

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 курсу  
Дисциплина: «Технологии сетевого программирования»

Выполнили: Гумаров Р. Т.,  
Инкин А. И.  
Группа: 6301-010302D

**Самара 2025**

## СОДЕРЖАНИЕ

Введение.....	4
1 Концепция проекта .....	5
1.1 Назначение проекта .....	5
1.2 Фреймворки и технологии .....	5
1.3 Основные сущности.....	6
1.4 Функциональные требования .....	6
2 Backend-часть .....	7
2.1 Архитектура БД.....	7
2.2 OpenApi-документация.....	7
2.3 Миграции .....	9
2.4 Entity-модели .....	10
2.5 Dto-объекты .....	11
2.6 Реализация контроллера.....	12
2.7 Security.....	13
2.7.1 Public и private rsa-ключи .....	13
2.7.2 Security-конфигурация.....	14
2.7.3 JWT-токены и аутентификация .....	15
3 Frontend-часть .....	17
3.1 Основной стэк.....	17
3.2 Основной функционал.....	17
3.3 Защита эндпоинтов .....	17
3.4 Главная страница.....	17
3.5 Страницы входа/регистрации .....	18
3.6 Страница объявления.....	18
3.7 Страница создания объявления .....	19
3.8 Страница «О нас».....	19
4 Docker-контейнеризация .....	21
4.1 Контейнер для БД .....	21
4.2 Minio контейнер .....	21

4.3 Упаковка backend-приложения в docker-контейнер.....	23
4.4 Упаковка frontend приложения.....	24
5 Итоги работы .....	25
Заключение .....	26
Приложение А .....	27

## ВВЕДЕНИЕ

Любой проект как правило должен начинаться с проблемы. Мы выявили, что не существует специализированных платформ для покупки и продажи фермерского мяса, где каждому можно выложить свое объявление. Однако спрос на такие сервисы есть, и часто сельский человек не может найти себе покупателей, которых приходится искать через знакомых, прямого выхода на городской рынок как правило нет, продажная цена для фермера зачастую занижена и невыгодна, особенно если он продает через мясных комерсантов или других посредников. Любой потребитель мяса, хочет покупать его натуральным, зная на чем он выращен и за как можно низкую цену.

Наш проект предлагает решение в виде платформы для покупки и продажи мяса. Наша цель – связать покупателя и продавца мясом.

Исходнику проекта лежат в соответствующих ссылках:

Репозиторий backend: <https://github.com/RaketaBoom/meatway-backend>

Репозиторий frontend: <https://github.com/Parsiifal/meatway-frontend>

## **1 Концепция проекта**

### **1.1 Назначение проекта**

Платформа для покупки и продажи мяса разных видов через удобный интерфейс сайта. Наша цель связать продавцов с их покупателями.

### **1.2 Фреймворки и технологии**

#### **Backend:**

1. Java – основной язык, на котором написан бекенд приложения. Очень популярен для бекенда, большое сообщество программистов, работающих на этом языке, постоянная поддержка и обновления со стороны разработчиков языка.

2. Spring Framework – фреймворк для написания бизнес-приложений на Java. Постоянно поддерживается авторами. Очень гибкий в настройке. Удобный контейнер внедрения зависимостей. Очень крупный фреймворк, включающий множество модулей под разные задачи. В проекте мы использовали Spring Boot, Spring Data JPA, Spring Security, Spring MVC

3. Hibernate – ORM-библиотека, позволяющая удобно взаимодействовать с БД через маппинг записей таблицы в объект Java, есть недостатки с оптимизацией и «магией», которую не любят опытные программисты (генерация SQL-запросов, вместо прописывания программистом), потому что можно поймать проблему “n+1” при соединении нескольких таблиц. Однако для нашего ненагруженного проекта, вполне подходящий инструмент.

4. PostgreSQL – реляционная СУБД для хранения данных нашего приложения. Очень популярная, постоянная поддержка со стороны авторов.

5. Liquibase – инструмент для создания миграций на БД

#### **Frontend:**

TypeScript – основной язык для разработки фронтенда и бекенда (в рамках Next.js). Это надстройка над JavaScript, добавляющая строгую типизацию, что уменьшает количество ошибок и улучшает поддержку кода. TypeScript активно развивается Microsoft и имеет огромное сообщество.

Next.js – React-фреймворк для создания полноценных веб-приложений с поддержкой SSR (Server-Side Rendering) и API-роутов. Позволяет писать фронтенд и бекенд в одном проекте. Поддерживается Vercel, имеет встроенный роутинг, оптимизацию изображений и удобную систему сборки. В проекте мы используем Next.js Middleware – для обработки запросов на уровне edge-сети.

Minio – объектное хранилище, совместимое с Amazon S3 API. Позволяет хранить и управлять файлами (изображения, документы и т. д.). Легко разворачивается в облаке или локально, поддерживает репликацию и шифрование. В нашем проекте Minio используется для хранения пользовательских загружаемых файлов.

### **1.3 Основные сущности**

Сущности хранятся в БД в виде записей в таблицах.

Вот основные сущности:

Пользователь – каждый, кто посещает наш сайт.

Объявления (говядина, баранина, птица, свинина, другое) – объявления о продаже мяса. Их может выложить только авторизованный пользователь.

Заказы – возможность забронировать объявления с мясом (реализация в будущем развитии проекта)

### **1.4 Функциональные требования**

Система должна создавать объявление по требованию пользователя.

Система должна возвращать объявление по требованию пользователя.

Система должна регистрировать и аутентифицировать пользователя через нашу платформу.

Пользователь должен иметь право изменить свое объявление.

Пользователь должен иметь право менять свой профиль.

Система должна запрещать доступ к изменению чужих профилей и объявлений

## 2 Backend-часть

### 2.1 Архитектура БД

Для реализации всех сущностей мы спроектировали следующую архитектуру БД (рисунок 1).

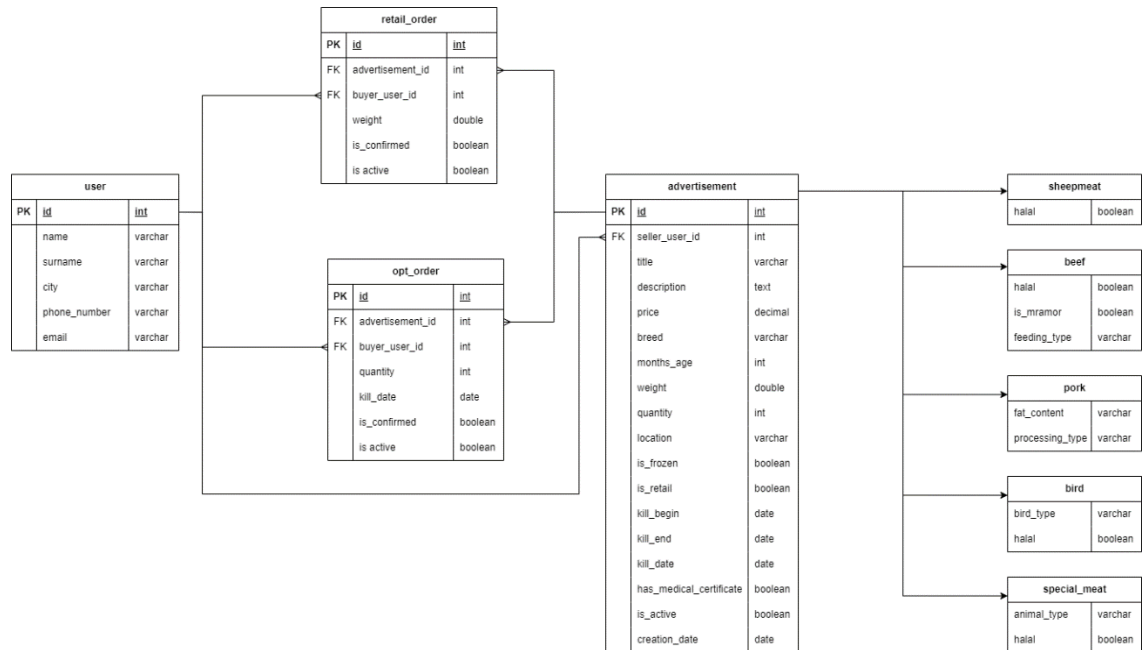


Рисунок 1 – Архитектура БД

user – хранит информацию о профиле пользователя

advertisement – таблица для объявлений. Содержит общие поля всех объявлений

beef, sheepmeat, pork, bird, special\_meat – унаследованные от advertisement

retail\_order, opt\_order – таблицы для хранения розничных и оптовых заказов

### 2.2 OpenApi-документация

Для описания всевозможных end-points нашего приложения мы сгенерировали swagger api документацию, которая содержит всю необходимую информацию о контрактах end points.

