Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное   
образовательное учреждение высшего образования   
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторной работе

Дисциплина: «Технологии сетевого программирования»

Выполнили: Зайнуллова А.М.

Тарасов П.Д.

Группа: 6304-010302D

Самара, 2025

1. **Проектирование приложения**

**Концепция приложения**

Онлайн-магазин одежды, обуви и аксессуаров обладающий удобным пользовательским интерфейсом, позволяющий быстро находить необходимую позицию среди всех имеющихся в ассортименте товаров различных категорий.

**Сценарии использования**

**Роли пользователей приложения:**

* Потребитель
* Администратор

**Сценарии использования приложения:**

Потребитель:

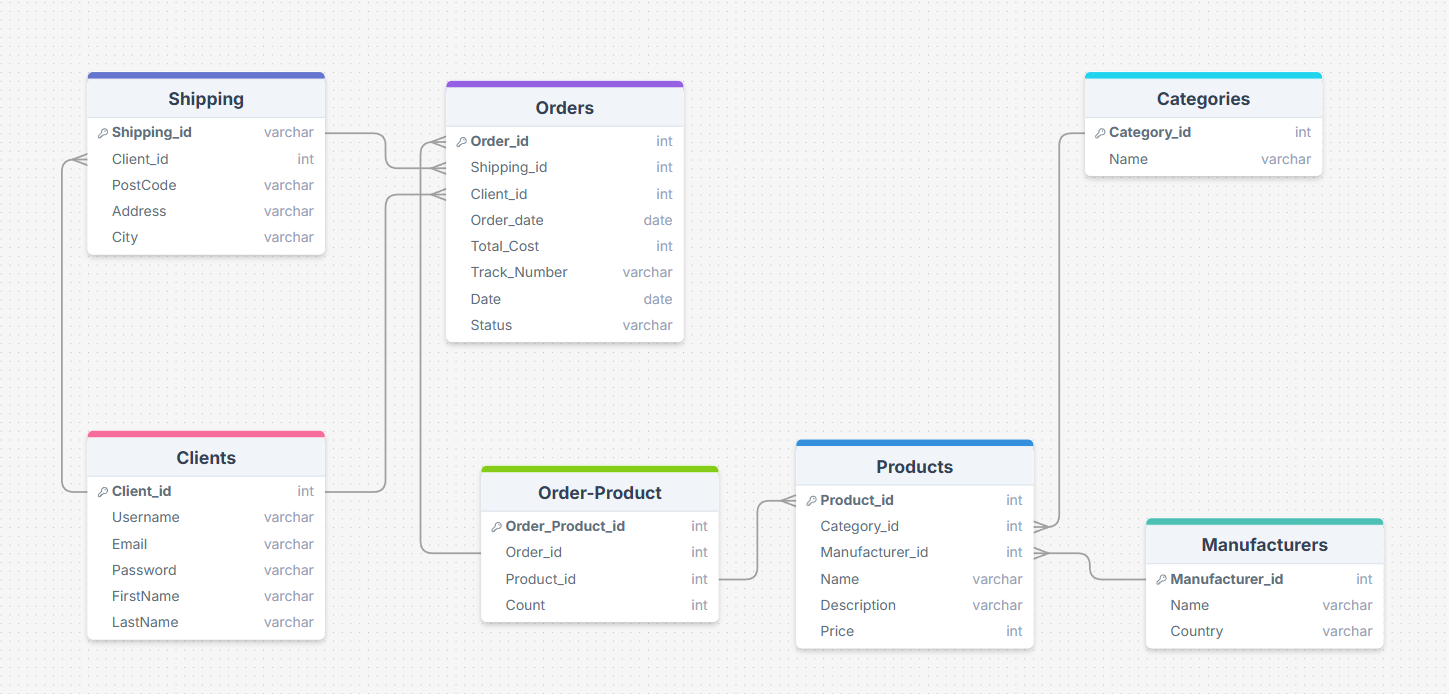
* Регистрация в системе магазина
* Выбор и добавление товара в корзину из имеющегося ассортимента
* Оформление заказа и доставки
* Отслеживание статуса заказа через личный кабинет покупателя

Администратор:

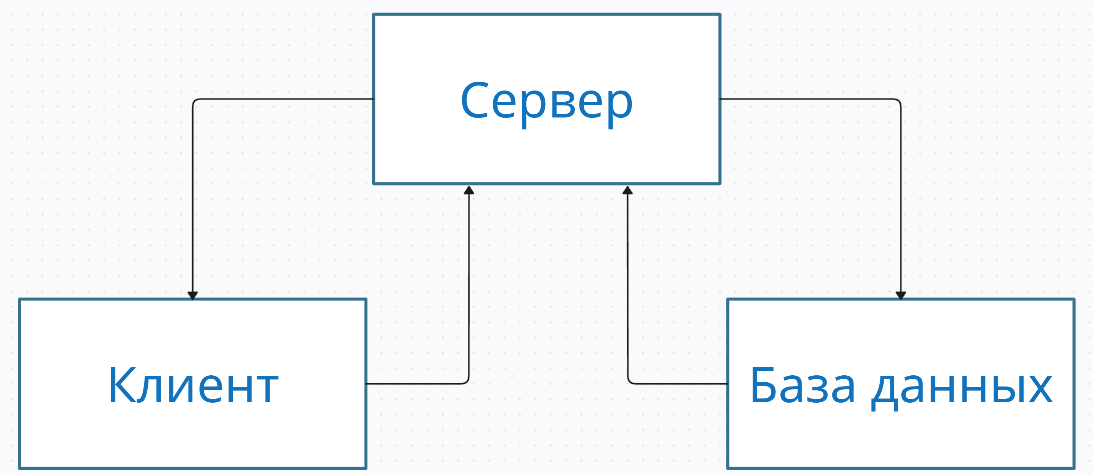
* Изменение наличия, цен, описания товара в магазине
* Обработка поступивших заказов
* Просмотр истории заказов магазина

**Архитектура приложения**

**Схема БД:**

****

**Схема приложения:**

****

**Стек технологий приложения**

**Серверная часть:**

Java – язык программирования для Back-end части приложения был выбран из-за его распространённости и большого выбора подходящих фреймворков.

Spring – фреймворк на котором будет строится большая часть приложения, фреймворк популярен, достаточно прост и функционален для поставленной задачи.

Hibernate – библиотека позволит упростить процесс разработки и настройки взаимодействия между сервером и базой данных.

GitHub – самая распространённая СКВ.

**Web сторона:**

Thymeleaf – фреймворк для упрощения разработки пользовательского интерфейса.

**База данных:**

PostgreSQL – СУБД хорошо подходящая в поставленной задаче.

**API:**

OpenAPI – экосистема разработки HTTP API. Позволяет взаимодействовать с ПО вне зависимости от особенностей разработки.

1. **Разработка базы данных**

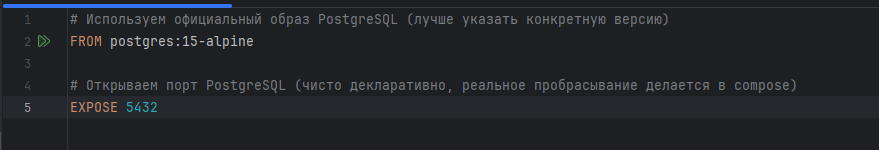
**Настройка базы данных**

* Развертывание базы данных **PostgreSQL** в контейнере **Docker**

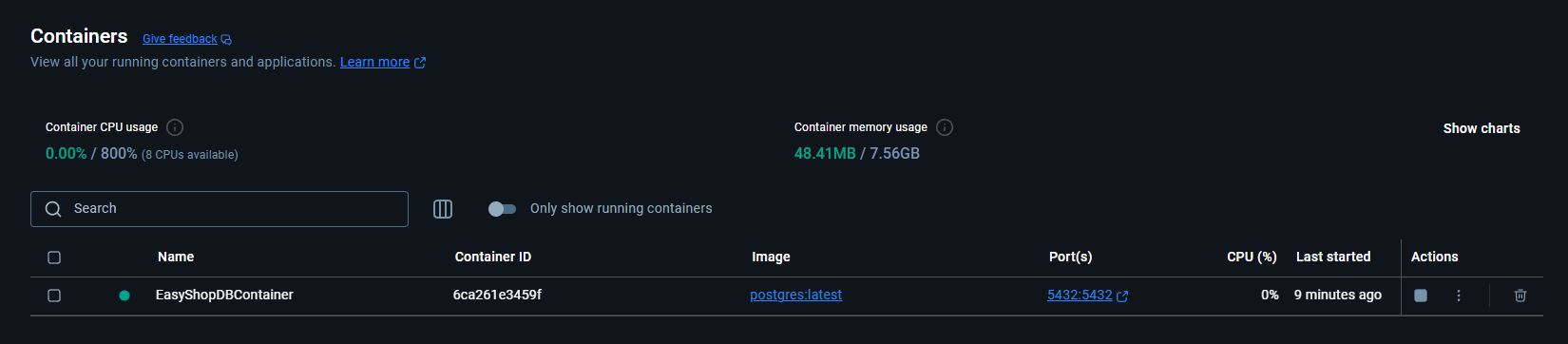
Развертывание производилось с помощью выполнения команды в командной строке операционной системы:

**docker run --name EasyShopDBContainer -e POSTGRES\_DB=EasyShopDocker -e POSTGRES\_USER=postgres -e POSTGRES\_PASSWORD=pa33word -p 5432:5432 -d postgres:latest**

