

Аннотация

Отчёт содержит 47 страниц, 36 иллюстраций, 6 листингов (большая часть из которых расположены в приложении А в конце отчёта), 4 литературных источника.

Целью данной курсовой работы являлось создание серверной части веб-приложения на тему «Сервис управления прокатом электросамокатов». В результате выполнения задания было разработано серверное приложение с MVC (Model View Controller) архитектурой, предоставляющее RESTful (Representational State Transfer) API (Application Programming Interface), который обеспечивает функционал по регистрации, авторизации и редактирования информации пользователей, поиска ближайших точек проката, тарификации использования самокатов (по тарифной ставке, либо по подписке), просмотра истории поездок, а так же по управлению точками проката и самокатами для администраторов. В дополнение в вышеописанному был разработан элементарный одностраничный сайт с документацией, позволящий выполнять все эти манипуляции в более удобном для пользователей формате графического интерфейса.

Для реализации слоя серверной логики использовалась Java 11, Spring Framework (Spring Context, Spring MVC, Spring Security, Spring Boot), в качестве ORM-фреймворка (Object-Relational Mapping) применялся Hibernate, система сборки и контроля зависимостей – Maven, для тестов были использованы JUnit и Mockito, а в роли СУБД (система управления базами данных) выступила PostgreSQL.

Исходный код разработанного приложения доступен для просмотра и скачивания по соответствующей ссылке в сети Интернет.

СОДЕРЖАНИЕ

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ 5](#_Toc122386471)

[ВВЕДЕНИЕ 7](#_Toc122386472)

[1 ОБЩИЕ СВЕДЕНИЯ 8](#_Toc122386473)

[1.1 Обоснование выбора средств ведения разработки 8](#_Toc122386474)

[2 ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ 9](#_Toc122386475)

[3 ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ 10](#_Toc122386476)

[3.1 Анализ предметной области 10](#_Toc122386477)

[3.2 Разработка серверной части веб-приложения 13](#_Toc122386478)

[3.3 Разработка клиентского представления веб-приложения 23](#_Toc122386479)

[3.4 Обзор разработанного программного продукта 25](#_Toc122386480)

[3.5 Подготовка к защите курсовой работы 37](#_Toc122386481)

[ЗАКЛЮЧЕНИЕ 38](#_Toc122386482)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 39](#_Toc122386483)

[ПРИЛОЖЕНИЕ А 41](#_Toc122386484)

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

|  |  |  |
| --- | --- | --- |
| БД | — | База данных |
| СУБД | — | Система управления базами данных |
| API | — | Application Programming Interface (программный интерфейс) |
| CRUD |  | Create, Read, Update, Delete (перечень базовых операций – создания, чтения, обновления, удаления) |
| CSS | — | Cascading Style Sheets (формальный язык описания внешнего вида документа, написанного с использованием языка разметки) |
| DAO | — | Data Access Object (объект, предоставляющий стандартные операции над сущностями, такие как добавление, обновление и удаление) |
| DDL | — | Data Definition Language (группа операторов определения данных в SQL) |
| DML | — | Data Manipulation Language (группа операторов управления данными в SQL) |
| DTO | — | Data Transfer Object (объект, представляющий программную сущность и хранящий соответствующие данные из полей БД) |
| HTML | — | HyperText Markup Language (стандартизированный язык разметки документов для просмотра веб-страниц в браузере) |
| IDE | — | Integrated Development Environment (интегрированная среда разрабоотки — комплекс программных средств, используемый программистами для разработки программного обеспечения) |
| JDK | — | Java Development Kit (бесплатно распространяемый компанией Oracle Corporation комплект разработчика приложений на языке Java) |
| JPA | — | **Java Persistence API (спецификация API Java EE, предоставляет возможность сохранять в удобном виде Java-объекты в базе данных)** |
| JSON | — | **JavaScript Object Notation (текстовый формат обмена данными, основанный на JavaScript)** |
| JWT | — | **JSON Web Token (открытый стандарт для создания токенов доступа, основанный на формате JSON)** |
| MVC | — | Model View Controller (архитектура проектирования приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер) |
| MVP | — | Minimum Viable Product (минимально жизнеспособный продукт — продукт, обладающий минимальными, но достаточными для удовлетворения первых потребителей функциями) |
| ORM | — | Object-Relational Mapping (технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных) |
| POJO | — | Plain Old Java Object («старый добрый Java-объект», простой Java-объект, не унаследованный от какого-то специфического объекта и не реализующий никаких служебных интерфейсов сверх тех, которые нужны для бизнес-модели) |
| REST | — | Representational State Transfer (архитектурный стиль взаимодействия компонентов распределённого приложения в сети) |
| UI | — | User Interface (пользовательский интерфейс) |

ВВЕДЕНИЕ

На данный момент информационные технологии проникли во все сферы государства, общества и бизнеса. Трудно представить организацию любого процесса без задействия современных технологий, поскольку они значительно его упрощают, систематизируют и оптимизируют. В частности, каждое уважающее себя учебное учреждение имеет свой интернет-портал, а любой сколько-нибудь значительный бизнес обязан иметь собственный сайт или приложение для ведения предпринимательской деятельности через Интернет. Более того, некоторые бизнес-услуги могли и вовсе быть немыслимы до широкого распространения веб-приложений.

Целью данной курсовой работы является разработка серверной части веб-приложения «Сервис управления прокатом электросамокатов» с использованием современных технологий на базе фреймворка Spring и архитектурного паттерна проектирования MVC [1].

Для систематизации работы и упрощения восприятия доклада процесс разработки был поделен на четыре основные части:

1. Изучение и выбор наиболее удобных и подходящих средств ведения разработки
2. Анализ предметной области
3. Разработка серверной части веб-приложения
4. Разработка клиенткого представления веб-приложения (графического интерфейса)

Приложение, созданное в результате выполнения курсовой работы должно иметь широкий и завершенный функционал для взаимодействия со спроектированной системой, базовый пользовательский интерфейс (в данном случае это сайт с документацией, который позволяет отсылать запросы серверу по разработанному API), реализовывать серверную логику написанную на Java с помощью Spring Framework, работать с реляционной базой данных и соответствовать архитектуре MVC [2].

1. ОБЩИЕ СВЕДЕНИЯ
   1. Обоснование выбора средств ведения разработки

Первым этапом любой разработки является выбор и установка соответсвующего программного обеспечения. Так в качестве среды разработки была выбрана IntelliJ IDEA Ultimate. Преимуществами IDEA является статус де факто стандарта при выборе IDE (Integrated Development Environment) в разработке на Java. В роли СУБД была выбрана PostgreSQL, так как она совершенно бесплатна, широко распространена и имеет продвинутый функционал. Для организации ORM (Object-Relational Mapping) и выполнения запросов к базе данных из среды Java был применён фреймворк Hibernate – самое популярное решение на данный момент. Версия JDK (Java Development Kit) выбрана 11, так как она стабильна и достаточно современна, включает в себя все самые важные и необходимые функции языка Java [3]. А где Java, там почти всегда и Spring [4] – стандарт при выборе фреймворка для разработки enterprise-приложений на Java. Развертываться приложение будет на Spring Boot, поскольку он обеспечивает простейшую «из коробки» интеграцию с остальным Java-приложением на Spring. В качестве системы сборки используется Maven, как наиболее популярный и гибко кастомизируемый инструмент. Для тестирования использовалась библиотека модульного тестирования JUnit и некоторые инструменты фреймворка Mockito [5], что опять же очень стандартно для разработки на Java.

При разработке UI (User Interface) применялся Swagger [6] – Java-реализация спецификации OpenAPI 3.0, которая выполняет автоматическую генерацию документации на основе разработанного API. В результате предоставляя одностраничный сайт, написанный на HTML (HyperText Markup Language) и CSS (Cascading Style Sheets), на котором можно ознакомиться со всеми эндпоинтами проекта и их описанием, а так же отправить тестовые запросы серверу непосредственно из вкладки в браузере. Swagger был выбран из-за своего удобства и простоты, так как основной упор проекта – это бэкенд.

1. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

Разработанное приложение несёт собой демонстративную, учебную цель. Оно подразумевается для использования в ознакомительных целях – в качестве прототипа, либо примера приложения на котором можно изучить то, как архитектурно и в деталях устроен современный программный продукт, а также то, из каких компонентов состоит серверная часть в общем цикле разработки распределенного клиент-серверного приложения [7].

Целевой аудиторией данного программного продукта могут быть начинающие backend-разработчики, желающие разобраться в устройстве полноценного RESTful веб-сервиса, либо frontend-разработчики, которым для создания своего pet-проекта может понадобиться внешний API, с которого можно брать и отсылать какие-либо данные.

1. ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ
2. Анализ предметной области

Сбор информации был начат с изучения уже существующих на рынке аналогов. Для этого было скачано и изучено популярное приложение для проката электросамокатов «Whoosh». На рисунке 1 представлен весь предоставляемый приложением функционал, основные элементы которого – выбор ближайшей точки проката и соответствующего самоката (рисунок 2), настройка профиля пользователя и просмотр истории поездок (рисунок 3) и возможность аренды как по подписке, так и по поминутной фиксированной ставке (рисунок 4), которые являются ядром всего приложения, на чем базируются все подобные сервисы.

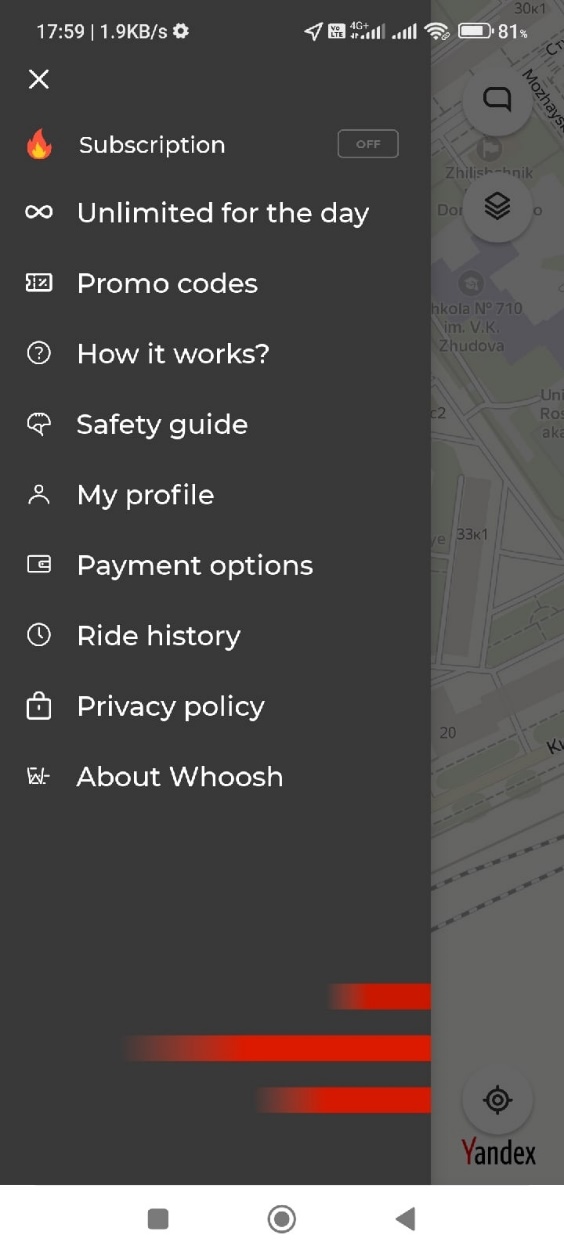


Рисунок 1 – Полный перечень опций, доступных в приложении «Whoosh»

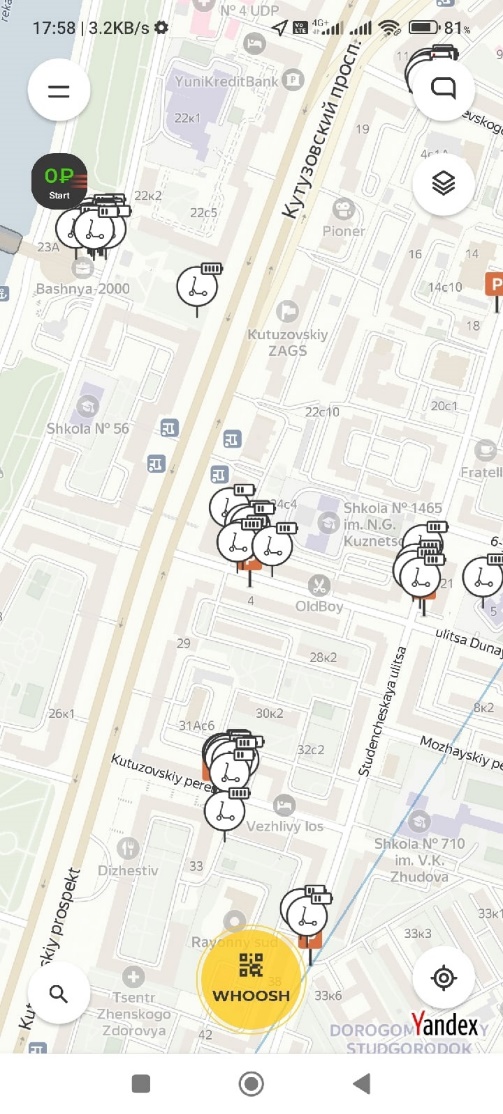


Рисунок 2 – Отображение ближайщих точек проката и заряда самокатов

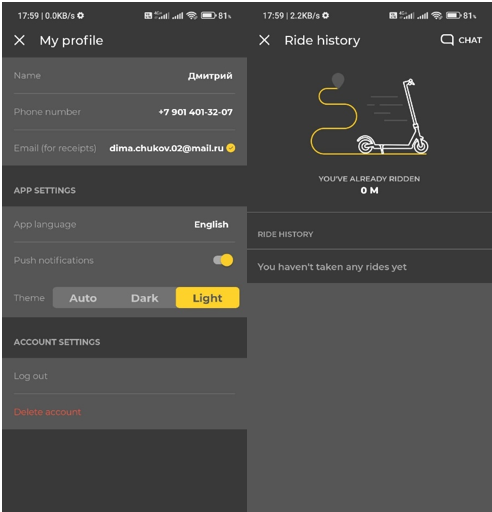


Рисунок 3 – Настройка профиля пользователя и просмотр истории поездок

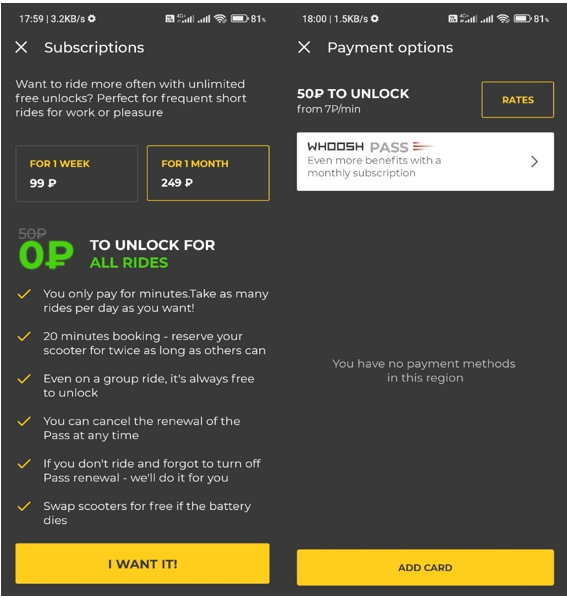


Рисунок 4 – Возможность покупки подписки или оплаты по тарифу

Проанализировав возможности данного сервиса можно сделать вывод, что для конкурентноспособности разрабатываемого приложения его MVP (Minimum Viable Product) должен обладать уже перечисленными выше функциями, а именно:

1. Регистрация, авторизация, редактирование информации пользователя
2. Получение информации о ближайших точках проката и заряде самокатов на них
3. Возможность проката как по подписке, так и по поминутному тарифу
4. Просмотр истории завершенных поездок

При этом такие «фичи» как промо коды, безлимит на один день, гайд по использованию, информация о приложении и политика конфиденциальности кажутся второстепенными или вовсе не относящимися к труду разработчика.

1. Разработка серверной части веб-приложения

Данное приложение разрабатывалось по архитектуре MVC [8], в разработке бэкенда на Java это интерпретируется следующим образом (рисунок 1): все классы разделены в своем взаимодействии на 3 логических уровня – слой модели (это сущности, DAO (Data Access Object) или JpaRepository, и сервисы для работы над этими сущностями), слой контроллера (в современной разработке принято делать контроллеры максимально тонкими, чтобы они лишь маппили запрос с клиента на соответствующий метод сервиса) и слой представления (так как приложение представляет RESTful API, то им можно считать возвращаемые в формате JSON результаты запросов).

Концептуально работа приложения выглядит так: запросы приходят по API на контроллеры, контроллеры передают их соответствующим методам сервисов, а сервисы оперируют над сущностями, полученными из базы данных через DAO [9].