### user-controller API Профилей пользователей

GET	/api/v1/users	Получить всех пользователей
POST	/api/v1/users	Создать пользователя по id
GET	/api/v1/users/{id}	Получить пользователя по id
DELETE	/api/v1/users/{id}	Внести изменения в профиль пользователя
PATCH	/api/v1/users/{id}	Внести изменения в профиль пользователя

Рисунок 2 – api документация по ручкам пользователя

### pork-controller API Объявлений свинины

GET	/api/v1/porks/{id}	Вывести информацию объявления со свининой
POST	/api/v1/porks/{id}	Создать объявление со свининой
DELETE	/api/v1/porks/{id}	Удалить объявление со свининой
PATCH	/api/v1/porks/{id}	Редактировать объявление со свининой
GET	/api/v1/porks	Вывести краткую информацию об объявлениях со свининой

### ad-controller API Объявлений

GET	/api/v1/ads	Вывести краткую информацию об объявлениях
-----	-------------	---

### specialmeat-controller API Объявлений специального мяса

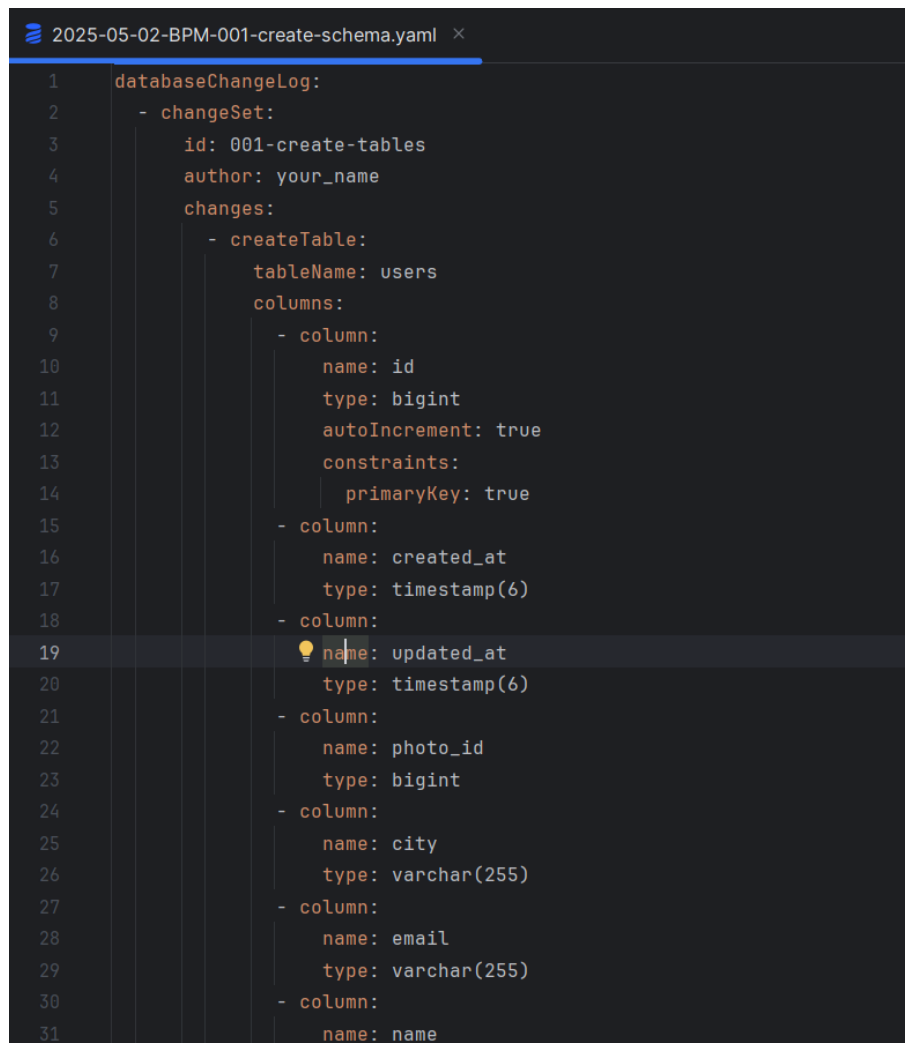
GET	/api/v1/specialmeats/{id}	Вывести информацию объявления со специальным мясом
POST	/api/v1/specialmeats/{id}	Создать объявление со специальным мясом
DELETE	/api/v1/specialmeats/{id}	Удалить объявление со специальным мясом
PATCH	/api/v1/specialmeats/{id}	Редактировать объявление со специальным мясом
GET	/api/v1/specialmeats	Вывести краткую информацию об объявлениях со специальным мясом

Рисунок 3 - api документация по ручкам объявления



## 2.3 Миграции

Для создания таблиц в БД и проверки, что такие таблицы есть, нужно накатывать миграции. В качестве инструмента для запуска миграций мы решили использовать liquibase. Мы подключили liquibase как зависимость проекта spring, вернее как стартер spring. При каждом запуске приложения «прогоняются» миграции, сверяется чек-сумма. На рисунке 4 пример файла миграции для создания всех таблиц.



```
2025-05-02-BPM-001-create-schema.yaml x
1  databaseChangeLog:
2    - changeSet:
3      id: 001-create-tables
4      author: your_name
5      changes:
6        - createTable:
7          tableName: users
8          columns:
9            - column:
10              name: id
11              type: bigint
12              autoIncrement: true
13              constraints:
14                primaryKey: true
15            - column:
16              name: created_at
17              type: timestamp(6)
18            - column:
19              name: updated_at
20              type: timestamp(6)
21            - column:
22              name: photo_id
23              type: bigint
24            - column:
25              name: city
26              type: varchar(255)
27            - column:
28              name: email
29              type: varchar(255)
30            - column:
31              name: name
```

Рисунок 4 – Миграции для создания таблиц в БД

```

1 databaseChangeLog:
2   changeSet:
3     id: 002-add-foreign-keys
4     author: your_name
5     changes:
6       - addForeignKeyConstraint:
7         baseTableName: advertisement
8         baseColumnNames: seller_user_id
9         constraintName: fk_advertisement_seller_user
10        referencedTableName: users
11        referencedColumnNames: id
12
13      - addForeignKeyConstraint:
14        baseTableName: advertisement_files
15        baseColumnNames: files_id
16        constraintName: fk_advertisement_files_file
17        referencedTableName: file
18        referencedColumnNames: id
19
20      - addForeignKeyConstraint:
21        baseTableName: advertisement_files
22        baseColumnNames: advertisement_id
23        constraintName: fk_advertisement_files_advertisement
24        referencedTableName: advertisement
25        referencedColumnNames: id
26
27      - addForeignKeyConstraint:
28        baseTableName: opt_order
29        baseColumnNames: advertisement_id
30        constraintName: fk_opt_order_advertisement
31        referencedTableName: advertisement
32        referencedColumnNames: id

```

Рисунок 5 – Миграции для создания ограничений и внешних ключей БД

## 2.4 Entity-модели

Для работы Spring Data JPA и маппинга записей таблицы БД в объекты Java. Необходимы Entity-модели.

В приложении А показан код entity-моделей для некоторых видов объявлений. Все виды объявления (beef, pork, sheepmeat, specialmeat, bird) наследуются от модели Advertisement. Обратите внимание т.к. используется стратегия наследования SINGLE\_TABLE, то все сущности разных типов будут храниться в одной таблице. И каждое поле каждой сущности будет содержаться в advertisement. Однако ORM будет обеспечивать однозначный маппинг в нужную модель по значению поля ad\_type.

## 2.5 Dto-объекты

Data transfer object – объекты, которые позволяют задать контракт для компонентов системы. Мы в нашем проекте стремились к гексогональной архитектуре. Т.е. каждый компонент имеет свои отдельные dto и изменение контракта одного компонента, не требует переписывания половины проекта, а займет только изменение конкретного dto.

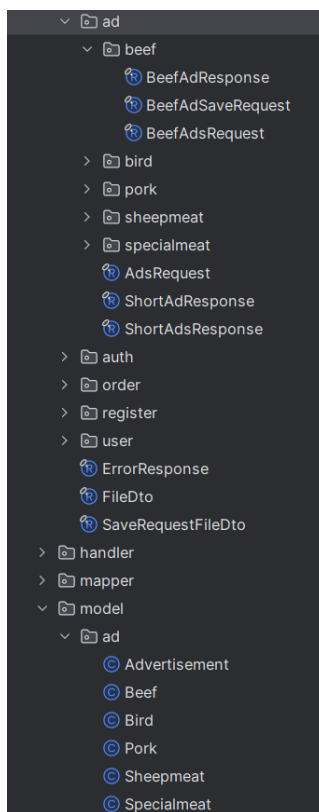


Рисунок 6 – dto в нашем проекте

```

public record PorkAdResponse( 16 usages  🧑 RaketaBoom
    Long id, 1 usage
    String meatType, 1 usage
    String title, 1 usage
    String description, 1 usage
    BigDecimal price, 1 usage
    String breed, 1 usage
    Integer monthsAge, 1 usage
    Integer weight, 2 usages
    Integer quantity, 1 usage
    String location, 1 usage
    Boolean isFrozen, 1 usage
    Boolean isRetail, 1 usage
    Date dateBegin, no usages
    Date dateEnd, no usages
    Date killDate, 1 usage
    Boolean hasMedicalCertificate, 1 usage
    Boolean isActive, 1 usage
    Date creationDate, 1 usage
    List<FileDto> files, 1 usage
    String fatContent, 1 usage
    String processingType, 1 usage
    UserProfileResponse sellerUser 1 usage
) {
}

```

Рисунок 7 – Пример dto PorkAdResponse

## 2.6 Реализация контроллера

Для доступа к end point извне по URL необходимо реализовать контроллер с указанием нужного метода запроса, тела запроса и URL.

Пример одного из наших контроллеров изображен на рисунке 8.

```

@Slf4j  ▲ RaketaBoom
@RestController
@Tag(name = AdvertisementController.AD_CONTROLLER, description = "API объявлений")
@RequestMapping(AdvertisementController.API_AD)
@RequiredArgsConstructor
public class AdvertisementController {

    public static final String AD_CONTROLLER = "ad-controller"; 2 usages
    public static final String ADS_PREFIX = "/ads"; 6 usages
    static final String API_VERSION = "v1"; 1 usage
    static final String API_PREFIX = "/api/" + API_VERSION; 1 usage
    public static final String API_AD = API_PREFIX + ADS_PREFIX; 1 usage

    private final AdService adService;

    @GetMapping  ▲ RaketaBoom
    @ResponseStatus(HttpStatus.OK)
    @Operation(
        summary = "Вывести краткую информацию об объявлениях",
        tags = {AD_CONTROLLER}
    )
    public ShortAdsResponse findAll(
        @RequestParam(required = false) Integer page,
        @RequestParam(required = false) Integer size,
        @RequestParam(required = false) String sort,
        @RequestBody(required = false) AdsRequest request
    ) {
        return adService.findAll(page, size, sort, request);
    }
}

```

Рисунок 8 – AdvertisementController

## 2.7 Security

На проекте мы использовали Spring security.

