Лабораторная работа № 7

Разработка API и серверной части приложения

Цель работы: получить навыки проектирования и разработки серверного приложения, реализации бизнес-логики, REST API, а также настройки CI/CD и инструментов контроля качества кода.

В рамках лабораторной работы было разработано серверное приложение для интернет-магазина техники **"TechStore"**. Приложение обеспечивает полный цикл работы интернет-магазина: от регистрации пользователей и просмотра каталога до оформления заказов и получения уведомлений на почту.

Техническое задание и выбор инструментов

Для реализации серверной части был выбран язык **Java 21** и фреймворк **Spring Boot 3.3.5**, так как этот стек является стандартом для разработки высоконагруженных корпоративных систем.

Используемый стек технологий:

- **Core:** Java 21, Spring Boot (Web, Data JPA, Security).
- **Database:** PostgreSQL (основное хранилище), Redis (кэширование и корзина).
- **Messaging:** Apache Kafka (событийно-ориентированная архитектура).
- **Migrations:** Liquibase.
- **Docs:** OpenAPI (Swagger).
- **Build & Deploy:** Maven, Docker, Docker Compose.
- **Quality:** SonarQube, Jacoco.

Архитектура приложения

Приложение построено по **монолитной архитектуре** с четким разделением на слои (Layered Architecture):

1. **Controller Layer:** Обработка HTTP-запросов (@RestController).
2. **Service Layer:** Бизнес-логика (@Service).
3. **Repository Layer:** Взаимодействие с БД (Spring Data JPA).

4. **DTO Layer:** Объекты передачи данных, маппинг через ModelMapper.

Структура проекта:

```
src/main/java/com/example/techstore
├── cart         // Модуль корзины (Redis)
├── catalog      // Товары, категории, бренды, фильтры
├── common       // Общие утилиты и исключения
├── config       // Конфигурация Security, Swagger, Cache
├── notification // Сервис уведомлений (Email)
├── order        // Заказы и события Kafka
├── review       // Отзывы и рейтинги
├── user         // Аутентификация и пользователи
└── TechStoreApplication.java
```

Реализация серверной части (Backend)

4.1. База данных и миграции (PostgreSQL + Liquibase)

В качестве СУБД используется PostgreSQL. Для управления схемой базы данных используется **Liquibase**. Это позволяет версионировать изменения БД и накатывать их автоматически при старте приложения.

Основные таблицы: users, roles, products, orders, reviews, carts (в Redis).

Пример скрипта миграции (001-initial-schema.sql):

```
CREATE TABLE products (
    product_id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    title VARCHAR(255) NOT NULL,
    price DECIMAL(10, 2) NOT NULL,
    ...
);
```

