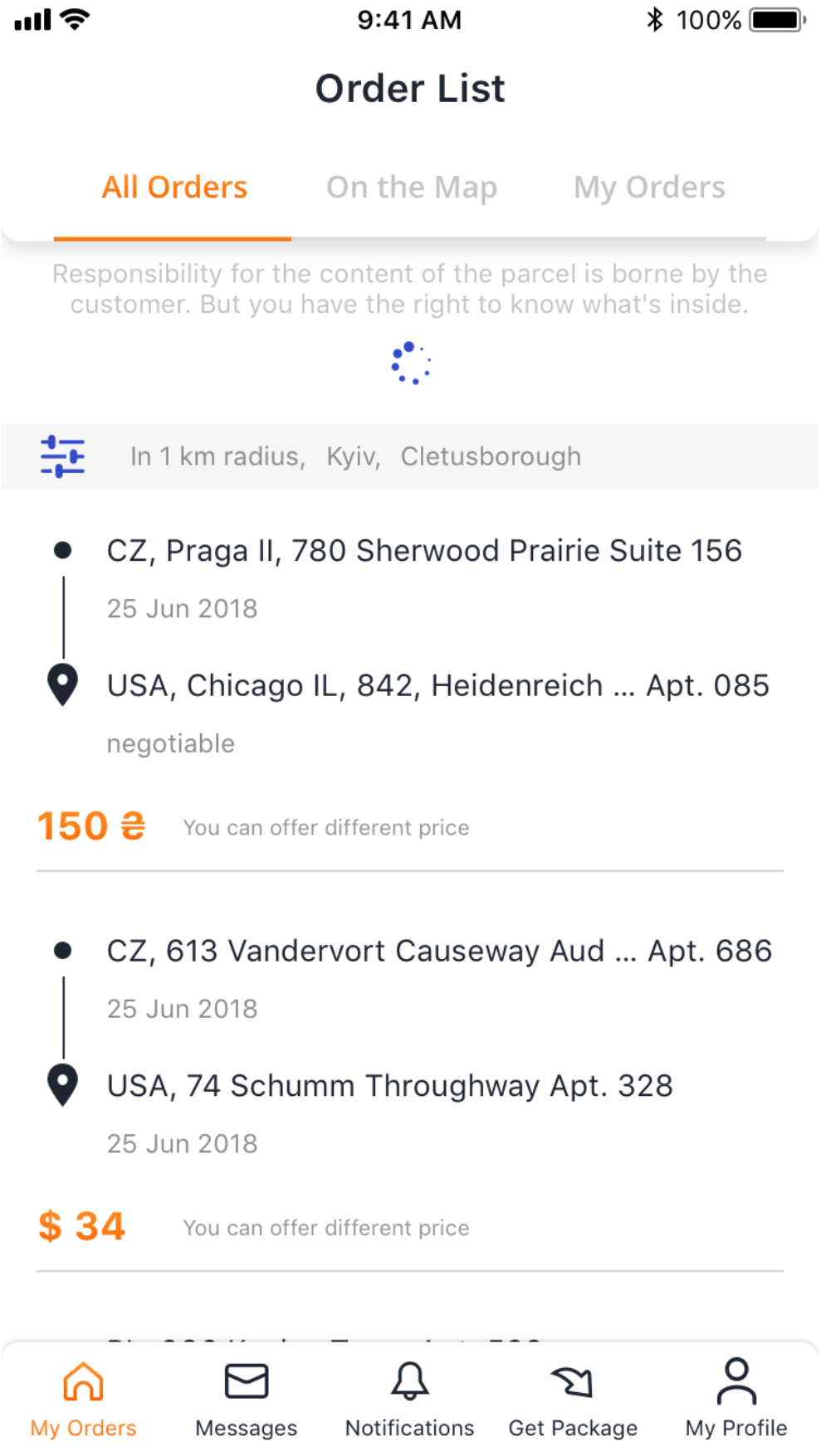
3.4.1.3 – UI Заповнення інформації користувача в системі

3.4.2. Сторінка всіх замовлень доступним до виконання кур'єрам.

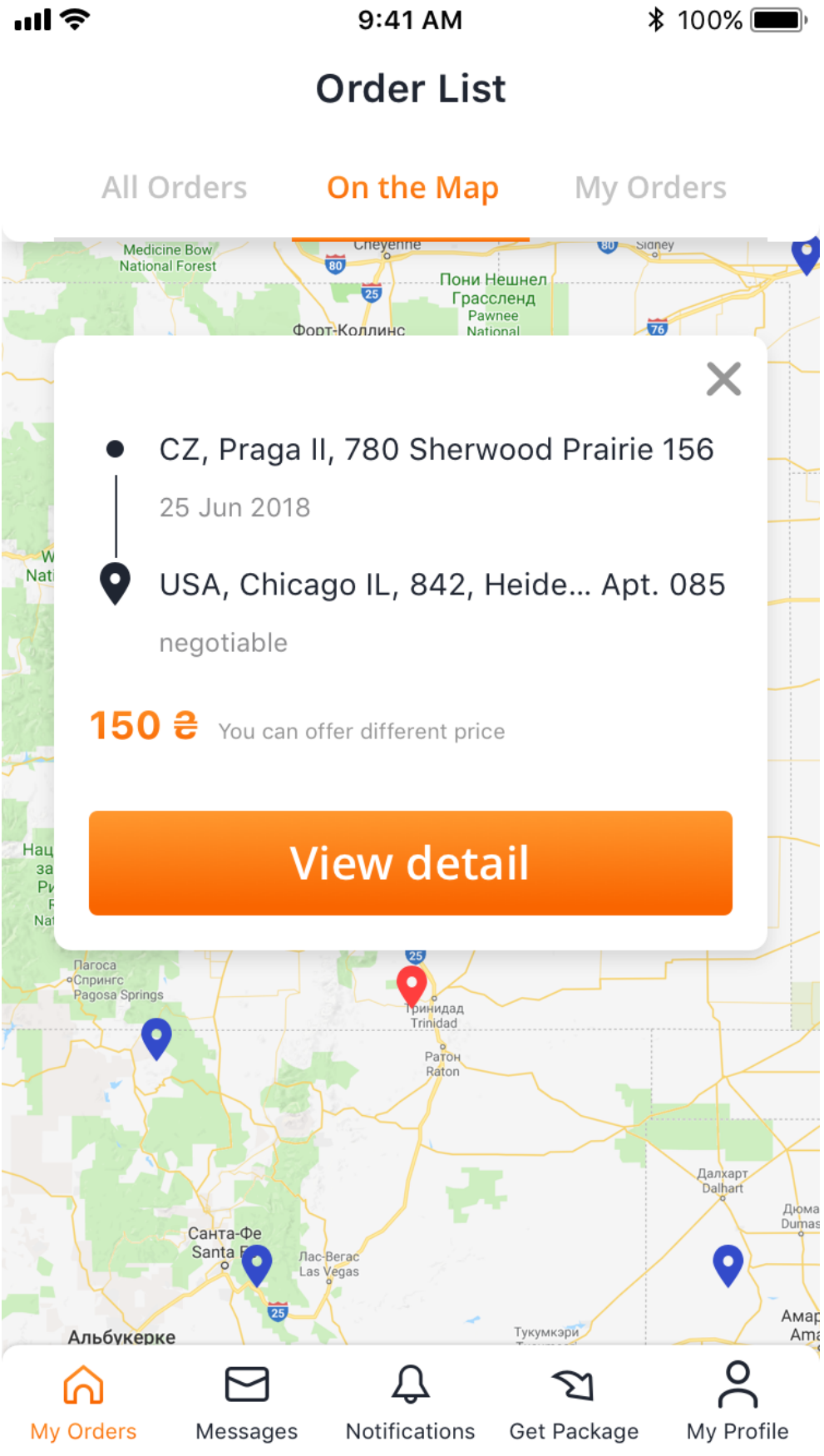
На цій сторінці користувач з ролью кур'єра має можливість отримати актуальну інформацію про всі доступні замовлення, також є можливість використати фільтр по відстані, містах, переглянути можливість відобразити замовлення списом або на карті.



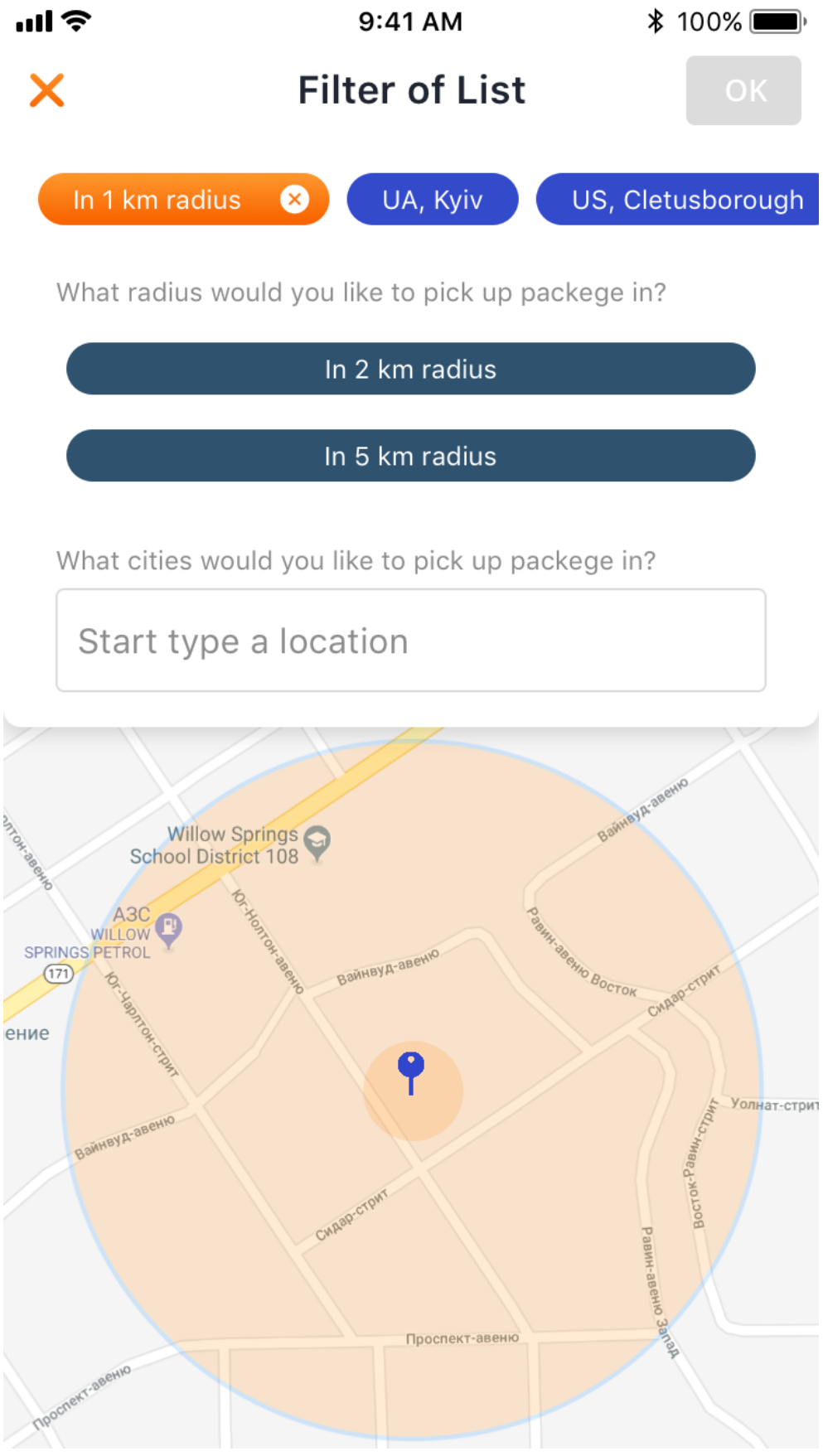
3.4.2.1 – UI відображення замовлень які виконує кур'єр.



3.4.2.2 – UI відображення всіх доступних замовлень.



3.4.2.3 – UI відображення замовлень на карті.

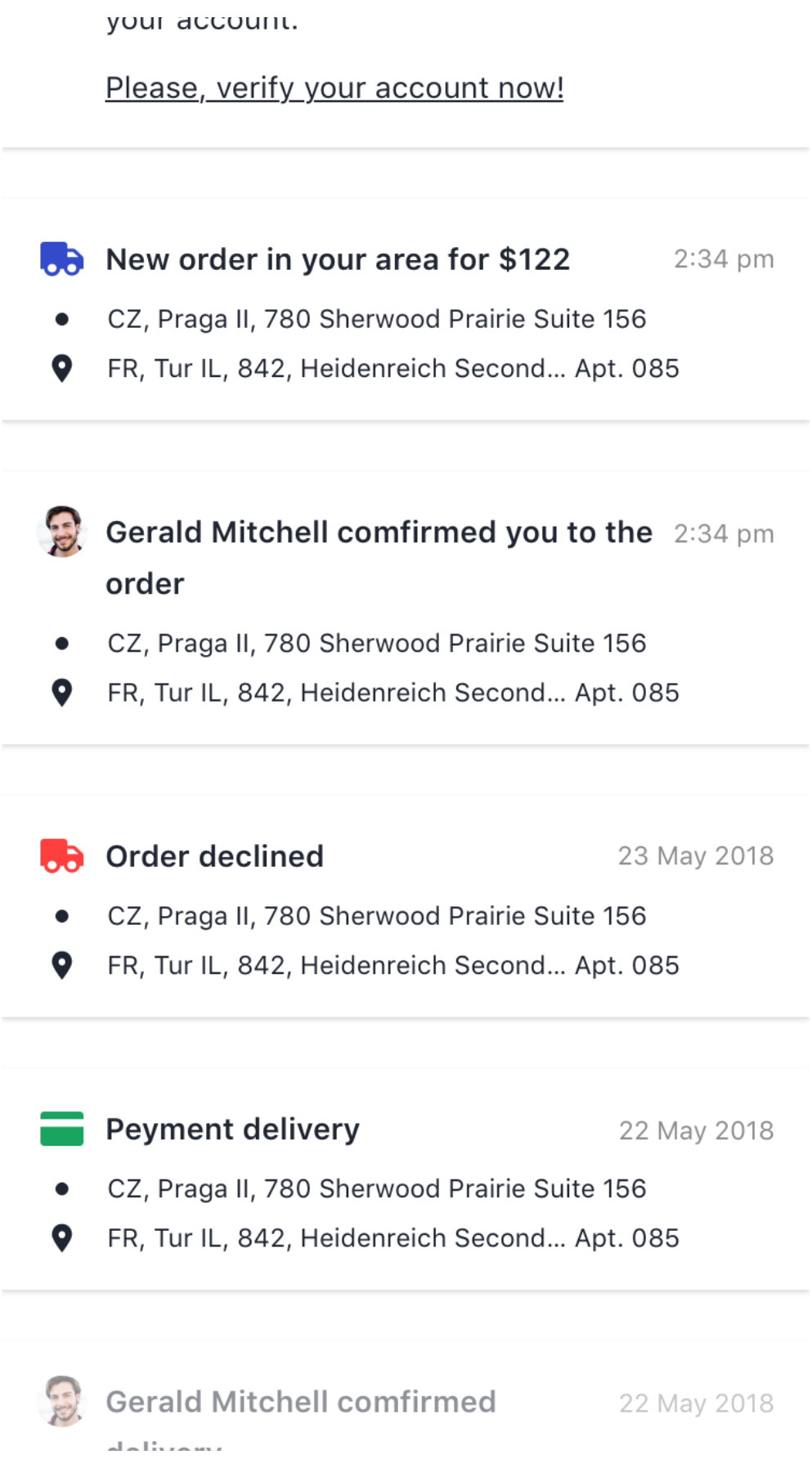


3.4.2.4 – UI фільтрування замовлень.

3.4.3. Нотифікаці про події на сервісі.

