Серверная часть (Node.js, Express, PostgreSQL)

**2024**

Adam

BNPN

27.12.2024



Оглавление

[Серверная часть (Node.js, Express, PostgreSQL) 2](#_Toc186204141)

[1. Структура проекта 2](#_Toc186204142)

[2. Описание структуры проекта, основные разделы и их описание 3](#_Toc186204143)

[3. Документация проекта (README.md) 5](#_Toc186204144)

[Markdown 5](#_Toc186204145)

[4. my-shop/server/package.json 6](#_Toc186204146)

[5. my-shop/server/Dockerfile 8](#_Toc186204147)

[6. my-shop/server/docker-compose.yml 9](#_Toc186204148)

[7. my-shop/server/init.sql 10](#_Toc186204149)

[8. my-shop/server/src/server.js 12](#_Toc186204150)

[9. my-shop/server/src/routes/authRoutes.js 13](#_Toc186204151)

[10. my-shop/server/src/routes/cartRoutes.js 13](#_Toc186204152)

[11. my-shop/server/src/routes/productRoutes.js 14](#_Toc186204153)

[12. my-shop/server/src/controllers/authController.js 15](#_Toc186204154)

[13. my-shop/server/src/controllers/productController.js 16](#_Toc186204155)

[14. my-shop/server/src/controllers/cartController.js 19](#_Toc186204156)

[15. my-shop/server/src/models/db.js 20](#_Toc186204157)

[16. my-shop/server/src/models/user.js 21](#_Toc186204158)

[17. my-shop/server/src/models/product.js 22](#_Toc186204159)

[18. my-shop/server/src/models/category.js 22](#_Toc186204160)

[19. my-shop/server/src/models/cart.js 23](#_Toc186204161)

[20. my-shop/server/src/middleware/authMiddleware.js 24](#_Toc186204162)

[21. my-shop/server/src/.env 25](#_Toc186204163)

# Серверная часть (Node.js, Express, PostgreSQL)

# Структура проекта

my-shop/

├── client/

│ └── (файлы клиентского приложения, например на React)

├── server/

│ ├── src/

│ │ ├── controllers/

│ │ │ ├── authController.js

│ │ │ ├── productController.js

│ │ │ └── cartController.js

│ │ ├── models/

│ │ │ ├── db.js

│ │ │ ├── user.js

│ │ │ ├── product.js

│ │ │ ├── category.js

│ │ │ └── cart.js

│ │ ├── routes/

│ │ │ ├── authRoutes.js

│ │ │ ├── productRoutes.js

│ │ │ └── cartRoutes.js

│ │ ├── middleware/

│ │ │ └── authMiddleware.js

│ │ ├── .env

│ │ ├── server.js

│ ├── Dockerfile

│ ├── docker-compose.yml

│ └── init.sql

└── README.md

# Описание структуры проекта, основные разделы и их описание

## 

Вся структура проекта уже была предоставлена выше.

 **client/** - здесь будет находиться ваш клиентский код (например, на React или другом фреймворке).

 **server/** - серверное приложение на Node.js с использованием Express и PostgreSQL.

 **src/** - основная логика приложения.

 **controllers/** - обработка логики маршрутов (например, авторизация, работа с продуктами и корзиной).

 **models/** - взаимодействие с базой данных для различных сущностей (пользователи, продукты, категории, корзины).

 **routes/** - определения маршрутов API.

 **middleware/** - промежуточные функции, такие как проверка авторизации.

 **.env** - файл с переменными окружения, такими как учетные данные базы данных и JWT секрет.

 **server.js** - главный файл, где инициализируется сервер и маршруты.

 **Dockerfile** - файл для построения Docker-образа приложения.

 **docker-compose.yml** - для оркестрации нескольких контейнеров (например, приложение и база данных).

 **init.sql** - SQL-скрипты для создания необходимых таблиц в базе данных.

 **README.md** - документация по проекту, включая инструкции по установке и использованию.

**my-shop/server/package.json** :

Файл package.json является неотъемлемой частью любого Node.js приложения, включая наш серверный проект. Он выполняет несколько важных функций:

Назначение package.json:

Управление зависимостями: В нем перечислены все пакеты (зависимости), которые необходимы для работы вашего приложения. Это позволяет легко установить их с помощью команды npm install.

Метаданные проекта: Файл содержит информацию о проекте, такую как имя, версия, лицензия и описание. Эта информация может быть полезной для разработчиков или для публикации проекта.

Скрипты: В package.json можно определить различные команды, которые могут быть запущены с помощью npm. Например, команды для тестирования, сборки или запуска приложения.

Настройка: Позволяет настраивать поведение приложения, например, задавать переменные окружения.

