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: Зяббарова К.З,

Буянов С.А.

Группа: 6302-010302D

Самара, 2025

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 2](#_Toc180655805)

[1 Архитектура приложения 3](#_Toc1470648144)

[2 Описание API в формате таблицы 4](#_Toc827726900)

[3 Описание работы с бд 6](#_Toc1773242880)

[4 Crud методы 8](#_Toc431700310)

[5 Тестирование API 10](#_Toc165489796)

[6 Аутентификация и авторизация 13](#_Toc1071979259)

[7 Структура интерфейса 14](#_Toc1937518673)

[8 Примеры рендеринга данных и передачи параметров 15](#_Toc2989023)

[9 Описание структуры контейнеризации и настройки окружения 17](#_Toc398501059)

[10 Скриншоты работающего приложения в контейнерах 19](#_Toc1079226325)

[Приложение А 20](#_Toc273049900)

# ВВЕДЕНИЕ

Cake room – приложение для кондитерской фабрики. Проект реализован с использованием Node.js, Express, TypeScript, в качестве базы данных – нереляционная MongoDB. Он предоставляет RESTful API для регистрации, авторизации, получения данных пользователей, товаров, вакансий, а также изменение товаров и вакансий. Приложение контейнеризировано с помощью Docker, что обеспечивает лёгкий запуск и масштабирование. Основное назначение — демонстрация навыков бэкенд и фронтенд разработки, настройки окружения и контейнеризации.

# 1 Архитектура приложения

**Общая структура**

Приложение реализовано по принципу **клиент-серверной архитектуры** и состоит из трёх основных компонентов, разнесённых по контейнерам:

1. Frontend (React + Nginx)
2. Backend (Node.js + Express + Mongoose)
3. База данных MongoDB

Все части связаны между собой через Docker Compose.

**Компоненты**

**Frontend (React + Nginx)**

* Сборка React-приложения происходит на этапе билда контейнера.
* Для раздачи фронтенд-приложения используется минимальный образ nginx:alpine, в который копируются собранные статические файлы React.
* Связь с backend осуществляется через API-запросы на http://localhost:3004.

**Backend (Node.js + Express)**

* REST API реализовано с помощью Express.js.
* Аутентификация построена с использованием JWT (два токена: access и refresh).
* Работа с базой — через ORM Mongoose.
* Все переменные конфигурации задаются через .env или docker-compose.yml.

**MongoDB**

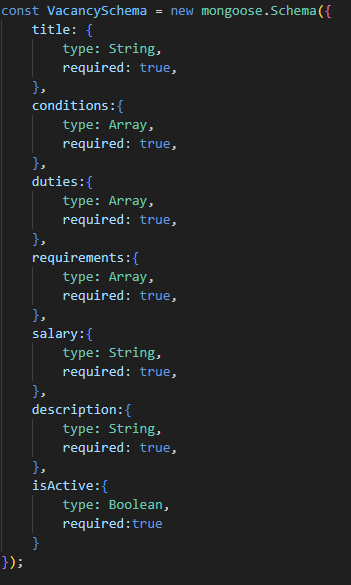
* Хранит данные в коллекциях.
* Используется как основное хранилище данных.
* Связь через localhost:27017

