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 лабораторным работам

Дисциплина: «Технологии сетевого программирования»

Выполнили: Шелудько П.В., Ялюхов А.Д.

Группа: 6302-010302D

Самара, 2025

СОДЕРЖАНИЕ

[0 Проектирование приложения 3](#_Toc199710697)

[1 Разработка базы данных 5](#_Toc199710698)

[2 Разработка API 10](#_Toc199710699)

[3 Авторизация 14](#_Toc199710700)

[4 Разработка клиентской части 17](#_Toc199710701)

[5 Финализация приложения и упаковка в Docker 20](#_Toc199710702)

1. Проектирование приложения

**Концепция приложения**

Планируется веб-приложение для онлайн записи на груминг и управление салоном. Пользователи смогут записываться на услуги, просматривать цены, выбирать мастеров, а администраторы – управлять расписанием и клиентами.

*Основные функции:*

Онлайн запись клиентов

Каталог услуг с ценами

Просмотр и выбор мастеов

Личный кабинет администратора (расписание, клиенты, услуги)

**Выбор стека технологий**

*Frontend*

* React.js – библиотека для создания интерфейса.
* React Router – маршрутизация страниц.
* Tailwind CSS – стилизация.
* Axios – отправка запросов на сервер.

*Backend*

* Node.js, Express
* PostgreSQL– база данных.

*Дополнительные инструменты*

* Git + GitHub – контроль версий.
* Postman – тестирование API.
* Swagger (OpenAPI) – документирование API.

**Проектирование архитектуры**

*Взаимодействие компонентов*

Frontend (React) ⟷ Backend (Node.js) ⟷ Database (PostgreSQL)

Схема базы данных ((PostgreSQL)