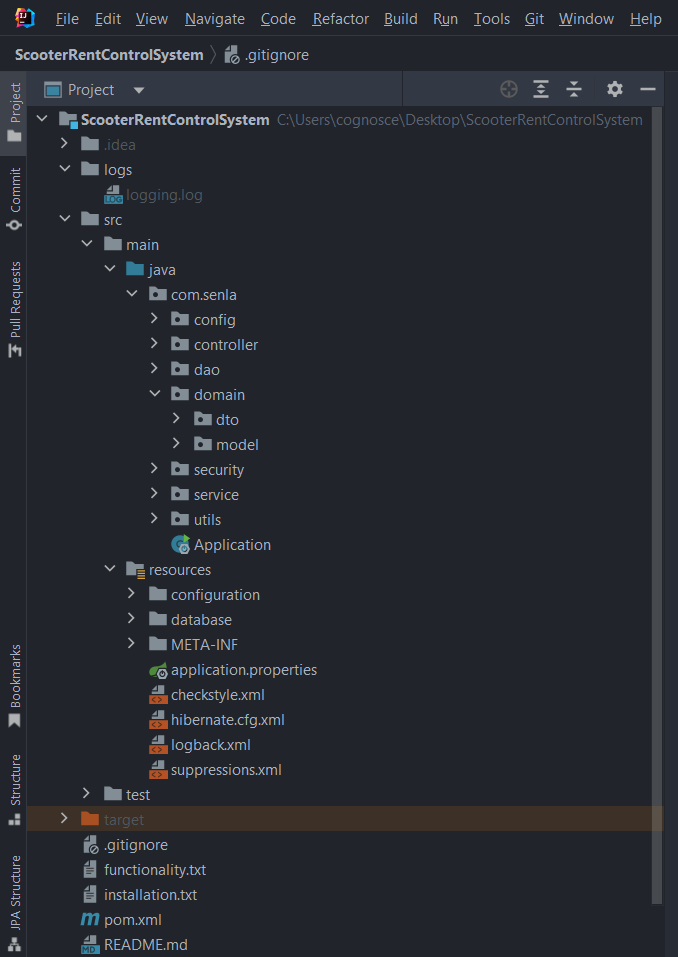
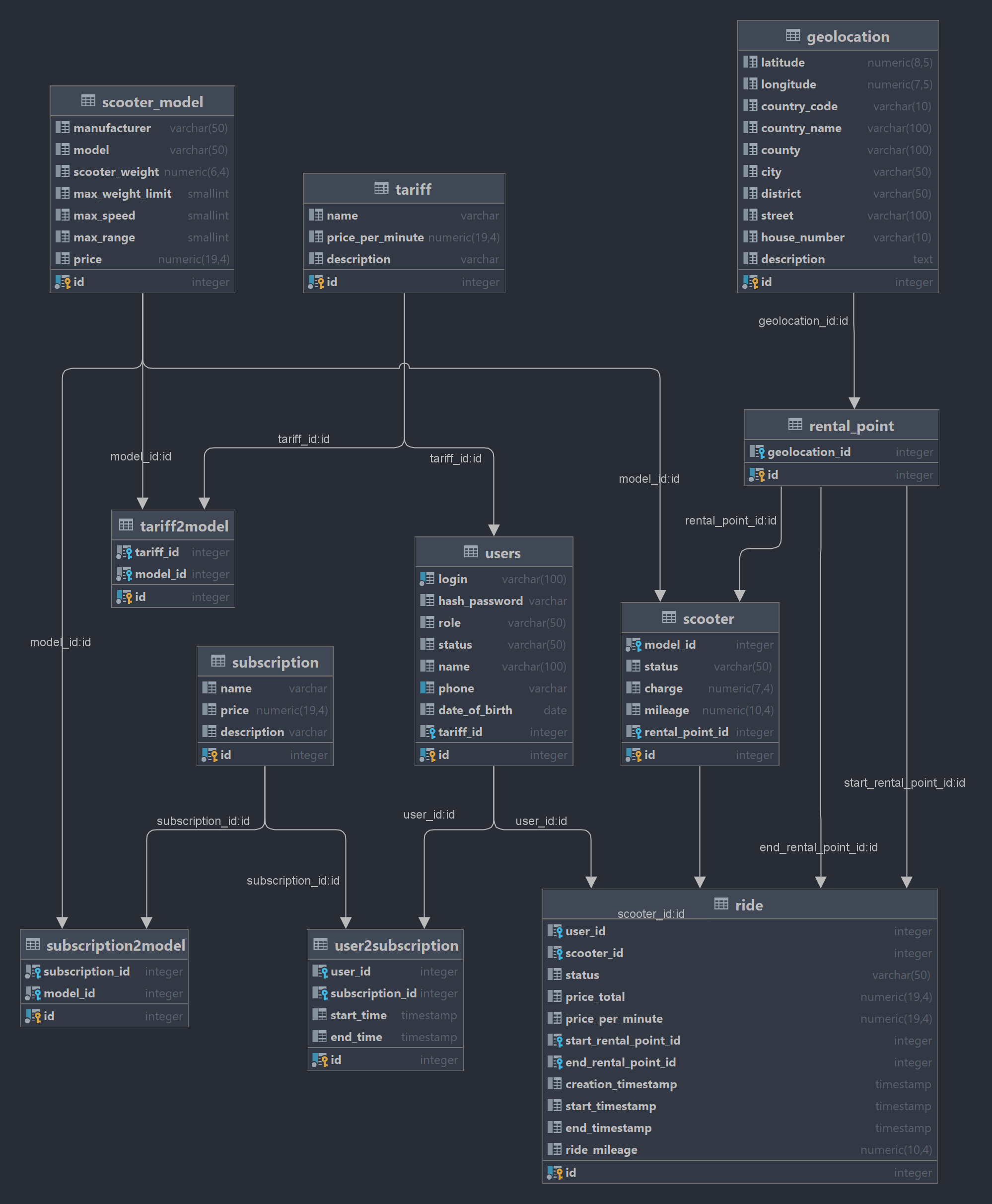


Рисунок 5 – Структура пакетов проекта в соответствие с MVC

* + 1. Проектирование базы данных

Проведя анализ предметной области и выделив функциональные требования к проекту, следующим этапом необходимо провести выделение основных сущностей в будущей системе.

Так, на рисунке 2 представлена разработанная схема базы данных, включающая в себя такие сущности как: геолокация, точка проката, модель самоката, экземпляр самоката, тариф, подписка, пользователь, поездка, а так же связи различные между ними.

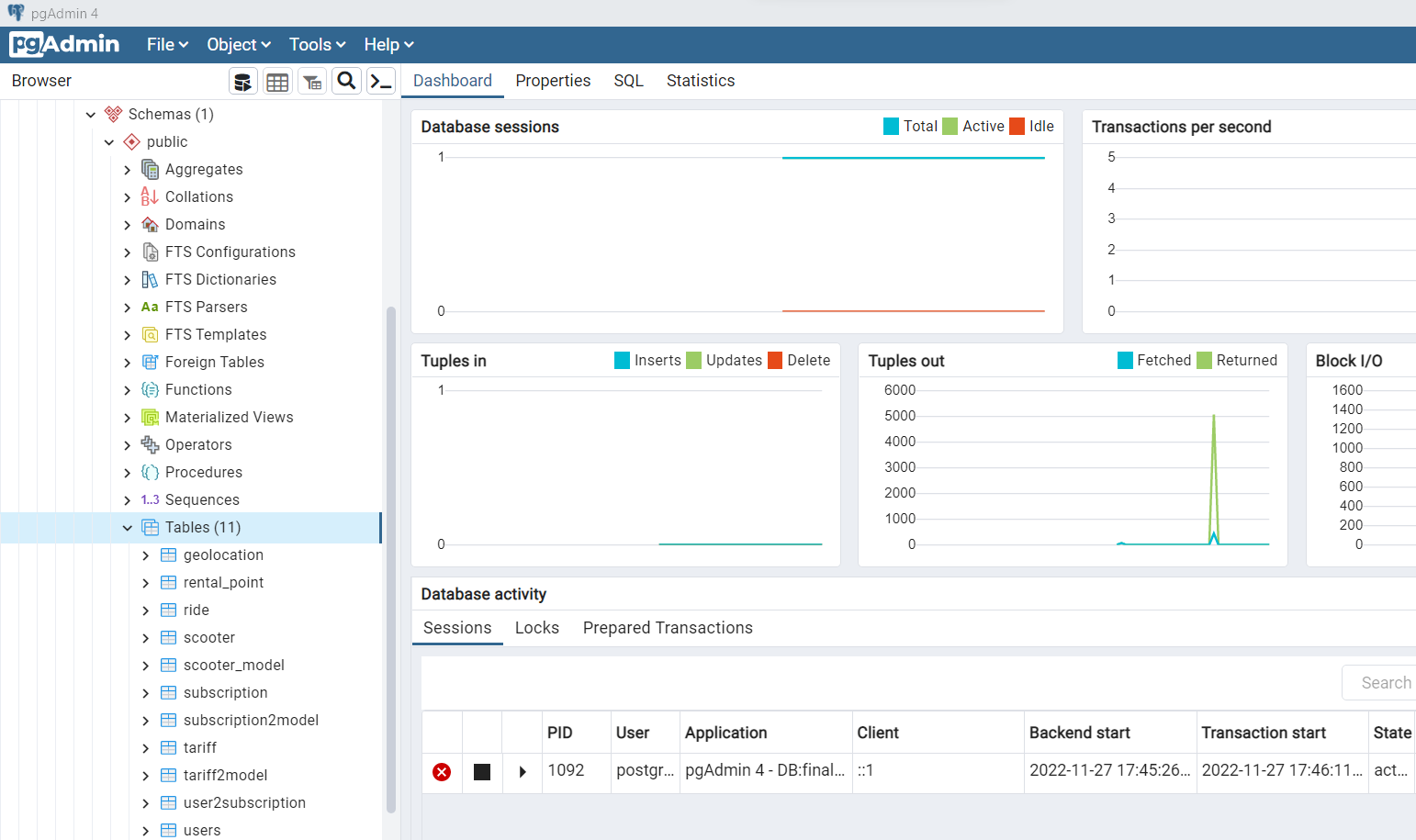


**Рисунок 6 – Схема спроектированной базы данных**

**База данных была создана в PostgreSQL с помощью двух скриптов – первый скрипт задает структуру таблиц с помощью DDL-команд, а второй – заполняет наобходимыми инициализирующими данными используя DML-команды в SQL [10].**

**На листинге 1 можно наблюдать отрывок из DML-скрипта, который вносит в таблицу данные о точках проката (настоящие географические объекты с соответствующими координатами на карте мира), о тарифах и подписках, а так же о моделях самокатов (их реальные характеристики были взяты с Яндекс.Маркета).**

**Для удобства мониторинга базы данных использовался графический интерфейс для PostgreSQL – pgAdmin [11]. Так, на рисунке 3 продемонстирована работоспособность созданной базы данных.**

****

**Рисунок 7 – Представление базы данных в pgAdmin**

1. Разработка слоя модели

На данном этапе сущностям в SQL-таблицах было разработано отображение в виде POJO-объектов в Java (рисунок 4). Hibernate [12] в свою очередь берёт на себя управление запросами к БД из Java-окружения и контроль согласованности между этими данными.

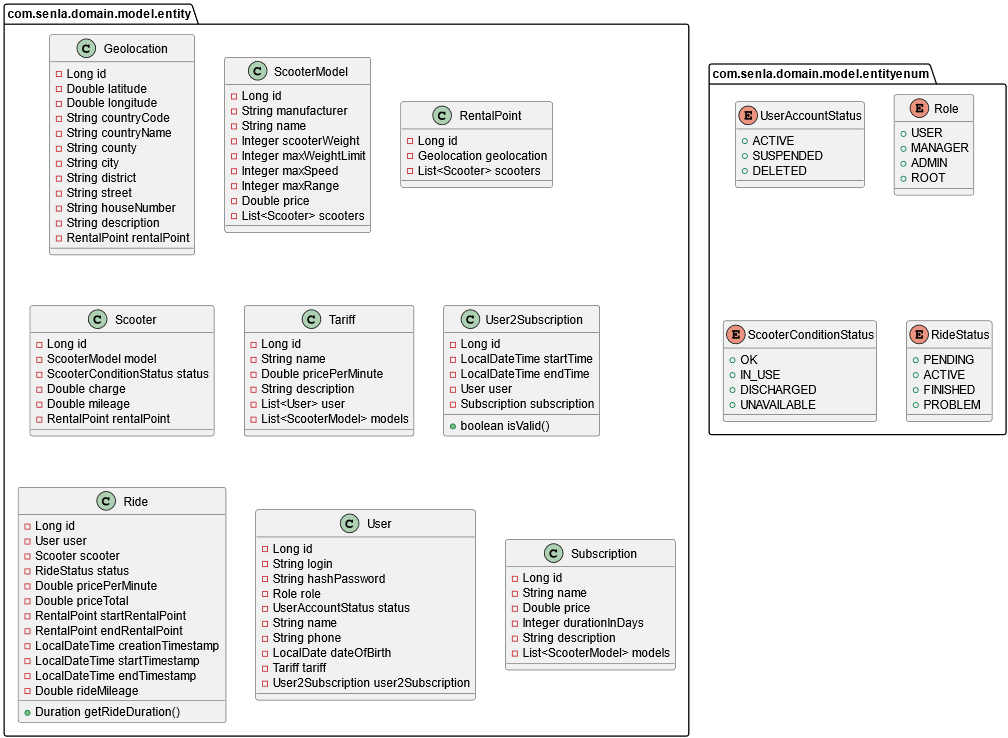


Рисунок 8 – Диаграмма сущностей разработанного приложения

Все операции над сущностями для базы данных происходят через класса-посредника – DAO. Каждый из таких классов является бином в контексте Spring, и внедряет в себя зависимость экземпляра entityManager – объекта, который управляет транзакциями в Hibernate. DAO преимущественно используются для выполнения базовых CRUD операций (которые с помощью механизма наследования были вынесены в базовый абстракный класс *AbstractDaoImpl* и переиспользованы во всех дочерних классов воизбежание дублирования), но так же поддерживают и более сложные запросы с использованием динамических параметров или JOIN благодаря Criteria API. C реализацией DAO на примере класса *RentalPointDaoImpl* можно ознакомиться на листинге 2. В современных проектах от использования DAO всё чаще отказываются в пользу применения JpaRepository из SpringData, но в образовательных целях использования DAO всё еще уместно. Одним из дальнейших этапов развития проекта так же рассматривается миграция на Spring Data.

Центральным местом всего приложения, реализующим основную бизнес-логику, является слой сервисов. Рассмотрим для примера листинг 3, на котором продемонстрированы 4 метода из сервиса поездок. Метод *createRideWithTariff* создает поездку по тарифу. Внутри этого метода у пользователя берётся текущий тариф, после чего вычисляется цена в минуту для предстоящей поездки. Далее идет проверка на корректность возможностей тарифа и статуса самоката, а затем создается объект поездки, сохраняется в базу данных и начинается 30-секундный отсчёт. Поскольку тарифом предусмотрена только базовая ставка, поверх которой могут накладываться дополнительная наценка, то пользователю предоставлен выбор – принять или не принимать поездку. Если пользователь не успеет принять поездку – она удаляется. За это отвечает метод *startCountdownForDeletionOfPendingRidesOfTheUser*, который создает отдельный поток в котором запускается таймер, и по истечению отсчета в асинхронном режиме удаляет ожидающую поездку пользователя. При этом всё это время пользователь может продолжать отсылать другие запросы на сервер.

Помимо этого, в данном сервисе расположены методы начала и завершения поездки, которые фиксируют временные метки для точного расчета итоговой цены в зависимости от поминутной ставки и целого количества минут, проведённых в пути.

Храня персональную информацию пользователей нельзя пройти мимо вопроса защиты данного приложения [13]. Для этих целей был пременён модуль Spring Security [14] и библиотека JJWT, генерирующая JWT-токены [15]. На листинге 4 представлен отрывок из класса, предоставляющего и валидирующего токены. В данном приложении не задействован Authorization Server, применяется базовая внутренняя password grant аутентификация, при которой пользователей передает свои логин и пароль, а в ответ получает временный JWT access token, по которому система будет его идентифицировать.

1. Разработка слоя контроллера

Контроллеры – компоненты системы, помеченные аннотацией @*RestController* и выполняющие связывающую функцию между эндпоинтами API и методами сервисов. В данном приложение представлены восемь контроллеров (рисунок 9).

