МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

(Самарский университет)

Институт информатики и кибернетики  
Кафедра технической кибернетики

**Итоговый отчет**

по курсу   
Технологии сетевого программирования

Выполнили студенты

Группы 6302-010302D

А.Б Афонин

А.Н. Никонов

Самара 2025

## Описание приложения

**Onlineshop – веб-приложение интернет-магазин. Пользователь может создать учётную запись, после чего набирать товары в корзину. При подтверждении заказа его статус меняется с черновика (корзины) на статус «в обработке». Аутентификация реализована с помощью JWT-токенов. Пользователь может сменить пароль, а также выйти из аккаунта.**

## Архитектура разработанного приложения

**Архитектура:**

* + **Backend:** Python, Django, Django Rest Framework, SimpleJWT.
  + **Frontend:** HTML, CSS, JavaScript
  + **База данных:** PostgreSQL
  + **Контейнеризация:** Docker

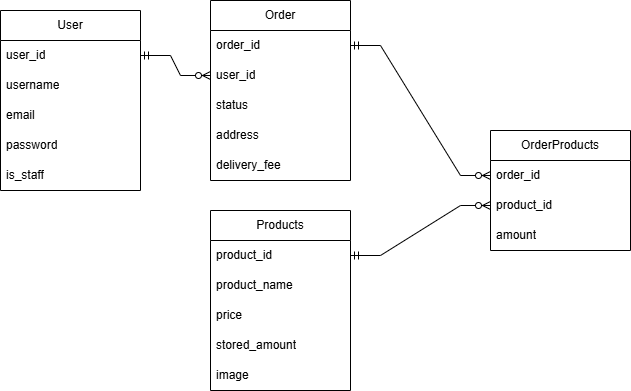
**Схема базы данных**

Рисунок 1 – Схема базы данных

**Сущности и их связи**

* Users (user\_id, username, password, email, is\_staff)
* Products (product\_id, product\_name, price, stored\_amount, image)
* OrderProducts (order\_id, product\_id, amount)
* Orders (order\_id, user\_id, status, address, delivery\_fee)

Для работы с содержимым базы данных как с Python классами использовалась встроенная Django ORM. Ниже приведены примеры создания ORM моделей и запросов к базе данных с использованием Django ORM.