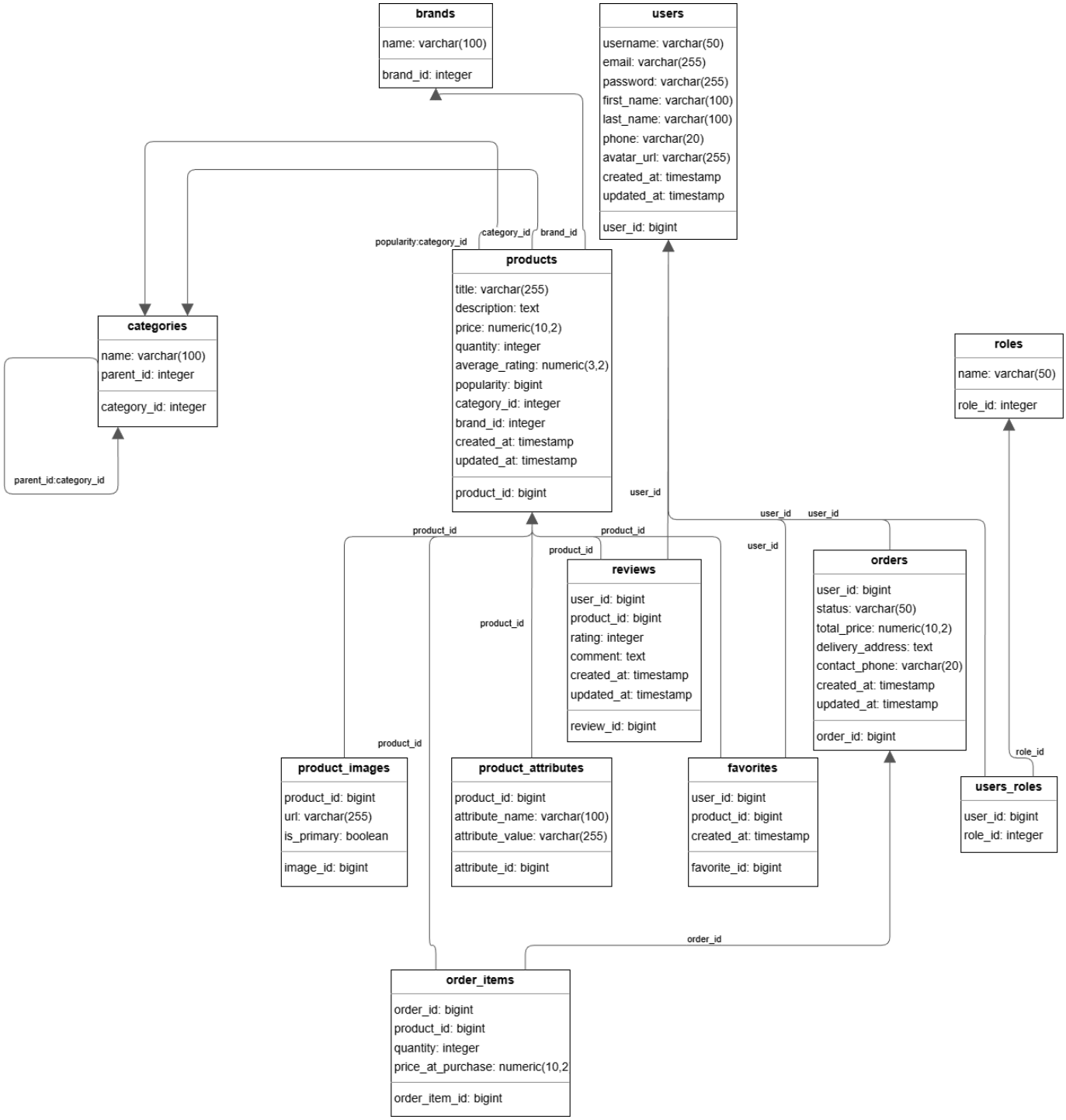

Рис. 1. Схема базы данных.

4.2. Безопасность и JWT (Spring Security)

Реализована Stateless-аутентификация на основе **JWT (JSON Web Tokens)**.

- При входе (/auth/sign-in) пользователю выдается токен.
- Фильтр JwtRequestFilter перехватывает запросы, валидирует токен и устанавливает контекст безопасности.
- Пароли хранятся в зашифрованном виде (BCryptPasswordEncoder).

4.3. Каталог товаров и фильтрация

Реализован гибкий поиск товаров с использованием **JPA Specifications**. Это позволяет фильтровать товары по множеству параметров одновременно:

- Поиск по названию/описанию.
- Диапазон цен (min/max).
- Бренд и Категория.
- Наличие на складе.
- Рейтинг.

Класс ProductSpecifications динамически собирает SQL-запрос WHERE условий.

Postman запрос GET /api/v1/products с параметрами фильтрации:

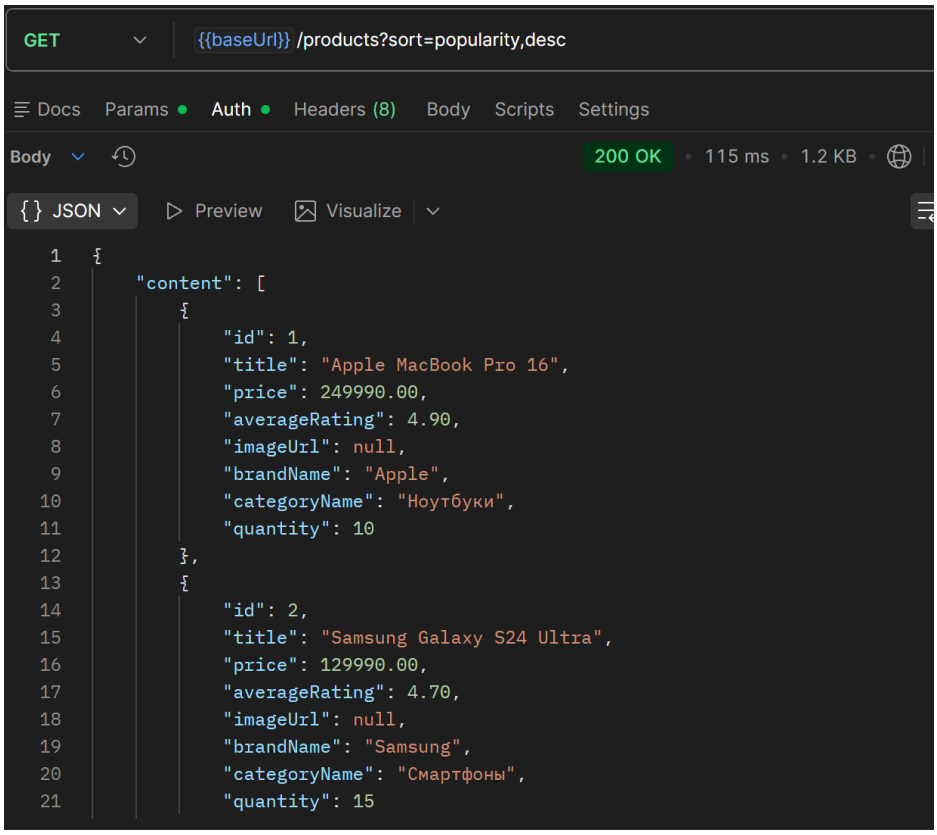

Рис. 2. Результат фильтрации товаров через Postman.

4.4. Корзина и кэширование (Redis)

Для высокой производительности корзина пользователя хранится не в PostgreSQL, а в **Redis**. Это снижает нагрузку на основную БД.

- TTL корзины: 7 дней.
- Сущности: CartDTO, CartItem.

Также настроено кэширование справочных данных (роли, категории) с помощью аннотации @Cacheable.

4.5. Заказы и асинхронные уведомления (Kafka)

Реализована событийно-ориентированная архитектура (EDA) для обработки заказов.

1. Пользователь создает заказ (POST /orders).
2. Сервис сохраняет заказ в PostgreSQL.

3. **Producer** отправляет событие OrderPlacedEvent в топик Kafka order-placed.
4. **Consumer** (NotificationService) читает событие и отправляет Email пользователю.

Используется **Avro Schema** для строгой типизации сообщений Kafka.

Схема события (order-placed.avsc):

```
{
  "type": "record",
  "name": "OrderPlacedEvent",
  "namespace": "com.example.techstore.order.event",
  "fields": [
    { "name": "orderId", "type": "long" },
    { "name": "email", "type": "string" },
    { "name": "firstName", "type": "string" },
    { "name": "totalPrice", "type": "string" },
    { "name": "itemsCount", "type": "int" }
  ]
}
```

API Документация (Swagger)

Для документирования REST API подключена библиотека **SpringDoc OpenAPI**. По адресу /swagger-ui.html доступен интерактивный интерфейс для тестирования запросов.

Реализованы контроллеры:

- AuthController: Регистрация, вход.
- CatalogController: Товары, бренды, категории.
- CartController: Управление корзиной.
- OrderController: Оформление заказов.
- ReviewController: Отзывы.

Страница Swagger UI с открытым списком контроллеров:

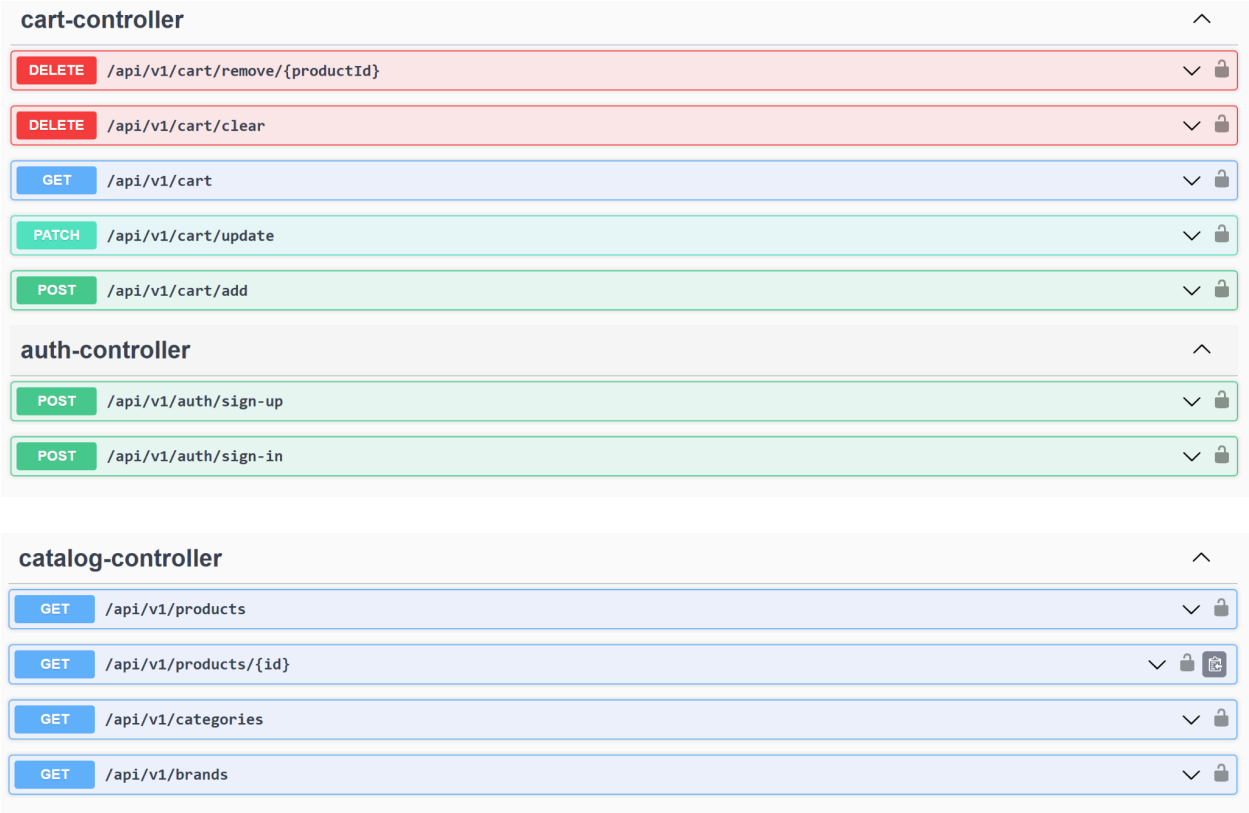

Рис. 3. Swagger UI документация API.

Качество кода и тестирование

В проекте настроены инструменты статического анализа и покрытия тестами:

1. **Unit-тесты:** Написаны тесты для сервисного слоя (AuthServiceImplTest, OrderServiceImplTest, UserServiceImplTest) с использованием **JUnit 5** и **Mockito**.
2. **Jacoco:** Плагин для проверки покрытия кода тестами.
3. **SonarQube:** Подключен плагин для анализа уязвимостей и дублирования.

Результаты прогона тестов в IDE:

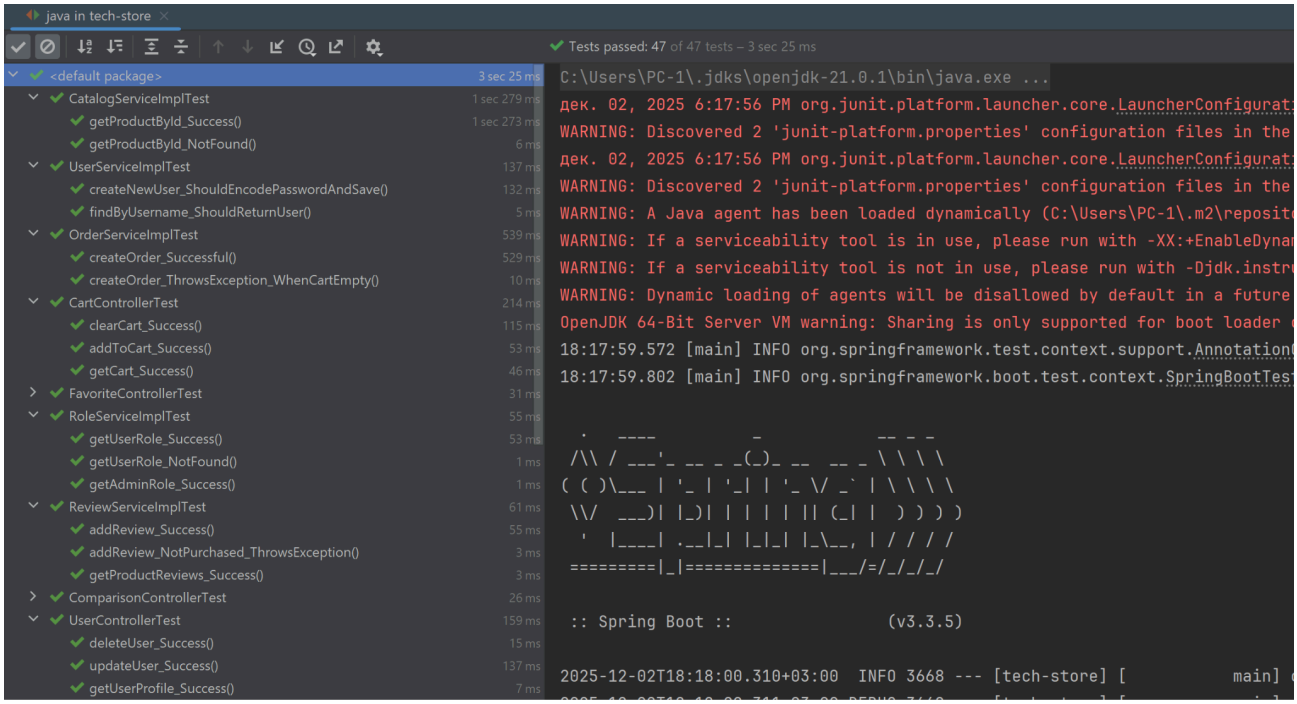

Рис. 4. Успешное прохождение Unit-тестов.

Развертывание (Docker)

Для контейнеризации приложения создан Dockerfile (Multistage build) и docker-compose.yml.

Среда включает в себя:

- backend-app: Приложение (порт 8080).
- postgres-main: База данных.
- redis: Кэш.
- zookeeper & kafka: Брокер сообщений.
- schema-registry: Реестр схем Avro.
- kafka-ui: Интерфейс для управления Kafka.

Вывод команды docker ps:

```
PS C:\programming\javaProjects\pchmi\lab7\tech-store\project12a-backend-bimbimbambam> docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED          STATUS                    PORTS
                                                NAMES
bcb10ed6009a   provectuslabs/kafka-ui:latest        "/bin/sh -c 'java --…"   21 minutes ago   Up 21 minutes             0.0.0.0:
8090->8080/tcp                                  kafka-ui
e721e7420832   90jer/tech-store:latest              "java -cp @/app/jib-…"   21 minutes ago   Up 21 minutes             0.0.0.0:
8080->8080/tcp                                  project12a-backend-bimbimbambam-backend-app-1
b4a5c527d396   confluentinc/cp-schema-registry:7.3.0   "/etc/confluent/dock…"   21 minutes ago   Up 21 minutes             0.0.0.0:
8081->8081/tcp                                  schema-registry
970c49e1d3a0   confluentinc/cp-kafka:7.3.0          "/etc/confluent/dock…"   21 minutes ago   Up 21 minutes             0.0.0.0:
9092->9092/tcp, 0.0.0.0:29092->29092/tcp        kafka
21e40af8ed3f   postgres:16                          "docker-entrypoint.s…"   21 minutes ago   Up 21 minutes (healthy)   0.0.0.0:
5433->5432/tcp                                  postgres-main
c0a64f1db349   redis:alpine                         "docker-entrypoint.s…"   21 minutes ago   Up 21 minutes             0.0.0.0:
6379->6379/tcp                                  redis
ca9111a25430   confluentinc/cp-zookeeper:7.3.0      "/etc/confluent/dock…"   21 minutes ago   Up 21 minutes             2888/tcp
, 0.0.0.0:2181->2181/tcp, 3888/tcp              zookeeper
```

Рис. 5. Запущенные контейнеры приложения.