На сторінку нотифікацій доступні всі нотифікації які доступні в системі. Тут зібрані різні типи нотифікацій, такі як:

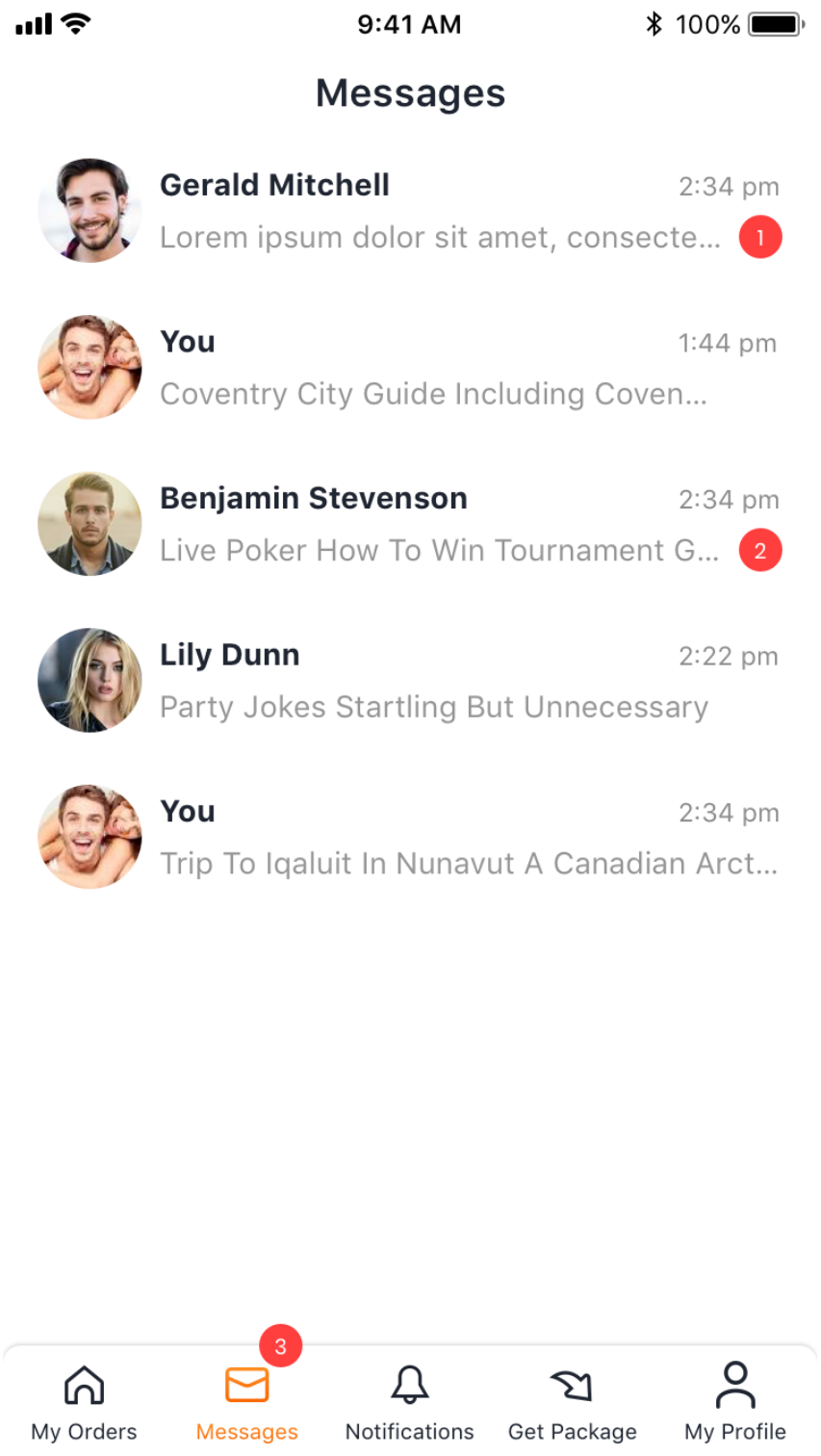
1. Нотифікації про акаунт користувача
2. Нотифікації що стосуються створення нових замовлень
3. Нотифікації про відгуки на замовлення від кур'єрів
4. Нотифікації про зміни статусу замовлень
5. Нотифікації про оплату
6. Нотифікації про відкуги та рейтинг



3.4.3 – Екран нотифікацій.

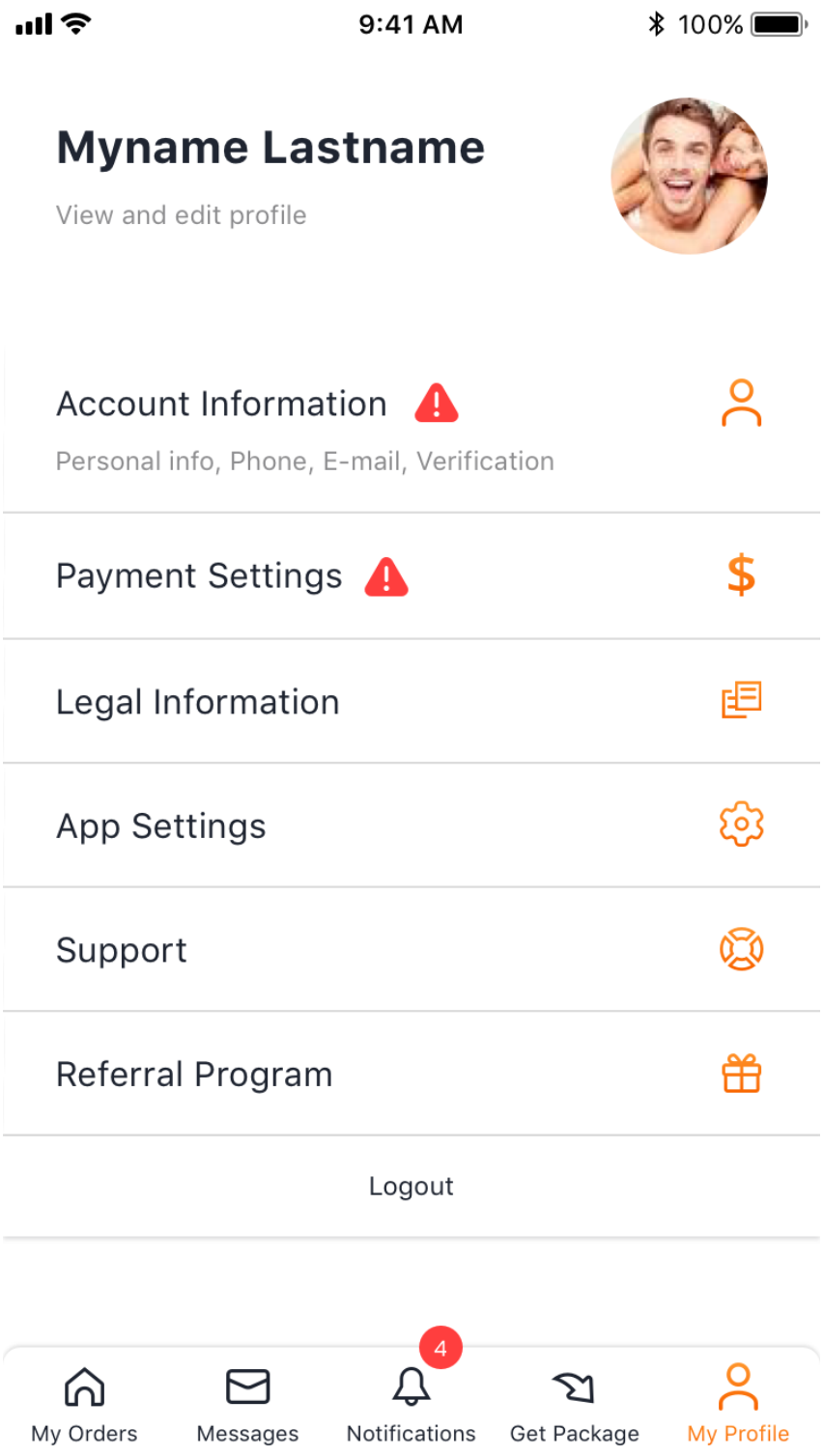
3.4.4. Мовідомлення між користувачами.

Користувачі маєть можливість обмінюватись повідомленнями в середині сервісу.



3.4.4 – Екран повідомлень.

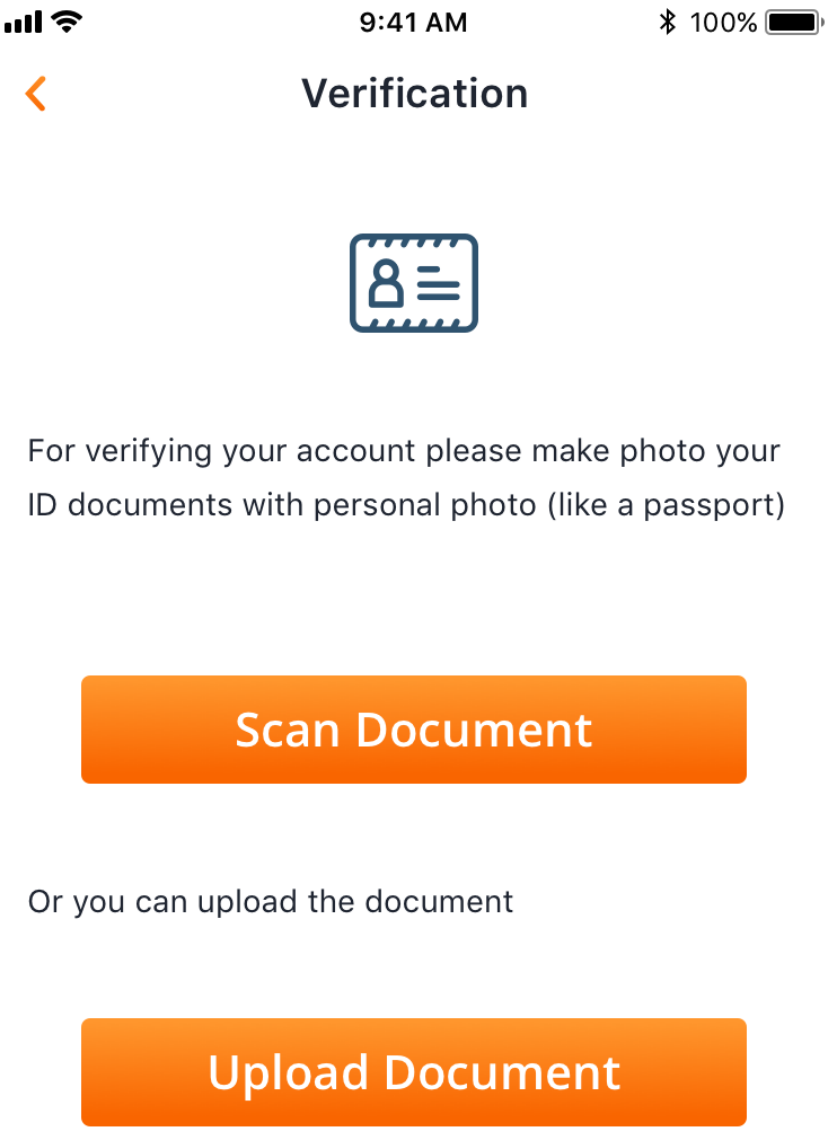
3.4.5. Профіль користувача.



3.4.5.1 – Екран профілю користувача.

Авторизований користувач має наступні можливості такі як:

1. Підтвердити свій профіль



3.4.5.2 – Екран підтвердження користувача.

2. Додати свої платіжні реквізити

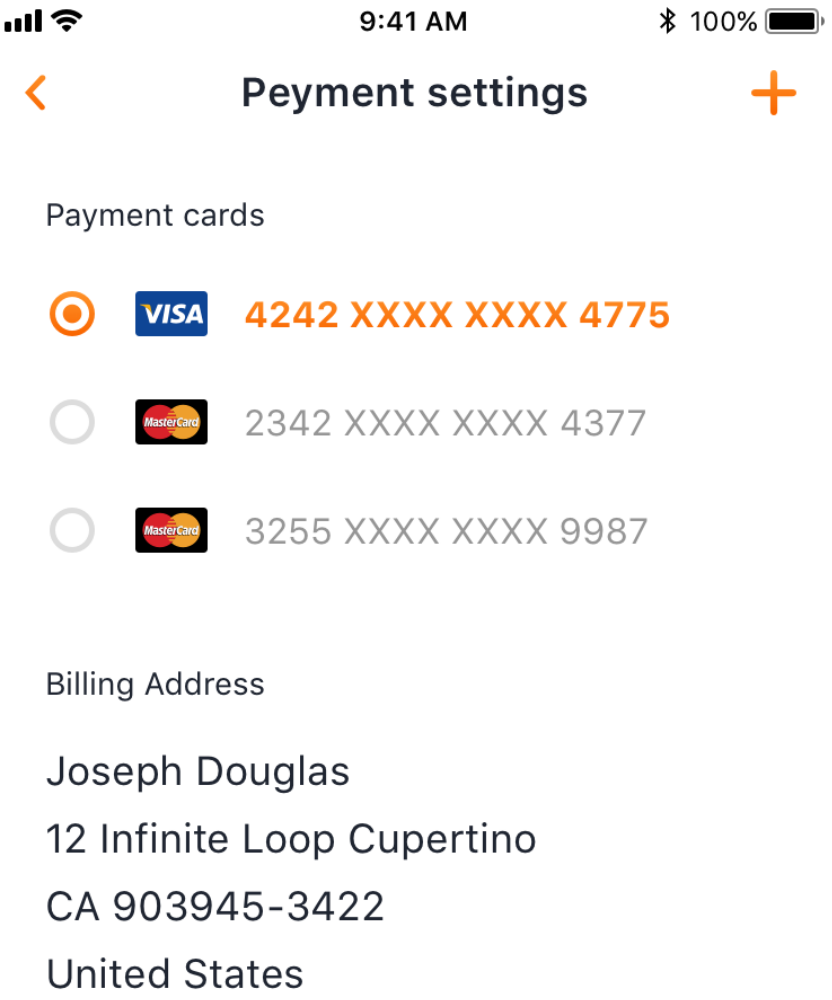


Рисунок 3.4.5.3 – Екран добавлення платіжної інформації користувача.

3. Переглянути юридичну інформацію про сервіс та відповідальність кожної із сторін

4. Налаштувати можливість отримання нотифікацій, пофідомлень
5. Можливість зв'язатися з технічною підтримкою
6. Редагування профілю

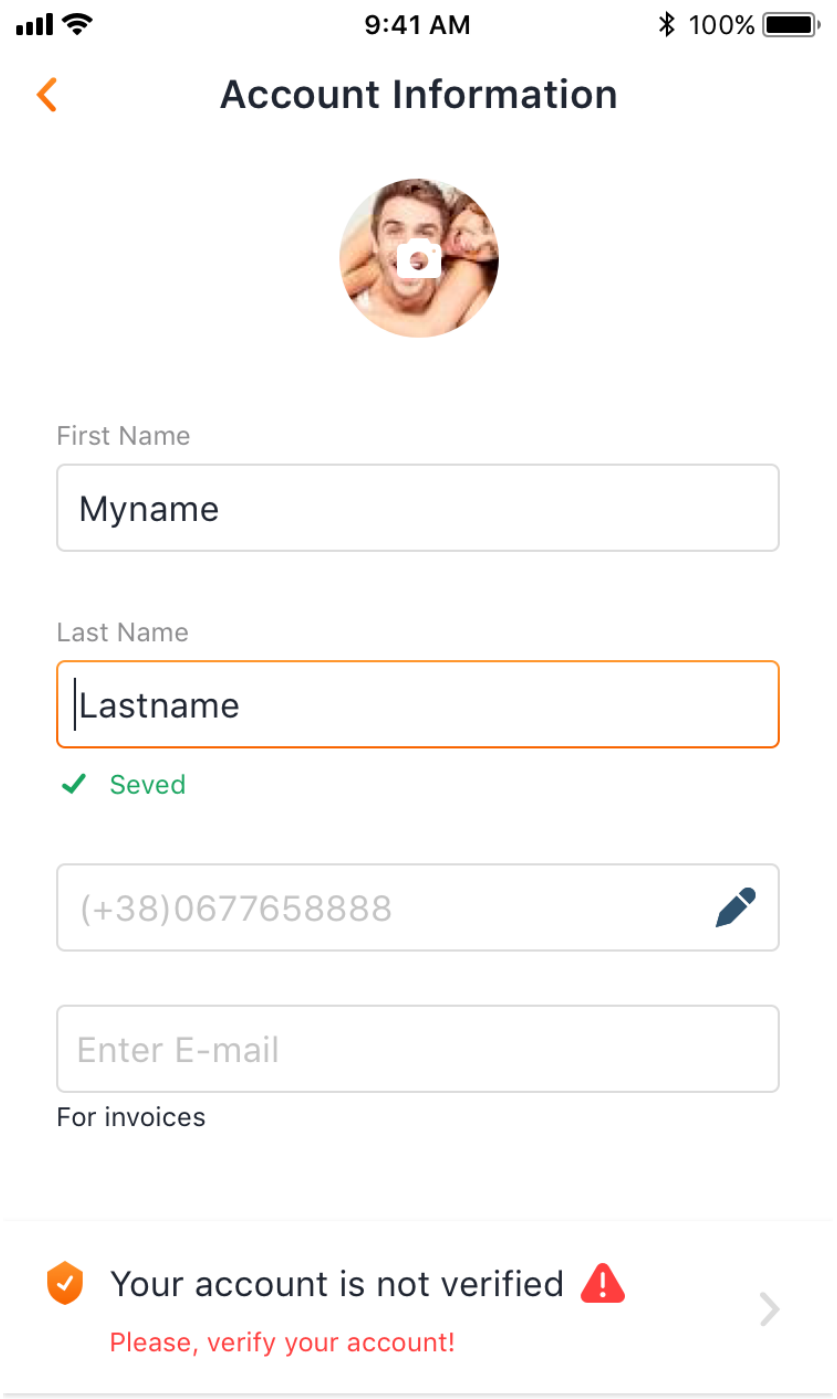


Рисунок 3.4.5.4 – Екран редагування профілю користувача.