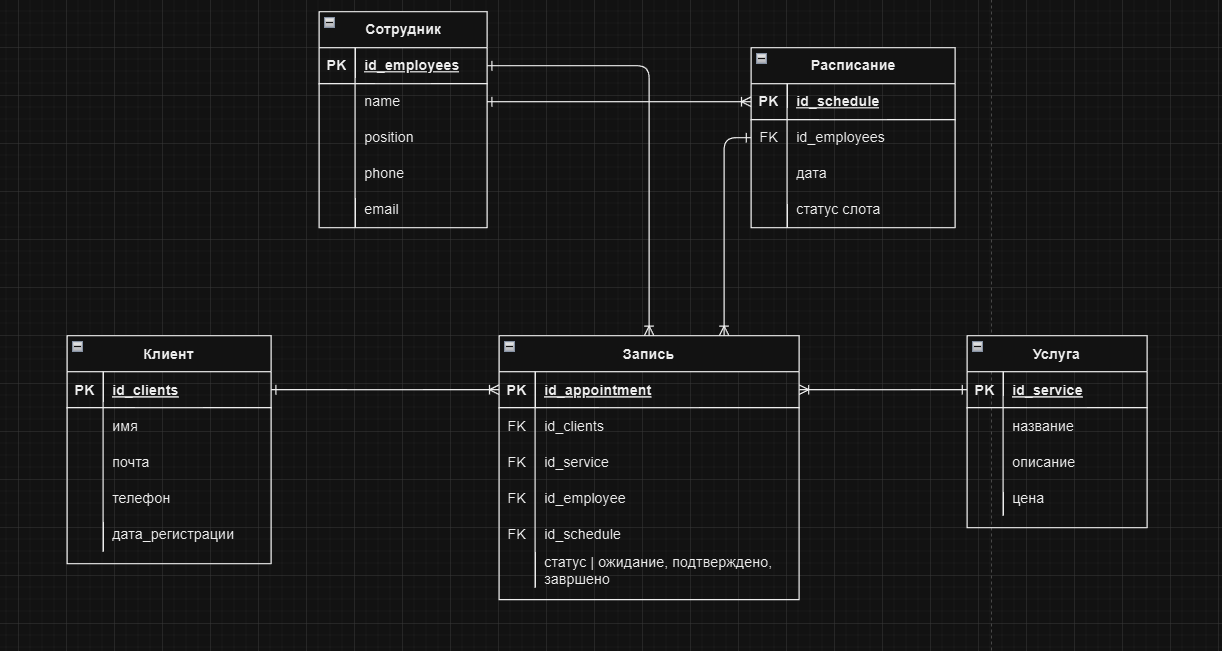
*Будет 3 основные сущности:*

1. User (Пользователь)

2. Appointment (Запись на услугу)

3. Service (Услуга)

**Пример схемы**

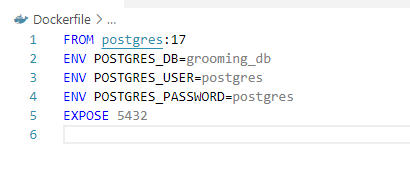


**API (OpenAPI/Swagger)**

|  |  |  |
| --- | --- | --- |
| POST | /auth/register | Регистрация пользователя |
| POST | /auth/login | Вход в систему |
| GET | /users/:id | Получение данных пользователя |
| POST | /appointments | Создание записи на услугу |
| GET | /appointments/:id | Получение информации о записи |
| PUT | /appointments/:id | Обновление статуса записи |
| GET | /services | Получение списка услуг |

1. Разработка базы данных
2. Настройка базы данных:
   1. Развернуть PostgreSQL в Docker;

* Создаем Dockerfile в каталоге проекта



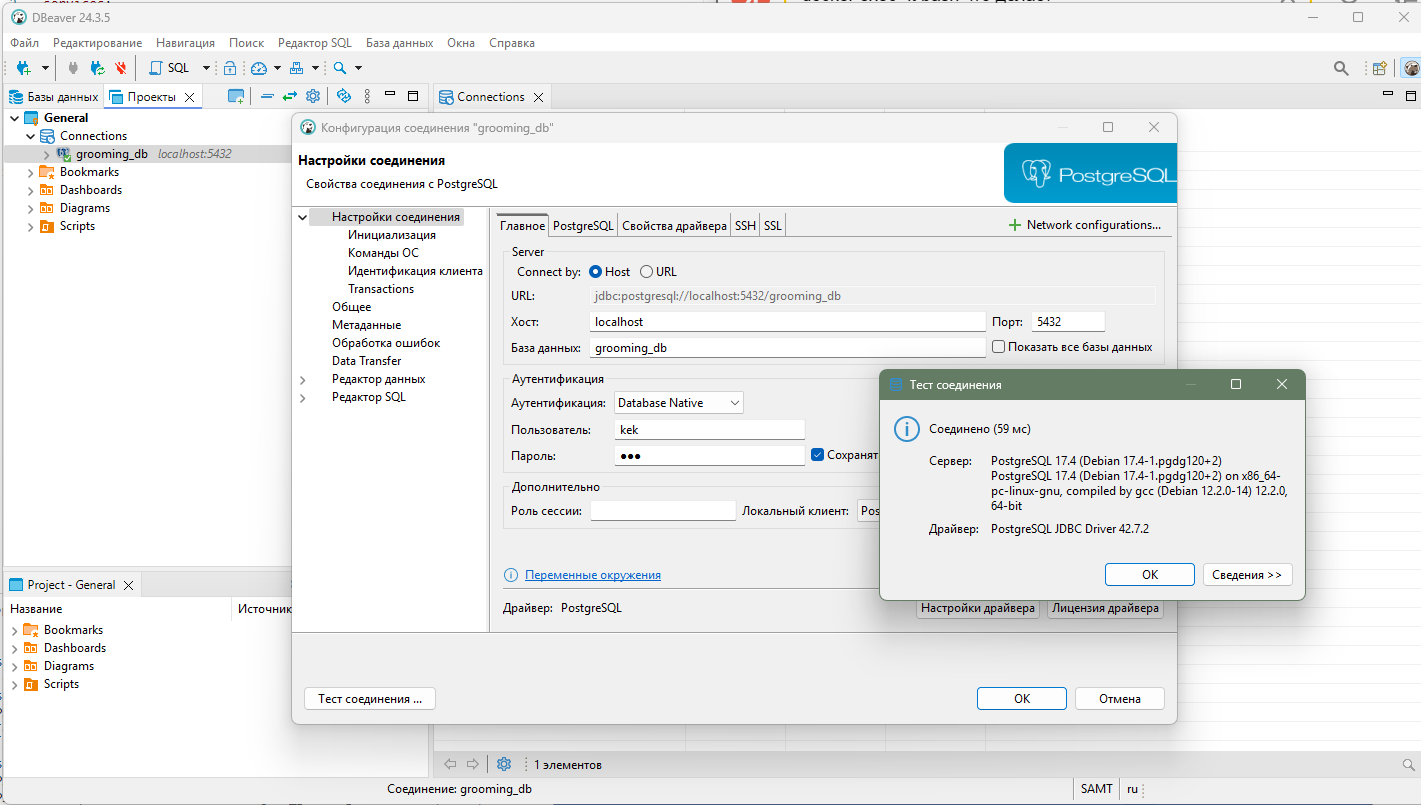
* Создаем контейнер grooming на порте 5432

