



Geekbrains

## **Разработка веб-приложения для маркетплейса ягод (Berry Market)**

Программа: Разработчик

Специализация: Программист

Алексеев Д.В.

Киров

2025

# **Содержание**

<b>Введение .....</b>	<b>3</b>
<b>Обоснование темы проекта .....</b>	<b>3</b>
<b>Цель проекта .....</b>	<b>3</b>
<b>Проблематика .....</b>	<b>3</b>
<b>Задачи проекта .....</b>	<b>3</b>
<b>Инструменты и технологии .....</b>	<b>4</b>
<b>Состав команды .....</b>	<b>4</b>
<b>Глава 1. Теоретическая часть.....</b>	<b>5</b>
<b>1.1 Технологии и подходы для разработки веб-приложений .....</b>	<b>5</b>
<b>1.2 Анализ существующих решений.....</b>	<b>7</b>
<b>Глава 2. Практическая часть .....</b>	<b>9</b>
<b>2.1 Архитектура приложения .....</b>	<b>9</b>
<b>2.2 Реализация функционала .....</b>	<b>12</b>
<b>2.3 Интерфейс пользователя и тестирование API .....</b>	<b>18</b>
<b>2.4 Пример использования .....</b>	<b>19</b>
<b>Заключение.....</b>	<b>25</b>
<b>Итоги .....</b>	<b>25</b>
<b>Практическая значимость работы .....</b>	<b>25</b>
<b>Перспективы развития .....</b>	<b>25</b>
<b>Список используемой литературы .....</b>	<b>27</b>
<b>Литература .....</b>	<b>27</b>
<b>Онлайн-ресурсы .....</b>	<b>27</b>
<b>Зависимости и библиотеки.....</b>	<b>27</b>
<b>Приложения .....</b>	<b>28</b>
<b>Приложение 1. Структура проекта.....</b>	<b>28</b>
<b>Приложение 2. Команды для работы с проектом .....</b>	<b>29</b>

## **Введение**

В наше время интернет-магазины и маркетплейсы стали обычным делом. Проект «Berry Market» – это веб-приложение для связи сборщиков ягод и заготовителей. Оно позволяет размещать объявления о покупке и продаже свежих ягод.

## **Обоснование темы проекта**

Ягоды собирают сезонно, и нужно быстро находить покупателей или продавцов. Обычные сайты объявлений не подходят для ягод – там нет нужных фильтров и категорий.

«Berry Market» решает эти проблемы. Он сделан специально для ягод: здесь есть разделение на группы (лесные, садовые, болотные), поиск по типу ягоды и цене, и удобная регистрация для сборщиков и заготовителей.

## **Цель проекта**

Создать REST API для маркетплейса ягод. Нужно реализовать:

- Регистрацию и авторизацию для сборщиков и заготовителей
- Создание, просмотр, изменение и удаление объявлений (CRUD)
- Поиск объявлений по типу ягоды, группе, цене и статусу
- Безопасность через JWT-токены
- Упаковку в Docker для простого запуска

## **Проблематика**

У обычных сайтов объявлений есть недостатки:

- Нет специализации для ягод
- Плохой поиск и фильтры
- Сложно настраивать и запускать
- Нет разделения на сборщиков и заготовителей
- «Berry Market» решает эти проблемы. Он написан на Java и Spring Boot.

