Департамент образования и науки города Москвы

Государственное автономное образовательное учреждение высшего образования города Москвы

«Московский городской педагогический университет»

Институт цифрового образования

Департамент информатики, управления и технологий

ДИСЦИПЛИНА:

Интеграция и развертывание программного обеспечения с помощью контейнеров

**Лабораторная работа 3.1**

**Docker Compose для мультиконтейнерных приложений**

Выполнила: Сергеева А. И., группа: АДЭУ-211

Преподаватель: Босенко Т.М.

Москва

2025

**Цель работы:** освоить использование Docker Compose для управления многоконтейнерными приложениями.

**Задачи:**

- Создать файл docker-compose.yml для указанного многоконтейнерного приложения.

- Запустить приложение с помощью Docker Compose.

- Проверить работоспособность приложения и взаимодействие между контейнерами.

- Выполнить индивидуальное задание.

**Ход работы:**

**Вариант st\_99.** **Создать файл docker-compose.yml для системы учета заказов (Django + PostgreSQL). Запустить приложение и проверить обработку заказов. Реализовать расчет среднего времени доставки.**

**Функциональные требования:**

1. Пользователь может управлять своими заказами, а именно создавать, изменять и удалять их, а также просматривать список заказов.

2. Хранение данных в PostgreSQL.

3. Расчет среднего времени доставки.

4. Аутентификация пользователя.

5. Интерфейс пользователя должен быть понятным и иметь вкладку с заказами.

Для начала был создан каталог проекта и выполнен переход в него на рисунке 1.

https://sun9-68.userapi.com/impg/Kw6_2mbFjkvPQsdFl5h2hK6ue76PkK0L3Qztsw/HAfJT0GMd7c.jpg?size=414x38&quality=95&sign=adc7b880eba0f53d5d47896171dcaeff&type=album

Рисунок 1 – Создание каталога проекта и переход в него

Далее необходимо было установить библиотеку для работы с Django на рисунке 2.

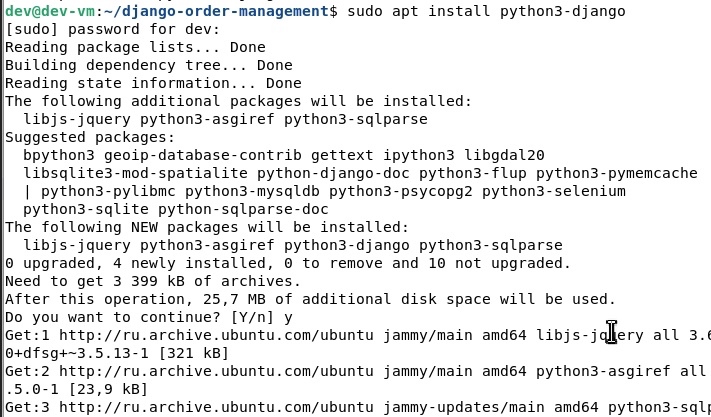


Рисунок 2 – Установка библиотеки для работы с Django

Далее создается новая папка order\_management, содержащая настройки проекта и настройки базы данных, такие как файл settings.py, маршрутизация urls.py и другие необходимые компоненты на рисунке 3.



Рисунок 3 – Создание папки management со всеми необходимыми настройками проекта и базы данных

Далее создается само приложение для учета заказов orders. Файл, который создался автоматически при выполнении проекта, а именно manage.py, отвечает за управление проектом, при обращении к нему вносятся изменения в проект. Команда startapp указывает на создание нового приложения. Процесс создания продемонстрирован на рисунке 4.



Рисунок 4 – Создание приложения orders

Зависимости для базы данных PostgreSQL прописаны в файле requirements.txt на рисунке 5.

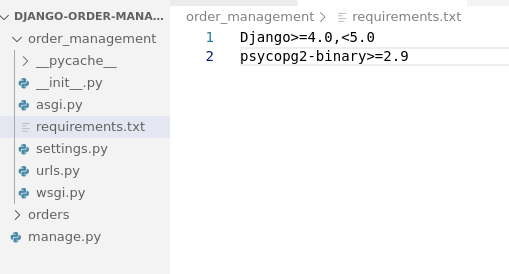


Рисунок 5 – Создание файла requirements.txt

В файле order\_management/settings.py были выполнены настройки подключения к базе данных PostgreSQL на рисунке 6.