Заключение:   
Файл package.json является жизненно важным для любого Node.js-приложения. Он значительно упрощает управление зависимостями и конфигурацией проекта, обеспечивая лучший путь для взаимодействия с вашим кодом как для вас, так и для других разработчиков. Создание этого файла в рамках вашего проекта будет полезно и необходимо для его дальнейшего развития и развертывания.

# Документация проекта (README.md)

# 

# Markdown

# Интернет-магазин

## Описание проекта

Этот проект представляет собой интернет-магазин, реализованный с использованием Node.js, Express и PostgreSQL. Он включает в себя авторизацию пользователей, CRUD операции для продуктов и обработку корзины.

## Структура проекта

- \*\*client/\*\* - клиентская часть приложения (может быть React или другой фреймворк)

- \*\*server/\*\* - серверная часть приложения

- \*\*src/\*\* - основная логика приложения

- \*\*routes/\*\* - определение маршрутов API

- \*\*controllers/\*\* - логика обработки запросов

- \*\*models/\*\* - модели базы данных

- \*\*.env\*\* - файл переменных окружения

- \*\*Dockerfile\*\* - настройки для создания Docker-образа

- \*\*docker-compose.yml\*\* - настройки для запуска приложения

- \*\*init.sql\*\* - скрипты для инициализации базы данных

## Установка и запуск

1. Склонируйте репозиторий:

bash   
git clone

cd my-shop/server  
  
markdown  
  
2. Убедитесь, что у вас установлен Docker и Docker Compose.

1. Запустите приложение:

bash

docker-compose up –build

markdown

4. После успешного запуска сервер будет доступен по адресу `http://localhost:5000`.

## API

- `POST /auth/register` — Регистрация пользователя

- `POST /auth/login` — Авторизация пользователя

- `GET /products` — Получить список всех товаров

- `GET /products/:id` — Получить товар по ID

- `POST /products` — Создать продукт

- `PUT /products/:id` — Обновить продукт по ID

- `DELETE /products/:id` — Удалить продукт по ID

- `GET /cart?user\_id=1` — Получить корзину пользователя

- `PUT /cart` — Добавить продукт в корзину

- `DELETE /cart` — Удалить продукт из корзины

## Комментарии

Этот проект нуждается в дальнейшей разработке, включая управление сессиями, безопасное хранение паролей и обработку ошибок. Ознакомьтесь с документацией по Express и PostgreSQL для дальнейшего изучения.

# my-shop/server/package.json

Json

{

"name": "my-shop",

"version": "1.0.0",

"description": "Online shop application",

"main": "server.js",

"scripts": {

"start": "node server.js",

"dev": "nodemon server.js",

"test": "echo \"No tests specified\" && exit 0"

},

"keywords": [

"ecommerce",

"shop",

"express",

"node"

],

"author": "Your Name",

"license": "ISC",

"dependencies": {

"express": "^4.17.1",

"jsonwebtoken": "^8.5.1",

"pg": "^8.6.0",

"dotenv": "^8.2.0"

},

"devDependencies": {

"nodemon": "^2.0.4"

}

}

Пояснение отдельных полей:

name: Название вашего проекта, в данном случае my-shop.

version: Версия вашего приложения, которую можно обновлять при каждом изменении.

description: Краткое описание функциональности.

main: Главный файл приложения (например, серверный файл).

scripts: Определяет команды, которые могут быть запущены в терминале:

start: Запускает приложение.

dev: Запускает приложение с помощью nodemon, который автоматически перезапускает сервер при изменениях (полезно для разработки).

test: Команда для тестирования, здесь просто выводит сообщение о том, что тесты не определены.

keywords: Массив ключевых слов, относящихся к проекту, помогает в поиске при публикации.

author: Имя автора проекта.

license: Лицензия для вашего проекта.

dependencies: Список зависимостей, необходимых для выполнения приложения, такие как express, jsonwebtoken, pg, dotenv.

devDependencies: Зависимости, необходимые только в процессе разработки, например, nodemon.

# my-shop/server/Dockerfile

Dockerfile

# Используем образ Node.js

FROM node:14

# Устанавливаем рабочую директорию

WORKDIR /usr/src/app

# Копируем package.json и устанавливаем зависимости

COPY package\*.json ./

RUN npm install

# Копируем все файлы проекта

COPY . .