# 2 Описание API в формате таблицы

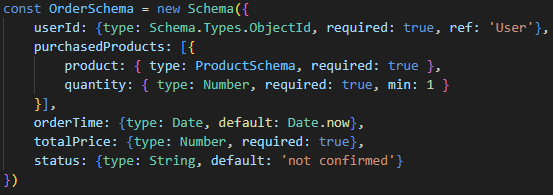
|  |  |  |  |
| --- | --- | --- | --- |
| Действие | Метод | URL | Описание |
| Получить все продукты | GET | /products | Возвращает список всех продуктов |
| Получить количество продуктов | GET | /products/sum | Возвращает общее количество продуктов |
| Получить продукты из каталога | GET | /products/catalog | Возвращает продукты, отфильтрованные по каталогу |
| Получить продукт по ID | GET | /products/:id | Возвращает продукт с заданным ID |
| Добавить продукт | POST | /admin/add/product | Добавляет новый продукт |
| Обновить продукт | POST | /admin/update/product | Обновляет информацию о продукте |
| Добавить вакансию | POST | /admin/add/vacancy | Добавляет новую вакансию |
| Обновить вакансию | POST | /admin/update/vacancy | Обновляет вакансию |
| Получить все вакансии | GET | /vacancies | Возвращает список всех вакансий |
| Получить вакансию по ID | GET | /vacancies/:id | Возвращает вакансию с заданным ID |
| Регистрация | POST | /registration | Регистрирует нового пользователя |
| Логин пользователя | POST | /login | Авторизует пользователя |
| Выход пользователя | POST | /logout | Производит выход пользователя |
| Обновление токена | GET | /refresh | Обновляет токен доступа |
| Получить всех пользователей | GET | /users | Возвращает список всех пользователей |
| Оформить заказ | POST | /orders | Создает новый заказ |
| Получение заказа по ID | GET | /orders/:id | Возвращает заказ с заданным ID |
| Получить все заказы | GET | /orders | Возвращает список всех заказов |
| Изменение статуса заказа | POST | /orders/status | Изменяет статус конкретного заказа |

# 3 Описание работы с бд

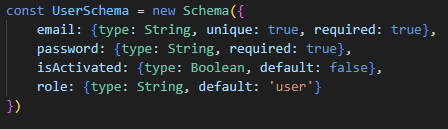
Пример - модель вакансии:



Модель заказа:

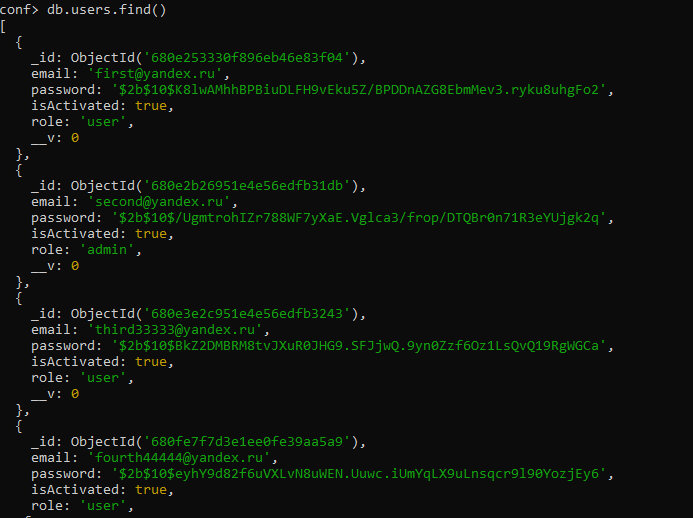


Модель пользователя:

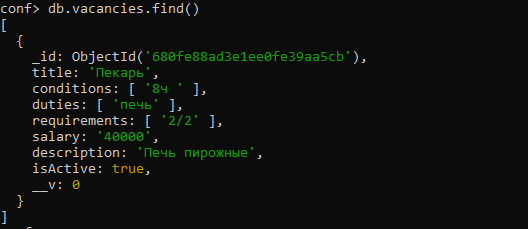


Список таблиц: orders, products, tokens, users, vacancies

Хранение пользователей в бд:



Хранение вакансий в бд:



# 4 Crud методы

Для каждой модели были реализованы следующие CRUD-методы:

Create: Создание новых записей в базе данных.

Read: Получение информации о существующих записях.

Update: Обновление информации о существующих записях.

Модели и методы:

1. Пользователь (User)

* Create: регистрация пользователя, активация аккаунта
* Read: получение всех пользователей
* Update: смена роли пользователя

1. Продукт (Product)

* Read: получение всех продуктов, получение продукта по ID, получение продукта из каталога, получение количества продуктов

1. Заказ (Order)

* Create: создание нового заказа
* Read: получение всех заказов, получение заказа по ID
* Update: изменение статуса заказа

1. Вакансия (Vacancy)

* Read: получение всех вакансий, получение вакансии по ID

1. Админ (Admin)

* Update: изменение продукта
* Update: изменение вакансии
* Create: добавление продукта
* Create: добавление вакансии

Настройка маршрутов и обработка запросов

Маршруты для работы с API настроены следующим образом:

* /users: для работы с пользователями
* /products: для работы с продуктами
* /orders: для работы с заказами
* /vacancies: для работы с вакансиями

Каждый маршрут обрабатывает соответствующие запросы, используя контроллеры, которые взаимодействуют с сервисами и моделями.