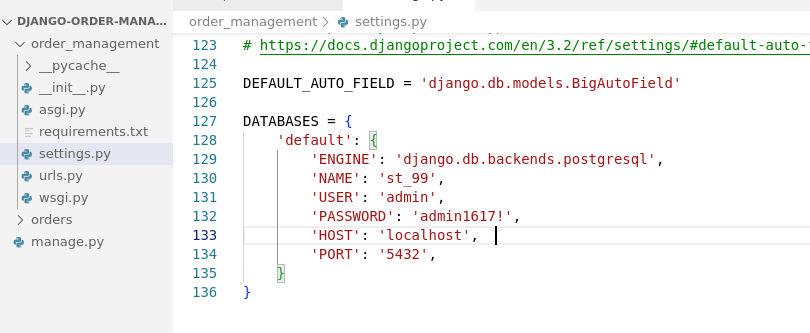


Рисунок 6 – Настройка подключения к базе данных PostgreSQL

Затем был отредактирован файл orders/models.py для создания модели учета заказов на рисунке 7. В модели определяются поля, а именно название продукта, в котором хранятся текстовые данные, дата доставки с типом поля даты, также ведется расчет средней длительности доставки.

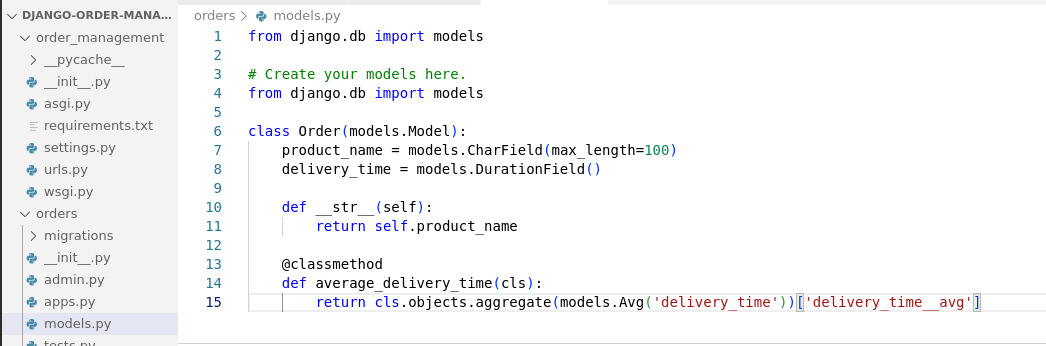


Рисунок 7 – Создание модели учета заказов

Далее была выполнена команда для создания и применения миграций, которая показана на рисунках 8-9. Миграции в Django — это механизм, который позволяет управлять изменениями в структуре базы данных приложения. Они обеспечивают удобный способ создания, изменения и удаления таблиц и полей в базе данных, синхронизируя их с моделями Django.



Рисунок 8 –Создание миграции

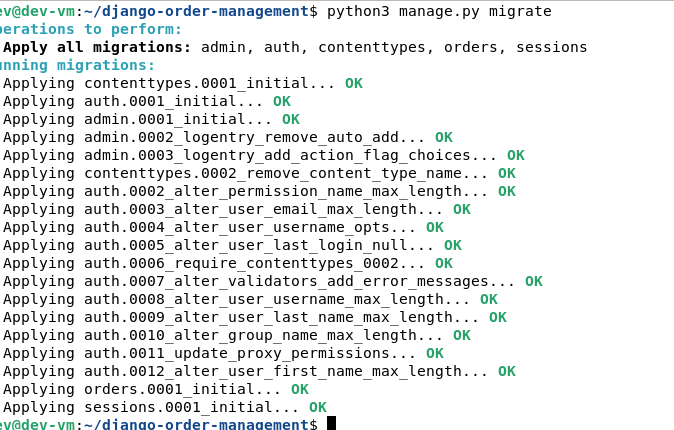


Рисунок 8 – Применение миграции

Был создан Dockerfile на основе образа python:3.9. Был добавлен путь к каталогу, в котором будут выполняться все команды, файл requirements.txt был скопирован в рабочий каталог контейнера с указанными зависимостями в нем, далее необходимые зависимости устанавливаются, далее все файлы копируются в текущую директорию рабочего каталога контейнера на рисунке 9.

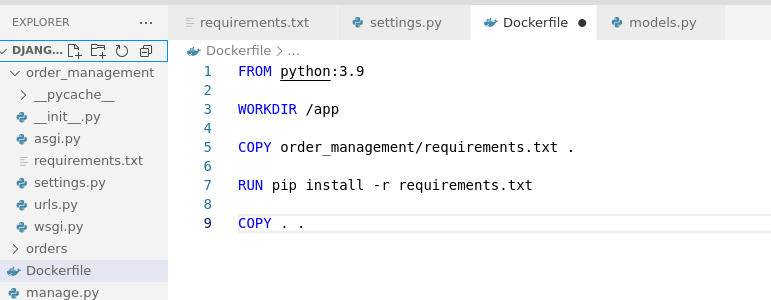


Рисунок 9 – Создание Dockerfile

Далее был создан docker-compose.yml на рисунке 10. Для запуска определен web сервис, настраивается доступ к приложению извне, настраивается сервис db настраивает подключение к базе данных.