## **Задачи проекта**

- Изучить существующие маркетплейсы
- Спроектировать архитектуру REST API на Spring Boot и PostgreSQL
- Настроить авторизацию через JWT
- Реализовать CRUD для объявлений
- Сделать поиск объявлений через Spring Data JPA Specifications
- Упаковать приложение в Docker

- Протестировать API через Bruno
- Проанализировать дальнейшее развитие приложения

## Инструменты и технологии

Для реализации проекта выбраны следующие технологии:

- **Java 21** и **Spring Boot 3.2.0**: для создания серверной логики и REST API
- **PostgreSQL**: для хранения данных о пользователях, ягодах и объявлениях
- **Spring Security** и **JWT**: для обеспечения безопасности приложения
- **Spring Data JPA**: для работы с базой данных
- **Docker** и **Docker Compose**: для контейнеризации и упрощения развертывания
- **Maven**: для управления зависимостями и сборки проекта
- **Bruno**: для тестирования REST API

## Состав команды

Проект выполнен самостоятельно. В роли:

- Backend-разработчика – написал REST API, бизнес-логику и безопасность
- Архитектора БД – спроектировал схему базы данных
- Тестировщика – проверил API через Bruno
- DevOps-инженера – настроил Docker для запуска

## **Глава 1. Теоретическая часть**

### **1.1 Технологии и подходы для разработки веб-приложений**

#### **Обзор Java и его применения в веб-разработке**

Java – популярный язык для веб-приложений. Его преимущества:

- **Работает везде:** код запускается на любой системе, где есть JVM
- **Быстро работает:** благодаря JIT-компиляции скорость высокая
- **Безопасный:** встроенная защита, автоматическое управление памятью
- **Масштабируемый:** можно делать большие приложения с Spring и Hibernate
- **Многопоточность:** обрабатывает много запросов одновременно

Java 21 – современная версия с поддержкой виртуальных потоков и улучшенной производительностью.

#### **Основные аспекты Spring Boot для построения REST API**

Spring Boot упрощает разработку Java-приложений. Основные возможности:

- **Автоконфигурация:** сам настраивает компоненты по зависимостям
- **REST API:** простые аннотации @RestController, @GetMapping, @PostMapping для создания API
- **Работа с БД:** Spring Data JPA упрощает работу с базами данных через репозитории
- **Управление зависимостями:** Dependency Injection делает код проще в тестировании
- **Профили:** можно использовать разные настройки для разработки и рабочей версии

#### **Использование PostgreSQL для хранения данных**

PostgreSQL – мощная открытая база данных. Преимущества:

- **Быстрая:** хорошо справляется с большими объемами данных и сложными запросами
- **Надежная:** поддерживает ACID-транзакции, репликацию, восстановление данных
- **Масштабируемая:** можно масштабировать вертикально и горизонтально
- **Расширяемая:** можно добавлять свои типы данных и функции
- **Совместимая:** работает с Spring Boot через JDBC
- **Поддержка JSON:** можно хранить JSON данные

В «Berry Market» PostgreSQL хранит данные о пользователях, ягодах и объявлениях. Spring Data JPA упрощает работу с базой данных.

### **Spring Security и JWT для обеспечения безопасности**

Spring Security – фреймворк для авторизации и аутентификации. Основные возможности:

- **Аутентификация:** можно использовать формы, базовую HTTP-авторизацию или JWT
- **Авторизация:** гибкий контроль доступа по ролям
- **Защита:** встроенная защита от CSRF, XSS и других атак
- **Интеграция:** хорошо работает с Spring Boot

JWT (JSON Web Token) – стандарт для передачи данных между сервером и клиентом. Преимущества:

- **Без состояния:** не нужно хранить сессии на сервере, все данные в токене
- **Масштабируемость:** удобно для микросервисов
- **Безопасность:** подпись гарантирует, что токен неизменен
- **Гибкость:** можно добавить любые данные в токен

В проекте JWT используется так: при входе генерируется токен, он нужен для всех защищенных запросов.

## **Docker для контейнеризации приложений**

Docker упаковывает приложение и все его зависимости в контейнер.  
Преимущества:

- **Изоляция:** каждое приложение в своем контейнере, не мешает другим
- **Переносимость:** работает одинаково на любой системе с Docker
- **Масштабируемость:** легко запустить несколько экземпляров
- **Простое развертывание:** вся настройка (приложение + БД) в docker-compose.yml

Docker Compose запускает несколько контейнеров вместе, например Spring Boot приложение и PostgreSQL.