### 2.7.1 Public и private rsa-ключи

Для генерации JWT-токенов нужен private rsa ключ. Этот ключ не должен никак попасть в руки третьих лиц, ведь при помощи него можно подделать токены и получить доступ к аккаунтам сервиса.

Public ключ нужен, чтобы расшифровывать payload информацию токена. Он позволяет получить информацию, которую несет в себе токен. Потеря этого ключа не несет опасности, потому что он не участвует в генерации токена и подделать секреты для пользователя невозможно.

В рамках лаб мы на проекте храним в отдельно resource-папке certs (рисунок снизу). Для боевого деплоя проекта на хост, нужен другой подход для

хранения ключей. Возможно создать защищенное vault-хранилище, как это сделано в некоторых IT-компаниях.

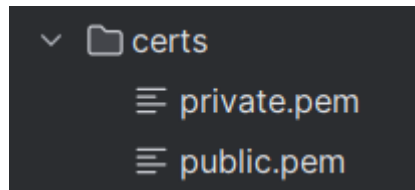


Рисунок 9 – Private и public rsa-ключи хранятся в certs-папке

### 2.7.2 Security-конфигурация

Также написана security конфигурация проекта, можете видеть, что все endpoint на проекте требуют аутентификации пользователя, кроме исключений, представленных на изображении ниже.

```
public class SecurityConfig {
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http
            .csrf(AbstractHttpConfigurer::disable)
            .cors(CorsConfigurer<HttpSecurity> cors -> cors.configurationSource(corsConfigurationSource()))
            .logout(AbstractHttpConfigurer::disable)
            .formLogin(AbstractHttpConfigurer::disable)
            .headers(AbstractHttpConfigurer::disable)
            .authorizeHttpRequests(AuthorizationManagerRequestMat... auth -> auth
                .requestMatchers(
                    "/swagger-ui.html",
                    "/swagger-ui/**",
                    "/v3/api-docs/**",
                    "/swagger-resources/**",
                    "/webjars/**",
                    "/actuator/**",
                    "/api/v1/auth/**",
                    "/api/v1/test/**",
                    "/api/v1/register",
                    "/api/v1/actuator/**"
                ).permitAll()
                .requestMatchers(HttpMethod.GET,
                    "/api/v1/ads/**",
                    "/api/v1/orders/**",
                    "/api/v1/users/**"
                ).permitAll()
                .anyRequest().authenticated()
            )
    }
}
```

Рисунок 10 – Конфигурация Spring Security

Пример регистрации пользователя:



Для защиты от изменения данных одного пользователя другим пользователем, необходимо идентифицировать того, кто дергает end-point. Мы решили эту проблему через идентификацию по токену. Мы из него получаем username пользователя и уже по нему изменяем необходимые данные:

```
@PostMapping  🌐  📌 RaketaBoom
@ResponseStatus(HttpStatus.CREATED)
@Operation(
    summary = "Создать объявление с говядиной",
    tags = {BEEF_CONTROLLER}
)
public BeefAdResponse createBeef(
    @RequestBody BeefAdSaveRequest request,
    @AuthenticationPrincipal Jwt jwt
) {
    return beefService.createBeefAd(request, jwtUtils.extractUsername(jwt.getTokenValue()));
}
```

Рисунок 12 – Получение username по токену



### **3 Frontend-часть**

#### **3.1 Основной стек**

Фронтэнд написан на React-фреймворке Next.js на языке TypeScript. Реализована загрузка и хранение пользовательских файлов в S3 хранилище.

#### **3.2 Основной функционал**

Основной функционал проекта заключается в возможности регистрации пользователей, возможности заполнения и изменения данных пользователя, в том числе аватарки, возможности просмотра как всех объявлений на главной странице с краткой информацией, так и отдельных объявлений на отдельной странице, возможности создания объявлений для всех типов мяса.

#### **3.3 Защита эндпоинтов**

Защита эндпоинтов реализована с помощью Middleware, который ограничивает доступ незарегистрированным пользователям к главной странице, странице аккаунта, странице просмотра и создания объявления. При попытке зайти на эти страницы незарегистрированного пользователя перенаправляет на страницу входа в аккаунт. Код представлен в приложении Б.

#### **3.4 Главная страница**

Код основных компонентов главной страницы представлен в приложении Б. На рисунке 13 продемонстрирована главная страница с загруженными с бекенда объявлениями.

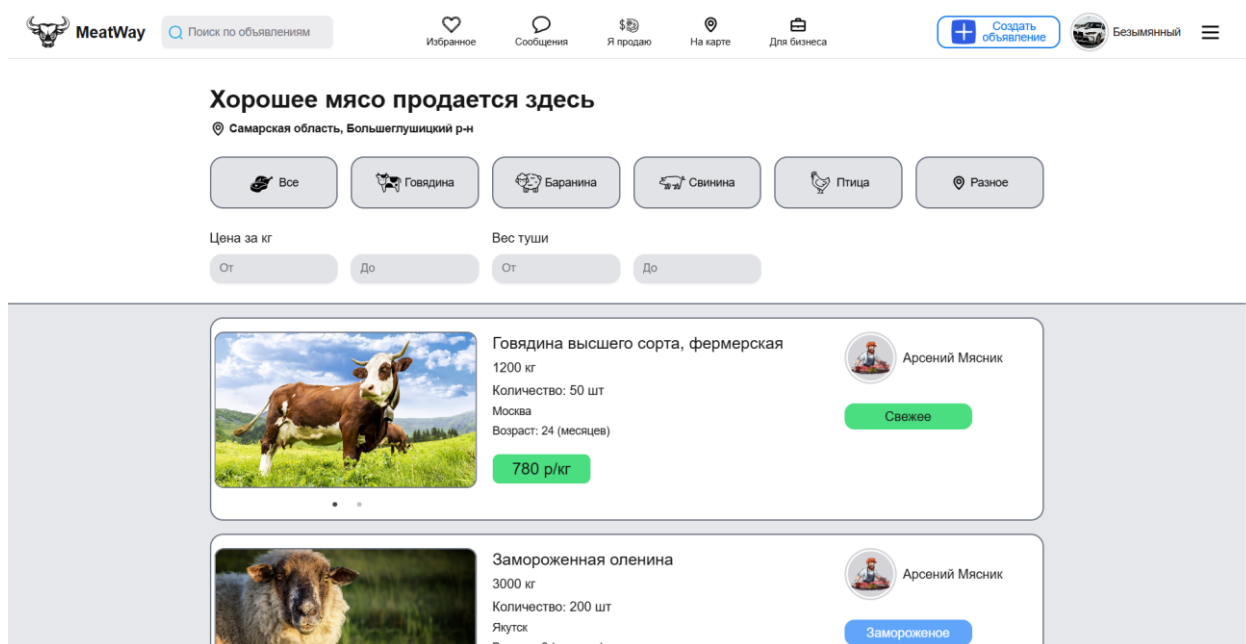


Рисунок 13 – Главная страница

### 3.5 Страницы входа/регистрации

На рисунке 14 представлены страницы входа и регистрации аккаунта.

Рисунок 14 – Страницы входа/регистрации

Все валидации реализованы с использованием библиотеки Zod и не позволяют отправить на сервер некорректные данные.

### 3.6 Страница объявления

На рисунке 15 продемонстрирована страница просмотра объявления.

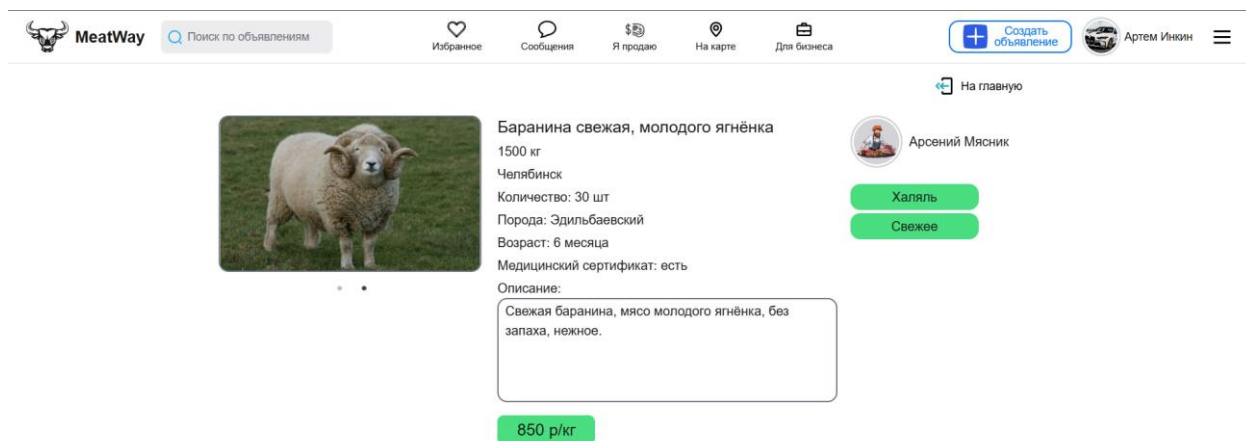


Рисунок 15 – Страница просмотра объявления

### 3.7 Страница создания объявления

На рисунке 16 продемонстрирована страница создания объявления.

The screenshot shows the 'Заполните объявление' (Fill out the ad) form on the MeatWay website. The form includes the following fields and options: 'Название' (Name), 'Тип мяса' (Type of meat) with a dropdown menu, 'Вес в кг' (Weight in kg), 'Местоположение' (Location), three checkboxes for 'Наличие медицинского сертификата' (Medical certificate), 'Замороженное' (Frozen), and 'В розницу' (Wholesale), 'Количество' (Quantity), 'Порода' (Breed), 'Возраст (месяцев)' (Age in months), 'Описание:' (Description) with a text area, and 'Цена за кг' (Price per kg). A 'Выбрать фото' (Choose photo) button is located at the bottom of the form. A 'На главную' (Home) link is visible at the top right of the form area.

