



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)

Кафедра инструментального и прикладного программного обеспечения (ИиППО)

КУРСОВАЯ РАБОТА

по дисциплине: Разработка серверных частей интернет-ресурсов

по профилю: Разработка программных продуктов и проектирование информационных систем

направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: «Разработка клиент-серверного фуллстек-приложения для размещения объявлений с использованием Spring Boot и React»

Студент: Верт Дмитрий Андреевич

Группа: ИКБО-24-20

Работа представлена к защите 11.05.2023 /Верт Д.А./
(подпись и ф.и.о. студента)

Руководитель: Куликов Александр Анатольевич, к.т.н., доцент

Работа допущена к защите 08.06.23 /Куликов А.А./
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: отлично
08.06.23 /ст.прек. Гачков А.В. /
08.06.23 /к.т.н. Куликов А.А. /

(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших
защиту)

М. РТУ МИРЭА. 2023



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА - Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине: Разработка клиент-серверных приложений
по профилю: Разработка программных продуктов и проектирование информационных систем
направления профессиональной подготовки: Программная инженерия (09.03.04)
Студент: Верт Дмитрий Андреевич
Группа: ИКБО-24-20
Срок представления к защите: 11.05.2023
Руководитель: к.т.н., доцент Куликов Александр Анатольевич

Тема: «Клиент-серверное фуллстек-приложение для размещения объявлений»

Исходные данные: PostgreSQL, Spring Boot, React, Node.js, Git, GitHub, Railway, ГОСТ 7.32-2017. Отчет о научно-исследовательской работе. Структура и правила оформления Нормативный документ: инструкция по организации и проведению курсового проектирования СМКО МИРЭА 7.5.1/04.И.05-18

Перечень вопросов, подлежащих разработке, и обязательного графического материала: 1. Провести анализ предметной области для выбранной темы с обоснованием выбора клиент-серверной архитектуры для разрабатываемого приложения; 2. Выбрать, ориентируясь на мировой опыт и стандарты в данной области, а также на выбранную тему, программный стек для реализации фуллстек разработки; 3. Дать описание архитектуры разрабатываемого клиент-серверного приложения, с помощью UML нотаций и с обоснованием выбора вида клиента(браузер, мобильное приложение, кроссплатформенный клиент и т.д.); 4. Провести реализацию фронтенд и бэкенд части клиент-серверного приложения, обеспечив версиюный контроль процесса разработки с помощью Git. В разработанном приложении должна обеспечиваться авторизация и аутентификация пользователя; 5. Разместить проект клиент-серверного приложения в репозитории GitHub с приложением в тексте отчёта ссылки на данный репозиторий; 6. Интегрировать проект на GitHub с Railway, с целью развёртывания разработанного клиент-серверного приложения в облаке. Ссылку на приложение в облаке Render привести в тексте пояснительной записки. 7. Провести проверку функционирования минимально жизнеспособного продукта с использованием сгенерированных тестовых данных. 8. Разработать презентацию с графическими материалами.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Зав. кафедрой ИиППО: Болбаков Р. Г. / Болбаков Р. Г. /, « 20 » 02 2023 г.

Задание на КР выдал: Куликов А.А. / Куликов А.А. /, « 20 » 02 2023 г.

Задание на КР получил: Верт Д.А. / Верт Д.А. /, « 20 » 02 2023 г.

АННОТАЦИЯ

Отчёт содержит 36 страниц, 22 иллюстрации, 8 листингов (большая часть из которых расположены в приложении А в конце отчёта), 15 литературных источников.

Целью данной курсовой работы являлось создание фуллстек-приложения по размещению объявлений. Данный программный продукт был разработан с использованием технологий Java 17, Spring Boot, React, PostgreSQL и Docker Compose. В приложении предусмотрена система авторизации пользователей, сводка созданных объявлений и зарегистрированных пользователей, список объявлений с адресом, картинкой и описанием, форма создания новых объявлений, а также панель администратора с возможностью блокировки/активации пользователей и удаления объявлений.