# Открываем порт

EXPOSE 5000

# Запускаем сервер

CMD ["node", "src/server.js"]

# my-shop/server/docker-compose.yml

yaml

version: '3.8'

services:

db:

image: postgres:latest

environment:

POSTGRES\_USER: example\_user

POSTGRES\_PASSWORD: example\_pass

POSTGRES\_DB: shop\_db

ports:

- '5432:5432'

server:

build: .

ports:

- '5000:5000'

depends\_on:

- db

environment:

DATABASE\_URL: postgres://example\_user:example\_pass@db:5432/shop\_db

# my-shop/server/init.sql

sql

CREATE TABLE users (

user\_id SERIAL PRIMARY KEY,

login VARCHAR(50) UNIQUE NOT NULL,

password VARCHAR(100) NOT NULL

);

CREATE TABLE categories (

category\_id SERIAL PRIMARY KEY,

category VARCHAR(50) NOT NULL

);

CREATE TABLE products (

product\_id SERIAL PRIMARY KEY,

title VARCHAR(50) NOT NULL,

description TEXT,

price DECIMAL(10, 2) NOT NULL,

category\_id INTEGER REFERENCES categories(category\_id)

);

CREATE TABLE pictures (

picture\_id SERIAL PRIMARY KEY,

picture VARCHAR(255) NOT NULL

);

CREATE TABLE cart (

cart\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

user\_id INTEGER REFERENCES users(user\_id)

);

CREATE TABLE cart\_product (

cart\_id UUID REFERENCES cart(cart\_id),

product\_id INTEGER REFERENCES products(product\_id),

PRIMARY KEY (cart\_id, product\_id)

);

# my-shop/server/src/server.js

javascript

const express = require('express');

const bodyParser = require('body-parser');

const cors = require('cors');

const authRoutes = require('./routes/authRoutes');

const productRoutes = require('./routes/productRoutes');

const cartRoutes = require('./routes/cartRoutes');

const app = express();

app.use(cors());

app.use(bodyParser.json());

// Маршруты

app.use('/auth', authRoutes);

app.use('/products', productRoutes);

app.use('/cart', cartRoutes);

// Ошибка 404

app.use((req, res) => {

res.status(404).send('404 Not Found');

});

// Запуск сервера

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`);

});

# my-shop/server/src/routes/authRoutes.js

Javascript

const express = require('express');

const { register, login } = require('../controllers/authController');

const router = express.Router();

// Регистрация

router.post('/register', register);

// Авторизация

router.post('/login', login);

module.exports = router;

# my-shop/server/src/routes/cartRoutes.js

Javascript

const express = require('express');

const { getCart, addProductToCart, removeProductFromCart } = require('../controllers/cartController');

const authMiddleware = require('../middleware/authMiddleware');

const router = express.Router();

// Получить корзину пользователя по user\_id

router.get('/', authMiddleware, getCart);

// Добавить продукт в корзину

router.put('/', authMiddleware, addProductToCart);

// Удалить продукт из корзины

router.delete('/', authMiddleware, removeProductFromCart);

module.exports = router;

# my-shop/server/src/routes/productRoutes.js

javascript

const express = require('express');

const { getProducts, getProductById, createProduct, updateProduct, deleteProduct } = require('../controllers/productController');

const router = express.Router();

// Получить все продукты

router.get('/', getProducts);

// Получить продукт по ID

router.get('/:id', getProductById);

// Создать продукт

router.post('/', createProduct);

// Обновить продукт

router.put('/:id', updateProduct);

// Удалить продукт

router.delete('/:id', deleteProduct);

module.exports = router;

# my-shop/server/src/controllers/authController.js

javascript

const jwt = require('jsonwebtoken');

const db = require('../models/db');

// Регистрация пользователя

const register = async (req, res) => {

const { login, password } = req.body;

try {

// Примеры валидации и создания пользователя...

res.status(201).send('User registered');

} catch (error) {

res.status(500).send('Server error');

}

};

// Авторизация

const login = async (req, res) => {

const { login, password } = req.body;

try {

// Примеры валидации и авторизации...

const token = jwt.sign({ userId }, 'secret', { expiresIn: '1h' });

res.json({ token });

} catch (error) {

res.status(500).send('Server error');

}

};

module.exports = { register, login };

# my-shop/server/src/controllers/productController.js

javascript

const db = require('../models/db');

// Получить все продукты

const getProducts = async (req, res) => {

try {

const products = await db.query('SELECT \* FROM product');

res.json(products.rows);

} catch (error) {

res.status(500).send('Server error');

}

};

// Получить продукт по ID

const getProductById = async (req, res) => {

const { id } = req.params;

try {

const product = await db.query('SELECT \* FROM product WHERE product\_id = $1', [id]);

if (product.rows.length === 0) return res.status(404).send('Product not found');

res.json(product.rows[0]);

} catch (error) {

res.status(500).send('Server error');

}

};

// Создать продукт

const createProduct = async (req, res) => {

const { title, description, price, category\_id } = req.body;

try {

await db.query('INSERT INTO product (title, description, price, category\_id) VALUES ($1, $2, $3, $4)', [title, description, price, category\_id]);

res.status(201).send('Product created');

} catch (error) {

res.status(500).send('Server error');

}

};