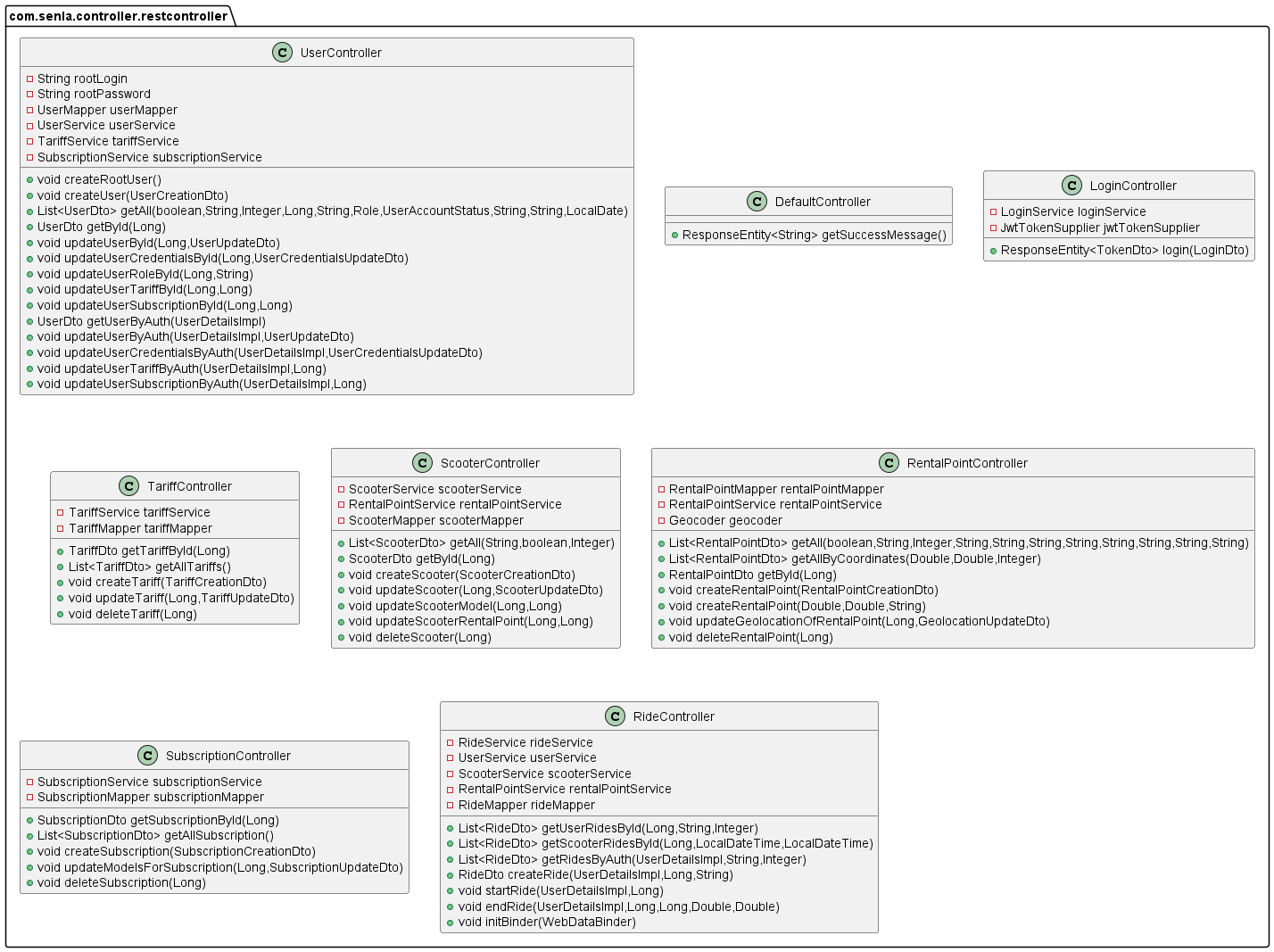


Рисунок 9 – Диаграмма контроллеров разработанного приложения

Шесть из них соответствуют работе с программнымим сущностями – пользователями, точками проката, самокатами, тарифами и подписками, а так же поездками. Оставшиеся два – *DefaultController* выполняет роль индикатора, свидетельствующего что запуск приложения был успешен, и *LoginController* – отвечает за аутентификацию пользователей.

Все эндпоинты проектировались в соответствие с архитектурным стилем REST. Пример того, как они выглядят представлен на рисунке 10.

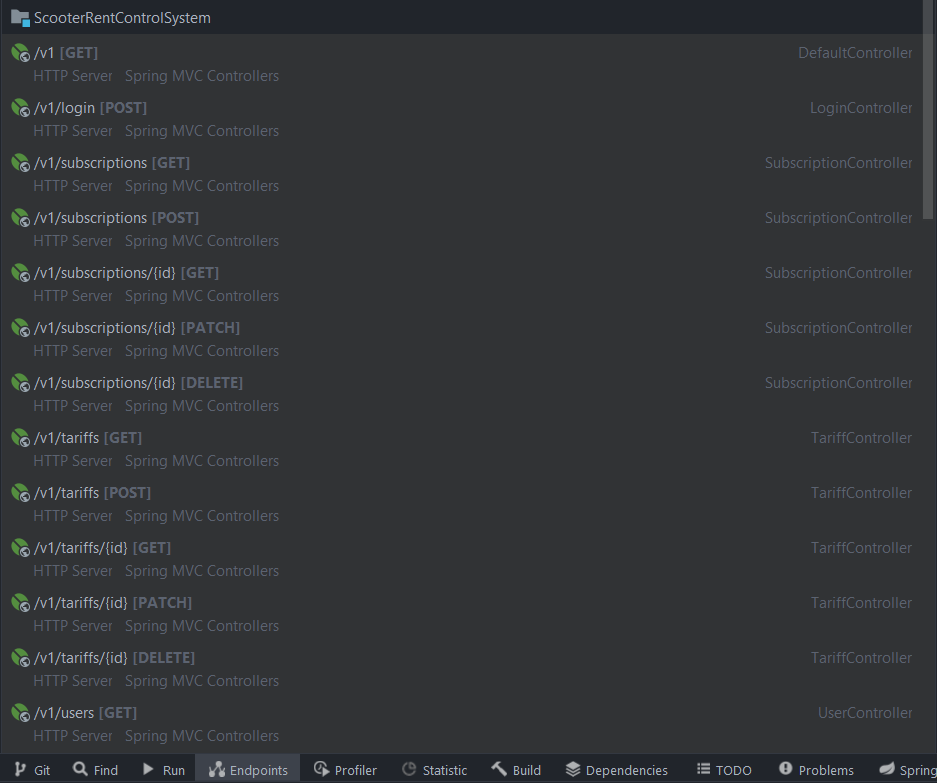


Рисунок 10 – Отрывок из списка эндпоинтов разработанного API

1. ****Разработка слоя представления****

Поскольку проект представляет RESTful API и не отдает HTML-страницы непосредственно (что позволяет на его основе построить распределенную систему и уменьшает зацепление между компонентами), то можно сказать, что слой представления – это данные в формате JSON, которые получает клиент.

Ответ в JSON формате формируется на основе DTO, которые с помощью соответсующих библиотек проходит процесс конвертации из/в POJO-объекты. Со списком разработанных DTO можно ознакомиться на рисунке 11.

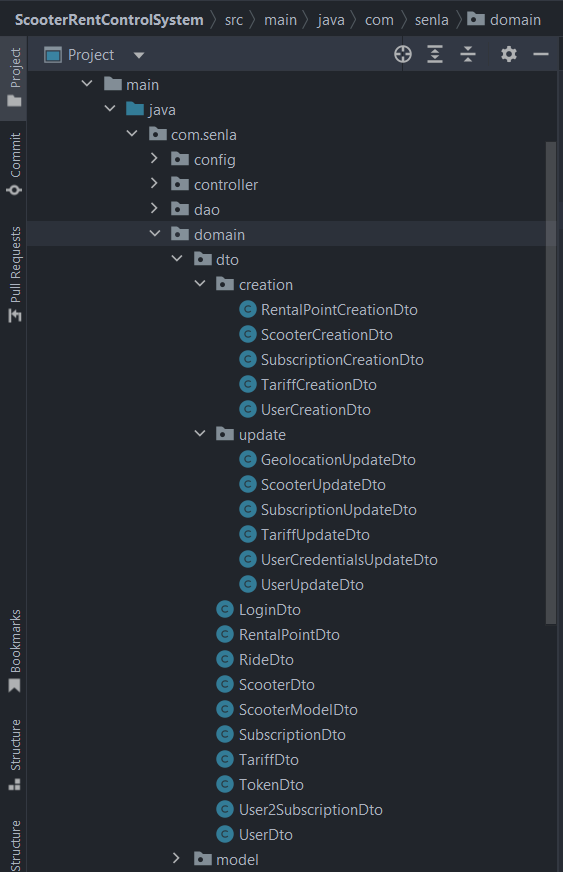


Рисунок 11 – DTO-классы проекта

Обратим внимание на класс RentalPointDto (листинг 5). В нем мы явно указываем ту структуру, которую собираемся передать на клиент. Таким образом, при отравке запроса, ответ будет содержать именно те поля, которые были описаны в DTO. Пример такого запроса представлен на рисунке 12.

*Листинг 5 – RentalPointDto.java*

*@Setter  
@Getter  
public class* RentalPointDto {  
  
 *private* Long id;  
 *private* Integer currentNumberOfScooters;  
 *private* Geolocation geolocation;  
 *private* Double distanceToClientInKm;  
 *private* List<ScooterDto> scooters;  
}

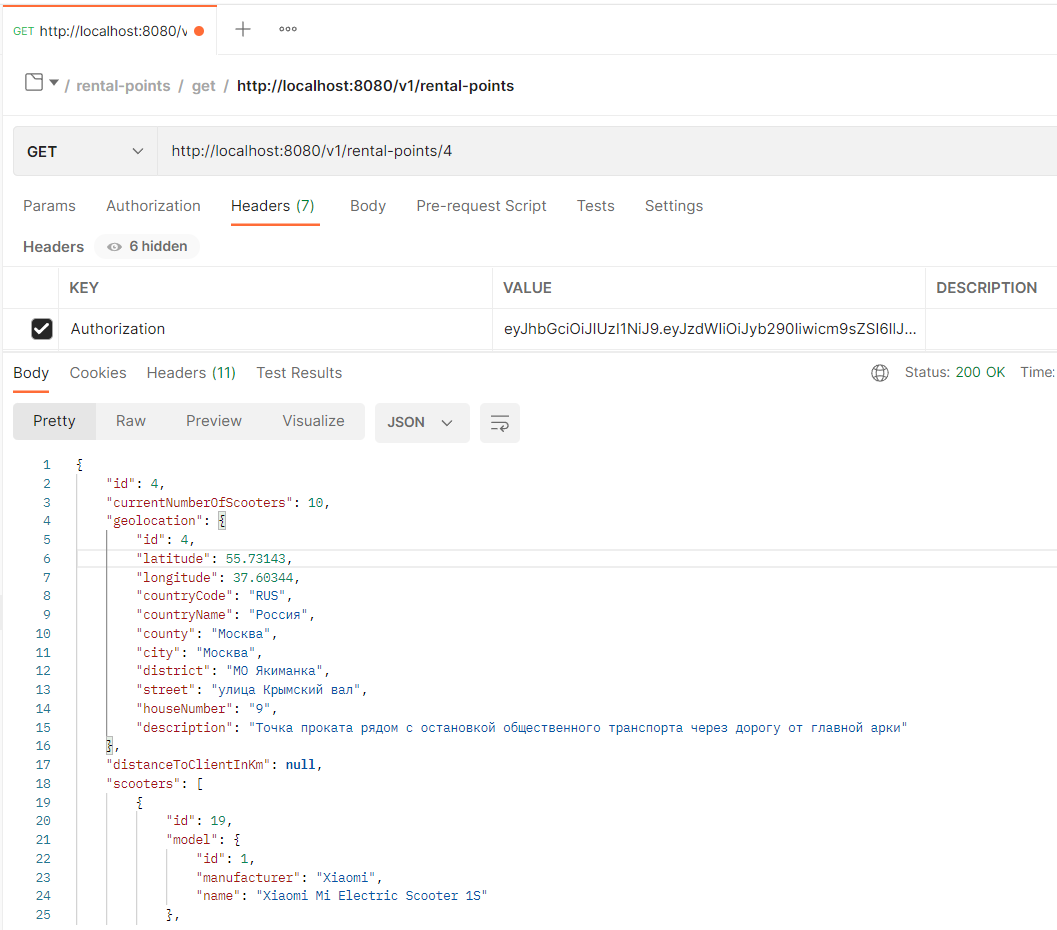


Рисунок 12 – Отображение RentalPointDto в JSON-формате на клиенте

1. Разработка юнит тестов

Важным этапом разработки любого программного продукта является тестирование [16]. Оно обеспечивает сохранность исполнения тех контрактов, которые закладывались в программные модули при их разработке, и в случае изменения каких-либо компонентов системы своей неисправностью сигнализируют о критических нарушениях логике работы программы.

На данном этапе развития приложения реализованы только модульные тесты [17], покрывающие главным образом ключевые сценарии работы слоя сервисов. Все они исправны, в чем можно убедиться на рисунке 13.

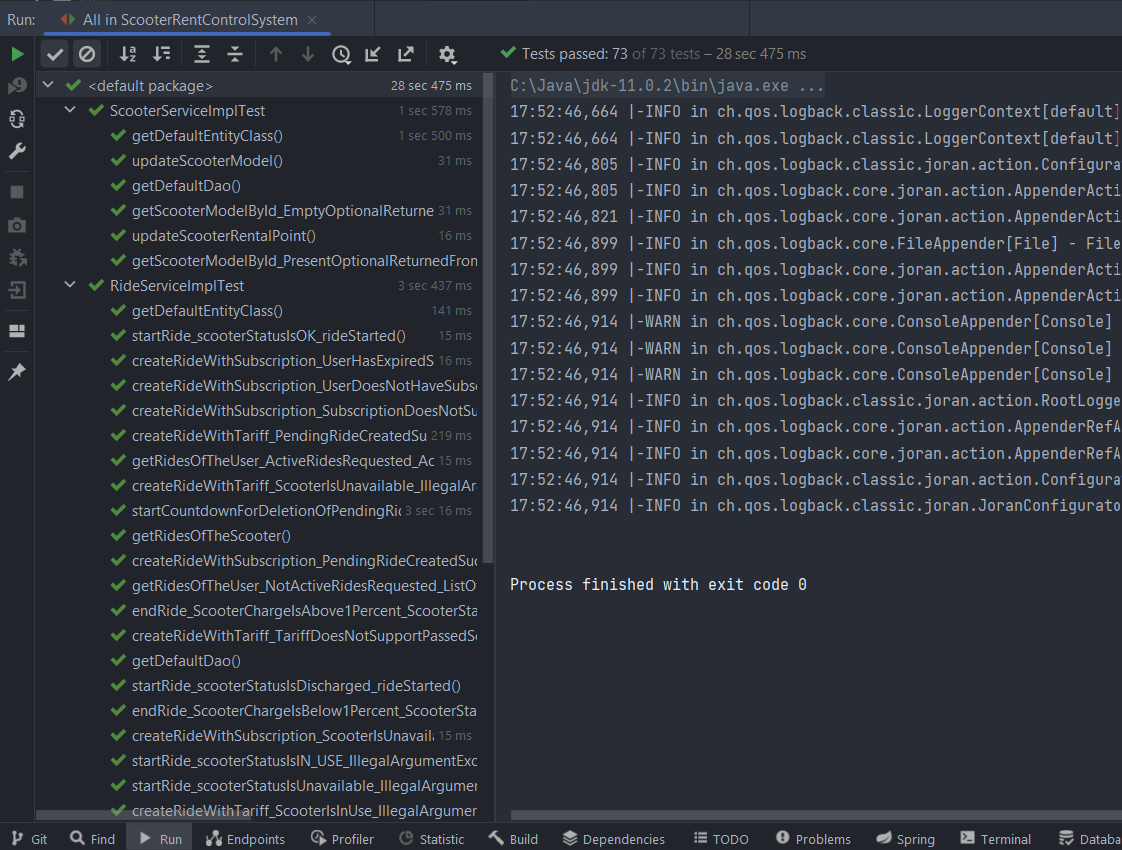


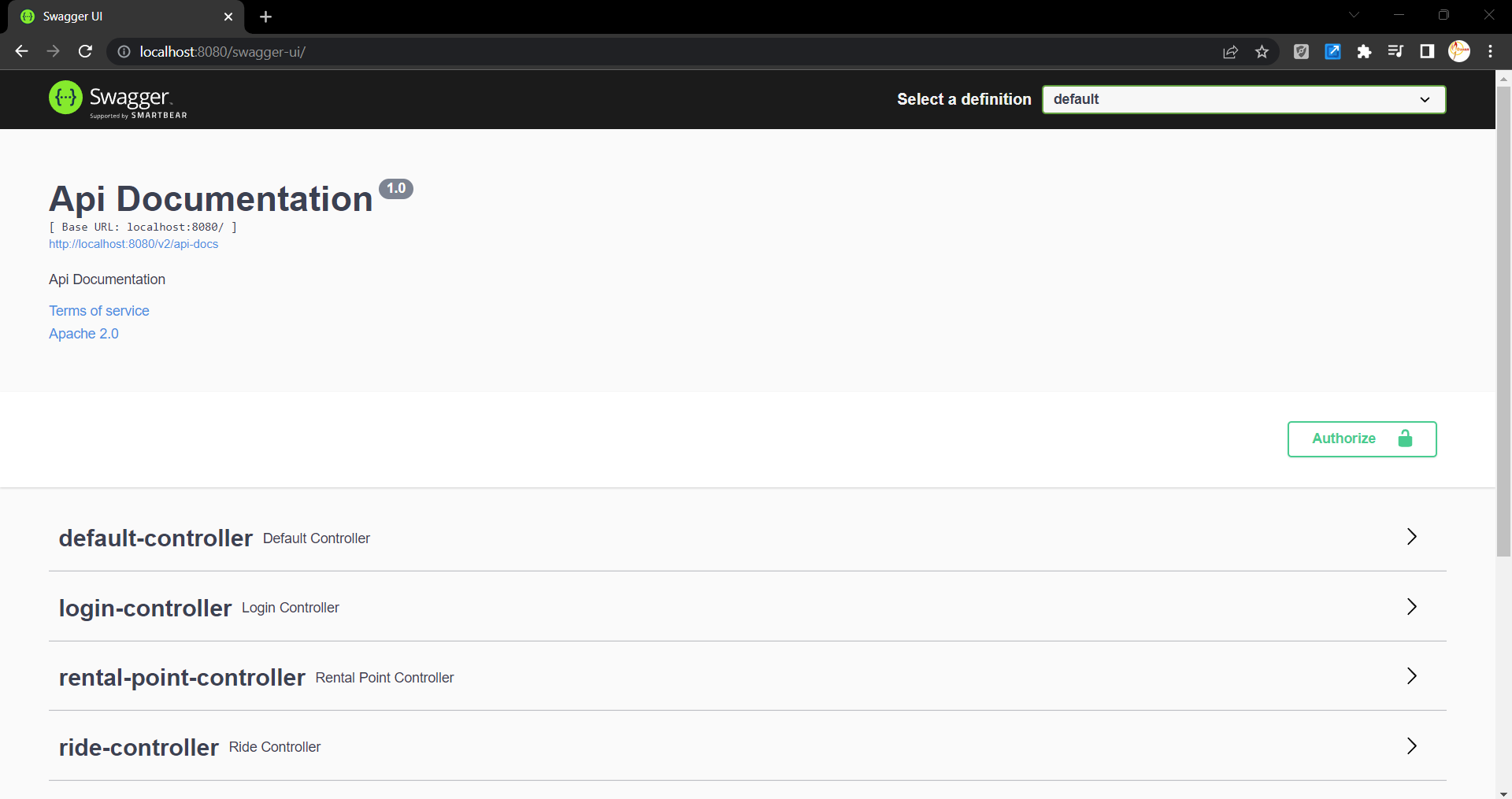
Рисунок 13 – Исправное функционирование тестов приложения