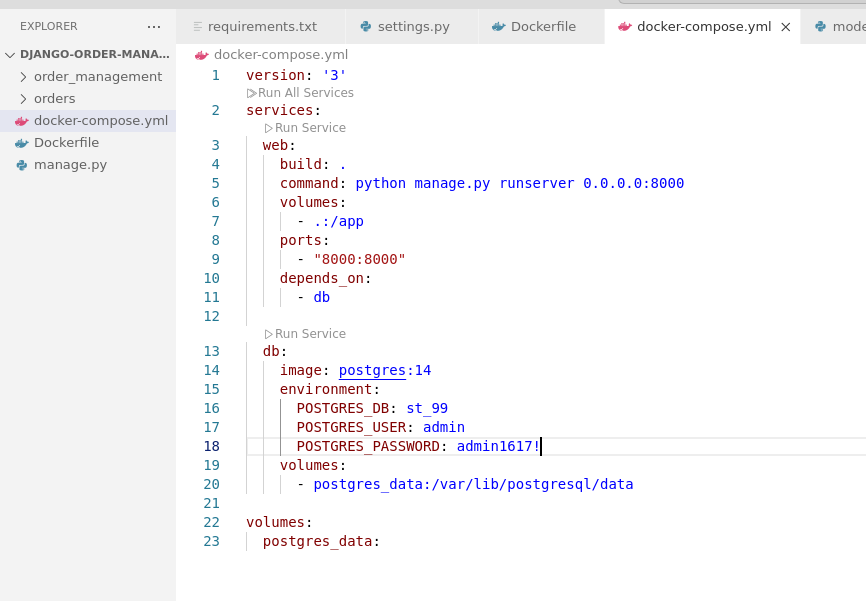


Рисунок 10 – Создание файла docker-compose.yml

Далее был выполнен запуск приложения на рисунке 11.

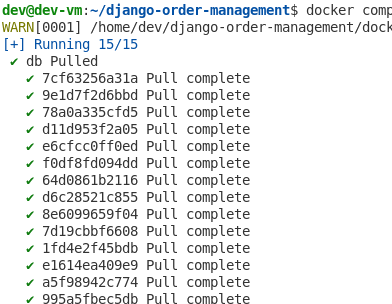


Рисунок 11 – Запуск приложения

При переходе по ссылке <http://localhost:8000> все работает, появляется интерфейс Django на рисунке 12.