Рисунок 16 – Страница создания объявления

### 3.8 Страница «О нас»

На рисунке 17 продемонстрирована страница «О нас».

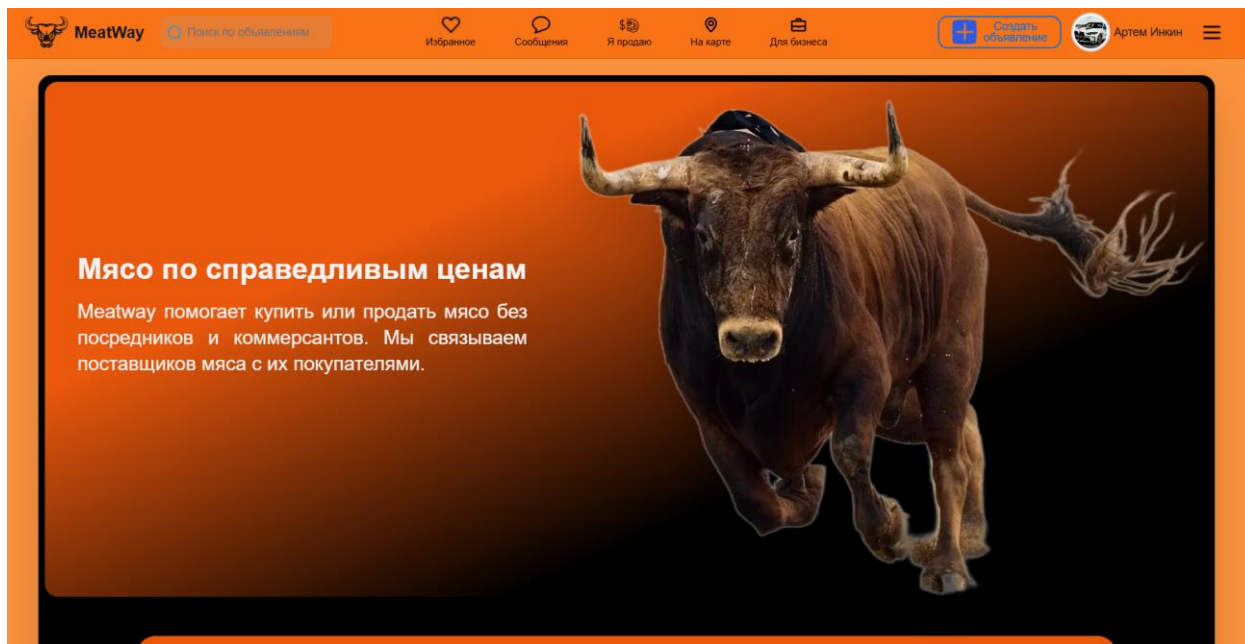


Рисунок 17 – Страница «О нас»

## 4 Docker-контейнеризация

После написания frontend и backend части, нужно придумать способ удобно запускать весь проект. Для деплоя проекта есть инструмент kubernetes, он больше подходит, когда проект размещают в нескольких подах на разных серверах для повышения отказоустойчивости. Однако на данном этапе на достаточно просто собирать проект в контейнерах на одном устройстве. Для этого подойдет Docker.

Docker – инструмент для контейнеризации приложений, позволяет гибко конфигурировать взаимодействие между контейнерами.

Нам необходимо было упаковать в контейнеры следующие компоненты:

- БД
- Minio – S3 хранилище для файлов
- Backend-приложение
- Frontend-приложение

### 4.1 Контейнер для БД

Для создания контейнера с БД мы использовали Docker-образ “postgres:17”. Конфигурация контейнера представлена на рисунке 18.

```
db:
  profiles:
    - dev
    - debug
  image: postgres:17
  container_name: meatway-db
  restart: unless-stopped
  env_file: .env
  environment:
    POSTGRES_USER: ${POSTGRES_USER}
    POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    POSTGRES_DB: ${POSTGRES_DB}
  ports:
    - "5433:5432"
  healthcheck:
    test: [ "CMD-SHELL", "pg_isready -U ${POSTGRES_USER} -d ${POSTGRES_DB}" ]
    interval: 3s
    timeout: 5s
  volumes:
    - ./meatway-backend/src/main/resources/schema_init.sql:/docker-entrypoint-initdb.d/init.sql:ro
  networks:
    - internal
```

Рисунок 18 – Конфигурация контейнера с БД

### 4.2 Minio контейнер

Для работы minio мы создали 2 контейнера:

1. **minio** – основной контейнер, в котором размещается s3-хранилище (рисунок 19);
2. **minio-init** – контейнер инициализации, который запускается после успешного старта основного minio контейнера и bucket в нем для хранения файлов, а затем отключается (рисунок 19).

```
minio:
  image: minio/minio
  container_name: meatway-minio
  hostname: minio
  ports:
    - "9000:9000"
    - "9001:9001"
  volumes:
    - minio_data:/data
  environment:
    - MINIO_ROOT_USER=minioadmin
    - MINIO_ROOT_PASSWORD=minioadmin
  command: server /data --console-address ":9001"
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:9000/minio/health/live"]
    interval: 30s
    timeout: 20s
    retries: 3
  networks:
    - internal
  restart: unless-stopped

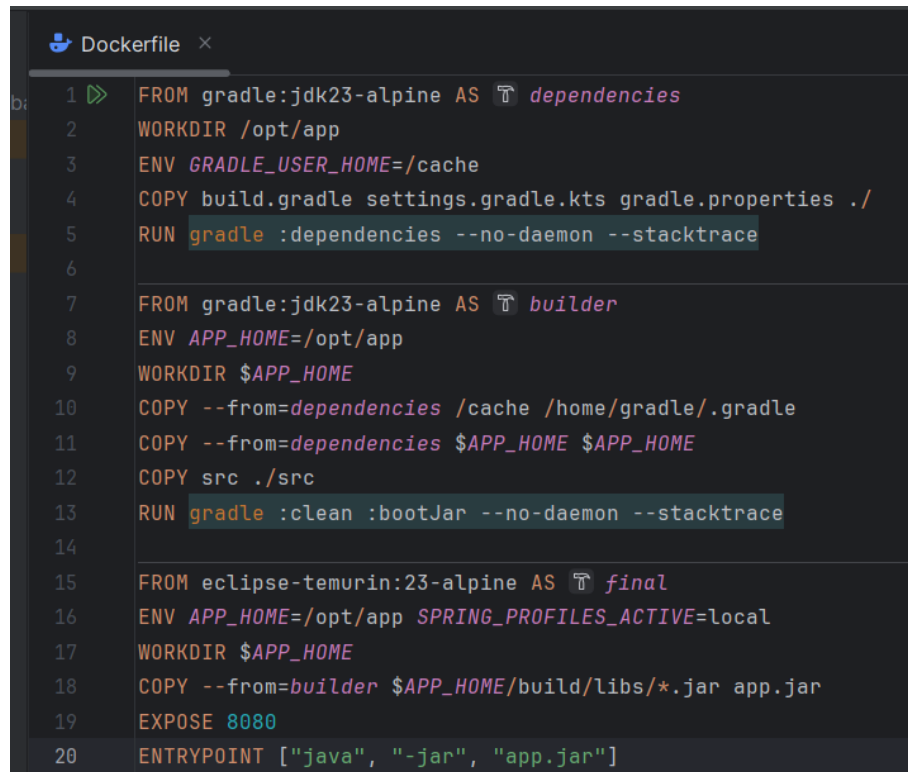
minio-init:
  image: minio/mc
  depends_on:
    minio:
      condition: service_healthy
  entrypoint: >
    sh -c "
      until mc alias set minio http://minio:9000 minioadmin minioadmin; do
        echo 'Waiting for MinIO...';
        sleep 2;
      done;
      mc mb minio/meatway-bucket --ignore-existing;
      mc anonymous set public minio/meatway-bucket;
      echo 'MinIO bucket initialized';
      # Настройка CORS
      mc admin config set minio cors 'config=[{
        \"allowedOrigins\": [\"*\"],
        \"allowedMethods\": [\"GET\", \"POST\", \"PUT\", \"DELETE\", \"HEAD\"],
        \"allowedHeaders\": [\"*\"]}]';
      mc admin service restart minio;
      echo 'MinIO initialized with CORS';
    "
  networks:
    - internal
```

Рисунок 19 – Контейнеры Minio

### 4.3 Упаковка backend-приложения в docker-контейнер

Чтобы указать наше приложение в docker-compose, необходимо сначала собрать образ. Dockerfile – это набор инструкций для создания образа. Мы разбили Dockerfile на несколько этапов сборки (рисунок 20):

1. Скачивание всех зависимостей проекта для Gradle
2. Упаковка приложения в JAR-файл
3. Копирование JAR-файла в контейнер и открытие порта 8080



```
Dockerfile
1 FROM gradle:jdk23-alpine AS dependencies
2 WORKDIR /opt/app
3 ENV GRADLE_USER_HOME=/cache
4 COPY build.gradle settings.gradle.kts gradle.properties ./
5 RUN gradle :dependencies --no-daemon --stacktrace
6
7 FROM gradle:jdk23-alpine AS builder
8 ENV APP_HOME=/opt/app
9 WORKDIR $APP_HOME
10 COPY --from=dependencies /cache /home/gradle/.gradle
11 COPY --from=dependencies $APP_HOME $APP_HOME
12 COPY src ./src
13 RUN gradle :clean :bootJar --no-daemon --stacktrace
14
15 FROM eclipse-temurin:23-alpine AS final
16 ENV APP_HOME=/opt/app SPRING_PROFILES_ACTIVE=local
17 WORKDIR $APP_HOME
18 COPY --from=builder $APP_HOME/build/libs/*.jar app.jar
19 EXPOSE 8080
20 ENTRYPOINT ["java", "-jar", "app.jar"]
```