1. Разработка клиентского представления веб-приложения

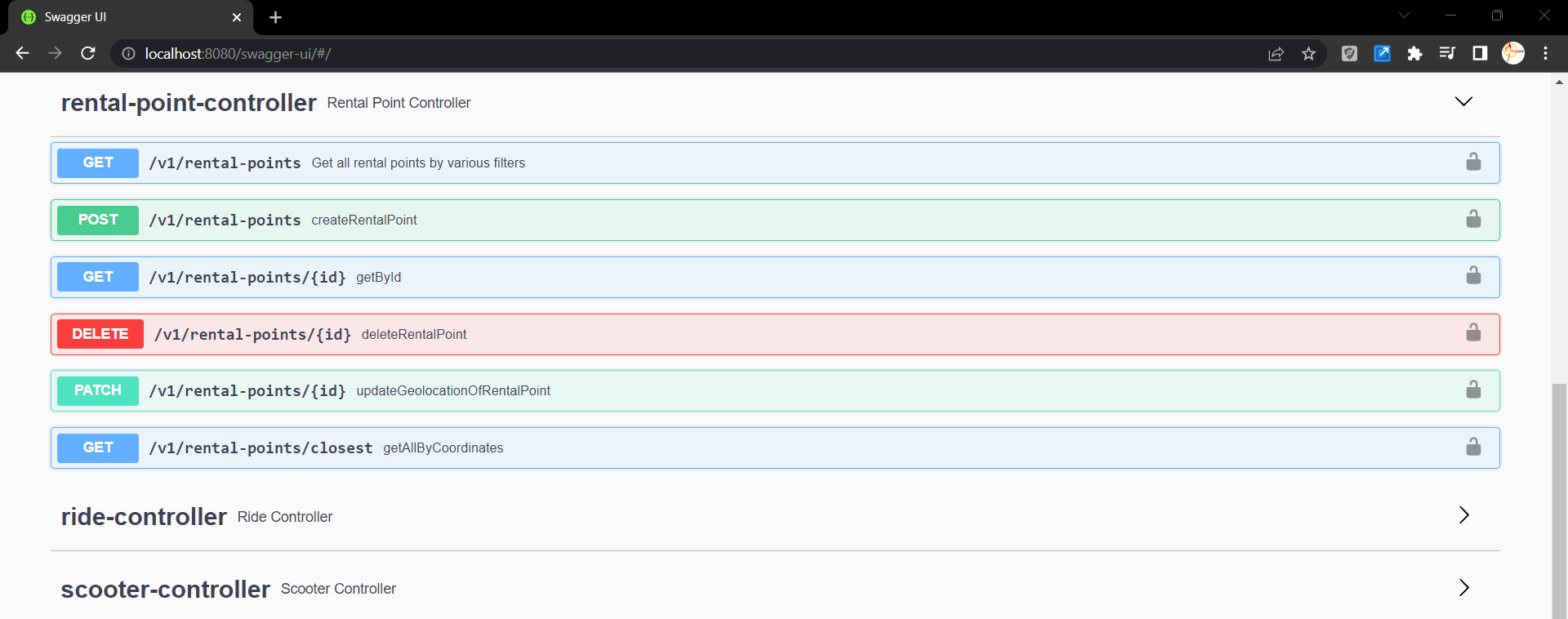
Разработкой клиентской части приложения обычно занимаются разработчики отдельной специализации – фронтендеры. Они с помощью javascript-фреймворков React, Angular или Vue.js отрисовывают внешний вид веб-приложения в соответствии с разработанным бэкендерами API [18].

Целью данной курсовой работы является разработки именно серверной части интернет-ресурса, поэтому клиентскому представлению было уделено не много внимания и в качестве самого быстрого и простого решения был интегрирован Swagger.

Swagger предоставляет документацию по имеющимся у API эндпоинтам, включая структуру DTO которые по ним передаются, коды ответов и т.д., в виде удобной веб-страницы [19]. С конфигурационным файлом Swagger можно ознакомиться на листинге 6, при запуске он автоматически с помощью HTML и CSS генерирует все необходимые файлы. Запустив приложение и перейдя по адресу http://localhost:8080/swagger-ui/, можно начать изучать возможности и структуру проекта, а также выполнять тестовые запросы (рисунки 14 и 15).

****

**Рисунок 14 – Swagger-страница разработанного приложения**

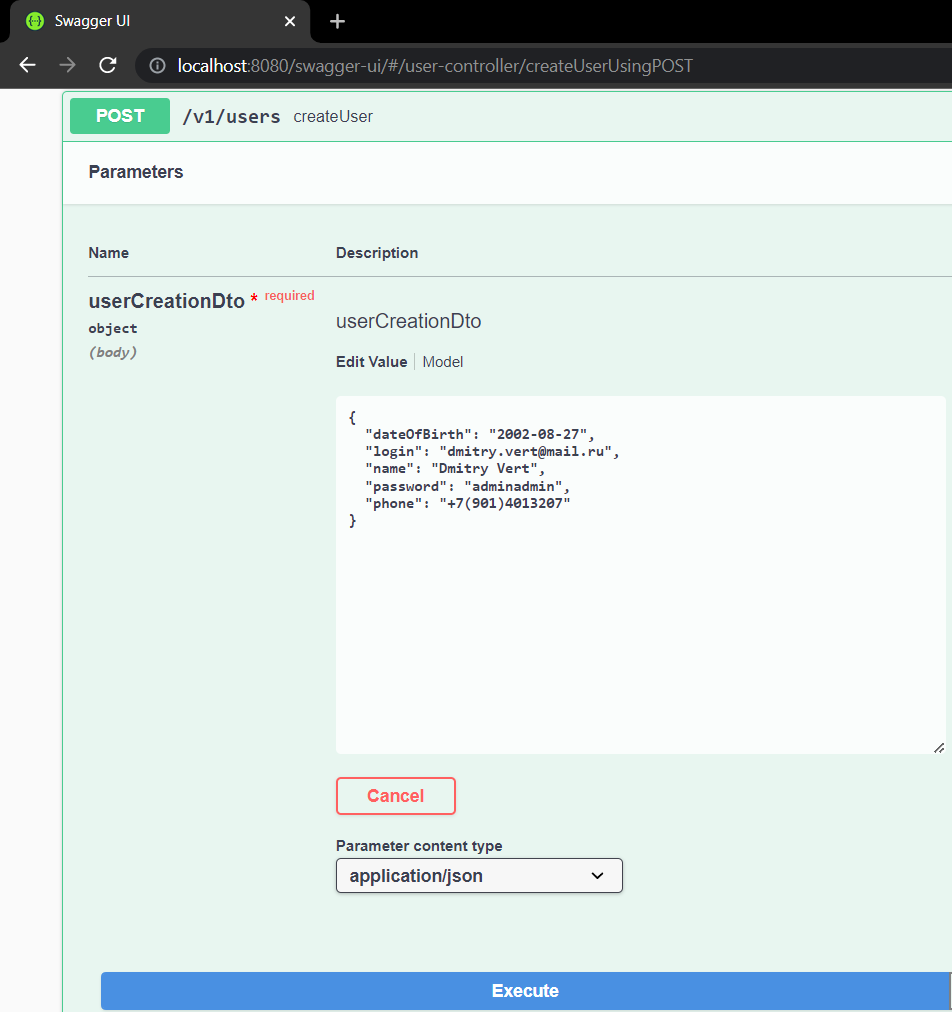
****

**Рисунок 15 – Визуальное представление эндпоинтов для точек проката**

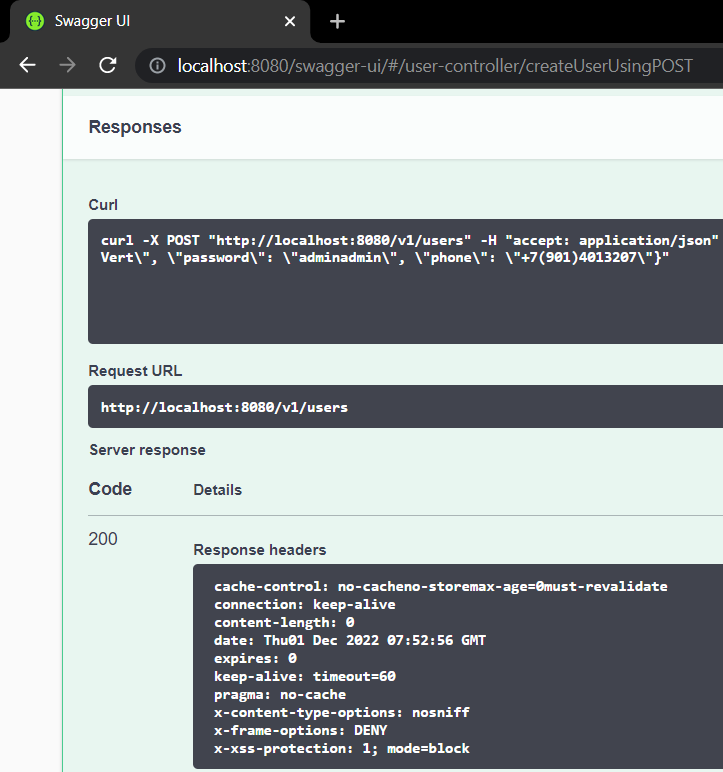
1. Обзор разработанного программного продукта

Проведём обзор разработанного функционала опираясь на основные требования, заявленные на этапе анализа предметной области.

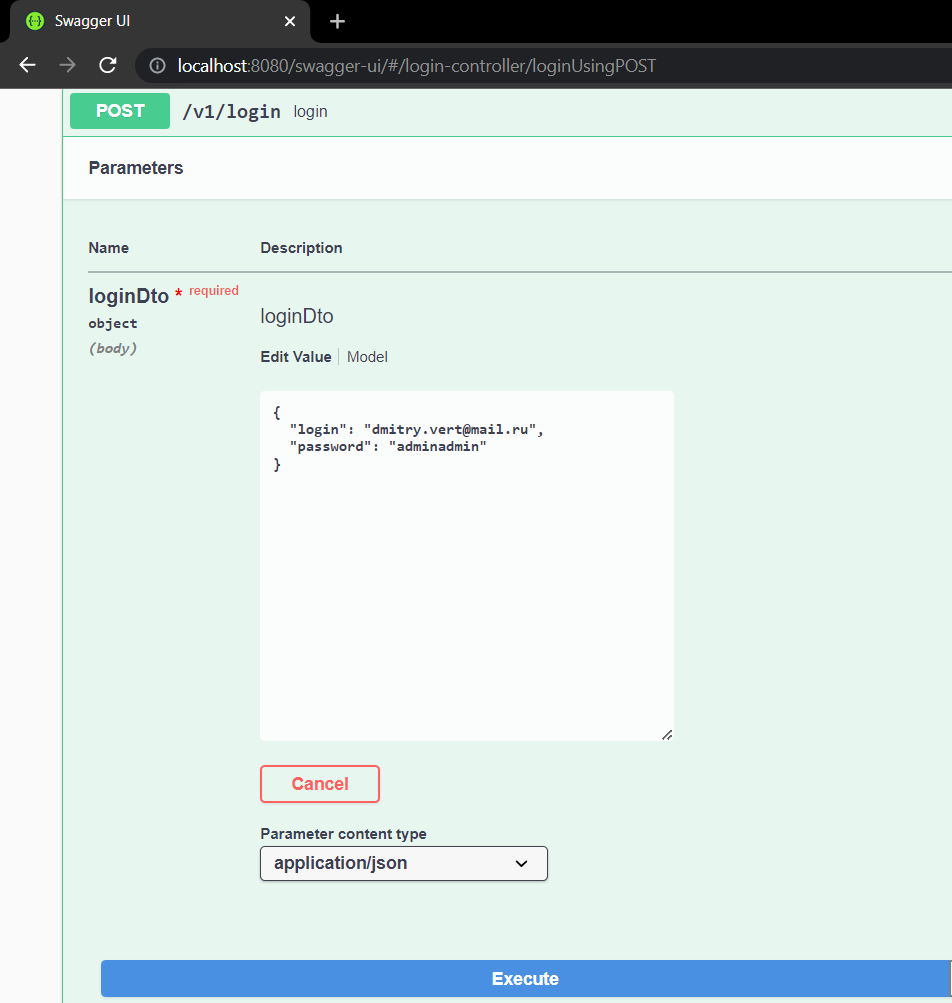
Так, на рисунках 16-23 можно наблюдать процесс регистрации, авторизации и изменение личных данных пользователя.



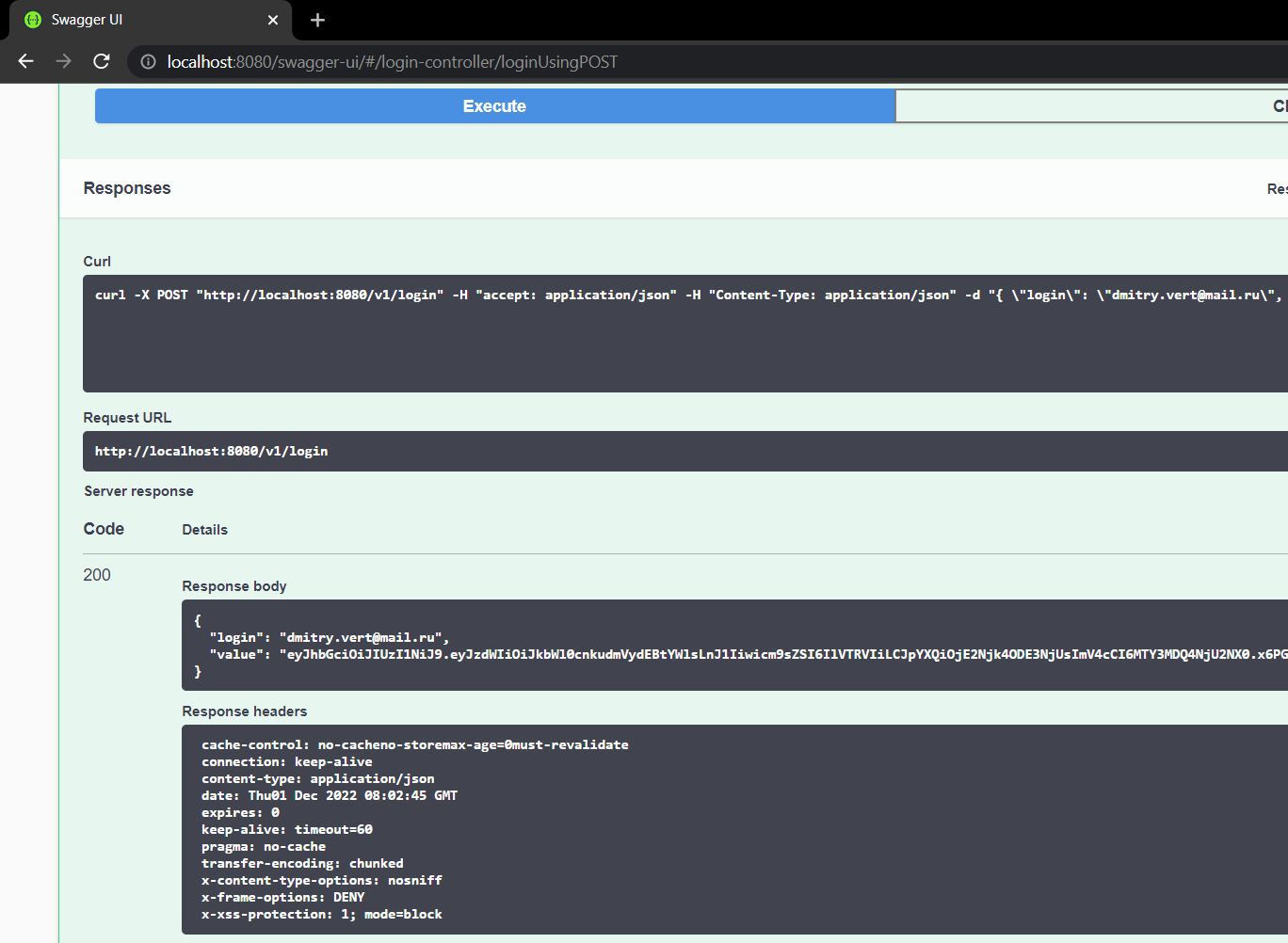
**Рисунок 16 – Форма создания нового пользователя**