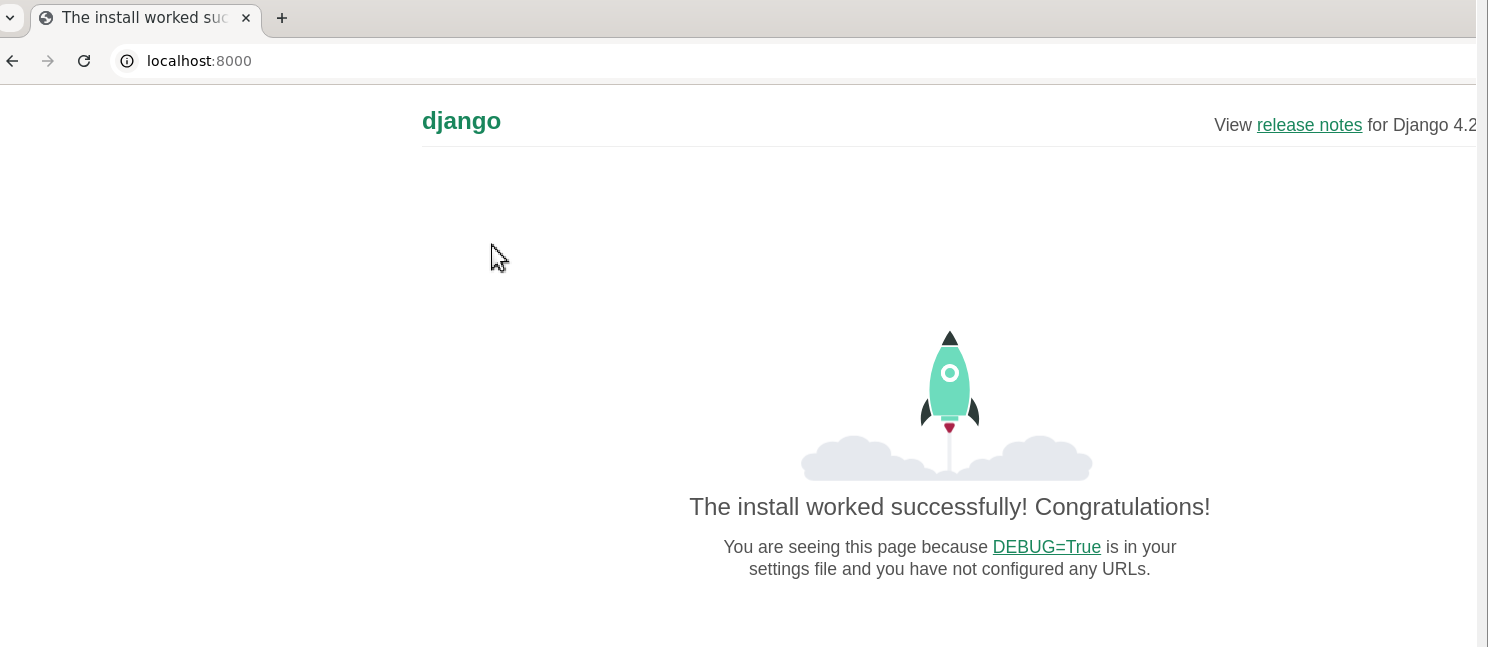


Рисунок 12 – Запуск Django

Далее в файл orders/views.py было добавлено представление, чтобы отправлять заказы и отображать среднее время доставки на рисунке 13.

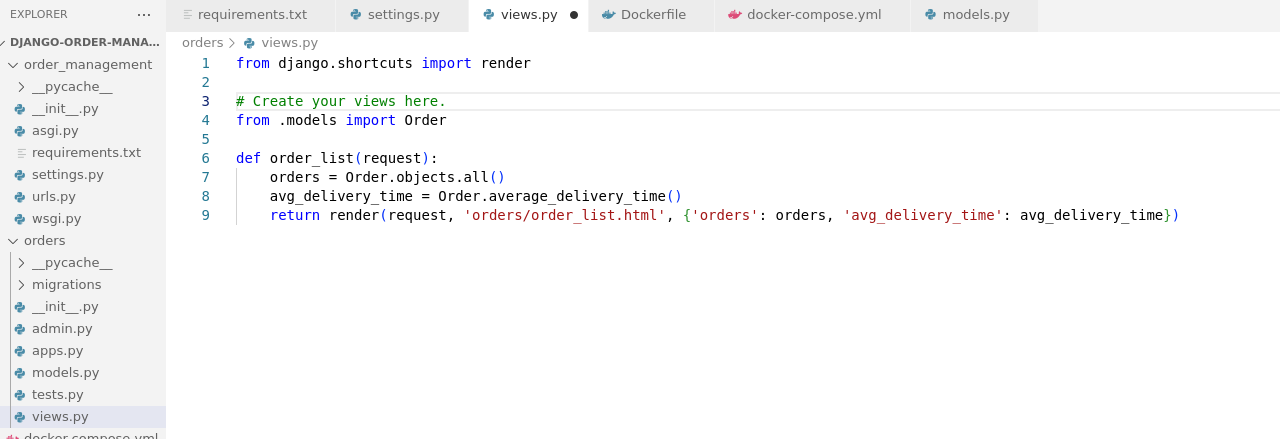


Рисунок 13 – Добавление представления для отправки заказов и отображения среднего времени доставки

Для указания маршрутизации были внесены изменения в файл order\_management/urls.py на рисунке 14. Когда пользователь переходит к /admin/, он будет перенаправлен в административный интерфейс Django. Когда пользователь переходит к /orders/, вызывается функция order\_list, которая определена в views.py приложения orders.

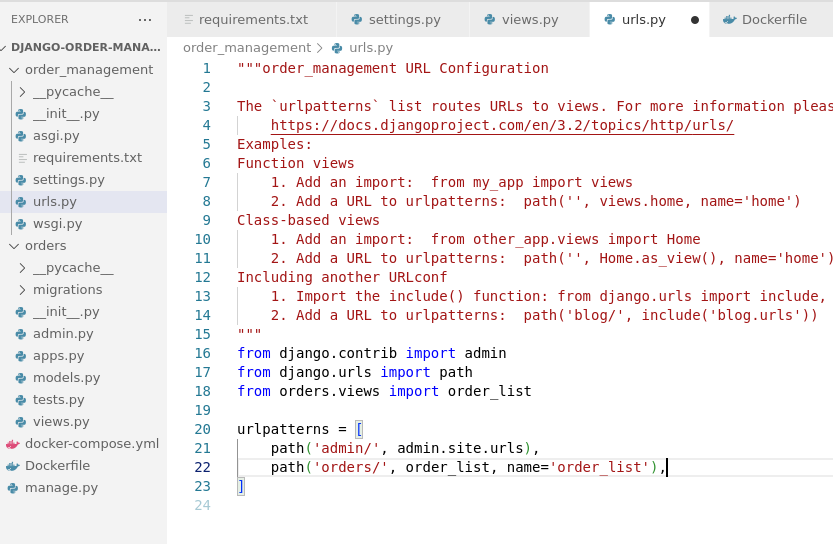


Рисунок 14 – Добавление маршрутизации для списка заказов

Далее создается html файл с шаблоном, как должно выглядеть отображение списка заказов и средней даты доставки на рисунке 15. Для каждого заказа отображается название и дата доставки, в конце отдельно выводится время доставки.