Рисунок 20 – Dockerfile нашего backend-приложения

```

26 meatway-api:
27   profiles:
28     - dev
29   build:
30     context: .
31     dockerfile: Dockerfile
32
33   container_name: meatway-api
34   depends_on:
35     db:
36       condition: service_healthy
37   environment:
38     SPRING_PROFILES_ACTIVE: dev
39     DB_URL: ${DB_URL}
40     DB_USERNAME: ${POSTGRES_USER}
41     DB_PASSWORD: ${POSTGRES_PASSWORD}
42   ports:
43     - "8080:8080"
44   networks:
45     - internal

```

Рисунок 21 – конфигурация контейнера backend-приложения

#### 4.4 Упаковка frontend приложения

На рисунке 22 представлена конфигурация контейнера фронтенда.

```

frontend:
  build:
    context: ./meatway-frontend
    dockerfile: Dockerfile
  container_name: meatway-frontend
  ports:
    - "3000:3000"
  environment:
    - REACT_APP_API_URL=http://localhost:8080 # URL для браузера
    - NEXT_PUBLIC_API_URL=http://meatway-api:8080 # URL для SSR (если Next.js)
    # Добавьте эти переменные:
    - MINIO_PUBLIC_URL=http://minio:9000 # для внутренних запросов
    - MINIO_BROWSER_URL=http://minio:9000 # для браузера
    - MINIO_ENDPOINT=minio
    - MINIO_PORT=9000
    - MINIO_BUCKET_NAME=meatway-bucket
    - MINIO_USE_SSL=false
    - MINIO_ROOT_USER=minioadmin
    - MINIO_ROOT_PASSWORD=minioadmin
  depends_on:
    minio:
      condition: service_healthy
  networks:
    - internal
  restart: unless-stopped

```



## **5 Итоги работы**

Таким образом у нас есть реализованное веб приложение, позволяющее создавать, редактировать, удалять объявления. Реализована авторизация и защита данных аккаунтов. Также реализована удобная фильтрация и сортировка объявлений.

## ЗАКЛЮЧЕНИЕ

Благодаря проделанной работе за семестр каждый из нас изучил на практике инструменты для создания веб-приложений со стороны backend и frontend. Также это позволило нам реализовать наши идеи и технические решения, которые мы принимали, когда сталкивались с проблемами. Неправильные решения вынуждали переделывать проект, и полностью погружаться в суть проблемы и возможных решений, что дало нам опыт и знание о подводных камнях тех или иных подходов.

Подробнее с кодом вы можете ознакомиться по ссылкам на наши репозитории:

Репозиторий backend: <https://github.com/RaketaBoom/meatway-backend>

Репозиторий frontend: <https://github.com/Parsiifal/meatway-frontend>

## ПРИЛОЖЕНИЕ А

### Код бекенд части

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="ad_type",
    discriminatorType = DiscriminatorType.STRING)
@Getter
public class Advertisement {
    @Id
    private Long id;

    private String title;

    private String description;

    private BigDecimal price;

    private String breed;

    private Integer monthsAge;

    private Double weight;

    private Integer quantity;

    private String location;

    private Boolean isFrozen;

    private Boolean isRetail;

    private Date killBegin;

    private Date killEnd;

    private Date killDate;

    private Boolean hasMedicalCertificate;

    private Boolean isActive;

    private Date creationDate;

    @OneToMany
    @JoinColumn(name = "advertisement_id")
    private List<RetailOrder> retailOrders;

    @OneToMany
    @JoinColumn(name = "advertisement_id")
    private List<OptOrder> optOrders;
}

@Entity
@DiscriminatorValue("beef")
public class Beef extends Advertisement {
    private Boolean halal;
    private Boolean isMramor;
    private String feedingType;
}
```

```

        @OneToMany
        @JoinColumn(name = "advertisement_id")
        private List<RetailOrder> retailOrders;
    }

    @Entity
    @DiscriminatorValue("bird")
    @Getter
    @Setter
    public class Bird extends Advertisement {
        private String birdType;
    }

    @OneToMany
    @JoinColumn(name = "advertisement_id")
    private List<OptOrder> optOrders;
}

@Entity
@DiscriminatorValue("beef")
public class Beef extends Advertisement {
    private Boolean halal;
    private Boolean isMramor;
    private String feedingType;
}

@Entity
@DiscriminatorValue("pork")
@Getter
@Setter
public class Pork extends Advertisement {
    private String fatContent;

    private String processingType;
}

@Slf4j
@RestController
@Tag(name = AdvertisementController.AD_CONTROLLER, description = "API
Объявлений")
@RequestMapping(AdvertisementController.API_AD)
@RequiredArgsConstructor
public class AdvertisementController {
    public static final String AD_CONTROLLER = "ad-controller";
    public static final String ADS_PREFIX = "/ads";
    static final String API_VERSION = "v1";
    static final String API_PREFIX = "/api/" + API_VERSION;
    public static final String API_AD = API_PREFIX + ADS_PREFIX;

    private final AdService adService;

    @GetMapping
    @ResponseStatus(HttpStatus.OK)
    @Operation(
        summary = "Вывести краткую информацию об объявлениях",
        tags = {AD_CONTROLLER}
    )
    public ShortAdsResponse findAll(
        @RequestParam(required = false) Integer page,
        @RequestParam(required = false) Integer size,
        @RequestParam(required = false) String sort,
        @RequestBody(required = false) AdsRequest request
    ) {

```

```

        return adService.findAll(page, size, sort, request);
    }
}

@Slf4j
@RestController
@Tag(name = BeefController.BEEF_CONTROLLER, description = "API Объявлений говядины")
@RequestMapping(BeefController.API_AD)
@RequiredArgsConstructor
public class BeefController {
    public static final String BEEF_CONTROLLER = "beef-controller";
    static final String API_VERSION = "v1";
    static final String API_PREFIX = "/api/" + API_VERSION;
    public static final String API_AD = API_PREFIX + AdvertisementController.ADS_PREFIX + "/beefs";

    private final BeefService beefService;
    private final JWTUtils jwtUtils;

    @GetMapping()
    @ResponseStatus(HttpStatus.OK)
    @Operation(
        summary = "Вывести краткую информацию об объявлениях с говядиной",
        tags = {BEEF_CONTROLLER}
    )
    public ShortAdsResponse findAll(
        @RequestParam(required = false) Integer page,
        @RequestParam(required = false) Integer size,
        @RequestParam(required = false) String sort,
        @RequestBody(required = false) BeefAdsRequest request
    ) {
        return beefService.findAll(page, size, sort, request);
    }

    @GetMapping("/{id}")
    @ResponseStatus(HttpStatus.OK)
    @Operation(
        summary = "Вывести информацию объявления с говядиной",
        tags = {BEEF_CONTROLLER}
    )
    public BeefAdResponse findById(@PathVariable int id) {
        return beefService.findById(id);
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    @Operation(
        summary = "Создать объявление с говядиной",
        tags = {BEEF_CONTROLLER}
    )
    public BeefAdResponse createBeef(
        @RequestBody BeefAdSaveRequest request,
        @AuthenticationPrincipal Jwt jwt
    ) {
        return beefService.createBeefAd(request, jwtUtils.extractUsername(jwt.getTokenValue()));
    }

    @PatchMapping("/{id}")
    @ResponseStatus(HttpStatus.OK)
    @Operation(

```

```

        summary = "Редактировать объявление с говядиной",
        tags = {BEEF_CONTROLLER}
    )
    public BeefAdResponse editById(
        @PathVariable int id,
        @RequestBody BeefAdSaveRequest request,
        @AuthenticationPrincipal Jwt jwt
    ) {
        return beefService.patchById(id, request, jwtUtils.extractUsername(jwt.getTokenValue()));
    }

    @DeleteMapping("/{id}")
    @ResponseStatus(HttpStatus.OK)
    @Operation(
        summary = "Удалить объявление с говядиной",
        tags = {BEEF_CONTROLLER}
    )
    public void deleteById(
        @PathVariable int id,
        @AuthenticationPrincipal Jwt jwt
    ) {
        beefService.deleteById(id, jwtUtils.extractUsername(jwt.getTokenValue()));
    }
}

@Configuration
@EnableConfigurationProperties(RsaKeyProperties.class)
@RequiredArgsConstructor
public class EncodersConfig {

    private final RsaKeyProperties rsaKeys;

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    JwtEncoder jwtEncoder() {
        return Stream.of(createJWK())
            .map(this::createJWKSource)
            .map(this::createJwtEncoder)
            .findFirst()
            .orElseThrow();
    }

    @Bean
    JwtDecoder jwtDecoder() {
        return NimbusJwtDecoder.withPublicKey(rsaKeys.publicKey()).build();
    }

    private JWK createJWK() {
        return new RSAKey.Builder(rsaKeys.publicKey())
            .privateKey(rsaKeys.privateKey())
            .build();
    }

    private JWKSource<SecurityContext> createJWKSource(JWK jwk) {
        return new ImmutableJWKSet<>(new JWKSet(jwk));
    }
}

```

```

        private JwtEncoder createJwtEncoder(JWKSource<SecurityContext> jwks) {
            return new NimbusJwtEncoder(jwks);
        }
    }

    @Configuration
    @EnableWebSecurity
    @RequiredArgsConstructor
    public class SecurityConfig {

        private final JwtDecoder jwtDecoder;
        private final PasswordEncoder passwordEncoder;
        private final UserService userDetailsService;