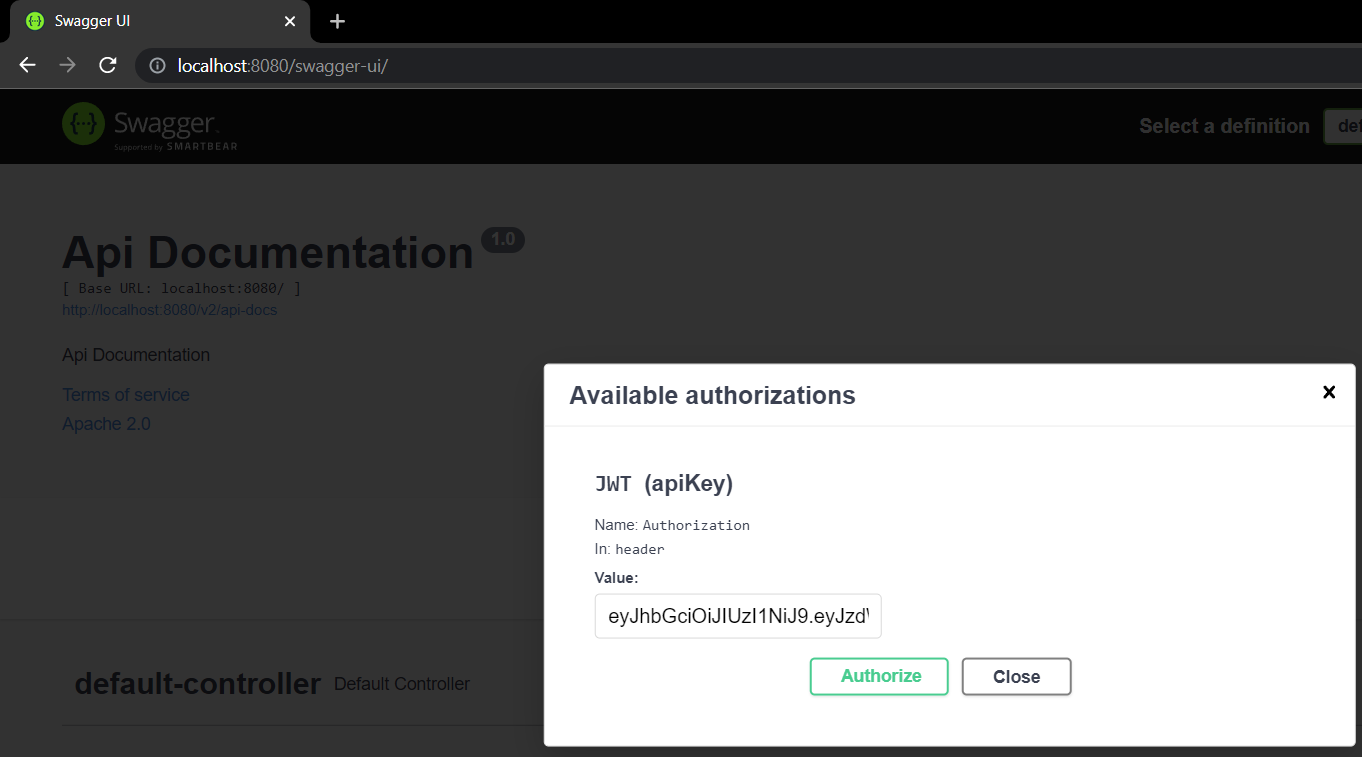
**Рисунок 17 – Успешное создание нового пользователя**



**Рисунок 18 – Форма авторизации пользователя**



**Рисунок 19 – Успешная авторизация**



**Рисунок 20 – Сохранение JWT-токена для аутентификации дальнейших запросов**

Приложение предоставлет возможность обновить параметры входа (логин и пароль), имя, номер телефона, а также почту. Для примера, обновим имя пользователя (рисунки 21-23).

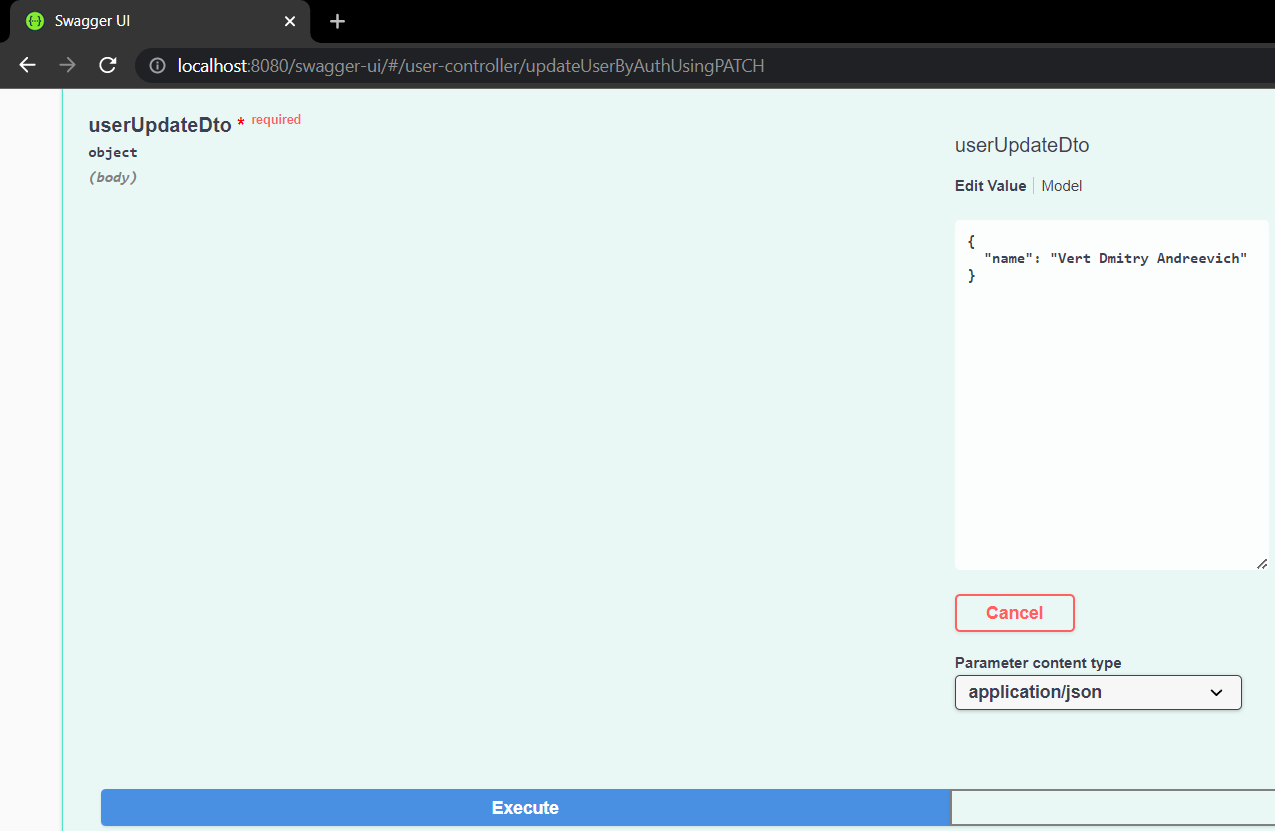


Рисунок 21 – Заполненная форма обновления имени пользователя

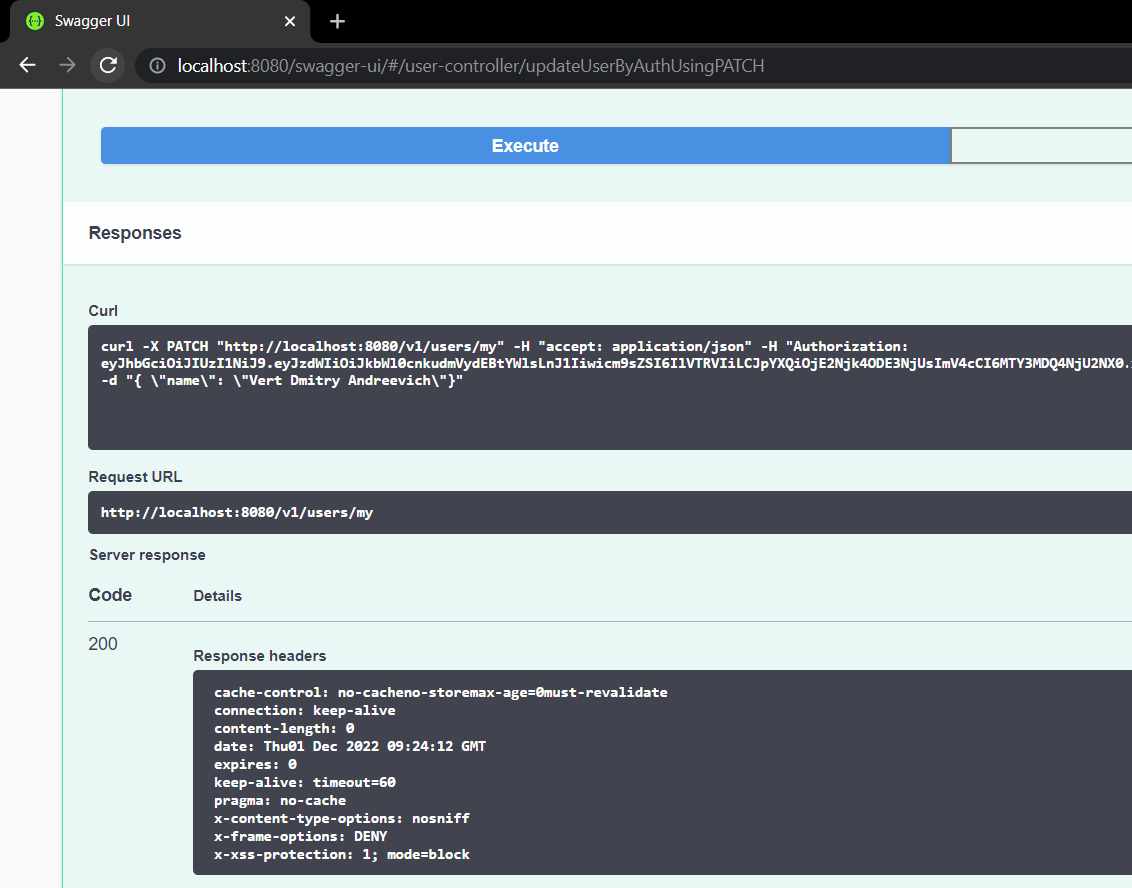


Рисунок 22 – Успешное обновление пользователя

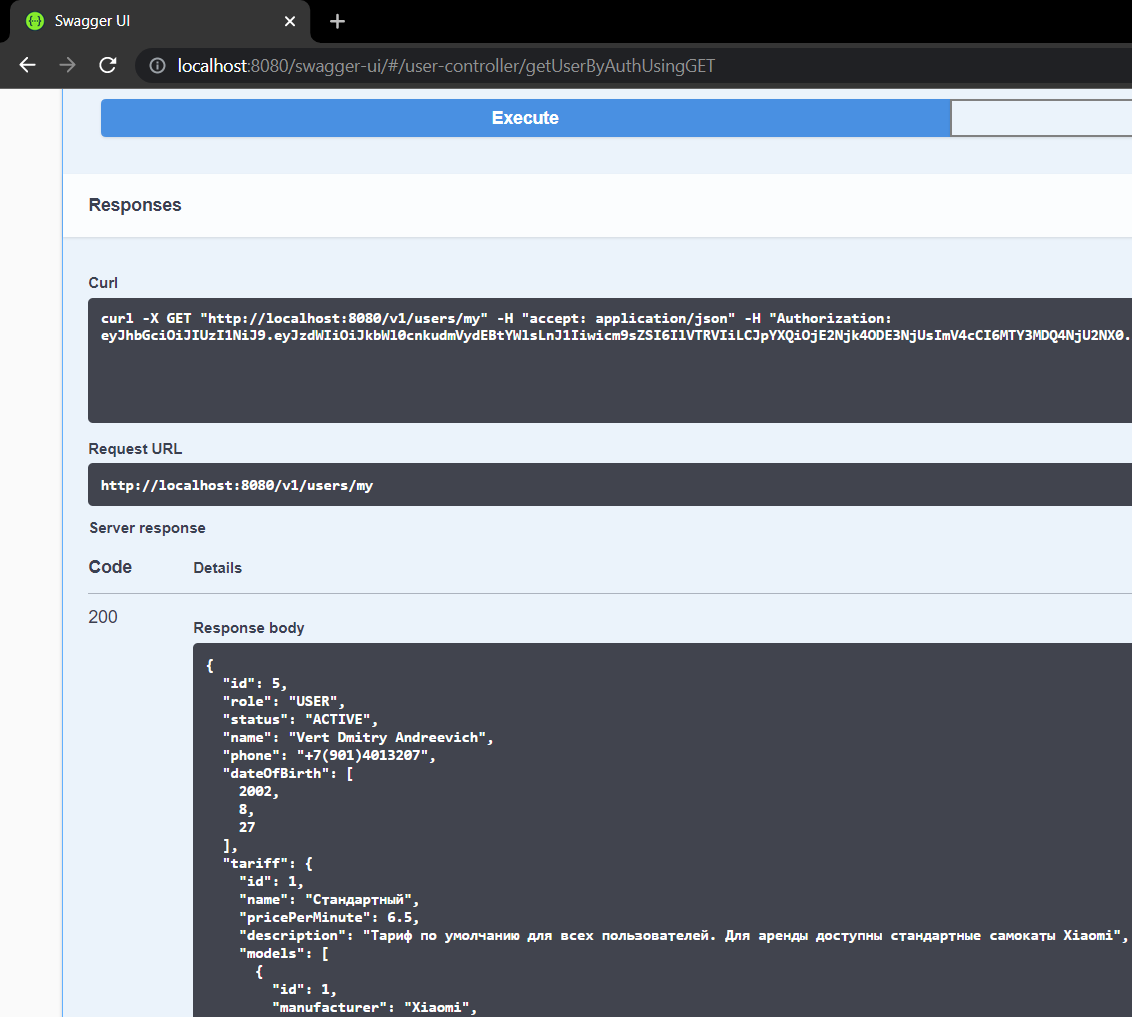


Рисунок 23 – Получение информации о своем профиле

Следующим этапом выполним проверку функционала по поиску точек проката и получению данных от них. В качестве примера примем мои координаты как клиента равными (55.7390918; 37.5439226), что приблизительно соответствует станции метро «Студенческая» в г. Москве.

На рисунках 24-28 представлен процесс поиска ближайших точек проката по переданным координатам и просмотр имеющихся самокатов на конкретно выбранном пункте.