В качестве стека технологий для реализации серверной части использовался Spring Boot, включающий Spring Data JPA для управления объектно-реляционной моделью и взаимодействия с базой данных PostgreSQL, а также Spring Security для конфигурирования политик безопасности приложения. Веб-интерфейс был реализован с использованием ReactJS. Приложение было развернуто в Docker-контейнерах с помощью Docker Compose на выделенном персональном сервере и доступно для использования всеми желающими по соответствующему URL.

Исходный код приложения доступен для просмотра и скачивания по соответствующей ссылке в сети Интернет.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ.....	5
ВВЕДЕНИЕ	7
1 ОБЩИЕ СВЕДЕНИЯ	8
1.1 Обоснование выбора средств ведения разработки	8
2 ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ	9
3 ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ	10
3.1 Анализ предметной области	10
3.2 Проектирование архитектуры приложения	13
3.3 Проектирование базы данных	14
3.4 Разработка серверной части веб-приложения.....	15
3.5 Разработка клиентской части веб-приложения	21
3.6 Развертывание приложения с помощью Docker Compose	23
3.7 Обзор разработанного программного продукта	25
3.8 Подготовка к защите курсовой работы	28
ЗАКЛЮЧЕНИЕ	30
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ	31
ПРИЛОЖЕНИЕ А	33

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

БД	—	База данных
СУБД	—	Система управления базами данных
API	—	Application Programming Interface (программный интерфейс)
CRUD		Create, Read, Update, Delete (перечень базовых операций – создания, чтения, обновления, удаления)
CSS	—	Cascading Style Sheets (формальный язык описания внешнего вида документа, написанного с использованием языка разметки)
DDL	—	Data Definition Language (группа операторов определения данных в SQL)
DML	—	Data Manipulation Language (группа операторов управления данными в SQL)
DTO	—	Data Transfer Object (объект, представляющий программную сущность и хранящий соответствующие данные из полей БД)
HTML	—	HyperText Markup Language (стандартизированный язык разметки документов для просмотра веб-страниц в браузере)
IDE	—	Integrated Development Environment (интегрированная среда разработки — комплекс программных средств, используемый программистами для разработки программного обеспечения)
JDK	—	Java Development Kit (бесплатно распространяемый компанией Oracle Corporation комплект разработчика приложений на языке Java)
JPA	—	Java Persistence API (спецификация API Java EE, предоставляет возможность сохранять в удобном виде Java-объекты в базе данных)
JSON	—	JavaScript Object Notation (текстовый формат обмена данными, основанный на JavaScript)

MVP	—	Minimum Viable Product (минимально жизнеспособный продукт — продукт, обладающий минимальными, но достаточными для удовлетворения первых потребителей функциями)
REST	—	Representational State Transfer (архитектурный стиль взаимодействия компонентов распределённого приложения в сети)
SSH	—	Secure Shell (сетевой протокол прикладного уровня, позволяющий производить удалённое управление операционной системой)
UI	—	User Interface (пользовательский интерфейс)
URL	—	Uniform Resource Locator (унифицированный указатель ресурса)

ВВЕДЕНИЕ

На данный момент информационные технологии проникли во все сферы государства, общества и бизнеса. Трудно представить организацию любого процесса без задействования современных технологий, поскольку они значительно его упрощают, систематизируют и оптимизируют. В частности, каждое уважающее себя учебное учреждение имеет свой интернет-портал, а любой сколько-нибудь значительный бизнес обязан иметь собственный сайт или приложение для ведения предпринимательской деятельности через Интернет. Более того, некоторые бизнес-услуги могли и вовсе быть немыслимы до широкого распространения веб-приложений.

Целью данной курсовой работы является разработка клиент-серверного фуллстек-приложения для размещения объявлений с использованием Spring Boot и React [1].