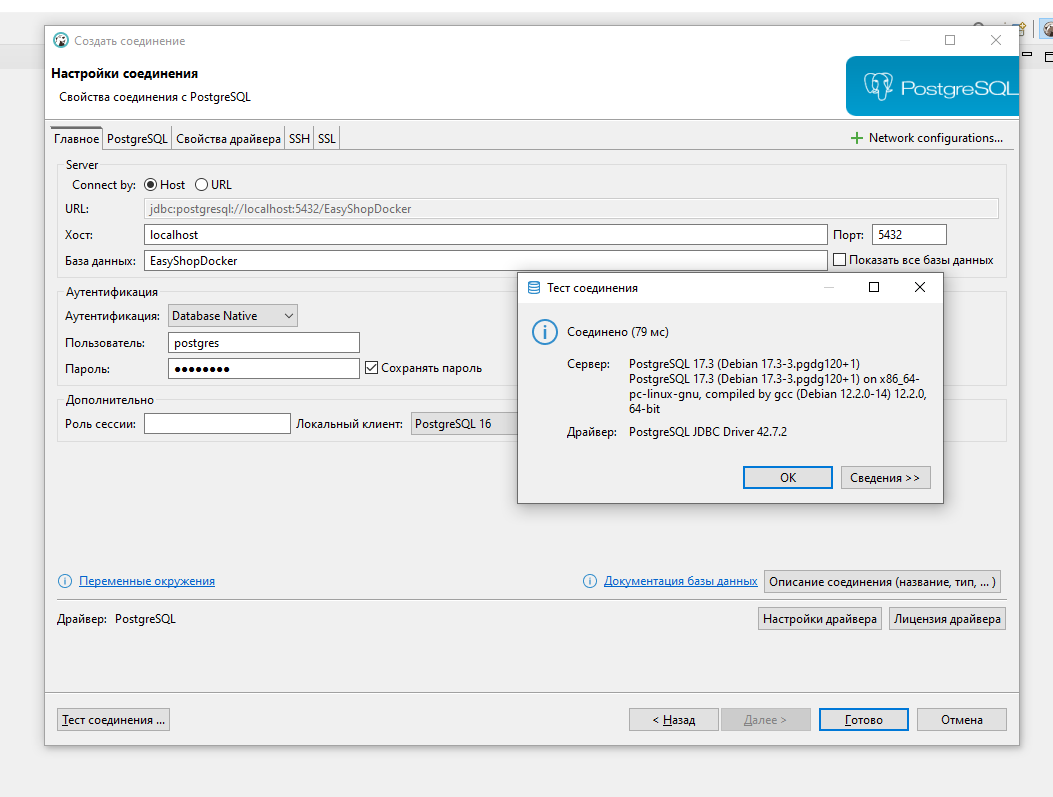
Или с помощью докер файла



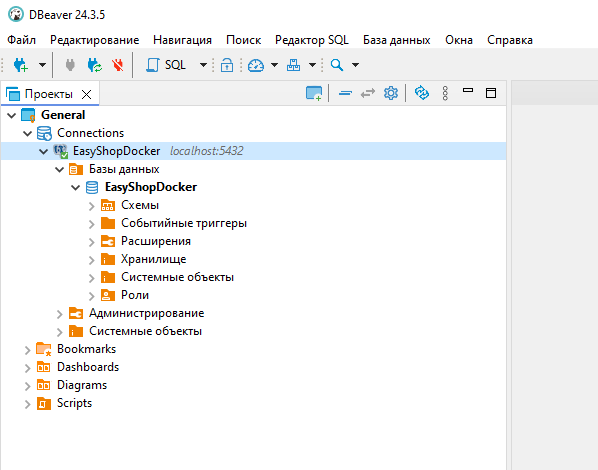
Запущенный контейнер можно наблюдать в приложении **Docker Desktop**

****

* Тестируем подключение к БД с помощью приложения **DBeaver**

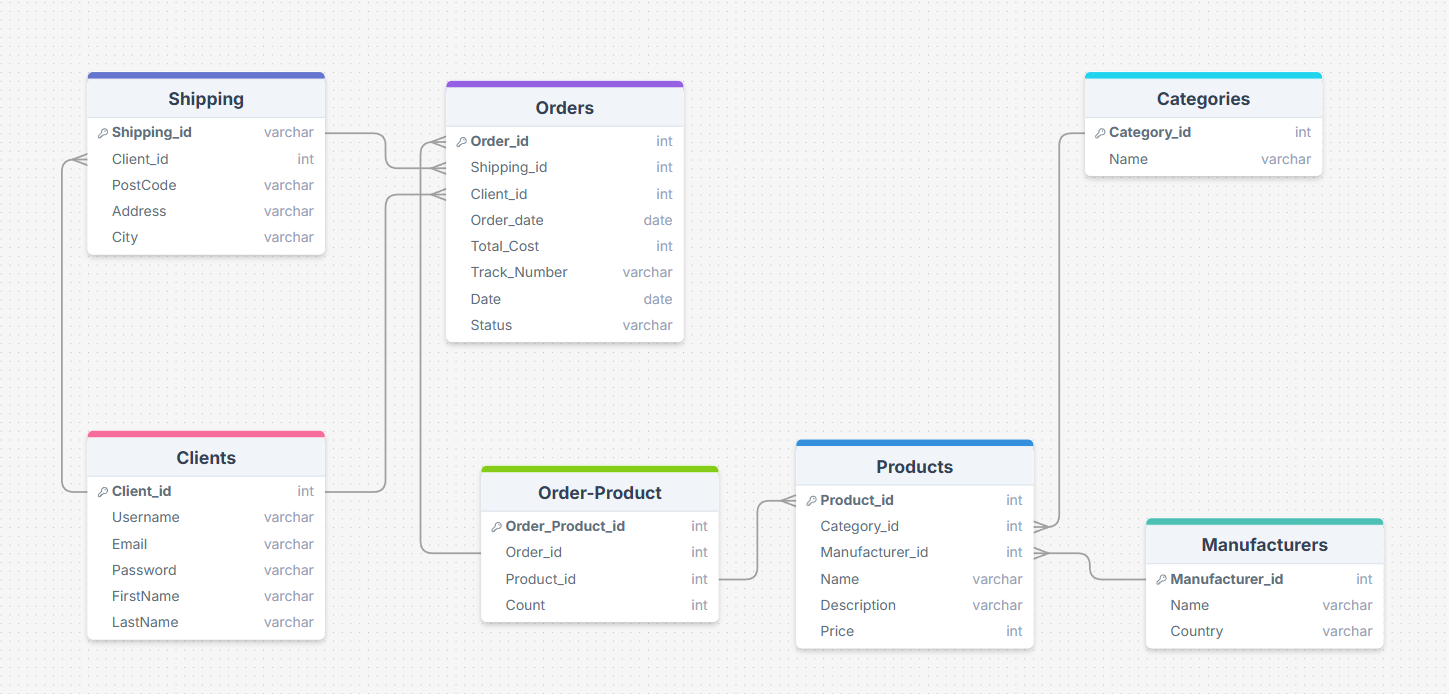
****

* После успешного тестирования можем сохранить подключение в **DBeaver.**

****

**Разработка ORM-модели**

* Структура сущностей и связей с ними имеет вид

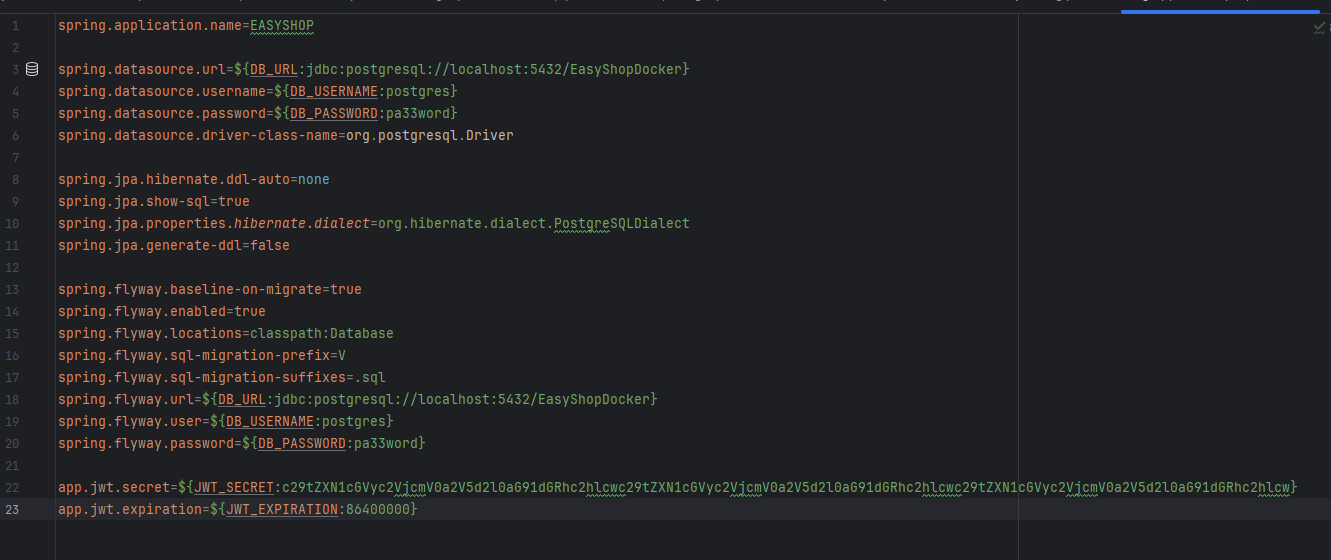


* В работе с ORM-моделью мы будем использовать **Hibernate**, **Spring Data JPA**, для этого используя **Maven**, добавляем зависимости необходимых фреймворков.