# 5 Тестирование API

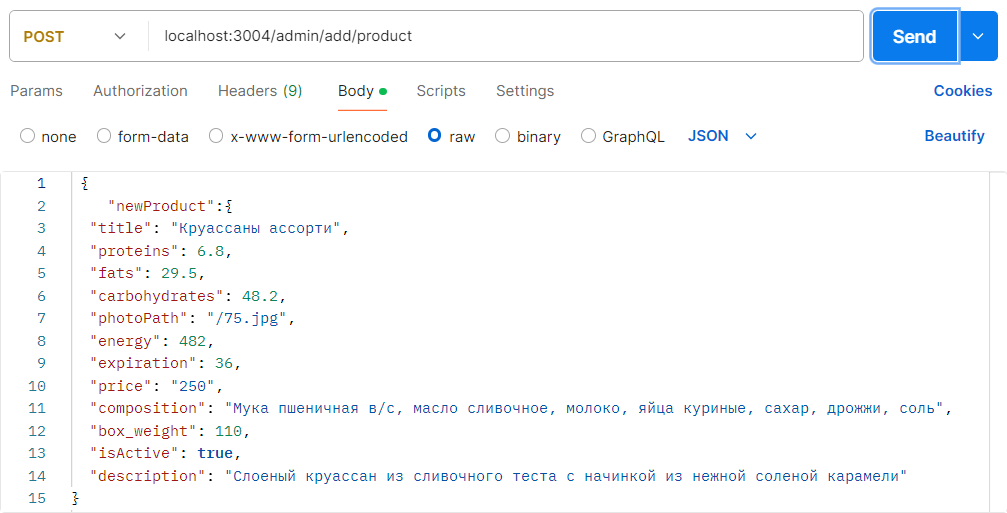
1. Использование Postman

Для тестирования API использовался Postman. Каждый маршрут был протестирован с различными методами (GET, POST, и т.д.), а также с правильными и неправильными данными.

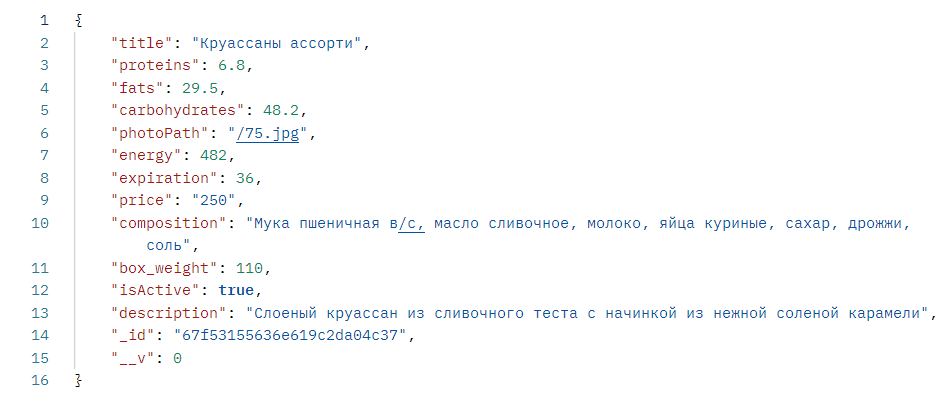
1. Примеры запросов и ответов

* Создание нового продукта

Запрос:

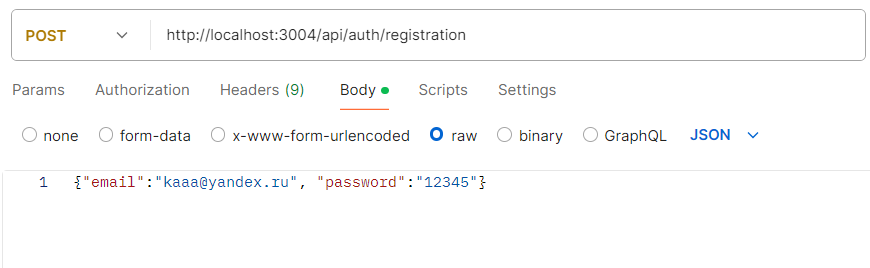


Ответ:



* Регистрация пользователя

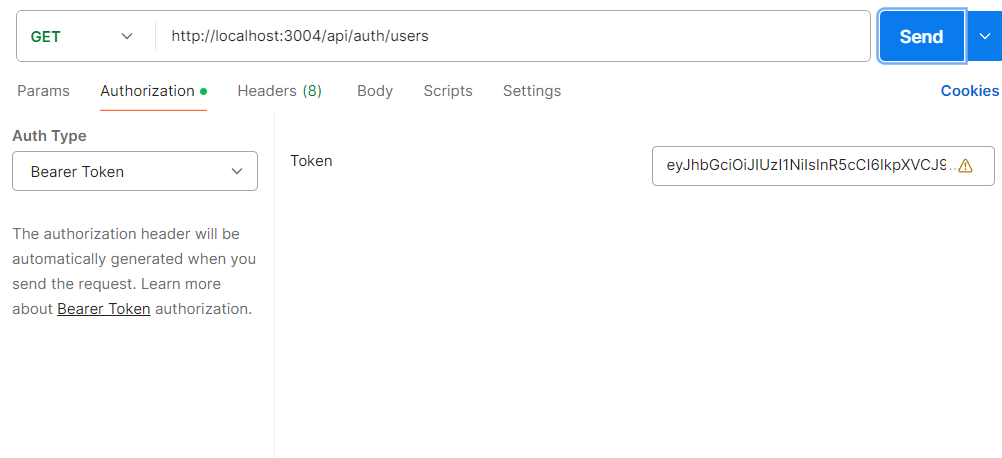
Запрос:

 Ответ:

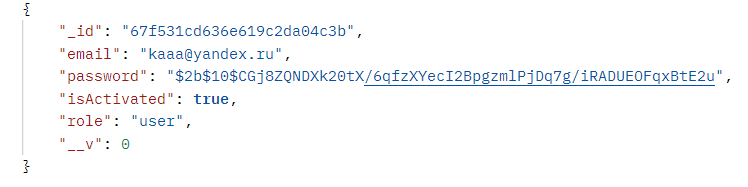


* Получение всех пользователей

Запрос:

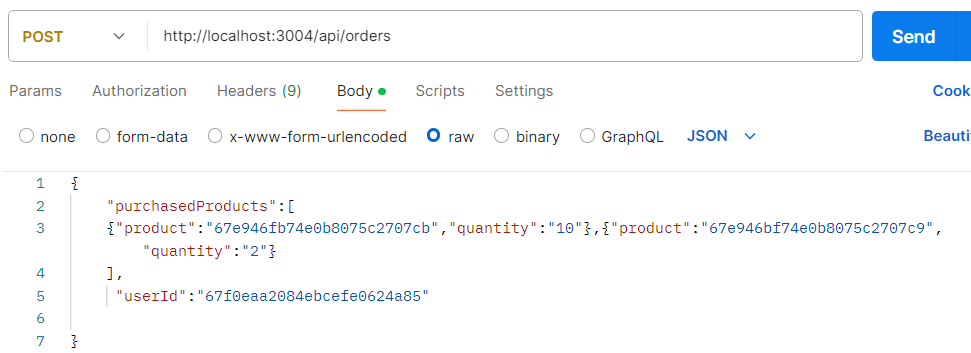


Ответ:

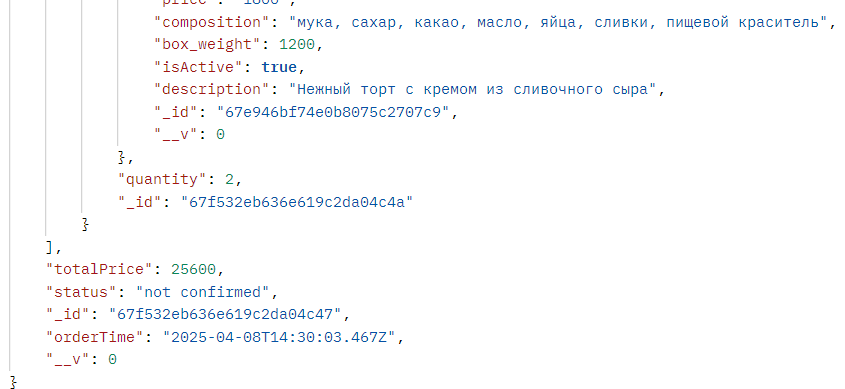
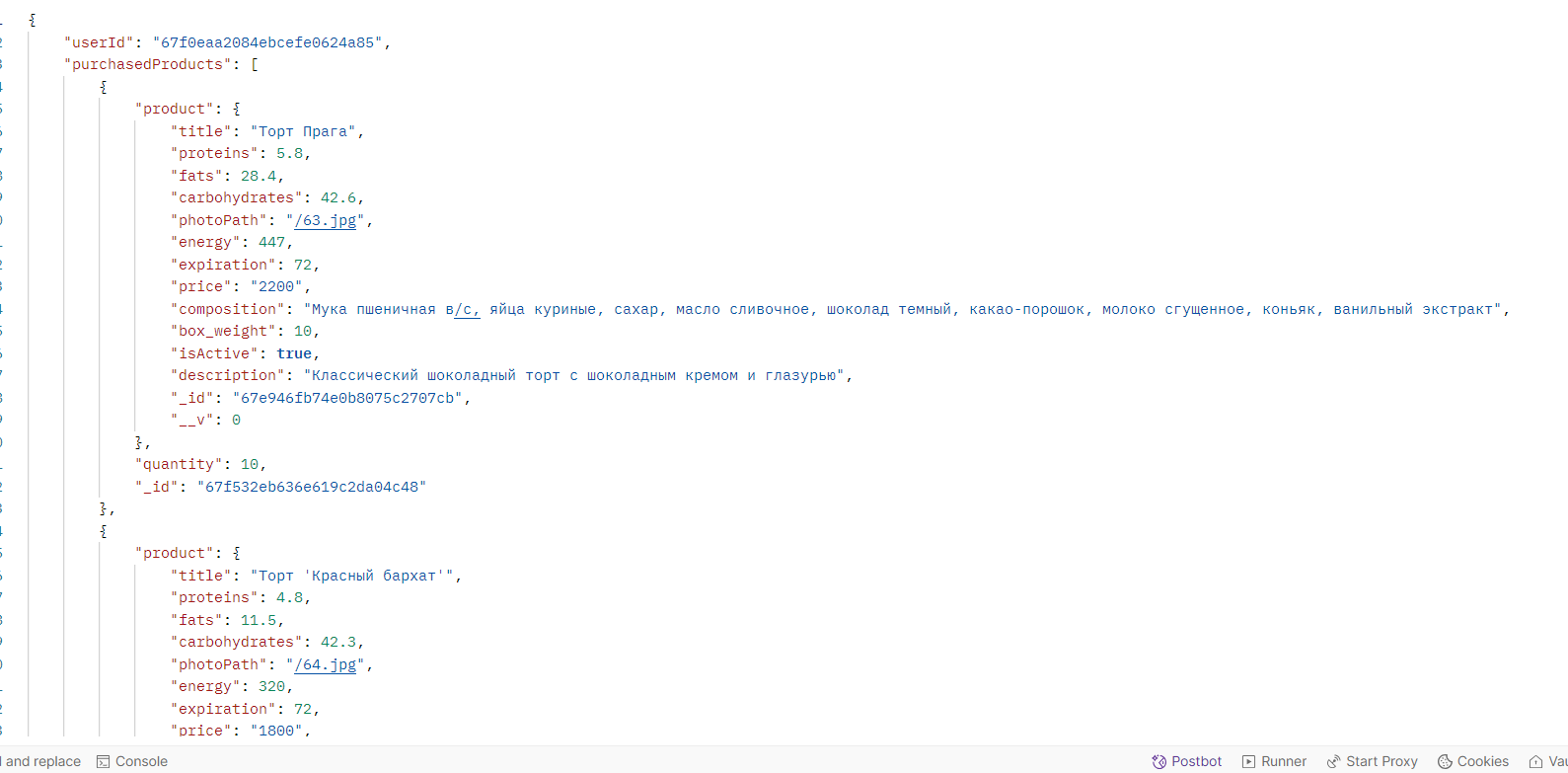