Для систематизации работы и упрощения восприятия отчета процесс разработки был поделен на семь основных частей:

1. Обоснование выбора средств ведения разработки.
2. Анализ предметной области.
3. Проектирование базы данных.
4. Разработка серверной части веб-приложения.
5. Разработка клиентской части веб-приложения.
6. Развертывание приложения на сервере.
7. Обзор разработанного программного продукта.

Приложение, созданное в результате выполнения курсовой работы должно представлять MVP (Minimum Viable Product) сервиса объявлений и иметь базово необходимый функционал для размещения и просмотра объявлений, иметь графический веб-интерфейс пользователя, производить аутентификацию и авторизацию пользователей, быть спроектировано в клиент-серверной архитектуре и развернуто на сервере, в готовом к работе состоянии для всех пользователей сети Интернет.

1 ОБЩИЕ СВЕДЕНИЯ

1.1 Обоснование выбора средств ведения разработки

Первым этапом любой разработки является выбор и установка соответствующего программного обеспечения. Так в качестве среды разработки была выбрана IntelliJ IDEA Ultimate. Преимуществами IDEA является статус де факто стандарта при выборе IDE (Integrated Development Environment) в разработке на Java.

Версия JDK (Java Development Kit) выбрана 17, так как она самая современная из долго поддерживаемых, включает в себя все самые важные и необходимые функции языка Java [2]. А где Java, там почти всегда и Spring [3] – стандарт при выборе фреймворка для разработки enterprise-приложений на Java. Для организации выполнения запросов к базе данных из среды Java был применён фреймворк Spring Data – интегрированный в экосистему Spring компонент. Развертываться приложение будет на Spring Boot, поскольку он обеспечивает простейшую «из коробки» интеграцию с остальным Java-приложением на Spring. В роли СУБД была выбрана PostgreSQL [4], так как она бесплатна, широко распространена, имеет продвинутый функционал, стабильна и масштабируема, в отличие от NoSQL решений. В качестве системы сборки используется Maven [5], как наиболее популярный и гибко кастомизируемый инструмент.

При разработке UI (User Interface) был использован React [6], который позволяет разрабатывать сложные интерфейсы и обеспечивает удобную и быструю разработку фронтенда.

Приложение развертывается в Docker контейнерах [7] на выделенном персональном сервере. Для управления контейнерами использовался Docker Compose.

2 ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

Разработанное приложение предназначено для создания и управления платформой онлайн объявлений. Оно предназначено для использования как полноценный продукт для тех, кто хочет размещать объявления в сети, а также для тех, кто хочет создавать собственные проекты, связанные с онлайн объявлениями.

Приложение предоставляет функции для создания, редактирования, просмотра и удаления объявлений, а также для поиска объявлений по различным критериям. Целевой аудиторией данного программного продукта могут быть как частные лица, которые ищут товары и услуги, так и компании, которые хотят разместить свои объявления в интернете для привлечения новых клиентов. Также, это может быть интересно для начинающих разработчиков, которые хотят понимать, как работает полноценный веб-сервис и как можно разрабатывать приложения, используя технологии, применяемые в данном проекте.

3 ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

3.1 Анализ предметной области

Сбор информации был начат с изучения уже существующих на рынке аналогов. Для этого в качестве лидера в нише был выбран и изучен популярный сайт «Avito», который предоставляет широкий перечень услуг, в том числе размещение объявлений. На рисунке 1 представлена главная страница сервиса, на которой можно наблюдать разбиение объявлений по различным категориям и общий поиск. Так же обратим внимание на яркую кнопку «Разместить объявление» в правом верхнем углу, а так же переход на страницу входа и регистрации рядом.