docker run --name grooming -p 5432:5432 -e POSTGRES\_USER=kek -e POSTGRES\_PASSWORD=kek -e POSTGRES\_DB=grooming\_db -d postgres:17

* PS C:\Users\polin> docker exec -it grooming bash
* root@fdec3c5e34ad:/# pg\_isready -U grooming

/var/run/postgresql:5432 - accepting connections

* root@fdec3c5e34ad:/# psql -U kek -d grooming\_db
  1. Подключиться к БД через DBeaver.



1. **Разработка ORM-моделей**:
   1. Определить сущности и связи между ними;

* Сущности, которые будут в базе данных:

1. Клиент (clients)
2. Сотрудник (employees)
3. Запись (appointments)
4. Услуга (services)
5. Услуги в записи (appointment\_services)

* Схема базы данных (по сущностям):

Один клиент может иметь много записей (1:M).

Один сотрудник может иметь много записей в расписании (1:M).

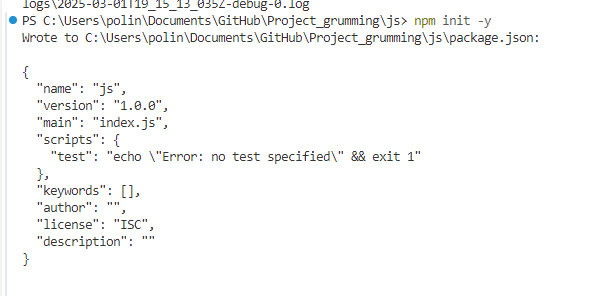
Одна услуга может быть включена в много записей (1:M).

Модель appointment\_services связывает услуги и записи, решая связь M:N.

* Установка Sequelize

Создаём папку проекта и устанавливаем зависимости:

npm init –y



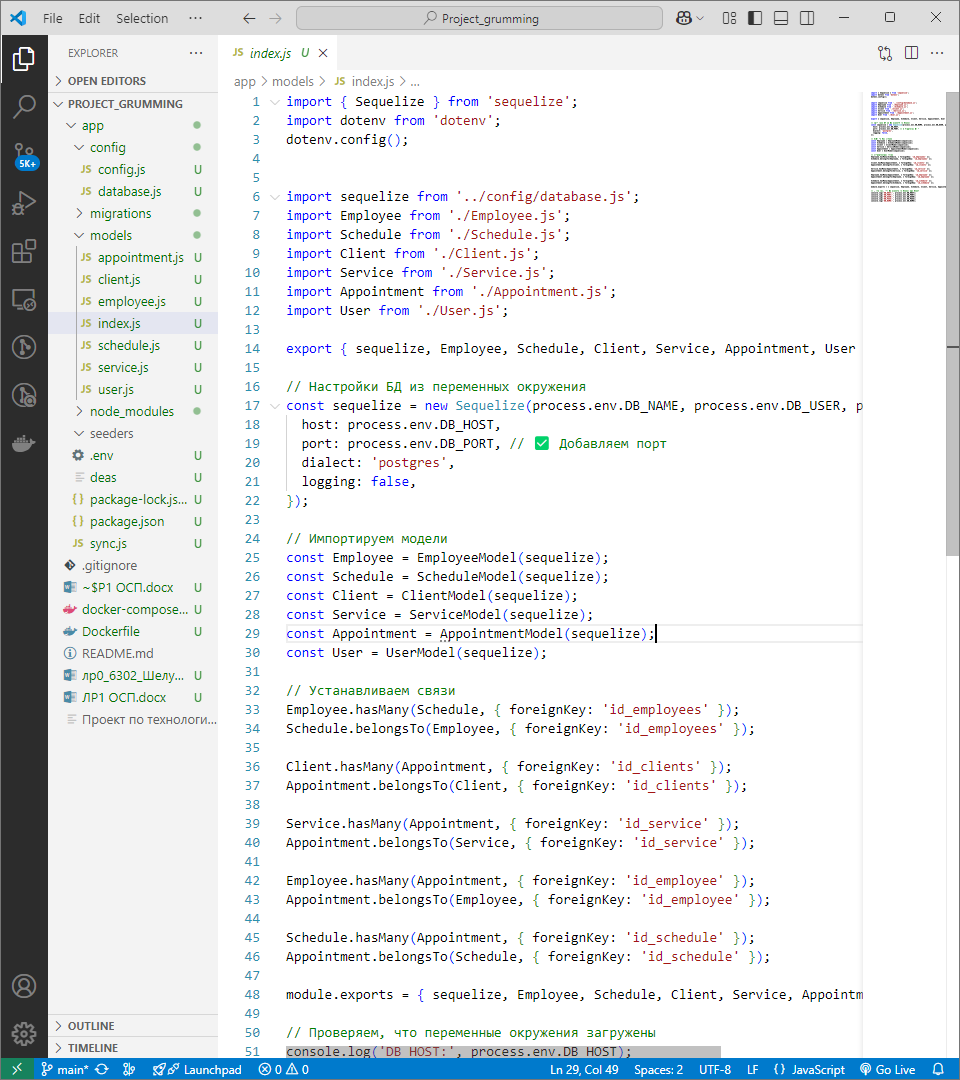
npm install sequelize pg dotenv

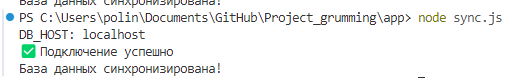
npm install sequelize pg pg-hstore sequelize-cli bcrypt dotenv