* Создание нового заказа

Запрос:



Ответ:



# 6 Аутентификация и 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.

Особенности реализации:

1. Middleware:

* Аутентификация
* Валидация данных
* Обработка ошибок
* CORS

1. Сервисный слой

* Работа с базой данных
* Генерация токенов
* Валидация данных

1. DTO (Data Transfer Objects)

* UserDto для безопасной передачи данных пользователя
* Структурирование данных для клиента

1. Технологии и инструменты

* Node.js
* Express.js
* MongoDB
* Mongoose
* JWT
* Dotenv для конфигурации
* Cookie-parser
* CORS

# 7 Структура интерфейса

Проект построен по принципам компонентного фронтенда:

**footer/, header/, menu/** — UI-обёртки.

**store/** — Redux-хранилище, содержит adminSlice, basketSlice и др.

**services/** — содержит функции для взаимодействия с сервером через axios (AdminService, UserService, AuthService)

**dashboard/** — layout приложения.

**modals/** — содержит модальные окна, то есть всплывающие интерфейсные компоненты, которые используются для отображения важной информации, ошибок и т.п.

**pages/** — здесь все основные страницы, которые подгружаются по маршрутам (MainPage, BasketPage, CatalogPage, AdminPage и т.д.).

**App.tsx**/ — является главной точкой входа в интерфейс приложения, где:

настраивается маршрутизация (на основе react-router-dom), подключается проверка авторизации пользователя, управляется отображение спиннера при загрузке, рендерятся основные страницы и защищённые админ-маршруты.

# 8 Примеры рендеринга данных и передачи параметров

1. **Получение параметра из URL**



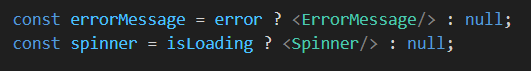
Используется react-router-dom, чтобы получить id продукта из URL

1. **Получение данных с сервера**



RTK Query делает запрос к серверу по ID.  
product — полученные данные, error и isLoading — статус запроса.

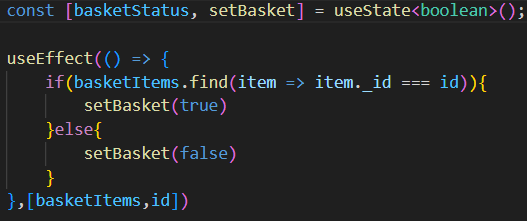
1. **Условный рендеринг состояния запроса**



Компонент показывает:

* <Spinner /> — пока идёт загрузка;
* <ErrorMessage /> — если произошла ошибка.

1. **Состояние товара в корзине**



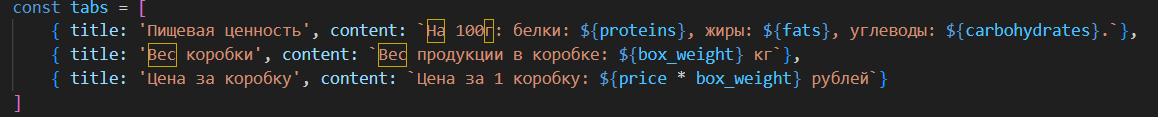
Сравнивает текущий product.\_id с теми, что есть в корзине. Если товар уже добавлен — кнопка "Добавить" блокируется.