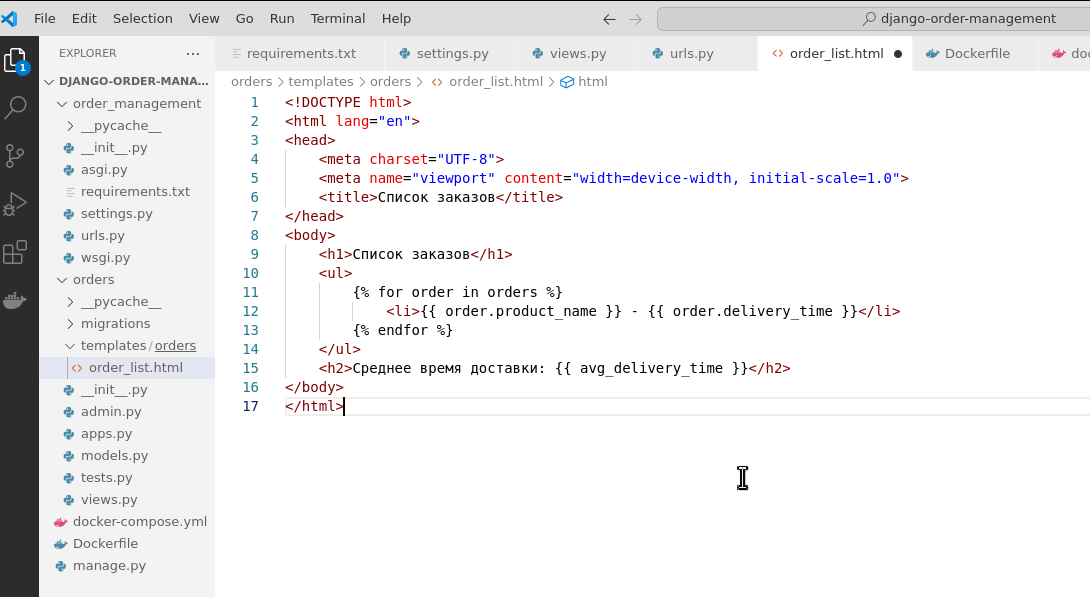


Рисунок 15 – Создание шаблона для отображения заказов и средней даты доставки

Для тестирования был создан пользователь. Для начала была открыта оболочка в контейнере web и выполнено создание пользователя на рисунках 16-17.



Рисунок 16 – Выполнение в оболочке контейнера web

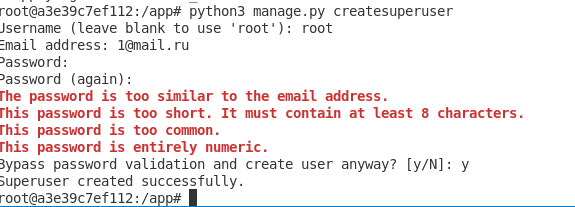


Рисунок 17 – Создание пользователя

Был выполнен вход на рисунке 18.