## **1.2 Анализ существующих решений**

### **Сравнение популярных приложений-викторин по языку Java**

### **Сравнение популярных маркетплейсов и платформ объявлений**

Популярные платформы объявлений:

#### **1. Авито:**

- Плюсы: много пользователей, хороший поиск
- Минусы: высокая комиссия, нет специализации для ягод, сложно настраивать

#### **2. Юла:**

- Плюсы: бесплатные объявления, поиск по геолокации
- Минусы: проект практически умер

## **Уникальные особенности проекта «Berry Market»**

«Berry Market» решает проблемы других платформ:

- **Для ягод:** есть группы (лесные, садовые, болотные) и типы ягод
- **Разные роли:** разделение на сборщиков (PICKER) и заготовителей (HARVESTER)
- **Удобный поиск:** по типу ягоды, группе, цене и статусу
- **Легкий запуск:** Docker позволяет запустить за несколько минут
- **REST API:** можно интегрировать с другими системами
- **Безопасность:** JWT-токены для авторизации

## **Глава 2. Практическая часть**

### **2.1 Архитектура приложения**

#### **Описание модульной структуры приложения**

Приложение разделено на модули для удобства. Основные части:

##### **1. Сущности (Entity):**

- Классы данных: User, Berry, BerryGroup, Advert
- Используют JPA для работы с таблицами в БД
- Определяют связи между таблицами

##### **2. Репозитории (Repository):**

- Интерфейсы для работы с БД: UserRepository, BerryRepository, BerryGroupRepository, AdvertRepository
- Spring Data JPA автоматически создает CRUD операции
- Можно добавлять свои запросы через @Query

##### **3. Сервисы (Service):**

- Бизнес-логика приложения: AuthService, AdvertService, BerryService
- Проверяют данные и выполняют правила
- Преобразуют Entity в DTO и обратно

##### **4. Контроллеры (Controller):**

- REST API: AuthController, AdvertController, BerryController, SearchController
- Обрабатывают HTTP запросы
- Используют DTO для обмена данными

##### **5. Безопасность:**

- Авторизация: SecurityConfig, JwtUtil, JwtAuthenticationFilter, UserDetailsServiceImpl
- Защищает endpoint и генерирует JWT токены

##### **6. Исключения:**

- Обработка ошибок через GlobalExceptionHandler

- Возвращает понятные сообщения об ошибках

## Используемые технологии: Spring Boot, JPA, Docker

### 1. Spring Boot:

- Упрощает разработку серверной части
- Встроенные инструменты для REST API
- Автоматическая настройка компонентов
- Работает с Spring Data JPA

### 2. Spring Data JPA:

- Работа с PostgreSQL
- Упрощает CRUD операции
- Specifications для динамических запросов
- Автоматически генерирует SQL

### 3. Docker:

- Упаковка приложения в контейнер
- Легко запускать на разных системах
- Упрощает настройку среды
- Docker Compose управляет приложением и БД вместе

## Структура базы данных

База данных состоит из следующих основных таблиц:

**users** – хранит информацию о пользователях:

- id – уникальный идентификатор
- email – электронная почта (уникальное значение)
- password – хешированный пароль
- name – имя пользователя
- type – тип пользователя
- created\_at – дата регистрации

**berry\_groups** – группы ягод:

- id – уникальный идентификатор
- name – название группы
- description – описание группы

**berry** – типы ягод:

- id – уникальный идентификатор
- name – название ягоды
- group\_id – внешний ключ на berry\_groups

**adverts** – объявления о покупке/продаже:

- id – уникальный идентификатор
- berry\_id – внешний ключ на berry
- user\_id – внешний ключ на users
- quantity – количество в килограммах
- price – цена за килограмм
- address – адрес текстом
- description – описание объявления

- type – тип объявления
- status – статус объявления
- created\_at – дата создания