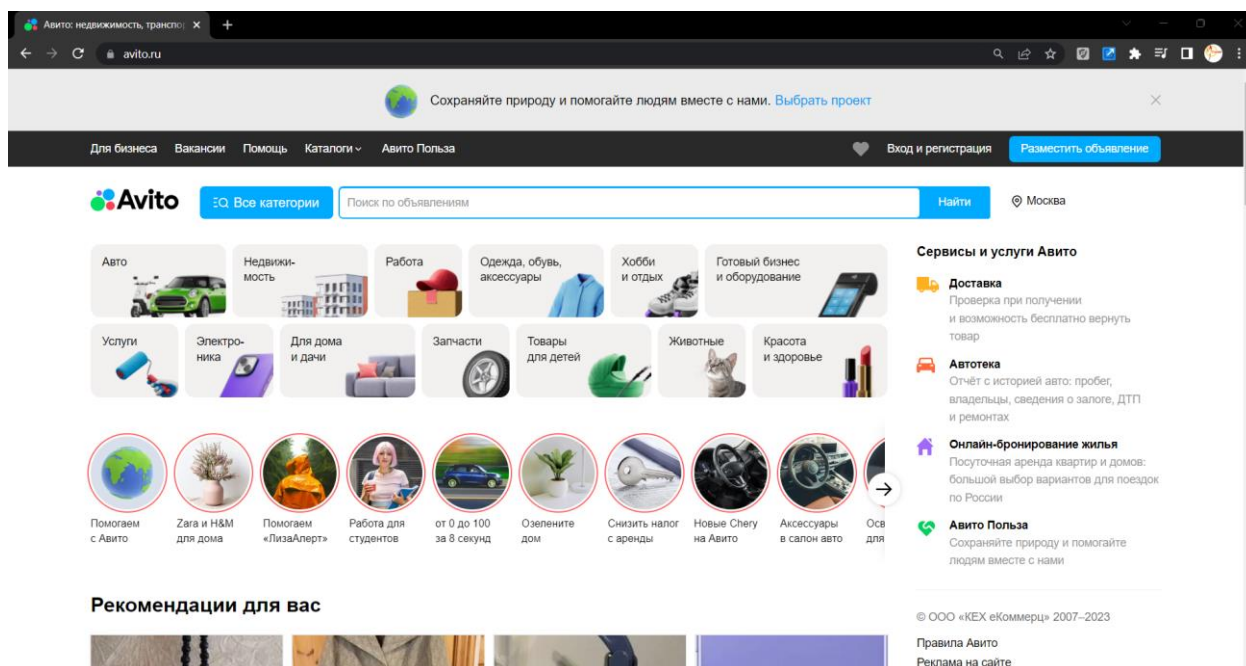


Рисунок 1 – Главная страница сервиса «Avito»

Спустившись ниже, можно наблюдать сами объявления, рекомендованные системой (рисунок 2). Объявление имеет главную фотографию, заголовок, цену, и дату публикации, а так же адрес. На одной странице собраны несколько объявлений, так, чтобы каждое не занимало много места и предоставляя только главную информацию.