****

Добавление отдельной зависимости **Hibernate** не требуется, так как она содержится в зависимости **spring-boot-starter-data-jpa**.

* Настройку **Hibernate** и **Spring Data** произведём с помощью файла конфигурации **Application.properties**.



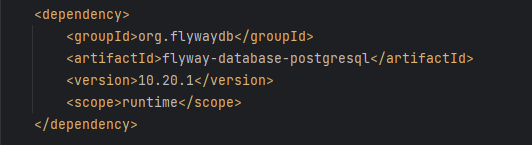
* Разработку сущностей выполняем с помощью **Java** классов и **Hibernate** аннотаций.

Пример класса сущности Категорий

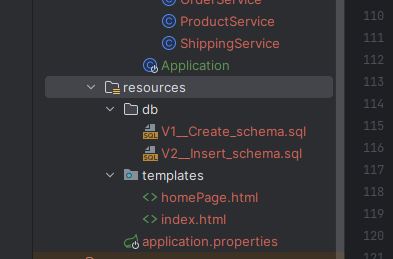


Помимо объявленных полей, каждый класс имеет методы GET и SET для каждого из объявленных полей и пустой конструктор.

* Для миграции базы данных будем использовать **Flyway**, добавляем зависимость в pom файл и определяем конфигурацию **Flyway** в представленном выше файле **Application.properties**.

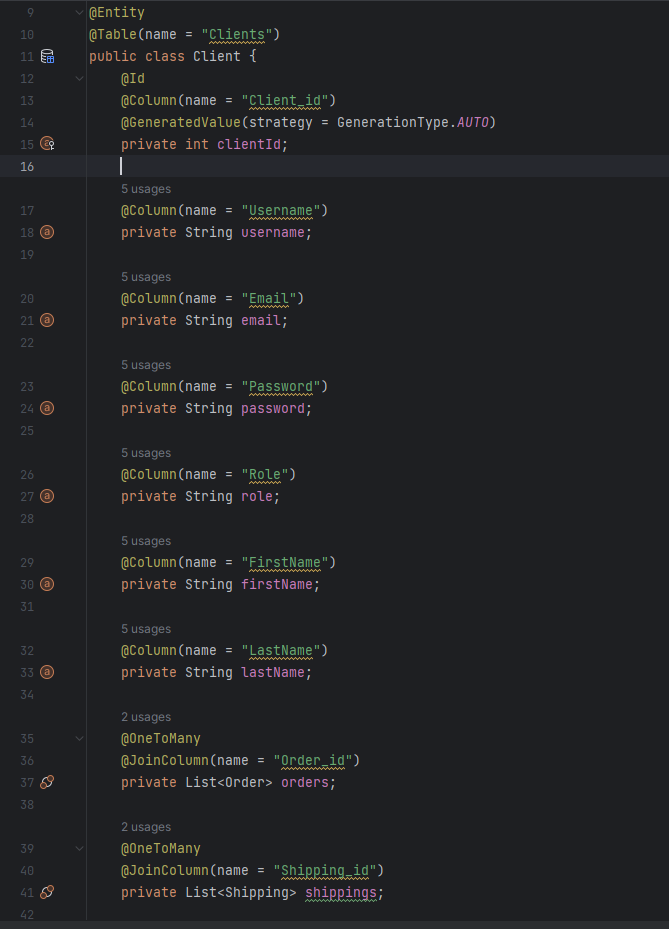


После настройки **Flyway** добавляем файлы бэкапа базы данных в указанную при конфигурации папку.



**Разработка модели пользователя**

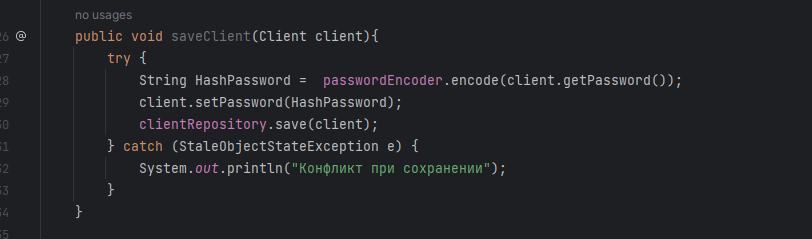
* Сущность пользователя, которая будет использоваться для аутентификации и авторизации пользователей сервиса, имеет вид



* Аутентификация, авторизация и хеширование выполнено с помощью фреймворков **Spring Security** и **Bcrypt**. Конфигурация **Spring Security** выполнена с помощью класса-контроллера вида:

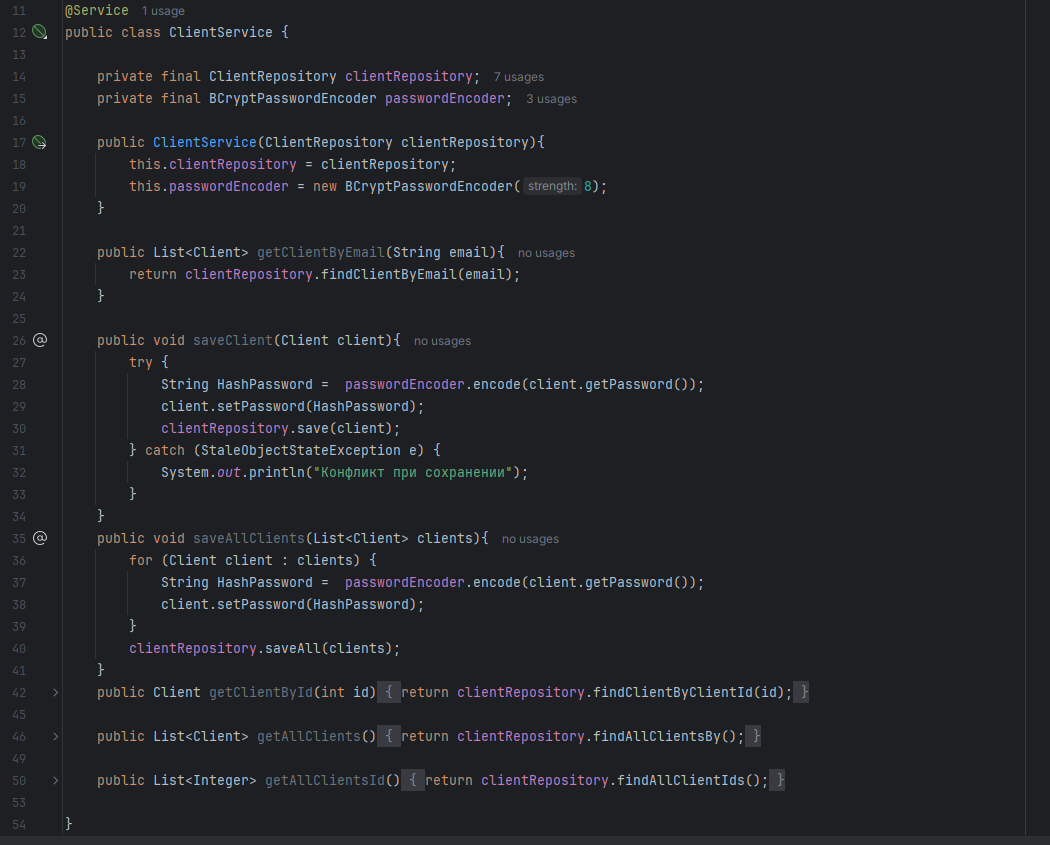


Хеширование паролей происходит в Service слое приложения при сохранении нового пользователя в базе данных.



**Функционал для работы с моделями**

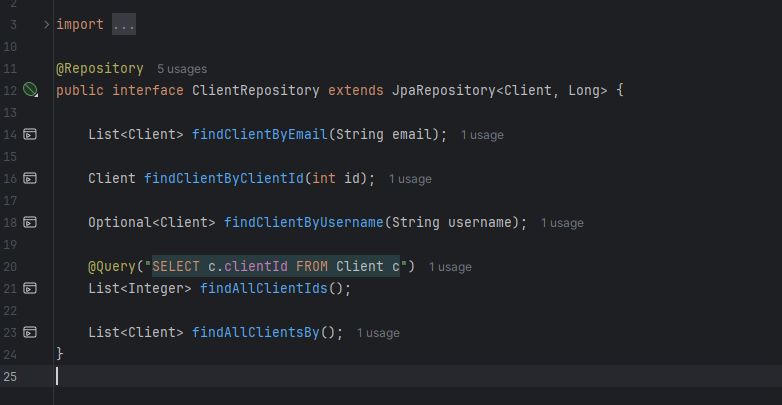
Для реализации функционала работы с моделями были выполнены Service и Repository слои приложения. Для каждой из сущностей был создан класс entityService, реализующий все необходимые методы работы с данной сущностью. Пример полной реализации методов в классе ClientService:



Класс ClientService индивидуально имеет поле типа BCryptPasswordEncoder passwordEncoder, данный объект класса используется для хеширования паролей пользователей при сохранении в базе данный, так же он участвует в авторизации пользователей при входе на вебсайт приложения.