## 2.2 Реализация функционала

### Создание REST API для управления вопросами и пользователями

Для реализации REST API в приложении «Berry Market» используется Spring Boot. Основные endpoint API включают:

#### 1. Регистрация

```
POST /api/auth/register
Content-Type: application/json

{
  "email": "user@example.com",
  "password": "password123",
  "name": "Иван Иванов",
  "userType": "PICKER"
}
```

Регистрация поддерживает два типа пользователей:

- PICKER – сборщик ягод, может размещать объявления о продаже
- HARVESTER – заготовитель, может размещать объявления о покупке

#### 2. Авторизация:

```
POST /api/auth/login
Content-Type: application/json

{
  "email": "user@example.com",
  "password": "password123"
}

Response:
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "email": "user@example.com",
  "name": "Иван Иванов",
  "userType": "PICKER"
}
```

После успешной авторизации возвращается JWT токен для дальнейших запросов.

### 3. Работы с данными

Получение всех групп ягод:

```
GET /api/berries/groups

Response:
[
  {
    "id": 1,
    "name": "Лесные",
    "description": "Дикорастущие ягоды, собираемые в лесу"
  },
  ...
]
```

Получение всех типов ягод:

```
GET /api/berries/types

Response:
[
  {
    "id": 1,
    "name": "Черника",
    "groupId": 1,
    "groupName": "Лесные"
  },
  ...
]
```

Получение типов ягод по группе:

```
GET /api/berries/groups/{groupId}/types
```

## 4. Работа с объявлениями

Все запросы объявлений требуют авторизации (заголовок Authorization: Bearer {token}).

### Создание объявления:

```
POST /api/adverts
Authorization: Bearer {token}
Content-Type: application/json

{
  "berryId": 1,
  "quantity": 10.5,
  "price": 250.00,
  "address": "Москва, Лесопарк Сокольники",
  "description": "Свежая черника, только что собранная",
  "type": "SELL"
}
```

### Получение объявления по ID:

```
GET /api/adverts/{id}
Authorization: Bearer {token}
```

### Получение своих объявлений:

```
(E) GET /api/adverts/my
Authorization: Bearer {token}
```

### Получение всех объявлений (с пагинацией):

```
POST /api/adverts/all
Authorization: Bearer {token}
Content-Type: application/json

{
  "page": 0,
  "size": 10
}
```

### Обновление объявления:

```
PUT /api/adverts/{id}
Authorization: Bearer {token}
Content-Type: application/json

{
  "berryId": 1,
  "quantity": 12.0,
  "price": 280.00,
  "address": "Москва",
  "description": "Обновленное описание",
  "type": "SELL"
}
```

Изменение статуса объявления:

```
PATCH /api/adverts/{id}/status?status=CLOSED
Authorization: Bearer {token}
```

Удаление объявления:

```
DELETE /api/adverts/{id}
Authorization: Bearer {token}
```

## 5. Поиск

Поиск объявлений:

```
POST /api/search
Content-Type: application/json

{
  "berryId": 1,          // optional
  "berryGroupId": 1,     // optional
  "minPrice": 200,       // optional
  "maxPrice": 400,       // optional
  "advertType": "SELL"   // optional: "SELL" or "BUY"
}
```

Поиск поддерживает комбинацию критериев для гибкой фильтрации объявлений.