        @Bean
        public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
        Exception {

            return http
                .csrf(AbstractHttpConfigurer::disable)
                .cors(cors -> cors.configurationSource(corsConfiguration-
        Source()))
                .logout(AbstractHttpConfigurer::disable)
                .formLogin(AbstractHttpConfigurer::disable)
                .headers(AbstractHttpConfigurer::disable)
                .authorizeHttpRequests(auth -> auth
                    .requestMatchers(
                        "/swagger-ui.html",
                        "/swagger-ui/**",
                        "/v3/api-docs/**",
                        "/swagger-resources/**",
                        "/webjars/**",
                        "/actuator/**",

                        "/api/v1/auth/**",
                        "/api/v1/test/**",
                        "/api/v1/register",
                        "/api/v1/actuator/**"
                    ).permitAll()
                    .requestMatchers(HttpMethod.GET,
                        "/api/v1/ads/**",
                        "/api/v1/orders/**",
                        "/api/v1/users/**"
                    ).permitAll()

                    .anyRequest().authenticated()
                )
                .sessionManagement(session -> session
                    .sessionCreationPolicy(SessionCreationPolicy.STATE-
        LESS)
                )
                .oauth2ResourceServer(rs -> rs
                    .jwt(jwt -> jwt.decoder(jwtDecoder))
                )
                .build();
        }

        @Bean
        CorsConfigurationSource corsConfigurationSource() {
            CorsConfiguration configuration = new CorsConfiguration();
            configuration.setAllowedOrigins(List.of("http://localhost:3000"));
        }
    }

```

```

        configuration.setAllowedMethods(List.of("*"));
        configuration.setAllowedHeaders(List.of("*"));
        configuration.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfiguration-
Source();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }

    @Bean
    public AuthenticationManager authenticationManager(HttpSecurity http)
throws Exception {
        return http.getSharedObject(AuthenticationManagerBuilder.class)
            .build();
    }

    @Bean
    public AuthenticationProvider daoAuthProvider() {
        var provider = new DaoAuthenticationProvider();
        provider.setUserDetailsService(userDetailsService);
        provider.setPasswordEncoder(passwordEncoder);
        return provider;
    }
}

@Service
@RequiredArgsConstructor
public class AdService {
    private final static int DEFAULT_PAGE = 0;
    private final static int DEFAULT_SIZE = 10;
    private final static String DEFAULT_SORT = "id";

    private final AdvertisementRepository advertisementRepository;
    private final AdMapper adMapper;

    public ShortAdsResponse findAll(Integer page, Integer size, String sort,
AdsRequest request) {
        Specification<Advertisement> spec = getAdvertisementSpecification(re-
quest);

        if (sort == null || sort.isBlank()) {
            sort = DEFAULT_SORT;
        }

        Pageable pageable = PageRequest.of(
            Optional.ofNullable(page).orElse(DEFAULT_PAGE),
            Optional.ofNullable(size).orElse(DEFAULT_SIZE),
            Sort.by(sort)
        );

        Page<Advertisement> pageResult = advertisementRepository.findAll(spec,
pageable);

        return new ShortAdsResponse(
            adMapper.toListShortAdvertisementResponse(pageResult.getCon-
tent()),
            pageResult.getTotalElements()
        );
    }
}

```



```

        private static Specification<Advertisement> getAdvertisementSpecification(AdsRequest request) {
            Specification<Advertisement> spec = Specification.where(null);

            if (request == null) {
                return spec;
            }

            if (request.isRetail() != null) {
                spec = spec.and((root, query, cb) -> cb.equal(root.get("isRetail"),
re-quest.isRetail()));
            }
            if (request.medicalCertificate() != null) {
                spec = spec.and((root, query, cb) -> cb.equal(root.get("medicalCer-
tificate"), re-quest.medicalCertificate()));
            }
            if (request.isFrozen() != null) {
                spec = spec.and((root, query, cb) -> cb.equal(root.get("isFrozen"),
re-quest.isFrozen()));
            }
            if (request.priceFrom() != null) {
                spec = spec.and((root, query, cb) -> cb.greaterThanOrE-
qualTo(root.get("price"), re-quest.priceFrom()));
            }
            if (request.priceTo() != null) {
                spec = spec.and((root, query, cb) -> cb.lessThanOrE-
qualTo(root.get("price"), re-quest.priceTo()));
            }
            if (request.weightFrom() != null) {
                spec = spec.and((root, query, cb) -> cb.greaterThanOrE-
qualTo(root.get("weight"), re-quest.weightFrom()));
            }
            if (request.weightTo() != null) {
                spec = spec.and((root, query, cb) -> cb.lessThanOrE-
qualTo(root.get("weight"), re-quest.weightTo()));
            }
            if (request.quantityFrom() != null) {
                spec = spec.and((root, query, cb) -> cb.greaterThanOrE-
qualTo(root.get("quantity"), request.quantityFrom()));
            }
            if (request.quantityTo() != null) {
                spec = spec.and((root, query, cb) -> cb.lessThanOrE-
qualTo(root.get("quantity"), re-quest.quantityTo()));
            }
            if (request.monthsAgeFrom() != null) {
                spec = spec.and((root, query, cb) -> cb.greaterThanOrE-
qualTo(root.get("monthsAge"), request.monthsAgeFrom()));
            }
            if (request.monthsAgeTo() != null) {
                spec = spec.and((root, query, cb) -> cb.lessThanOrE-
qualTo(root.get("monthsAge"), re-quest.monthsAgeTo()));
            }
            if (request.dateBegin() != null) {
                spec = spec.and((root, query, cb) -> cb.greaterThanOrE-
qualTo(root.get("createdAt"), request.dateBegin()));
            }
            if (request.dateEnd() != null) {
                spec = spec.and((root, query, cb) -> cb.lessThanOrE-
qualTo(root.get("createdAt"), re-quest.dateEnd()));
            }
            return spec;
        }
    }

```

## ПРИЛОЖЕНИЕ Б

### Код фронтенд части

```
import { NextRequest, NextResponse } from "next/server";
import { decodeJwt } from "jose";

const protectedRoutes = ["/main", "/account", "/advertisement", "/create-ad"];

export default async function middleware(req: NextRequest) {

  const path = req.nextUrl.pathname;
  const token = req.cookies.get("token")?.value; // Получаем токен из кука

  // Для защищенных маршрутов
  if (protectedRoutes.some(route => path.startsWith(route))) {
    if (!token) {
      return NextResponse.redirect(new URL("/auth/login", req.nextUrl));
    }
    // Проверяем не протух ли токен
    const redirect = await isTokenOutdated(req, token);
    if (redirect) return redirect;
  }

  // Если пользователь аутентифицирован и пытается сделать это снова
  if (["/auth/login", "/auth/registration"].includes(path) && token) {
    // Проверяем не протух ли токен
    const redirect = await isTokenOutdated(req, token);
    if (redirect) return redirect;
    return NextResponse.redirect(new URL("/main", req.nextUrl));
  }

  return NextResponse.next();
}

// Проверяем токен на протухание
async function isTokenOutdated(req: NextRequest, token: string) {
  try {
    const payload = decodeJwt(token);
    if (payload.exp && Date.now() > payload.exp * 1000) { // токен протух
      const res = NextResponse.redirect(new URL("/auth/login", req.nextUrl));
      res.cookies.delete("token");
      return res;
    }
    return undefined;
  }
  catch (error) { // Если возникла какая-то ошибка
    const res = NextResponse.redirect(new URL("/auth/login", req.nextUrl));
    res.cookies.delete("token");
    return res;
  }
}

export const MainPage = () => {

  const [selectedType, setSelectedType] = useState<string>("all");
  const [ads, setAds] = useState<AdvertisementUnion[]>([]);
  const [error, setError] = useState<string | null>(null);
```

```

// Загрузить объявления конкретного типа (в том числе всех)
useEffect(() => {
  const fetchAds = async (type: string) => {
    try {
      setError(null); // Сбрасываем ошибки перед каждым новым запросом
      const { data, error } = await getAdvertisements(type);
      if (error) throw new Error(error);
      if (data) setAds(data);
    }
    catch (err) {
      console.error(err);
      setError(err instanceof Error ? err.message : "Ошибка загрузки объявлений!");
      setAds([]);
    }
    finally {
      console.log(selectedType);
    }
  };
  fetchAds(selectedType);
}, [selectedType]);

return (
  <div>
    {/* <GridDev/> */}

    <div className="w-4/5 max-w-screen-lg mx-auto">
      {/* Верхняя надпись и местоположение */}
      <TopContent/>
      {/* Кнопки выбора мяса */}
      <ButtonsMeatType onSelect={setSelectedType}/>
      {/* Фильтры */}
      <Filters/>
    </div>

    <div className="mt-6 bg-gray-200 min-h-[70vh] border-t-2 border-gray-500">
      <div className="w-4/5 max-w-screen-lg mx-auto">
        {/* Объявления */}
        <Advertisement advertisements={ads} error={error || undefined}/>
      </div>
    </div>
  </div>
)

```

```

        </div>

        </div>
    );
};

interface AdvertisementProps {
    advertisements: AdvertisementUnion[];
    error?: string;
}

export const Advertisement = ({ advertisements, error }: AdvertisementProps) =>
{

    const [avatars, setAvatars] = useState<Record<string, string>>({});
    const [isLoading, setIsLoading] = useState<boolean>(true);

    // Настройки карусели изображений
    const settings = {
        dots: true,
        infinite: true,
        speed: 500,
        slidesToShow: 1,
        slidesToScroll: 1,
        arrows: false,
    };

    // Получаем из minio аватарки владельцев объявлений
    useEffect(() => {
        advertisements.forEach((ad) => {
            if (!ad.id || !avatars[ad.id]) return;
            const avatar = ad.sellerUser?.photo?.path ?? "Default avatar.jpg";

            fetch(`/api/download?filename=${avatar}`)
                .then((res) =>
                    res.ok ? res.json() : new Error("File not found")
                )
                .then((avatarData) => {
                    setAvatars((prev) => ({
                        ...prev,
                        [ad.id!]: avatarData.data.url,
                    }));
                });
        });
    });
}

```