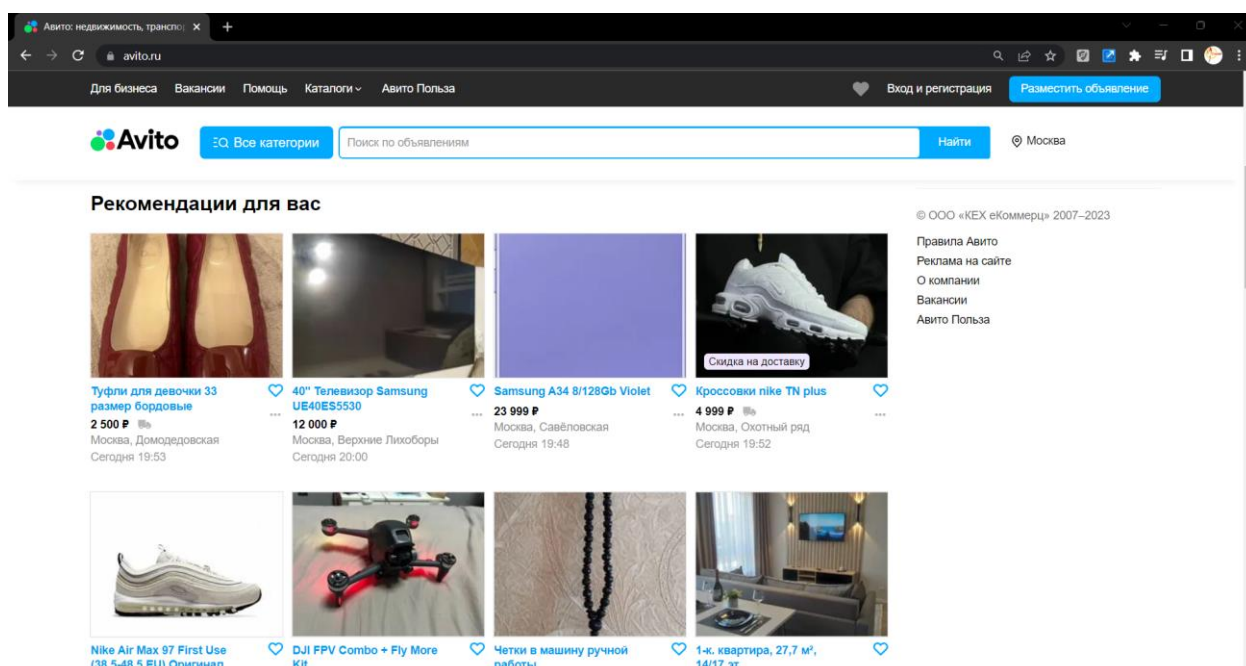


Рисунок 2 – Вид объявлений на «Авито»

Вторым сервисом для анализа выступит «Юла» (рисунок 3). Здесь, так же как и у «Авито», на главной странице располагается поиск по объявлениям, выбор категорий, авторизация, и яркая зеленая кнопка «Разместить объявление».

