МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ имени академика С.П.КОРОЛЕВА»

ИНСТИТУТ ИНФОРМАТИКИ И КИБЕРНЕТИКИ

КАФЕДРА ТЕХНИЧЕСКОЙ КИБЕРНЕТИКИ

ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ

Тема: «Проектирование приложения»

Дисциплина: «Технологии сетевого программирования»

Выполнил: Ашугян Т.А., гр.6303

Проверил: Кашапов А.И.

Самара 2025

СОДЕРЖАНИЕ

[Введение 3](#_Toc199606722)

[1 Структура приложения 4](#_Toc199606723)

[1.1 Компоненты приложения 4](#_Toc199606724)

[1.2 Docker-compose 4](#_Toc199606725)

[2 Описание API 5](#_Toc199606726)

[3 Аутентификация и авторизация 6](#_Toc199606727)

[3.1 JWT-аутентификация 6](#_Toc199606728)

[4 Пользовательский интерфейс 7](#_Toc199606729)

[5 Структура контейнеризации и настройки окружения 8](#_Toc199606730)

[5.1 Архитектура контейнеров 8](#_Toc199606731)

[5.2 Docker-compose 8](#_Toc199606732)

[6 Демонстрация работы приложения в контейнерах 9](#_Toc199606733)

[Приложение А 10](#_Toc199606734)

[Ссылка на GitHub с проектом 10](#_Toc199606735)

[Приложение Б 11](#_Toc199606736)

[Файл docker-compose.yml 11](#_Toc199606737)

[Dockerfile серверной части 11](#_Toc199606738)

Введение

AutoService – приложение для записи на услуги в автосервис. Стек технологий приложения: Python, Django, PostgreSQL. Все компоненты конечной системы контейнеризованны с помощью Docker для обеспечения изоляции окружения, удобства развертывания и масштабирования.

Ссылка на GitHub страницу со всеми файлами проекта можно найти в приложении А.

1. Структура приложения
   1. Компоненты приложения

Приложение разделено на 3 компонента: база данных, серверное приложение (backend).

1. **База данных**
   * Хранит все данные приложения: пользователей, услуги, записи.
2. **Серверное приложение**
   * Обрабатывает запросы клиента;
   * Выполняет внутреннюю логику приложения;
   * Взаимодействует с базой данных;
   * Возвращает данные клиенту.
   1. Docker-compose

Каждый компонент приложения был упакован в контейнер и их совокупность управляется через файл docker-compose.yml.

Особенности работы компонентов, прописанные в docker-compose:

* Строгая последовательность запуска компонентов (база данных, сервер) позволяет избежать ситуаций, когда клиент отправляет запрос на еще не запущенный сервер;
* При остановке контейнера он снова запускается, как только это становится возможным;
* Получение переменных из .env файла через docker-compose повышает безопасность кода.

Код docker-compose файла приведен в приложении Б.

1. Описание API

Структура API, включая методы, URL, параметры, форматы запросов и ответов, описана в таблице 1.

Таблица 1 – Описание API

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Действие | Метод | URL | Параметры | Формат запроса | Формат ответа |
| Регистрация | POST | register/ | username, password, ... | { "username": "user", "password": "1234", ... } | { "token": "...", "user": {...} } |
| Вход | POST | login/ | username, password | { "username": "user", "password": "1234" } | { "access": "...", "refresh": "..." } |
| Просмотр своих записей | GET | my\_appointments/ | JWT токен | — | [ { "id": 1, "service": "...", "date": "...", ... }, ... ] |
| Создание записи | POST | appointments/create/ | JWT токен, service, datetime | { "service": 3, "date": "2025-06-01", "time": "14:30" } | { "id": 10, "status": "pending", "master": "...", ... } |
| Смена пароля | POST | change-password/ | Старый/новый пароль | { "old\_password": "1234", "new\_password": "5678" } | { "detail": "Password changed successfully" } |
| Список всех услуг | GET | services/ | — | — | [ { "id": 1, "name": "Замена масла", ... }, ... ] |
| Выход | POST | logout/ | refresh токен | { "refresh": "..." } | { "detail": "Logout successful" } |

1. Аутентификация и авторизация

Серверное приложение использует механизм JTW-аутентификации для управления доступом пользователей. В качестве механизма авторизации выступает разрешение isAuthenticated позволяющее доступ к определенным эндпоинтам только аутентифицированным пользователям.