7. Реферальна програма

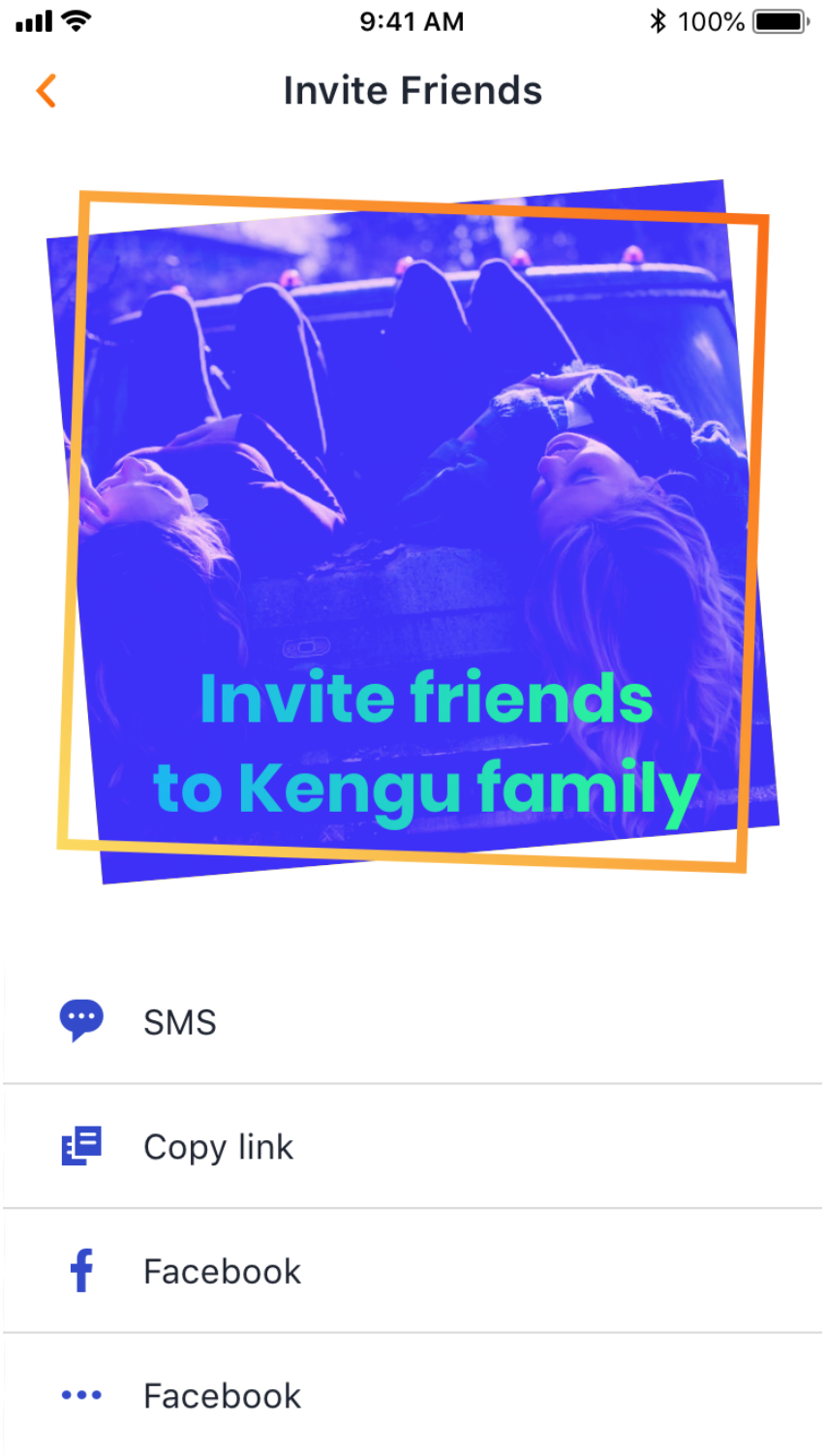


Рисунок 3.4.5.5 – Екран реферальної програми.

3.4.6. Створення нового замовлення та відгуки на нього.

Користувач якому потрібно отримати доставку має можливість самостійно створити замовлення де вкаже всю інформацію по даному замовленню, а кур'єри, які зацікавлені в виконанні замовлення матимуть змогу залишити свою заявку на виконання замовлення.

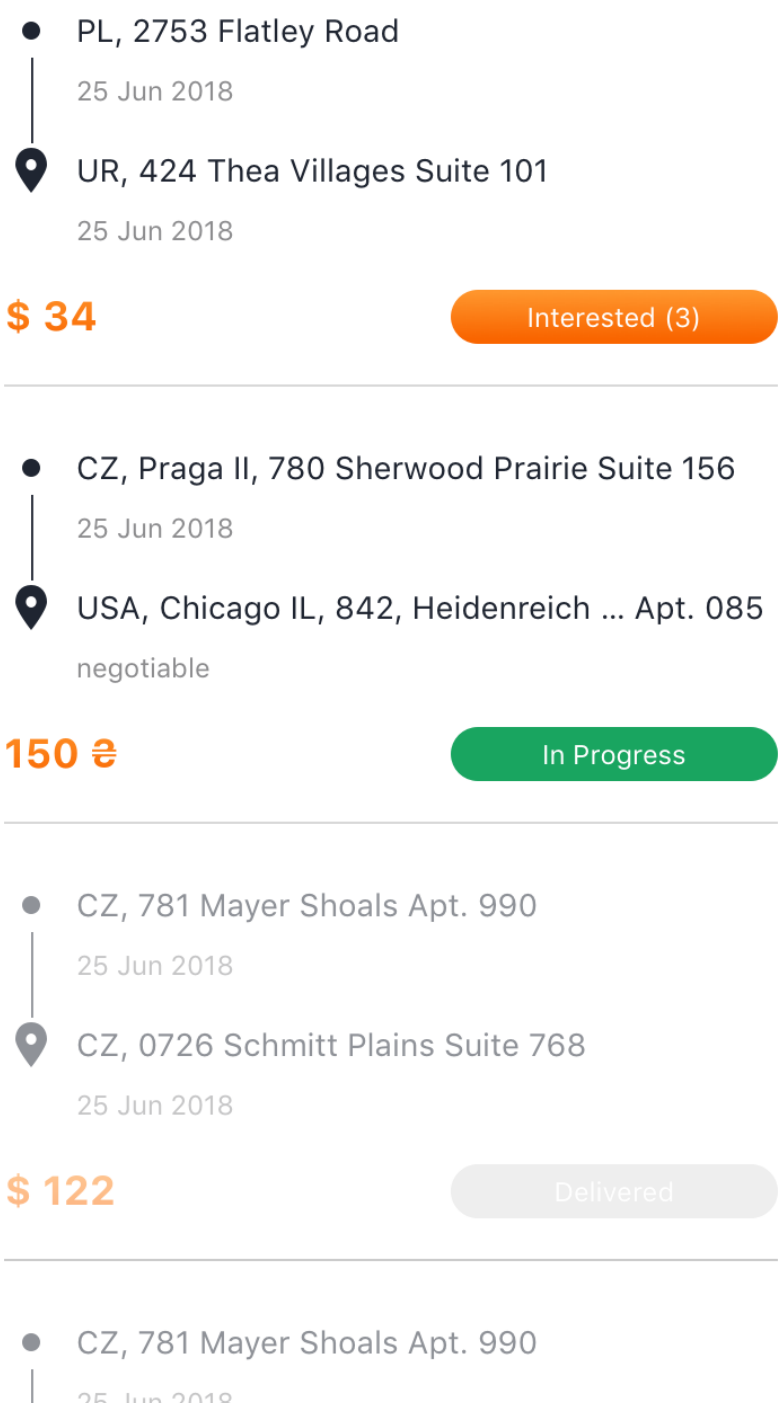


Рисунок 3.4.6.1 – Екран замовлень кур'єра і переходу до створення нового замовлення.

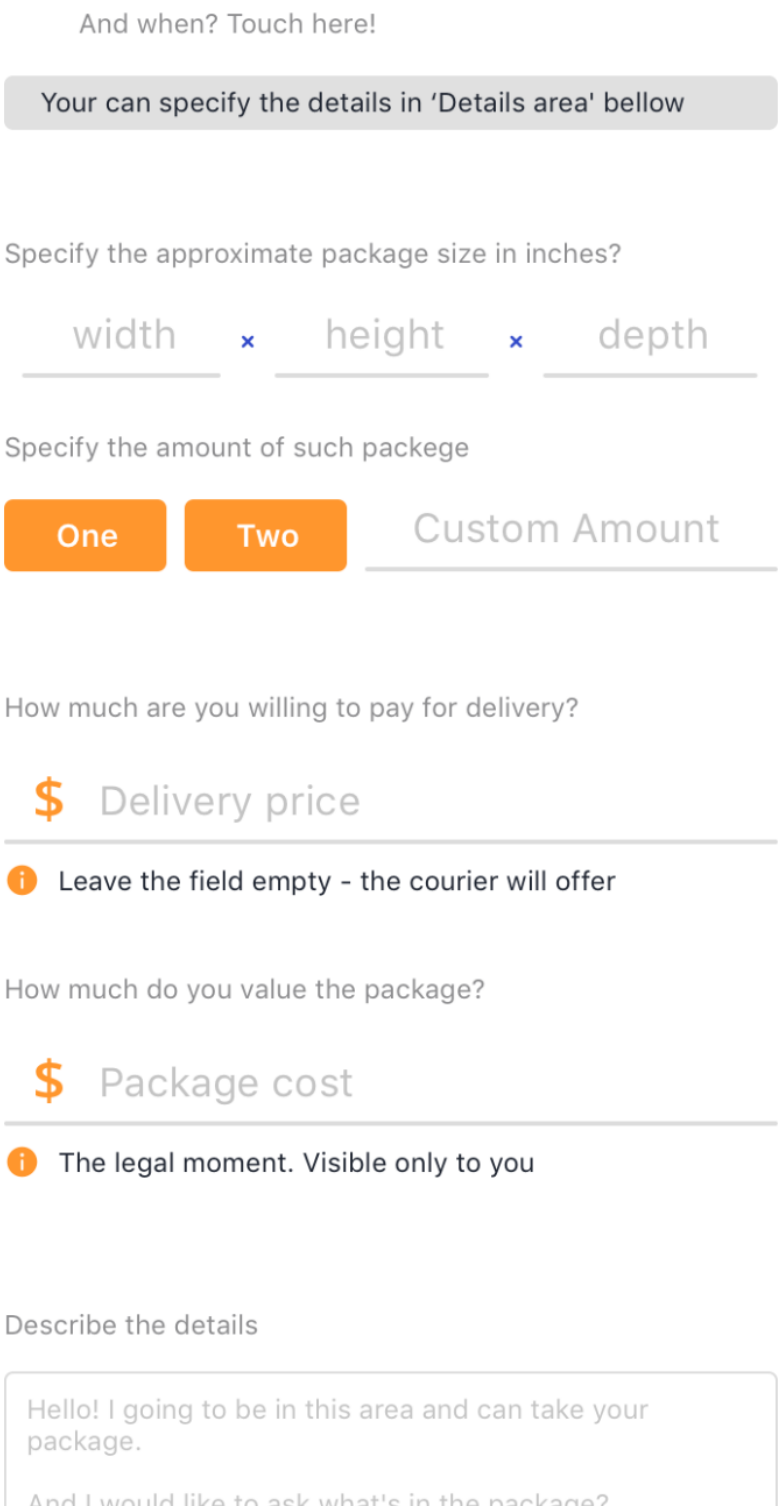


Рисунок 3.4.6.2 – Екран створення нового замовлення.

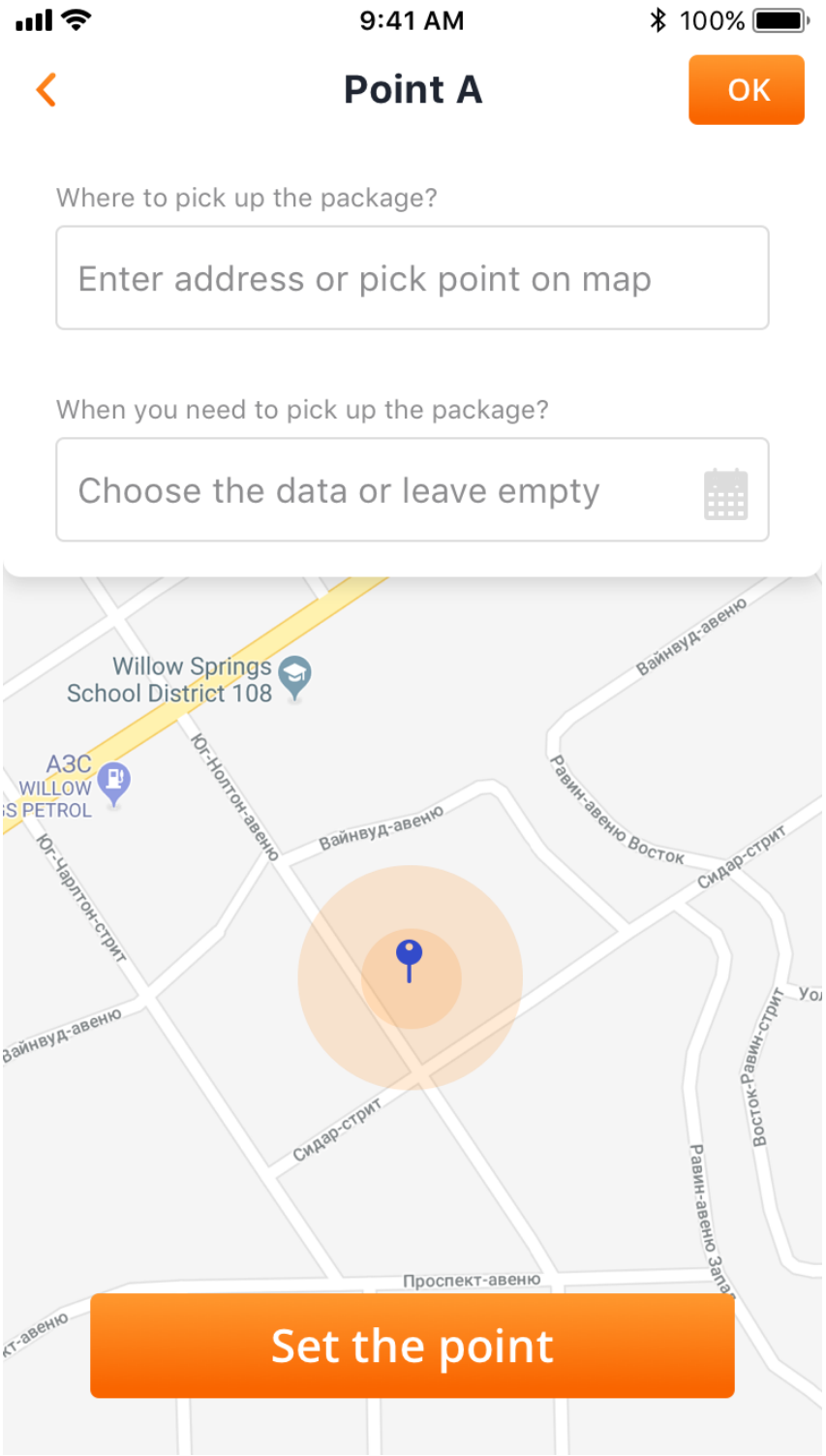


Рисунок 3.4.6.3 – Вибір адреси і часу.