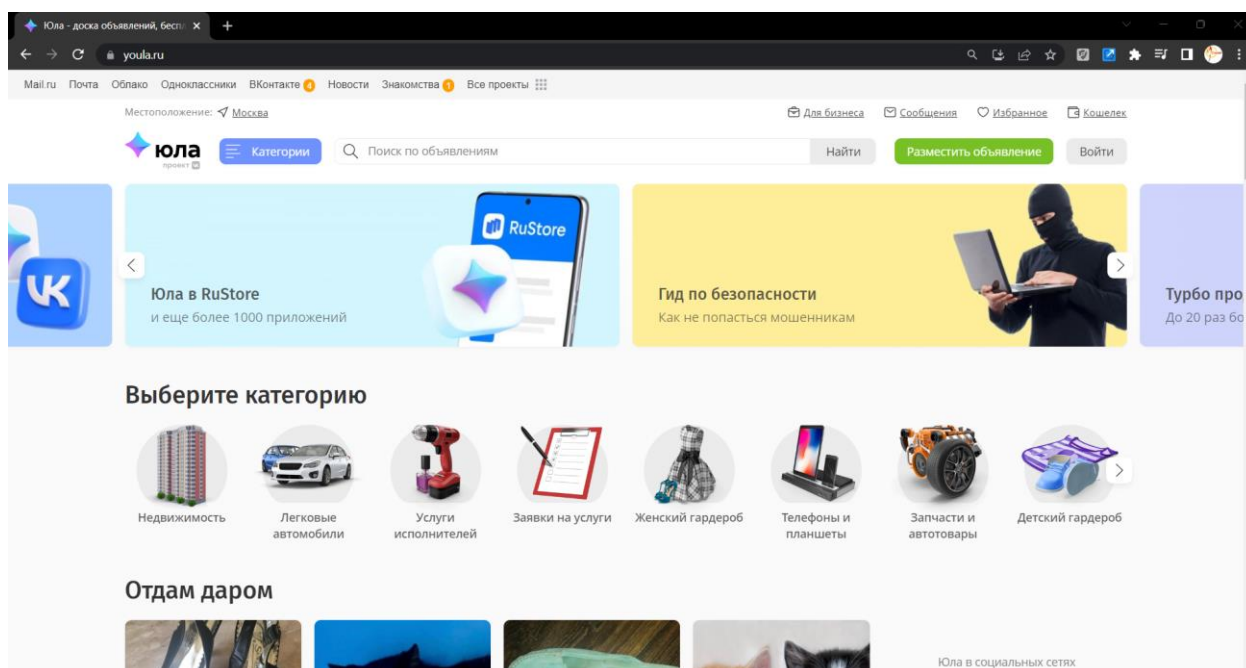


Рисунок 3 – Главная страница сервиса «Юла»

Сами объявления на «Юле» выглядят несколько иначе (рисунок 4) – на них представлено меньше информации, лишь цена, заголовок и фото. Компонировка объявлений же идентична.

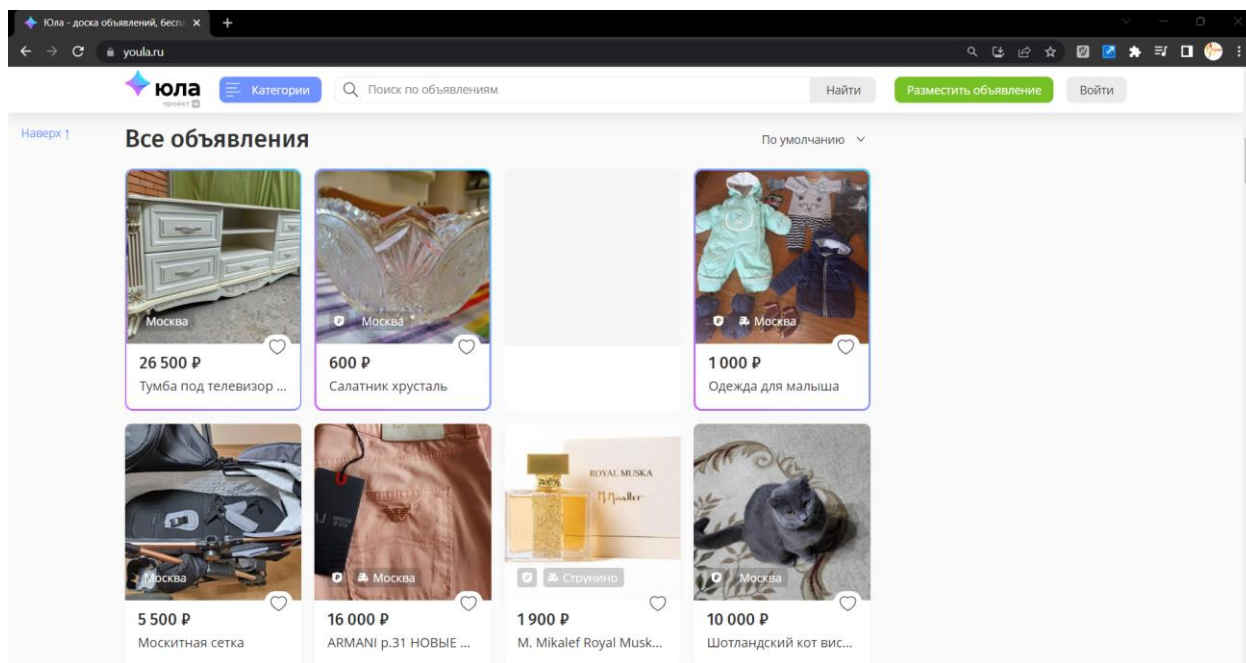


Рисунок 4 – Вид объявлений на «Юле»

Проанализировав возможности данных сервисов можно сделать вывод, что для конкурентноспособности разрабатываемого приложения MVP должен обладать перечисленными выше качествами, а именно:

1. Регистрация, авторизация.
2. Просмотр списка объявлений, у каждого из которых есть фотография, и дополнительная информация, включая заголовок, адрес, цену, дату и описание.
3. Возможность размещения собственного объявления.
4. Поиск по объявлениям.
5. Панель администратора для удобства возможной модерации сайта.