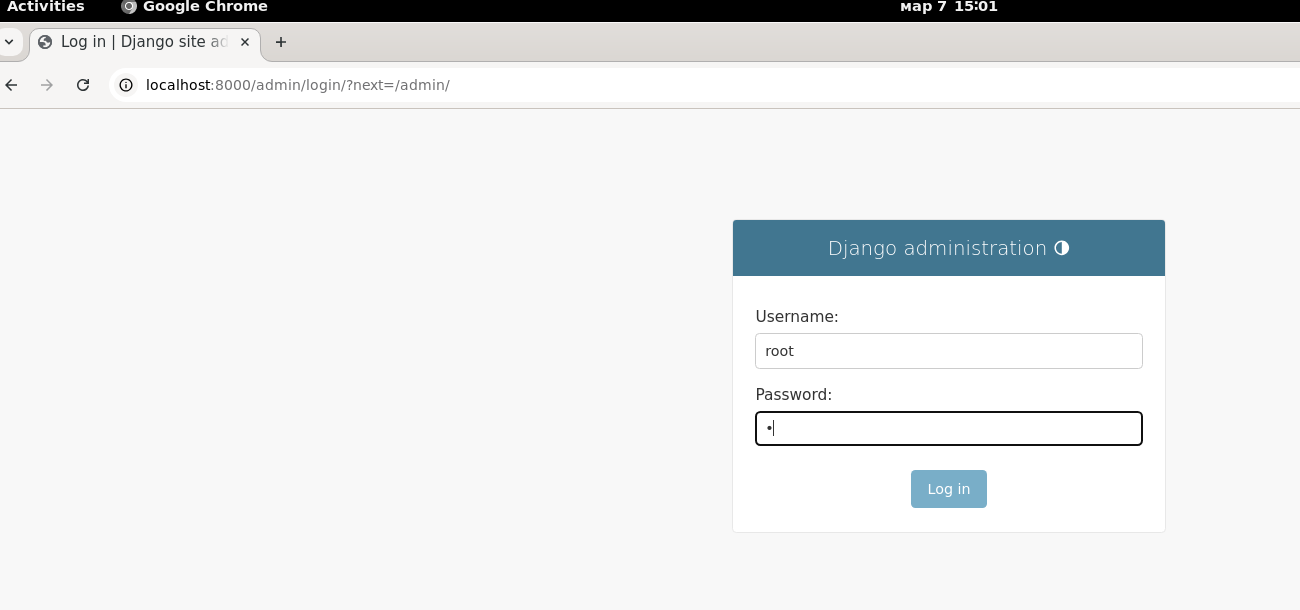


Рисунок 18 – Ввод логина и пароля

Для отображения заказов необходимо также внести изменения в admin.py на рисунке 19.

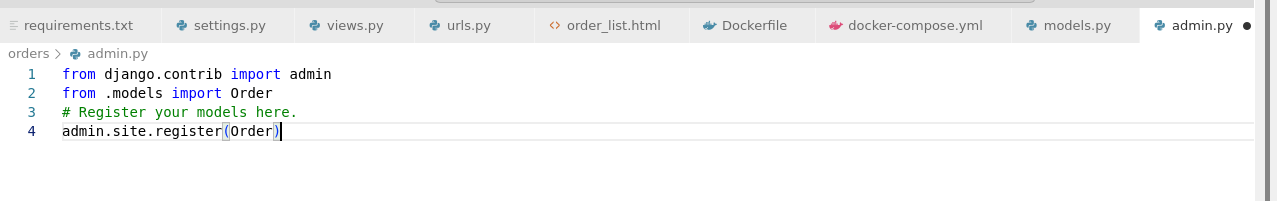


Рисунок 19 – Указание модели в admin.py

Далее показано тестовое добавление заказов на рисунках 20-23.