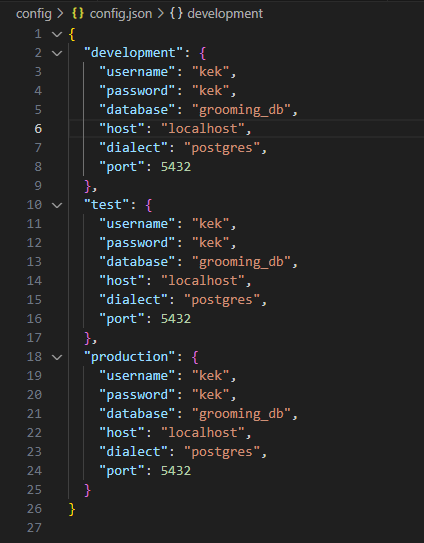
npm install --save-dev sequelize-cli

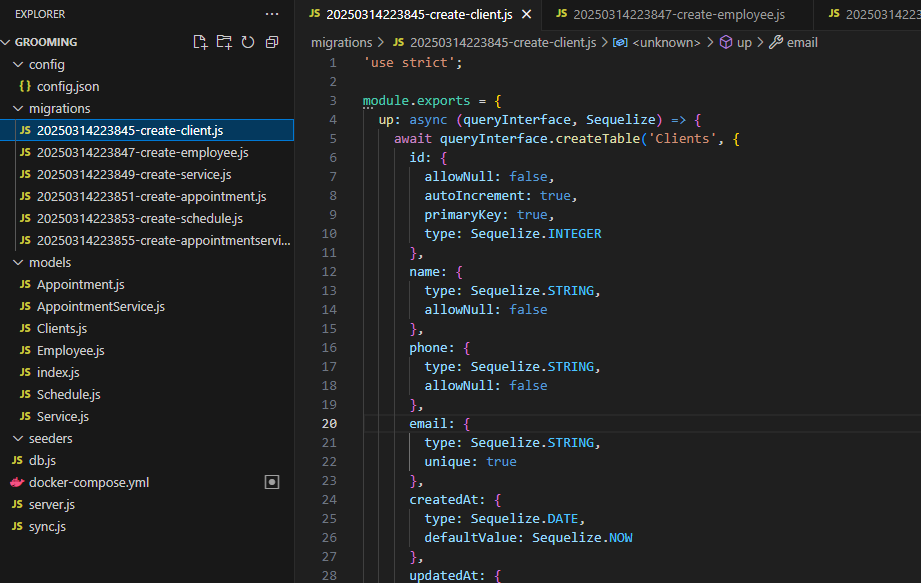
npx sequelize-cli init Инициализация Sequelize

* 1. Определение моделей и связей

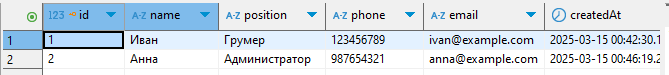


* Синхронизируем базы данных
* 
  1. Настроить миграции.





npx sequelize-cli db:migrate --env development



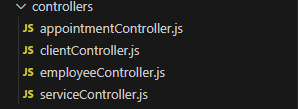
1. Разработка API

Реализован CRUD API для работы с сервисами:

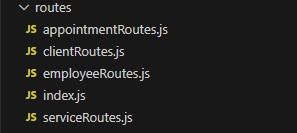
Были добавлены маршруты для выполнения операций создания (POST), получения (GET), обновления (PUT) и удаления (DELETE) сервисов.

Каждая операция взаимодействует с базой данных через модель Service, использующую Sequelize ORM для обработки запросов и работы с PostgreSQL.

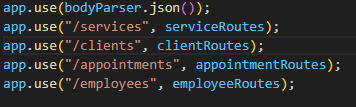
В файле serviceController.js были реализованы функции для обработки запросов: создание нового сервиса, получение всех сервисов, обновление сервиса по ID и удаление сервиса. Контроллер содержит функции для обработки CRUD операций для сервиса.