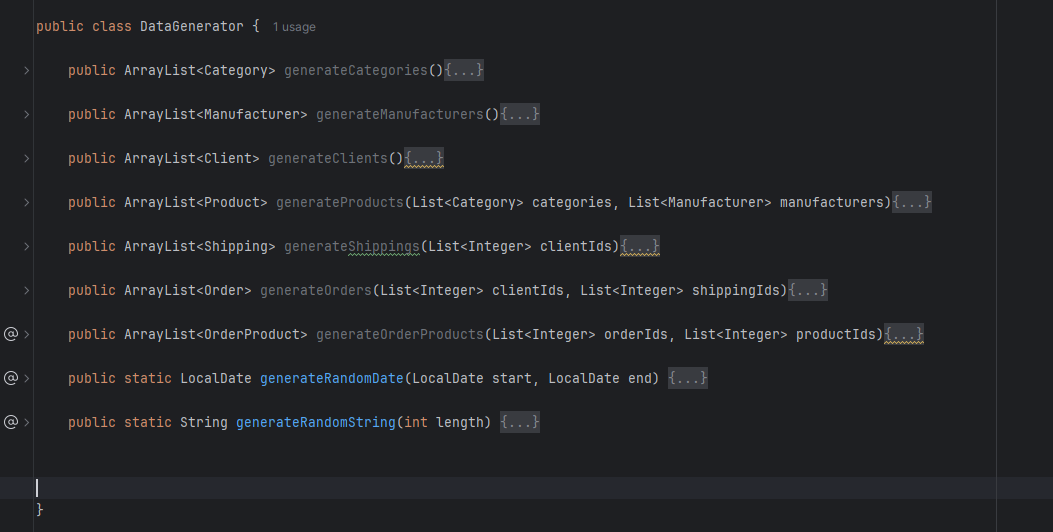
Каждый из Service классов имеет свой экземпляр соответствующего интерфейса слоя Repository, который класс использует для взаимодействия с базой данных.

Пример реализации ClientRepository:



* Заполнение тестовыми данными выполнено через отдельный класс для генерации массивов подходящих записей.

Определённые методы класса генерации данных:



1. **Разработка API**

**API Контроллеры:**

1. **Аутентификация (/api/auth)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Путь** | **Доступ** | **Описание** |
| POST | /api/auth/login | Все | Аутентификация пользователя. Принимает login и password, возвращает JWT. |
| POST | /api/auth/registration | Все | Регистрация нового пользователя. Принимает данные Client и сохраняет в БД. |

1. **Категории (/api/categories)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Путь** | **Доступ** | **Описание** |
| GET | /api/categories | Все | Получение всех категорий |
| GET | /api/categories/id/{id} | Все | Получение категории по ID |
| GET | /api/categories/name/{name} | Все | Получение категории по названию |
| GET | /api/categories/id/all | Все | Получение всех ID категорий |
| GET | /api/categories/name/all | Все | Получение всех названий категорий |
| POST | /api/categories | ADMIN | Создание новой категории |
| DELETE | /api/categories/id/{id} | ADMIN | Удаление категории по ID |

1. **Клиенты (/api/clients)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Путь** | **Доступ** | **Описание** |
| GET | /api/clients/my | isAuthenticated() | Получение данных текущего авторизованного пользователя |
| PUT | /api/clients/my | isAuthenticated() | Обновление данных текущего пользователя (email, имя, фамилия, пароль) |
| POST | /api/clients/my/change-password | isAuthenticated() | Изменение пароля текущего пользователя |
| GET | /api/clients | hasRole('ADMIN') | Получение списка всех клиентов |
| GET | /api/clients/id/{id} | hasRole('ADMIN') | Получение клиента по ID |
| GET | /api/clients/id/all | hasRole('ADMIN') | Получение списка всех ID клиентов |
| GET | /api/clients/username/{username} | hasRole('ADMIN') | Получение клиента по username |
| GET | /api/clients/email/{email} | hasRole('ADMIN') | Получение клиента по email |
| PUT | /api/clients | hasRole('ADMIN') | Обновление данных клиента (только для администратора) |
| DELETE | /api/clients/id/{id} | hasRole('ADMIN') | Удаление клиента по ID |
| POST | /api/clients/login-check | permitAll() | Проверка доступности логина для регистрации |

1. **Производители (/api/manufacturers)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Путь** | **Доступ** | **Описание** |
| GET | /api/manufacturers | Все | Получение списка всех производителей |
| GET | /api/manufacturers/id/{id} | Все | Получение производителя по ID |
| GET | /api/manufacturers/country/{country} | Все | Получение производителей по стране |
| POST | /api/manufacturers | ADMIN | Создание нового производителя |

1. **Товары (/api/products)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Путь** | **Доступ** | **Описание** |
| GET | /api/products | Все | Получение списка всех товаров |
| GET | /api/products/{page} | Все | Получение 9 товаров с указанной страницы |
| GET | /api/products/category/{categoryId}/page/{page} | Все | Получение 9 товаров категории с указанной страницы |
| GET | /api/products/category/{categoryId}/pages | Все | Получение общего количества страниц товаров категории |
| GET | /api/products/pages | Все | Получение общего количества страниц всех товаров |
| GET | /api/products/id/{id} | Все | Получение товара по ID |
| GET | /api/products/categories/id/{id} | Все | Получение товаров по ID категории |
| GET | /api/products/categories/name/{name} | Все | Получение товаров по названию категории |
| GET | /api/products/manufacturers/id/{id} | Все | Получение товаров по ID производителя |
| GET | /api/products/manufacturers/name/{name} | Все | Получение товаров по названию производителя |
| POST | /api/products | ADMIN | Создание нового товара |