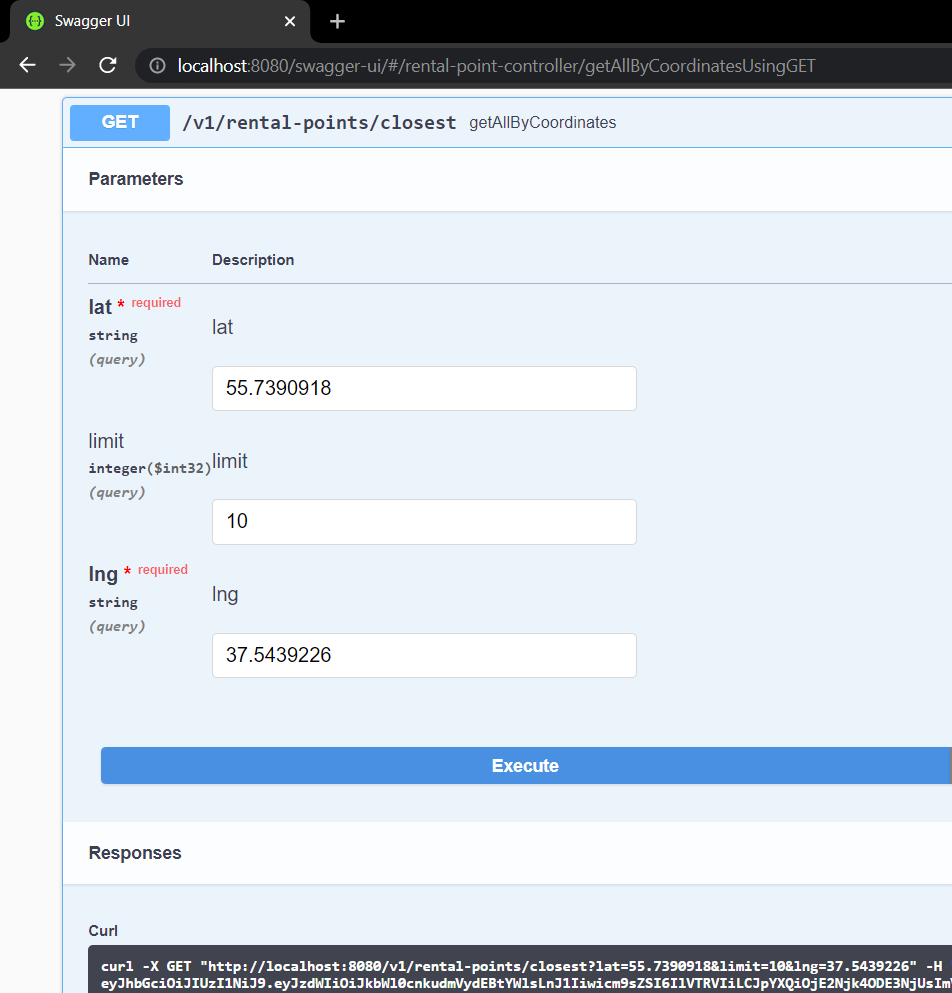


Рисунок 24 – Заполнение запроса для получения ближайших точек проката

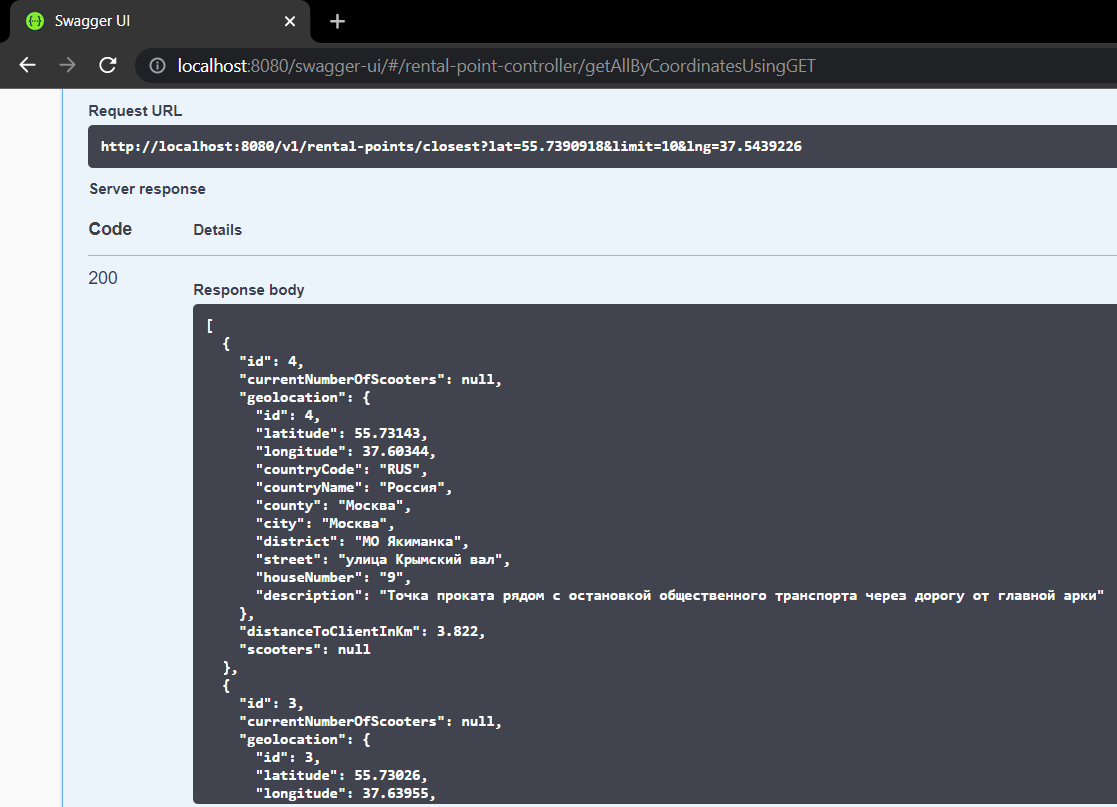


Рисунок 25 – До ближайшей точки проката 3.8 километра

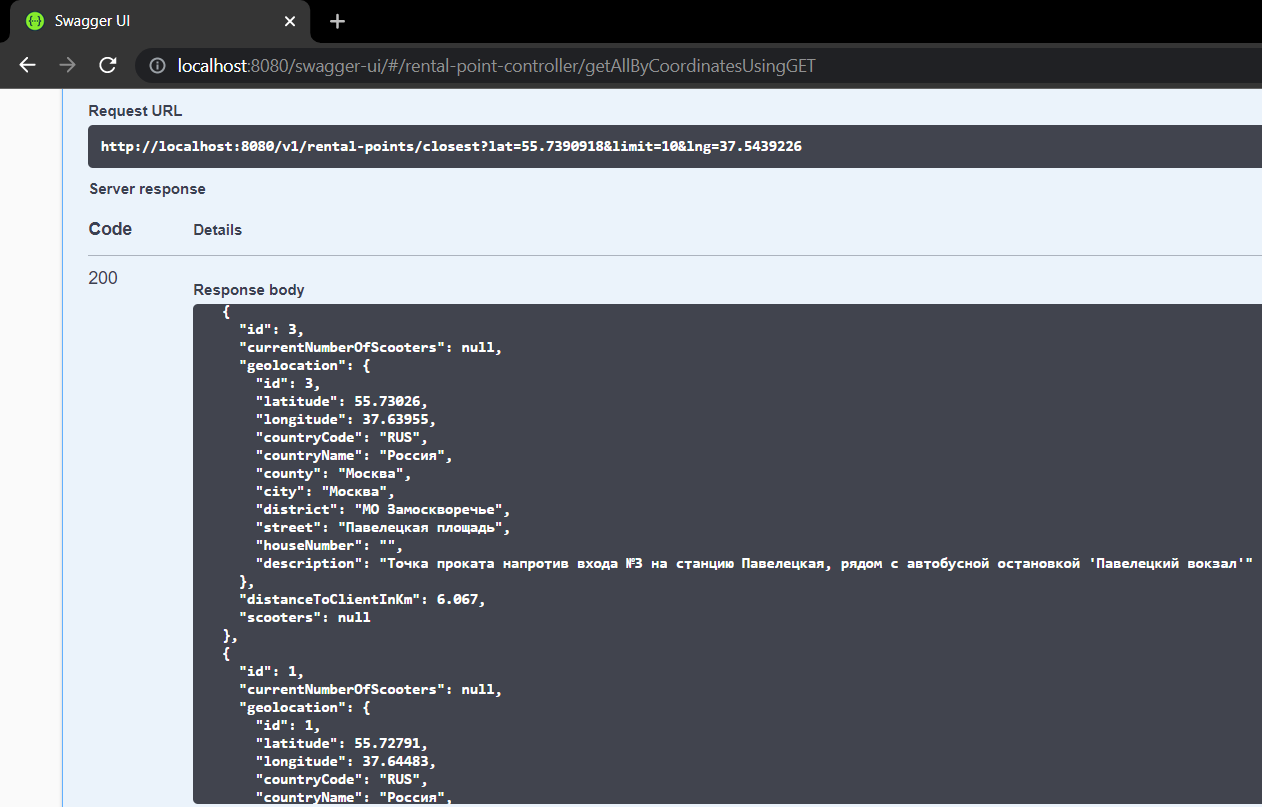


Рисунок 26 – До второй по удалённости точки проката 6 километров

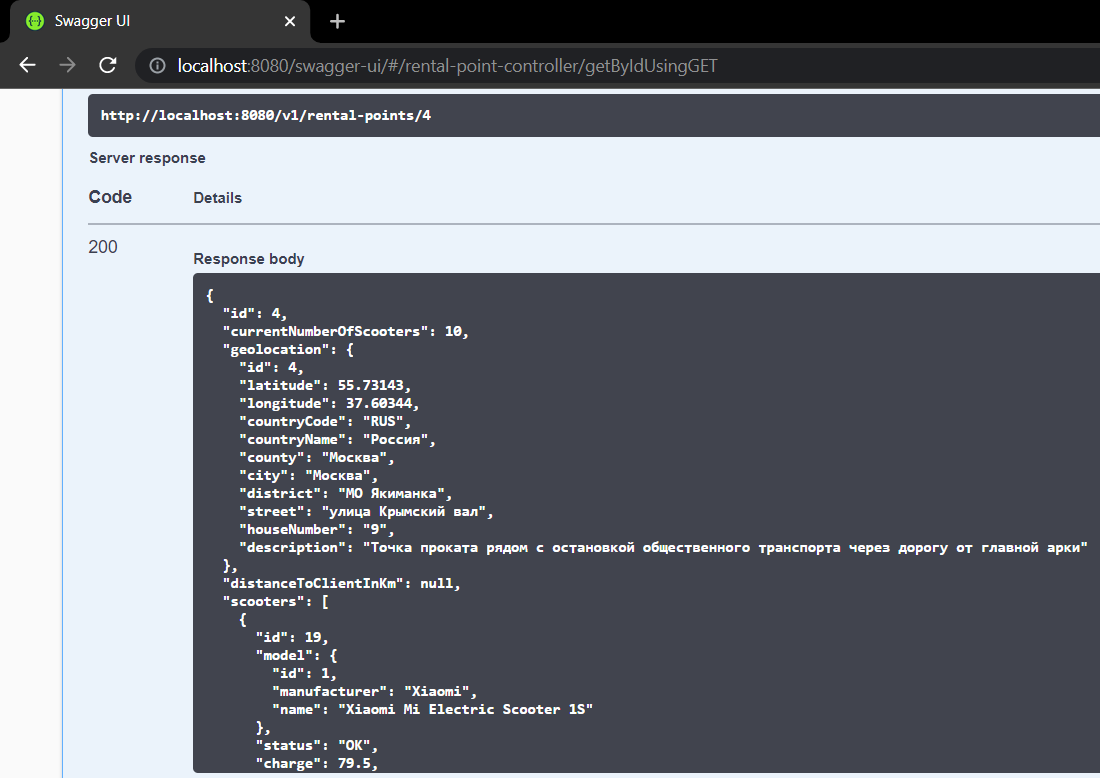


Рисунок 27 – Получение детальной информации о конкретной точке проката

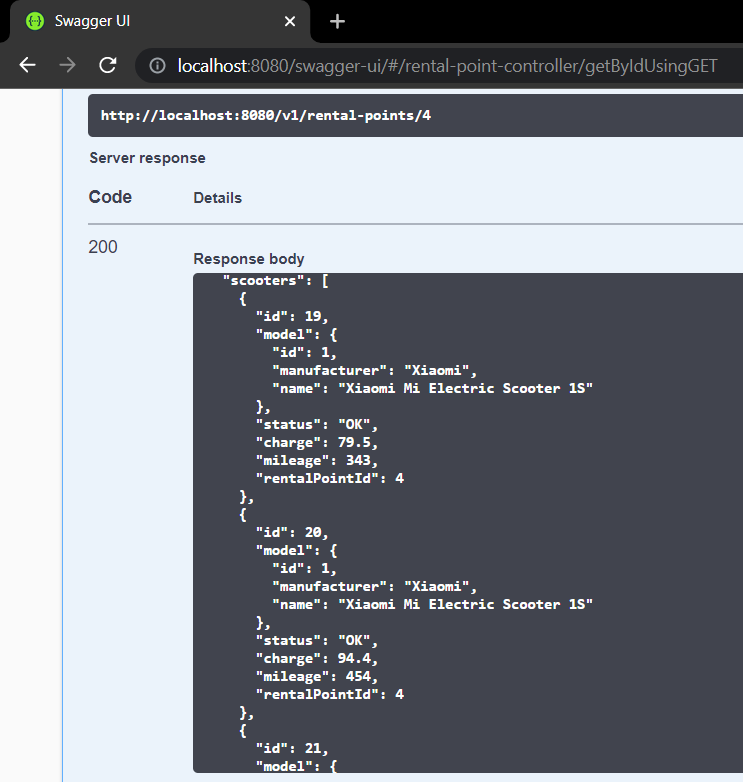


Рисунок 28 – Отрывок списка имеющихся самокатов на конкретной точке

Последним рассмотренным пунктом будет создание поездки, её подтверждение, завершение и последующий просмотр истории поездок пользователя.

На рисунках 29-35 представлен весь вышеописанный процесс. Создание поездки через подписку аналогично тарифу (за исключением того, что подписку нужно сначала установить), а потому вся эта опция была опущена в виду избыточности.