## Реализация обработки ответов и подсчёта результатов

Серверная логика включает:

### 1. Валидация данных:

- Bean Validation (@Valid, @NotNull, @Email)
- Проверка правил (например, пользователь может редактировать только свои объявления)

## **2. Авторизация:**

- Проверка JWT токена в каждом защищенном запросе
- Извлечение ID пользователя из токена
- Проверка прав доступа

## **3. Работа с БД:**

- Spring Data JPA для CRUD операций
- Specifications для динамических запросов поиска
- Оптимизация через lazy loading

## **4. Обработка ошибок:**

- Централизованная обработка через @ControllerAdvice
- Структурированные сообщения об ошибках
- Разные типы ошибок (400, 401, 403, 404, 500)

## **5. Преобразование данных:**

- Entity → DTO для отправки клиенту
- DTO → Entity при сохранении

## **Реализация JWT аутентификации**

Аутентификация на JWT токенах:

### **1. Генерация токена (JwtUtil):**

- Создание токена с данными (email, userId, userType)
- Подписание секретным ключом

- Время жизни – 24 часа

## **2. Фильтр (JwtAuthenticationFilter):**

- Перехватывает каждый HTTP запрос
- Извлекает токен из заголовка Authorization
- Проверяет токен и устанавливает аутентификацию

## **3. Конфигурация:**

- Настройка доступа к endpoint
- Публичные: /api/auth/\*\*
- Защищенные: /api/adverts/\*\*, /api/berries/\*\*, /api/search
- BCrypt для хеширования паролей

## **2.3 Интерфейс пользователя и тестирование API**

### **Тестирование с помощью Bruno**

Проект – это REST API без фронтенда, поэтому тестируем через Bruno.

#### **Структура коллекции:**

- **Auth** – регистрация и авторизация
- **Berries** – работа с группами и типами ягод
- **Adverts** – CRUD операции с объявлениями
- **Search** – поиск объявлений

#### **Переменные:**

- baseUrl – URL API (по умолчанию `http://localhost:8080`)
- authToken – JWT токен (сохраняется после логина)

#### **Последовательность тестирования:**

1. Регистрация пользователей
2. Авторизация
3. Получение списка ягод
4. Создание объявлений
5. Поиск объявлений
6. Обновление объявлений
7. Удаление объявлений

## Обработка ошибок и валидация

API возвращает ошибки в JSON:

```
{  
    "timestamp": "2024-10-31T12:00:00",  
    "status": 404,  
    "error": "Not Found",  
    "message": "Advert not found",  
    "path": "/api/adverts/999"  
}
```

### Коды ошибок:

- 400 – некорректные данные
- 401 – неверный логин/пароль или нет токена
- 403 – нет прав (например, редактировать чужое объявление)
- 404 – ресурс не найден
- 500 – ошибка сервера

## 2.4 Пример использования

### Примеры запросов к API

#### 1. Регистрация сборщика ягод

```
POST /api/auth/register HTTP/1.1  
Host: localhost:8080  
Content-Type: application/json  
  
{  
    "email": "picker@example.com",  
    "password": "password123",  
    "name": "Иван Иванов",  
    "userType": "PICKER"  
}
```

### Ответ:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "id": 1,
  "email": "picker@example.com",
  "name": "Иван Иванов",
  "userType": "PICKER"
}
```

## 2. Авторизация

```
POST /api/auth/login HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "email": "picker@example.com",
  "password": "password123"
}
```

### Ответ:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJwawNrzXAZXhhbXBsZS5jb20iLCJ1c2VySwQiOjEsInVzZXJUeXBlIjoiUElDS0VSIiwIZXhwIjoxNzI5ODcyODAwfQ...",
  "email": "picker@example.com",
  "name": "Иван Иванов",
  "userType": "PICKER"
}
```

## 3. Создание объявления о продаже

```
POST /api/adverts HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

{
  "berryId": 1,
  "quantity": 10.5,
  "price": 250.00,
  "address": "Москва, Лесопарк Сокольники",
  "description": "Свежая черника, собранная сегодня утром",
  "type": "SELL"
}
```

### Ответ:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "id": 1,
  "berry": {
    "id": 1,
    "name": "Черника",
    "groupId": 1,
    "groupName": "Лесные"
  },
  "quantity": 10.5,
  "price": 250.00,
  "address": "Москва, Лесопарк Сокольники",
  "description": "Свежая черника, собранная сегодня утром",
  "type": "SELL",
  "status": "ACTIVE",
  "createdAt": "2024-10-31T12:00:00"
}
```

#### 4. Поиск объявлений по типу ягоды