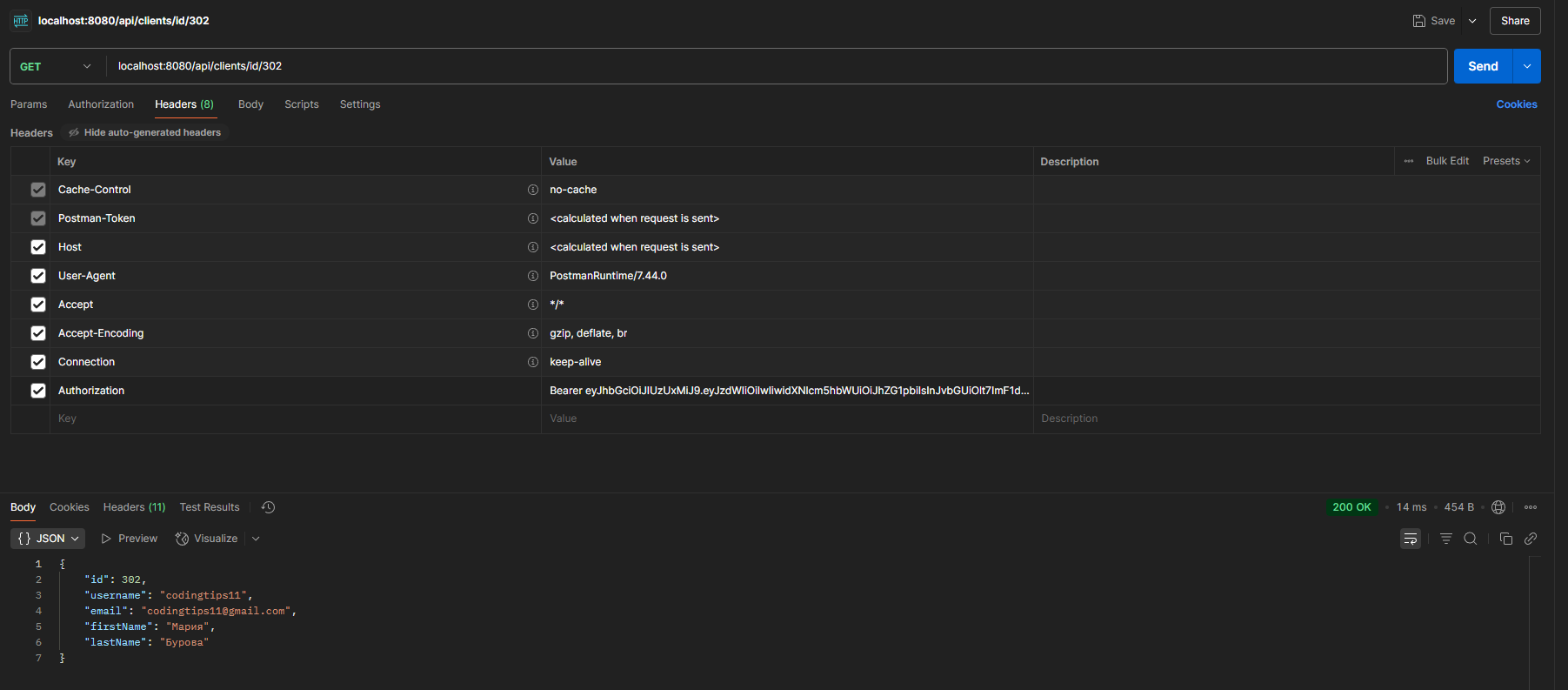
1. **Заказы (/api/orders)**

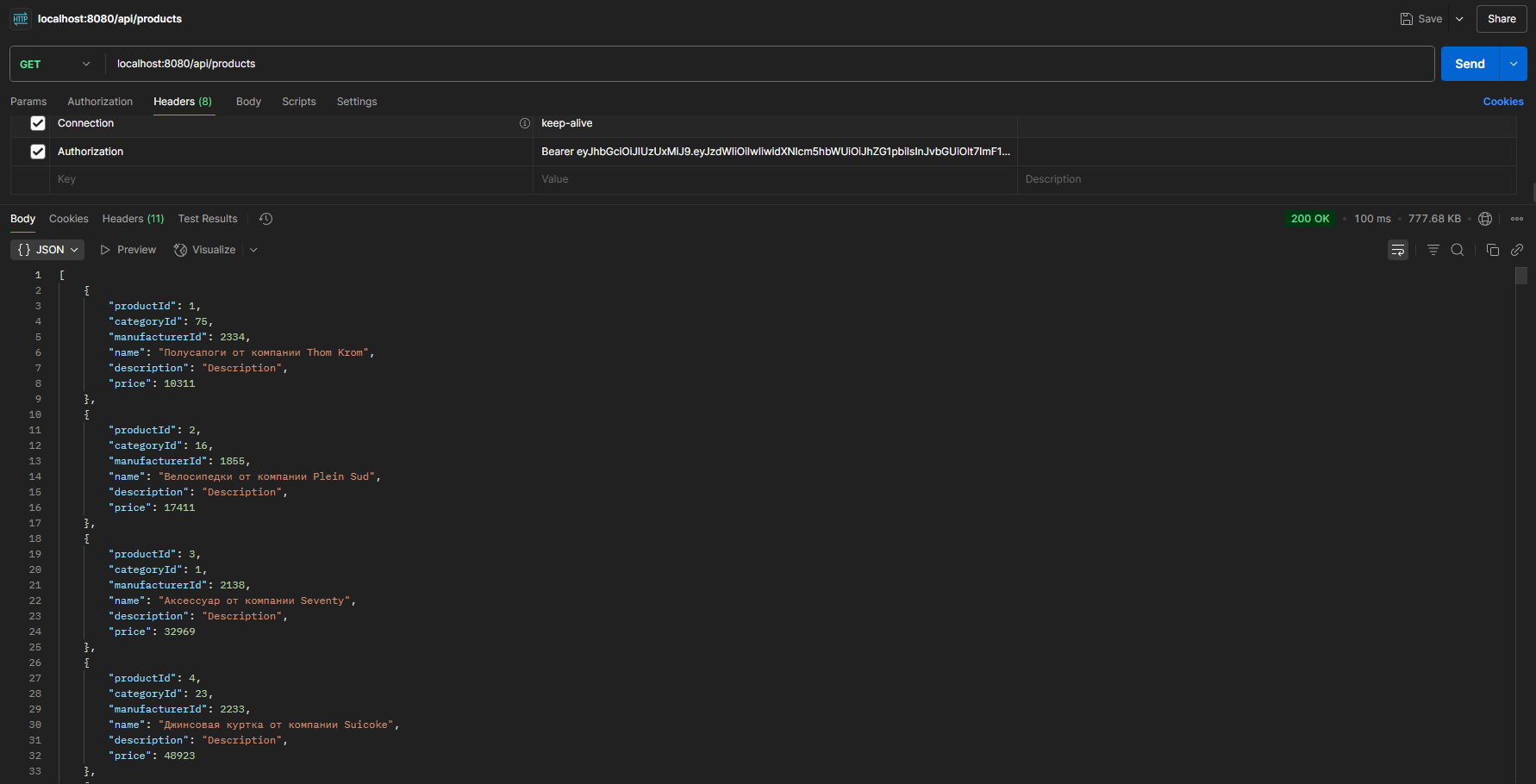
|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Путь** | **Доступ** | **Описание** |
| GET | /api/orders | Все | Получение списка всех заказов |
| GET | /api/orders/my | Авторизован | Получение заказов текущего пользователя |
| POST | /api/orders/my | Авторизован | Создание нового заказа для текущего пользователя |
| GET | /api/orders/id/{id} | Все | Получение заказа по ID |
| GET | /api/orders/clients/id/{id} | ADMIN | Получение заказов по ID клиента (только для администратора) |
| GET | /api/orders/shippings/id/{id} | ADMIN | Получение заказов по ID адреса доставки (только для администратора) |
| POST | /api/orders | ADMIN | Создание заказа (только для администратора) |

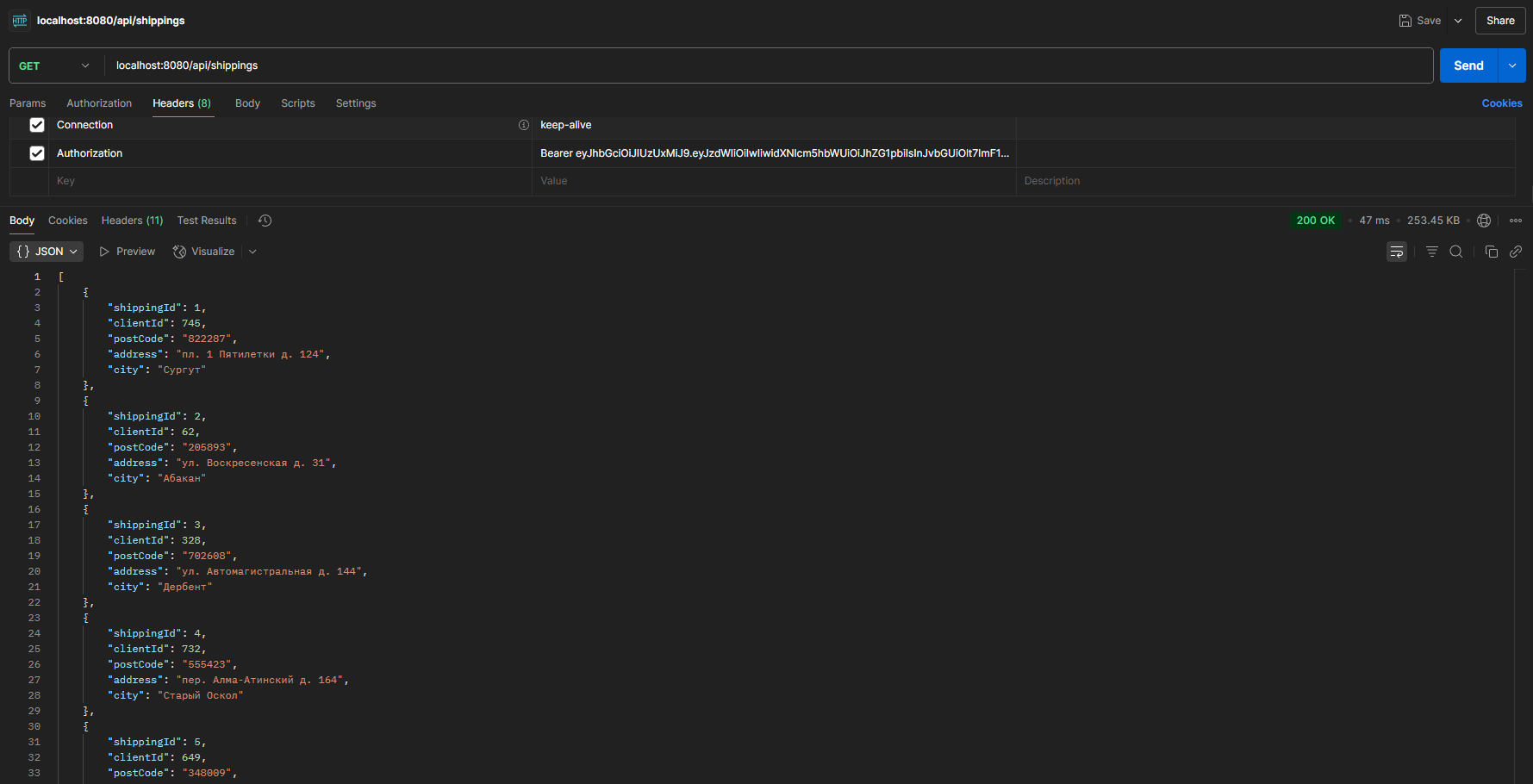
1. **Адреса доставки (/api/shippings)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Путь** | **Доступ** | **Описание** |
| GET | /api/shippings/my | Авторизован | Получение адресов доставки текущего пользователя |
| PUT | /api/shippings/my | Авторизован | Добавление/обновление адреса доставки текущего пользователя |
| DELETE | /api/shippings/my/{shippingId} | Авторизован | Удаление адреса доставки текущего пользователя |
| GET | /api/shippings | ADMIN | Получение всех адресов доставки (только для администратора) |
| GET | /api/shippings/id/{id} | ADMIN | Получение адреса доставки по ID (только для администратора) |
| GET | /api/shippings/clients/id/{id} | ADMIN | Получение адресов доставки по ID клиента (только для администратора) |
| GET | /api/shippings/city/{city} | ADMIN | Получение адресов доставки по городу (только для администратора) |
| POST | /api/shippings | ADMIN | Создание адреса доставки (только для администратора) |

Примеры работы запросов:

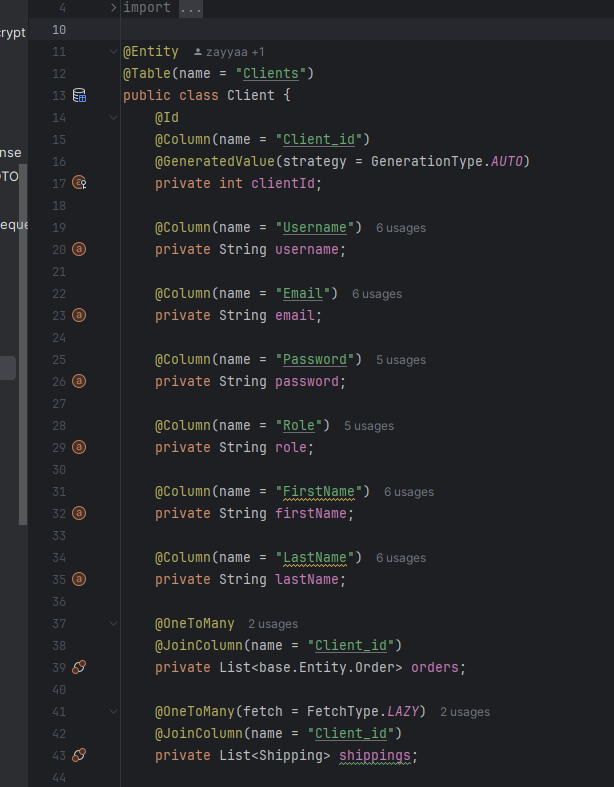




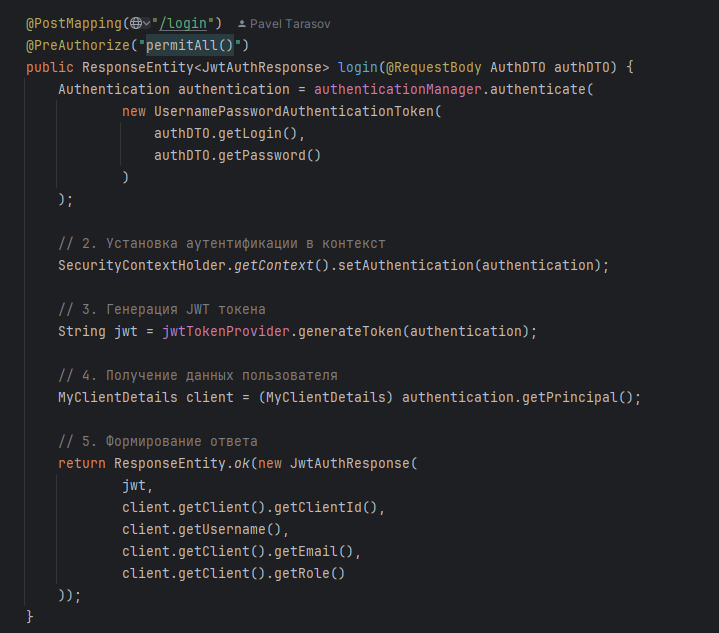


1. **Авторизация**

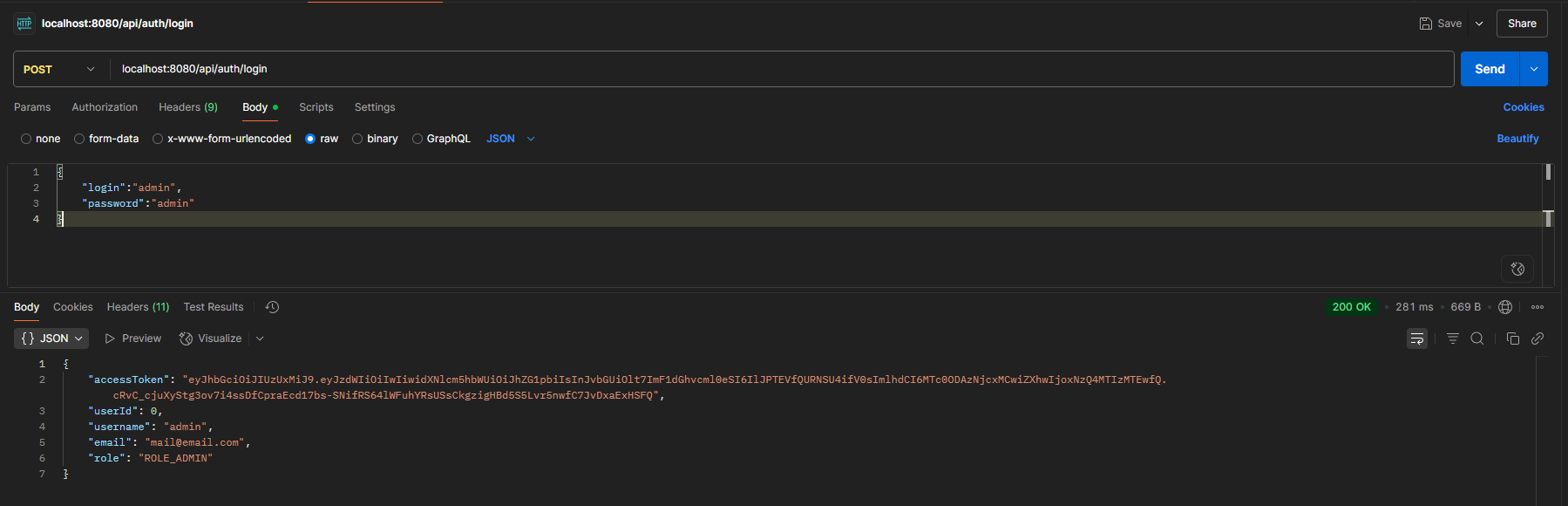
Модель пользователя



Метод генерации JWT токена

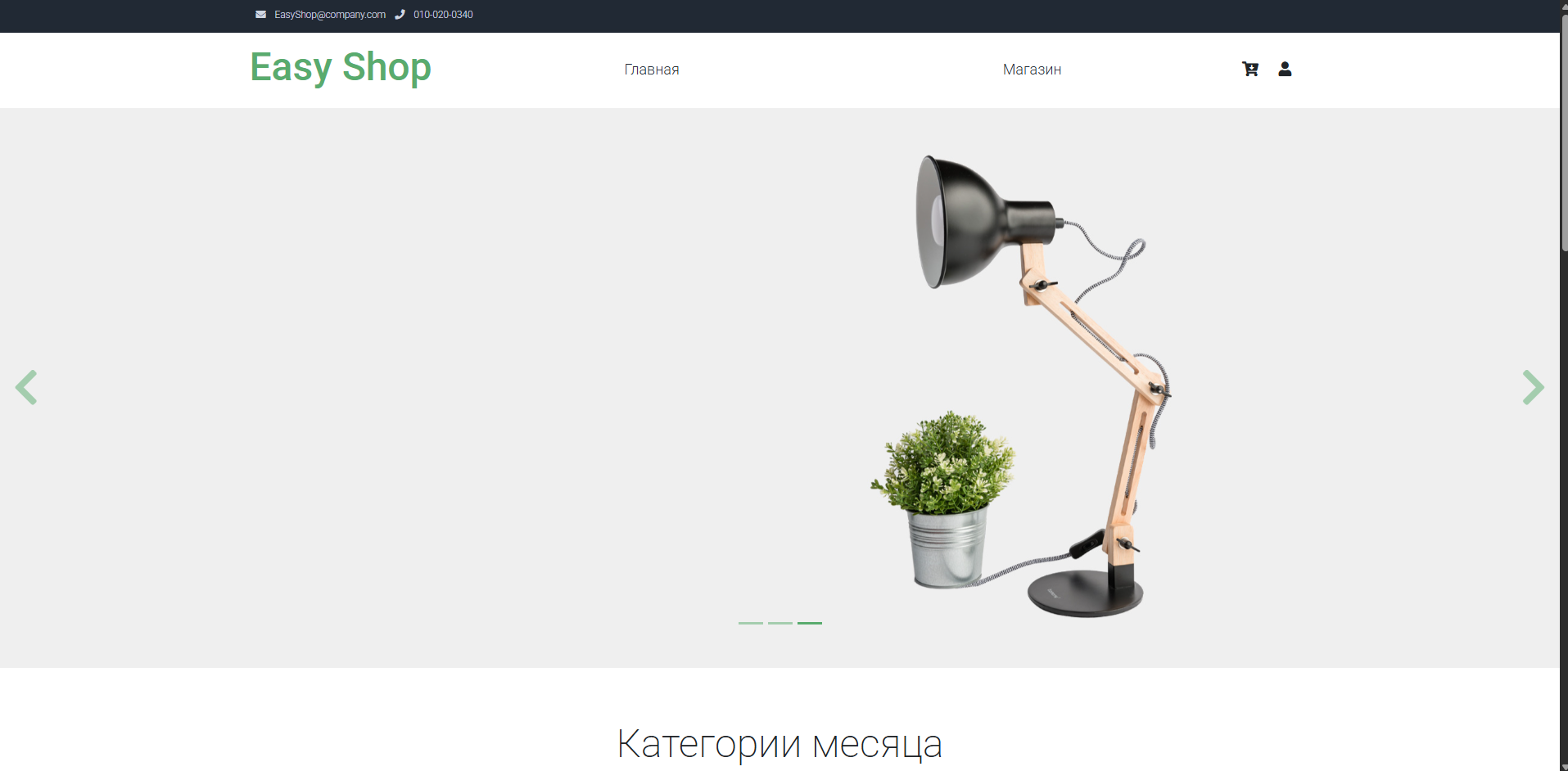


Запрос авторизации