```

    })
    .catch(() => {
      setAvatars((prev) => ({
        ...prev,
        [ad.id!]: "Default avatar.jpg",
      }));
    });
  });
  setIsLoading(false);
}, [advertisements, avatars]);

// Перемешивание объявлений в случайном порядке
const shuffledAds = useМемо(() => {
  const arr = [...advertisements]; // создаём копию
  // Алгоритм Фишера-Йетса
  for (let i = arr.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1));
    [arr[i], arr[j]] = [arr[j], arr[i]];
  }
  return arr;
}, [advertisements]);

if (isLoading) {
  return (
    <><CustomSpinner mt={5}/></>
  );
}

if (error) {
  if(error === "Нет объявлений данного типа!") {
    return (
      <p className="mt-4 text-2xl text-center text-gray-500">Пока нет ни од-
ного объявления!</p>
    );
  }
  return (
    <div className="mt-4 p-4 bg-red-100 border border-red-400 text-red-700
rounded-lg">
      <p className="font-bold">Ошибка загрузки объявлений!</p>
      <p>{error}</p>
    </div>
  );
}

```

```

        <button
            className="mt-2 px-4 py-2 bg-red-500 text-white rounded hover:bg-red-
600"

            onClick={() => window.location.reload()}>
                Попробовать снова
        </button>
    </div>
    );
}

//console.log(avatars);
//console.log(advertisements);

const defaultUrlPath = "http://localhost:9000/meatway-bucket/";

return (
    <>
        {shuffledAds.map((ad) => (
            <div key={ad.id} className="mt-4 bg-white rounded-2xl border-2 border-
gray-500">

                <div className="gridLg">

                    {/* Изображения объявления */}
                    <div className="col-span-4 col-start-1 pl-1 pt-4">
                        {ad.files?.length != undefined && ad.files?.length > 1 ? (
                            // Слайдер для нескольких изображений
                            <div className="pb-8">
                                <Slider {...settings}>
                                    {ad.files?.map((file, index) => (
                                        <div key={` ${ad.id} - ${index} `} className="pr-1">
                                            <div className="h-48 w-full border-2 border-gray-500
rounded-xl overflow-hidden aspect-square relative">
                                                {/* src={defaultUrlPath + file.path} */}
                                                <Image
                                                    src={defaultUrlPath + file.path}
                                                    alt={`Изображение ${index + 1} - ${ad.title}`}
                                                    fill
                                                    sizes="(max-width: 768px) 100vw, 50vw"
                                                    className="object-cover object-center"

```

```

        quality={85}
        style={{
            borderRadius: "0.5rem"
        }}
        unoptimized={true} // Важное исправление!
    />
</div>
</div>
    )})
</Slider>
</div>
) : ad.files?.length === 1 ? (
// Одиночное изображение без слайдера
<div className="pr-1 pb-4">
    <div className="h-48 w-full border-2 border-gray-500 rounded-
xl overflow-hidden aspect-square relative">
        {/* src={defaultUrlPath + ad.files[0].path} */}
        <Image
            src={defaultUrlPath + ad.files[0].path}
            alt={`Единственное изображение - ${ad.title}`}
            fill
            sizes="(max-width: 768px) 100vw, 50vw"
            className="object-cover object-center"
            quality={85}
            style={{
                borderRadius: "0.5rem"
            }}
            unoptimized={true}
        />
    </div>
</div>
) : (
// Дефолтная картинка если нет изображений
<div className="pr-1 pb-4">
    <div className="h-48 w-full border-2 border-gray-500 rounded-
xl overflow-hidden aspect-square relative">
        {/* src={defaultUrlPath + "default-adv-image.jpg"} */}
        <Image
            src={defaultUrlPath + "default-adv-image.jpg"}
            alt="Изображение отсутствует"
            fill

```

```

        sizes="(max-width: 768px) 100vw, 50vw"
        className="object-cover object-center"
        quality={85}
        style={{
            borderRadius: "0.5rem"
        }}
        unoptimized={true}
    />
</div>
</div>
))
</div>

```

```

{ /* Информация объявления */ }
<div className="col-span-5 col-start-5 mt-4 mb-4">
    <Link href={` /advertisement/${ad.id}?meatType=${ad.meatType}`}
        className="text-xl">
        {ad.title}
    </Link>
    <p className="mt-1">{ad.weight || "Не указано"} кг</p>
    <p className="mt-1">Количество: {ad.quantity + " шт"}</p>
    <p className="mt-1 text-sm">{ad.location}</p>
    <p className="mt-1 text-sm">Возраст: {ad.monthsAge ? ad.monthsAge
+ " (месяцев)" : "не указано"}</p> { /* Возраст */ }
    <p className="mt-[18px] w-[120px] text-xl text-center bg-green-
400 p-1 rounded-lg">{ad.price} p/кг</p>
</div>

```

```

{ /* Владелец объявления и доп. информация */ }
<div className="col-start-10 col-end-13 grid grid-cols-subgrid gap-
x-4 content-start mt-4">
    <div className="col-span-3 flex flex-row items-center gap-x-3 p-
1">

```

{ /\* По какой то причине некоторые картинки могут отображаться в аватарах с плохим качеством.

Я потратил много времени, чтоб понять почему так, но результатов это не дало. И дело не в либе,

с другими либами также. Есть 2 предположения почему так может происходить:



1. Изменение/ограничение размеров компонента или картинки не стандартными либовскими средствами, что ломает рендер.

2. Сами изображения шакальные и некорректно рендерятся. \*/}

```
<Avatar isBordered size="lg" src={avatars[ad.id!]} />
<p
      className="text-md
      whitespace-
nowrap">`${ad.sellerUser?.name || "Безымянный"} ${ad.sellerUser?.surname || ""}`</p>
</div>
```

```
<div className="col-span-2 text-center mt-5">
  {/* Халяль */}
  {"isHalal" in ad ? ad.isHalal ? (<p className="bg-green-400 p-
1 rounded-xl">Халяль</p>) :
    (<p className="bg-red-400 text-white p-1 rounded-
xl">Халяль</p>) : <></>
  }
```

```
  {/* Мраморность */}
  {"isMramor" in ad ? ad.isMramor ? (<p className="bg-green-400
p-1 rounded-xl mt-1">Мраморное</p>) :
    (<p className="bg-red-400 text-white p-1 rounded-xl mt-
1">Мраморное</p>) : <></>
  }
```

```
  {/* Замороженое или свежее*/}
  {ad.isFrozen ? (<p className="bg-blue-400 text-white p-1
rounded-xl mt-1">Замороженое</p>) :
    (<p className="bg-green-400 p-1 rounded-xl mt-1">Свежее</p>)}
  }
</div>
</div>
```

```

  </div>
  </div>
  )})
</>
);
};
```

```
export const CreateAd = () => {
```

```
  const router = useRouter();
```

```
  const fileInputRef = useRef<HTMLInputElement>(null);
```

```

const [error, setError] = useState<string | null>(null);
const [meatType, setMeatType] = useState<string>("");
const [selectedFiles, setSelectedFiles] = useState<File[]>([]);

const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {
  event.preventDefault();
  setError(null);
  try {
    const formData = new FormData(event.currentTarget);
    const rawData = Object.fromEntries(formData.entries());
    // Нужно отпарсить все НЕ строковые значения, так как FormData собирает
их все в виде строк
    const baseData = {
      ...rawData,
      weight: parseInt(rawData.weight as string, 10),
      quantity: parseInt(rawData.quantity as string, 10),
      monthsAge: parseInt(rawData.monthsAge as string, 10),
      price: parseInt(rawData.price as string, 10),
      hasMedicalCertificate: rawData.hasMedicalCertificate === "true",
      isFrozen: rawData.isFrozen === "true",
      isRetail: rawData.isRetail === "true",
    };
    // Нужно вынести в отдельные данные, так как это индивидуальные поля для
некоторых типов
    const beefData = {
      isHalal: rawData.isHalal === "true",
      isMramor: rawData.isMramor === "true",
    };
    const sheepAndSpecialData = {
      isHalal: rawData.isHalal === "true",
    };

    // Объединяем данные для нужных типов
    let parsedData;
    if (meatType === "beef") parsedData = { ...baseData, ...beefData };
    else if (meatType === "sheepmeat" || meatType === "specialmeat")
parsedData = { ...baseData, ...sheepAndSpecialData };
    else parsedData = baseData;

    let uploadedFileNames: string[] = [];
    // Загружаем файлы в Minio, если они есть

```

```

if (selectedFiles.length > 0) {
  const filesData = new FormData();
  selectedFiles.forEach(file => {filesData.append("files", file)});

  const response = await fetch("/api/upload", {
    method: "POST",
    body: filesData,
  });
  if (!response.ok) throw new Error("Ошибка загрузки файлов в храни-
лище!");

  const result: UploadResponse = await response.json();

  if (result.status === "success" && result.data) {
    uploadedFileNames = result.data.map(file => file.fileName);
  }
  else {
    throw new Error("Не удалось загрузить фото объявления!");
  }
}

// Собираем только имена файлов в FilePath[]
const filePaths: FilePath[] = uploadedFileNames.map(name => ({ path: name,
}));

// Включаем files в данные
parsedData = { ...parsedData, files: filePaths };

//console.log(Object.getPrototypeOf(parsedData) === Object.prototype);

//const ad = JSON.parse(json) as AdvertisementUnion;
//console.warn(ad);

const json = JSON.stringify(parsedData);
console.warn(json);
const success = await createAd(json, meatType);
if (success) {
  addToast({ title: "Объявление создано", color: "primary", variant:
"solid", timeout: 3500,
    classNames: { base: "mt-[6vh]", title: "text-md" } });
  router.push("/main");
}

```