* 1. JWT-аутентификация

При входе в систему пользователь передаёт свои учётные данные (логин и пароль) в теле POST-запроса на эндпоинт login/. В случае успешного выполнения запроса сервер возвращает access и refresh токены в теле ответа в формате JSON.

Клиент сохраняет полученные токены и при последующих запросах передаёт access токен в заголовке Authorization, чтобы пройти аутентификацию:

Authorization: Bearer <access\_token>

Если access токен истекает, пользователь может получить новый, используя refresh токен. Access токен действителен в течение 5 минут, а refresh токен — 1 день. Передача токенов через Cookie **не используется** — клиент самостоятельно управляет токенами.

1. Пользовательский интерфейс

Пользовательский интерфейс приложения реализован с помощью HTML и шаблонизаторов и разделен на несколько страниц:

* Страницы входа, регистрации и просмотра услуг.
* Создание и просмотр своих записей, для авторизованных.
* Смена пароля для клиентов.

В клиентской части приложения используется **классическая HTML-навигация**, построенная на Django. Все страницы (регистрация, вход, просмотр услуг, создание записи и т.д.) генерируются на сервере и возвращаются как HTML-документы.

1. Структура контейнеризации и настройки окружения

Все компоненты конечной системы контенеризованы с помощью Docker для обеспечения изоляции окружения, удобства развертывания и масштабирования.

* 1. Архитектура контейнеров

Архитектура контейнеров включает в себя 2 компонента:

* 1. PostgreSQL – контейнер на основе образа последней версии postgres, в котором развернута база данных. Использует volume postgres\_data:/var/lib/postgresql/data для сохранения данных между перезапусками контейнера.
  2. Backend – контейнер содержит Django приложение собранный на основе образа python версии 3.10. Серверное приложение доступно на порте 8000.

Код docker файла приведен в приложении Б.

* 1. Docker-compose

Для управления всеми контейнерами используется файл docker-compose.yml, который описывает конфигурацию всех компонентов и их связи.

1. Демонстрация работы приложения в контейнерах

Упакованное приложение запускается командой docker-compose up, после чего сервер становится доступен по адресу 127.0.0.1:8000, а клиент – 127.0.0.1:8000. На рисунке 1 показана работа контейнеров и приложения в браузере.

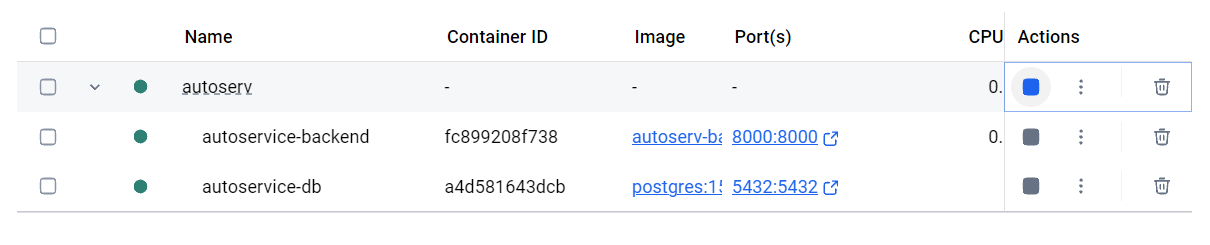


Рисунок 1 – Запущенные контейнеры в приложении Docker Desktop

Приложение А

Ссылка на GitHub с проектом

https://github.com/Tigr195/AutoService

Приложение Б

Файл docker-compose.yml

version: '3.9'

services:

db:

image: postgres:15

container\_name: autoservice-db

env\_file:

- ./.env

volumes:

- postgres\_data:/var/lib/postgresql/data

ports:

- "5432:5432"

backend:

build: AutoService/

env\_file:

- ./.env

container\_name: autoservice-backend

ports:

- "8000:8000"

depends\_on:

- db

volumes:

postgres\_data:

Dockerfile серверной части

FROM python:3.12-slim

ENV PYTHONDONTWRITEBYTECODE 1

ENV PYTHONUNBUFFERED 1

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

CMD ["gunicorn", "AutoService.wsgi:application", "--bind", "0.0.0.0:8000"]