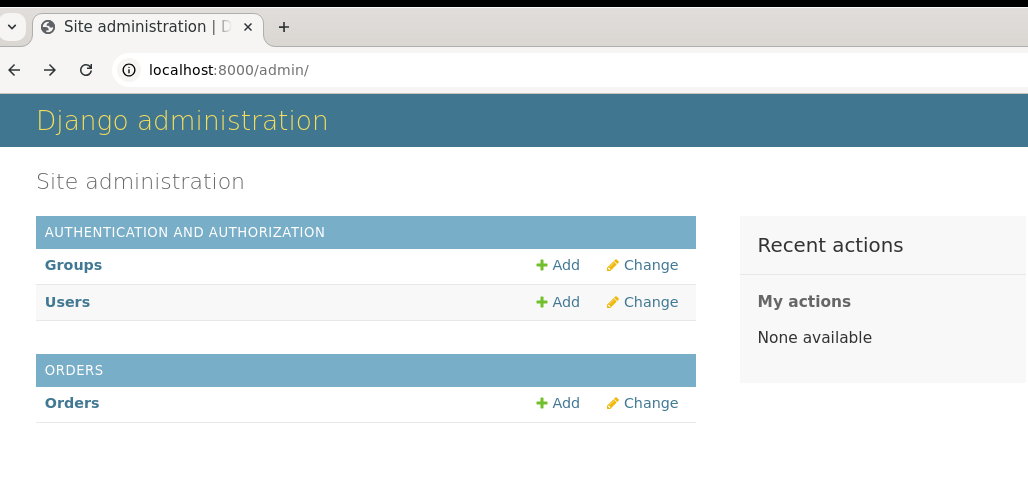


Рисунок 20 – Главный интерфейс

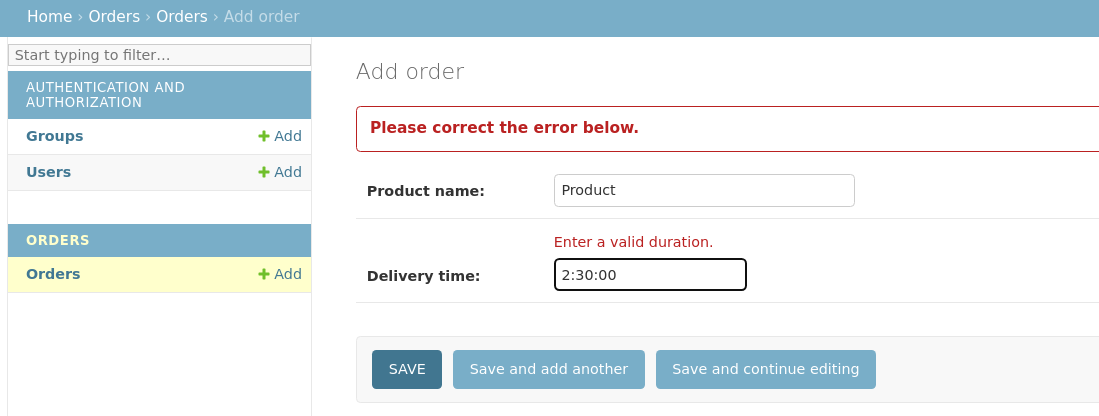


Рисунок 21 – Пример создания заказа с датой доставки 2 часа 30 минут

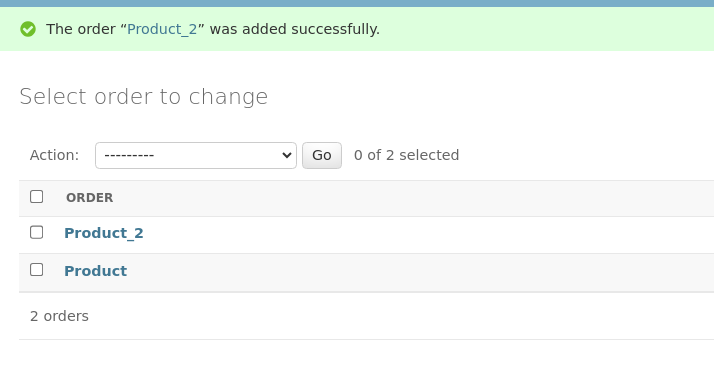


Рисунок 22 – Список созданных заказов

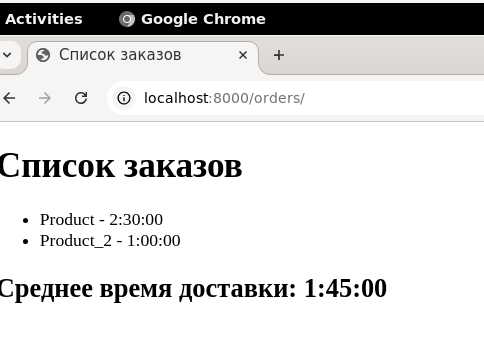


Рисунок 23 – Отображение списка заказов и среднего времени доставки

Схема взаимодействия пользователя с системой представлена на рисунке 24.

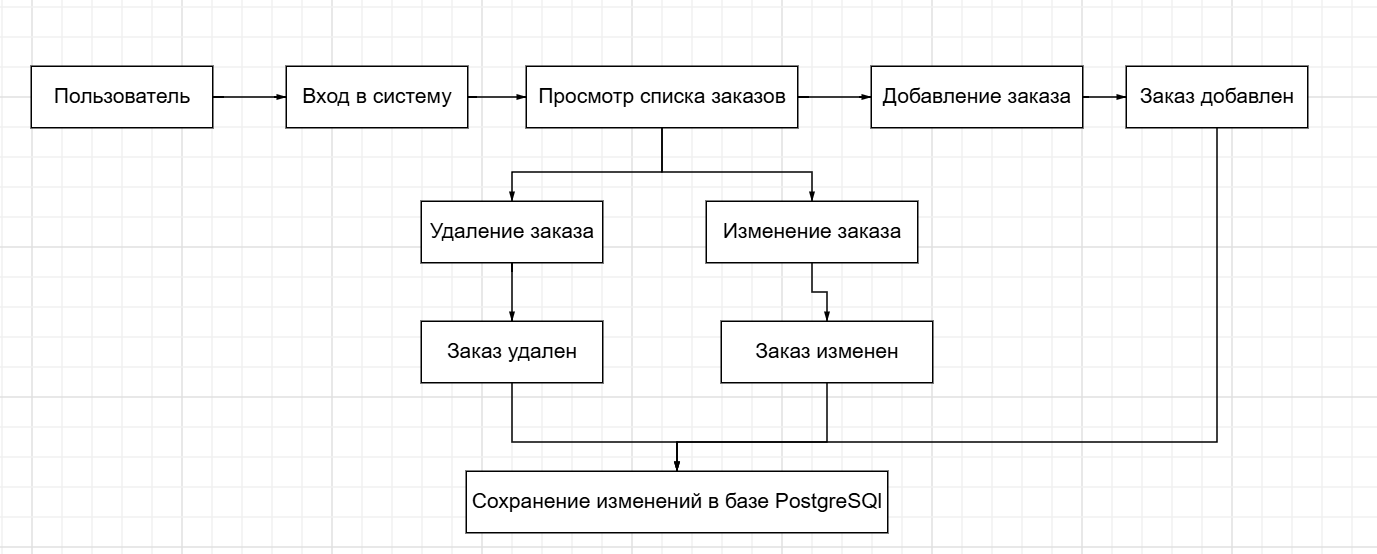


Рисунок 24 – Схема взаимодействия пользователя с системой

Структура папки проекта показана на рисунке 25.