3.2 Проектирование архитектуры приложения

Данное приложение разрабатывается в клиент-серверной архитектуре, где клиентскую часть представляет фронтенд, написанный на JavaScript библиотеке React, а серверную часть - бэкенд, написанный на Java с использованием фреймворка Spring. Фронтенд взаимодействует с сервером посредством RESTful API, предоставляемого бэкендом [8].

В рамках данной архитектуры, все запросы клиента обрабатываются на сервере, где они принимаются соответствующими методами контроллеров. Контроллеры, в свою очередь, вызывают необходимые сервисы, которые взаимодействуют с базой данных и возвращают результаты запросов в виде JSON-объектов. Таким образом, в данном приложении фронтенд и бэкенд представляют отдельные компоненты, которые взаимодействуют друг с другом посредством RESTful API по сети. Схематичная диаграмма спроектированной архитектуры сервиса представлена на рисунке 5.

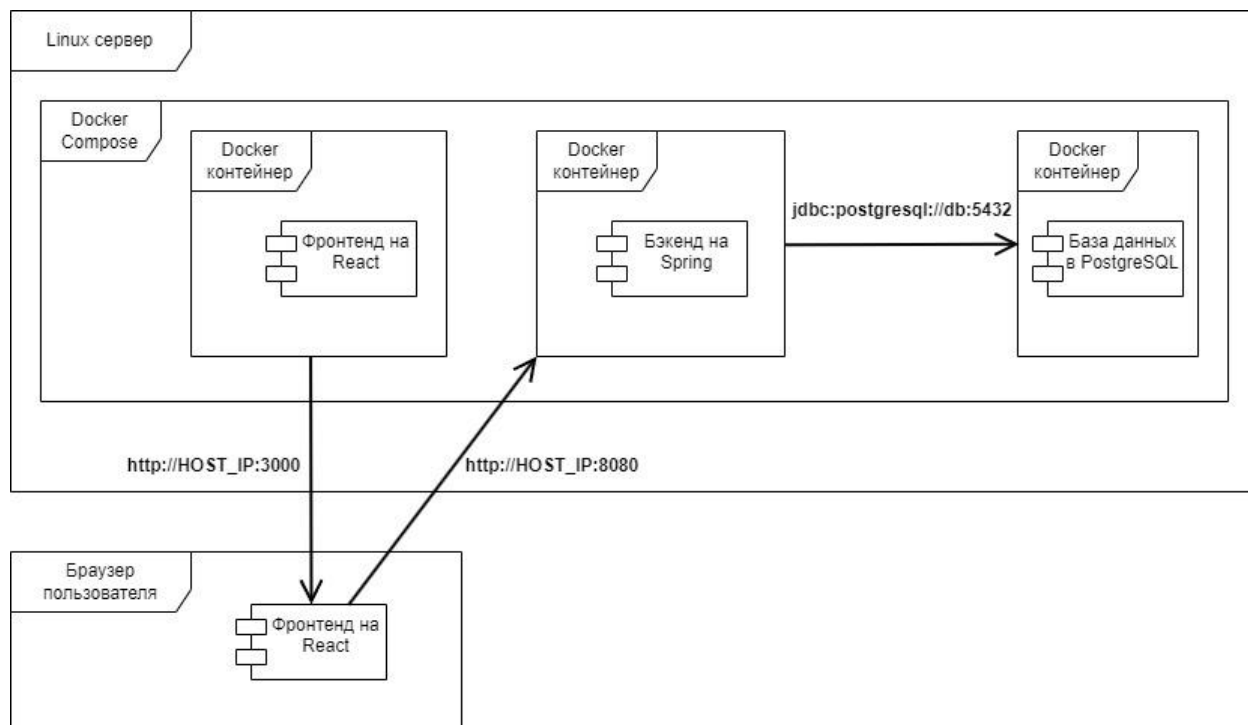


Рисунок 5 – Архитектура сервиса размещения объявлений

База данных так же запущена независимо от бэкенда, что обеспечивает общую низкую связанность этих трех компонентов и позволяет в будущем разнести их на три физически отдельных сервера, что гарантирует хороший потенциал для масштабирования такой распределенной системы. Однако, на данный момент все три компонента будут развернуты на одном хосте.