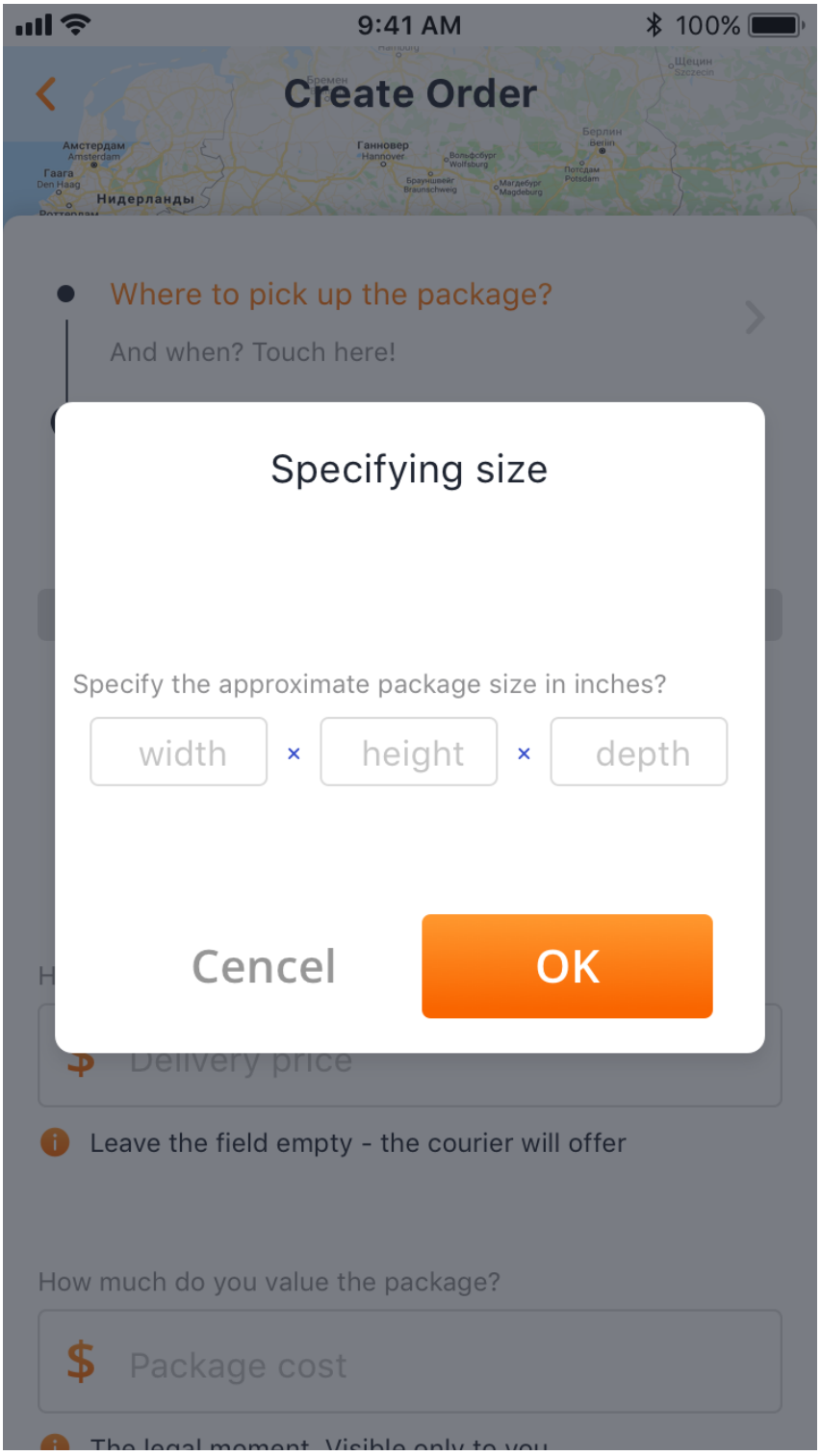


Рисунок 3.4.6.4 – Екран вказання розмірів замовлення.

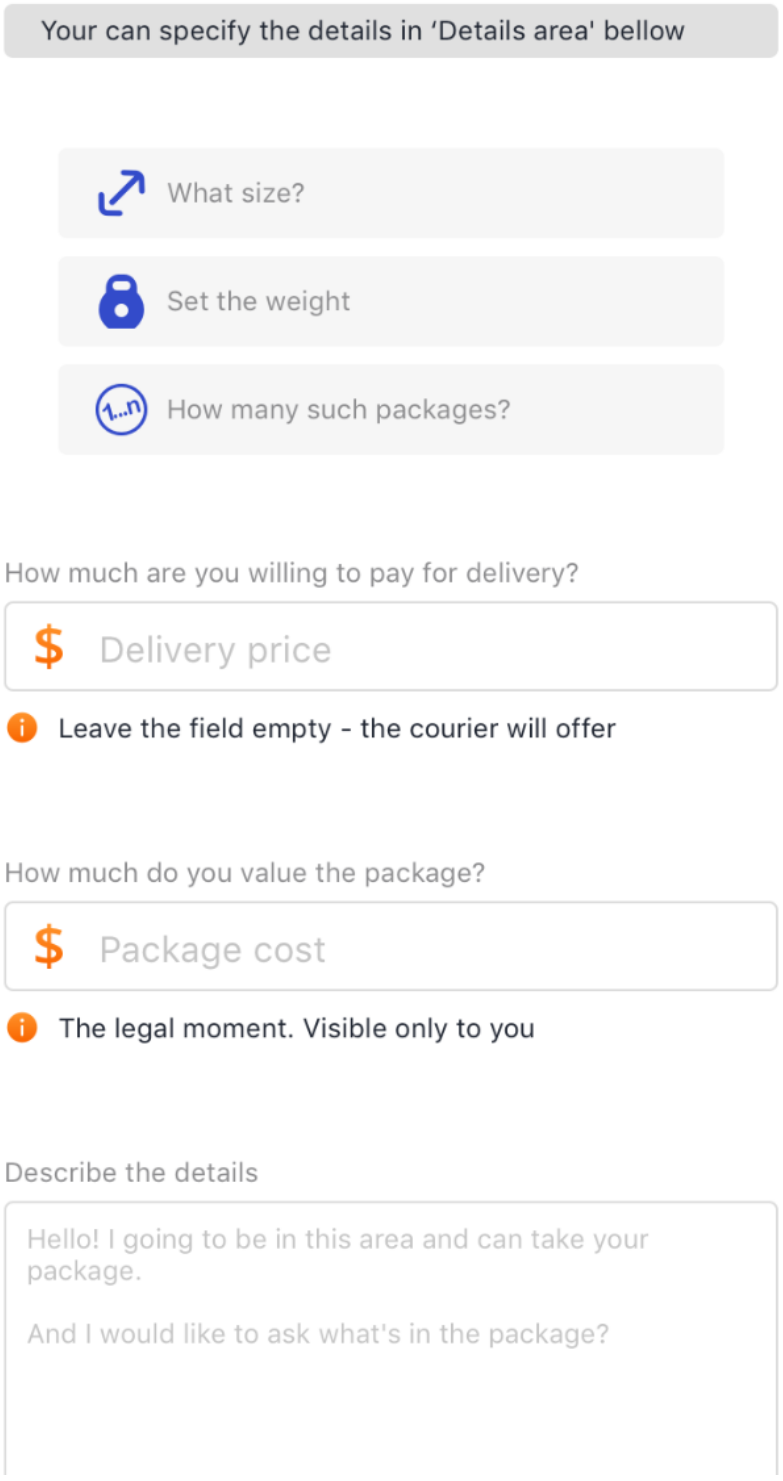


Рисунок 3.4.6.5 – Екран дає можливість вказати ціну за доставку а також вказати ціну страхування замовлення.

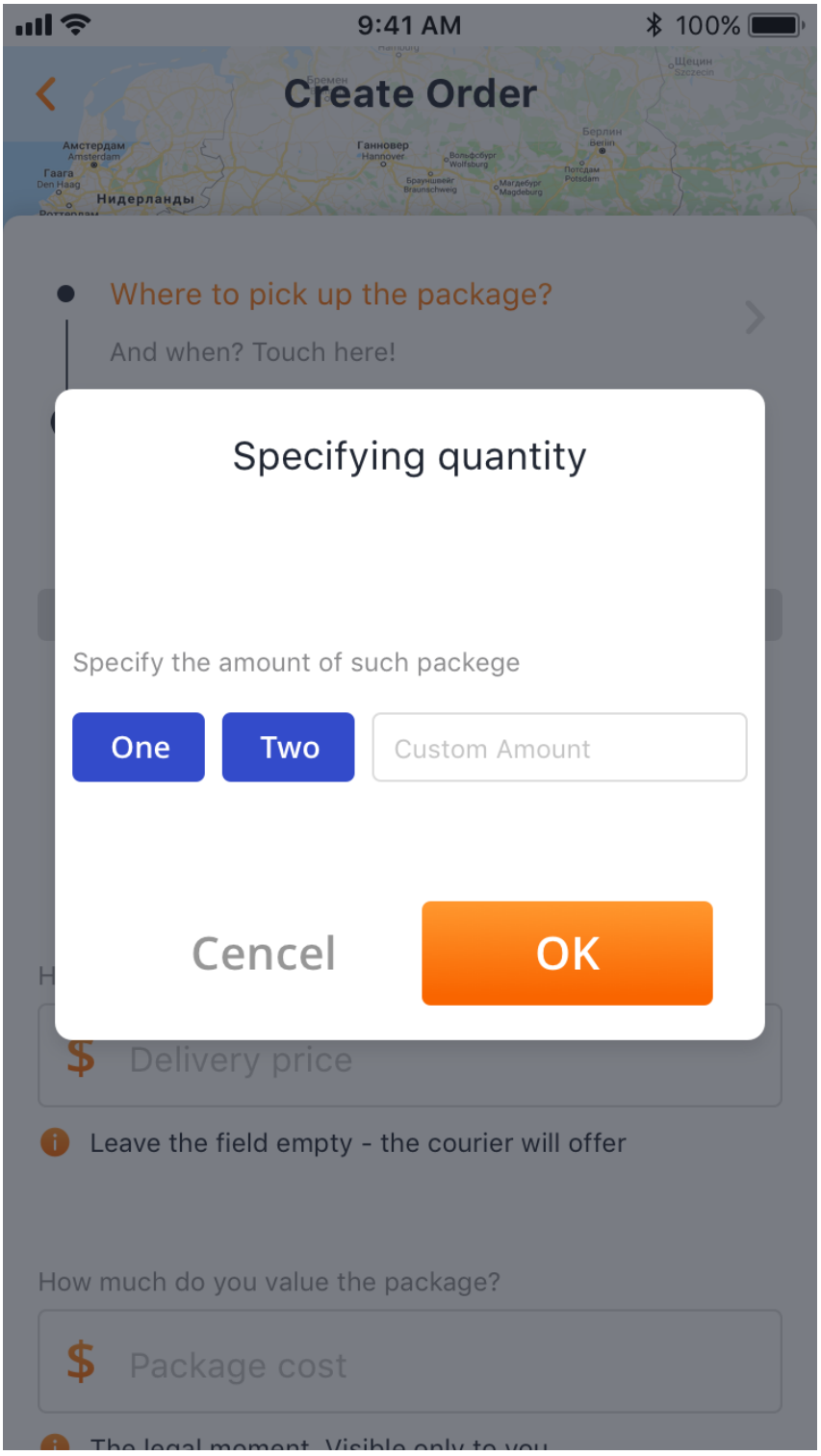


Рисунок 3.4.6.6 – Можливіть вказати скільки товарів буде в замовленні.

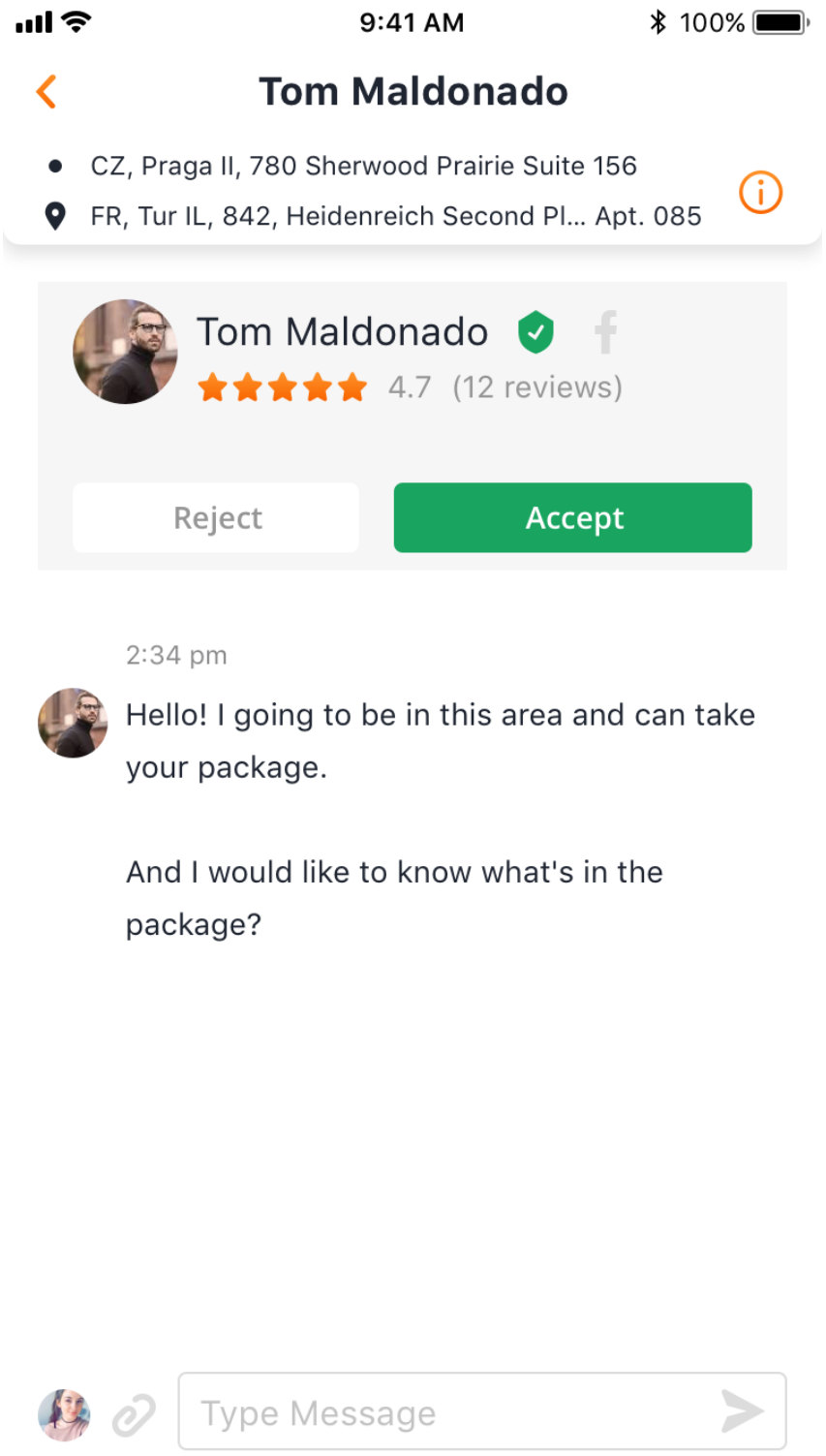


Рисунок 3.4.6.7 – Екран відкугу на замовлення а також коментрів від кур'єра.

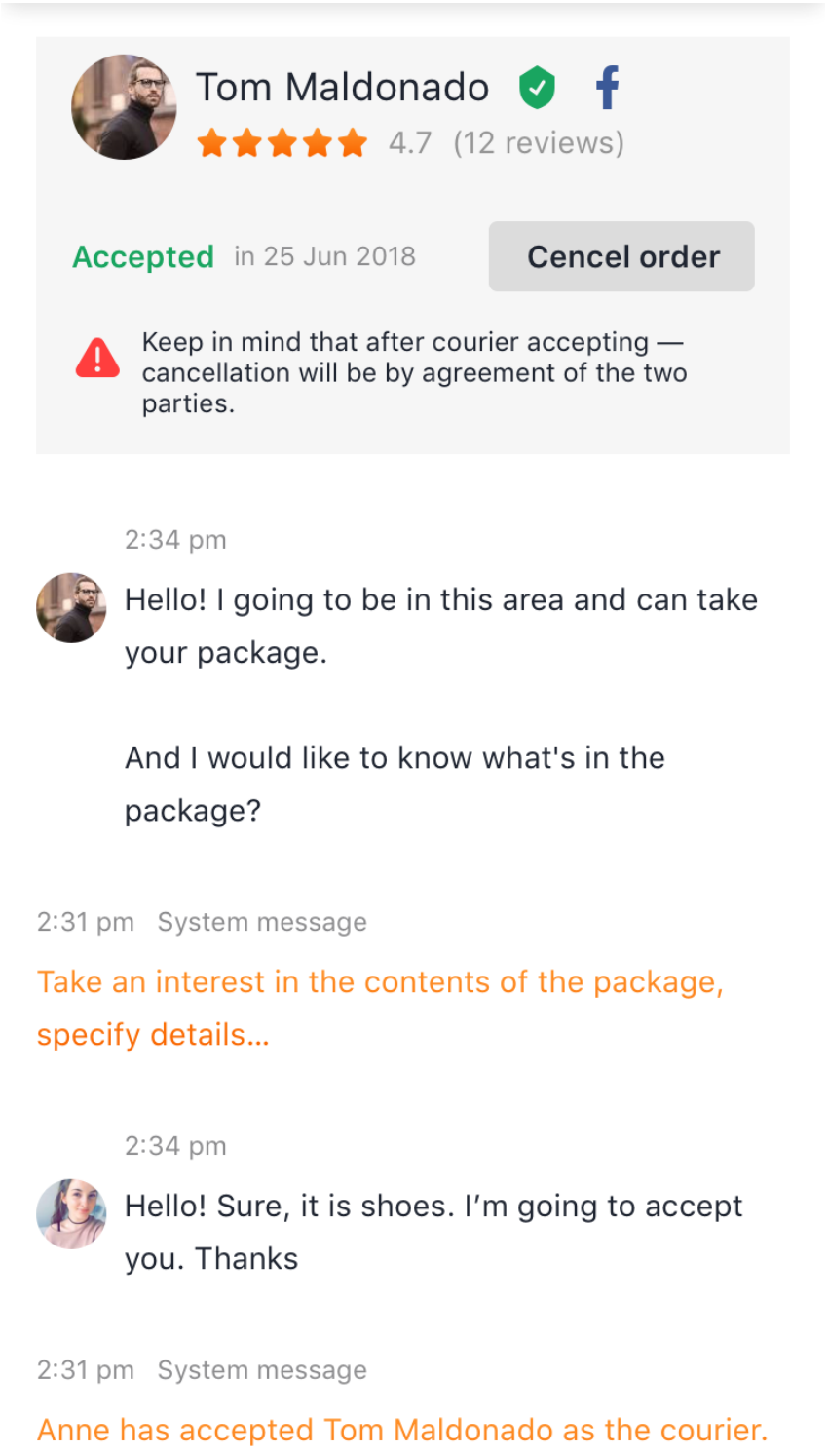


Рисунок 3.4.6.8 – Екран всіх відгуків на замовлення.