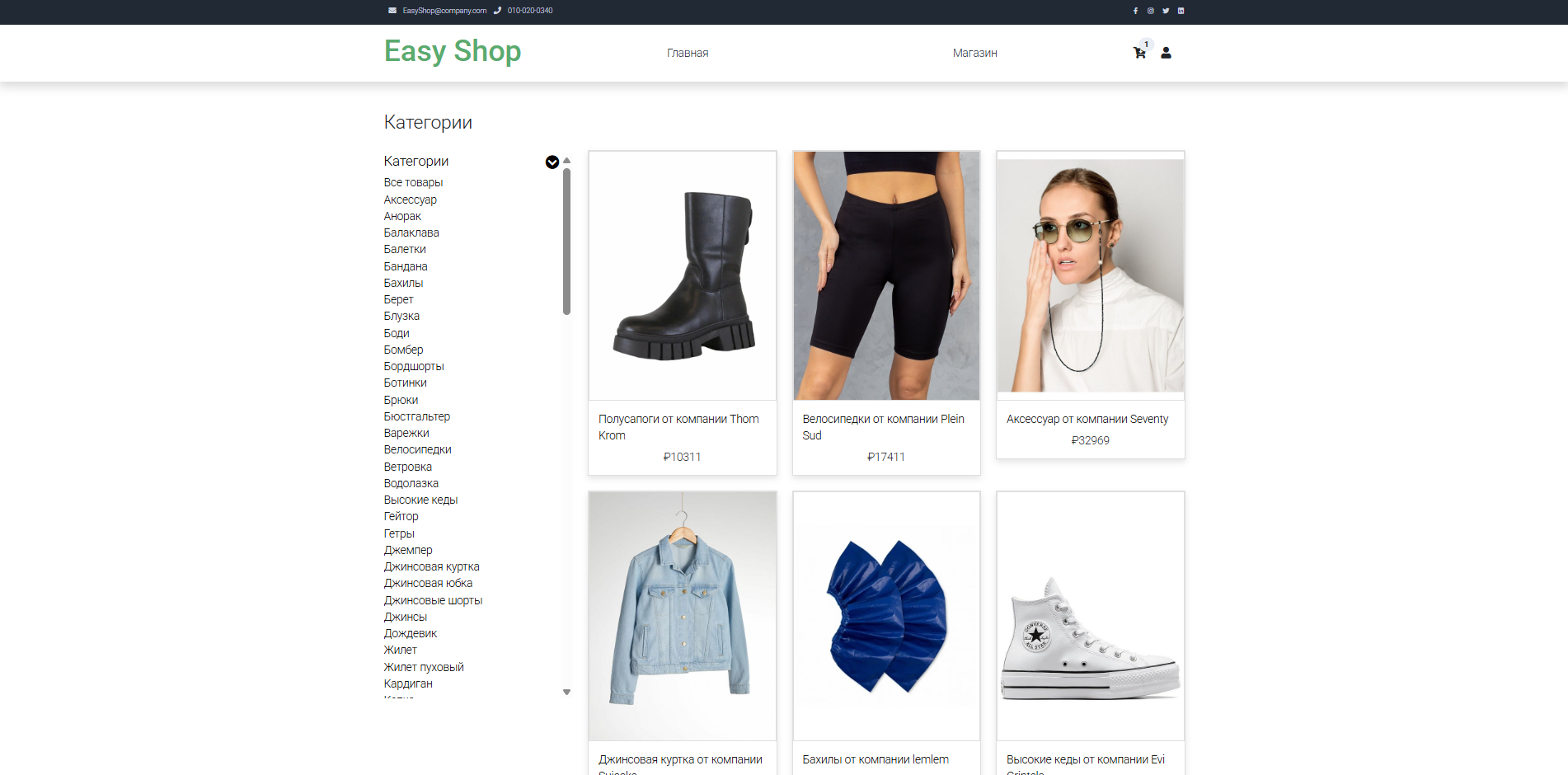


1. **Разработка клиентской части**

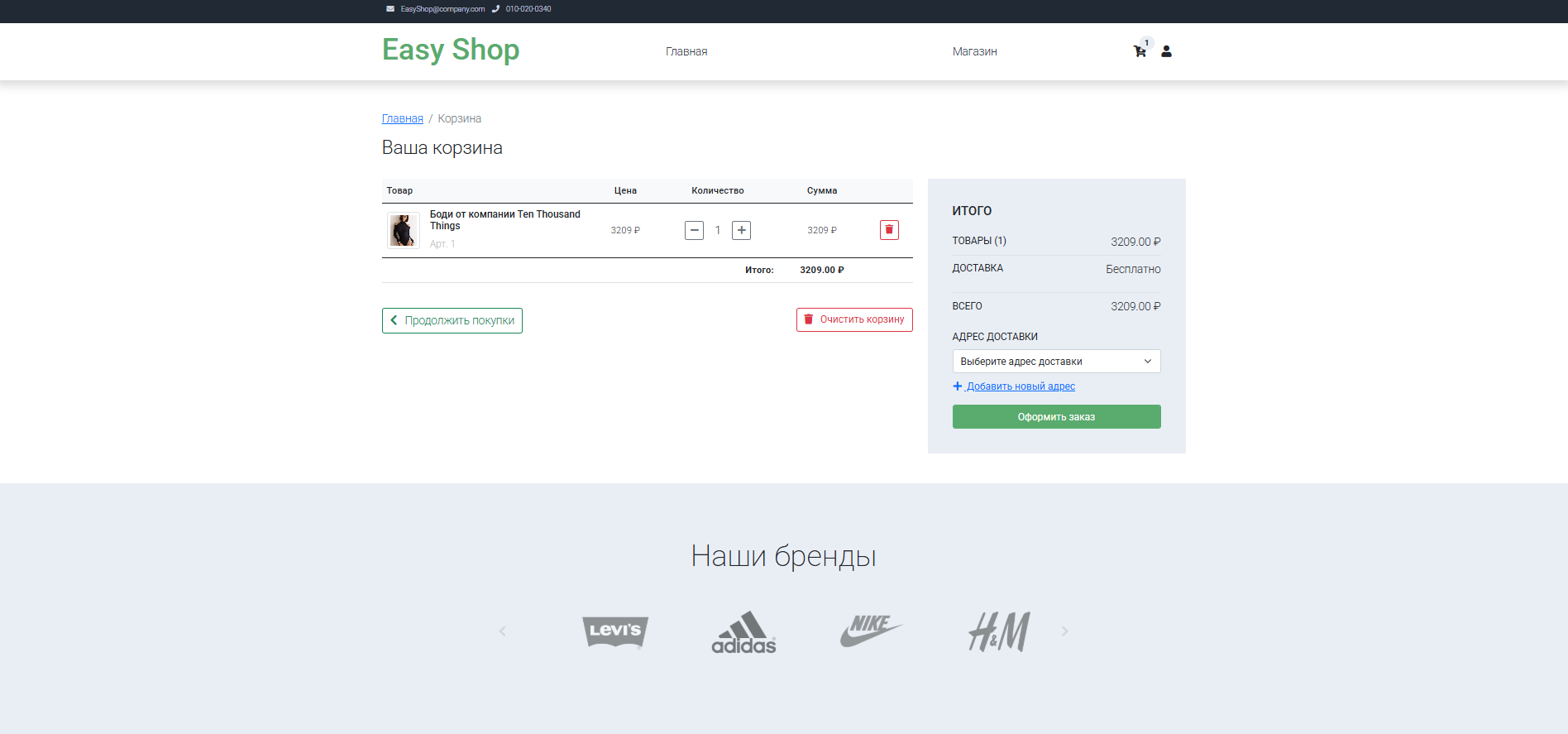
Главная страница интерфейса



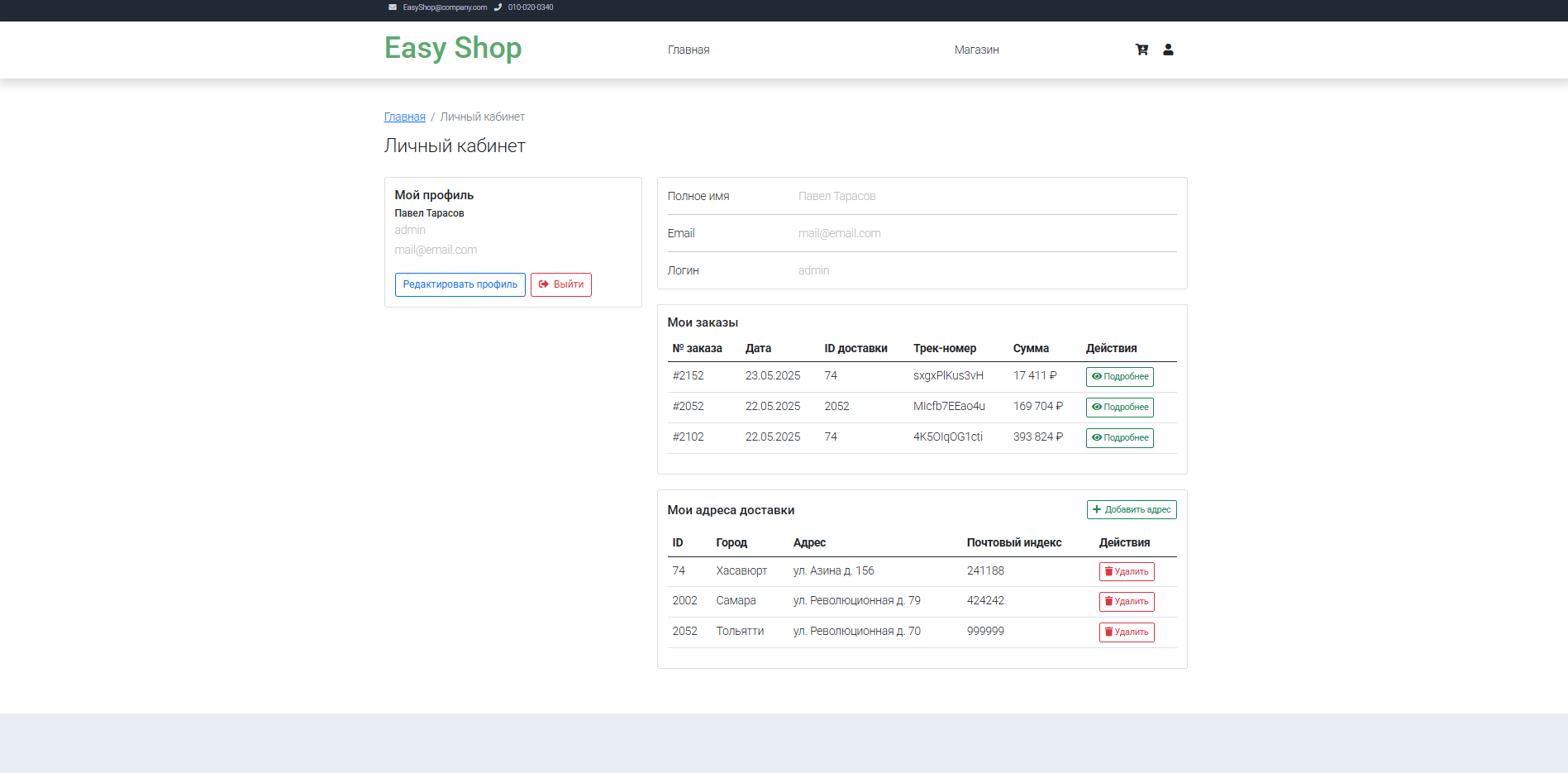
Страница со списком товаров и фильтром по категории