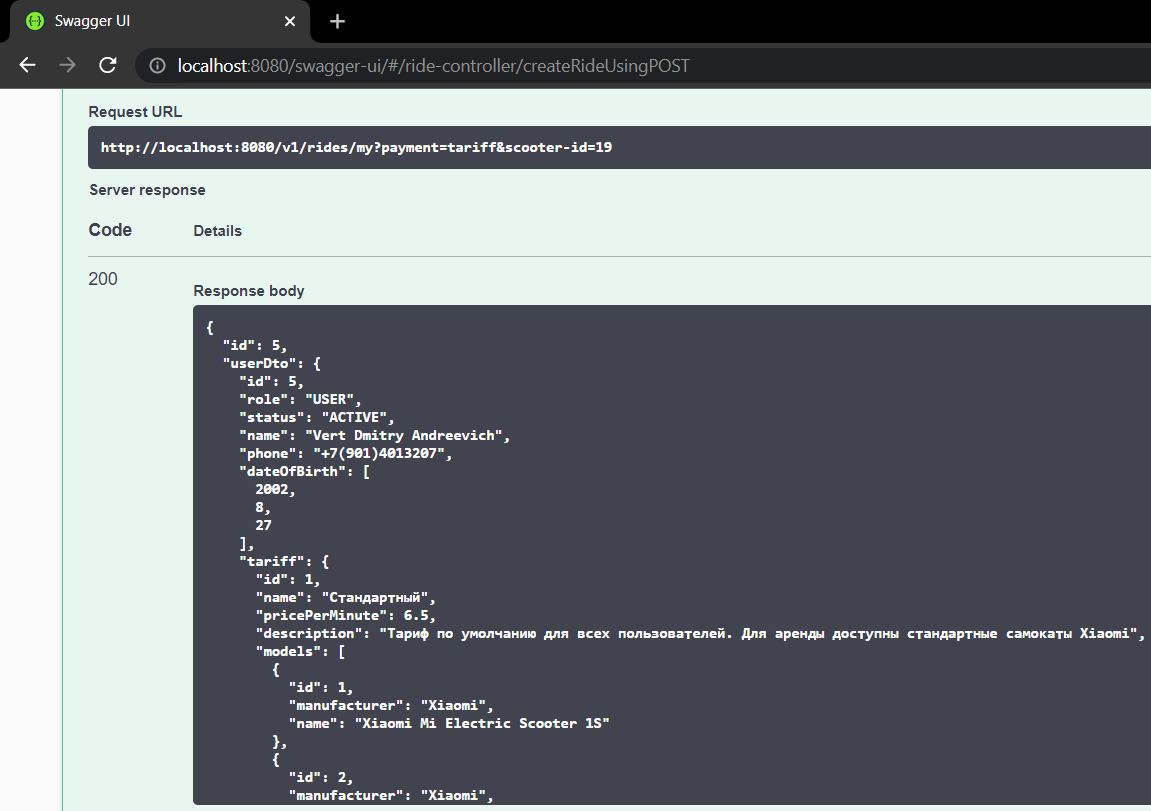


Рисунок 29 – Создание поездки с конкретным самокатом по стандартному тарифу

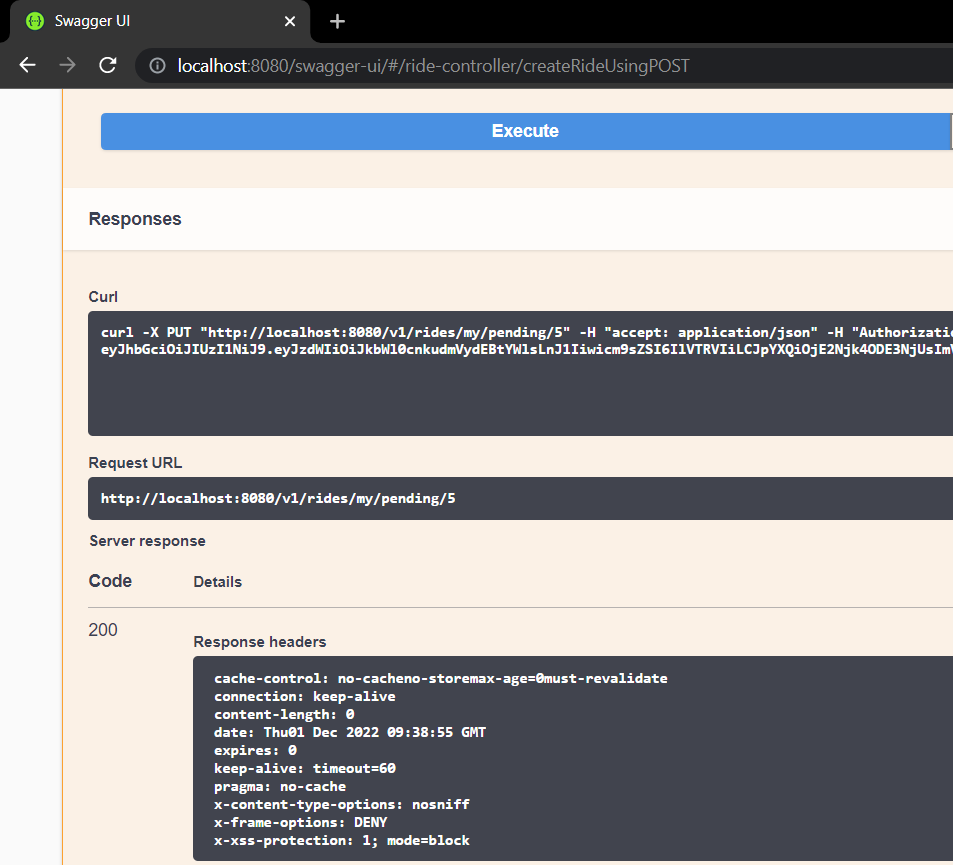


Рисунок 30 – Подтверждение созданной поездки

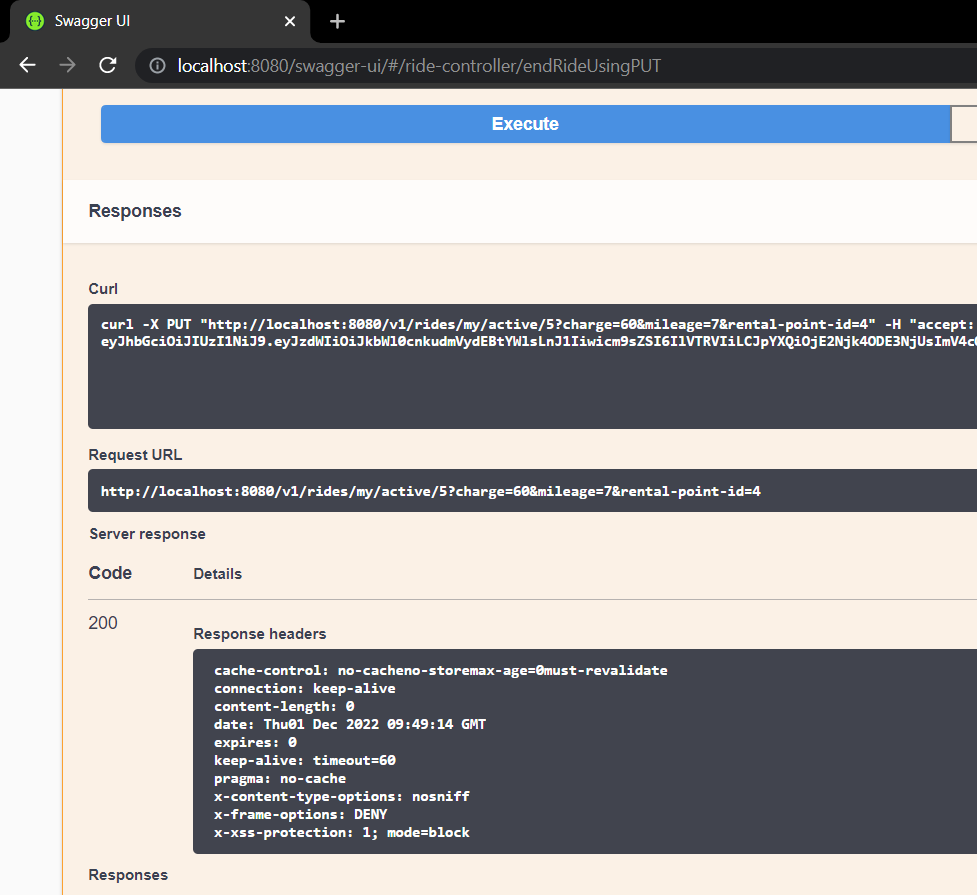


Рисунок 31 – Завершение активной поездки



Рисунок 32 – Получение истории поездок со всей детальной информацией

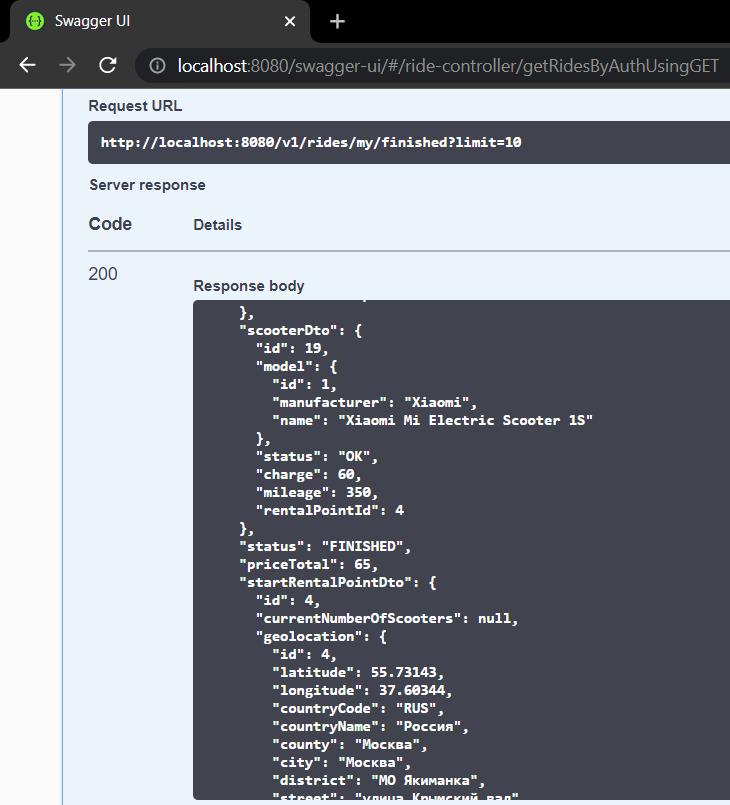


Рисунок 33 – Информация о состоянии самоката после поездки, статусе поездки, итоговой цене, сведения о начальной точке проката

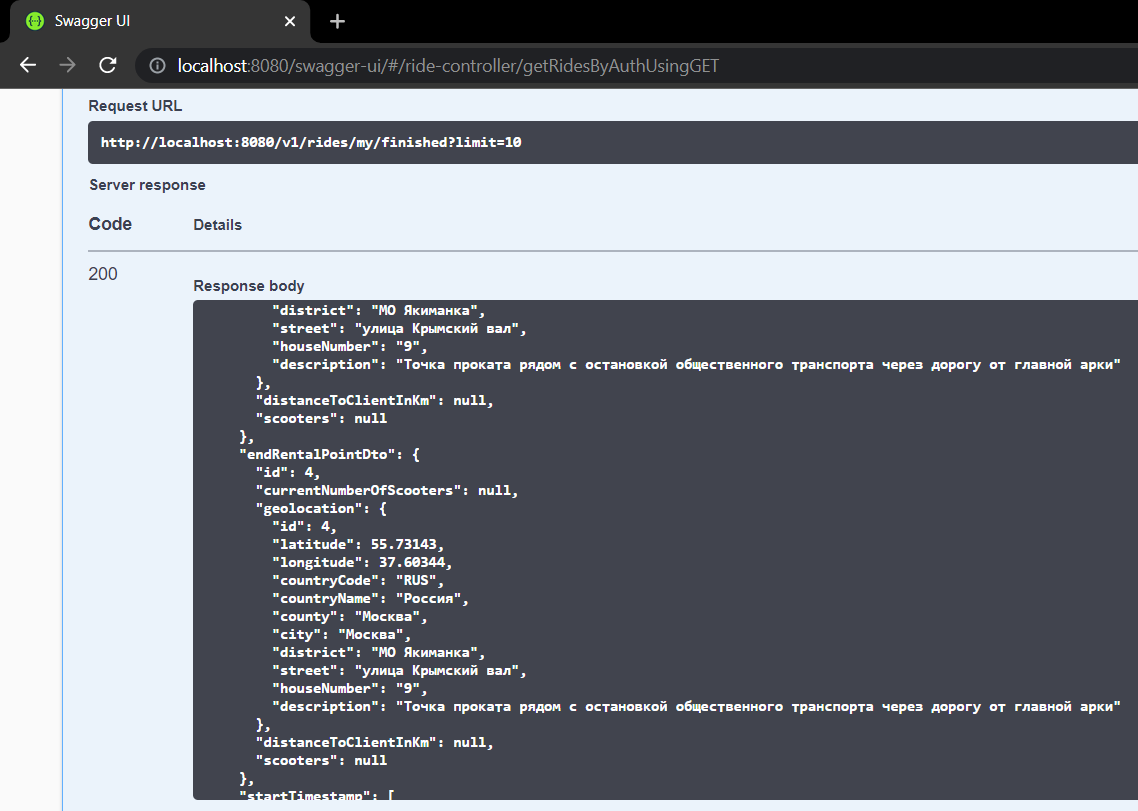


Рисунок 34 – Данные о конечной точке проката, в которой поездка была завершена

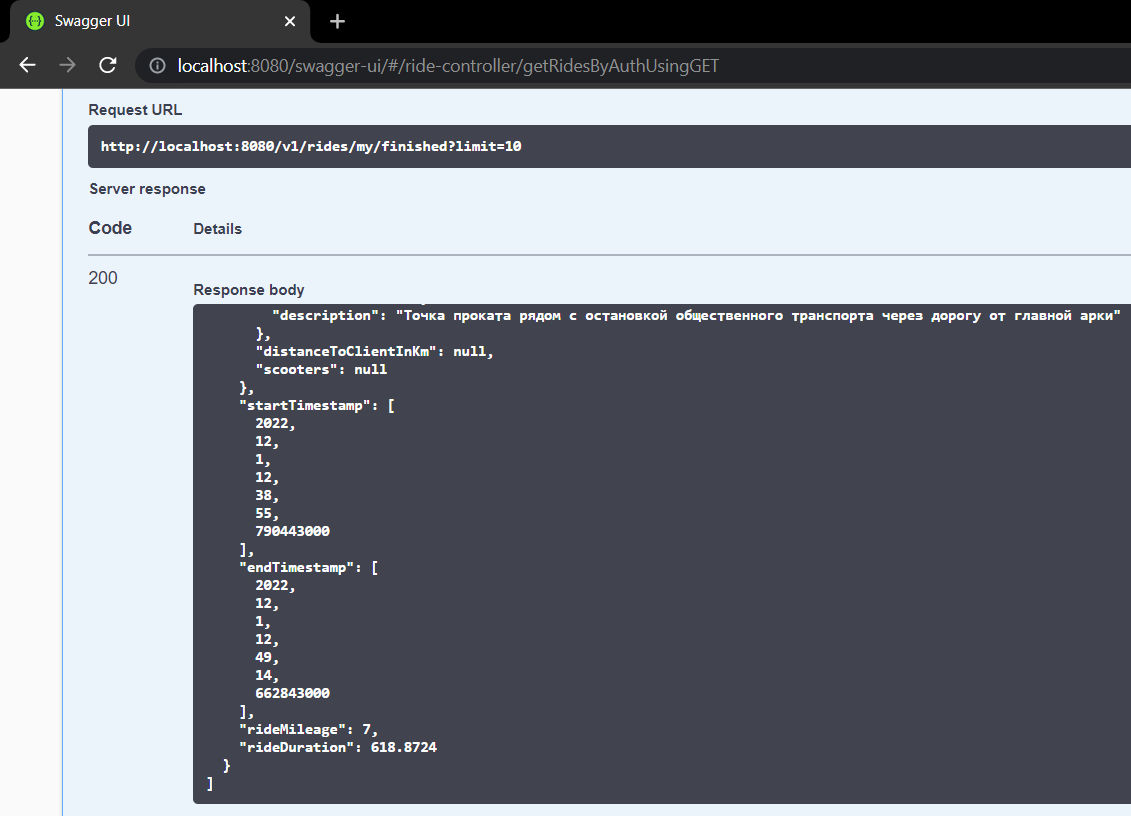
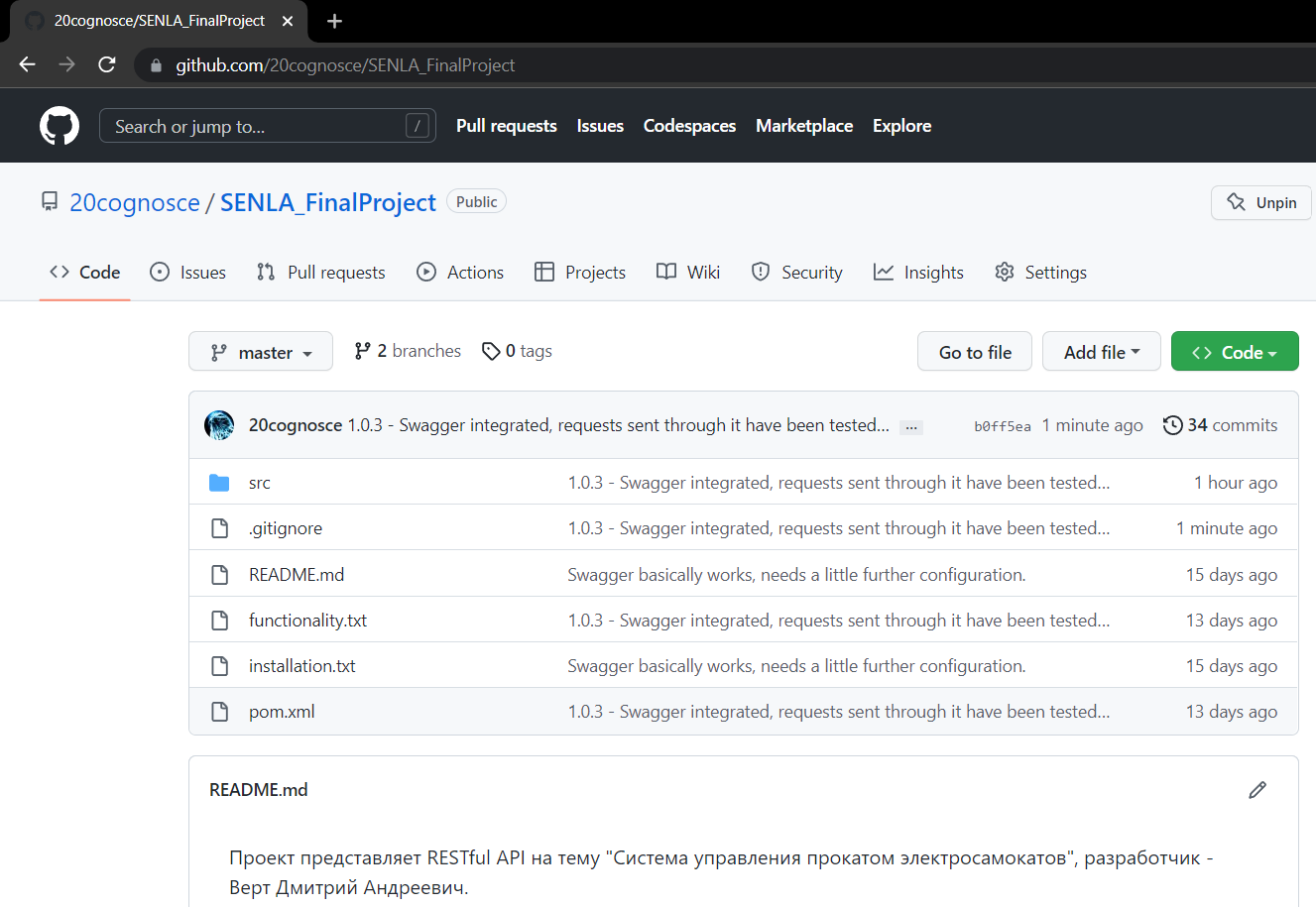


Рисунок 35 – Данные о времени начала и конца поездки, пройденном расстоянии, длительности поездки в секундах

1. Подготовка к защите курсовой работы

Исходный код разработанного в рамках курсовой работы приложения загружен в сеть Интернет (рисунок 36) и доступен по публичной ссылке (указана в приложении А).



**Рисунок 36 – Репозиторий проекта**

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана серверная часть web-приложения «Система управления прокатом электросамокатов», с применением стека технологий, основанных на Java фреймворке Spring.

Также в ходе создания приложения были применены инструменты для взаимодействия программы с базой данных в PostgreSQL, проведена работа с безопасностью приложения с помощью Spring Security и JWT, запрограммирован слой бизнес-логики с обширным функционалом, реализована архитектура MVC, составлена документация к проекту с помощью Swagger, а  благодаря JUnit и Mockito написаны и исполнены модульные тесты.

Был проведен анализ предметной области разработанного web-приложения. Итоговый проект полностью работоспособен, соответствует заявленной тематике и реализует заявленный на этапе анализа функционал по работе системы управления прокатом электросамокатов. Таким образом, разработанный программный продукт выполняет свою основную функцию работоспособного учебного прототипа, предназначенного для дальнейшего практического изучения принципов и подходов в системном дизайне распределенных приложений [20].

В итоге были освоены компетенции по проектированию программного интерфейса с нуля до полноценной информационной системы, реализованной в трёхслойной архитектуре; проведению интеграции между сторонними API и иными программными модулями; анализу функциональных требований и переносу бизнес-задач с естественного языка в исполняющий их программный код; документированию и презентации разработанного программного продукта.