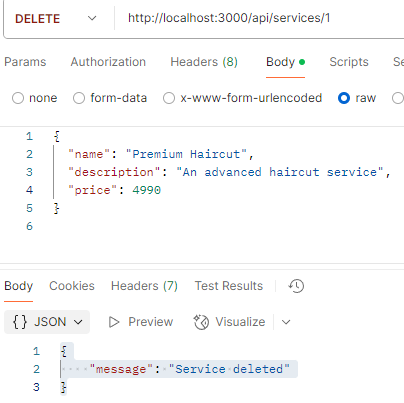
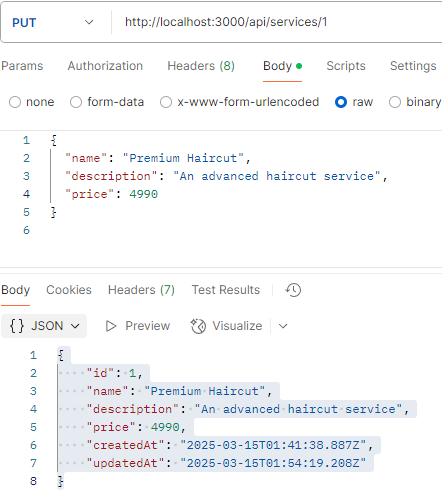
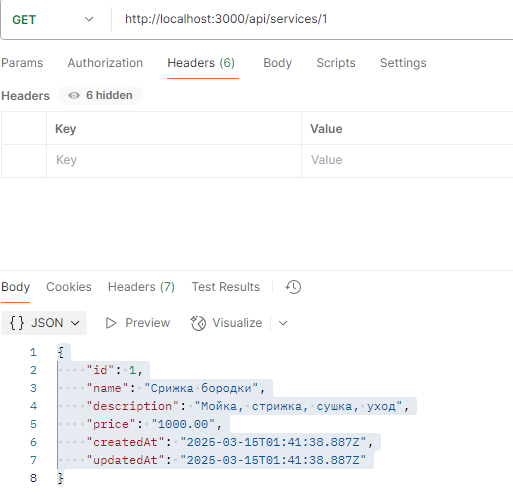
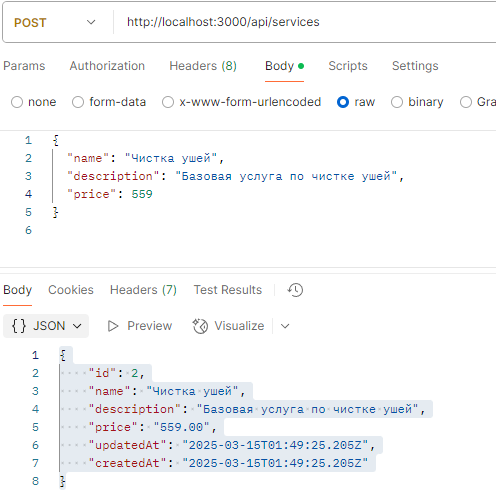


В файле serviceRoutes.js были настроены маршруты, которые обрабатывают соответствующие HTTP-запросы и вызывают методы контроллеров для обработки данных. Маршруты определяют пути для API и связывают их с функциями контроллера.



Основной сервер импортирует и использует маршруты для сервиса.