// Обновить продукт

const updateProduct = async (req, res) => {

const { id } = req.params;

const { title, description, price, category\_id } = req.body;

try {

await db.query('UPDATE product SET title = $1, description = $2, price = $3, category\_id = $4 WHERE product\_id = $5', [title, description, price, category\_id, id]);

res.send('Product updated');

} catch (error) {

res.status(500).send('Server error');

}

};

// Удалить продукт

const deleteProduct = async (req, res) => {

const { id } = req.params;

try {

await db.query('DELETE FROM product WHERE product\_id = $1', [id]);

res.send('Product deleted');

} catch (error) {

res.status(500).send('Server error');

}

};

module.exports = { getProducts, getProductById, createProduct, updateProduct, deleteProduct };

# my-shop/server/src/controllers/cartController.js

Javascript

const db = require('../models/db');

// Получить корзину пользователя

const getCart = async (req, res) => {

const userId = req.user.userId; // Получаем userId из токена

try {

const cart = await db.query('SELECT \* FROM cart WHERE user\_id = $1', [userId]);

res.json(cart.rows);

} catch (error) {

res.status(500).send('Server error');

}

};

// Добавить продукт в корзину

const addProductToCart = async (req, res) => {

const userId = req.user.userId;

const { productId } = req.body; // productId из запроса

try {

await db.query('INSERT INTO cart\_product (cart\_id, product\_id) VALUES ($1, $2)', [cart\_id, productId]);

res.send('Product added to cart');

} catch (error) {

res.status(500).send('Server error');

}

};

// Удалить продукт из корзины

const removeProductFromCart = async (req, res) => {

const userId = req.user.userId;

const { productId } = req.body;

try {

await db.query('DELETE FROM cart\_product WHERE product\_id = $1 AND cart\_id = $2', [productId, cart\_id]);

res.send('Product removed from cart');

} catch (error) {

res.status(500).send('Server error');

}

};

module.exports = { getCart, addProductToCart, removeProductFromCart };

# my-shop/server/src/models/db.js

javascript

const { Pool } = require('pg');

// Создаем пул соединений к базе данных

const pool = new Pool({

user: process.env.POSTGRES\_USER,

host: 'db', // имя сервиса из docker-compose

database: process.env.POSTGRES\_DB,

password: process.env.POSTGRES\_PASSWORD,

port: 5432,

});

// Экспортируем запросы к базе данных

module.exports = {

query: (text, params) => pool.query(text, params),

};

# my-shop/server/src/models/user.js

javascript

const db = require('./db');

// Модель пользователя

const User = {

// Метод для создания нового пользователя

create: async (login, password) => {

const result = await db.query('INSERT INTO users (login, password) VALUES ($1, $2) RETURNING user\_id', [login, password]);

return result.rows[0].user\_id;

},

// Метод для получения пользователя по логину

getByLogin: async (login) => {

const result = await db.query('SELECT \* FROM users WHERE login = $1', [login]);

return result.rows[0];

}

};

module.exports = User;

# my-shop/server/src/models/product.js

javascript

const db = require('./db');

// Модель продукта

const Product = {

// Метод для создания нового продукта

create: async (title, description, price, category\_id) => {

await db.query('INSERT INTO products (title, description, price, category\_id) VALUES ($1, $2, $3, $4)', [title, description, price, category\_id]);

},

// Метод для получения всех продуктов

getAll: async (filters) => {

let query = 'SELECT \* FROM products';

// Реализуйте фильтрацию на основе filters

return await db.query(query);

}

};

module.exports = Product;

# my-shop/server/src/models/category.js

Javascript

const db = require('./db');

// Модель категории

const Category = {

getAll: async () => {

const result = await db.query('SELECT \* FROM categories');

return result.rows;

}

};

module.exports = Category;

# my-shop/server/src/models/cart.js

Javascript

const db = require('./db');

// Модель корзины

const Cart = {

// Метод для получения корзины по user\_id

getByUserId: async (userId) => {

const result = await db.query('SELECT \* FROM cart WHERE user\_id = $1', [userId]);

return result.rows[0];

}

};

module.exports = Cart;

# my-shop/server/src/middleware/authMiddleware.js

Javascript

const jwt = require('jsonwebtoken');

// Middleware для проверки авторизации

const authMiddleware = (req, res, next) => {

const token = req.header('Authorization')?.replace('Bearer ', '');

if (!token) return res.status(401).send('Access denied');

try {

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

req.user = decoded; // Сохраняем информацию о пользователе в req

next();

} catch (error) {

res.status(400).send('Invalid token');

}

};

module.exports = authMiddleware;

# my-shop/server/src/.env

Dotenv

POSTGRES\_USER=example\_user

POSTGRES\_PASSWORD=example\_pass

POSTGRES\_DB=shop\_db

JWT\_SECRET=your\_jwt\_secret\_here

## 