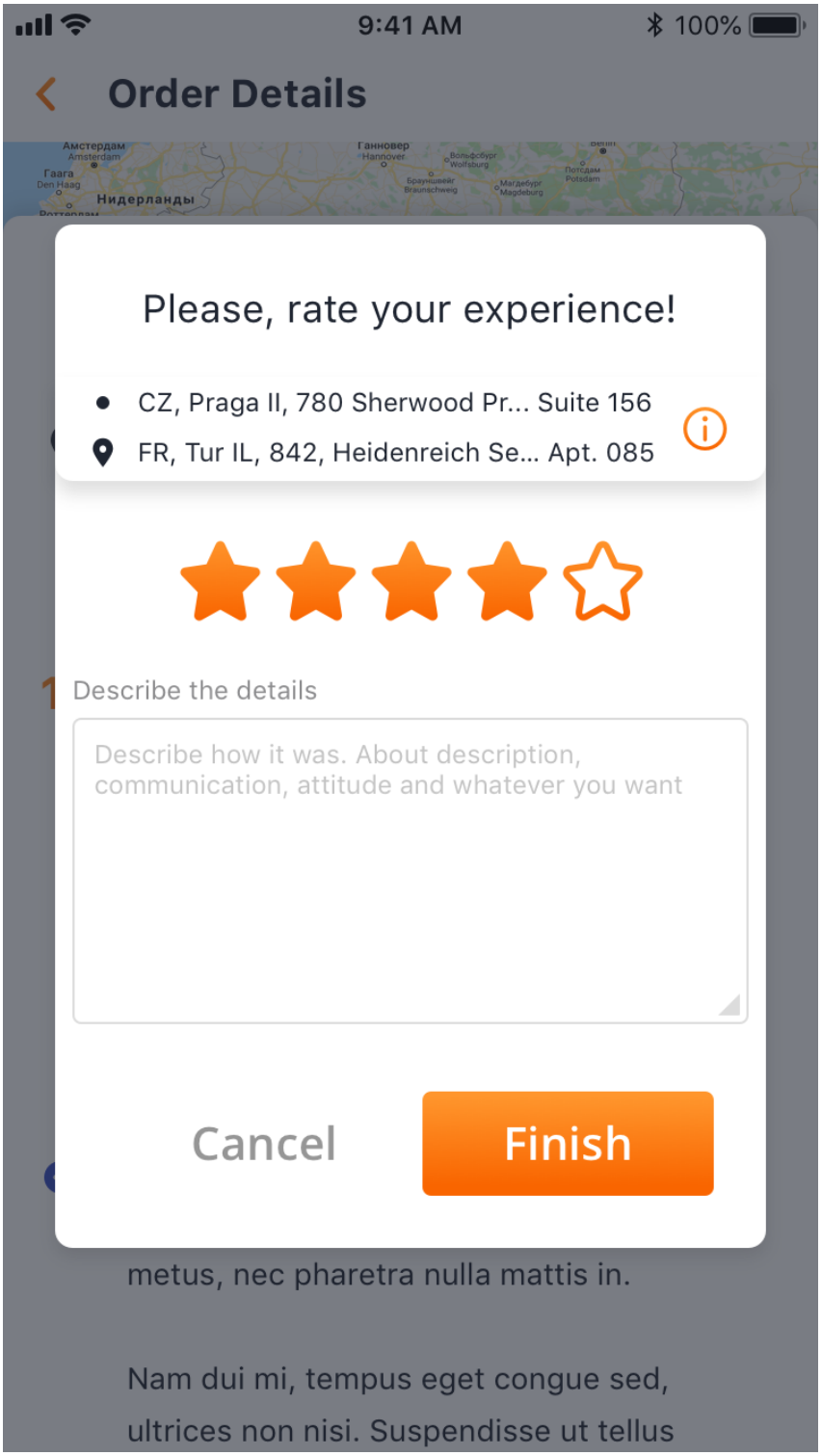


Рисунок 3.4.6.9 – Екран залилення відгуку про виконання замовлення.

3.4.7. Адмін панель сервісу.

Адмін-панель Django [1] - це потужний інструмент для керування адміністративними аспектами вашого веб-сервісу. Вона надає зручний

інтерфейс для адміністрування Django [1] додатку безпосередньо через веб-браузер. Основні особливості адмін-панелі Django [1] включають:

1. **Автоматично створювані форми**: Django Admin [1] автоматично генерує адміністративні форми для моделей бази даних, що дозволяє зручно додавати, змінювати та видаляти дані.
2. **Контроль доступу**: Є можливість налаштовувати права доступу для користувачів адмін-панелі, визначаючи, які користувачі мають доступ до конкретних частин адміністративного інтерфейсу.
3. **Кастомізація інтерфейсу**: Можно кастомізувати вигляд адмін-панелі, додавши власні стилі, зображення та додаткові JS-скрипти.
4. **Пошук і фільтрація**: Адмін-панель дозволяє швидко знаходити та фільтрувати дані в базі даних.
5. **Розширені можливості**: Також є можливість додавати власні функціональність і зв'язки між моделями в адмін-панелі.

Адмін-панель Django [1] - це потужний інструмент, який значно спрощує адміністрування веб-сервісу, забезпечуючи доступ до бази даних та інших аспектів системи через зручний веб-інтерфейс.

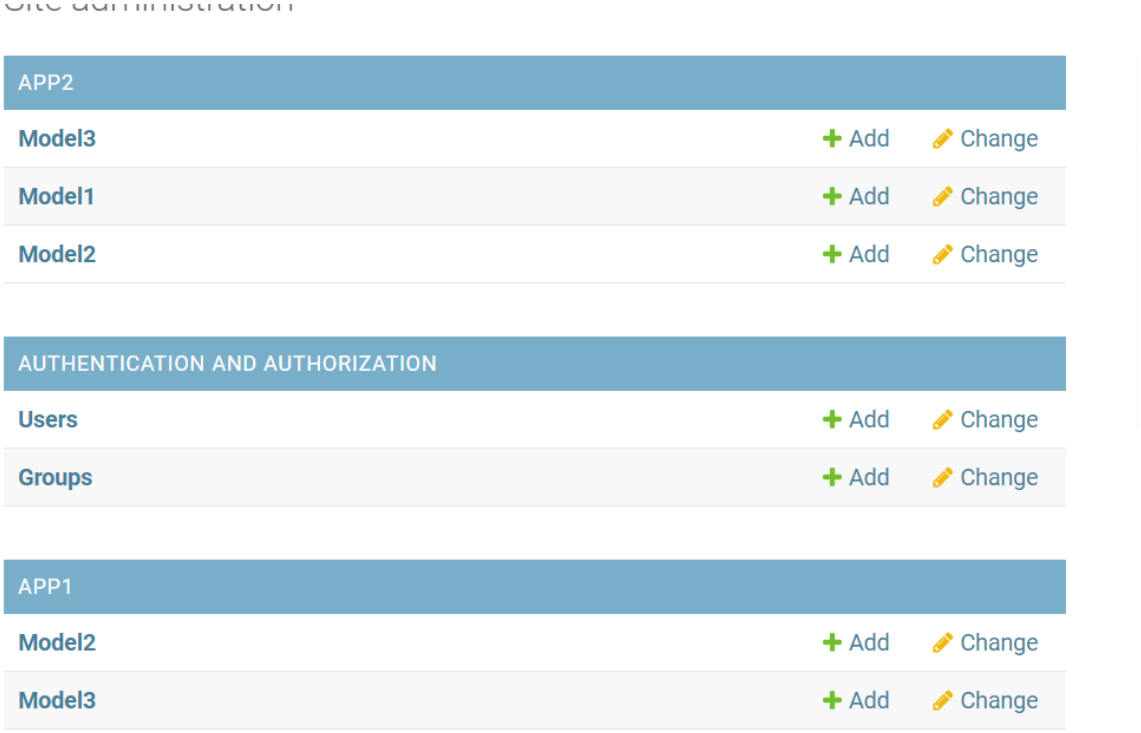


Рисунок 3.4.7 – Адмін-панель Django [1].

Висновки

У цьому дипломному проекті була розглянута розробка сервісу P2P доставки з використанням фреймворку Django [1] та мови програмування Swift [2] для розробки мобільного додатку. Під час аналізу існуючих сервісів P2P доставки був виявлений ряд функціональних особливостей, які необхідно було врахувати впроекті, такі як замовлення доставки, відстеження посилок, оплата послуг, сповіщення користувачів та інші.

Архітектурою було обрано MVC для серверної частини та мобільного додатку, що дозволяє розділити логіку додатку на моделі, представлення та контролери. Це спростило розробку та підтримку проекту.

Для серверної частини використовується фреймворк Django [1], який надає потужні можливості для створення веб-додатків та API. Також були використані різні сторонні бібліотеки, такі як Braintree для оплати, PassportEye для розпізнавання документів, Twilio для сповіщень та інші.

Мобільний додаток був розроблений з використанням мови програмування Swift [2] та ряду бібліотек, включаючи Google Maps для відображення карт, RxSwift [10] для реактивного програмування, Stripe для оплати та Firebase для реального часу спілкування.

Обгрунтування вибору технологій та інструментів було здійснено на основі потреб проекту та їхньої придатності для вирішення конкретних завдань.

В результаті роботи був створений функціональний сервіс P2P доставки та мобільний додаток, які відповідають вимогам користувачів та дозволяють ефективно управляти процесом доставки. Даний проект є важливим внеском у галузь P2P доставки та може бути подальшим об'єктом розширення та розвитку.

Даний проект показує, що використання Django [1] та Swift [2] разом з розумним вибором технологій та інструментів може значно полегшити розробку та розгортання подібних сервісів.

Список використаної літератури

1. Django Documentation. [Онлайн]. Доступно: https://docs.djangoproject.com/en/
2. Swift Documentation. [Онлайн]. Доступно: https://developer.apple.com/documentation/swift/
3. "RESTful Web Services" by Leonard Richardson, Sam Ruby, and David Heinemeier Hansson.
4. "iOS Programming: The Big Nerd Ranch Guide" by Christian Keur and Aaron Hillegass.
5. "Designing Data-Intensive Applications" by Martin Kleppmann.
6. "Python Crash Course" by Eric Matthes.
7. "Swift in Depth" by Tjeerd in 't Veen.
8. "The Django REST framework" by Tom Christie. [Онлайн]. Доступно: https://www.django-rest-framework.org/
9. "Swift UI by Example" by Paul Hudson. [Онлайн]. Доступно: https://www.hackingwithswift.com/swiftui
10. "RxSwift Documentation" by the RxSwift Community. [Онлайн]. Доступно: https://github.com/ReactiveX/RxSwift
11. Reenskaug, Trygve. "Thing-Model-View-Editor." [Онлайн]. Доступно: https://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html
12. Apple Documentation on MVC Architecture in iOS: [Онлайн]. Доступно: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

Додатки

Файл **urls.py**

```
"""kengu URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
        https://docs.djangoproject.com/en/1.11/topics/http/urls/
Examples:
Function views
        1. Add an import:  from my_app import views
        2. Add a URL to urlpatterns:  url(r'^$', views.home, name='home')
Class-based views
        1. Add an import:  from other_app.views import Home
        2. Add a URL to urlpatterns:  url(r'^$', Home.as_view(), name='home')
Including another URLconf
        1. Import the include() function: from django.conf.urls import url, include
        2. Add a URL to urlpatterns:  url(r'^blog/', include('blog.urls'))
"""
from django.conf.urls import url, include
from django.conf import settings
from django.contrib import admin
from django.conf.urls.static import static
from django.contrib.staticfiles.urls import staticfiles_urlpatterns

from authentication.views import (GetTokenAPIView,
                                  RefreshTokenAPIView,
                                  VerifyTokenAPIView,
                                  FacebookLoginAPIView)


urlpatterns = [
        url(r'^admin', admin.site.urls),
        url(r'^auth/login$', GetTokenAPIView.as_view()),
        url(r'^auth/login/facebook$', FacebookLoginAPIView.as_view()),
        url(r'^auth/refresh-token$', RefreshTokenAPIView.as_view()),
        url(r'^auth/verify-token$', VerifyTokenAPIView.as_view()),
        url(r'^payment/', include('payment.urls', namespace='payment')),
        url(r'^users/', include('authentication.urls', namespace='users')),
        url(r'^delivery/', include('delivery.urls', namespace='delivery')),
        url(r'^notification/', include('notifications.urls', namespace='notification')),
        url(r'^silk/', include('silk.urls', namespace='silk'))
```

```
]

urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
urlpatterns += staticfiles_urlpatterns()
```

Файл **development.py**

```
import stripe


DEBUG = True

SILKY_PYTHON_PROFILER = True

SECRET_KEY = 'y$)!34kc_(8%f_ya^x-y4k(7w%%g2tgs14hm!++=!co&+p-c78'

ALLOWED_HOSTS = ['18.191.181.36', '34.203.36.63', '0.0.0.0', '127.0.0.1']

BASE_URL = '127.0.0.1:8000'

ACCOUNT_SID = 'AC5c4de506597ee0280b264749f2cd279b'
AUTH_TOKEN = 'c0ef68144a3463ef211e7e8fc7dd15d1'
TWILIO_NUMBER = '+14159158286'

MAILJET_API_KEY = 'd4e6de64c134398d72b9b09abf896420'
MAILJET_API_SECRET = 'a679a4753b1a1f3c41abe8a06a7e0ebb'
MAILJET_SENDER = 'hello@kengu.app'

STRIPE_API_KEY = 'sk_test_J5kAXNQmsZPUIeKqsMz8xerd'
stripe.api_key = STRIPE_API_KEY
STRIPE = stripe

AUTH_USER_MODEL = 'authentication.User'


FCM_DJANGO_SETTINGS = {
        "FCM_SERVER_KEY":
'AAAAHwWb6zo:APA91bGvLh70a-dnUPedON3FNOqYPuIfsY2Rsnhxm5PL6Qb5K
IbKBSBI8kK9oHtyEJhCPwZjeD_6wLvMYD4O7wHOnycZ4vbvEglcGTrQbWb5CQ
```

```
zU7EzL8YpATaxhISnce1xq2ayLkZSi'
}


DATABASES = {
        'default': {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        'NAME': 'kengu_neomic',
        'USER': 'postgres',
        'PASSWORD': 'kengu_admin_password',
        'HOST': 'localhost',
        'PORT': '5432',
        }
}


REST_FRAMEWORK = {
        'DEFAULT_RENDERER_CLASSES': (
        'utils.renderer.UJSONRenderer',
        ),
        # 'DEFAULT_PARSER_CLASSES': (
        #       'utils.parserer.UJSONParser',
        # ),
        'DEFAULT_PAGINATION_CLASS':
'rest_framework.pagination.PageNumberPagination',
        'PAGE_SIZE': 10,
        'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
        ),
        'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_jwt.authentication.JSONWebTokenAuthentication',
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.BasicAuthentication',
        ),
}


from .base import *

INSTALLED_APPS += ['silk']

MIDDLEWARE += ['silk.middleware.SilkyMiddleware']
```