```
POST /api/search HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "berryId": 1,
  "advertType": "SELL"
}
```

**Ответ:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": 1,
    "berry": {
      "id": 1,
      "name": "Черника",
      "groupId": 1,
      "groupName": "Лесные"
    },
    "quantity": 10.5,
    "price": 250.00,
    "address": "Москва, Лесопарк Сокольники",
    "description": "Свежая черника, собранная сегодня утром",
    "type": "SELL",
    "status": "ACTIVE",
    "createdAt": "2024-10-31T12:00:00"
  }
]
```

## 5. Поиск объявлений по диапазону цен

```
POST /api/search HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "minPrice": 200,
  "maxPrice": 300,
  "advertType": "SELL"
}
```

## 6. Обновление объявления

```
PUT /api/adverts/1 HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
{
  "berryId": 1,
  "quantity": 12.0,
  "price": 280.00,
  "address": "Москва, Измайловский парк",
  "description": "Обновленное описание",
  "type": "SELL"
}
```

## Развертывание приложения с Docker

Приложение запускается через Docker Compose.

### Файл docker-compose.yml:

```
version: "3.8"

services:
  postgres:
    image: postgres:16-alpine
    container_name: berry-market-db
    environment:
      POSTGRES_DB: berry_market
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: berry-market-app
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://postgres:5432/berry_market
      SPRING_DATASOURCE_USERNAME: postgres
      SPRING_DATASOURCE_PASSWORD: postgres
    ports:
      - "8080:8080"
    depends_on:
      - postgres
```

### Команды для запуска:

```
# Запуск
docker-compose up --build -d

# Логи
docker-compose logs -f app

# Остановка
docker-compose down
```

После запуска:

- Создаются таблицы в БД
- Доступно на <http://localhost:8080>

## **Заключение**

### **Итоги**

Разработано REST API «Berry Market» для маркетплейса ягод. Выполнено:

- **Архитектура:** модульная структура (Entity, Repository, Service, Controller)
- **Авторизация:** JWT с двумя типами пользователей (сборщики и заготовители)
- **REST API:** полный набор endpoint для объявлений, ягод и пользователей
- **Поиск:** фильтрация по типу ягоды, группе, цене и статусу через Spring Data JPA Specifications
- **Безопасность:** BCrypt для паролей, проверка прав доступа, валидация
- **Docker:** контейнеризация для простого запуска

Приложение протестировано через Bruno, основные сценарии работают корректно.

### **Практическая значимость работы**

Приложение можно использовать как:

- Базовую платформу для региональных маркетплейсов ягод
- Обучающий пример для изучения Spring Boot, JWT и REST API
- Основу для расширения (рейтинги, чаты, платежи)

### **Перспективы развития**

Можно добавить:

- Frontend – веб-интерфейс или мобильное приложение
- Рейтинги и отзывы – для повышения доверия
- Уведомления – email или push о новых объявлениях
- Геолокация – карты и поиск по радиусу

- Аналитика – dashboard для администраторов
- Платежи – интеграция с платежными системами
- Рекомендации – ML для подбора объявлений

## **Список используемой литературы**

### **Литература**

1. Блинов А. В., Романчик В. С. Java. Методы программирования. – СПб.: Питер, 2021. – 720 с.
2. Шилдт Г. Полный справочник по Java. – 11-е изд. – М.: Вильямс, 2020. – 960 с.
3. Эккель Б. Философия Java. – 4-е изд. – СПб.: Питер, 2020. – 1168 с.
4. Уоллс К. Spring в действии. – 6-е изд. – СПб.: Питер, 2023. – 624 с.
5. Гутман Э., Майерс Г., Томас Д. Spring Boot в действии. – СПб.: Питер, 2022. – 368 с.