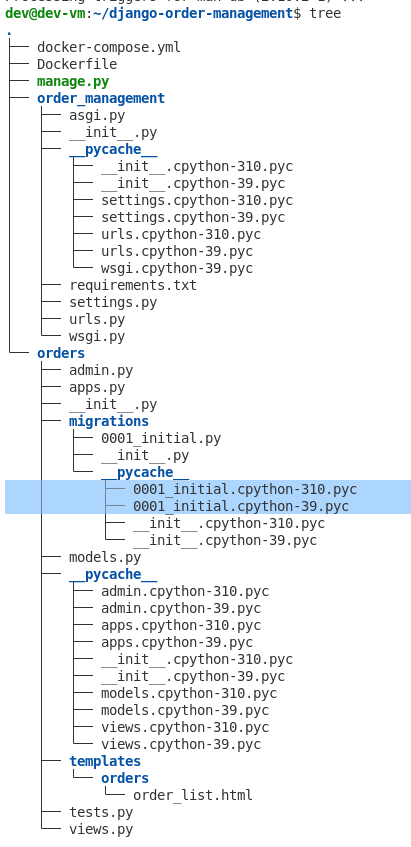


Рисунок 25 – Структура папки проекта

**Общий вывод**:

В ходе лабораторной работы было освоено использование Docker Compose для управления многоконтейнерными приложениями. Были разработаны функциональные требования и схема взаимодействия пользователя с системой. Также был получен опыт по работе с Django, созданы тестовые данные, представление для отображения данных, пользователи, настроены типы данных, подключение к базе данных. Django предоставило полный набор инструментов и библиотек, которые позволили разработать приложение от начала до конца. Встроенные функции, такие как аутентификация, администрирование, маршрутизация, упрощают процесс разработки. Цель и задачи были выполнены.

**Ответы на контрольные вопросы:**

1. Что такое Docker Compose и для чего он используется?

Docker Compose — это инструмент для определения и запуска многоконтейнерных Docker приложений. Он позволяет разработчикам описывать конфигурацию сервисов, сетей и томов в одном YAML-файле (обычно docker-compose.yml), что обеспечивает более простое управление сложными проектами, состоящими из нескольких контейнеров.

2. Какие основные преимущества использования Docker Compose для управления многоконтейнерными приложениями?

* Упрощение конфигурации: Все настройки сервисов описаны в одном файле, что делает управление проектом более организованным.
* Легкость в запуске и остановке: Команды docker-compose up и docker-compose down позволяют легко запускать и останавливать все сервисы одновременно.
* Изоляция среды: Docker Compose изолирует зависимости приложения в контейнерах, исключая конфликты с зависимостями других проектов.
* Удобство разработки: Монтирование локальных каталогов в контейнеры (с помощью volumes) позволяет разработчикам вносить изменения в код без необходимости перезапуска контейнеров.
* Управление зависимостями.

3. Какие основные разделы и директивы используются в файле docker-compose.yml?

* version: Указывает версию схемы Docker Compose.
* services: Определяет сервисы, которые будут запущены в контейнерах. Каждый сервис может иметь свои собственные настройки, такие как:
* build: Определяет, как строить образ (например, из Dockerfile).
* image: Указывает используемый образ Docker.
* command: Команда, которая будет выполнять в контейнере.
* volumes: Определяет, какие директории будут монтироваться из хостовой системы или других контейнеров.
* ports: Перенаправление портов из контейнера на хост.
* depends\_on: Определяет зависимости между сервисами.
* volumes: Определяет именованные тома, которые могут использоваться сервисами для хранения данных.

4. Как запустить многоконтейнерное приложение с помощью Docker Compose?

- Убедитесь, что у вас есть файл docker-compose.yml с правильной конфигурацией.

- Перейдите в каталог, где находится файл docker-compose.yml в терминале.

- Выполните команду:

docker compose up

Эта команда будет собирать все указанные образы и запускать контейнеры.

Если вы хотите запустить контейнеры в фоновом режиме (detached mode), добавьте флаг -d:

docker compose up –d

5. Как остановить и удалить контейнеры, запущенные с помощью Docker Compose?

docker compose down