Все цели и задачи, поставленные в листе задания на курсовую работу, были выполнены.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. How to Implement MVC Architecture in Java? [Электронный ресурс]. – URL: https://www.edureka.co/blog/mvc-architecture-in-java (дата обращения 02.11.2022).
2. Тузовский, А. Ф. Проектирование и разработка web-приложений : учебное пособие для вузов / А. Ф. Тузовский. — Москва : Издательство Юрайт, 2021. — 218 с. — (Высшее образование). — ISBN 978-5-534-00515-8. — Текст : электронный // Образовательная платформа Юрайт [Cайт]. — URL: https://urait.ru/bcode/469982 (дата обращения: 01.12.2022).
3. JDK 11 – OpenJDK [Электронный ресурс]. – URL: https://openjdk.org/projects/jdk/11 (дата обращения 03.11.2022).
4. Level up your Java code and explore what Spring can do for you. [Электронный ресурс]. – URL: https://spring.io (дата обращения 03.11.2022).
5. Статья «Mockito и как его готовить» [Электронный ресурс]. – URL: https://habr.com/ru/post/444982 (дата обращения 04.11.2022).
6. OpenAPI Specification [Электронный ресурс]. – URL: https://swagger.io/specification (дата обращения 25.11.2022).
7. Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. — СПб. : Питер, 2021. — 352 c.
8. Раджпут Д. Spring. Все паттерны проектирования. – СПб.: Питер, 2019.
9. Меджуи М., Уайлд Э., Митра Р., Амундсен М. Непрерывное развитие API. Правильные решения в изменчивом технологическом ландшафте. – СПб.: Питер, 2020.
10. Учебник по языку SQL (DDL, DML) на примере диалекта MS SQL Server. [Электронный ресурс]. – URL: https://habr.com/ru/post/255523 (дата обращения 14.11.2022).
11. pgAdmin 4 – PostgreSQL Tools Docs [Электронный ресурс]. – URL: https://www.pgadmin.org/docs/pgadmin4/6.12/index.html (дата обращения 11.11.2022).
12. Hibernate Getting Started Guide [Электронный ресурс]. – URL: https://docs.jboss.org/hibernate/orm/6.1/quickstart/html\_single (дата обращения 15.11.2022).
13. Хоффман Эндрю Х85 Безопасность веб-приложений. — СПб.: Питер, 2021. — 336 с.: ил. — (Серия «Бестселлеры O’Reilly»).
14. Introduction to Java Config for Spring Security [Электронный ресурс]. – URL: https://www.baeldung.com/java-config-spring-security (дата обращения 13.11.2022).
15. JSON Web Token (JWT) — это открытый стандарт (RFC 7519) для создания токенов доступа, основанный на формате JSON. [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/JSON\_Web\_Token (дата обращения 16.11.2022).
16. Канер Сэм, Фолк Джек, Нгуен Енг Кек. Тестирование программного обеспечения. – Киев : Издательство ДиаСофт, 2001. – ISBN 966-7393-87-9. — Текст : электронный // Образовательная платформа ФКТиПМ [Электронный ресурс]. — URL: https://fktpm.ru/file/224-sem-kaner-dzek-folk-eng-kek-nguen-testirovanie-programmnogo-obespeceniya-fundamentalnye-koncepcii-menedzmenta-biznes-prilozenii.pdf (дата обращения: 18.11.2022).
17. Testing in Spring Boot [Электронный ресурс]. – URL: https://www.baeldung.com/spring-boot-testing (дата обращения 18.11.2022).
18. Эспозито Дино. Разработка современных веб-приложений: анализ предметных областей и технологий. – Издательство Вильямс, 2017.
19. Сэмми Пьюривал. Основы разработки веб-приложений. – СПб.: Питер, 2015. – ISBN: 978-5-496-01226-3.
20. Как проходят архитектурные секции собеседования в Яндексе: практика дизайна распределённых систем. [Статья]. – URL: https://habr.com/ru/company/yandex/blog/564132 (дата обращения 25.11.2022).

ПРИЛОЖЕНИЕ А

Исходный код веб-приложения доступен по ссылке – https://github.com/20cognosce/SENLA\_FinalProject.

*Листинг 1 – Отрывок файла dml-script.sql*

insert into geolocation(id, latitude, longitude,  
 country\_code, country\_name, county,  
 city, district, street, house\_number,  
 description)  
VALUES (1, 55.72791, 37.64483, 'RUS', 'Россия', 'Москва', 'Москва', 'МО Замоскворечье', 'Летниковская улица', '4c5',  
 'Точка проката напротив магазина EUROSTAR Express'),  
 (2, 55.72284, 37.64345, 'RUS', 'Россия', 'Москва', 'Москва', 'МО Замоскворечье', 'Летниковская улица',  
 '20 стр 2', 'Точка проката внутри первого гаража у дороги'),  
 (3, 55.73026, 37.63955, 'RUS', 'Россия', 'Москва', 'Москва', 'МО Замоскворечье', 'Павелецкая площадь',  
 '', 'Точка проката напротив входа №3 на станцию Павелецкая, рядом с автобусной остановкой ''Павелецкий вокзал'''),  
 (4, 55.73143, 37.60344, 'RUS', 'Россия', 'Москва', 'Москва', 'МО Якиманка', 'улица Крымский вал', '9',  
 'Точка проката рядом с остановкой общественного транспорта через дорогу от главной арки'),  
  
 (5, 52.96346, 36.06547, 'RUS', 'Россия', 'Орловская область', 'Орёл', 'Заводской район', 'Александровский мост',  
 '', 'Точка проката в начале Ленинского моста со стороны сквера Ермолова'),  
 (6, 52.96558, 36.06844, 'RUS', 'Россия', 'Орловская область', 'Орёл', 'Советский район', 'Богоявленская площадь',  
 '', 'Точка проката в начале Подвесного моста со стороны Детского парка, рядом с лавкой с мороженым'),  
 (7, 53.27989, 36.57833, 'RUS', 'Россия', 'Орловская область', 'Мценск', 'Мценск', 'улица Дзержинского',  
 '', 'Точка проката рядом с памятником И.С. Тургеневу'),  
  
 (8, 53.90675, 27.57287, 'BLR', 'Республика Беларусь', 'Минск', 'Минск', '', 'улица Фрунзе',  
 '2 корп 9', 'Точка проката рядом со входной аркой в Центральный детский парк'),  
 (9, 53.90235, 27.57297, 'BLR', 'Республика Беларусь', 'Минск', 'Минск', '', 'Первомайская улица',  
 '3А', 'Точка проката напротив кафе Family Club');  
  
insert into rental\_point(id, geolocation\_id)  
VALUES (1, 1),  
 (2, 2),  
 (3, 3),  
 (4, 4),  
 (5, 5),  
 (6, 6),  
 (7, 7),  
 (8, 8),  
 (9, 9);

*Продолжение листинга 1*

insert into tariff(id, name, price\_per\_minute, description)  
VALUES (1, 'Стандартный', 5,  
 'Тариф по умолчанию для всех пользователей. Для аренды доступны стандартные самокаты Xiaomi'),  
 (2, 'Молниеносный', 7.5,  
 'Тариф для тех кто любит скорость. В аренду можно брать мощные самокаты Ninebot Max');  
  
insert into subscription(id, name, price, duration\_in\_days, description)

VALUES (1, 'Стандарт', 499, 30, 'Подписка на безлимитное использование стандартных самокатов за 499 рублей в месяц.'),  
 (2, 'Молния', 749, 30, 'Молниеносная подписка на безлимитное использование мощных самокатов за 749 рублей в месяц.'),  
 (3, 'Вечная', 19990, 36525, 'Вечная подписка без ограничений только для уникальных пользователей');  
  
insert into scooter\_model(id, manufacturer, model, scooter\_weight, max\_weight\_limit, max\_speed, max\_range, price)  
VALUES (1, 'Xiaomi', 'Xiaomi Mi Electric Scooter 1S', 12.5, 100, 25, 45, 34490),  
 (2, 'Xiaomi', 'Mijia M365 Pro', 14.2, 100, 25, 45, 37990),  
  
 (3, 'Ninebot', 'KickScooter MAX 30', 19.1, 100, 30, 65, 44990),  
 (4, 'Ninebot', 'KickScooter MAX G30P', 19.2, 100, 30, 60, 54990);

*Листинг 2 – RentalPointDaoImpl.java*

import com.senla.dao.RentalPointDao;  
import com.senla.domain.model.entity.RentalPoint;  
import org.hibernate.query.criteria.internal.OrderImpl;  
import org.springframework.stereotype.Repository;  
  
import javax.persistence.TypedQuery;  
import javax.persistence.criteria.CriteriaBuilder;  
import javax.persistence.criteria.CriteriaQuery;  
import javax.persistence.criteria.Expression;  
import javax.persistence.criteria.JoinType;  
import javax.persistence.criteria.ParameterExpression;  
import javax.persistence.criteria.Root;  
import java.util.List;  
import java.util.Optional;  
  
@Repository  
public class RentalPointDaoImpl extends AbstractDaoImpl<RentalPoint> implements RentalPointDao {  
  
 @Override  
 protected Class<RentalPoint> daoEntityClass() {  
 return RentalPoint.class;  
 }  
  
 @Override  
 public List<RentalPoint> getAllTheClosest(double latitude, double longitude, int limit) {  
 CriteriaBuilder cb = entityManager.getCriteriaBuilder();  
 CriteriaQuery<RentalPoint> cq = cb.createQuery(RentalPoint.class);  
 Root<RentalPoint> root = cq.from(RentalPoint.class);

*Продолжение листинга 2*

ParameterExpression<Double> doubleParam1 = cb.parameter(Double.class);  
 ParameterExpression<Double> doubleParam2 = cb.parameter(Double.class);  
 Expression<Double> func =  
 cb.function("calculate\_distance\_to\_point", Double.class, root.get("id"), doubleParam1, doubleParam2);  
  
 TypedQuery<RentalPoint> q = entityManager.createQuery(cq  
 .select(root)  
 .orderBy(new OrderImpl(func, true))

);  
 q.setParameter(doubleParam1, latitude);  
 q.setParameter(doubleParam2, longitude);  
 q.setMaxResults(limit);  
  
 return q.getResultList();  
 }  
  
 @Override  
 public Optional<RentalPoint> getByIdWithScooters(long id) {  
 CriteriaBuilder cb = entityManager.getCriteriaBuilder();  
 CriteriaQuery<RentalPoint> cq = cb.createQuery(RentalPoint.class);  
 Root<RentalPoint> root = cq.from(RentalPoint.class);  
  
 root.fetch("scooters", JoinType.LEFT);  
  
 TypedQuery<RentalPoint> q = entityManager.createQuery(cq  
 .select(root)  
 .where(cb.equal(root.get("id"), id)));  
  
 RentalPoint rentalPoint = q.getSingleResult();  
 return Optional.of(rentalPoint);  
 }  
}

*Листинг 3 – Отрывок класса RideServiceImpl.java*

@Slf4j  
@RequiredArgsConstructor  
@Service  
public class RideServiceImpl extends AbstractServiceImpl<Ride, RideDao> implements RideService {  
  
 private final ScheduledExecutorService scheduler = Executors.newScheduledThreadPool(1);  
 private final TariffService tariffService;  
 private final RideDao rideDao;  
  
 ...  
  
 @Transactional  
 @Override  
 public Ride createRideWithTariff(Scooter scooter, User user) {  
 Tariff tariff = user.getTariff();  
 Double pricePerMinute = tariffService.calculateRidePricePerMinute(scooter.getRentalPoint(), tariff, scooter);

*Продолжение листинга 3*

List<Long> models = tariff.getModels().stream().map(ScooterModel::getId).collect(toList());  
 if (!models.contains(scooter.getModel().getId())) {  
 throw new IllegalArgumentException("Выбранный самокат не поддерживается вашим тарифом");  
 }  
 if (scooter.getStatus() == ScooterConditionStatus.UNAVAILABLE || scooter.getStatus() == ScooterConditionStatus.IN\_USE) {  
 throw new IllegalArgumentException("Выбранный самокат не доступен");  
 }

Ride ride = Ride.builder()  
 .user(user)  
 .scooter(scooter)  
 .creationTimestamp(LocalDateTime.now())  
 .status(RideStatus.PENDING)  
 .startRentalPoint(scooter.getRentalPoint())  
 .pricePerMinute(pricePerMinute)  
 .build();  
  
 getDefaultDao().create(ride);  
 startCountdownForDeletionOfPendingRidesOfTheUser(Duration.ofSeconds(30), user, 30);  
 return ride;  
 }  
  
 ...  
  
 @Transactional  
 @Override  
 public void startRide(Ride ride) {  
 if (ride.getScooter().getStatus() == ScooterConditionStatus.UNAVAILABLE  
 || ride.getScooter().getStatus() == ScooterConditionStatus.IN\_USE) {  
 throw new IllegalArgumentException("Выбранный самокат не доступен");  
 }  
 ride.setStartTimestamp(LocalDateTime.now());  
 ride.setStatus(RideStatus.ACTIVE);  
 ride.getScooter().setStatus(ScooterConditionStatus.IN\_USE);  
 ride.getScooter().setRentalPoint(null);  
 getDefaultDao().update(ride);  
 // if the ride has been deleted after getting it in controller,  
 // entity manager will throw exception on updating deleted entity  
 }  
  
 @Transactional  
 @Override  
 public void endRide(Ride ride, RentalPoint rentalPoint, Double mileage, Double charge) {  
 //assuming mileage is fair  
 Scooter scooter = ride.getScooter();  
 LocalDateTime startTime = ride.getStartTimestamp();  
 LocalDateTime endTime = LocalDateTime.now();  
 ride.setEndTimestamp(endTime);  
 ride.setEndRentalPoint(rentalPoint);  
 ride.setRideMileage(mileage);

*Продолжение листинга 3*

ride.setPriceTotal(ride.getPricePerMinute() \* Duration.between(startTime, endTime).toMinutes());

ride.setStatus(RideStatus.FINISHED);  
  
 scooter.setMileage(scooter.getMileage() + mileage);  
 scooter.setCharge(charge);  
 scooter.setRentalPoint(rentalPoint);  
 if (charge <= 1) {  
 scooter.setStatus(ScooterConditionStatus.DISCHARGED);  
 } else {  
 scooter.setStatus(ScooterConditionStatus.OK);  
 }

getDefaultDao().update(ride);  
 }  
  
 @Override  
 public void startCountdownForDeletionOfPendingRidesOfTheUser(Duration minPendingRideLifetime, User user,  
 long delayInSeconds) {  
 scheduler.schedule(() -> {  
 try {  
 getDefaultDao().deletePendingRidesOfTheUser(minPendingRideLifetime, user);  
 } catch (Exception e) {  
 log.error("Не удалось удалить поездки пользователя с истекшим сроком ожидания", e);  
 throw new RuntimeException(e);  
 }  
 }, delayInSeconds, TimeUnit.SECONDS);  
 }  
  
 ...  
}

*Листинг 4 – Отрывок класса JwtTokenSupplier.java*

import io.jsonwebtoken.Claims;  
import io.jsonwebtoken.JwtException;  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureAlgorithm;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.postgresql.shaded.com.ongres.scram.common.bouncycastle.base64.Base64;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.stereotype.Component;  
  
import javax.servlet.http.HttpServletRequest;  
import java.nio.charset.StandardCharsets;  
import java.util.Date;

*Продолжение листинга 4*

@Slf4j  
@RequiredArgsConstructor  
@Component  
public class JwtTokenSupplier {  
  
 private final UserDetailsService userDetailsService;  
 @Value("${jwt.secret.key}")  
 private String secretKey;  
 @Value("${expirationTimeInSeconds}")  
 private Long expirationTimeInSeconds;  
 @Value("${authorization.header}")  
 private String authorizationHeader;  
  
 public String generateToken(String login, String role) {

Date now = new Date();  
 Date valid = new Date(now.getTime() + expirationTimeInSeconds \* 1000);  
  
 Claims claims = Jwts.claims().setSubject(login);  
 claims.put("role", role);  
 return Jwts.builder()  
 .setClaims(claims)  
 .setIssuedAt(now)  
 .setExpiration(valid)  
 .signWith(SignatureAlgorithm.HS256, Base64.toBase64String(secretKey.getBytes(StandardCharsets.UTF\_8)))  
 .compact();  
 }  
  
 public boolean validateToken(String token) {  
 try {  
 return !Jwts.parser().setSigningKey(secretKey.getBytes(StandardCharsets.UTF\_8)).parseClaimsJws(token).getBody().getExpiration().before(new Date());  
 } catch (Exception e) {  
 log.error("JWT token validation failed", e);  
 throw new JwtException("JWT token validation failed", e);  
 }  
 }

...

}

*Листинг 6 – SwaggerConfig.java*

*@Configuration  
public class* SwaggerConfig {  
  
 *@Bean  
 public* Docket api() {  
 *return new* Docket(DocumentationType.SWAGGER\_2)  
 .securityContexts(List.of(securityContext()))  
 .securitySchemes(List.of(apiKey()))  
 .select()  
 .apis(RequestHandlerSelectors.basePackage("com.senla.сontroll

er.restcontroller"))

*Продолжение листинга 6*

.paths(PathSelectors.any())  
 .build();  
 }  
  
 *@Bean  
 public* InternalResourceViewResolver defaultViewResolver() {  
 *return new* InternalResourceViewResolver();  
 }  
  
 *private* ApiKey apiKey() {  
 *return new* ApiKey("JWT", "Authorization", "header");  
 }  
  
 *private* SecurityContext securityContext() {  
 *return* SecurityContext.builder().securityReferences(defaultAuth()).build();  
 }

*private* List<SecurityReference> defaultAuth() {  
 AuthorizationScope authorizationScope = *new* AuthorizationScope("global", "accessEverything");  
 AuthorizationScope[] authorizationScopes = *new* AuthorizationScope[1];  
 authorizationScopes[0] = authorizationScope;  
 *return* List.of(*new* SecurityReference("JWT", authorizationScopes));  
 }  
}