1. **Добавление товара в корзину**



При клике товар отправляется в Redux store через dispatch(addItem(product)).

1. **Рендеринг вкладок (табы)**



Переключаются при клике, показывая разные данные, связанные с товаром.

1. **Итоговый интерфейс**

Пользователь:

* Переходит по ссылке /catalog/:id;
* Получает карточку товара по ID;
* Видит спиннер/ошибку/данные;
* Может добавить товар в корзину (если ещё не добавлен);
* Переключает вкладки с деталями.

# 9 Описание структуры контейнеризации и настройки окружения

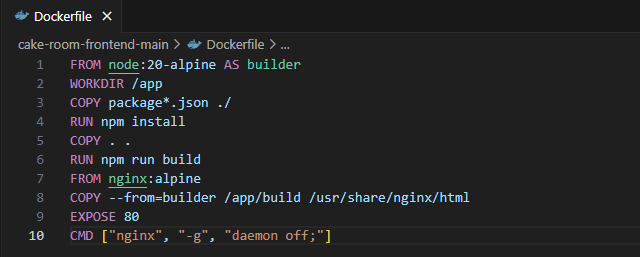
Проект использует Docker и Docker Compose для упаковки и развертывания фронтенд- и бэкенд-частей приложения, а также базы данных MongoDB. Контейнеризация позволяет легко запускать проект на любом сервере или машине, где установлен Docker.

**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:



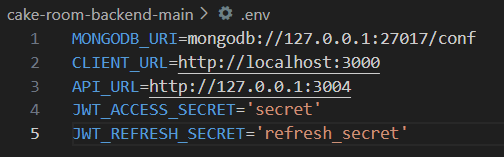
**Бэкенд**

* Образ my-cakeroom-backend представляет сервер на Node.js, использующий Mongoose для подключения к MongoDB.
* Контейнер настраивается через переменные окружения: MONGODB\_URI, CLIENT\_URL, API\_URL, JWT.
* Порт 3004 проброшен наружу.

**MongoDB**

* Используется официальный образ mongo:latest;
* Данные сохраняются в mongodb\_data, что обеспечивает сохранность между перезапусками;
* Порт 27017 проброшен на хост.