### **Онлайн-ресурсы**

6. Документация по Spring Framework: <https://spring.io/docs>
7. Документация по Spring Boot: <https://spring.io/projects/spring-boot>
8. Документация по Spring Security: <https://spring.io/projects/spring-security>
9. Документация по PostgreSQL: <https://www.postgresql.org/docs/>
10. Docker Documentation: <https://docs.docker.com/>
11. JWT.io – информация о JSON Web Tokens: <https://jwt.io/>
12. REST API Tutorial: <https://restfulapi.net/>
13. Spring Data JPA Reference Documentation: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>

### **Зависимости и библиотеки**

14. Spring Boot 3.2.0: <https://spring.io/projects/spring-boot>
15. Spring Data JPA: <https://spring.io/projects/spring-data-jpa>
16. Spring Security: <https://spring.io/projects/spring-security>
17. PostgreSQL JDBC Driver: <https://jdbc.postgresql.org/>
18. JJWT (Java JWT Library): <https://github.com/jwtk/jjwt>
19. Lombok: <https://projectlombok.org/>
20. Maven: <https://maven.apache.org/>
21. Docker: <https://www.docker.com/>
22. Bruno API Client: <https://www.usebruno.com/>

## Приложения

### Приложение 1. Структура проекта

```
berry-market/
└── src/main/java/com/berry/
    ├── BerryMarketApplication.java
    ├── config/
    │   └── SecurityConfig.java
    ├── controller/
    │   ├── AdvertController.java
    │   ├── AuthController.java
    │   ├── BerryController.java
    │   └── SearchController.java
    ├── dto/
    │   ├── AdvertRequest.java
    │   ├── AdvertResponse.java
    │   ├── AuthResponse.java
    │   ├── BerryGroupRequest.java
    │   ├── BerryGroupResponse.java
    │   ├── BerryRequest.java
    │   ├── BerryResponse.java
    │   ├── GetAllAdvertsRequest.java
    │   ├── LoginRequest.java
    │   ├── RegisterRequest.java
    │   ├── SearchRequest.java
    │   └── UserResponse.java
    ├── entity/
    │   ├── Advert.java
    │   ├── AdvertStatus.java
    │   ├── AdvertType.java
    │   ├── Berry.java
    │   ├── BerryGroup.java
    │   ├── User.java
    │   └── UserType.java
    ├── exception/
    │   ├── BadRequestException.java
    │   ├── ErrorResponse.java
    │   ├── GlobalExceptionHandler.java
    │   ├── ResourceNotFoundException.java
    │   └── UnauthorizedException.java
    ├── repository/
    │   ├── AdvertRepository.java
    │   ├── BerryGroupRepository.java
    │   ├── BerryRepository.java
    │   └── UserRepository.java
    ├── security/
    │   ├── JwtAuthenticationFilter.java
    │   ├── JwtUtil.java
    │   └── UserDetailsServiceImpl.java
    ├── service/
    │   ├── AdvertService.java
    │   ├── AuthService.java
    │   └── BerryService.java
    └── src/main/resources/
        ├── application.properties
        ├── application-dev.properties
        └── application-prod.properties
└── bruno/
    ├── Auth/
    └── Berries/
```

```
└── Adverts/
    └── Search/
    ├── docker-compose.yml
    ├── Dockerfile
    └── pom.xml
```

## Приложение 2. Команды для работы с проектом

### Сборка и запуск без Docker

```
# Компиляция проекта
mvn clean compile

# Сборка JAR файла
mvn clean package

# Запуск приложения
mvn spring-boot:run
```

### Работа с Docker

```
# Сборка и запуск контейнеров
docker-compose up --build -d

# Просмотр логов приложения
docker-compose logs -f app

# Просмотр логов базы данных
docker-compose logs -f postgres

# Остановка контейнеров
docker-compose down

# Остановка и удаление volumes
docker-compose down -v
```