```

    }
    else throw new Error ("Непредвиденная ошибка создания объявления!");
  }
  catch (err) {
    console.error(err);
    setError(err instanceof Error ? err.message : "Непредвиденная ошибка со-
здания объявления!");
  };
};

// При выборе типа мяса
const handleSelectChange = (keys: SharedSelection) => {
  if (keys === "all") return;
  const [first] = Array.from(keys);
  setMeatType(first as string);
};

// Вызывается при выборе файлов
const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  const files = e.currentTarget.files;
  if (!files) return;
  setSelectedFiles(prev => [...prev, ...Array.from(files)]);
  // Очищаем input, чтобы повторный выбор тех же файлов сработал
  e.currentTarget.value = "";
};

export const RegistrationPage = () => {

  const router = useRouter();
  const [loading, setLoading] = useState(false);
  const [isVisible, setIsVisible] = useState(false);

  const toggleVisibility = () => setIsVisible(!isVisible);

  /*
  Если ввод с пустого поля, то валидация применяется только при потере фокуса.
  А если изменение уже введенных данных, ошибочных или корректных, то применяется
валидация при изменении,
  так как поле сохраняет статус "в фокусе", пока не будет очищено.
  Проверка совпадения введенных паролей осуществляется только при отправке
формы.
  */

```

```

const {
  formData,
  errors,
  touched,
  setErrors,
  handleChange,
  handleBlur,
  handleSubmit
} = useAuthValidators<RegistrationFormData>({
  schema: registrationSchema,
  defaultValues: {
    email: "",
    password: "",
    confirmPassword: "",
  },
  onSubmit: async (data) => {

    setLoading(true);
    try {
      const response = await fetch(`${process.env.NEXT_PUB-
LIC_SERVER_URL}/api/v1/register`, {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          "Accept": "application/json",
        },
        body: JSON.stringify({
          email: data.email,
          password: data.password,
          confirmPassword: data.confirmPassword,
        }),
      });

      const result = await response.json();

      // Обработка ошибок с сервера
      if (!response.ok) {
        if (result.message === `Email ${formData.email} already in use`) {
          setErrors({ email: "Этот email уже зарегистрирован" });
        } else {

```

```

        setErrors({ general: result.error || "Ошибка регистрации" });
    }
    return;
}
else {
    addToast({
        title: "Профиль зарегистрирован",
        description: "Пожалуйста, выполните вход",
        color: "primary",
        variant: "solid",
        timeout: 3500,
        classNames: {
            base: "mt-[6vh]",
            title: "text-md",
        }
    });
    router.push("/auth/login");
}

}
catch (error) {
    setErrors({ general: "Ошибка входа!" });
}
finally {
    setLoading(false);
}
},
});

```

```

export const Advertisement = () => {

```

```

    const router = useRouter();
    const search = useSearchParams();
    const params = useParams();
    const meatType = search.get("meatType");
    const adId = params.adId as string;
    const [ad, setAd] = useState<AdvertisementUnion>();
    const [error, setError] = useState<string | null>(null);
    const [loading, setLoading] = useState<boolean>(true);
    const [avatarURL, setAvatarURL] = useState<string>();
    const [adPhotoURLs, setAdPhotoURLs] = useState<string[]>([]);

```

```

// Настройки карусели изображений
const settings = {
  dots: true, // Отображаются ли точки навигации?
  infinite: true, // Бесконечная прокрутка?
  speed: 500, // Скорость анимации
  slidesToShow: 1, // Сколько слайдов показывать
  slidesToScroll: 1, // Сколько слайдов прокручивать
  arrows: false, // Отключены стрелки
};

// Получаем информацию об объявлении
useEffect(() => {
  const fetchAdvertisement = async () => {
    try {
      if (!meatType || !adId) throw new Error("Не указаны параметры объяв-
ния!");

      const data = await getAdvertisementById(meatType, adId);
      setAd(data);
    }
    catch (err) {
      console.error(err);
      setError(err instanceof Error ? err.message : "Непредвиденная ошибка
загрузки объявлений!");
    }
  };

  fetchAdvertisement();
}, [meatType, adId]);

// Получаем аватарку пользователя
useEffect(() => {
  const fetchFile = async () => {
    try {
      const filename = ad?.sellerUser?.photo?.path || "Default avatar.jpg";
      const response = await fetch(`/api/download?filename=${filename}`);
      if (!response.ok) {
        setError("Ошибка получения аватарки пользователя!");
      }
      const data = await response.json();
      setAvatarURL(data.data.url);
    }
  };
}, [ad]);

```

```

    }
    catch (error) {
        setError("Ошибка получения аватарки пользователя!");
    }
};
if (ad) fetchFile();
}, [ad]);

// Получаем изображения объявления
useEffect(() => {
    const fetchAdPhoto = async () => {
        try {
            const filename = ad?.files;
            const urls = [];
            if (filename == undefined || filename.length === 0) {
                const defaultPhoto = "default-adv-image.jpg";
                const response = await fetch(`/api/download?filename=${default-
Photo}`);
                if (!response.ok) throw new Error("Ошибка получения фотографии объяв-
ления по умолчанию!");
                const data = await response.json();
                urls.push(data.data.url);
            }
            else {
                for (let i = 0; i < filename.length; i++) {
                    const response = await fetch(`/api/download?filename=${file-
name[i].path}`);
                    if (!response.ok) throw new Error(`Ошибка получения ${i}-той фото-
графии объявления!`);
                    const data = await response.json();
                    urls.push(data.data.url);
                }
            }
            setAdPhotoURLs(urls);
        }
        catch (err) {
            console.error(err);
            setError(err instanceof Error ? err.message : "Непредвиденная ошибка
загрузки объявлений!");
        }
    }

```



```

        finally {
            setLoading(false);
        }
    };
    if (ad) fetchAdPhoto();
}, [ad]);

if (loading) {
    return (
        <><CustomSpinner mt={5}/></>
    );
}

if (error || !ad) {
    return (
        <div className="p-6 flex flex-col items-center justify-center">
            <p className="text-2xl">Не удалось загрузить данные объявления!</p>
            <p className="text-red-500">{error}</p>
            <button onClick={() => router.back()} className="mt-4 underline mx-
auto">Назад</button>
        </div>
    );
}

export const AccountComponent = () => {

    const [userData, setUserData] = useState<UserData | null>(null);
    const [fileData, setFileData] = useState<{ url: string; filename: string } |
null>(null);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState<string | null>(null);
    const fileInputRef = useRef<HTMLInputElement | null>(null);
    const [isLoading, setIsLoading] = useState(false); // для спиннера при изме-
нении аватарки

    // Получение данных пользователя
    useEffect(() => {
        const fetchUserData = async () => {
            try {
                const data = await getUserData();

```

```

        setUserData(data);
    }
    catch (error) {
        setError("Ошибка получения данных пользователя!");
    }
};

fetchUserData();
}, []);

// Получение аватарки пользователя
useEffect(() => {
    const fetchFile = async () => {
        try {
            const filename = userData?.photo?.path || "Default avatar.jpg";
            const response = await fetch(`/api/download?filename=${filename}`);

            if (!response.ok) {
                setError("Ошибка получения аватарки пользователя!");
            }

            const data = await response.json();
            console.log(data);
            setFileData(data.data);
        }
        catch (error) {
            console.log(error);
            setError("хз!");
        }
        finally {
            setLoading(false);
        }
    };

    if (userData) fetchFile();
}, [userData]);

// Обновление данных пользователя
const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {

```

```

event.preventDefault();
setError(null);

try {
  const formData = new FormData(event.currentTarget);

  // Собираем заполненные данные из формы
  const updateFields: UpdateUserDataTypes = {
    name: formData.get("name")?.toString() || undefined,
    surname: formData.get("surname")?.toString() || undefined,
    city: formData.get("city")?.toString() || undefined,
    phoneNumber: formData.get("phoneNumber")?.toString() || undefined
  };

  await updateUserData(updateFields);
  const freshData = await getUserData();
  setUserData(freshData);
}
catch (error) {
  setError("Ошибка обновления данных пользователя!");
};
};

// Обновление аватарки пользователя
const handleUpload = async (e: React.FormEvent) => {
  e.preventDefault();

  if (!fileInputRef.current?.files?.length) return;
  setIsLoading(true);

  const formData = new FormData();
  const files: File[] = Array.from(fileInputRef.current.files);
  files.forEach(file => {formData.append("files", file)});

  try {
    const response = await fetch("/api/upload", {
      method: "POST",
      body: formData,
    });
  }

```

```

    if (!response.ok) {
      setError("Ошибка загрузки новой аватарки в хранилище!");
    }

    const result: UploadResponse = await response.json();

    // Если аватарка успешно загружена в minio, отправим на сервер название
    файла и перезагружаем ее
    if (result.status === "success" && result.data?.[0]) {

      const updateUserAvatar: UpdateUserDataTypes = {
        photo: {
          path: result.data[0].fileName
        }
      };

      await updateUserData(updateUserAvatar);

      const freshData = await getUserData();
      setUserData(freshData);
      console.info("Uploaded files:", result.data);
    }
    else {
      console.error("Upload failed:", result.message);
    }
  }
  catch (error) {
    setError("Ошибка обновления аватарки пользователя!");
  }
  finally {
    setIsLoading(false);
  }
};

if (loading) {
  return (
    <div className="grid grid-cols-12 gap-x-4 mt-8 animate-pulse">
      <div className="col-span-5 col-start-2 h-72 bg-gray-300 rounded-
xl"></div>

```

```

        <div className="col-span-5 col-start-7 h-96 bg-gray-300 rounded-
xl"></div>
    </div>
    );
}

if (error) {
    return (
        <div className="max-w-md mx-auto p-4 text-red-500 text-center">
            {error}
        </div>
    );
}

if (!fileData || !userData) {return <p>Тут ничего нет</p>;}

```