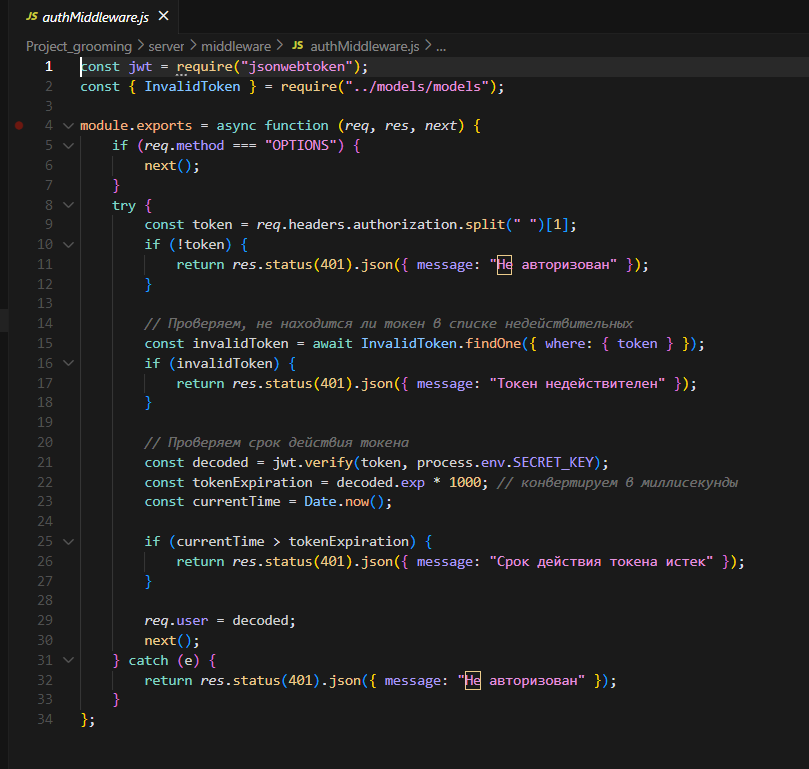


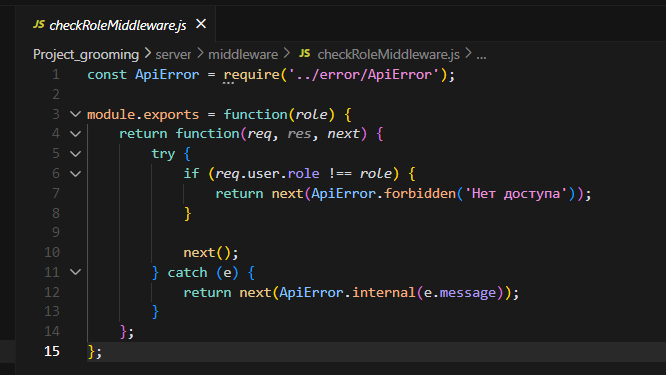


1. Авторизация

Реализована система регистрации и входа пользователей, JWT аутентификация (access и refresh токены).

Middleware для проверки авторизации, роли пользователей (admin, user), защита маршрутов от несанкционированного доступа.







1. Middleware:

* Аутентификация
* Валидация данных
* Обработка ошибок

1. Сервисный слой

* Работа с базой данных
* Генерация токенов
* Валидация данных

1. Разработка клиентской части

Основные функции:

1. **Аутентификация:**

* Регистрация
* Вход
* Выход

1. **Управление услугами:**

* Просмотр услуг
* Фильтрация по категориям
* Создание/редактирование (админ)

1. **Записи:**

* Создание записи
* Просмотр своих записей
* Отмена записи

1. **Админ-панель:**

* Управление услугами
* Просмотр всех записей
* Управление расписанием

Структура:

1. Общедоступные страницы:

- Home.js - главная страница

- Shop.js - страница с услугами

- Auth.js - страница авторизации/регистрации

1. Защищенные страницы

- Admin.js - админ-панель

- UserAppointments.js - страница с записями пользователя

1. Дополнительные страницы

- AdminAppointments.js - страница управления записями для админа

- Services.js - страница услуг

1. Компоненты

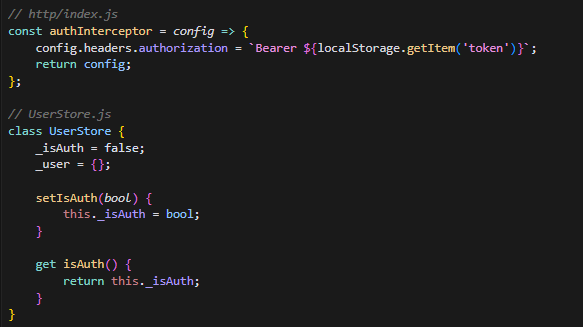
- Services.js - компонент для отображения услуг

- CreateAppointment.js - модальное окно для создания записи

- ServiceItem.js - карточка услуги

- TypeBar.js - фильтр по категориям услуг

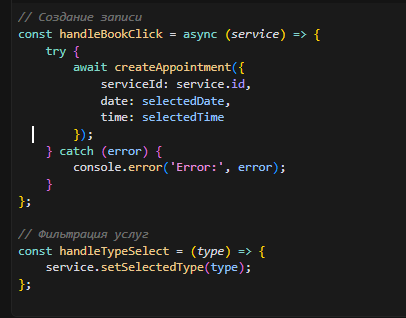
Аутентификация:



Рендеринг данных:



Передача данных:



1. Финализация приложения и упаковка в Docker

Проект использует Docker и Docker Compose для упаковки и развертывания фронтенд- и бэкенд-частей приложения, а также базы данных MongoDB. Контейнеризация позволяет легко запускать проект на любом сервере или машине, где установлен Docker.