**Переменные окружения .env**



# 10 Скриншоты работающего приложения в контейнерах

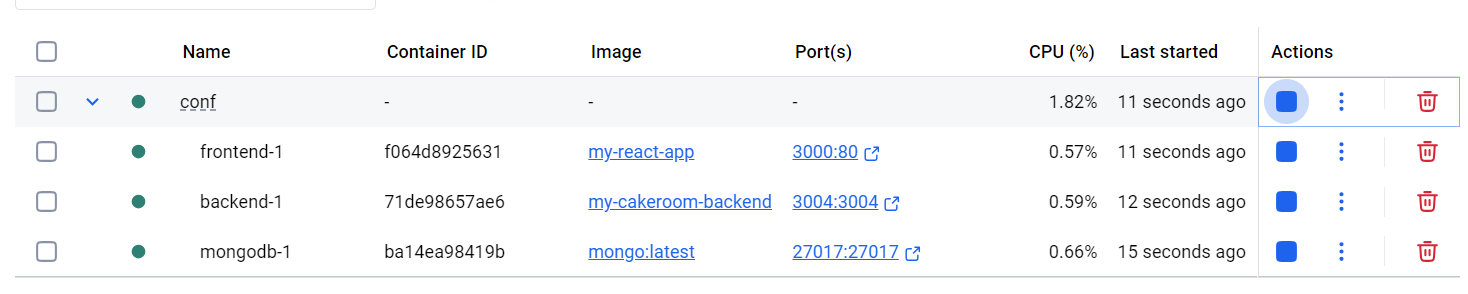


Рисунок 1 – Запущены все контейнеры в Docker Desktop

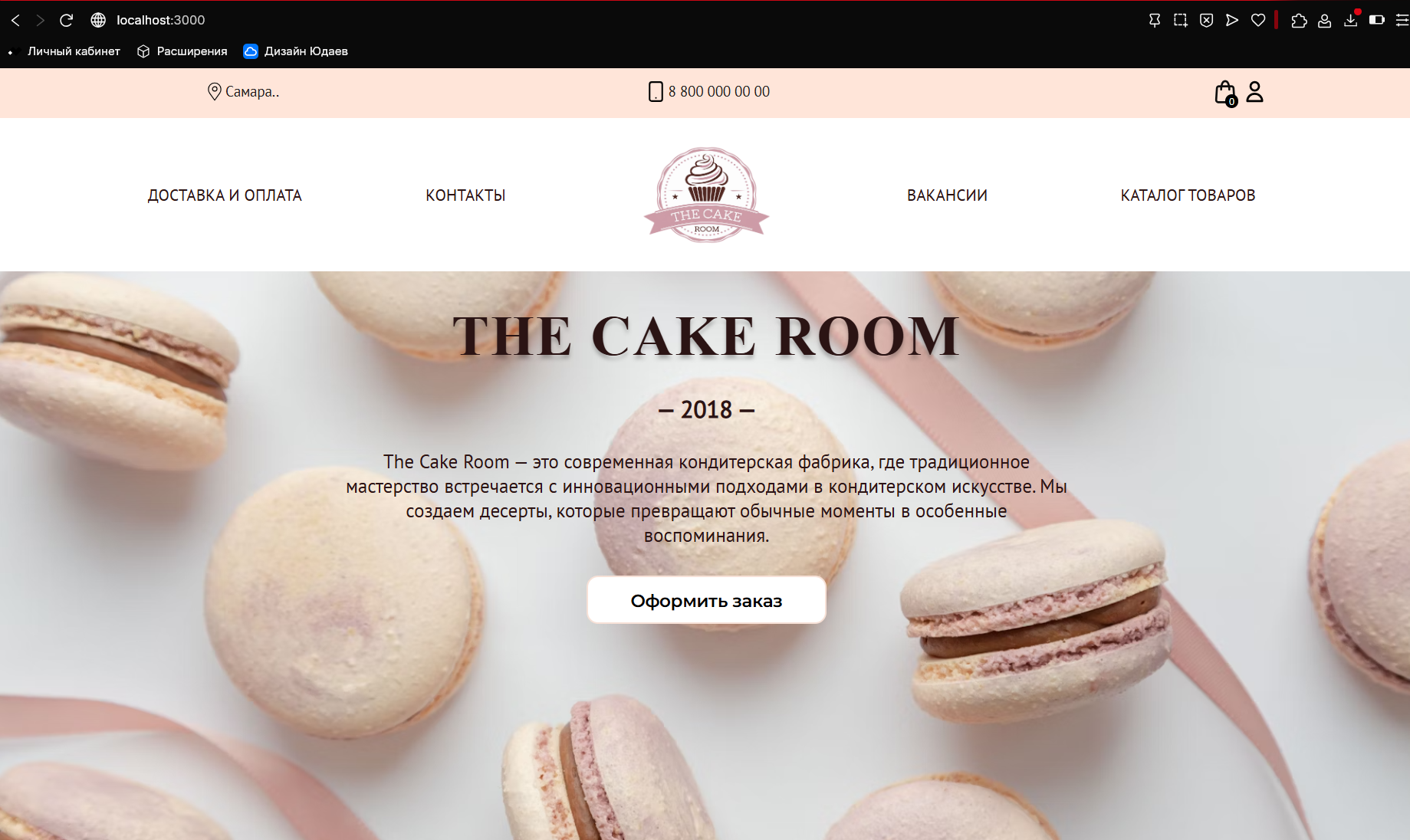


Рисунок 2 – Открыта страница на localhost:3000

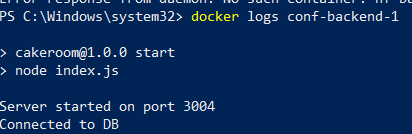


Рисунок 3 – Показывает, что backend успешно запустился и подключился к MongoDB

# Приложение А

Весь код проекта доступен по ссылке:

<https://github.com/sbkz/ssausbkz/tree/main/project>