Страница корзины с товарами

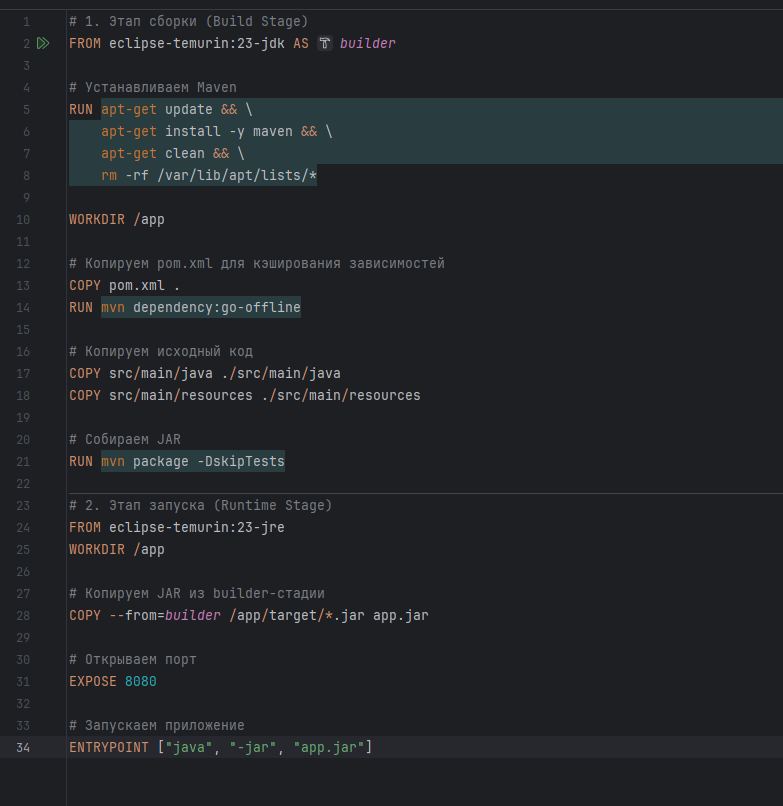


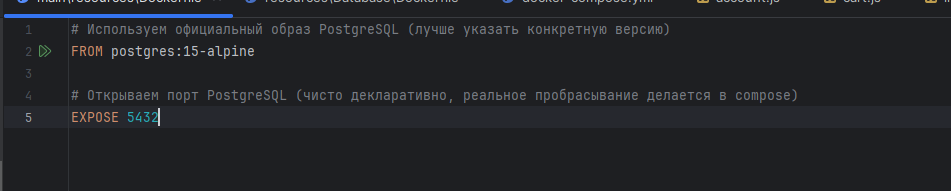
Страница личного кабинета пользователя

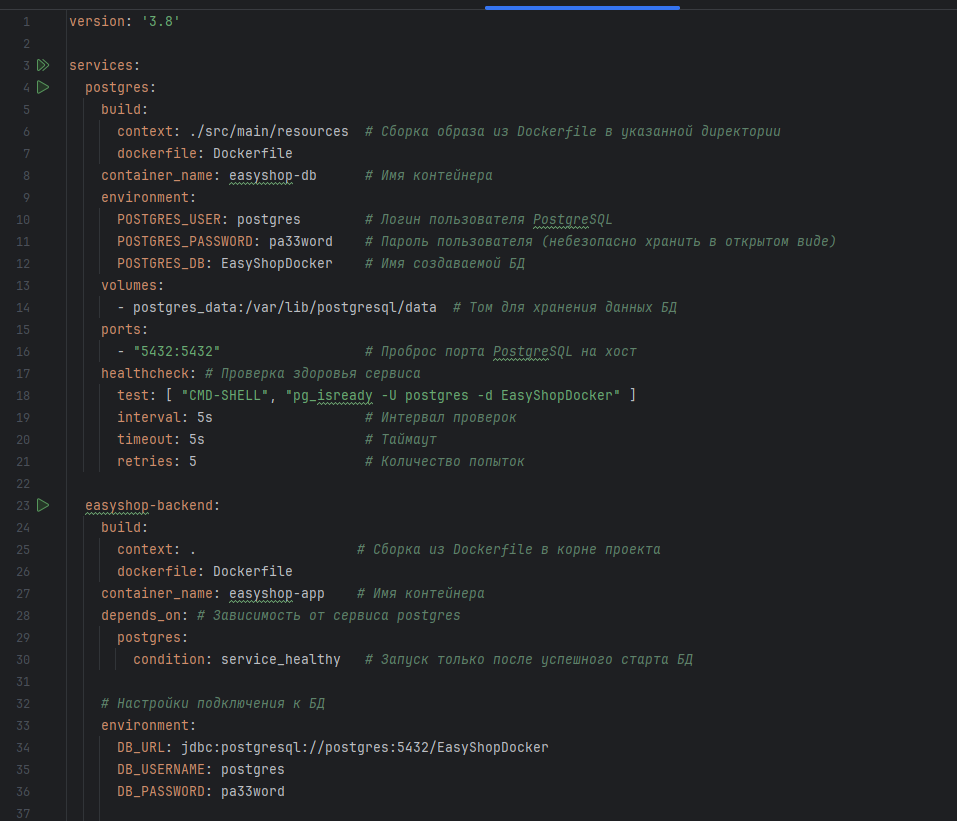


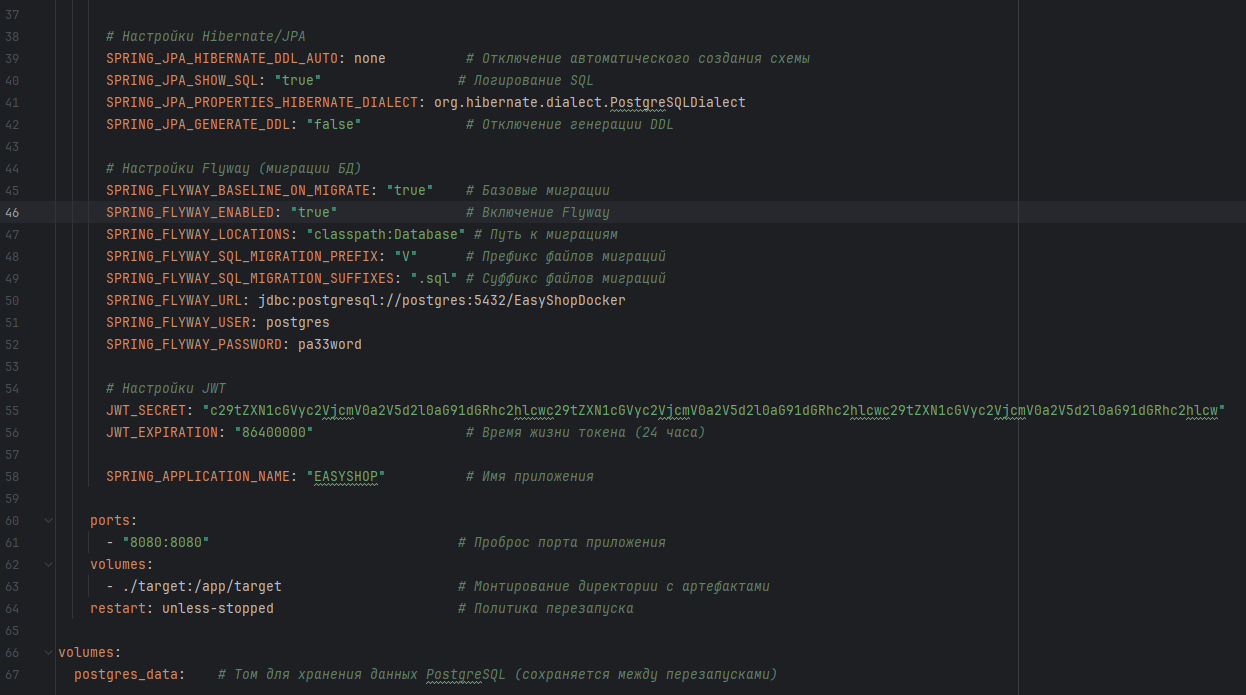
1. **Финализация приложения и упаковка в Docker**

После настройки и тестирования приложения используем докер контейнеры для упаковки приложения, код докер файлов:









После сборки приложения в контейнере получаем готовый к запуску контейнер