Структура контейнеров

Фронтенд

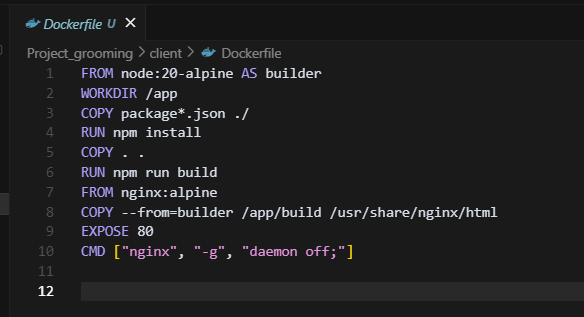
Dockerfile построен на основе node:20-alpine:

• Производит сборку React-приложения (npm run build);

• Далее артефакты сборки копируются в образ nginx:alpine;

• Nginx используется как веб-сервер для отдачи статики;

• Порт 80 контейнера проброшен на 3000 хоста:



Бэкенд

• Образ backend-app представляет сервер на Node.js, использующий Sequelize для подключения к PostgreSQL.

• Контейнер настраивается через переменные окружения: environment:

PORT: 5000

DB\_NAME: online\_grooming

DB\_USER: postgres

DB\_PASSWORD: root

DB\_HOST: postgresdb

DB\_PORT: 5432

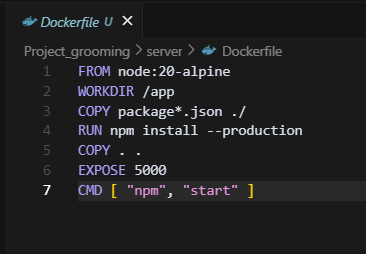
SECRET\_KEY: 12345

REFRESH\_SECRET\_KEY: 12345.

• Порт 5000 проброшен наружу.

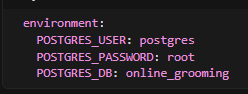
* Монтируется том для статических файлов:



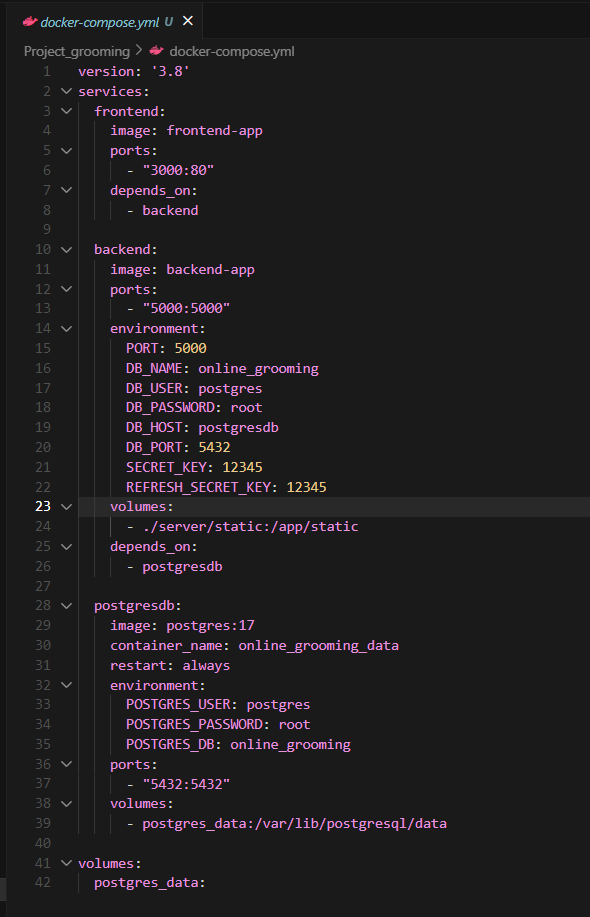


PostgreSQL

* + - * Используется официальный образ postgres:17;
      * Данные сохраняются в postgres\_data, что обеспечивает сохранность между перезапусками;
      * Порт 5432 проброшен на хост;
    - Настройка через переменные окружения:.



Docker-compose.yml



Контейнеры в докере