Файл **models.py**

```
import os
import binascii

from django.db import models
from django.conf import settings
from django.core.validators import MaxValueValidator, MinValueValidator
from django.contrib.auth.base_user import BaseUserManager, AbstractBaseUser
from django.contrib.auth.models import PermissionsMixin
from django.core.urlresolvers import reverse
from django.utils.translation import ugettext_lazy as _


from phonenumber_field.modelfields import PhoneNumberField
from utils.sending import send_mail, send_sms


def generate_key():
        """
        Generates a pseudo random code using os.urandom and binascii.hexlify
        """
        return binascii.hexlify(os.urandom(32)).decode()


class UserManager(BaseUserManager):
        use_in_migrations = True

        def _create_user(self, phone_number, password, **extra_fields):
        """
        Creates and saves a User with the given phone and password.
        """
        if not phone_number:
        raise ValueError('The given phone must be set')
        user = self.model(phone_number=phone_number, **extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

        def create_user(self, phone_number, password=None, **extra_fields):
        extra_fields.setdefault('is_superuser', False)
        extra_fields.setdefault('is_staff', False)
        return self._create_user(phone_number, password, **extra_fields)
```

```
        def create_superuser(self, phone_number, password, **extra_fields):
        extra_fields.setdefault('is_superuser', True)
        extra_fields.setdefault('is_staff', True)
        if extra_fields.get('is_superuser') is not True:
        raise ValueError('Superuser must have is_superuser=True.')

        return self._create_user(phone_number, password, **extra_fields)


class User(AbstractBaseUser, PermissionsMixin):
        password = models.CharField(max_length=128, blank=True)
        main_card = models.PositiveIntegerField(blank=True, null=True)
        phone_number = PhoneNumberField(blank=True, null=True, unique=True)
        email = models.EmailField(blank=True, null=True)
        username = models.CharField(max_length=30, blank=True, null=True)
        first_name = models.CharField(max_length=30, blank=True, null=True)
        last_name = models.CharField(max_length=30, blank=True, null=True)
        currency = models.CharField(max_length=3, default='usd')
        created = models.DateTimeField(auto_now_add=True)
        avatar = models.ImageField(upload_to='avatars',
                        default='avatars/404_avatar.jpg',
                        null=True,
                        blank=True)
        is_active = models.BooleanField(default=True)
        is_email_verified = models.BooleanField(default=False)
        is_phone_verified = models.BooleanField(default=False)
        is_facebook_verified = models.BooleanField(default=False)
        is_passport_verified = models.BooleanField(default=False)
        is_payment_verified = models.BooleanField(default=False)
        is_staff = models.BooleanField(default=False,)
        is_courier = models.BooleanField(default=False)

        objects = UserManager()
        USERNAME_FIELD = 'phone_number'
        REQUIRED_FIELDS = []

        class Meta:
        verbose_name = _('user')
        verbose_name_plural = _('users')

        def __str__(self):
        return self.get_full_name()
```

```
        def get_full_name(self):
        return f'{self.first_name} {self.last_name}'.strip()

        def get_short_name(self):
        return self.first_name


class UserRating(models.Model):
        from delivery.models import Delivery

        author = models.ForeignKey(User,
                        related_name='rating_author',
                        on_delete=models.CASCADE)

        user = models.ForeignKey(User,
                        related_name='rating_user',
                        on_delete=models.CASCADE)

        delivery = models.ForeignKey(Delivery, on_delete=models.CASCADE)
        rating = models.PositiveSmallIntegerField(
        default=1,
        validators=[MinValueValidator(0),
                MaxValueValidator(5)])

        message = models.CharField(max_length=255, blank=True, null=True)

        class Meta:
        db_table = 'user_rating'


class ResetPasswordToken(models.Model):
        class Meta:
        verbose_name = _("Password Reset Token")
        verbose_name_plural = _("Password Reset Tokens")

        user = models.ForeignKey(
        User,
        related_name='password_reset_tokens',
        on_delete=models.CASCADE,
        )

        created = models.DateTimeField(auto_now_add=True)
```

```
        key = models.CharField(
        _("Key"),
        max_length=64,
        primary_key=True
        )

        ip_address = models.GenericIPAddressField(
        _("The IP address of this session"),
        default="127.0.0.1"
        )
        user_agent = models.CharField(
        max_length=256,
        verbose_name=_("HTTP User Agent"),
        default=""
        )

        def save(self, *args, **kwargs):
        if not self.key:
        self.key = generate_key()
        return super(ResetPasswordToken, self).save(*args, **kwargs)

        def __str__(self):
        return "Password reset token for user {user}".format(user=self.user)


class PhoneVerify(models.Model):
        user = models.OneToOneField(User, on_delete=models.CASCADE)
        phone = PhoneNumberField(blank=True, null=True)
        code = models.PositiveSmallIntegerField()
        date_created = models.DateTimeField(auto_now_add=True)

        def save(self, *args, **kwargs):
        send_sms(str(self.phone), settings.TWILIO_NUMBER, f'{self.code}')
        return super(PhoneVerify, self).save(*args, **kwargs)


class EmailVerify(models.Model):
        user = models.ForeignKey(User, on_delete=models.CASCADE)
        email = models.EmailField()
        code = models.CharField(max_length=64)
        date_created = models.DateTimeField(auto_now_add=True)
```

```
        def save(self, *args, **kwargs):
        if not self.code:
        self.code = generate_key()
        return super(EmailVerify, self).save(*args, **kwargs)

        def send(self):
        url = reverse('users:confirm-email',
                kwargs={'code': self.code})
        email_body = f'<a href="{settings.BASE_URL}{url}">Verify</a>'
        r = send_mail(self.email, settings.MAILJET_SENDER, email_body)

        return r


class Passport(models.Model):
        user = models.ForeignKey(User, on_delete=models.CASCADE)
        photo = models.FileField(upload_to='passport', blank=True, null=True)


class TokenCard(models.Model):
        user = models.ForeignKey(User, on_delete=models.CASCADE)
        recToken = models.TextField(unique=True)
        cardPan = models.CharField(max_length=12)
        cardType = models.CharField(max_length=20)
        transactionStatus = models.CharField(max_length=40)
        issuerBankName = models.CharField(max_length=40)
        date_created = models.DateTimeField(auto_now_add=True)
        is_active = models.BooleanField(default=True)


class FacebookData(models.Model):
        user = models.OneToOneField(User,
                        related_name='facebook',
                        on_delete=models.CASCADE,
                        blank=True,
                        null=True)
        access_token = models.TextField()
        facebook_id = models.CharField(max_length=128, blank=True, null=True)
```

Файл **serializers.py**

```
import serpy

from django.contrib.gis.db.models import Q, Avg
from rest_framework import serializers
from drf_extra_fields.geo_fields import PointField
from utils.serializer_fields import StdImageField

from authentication.models import User, UserRating
from .models import (Address,
                     Delivery,
                     DeliveryPhoto,
                     ConfirmedDelivery,
                     CourierPropouse,)


class AddressSerializer(serializers.ModelSerializer):
        location = PointField()

        class Meta:
        model = Address
        fields = '__all__'
        read_only_fields = ('created', 'user',)


class RatingSerializer(serpy.Serializer):
        count = serpy.MethodField()
        score = serpy.MethodField()

        def get_count(self, user):
        return user.count

        def get_score(self, user):
        score = None
        if user.score:
        score = round(user.score, 1)
        return score


class FastUserCreateSerializer(serpy.Serializer):
        id = serpy.Field()
        first_name = serpy.Field()
        last_name = serpy.Field()
        phone_number = serpy.MethodField()
```

```
        avatar = serpy.MethodField()

        def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        if 'context' in kwargs:
        self.context = kwargs['context']

        def get_phone_number(self, user):
        return str(user.phone_number)

        def get_avatar(self, user):
        request = self.context.get('request')
        photo_url = user.avatar.url
        return request.build_absolute_uri(photo_url)


class FastUserListSerializer(FastUserCreateSerializer):
        rating = serpy.MethodField()

        def get_rating(self, user):
        return RatingSerializer(user).data


class DeliveryPhotoSerializer(serializers.ModelSerializer):
        photo = serializers.ImageField(max_length=None, use_url=True)

        class Meta:
        model = DeliveryPhoto
        fields = ('photo',)


class DeliveryPhotoUrlSerializer(serializers.ModelSerializer):
        photo = StdImageField()

        class Meta:
        model = DeliveryPhoto
        fields = ('photo',)


class DeliveryCreateSerializer(serializers.ModelSerializer):
        user = serializers.SerializerMethodField(read_only=True)
        photos = serializers.SerializerMethodField(read_only=True)
        status = serializers.SerializerMethodField(read_only=True)
```

```
courier_status = serializers.SerializerMethodField(read_only=True)
address_from = AddressSerializer()
address_to = AddressSerializer()

class Meta:
model = Delivery
fields = ('id',
          'address_from',
          'address_to',
          'description',
          'price',
          'quantity',
          'weight',
          'width',
          'height',
          'depth',
          'date_created',
          'user',
          'photos',
          'status',
          'courier_status',
          'transfer_date_start',
          'transfer_date_end',
          'receiving_date_start',
          'receiving_date_end',
          'transfer_time_start',
          'transfer_time_end',
          'receiving_time_start',
          'receiving_time_end')
read_only_fields = ('date_created',)

def create(self, validated_data):
user = self.context.get('user')
address_from_data = validated_data.pop('address_from')
address_to_data = validated_data.pop('address_to')
from_ = Address.objects.get_or_create(**address_from_data, user=user)[0]
to = Address.objects.get_or_create(**address_to_data, user=user)[0]
delivery = Delivery.objects.create(**validated_data,
                                   customer=user,
                                   address_from=from_,
                                   address_to=to,
                                   currency=user.currency)
return delivery
```

```
        def get_user(self, delivery):
        user = delivery.customer
        serializer = FastUserCreateSerializer(user,
                                    context=self.context)
        return serializer.data

        def get_photos(self, delivery):
        serializer = DeliveryPhotoUrlSerializer(delivery.deliveryphoto_set,
                                    many=True,
                                    context=self.context)
        return serializer.data

        DELIVERY_STATUS = {
        'open': 0,
        'interested': 1,
        'closed': 2,
        'in progress': 3,
        'delivered_without_rate': 4,
        'delivered': 5
        }

        def get_status(self, delivery):
        status = None

        try:
        if delivery.status == 'open' and len(delivery.propouse):

                status = 'interested'  # if the delivery has one or more offers from the
courier

        elif UserRating.objects.filter(author=delivery.customer,
delivery=delivery).exists():

                status = 'delivered'

        except Exception:
        pass

        if status is None:
        status = delivery.status
        if status == 'done':
        status = 'delivered_without_rate'
```

```
return self.DELIVERY_STATUS[status]

COURIER_STATUS = {
None: None,
'my': 0,
'interested': 1,
'accepted': 2,
'picked': 3,
'delivered_without_rate': 4,
'delivered': 5
}

def get_courier_status(self, delivery):
user = self.context.get('request').user
status = None

if delivery.customer == user:
status = 'my'
return self.COURIER_STATUS[status]

try:
confirmed = delivery.confirmed
if delivery.status == 'closed':
        status = 'accepted'  # if the delivery is complete
        return self.COURIER_STATUS[status]

elif delivery.status == 'in progress' and not confirmed.is_delivered:
        status = 'picked'
        return self.COURIER_STATUS[status]

elif confirmed.is_delivered:
        if delivery.customer == user or delivery.confirmed.courier == user:
        if UserRating.objects.filter(Q(user=user) | Q(author=user),
                        delivery=delivery).exists():
        status = 'delivered'
        return self.COURIER_STATUS[status]

        status = 'delivered_without_rate'
        return self.COURIER_STATUS[status]

except Exception as e:
pass
```

```
        try:
        if CourierPropouse.objects.filter(courier_id=user.id,
delivery=delivery).exists():
                status = 'interested' # if the delivery has one or more offers from the
courier
                return self.COURIER_STATUS[status]

        except Exception as e:
        pass

        return self.COURIER_STATUS[status]


class DeliveryListSerializer(DeliveryCreateSerializer):
        dist = serializers.DecimalField(source='distance.m',
                        max_digits=10,
                        decimal_places=2,
                        required=False,
                        read_only=True)
        photos = serializers.SerializerMethodField(read_only=True)
        user = serializers.SerializerMethodField(read_only=True)
        status = serializers.SerializerMethodField(read_only=True)
        courier_status = serializers.SerializerMethodField(read_only=True)
        courier_list = serializers.SerializerMethodField(read_only=True)
        channels = serializers.SerializerMethodField(read_only=True)

        class Meta:
        model = Delivery
        fields = ('id',
                'dist',
                'address_from',
                'address_to',
                'photos',
                'user',
                'description',
                'status',
                'courier_status',
                'courier_list',
                'channels',
                'price',
                'quantity',
                'weight',
```

```
                'width',
                'height',
                'depth',
                'date_created',
                'transfer_date_start',
                'transfer_date_end',
                'receiving_date_start',
                'receiving_date_end',
                'transfer_time_start',
                'transfer_time_end',
                'receiving_time_start',
                'receiving_time_end')
        read_only_fields = ('date_created', 'dist')

        def get_user(self, delivery):
        user = delivery.customer
        user.count = delivery.customer_count
        user.score = delivery.customer_score
        serializer = FastUserListSerializer(user,
                                            context=self.context)
        return serializer.data

        def get_courier_list(self, delivery):
        serializer = CourierPropouseListSerializer(delivery,
                                            context=self.context)
        return serializer.data

        def get_channels(self, delivery):
        chats = delivery.chatchannel_set.all()
        data = {}
        for chat in chats:
        data[chat.user_id] = chat.channel
        return data


class CourierPropouseSerializer(serializers.ModelSerializer):
        courier = serializers.SerializerMethodField()

        class Meta:
        model = CourierPropouse
        fields = ('courier_price', 'courier', 'interviewed')
        read_only_fields = ('interviewed',)
```

```
        def get_courier(self, propouse):
        courier = propouse.courier
        courier.score = propouse.score
        courier.count = propouse.count
        data = FastUserListSerializer(courier, context=self.context).data
        return data

        def create(self, validated_data):
        delivery = self.context.get('delivery')
        courier = self.context.get('courier')

        propouse, created = CourierPropouse.objects.update_or_create(
        delivery=delivery,
        courier=courier,
        defaults={'courier_price': validated_data.get('courier_price'),
                'is_active': True,
                'interviewed': False}
        )

        return propouse


class CourierPropouseListSerializer(serializers.Serializer):
        confirmed_courier = serializers.SerializerMethodField()
        courier_propouse = serializers.SerializerMethodField()

        def get_confirmed_courier(self, delivery):
        try:
        confirmed_courier = delivery.confirmed.courier
        confirmed_courier.score = delivery.confirmed_score
        confirmed_courier.count = delivery.confirmed_count
        data = FastUserListSerializer(confirmed_courier,
                        context=self.context).data
        return data
        except:
        return None

        def get_courier_propouse(self, delivery):
        propouse = delivery.propouse

        data = CourierPropouseSerializer(
        propouse,
        many=True,
```

```
            context=self.context
            ).data

            return data


class ConfirmCourierSerializer(serializers.ModelSerializer):
            class Meta:
            model = ConfirmedDelivery
            fields = ('courier', 'delivery', 'final_price')
            read_only_fields = ('delivery', 'final_price', 'created')

            def create(self, validated_data):
            courier = validated_data.get('courier')
            delivery = validated_data.get('delivery')
            final_price = validated_data.get('final_price')

            obj, created = ConfirmedDelivery.objects.update_or_create(
            delivery=delivery,
            defaults={
                        'courier': courier,
                        'final_price': final_price,
                        'is_delivered': False,
                        'is_transferred': False
            }
            )

            return obj


class InterviewedSerializer(serializers.Serializer):
            courier_id = serializers.IntegerField()
```

Файл **views.py**

```
from django.db.models import Prefetch

from rest_framework import status, mixins
from rest_framework.response import Response
from rest_framework.generics import ListAPIView, CreateAPIView
from rest_framework.viewsets import ViewSet, GenericViewSet
from fcm_django.models import FCMDevice
```

```
from .serializers import (NotificationSerializer,
                          GeoNotificationSerializer,
                          ChatChannelSerializer,
                          DeviceSerializer)
from .models import Notification, GeoNotification
from delivery.models import Delivery, CourierPropouse


class NotificationAPIView(ListAPIView):
        serializer_class = NotificationSerializer

        def get_queryset(self):
        delivery_prefetch = Prefetch(
        'delivery',
        Delivery.objects
                .select_related('address_to')
                .select_related('address_from')
        )

        return Notification.objects \
                .prefetch_related('courier') \
                .prefetch_related(delivery_prefetch) \
                .filter(user=self.request.user)

        def get_serializer_context(self):
        return {'request': self.request}


class GeoNotificationListCreateDeleteView(mixins.ListModelMixin,
                                          mixins.CreateModelMixin,
                                          mixins.DestroyModelMixin,
                                          GenericViewSet):
        serializer_class = GeoNotificationSerializer

        def get_queryset(self):
        return GeoNotification.objects.filter(user=self.request.user)

        def get_serializer_context(self):
        return {'user': self.request.user}


class ChatChannelCreateAPIView(CreateAPIView):
```

```
        serializer_class = ChatChannelSerializer


class RegisterDeviceViewSet(ViewSet):
        def register_apple(self, request):
        serializer = DeviceSerializer(data=request.data)
        if serializer.is_valid():
        registration_id = serializer.validated_data.get('register_id')
        FCMDevice.objects.get_or_create(
                registration_id=registration_id,
                user=request.user,
                type='ios'
        )

        return Response(status=status.HTTP_201_CREATED)

        return Response(status=status.HTTP_400_BAD_REQUEST)

        def unregister_apple(self, request):
        serializer = DeviceSerializer(data=request.data)
        if serializer.is_valid():
        registration_id = serializer.validated_data.get('register_id')

        try:
                FCMDevice.objects.get(
                registration_id=registration_id,
                user=request.user,
                type='ios'
                ).delete()
        except Exception:
                pass

        return Response(status=status.HTTP_200_OK)

        return Response(status=status.HTTP_400_BAD_REQUEST)
```