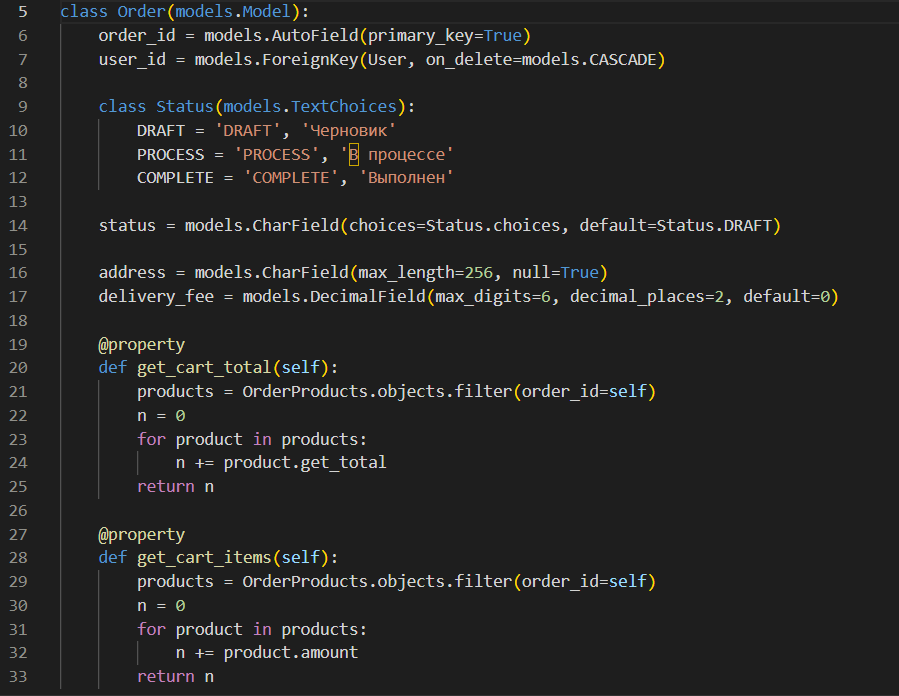


Рисунок 2 – Пример реализации модели заказа для базы данных

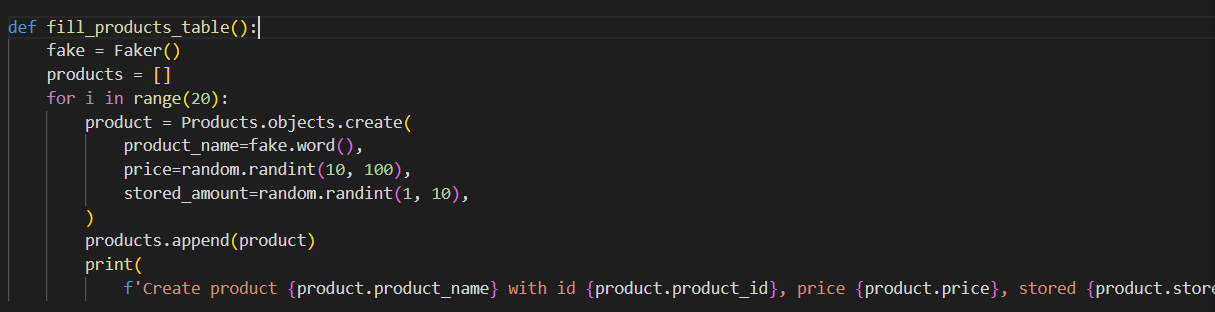


Рисунок 3 – Пример заполнения базы данных тестовыми продуктами

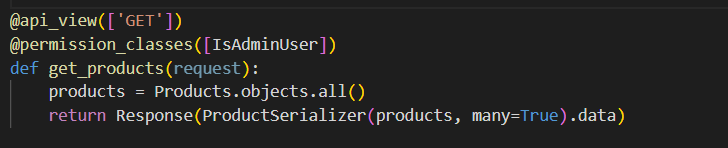


Рисунок 4 – Пример запроса для получения данных

## Структура проекта

Проект состоит из четырёх основных категорий:

* **Пользователи**  – управление пользователями и их данными.
* **Заказы** – создание и управление заказами и их содержимым.
* **Продукты** – работа с товарами.
* **Фронтенд** – клиентская часть приложения.

## Описание основных методов

## 1) Пользователи

## /api/token/refresh – обновление токена

## /users – список всех пользователей (для админа)

## /auth/register – обработка регистрации

## /auth/login – обработка авторизации

## /auth/logout – выход из учётной записи

## /api/change\_password – смена пароля

## /api/user\_info – информация о пользователе

## 2) Заказы

## /update\_item – изменение количества товаров в корзине

## /api/cart – информация о корзине

## /orders – список заказов (для админа)

## /orders/{id} – информация о заказе (для админа)

## /orders/{id}/details – информация о продуктах заказа (для админа)

## /orders/create – создание заказа (для админа)

## /api/process\_order – обработка заказа

## /order\_products – список продуктов всех заказов (для админа)

## /order\_products/create – добавление продукта к заказу (для админа)

## 3) Продукты

## /products – список продуктов (для админа)

## /products/{id} – информация о товаре (для админа)

## /products/create – создание товара (для админа)

4) Фронтенд

* / - основная страница с продуктами
* /cart – корзина пользователя
* /login – вход в аккаунт
* /register – создание учетной записи
* /me – личный профиль
* /payment – подтверждение заказа

## Защита эндпоинтов

## Некоторые эндпоинты в нашем проекте были защищены от входа определённых групп пользователей. В частности, некоторые эндпоинты доступны только авторизованным пользователям, а некоторые – только администратору.

## Для защиты эндпоинтов использовался Django Rest Framework (DRF) и его декораторы. Для проверки аутентификации был написан класс авторизации, который считывал JWT токен из cookie файлов.

## 

## Рисунок 5 – Класс для проверки авторизации

## 

## Рисунок 6 – Установка нашего класса авторизации на использование по умолчанию

## 

## Рисунок 7 – Пример использования декораторов DRF для защиты эндпоинтов

## Клиентская часть проекта

## Для работы с HTML файлами в проекте был использован Django Template Language. В частности, все HTML страницы расширяли основной файл main.html, который содержал в себе навигационную панель. Также он отвечал за получение CSRF токена, который далее использовался при запросах на сервер.

## В некоторых HTML шаблонах мы использовали JavaScript для выполнения запросов к API с помощью функции fetch. Для получения данных с эндпоинтов, доступных только авторизованным пользователям, мы реализовали функцию fetchauth. При истечении срока жизни access токена сервер возвращал статус 401 (Unauthorized). В этом случае fetchauth делал запрос на обновление токена, и в случае успеха повторял изначальный запрос.