Файл **AppDelegate.swift**

```
//
```

```
//  AppDelegate.swift
//  Kengu
//

import UIKit
import IQKeyboardManagerSwift
import Branch
import Mixpanel

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

	var window: UIWindow?
	var appDependencies = AppDependencies()
	var deviceToken: String = ""
	var firebaseToken: String = ""
	var notificationPayload = [String: Any]()


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

	IQKeyboardManager.shared.enable = true
	IQKeyboardManager.shared.disabledDistanceHandlingClasses =
[ChatViewController.self]

	window = UIWindow(frame: UIScreen.main.bounds)
	window?.backgroundColor = .white
	window?.tintColor = .white

	setupTabBar()

	let rootVC: UIViewController = SplashScreenViewController(nibName:
"SplashScreenViewController", bundle: nil)

	window?.rootViewController = UINavigationController(rootViewController:
rootVC)
	window?.makeKeyAndVisible()

	appDependencies.setupDependenciesForApplication(application,
launchOptions: launchOptions)
```

```
        return true
        }

        func applicationWillResignActive(_ application: UIApplication) { }

        func applicationDidEnterBackground(_ application: UIApplication) { }

        func applicationWillEnterForeground(_ application: UIApplication) { }

        func applicationDidBecomeActive(_ application: UIApplication) {
        appDependencies.kenguAppDidBecomeActive(application)
        }

        func applicationWillTerminate(_ application: UIApplication) { }

        func application(_ application: UIApplication,
didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {
        appDependencies.kenguApp(application,
didRegisterForRemoteNotificationsWithDeviceToken: deviceToken)
        }

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

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

        func application(_ app: UIApplication, open url: URL, options:
[UIApplication.OpenURLOptionsKey : Any] = [:]) -> Bool {
        return appDependencies.kenguApp(app, open: url, options: options)
        }

        func application(_ application: UIApplication, open url: URL,
sourceApplication: String?, annotation: Any) -> Bool {
        return appDependencies.kenguApp(application, open: url, sourceApplication:
sourceApplication, annotation: annotation)
```

```
        }

        private func application(_ application: UIApplication, continue userActivity:
NSUserActivity, restorationHandler: @escaping ([Any]?) -> Void) -> Bool {
        return appDependencies.kenguApp(application:application,
continue:userActivity, restorationHandler:restorationHandler)
        }

        func application(_ application: UIApplication, performActionFor
shortcutItem: UIApplicationShortcutItem, completionHandler: @escaping (Bool) ->
Void) {
        appDependencies.kenguApp(application, performActionFor: shortcutItem,
completionHandler: completionHandler)
        }

}

extension AppDelegate {

        func switchToTabController(_ user: User? = nil, transition:
UIView.AnimationOptions = .transitionFlipFromLeft) {

        if let user = user {
        if user.id != -1 {
                SendBirdHelper.login(user: user)
        }
        UserDefaults.standard.set(user.id, forKey: "id")
        KenguAnalytics.shared.logLoginEvent(withUserRole: UserType.type ==
.courier ? "Courier" : "Customer", phone: user.phone, email: user.email)
        Branch.getInstance().setIdentity(String(user.id))
        Mixpanel.mainInstance().identify(distinctId: String(user.id))
        }

        UIView.transition(with: self.window!, duration: 0.5, options: transition,
animations: {
        let tabScreen = UIStoryboard.main().viewController(withID:
.tabBarViewController) as! TabBarViewController
        tabScreen.userAfterLogin = user
        AppDelegate.shared.rootNavController.viewControllers.removeAll()
        self.window?.rootViewController =
UINavigationController(rootViewController: tabScreen)
        }, completion: nil)
        }
```

```
        func switchToLogin(transition: UIView.AnimationOptions =
.transitionFlipFromLeft){
        UIView.transition(with: self.window!, duration: 0.5, options: transition,
animations: {
        let authVC = UIStoryboard.auth().viewController(withID:
.authViewController) as! AuthViewController
        AppDelegate.shared.rootNavController.viewControllers.removeAll()
        self.window?.rootViewController =
UINavigationController(rootViewController: authVC)
        }, completion: nil)
        }

        func switchTo(tab: TabKind) {
        guard let tabBarVc =
AppDelegate.shared.rootNavController.viewControllers.first as?
TabBarViewController else {
        print(" ❗ Failed to recive TabBarViewController in \(self) ❗ ")
        return
        }
        tabBarVc.selectedTab = tab
        }

}

// MARK: - Setup Appereecne
private extension AppDelegate {

        func setupTabBar() {
        UITabBarItem.appearance().setTitleTextAttributes(
        [NSAttributedString.Key.foregroundColor: UIColor.dark,
        NSAttributedString.Key.font: UIFont.tabBarItem], for:.normal)
        UITabBarItem.appearance().setTitleTextAttributes(
        [NSAttributedString.Key.foregroundColor: UIColor.gradientRed,
        NSAttributedString.Key.font: UIFont.tabBarItem], for:.selected)

        UITabBar.appearance().tintColor = .gradientRed
        UITabBar.appearance().barTintColor = .white
        UITabBar.appearance().unselectedItemTintColor = .dark
        }

}
```

```
extension AppDelegate {

	static var shared: AppDelegate {
	return UIApplication.shared.delegate as! AppDelegate
	}

	var rootNavController: UINavigationController {
	return window!.rootViewController as! UINavigationController
	}

	func sendPushTokenToBack() {

	BackEndNetworking.shared.addDeviceToken(token: self.firebaseToken,
success: {
	debugPrint("Token succesefully added")
	}) {
	debugPrint("Can't add token")
	}
	}

	func clearPayload() {
	notificationPayload = [String: Any]()
	}

}
```

Файл **Networking.swift**

```
import Alamofire
import SwiftyJSON
import SwiftKeychainWrapper

class BackEndNetworking: NSObject {

	static let shared = BackEndNetworking()

	let baseUrl = "http://134.209.197.237"

	typealias ErrorBlock = (_ errorCode:Int, _ errorTitle:String, _
errorMessage:String) -> ()

	private var userToken: String?
```

```
	let wrapper = KeychainWrapper.standard

	var token: String? {
	get {
		let key = wrapper.string(forKey: DefaultsKeys.Token.rawValue)

		if key == "" {
		return nil
		}

		return key
	}
	set(newToken) {
		guard newToken != nil else {
		wrapper.set("", forKey: DefaultsKeys.Token.rawValue)
		return
		}
		wrapper.set(newToken!, forKey: DefaultsKeys.Token.rawValue)
	}
	}

	override init() {
	Alamofire.SessionManager.default.delegate.taskWillPerformHTTPRedirection
= { session, task, response, request in
		var redirectedRequest = request

		if let originalRequest = task.originalRequest,
		let headers = originalRequest.allHTTPHeaderFields,
		let authorizationHeaderValue = headers["Authorization"]
		{
		var mutableRequest = request
		mutableRequest.setValue(authorizationHeaderValue,
forHTTPHeaderField: "Authorization")
		redirectedRequest = mutableRequest
		}

		return redirectedRequest
	}
	}

	func otherHeaders() -> [String: String] {
	var appHeaders = [String:String]()
		appHeaders["Content-Type"] = "application/json"
	return appHeaders
	}

	func authHeaders() -> [String: String] {
	var appHeaders = self.otherHeaders()
	appHeaders["Authorization"] = "JWT \(token ?? "")"
	return appHeaders
	}
```

```
	func onlyAuthHeader() -> [String: String] {
	return ["Authorization":"JWT \(token ?? "")"]
	}



	// MARK: - Token



	// MARK: - Login
	func loginWith(number: String,
	            success:@escaping () -> (),
	            failure:@escaping ErrorBlock) {


	let requestURL = baseUrl + "/auth/login"


	Alamofire.request(requestURL, method: .post, parameters:
["phone_number":number], encoding: JSONEncoding.default, headers: nil)

	        .validate()
	        .responseJSON { response in
	        if response.response?.statusCode == 200 {
	            success()
	            return
	        }
	        switch response.result {
	        case .success:
	            success()

	        case .failure(let error):
	            print(error)
	            print(NSString(data:response.data!,
encoding:String.Encoding.utf8.rawValue) ?? "no error data")
	            var errorCode = 0
	            if let code = response.response?.statusCode {
	                errorCode = code
	            }
	            failure(errorCode, NSLocalizedString("REQUEST_AUTH_ERROR",
comment: ""), error.localizedDescription)
	        }
	}
	//     debugPrint(request)
	}

	// MARK: - Facebook

	func facebookLogin(token: String,
```

```
                  facebookId: String,
                  successCreate:@escaping (_ facebookId: String) -> (),
                  successLogin:@escaping (_ user: User) -> (),
                  failure:@escaping () -> ()) {


      let requestURL = baseUrl + "/auth/login/facebook"


      Alamofire.request(requestURL, method: .post, parameters: ["access_token":
token, "facebook_id": facebookId], headers: nil)

            .responseJSON { response in

            switch response.result {
            case .success:

                  let json = JSON(response.result.value!)
                  print(json)
                  if let id = json["facebook_id"].string {
                        successCreate(id)
                  } else if let token = json["token"].string {
                        self.token = token
                        successLogin(User.from(json["user"]))
                  }


            case .failure(let error):
                  print(error)
                  failure()
            }
      }
      }

      func facebookGetToken(phoneNumber: String,
                        facebookId: String,

                        success:@escaping (_ phoneNumber: String) -> (),
                        failure:@escaping () -> ()) {


      let requestURL = baseUrl + "/users/facebook/"


      Alamofire.request(requestURL, method: .post, parameters: ["phone_number":
phoneNumber, "facebook_id": facebookId], headers: nil)

            .responseJSON { response in
            let code = response.response?.statusCode ?? 400
            if code >= 200 && code < 300 {
                  success(phoneNumber)
```

```
                return
            }
            switch response.result {
            case .success:

                success(phoneNumber)

            case .failure(let error):
                print(error)
                failure()
            }
        }
        }


        // MARK: - Phone

        func sendSMS(phone: String,
            success:@escaping () -> (),
            failure:@escaping () -> ()) {


        let requestURL = baseUrl + "/users/phone-verify/"


        Alamofire.request(requestURL, method: .post, parameters: ["phone":
phone], headers: nil)

            .responseJSON { response in

            switch response.result {
            case .success:
                success()
                let json = JSON(response.result.value!)
                print(json)

            case .failure(let error):
                print(error)
                failure()
            }
        }
        }

        func confirmCode(code: String,
                phone: String,
                success:@escaping (_ user: User) -> (),
                failure:@escaping () -> ()) {


        let requestURL = baseUrl + "/users/phone-verify/confirm/"
```

```
      Alamofire.request(requestURL, method: .post, parameters: ["phone": phone,
"code": code], encoding: JSONEncoding.default, headers: nil)

            .responseJSON { response in

            switch response.result {
            case .success:
                  if let result = response.result.value {
                        let json = JSON(result)
                        print(json)
                        self.token = json["token"].string
                        success(User.from(json["user"]))
                  } else {
                        failure()
                  }

            case .failure(let error):
                  print(error)
                  failure()
            }
      }
      }

      // MARK: - Email

      func emailVerify(email: String,
                  success:@escaping () -> (),
                  failure:@escaping () -> ()) {


      let requestURL = baseUrl + "/users/email-verify/"


      Alamofire.request(requestURL, method: .post, parameters: ["email":
email], headers: authHeaders())

            .responseJSON { response in

            switch response.result {
            case .success:

                  let json = JSON(response.result.value!)
                  print(json)

                  success()


            case .failure(let error):
                  print(error)
                  failure()
```

```
            }
      }
      }

      func sendConfirmUrl(email: String,
                          success:@escaping () -> (),
                          failure:@escaping () -> ()) {


      let requestURL = baseUrl + "/users/password/"


      Alamofire.request(requestURL, method: .post, parameters: ["email":
email], headers: authHeaders())

            .responseJSON { response in

            switch response.result {
            case .success:

                  let json = JSON(response.result.value!)
                  print(json)

                  success()


            case .failure(let error):
                  print(error)
                  failure()
            }
      }
      }

      // MARK: - Users

      func createUser(user: User,
                  success:@escaping () -> (),
                  failure:@escaping () -> ()) {


      let requestURL = baseUrl + "/users/"


      Alamofire.request(requestURL, method: .post, parameters:
user.newUserRequestParams(), encoding: JSONEncoding.default, headers:
authHeaders())

            .responseJSON { response in

            switch response.result {
            case .success:
```

```
                let json = JSON(response.result.value!)
                print(json)
                success()
            case .failure(let error):
                print(error)
                failure()

            }
        }
        }

        func chooseMainCard(token: String,
                            success:@escaping () -> (),
                            failure:@escaping () -> ()) {


        let requestURL = baseUrl + "/users/main-card/"


        Alamofire.request(requestURL, method: .post, parameters: ["recToken":
token], encoding: JSONEncoding.default, headers: authHeaders())

            .responseJSON { response in

            switch response.result {
            case .success:

                let json = JSON(response.result.value!)
                print(json)
                success()
            case .failure(let error):
                print(error)
                failure()

            }
        }
        }


        func updateUser(user: User,
                        success:@escaping () -> (),
                        failure:@escaping () -> ()) {


        let requestURL = baseUrl + "/users/"


        Alamofire.request(requestURL, method: .put, parameters:
user.newUserRequestParams(), encoding: JSONEncoding.default, headers:
authHeaders())
```

```
            .responseString { response in

            switch response.result {
            case .success:
                  success()
            case .failure(let error):
                  print(error)
                  failure()
            }
      }


      }

      func getAllUsers(success:@escaping () -> (),
                  failure:@escaping ErrorBlock) {

      let requestURL = baseUrl + "/users/"

      Alamofire.request(requestURL, method: .get, parameters: nil, headers:
authHeaders())

            .responseJSON { response in

            switch response.result {
            case .success:
                  let json = JSON(response.result.value!)
                  print(json)
            case .failure(let error):
                  print(error)

            }
      }
      }

      func getUser(id: String,
                  success:@escaping (_ user: User) -> (),
                  failure:@escaping () -> ()) {


      let requestURL = baseUrl + "/users/\(id)/"


      Alamofire.request(requestURL, method: .get, parameters: nil, headers:
authHeaders())
            .responseJSON { response in
            switch response.result {
            case .success:
                  let user = User.from(JSON(response.result.value!))
                  success(user)
```

```
            case .failure(let error):
                  print(error)
                  failure()
            }
      }
      }

      func uploadAvatar(  image: UIImage,
                        success:@escaping (_ url: String) -> (),
                        failure:@escaping () -> ()) {

      let requestURL = baseUrl + "/users/avatar/"

      Alamofire.upload(
            multipartFormData: { MultipartFormData in
            //     multipartFormData.append(imageData, withName: "user",
fileName: "user.jpg", mimeType: "image/jpeg")

            MultipartFormData.append(image.jpegData(compressionQuality: 0.5)!,
withName: "avatar", fileName: "1photo.jpeg", mimeType: "image/jpeg")

      },to: requestURL,
      method: .put,
      headers: authHeaders(),
            encodingCompletion: { encodingResult in
            switch encodingResult {
            case .success(let upload, _, _):
            upload.responseJSON { response in
                  if let data = response.result.value {
                        let json = JSON(data)
                        if let url = json.string {
                        success(url)
                        }
                  }
            }
            case .failure(let encodingError):
            print("encoding Error : \(encodingError)")
            }
      })
      }

      //verify



      func verifyPassport(  image: UIImage,
                        success:@escaping () -> (),
                        failure:@escaping (_ message: String, _ showImage:
Bool) -> ()) {

      let requestURL = baseUrl + "/users/passport-verify/"
```

```
	Alamofire.upload(
		multipartFormData: { MultipartFormData in

		MultipartFormData.append(image.jpegData(compressionQuality: 1)!,
withName: "photo", fileName: "pass.jpeg", mimeType: "image/jpeg")

	},to: requestURL,
	method: .post,
	headers: authHeaders(),
	encodingCompletion: { encodingResult in
		switch encodingResult {
		case .success(let upload, _, _):

		upload.responseString { response in
			debugPrint(response)
			if let code = response.response?.statusCode {
				if code >= 200 && code < 300 {
				success()
				} else {
				failure("Please take a photo like in the example
below", true)
				}
			}
		}
		case .failure(let encodingError):
		print("encoding Error : \(encodingError)")
		failure("Doesn't load photo", false)

		}
	})
	}


	// MARK: - Payment

	func addPaymentCard(token: String,
		success:@escaping () -> (),
		failure:@escaping () -> ()) {


	let requestURL = baseUrl + "/stripe-card/"


	Alamofire.request(requestURL, method: .post, parameters: ["card_token":
token], headers: authHeaders())

		.responseJSON { response in

		switch response.result {
		case .success:
```

```
                success()
            case .failure(let error):
                print(error)
                failure()
            }
        }
        }

        // MARK: - Orders

        func createOrder(order: Order,
                success:@escaping (_ order: Order) -> (),
                failure:@escaping () -> ()) {

        let requestURL = baseUrl + "/delivery/"


        Alamofire.request(requestURL, method: .post, parameters:
order.newOrderRequestParams(), encoding: JSONEncoding.default, headers:
authHeaders())

            .responseJSON { response in

            switch response.result {
            case .success:

                let json = JSON(response.result.value!)
                print(json)
                success(Order.fromJSON(json))

            case .failure(let error):
                failure()
                print(error)

            }
        }
        }

        func deleteOrder(id: String,
                success:@escaping () -> (),
                failure:@escaping () -> ()) {

        let requestURL = baseUrl + "/delivery/\(id)/delete/"


        Alamofire.request(requestURL, method: .delete, parameters: nil, encoding:
JSONEncoding.default, headers: authHeaders())

            .responseJSON { response in

            switch response.result {
```

```
                case .success:
                        success()
                case .failure:
                        failure()
                }
        }
        }


        func uploadPhotoOrder(  image: UIImage,
                                id: String,
                                success:@escaping (_ url: String) -> (),
                                failure:@escaping () -> ()) {

        let requestURL = baseUrl + "/delivery/\(id)/photo/"

        Alamofire.upload(
                multipartFormData: { MultipartFormData in
                MultipartFormData.append(image.jpegData(compressionQuality: 0.8)!,
withName: "photo", fileName: "1photo.jpeg", mimeType: "image/jpeg")
        },to: requestURL,
        method: .post,
        headers: authHeaders(),
        encodingCompletion: { encodingResult in
                switch encodingResult {
                case .success(let upload, _, _):
                upload.responseJSON { response in

                        if let data = response.result.value {
                                let json = JSON(data)
                                print("upload photo order: \(json)")
                                if let url = json["original"].string {
                                success(url)
                                } else {
                                failure()
                                }
                        } else {
                                failure()
                        }
                }
                case .failure(let encodingError):
                print("encoding Error : \(encodingError)")
                failure()
                }
        })


        }

        func orderList(_ type: OrderListType, filter: FilterPoint? = nil,
                        success:@escaping (_ feed: [Order], _ next: String, _ count:
```

```
Int) -> (),
                    failure:@escaping () -> ()) {

        var requestURL: String

        switch type {
        case .all:
                requestURL = baseUrl + "/delivery/"
        case .courierMyOrders:
                requestURL = baseUrl + "/users/as-courier/"
        case .courier:
                requestURL = baseUrl + "/delivery/"
        case .customer:
                requestURL = baseUrl + "/users/as-customer/"
        }


        Alamofire.request(requestURL, method: .get, parameters: filter?.params(),
encoding: URLEncoding.default , headers: authHeaders())

                .responseJSON { response in

                switch response.result {
                case .success:

                        let json = JSON(response.result.value!)
                        print(json)
                        let feed: [Order] = Order.fromJSON(json)
                        success(feed, json["next"].string ?? "", json["count"].int
?? 0)
//                      print(feed)
                case .failure(let error):
                        print(error)
                }
        }
        }

        func map(bounds: MapBounds,
                        success:@escaping (_ feed: [Order]) -> (),
                        failure:@escaping () -> ()) {

        let requestURL = baseUrl + "/delivery/map/"


        Alamofire.request(requestURL, method: .get, parameters:
bounds.toParams(), encoding: URLEncoding.default , headers: authHeaders())

                .responseJSON { response in

                switch response.result {
                case .success:
```

```
                let json = JSON(response.result.value!)
                let feed: [Order] = Order.fromJSONonMap(json)
                success(feed)
                print("map success")
            case .failure(let error):
                print("map failed")
                print(error)
            }
        }
        }

        func orderListNextPage(_ url: String,
                               success:@escaping (_ feed: [Order], _ next: String?,
_ count: Int) -> (),
                               failure:@escaping () -> ()) {


        Alamofire.request(url, method: .get, encoding: JSONEncoding.default,
headers: authHeaders())

            .responseJSON { response in

            switch response.result {
            case .success:

                let json = JSON(response.result.value!)
                print(json)
                let feed: [Order] = Order.fromJSON(json)
                success(feed, json["next"].string, json["count"].int!)

            case .failure(let error):
                print(error)
            }
        }
        }

        func order(_ id: String,
            success:@escaping (_ order: Order) -> (),
                               failure:@escaping () -> ()) {

        let requestURL = baseUrl + "/delivery/\(id)/"
        Alamofire.request(requestURL, method: .get, encoding:
JSONEncoding.default, headers: authHeaders())

            .responseJSON { response in

            switch response.result {
            case .success:
                let json = JSON(response.result.value!)
                let order: Order = Order.fromJSON(json)
```

```
                success(order)
            case .failure(let error):
                print(error)
            }
        }
        }


        func markAsInterestingOrder(id: String,
                                    price: Double,
                                    success:@escaping () -> (),
                                    failure:@escaping () -> ()) {

        let requestURL = baseUrl + "/delivery/\(id)/"


        Alamofire.request(requestURL, method: .post, parameters: [
"courier_price": price], encoding: JSONEncoding.default, headers: authHeaders())

            .responseString { response in

            switch response.result {
            case .success:
                success()
                let json = JSON(response.result.value!)

                print(json)
            case .failure(let error):
                print(error)

            }
        }
        }

        func cancelInterestingOrder(id: String,
                                    success:@escaping () -> (),
                                    failure:@escaping () -> ()) {

        let requestURL = baseUrl + "/delivery/\(id)/"


        Alamofire.request(requestURL, method: .delete, parameters: nil, encoding:
JSONEncoding.default, headers: authHeaders())

            .responseString { response in

            switch response.result {
            case .success:
                if let code = response.response?.statusCode {
                    if 500 == code {
                    failure()
```

```
                        return
                        }
                }
                success()

        case .failure(let error):
                print(error)

        }
    }
    }


    func chooseCourier(orderId: String,
                courierId: Int,
                success:@escaping () -> (),
                failure:@escaping () -> ()) {

    let requestURL = baseUrl + "/delivery/\(orderId)/confirm-courier/"


    Alamofire.request(requestURL, method: .post, parameters: ["courier":
courierId], encoding: JSONEncoding.default, headers: authHeaders())

        .responseString { response in

        if let code = response.response?.statusCode {
                if code == 201 {
                        success()
                        return
                }
        }

        switch response.result {
        case .success:
                print(response.result.value as Any)
                success()

        case .failure(let error):
                print(error)
                failure()

        }
    }
    }

    func cancelByCustomer(orderId: String,
                        success:@escaping () -> (),
                        failure:@escaping () -> ()) {

    let requestURL = baseUrl + "/delivery/\(orderId)/customer-cancels/"
```

```
      Alamofire.request(requestURL, method: .get, parameters: nil, encoding:
JSONEncoding.default, headers: authHeaders())

            .responseJSON { response in

            if let code = response.response?.statusCode {
                  if code == 202 {
                        success()
                        return
                  }
            }
            switch response.result {
            case .success:
                  success()
            case .failure(_):
                  failure()
            }
      }
      }

      func cancelByCourier(orderId: String,
                        success:@escaping () -> (),
                        failure:@escaping () -> ()) {

      let requestURL = baseUrl + "/delivery/\(orderId)/courier-cancels/"


      Alamofire.request(requestURL, method: .get, parameters: nil, encoding:
JSONEncoding.default, headers: authHeaders())

            .responseJSON { response in

            switch response.result {
            case .success:

                  let json = JSON(response.result.value!)
                  success()
                  print(json)
            case .failure(let error):
                  print(error)
                  failure()

            }
      }
      }

      func confirmByCustomer(orderId: String,
                        success:@escaping () -> (),
                        failure:@escaping () -> ()) {
```

```
	let requestURL = baseUrl + "/delivery/\(orderId)/confirmed-by-customer/"


	Alamofire.request(requestURL, method: .get, parameters: nil, encoding:
JSONEncoding.default, headers: authHeaders())

		.responseJSON { response in
		print(response.response?.statusCode)
		if let code = response.response?.statusCode {
			if 200...300 ~= code {
				let json = JSON(response.result.value!)

				print(json)
				success()
				return
			}
		}

		switch response.result {
		case .success:
			let json = JSON(response.result.value!)

			print(json)
			success()
		case .failure(let error):
			print(error)
			failure()

		}
	}
	}

	func confirmByCourier(orderId: String,
				success:@escaping () -> (),
				failure:@escaping () -> ()) {

	let requestURL = baseUrl + "/delivery/\(orderId)/confirmed-by-courier/"


	Alamofire.request(requestURL, method: .get, parameters: nil, encoding:
JSONEncoding.default, headers: authHeaders())

		.responseJSON { response in
		print(response.response?.statusCode)
		if response.response?.statusCode == 202 {
			success()
		}
		switch response.result {
		case .success:
```

```
                    let json = JSON(response.result.value!)
                    success()
                    print(json)
              case .failure(let error):
                    print(error)
                    failure()

              }
       }
       }

       // MARK: - Notifications

       func notificationList(
       success:@escaping (_ notifications: [NotificationItem], _ next: String, _
count: Int) -> (),
       failure:@escaping () -> ()) {

       let requestURL = baseUrl + "/notification/"


       Alamofire.request(requestURL, method: .get, parameters: nil, encoding:
JSONEncoding.default, headers: authHeaders())

              .responseJSON { response in

              switch response.result {
              case .success:
                    if let data = response.result.value {
                          let json = JSON(data)
                          print(json)

success(NotificationItem.from(json),json["next"].string ?? "", json["count"].int
?? 0)

                    }
              case .failure(let error):
                    print(error)

              }
       }

       }

       func notificationListNextPage(_ url: String,
                          success:@escaping (_ notifications:
[NotificationItem], _ next: String?, _ count: Int) -> (),
                          failure:@escaping () -> ()) {


       Alamofire.request(url, method: .get, encoding: JSONEncoding.default,
```

```
headers: authHeaders())

            .responseJSON { response in

            switch response.result {
            case .success:

                  let json = JSON(response.result.value!)
                  print(json)
                  let notifications: [NotificationItem] =
NotificationItem.from(json)
                  success(notifications, json["next"].string,
json["count"].int!)

            case .failure(let error):
                  print(error)
            }
      }
      }

      // Chat Room


      func saveChatRoom(channelUrl: String,
                  courierId: Int,
                  orderId: Int,
                  success:@escaping () -> (),
                  failure:@escaping () -> ()) {

      let requestURL = baseUrl + "/notification/chat/"


      Alamofire.request(requestURL, method: .post, parameters: ["channel":
channelUrl, "user": courierId, "delivery": orderId], encoding:
JSONEncoding.default, headers: authHeaders())

            .responseJSON { response in

            switch response.result {
            case .success:
                  if let data = response.result.value {
                        let json = JSON(data)
                        print(json)
                        success()

                  }
            case .failure(let error):
                  print(error)

            }
      }
```

```
        }
}



enum DefaultsKeys: String {
        case Token = "token"

}


enum OrderListType {
        case all
        case customer
        case courier
        case courierMyOrders
}
```

Файл **Order.swift**

```
//
//  Order.swift
//  Kengu
//
//

import Foundation
import Alamofire
import SwiftyJSON
import IGListKit

class Order {
        var id = 0
        var addressFrom: Address?
        var addressTo: Address?

        var acceptedCourier: User?
        var description = ""
        var status: OrderStatus = .null
        var statusByUser: OrderStatusByUser = .null
        var groups = [UsersGroup]()
        var channels = [String:String]()
        var price = ""
```

```
var package_cost = 0.0

var quantity = -1
var weight = ""
var width = -1
var height = -1
var depth = -1

var date_created = ""
var transfer_date_start = ""
var transfer_date_end = ""
var receiving_date_start = ""
var receiving_date_end = ""
var transfer_time_start = ""
var transfer_time_end = ""
var receiving_time_start = ""
var receiving_time_end = ""
var user = User()
var photos = [String]()
var thumbnailPhotos = [String]()
var courierPrice = "-1"

// optional
var images = [UIImage]()

func setCourierPrice(){
let userId = UserDefaults.standard.integer(forKey: "id")

for group in groups {
for user in group.users {
        if user.id == userId {
        courierPrice = user.propousePrice
        }
}
}

}

func photosFromJSON(_ json: JSON) {
if let array = json.array {
for photo in array {
        if let url = photo["photo"]["original"].string {
        photos.append(url)
```

```
                }
                if let url = photo["photo"]["thumbnail"].string {
                thumbnailPhotos.append(url)
                }
        }
        }

        }
}

//static methods
extension Order {

        static func channelsFromJSON(_ json: JSON) -> [String:String] {
        var channels = [String:String]()
        if let dict = json.dictionary {
        for channel in dict {
                if let value = channel.value.string{
                channels[channel.key] = value
                }
        }
        }

        return channels
        }

        static func fromJSON(_ json: JSON) -> Order{

        let order = Order()

        order.id = json["id"].int ?? -1
        order.addressFrom = Address.from(json["address_from"])
        order.addressTo = Address.from(json["address_to"])
        order.description = json["description"].string ?? ""
        order.price = json["price"].string ?? ""
        order.package_cost = json["package_cost"].double ?? 0
        order.quantity = json["quantity"].int ?? 0
        order.weight = json["weight"].string ?? ""
        order.width = json["width"].int ?? 0
        order.height = json["height"].int ?? 0
        order.depth = json["depth"].int ?? 0
        order.status = OrderStatus.getStatus(json["status"].int)
```

```
        order.user = User.from(json["user"])
        order.acceptedCourier =
User.from(json["courier_list"]["confirmed_courier"])
        order.channels = Order.channelsFromJSON(json["channels"])
        order.groups = UsersGroup.from(json["courier_list"])
        order.photosFromJSON(json["photos"])

        order.statusByUser =
OrderStatusByUser.getStatus(json["courier_status"].int)

        order.date_created = json["date_created"].string ?? ""
        order.transfer_date_start = json["transfer_date_start"].string ?? ""
        order.receiving_date_start = json["receiving_date_start"].string ?? ""
        order.receiving_date_end = json["receiving_date_end"].string ?? ""
        order.transfer_time_start = json["transfer_time_start"].string ?? ""
        order.transfer_time_end = json["transfer_time_end"].string ?? ""
        order.receiving_time_start = json["receiving_time_start"].string ?? ""
        order.receiving_time_end = json["receiving_time_end"].string ?? ""

        order.setCourierPrice()

        return order
        }

        static func fromJSON(_ json: JSON) -> [Order]{
        var feed = [Order]()

        if let items = json["results"].array {
        items.forEach({ feed.append(Order.fromJSON($0)) })
        }

        return feed
        }

        static func fromJSONNonMap(_ json: JSON) -> [Order]{
        var feed = [Order]()

        if let items = json.array {
        items.forEach({ feed.append(Order.fromJSON($0)) })
        }

        return feed
        }
```

```
}

extension Order {
        func newOrderRequestParams() -> Parameters {
        let params: Parameters = [
        "address_from": addressFrom?.params() ?? "",
        "address_to": addressTo?.params() ?? "",
        "description": description,
        "package_cost": package_cost,
        "price": price,
        "quantity":quantity,
        "weight":weight,
        "width":width,
        "height":height,
        "depth":depth,
        "transfer_date_start": transfer_date_start,
        "transfer_date_end":  transfer_date_end,
        "receiving_date_start": receiving_date_start,
        "receiving_date_end": receiving_date_end,
        "transfer_time_start": transfer_time_start ,
        "transfer_time_end": transfer_time_end,
        "receiving_time_start": receiving_time_start,
        "receiving_time_end": receiving_time_end,


        ]
        print(params)
        return params
        }

        func checkEmptyFields() -> Bool{
        return addressTo == nil || addressFrom == nil || width == -1 || quantity == -1 ||
price == "" || package_cost == 0 || weight == "" || description == ""
        }
}


extension Order: ListDiffable {
        func diffIdentifier() -> NSObjectProtocol {
        return id as NSObjectProtocol
        }

        func isEqual(toDiffableObject object: ListDiffable?) -> Bool {
```

```
        return (object as! Order).id == id
        }
}

// Status

enum OrderStatus{
        case byDefault
        case interested
        case accepted
        case inProgress
        case delivered
        case rated
        case null

        static func getStatus(_ number: Int?) -> OrderStatus{
        // default 0
        // interested 1
        // accepted 2
        // in progress 3
        // delivered 4

        switch number {
        case 0: return .byDefault
        case 1: return .interested
        case 2: return .accepted
        case 3: return .inProgress
        case 4: return .delivered
        case 5: return .rated
        default: return .null
        }
        }
}

enum OrderStatusByUser{

        case mine
        case interested
        case accepted
        case picked
        case delivered
        case rated
        case null
```

```
        static func getStatus(_ number: Int?) -> OrderStatusByUser{

        switch number {
        case 0: return .mine
        case 1: return .interested
        case 2: return .accepted
        case 3: return .picked
        case 4: return .delivered
        case 5: return .rated
        default: return .null
        }
        }
}
```