## Также JavaScript использовался, чтобы обрабатывать нажатия кнопок на страницах сайта.

## 

## Рисунок 8 – Реализация функции fetchauth

## 

## Рисунок 9 – Пример использования JavaScript для обработки нажатий на кнопки

## Контейнеризация

## Финальное приложение и база данных были упакованы в docker контейнеры. Файл Dockerfile используется для создания образа сервера. Он включает в себя установку необходимых библиотек, которые перечислены в файле requirements.txt.

## В файле docker-compose.yml мы указали два сервиса: django и pgdb, отвечающие за контейнеры для сервера и базы данных соответственно. Для них были указаны необходимые настройки. При этом контейнер django создаётся позже контейнера pgdb.

## Кроме того, база данных будет использовать том (volume) для хранения данных. При остановке контейнера pgdb данные не будут потеряны.

## 

## Рисунок 10 – Содержимое Dockerfile

## Итоги работы

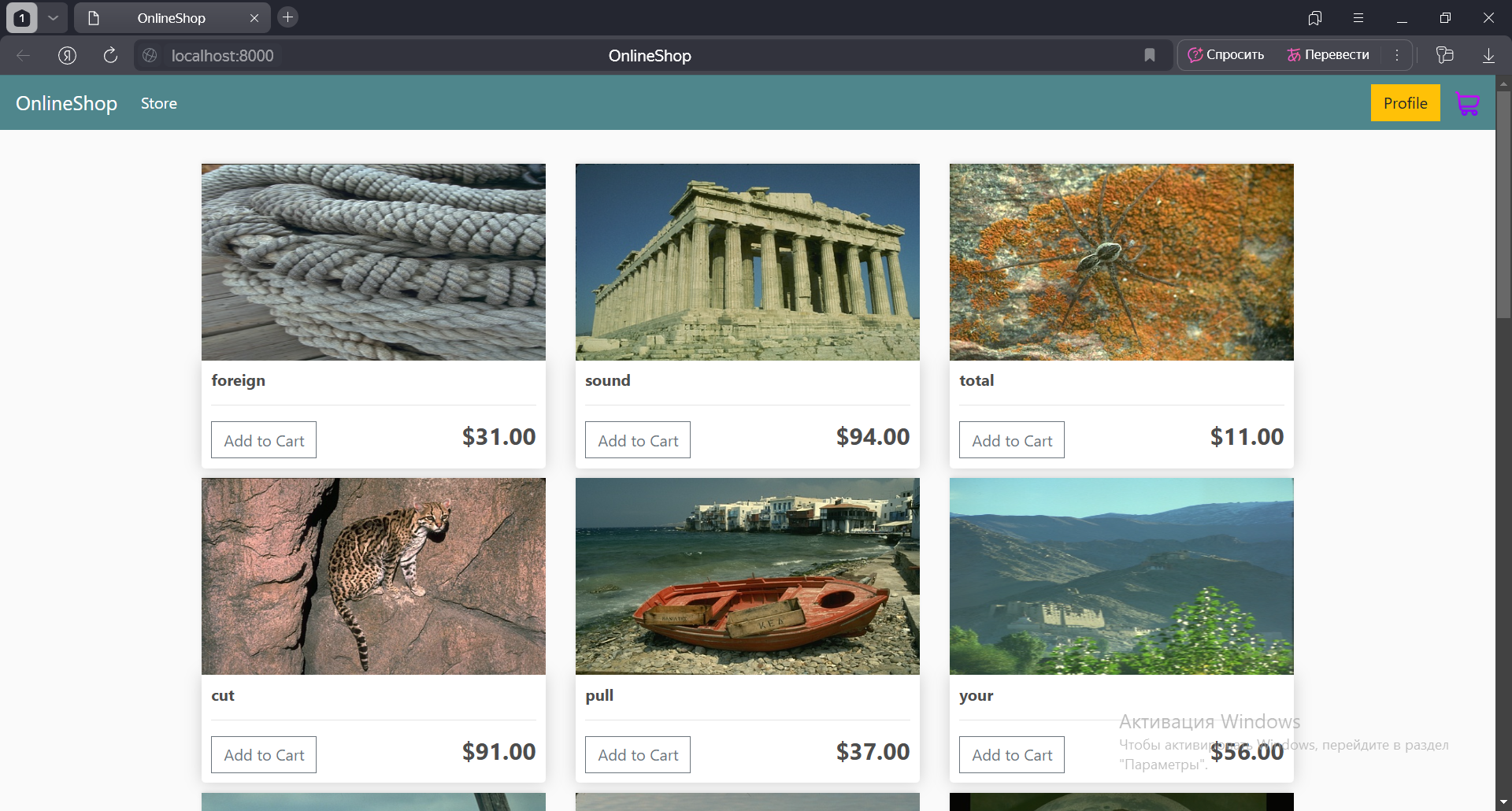


Рисунок 11 – Домашняя страница

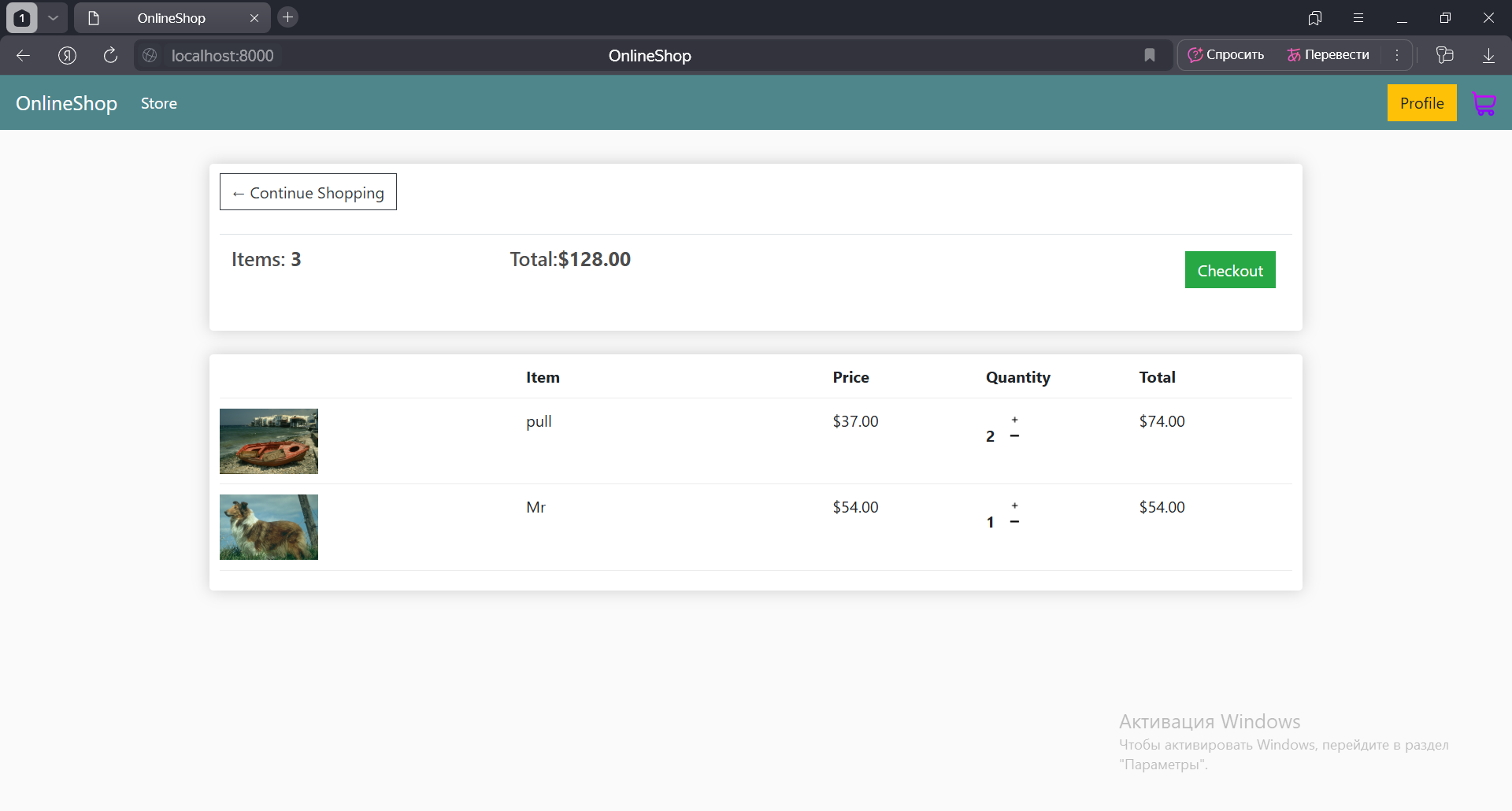


Рисунок 12 – Интерфейс корзины

Ссылка на репозиторий GitHub:

<https://github.com/AlNikonov/OnlineShop/>