3.3 Проектирование базы данных

После проведения анализа предметной области и выделения функциональных требований к проекту, следующим этапом необходимо провести описание главных сущностей в будущей системе.

Так, на рисунке 6 представлена разработанная схема базы данных, включающая в себя такие сущности как: пользователь, объявление, адрес и изображение, а так же связи между ними.

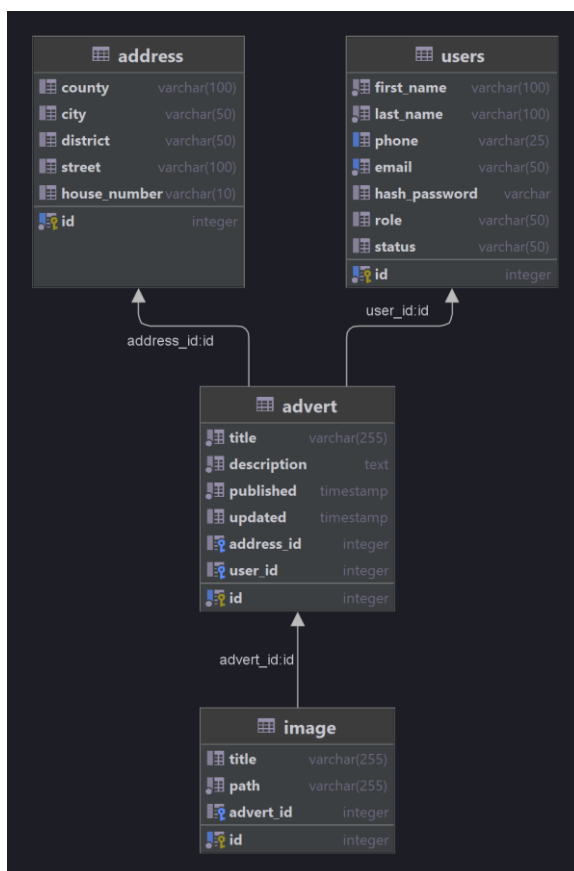


Рисунок 6 – Схема спроектированной базы данных

База данных была создана в PostgreSQL с помощью инициализирующего скрипта, запускаемого Spring при старте приложения и подтверждения соединения с базой данных. Данный скрипт создает структуру таблиц с помощью DDL-команд [9]. Сама база данных запущена внутри Docker контейнера для удобства последующего развертывания внутри единого контейнеризированного окружения разрабатываемого приложения [10]. На листинге 1 можно ознакомиться с содержимым этого скрипта.

3.4 Разработка серверной части веб-приложения

3.4.1 Разработка слоя доменных сущностей

На данном этапе для сущностей в SQL-таблицах были разработаны отображение в виде Java объектов, а так же и DTO (Data Transfer Object), которые будут передаваться на клиент (рисунок 7). Spring Data JPA в свою очередь берёт на себя управление запросами к БД из Java-окружения и обработку ответов от PostgreSQL.

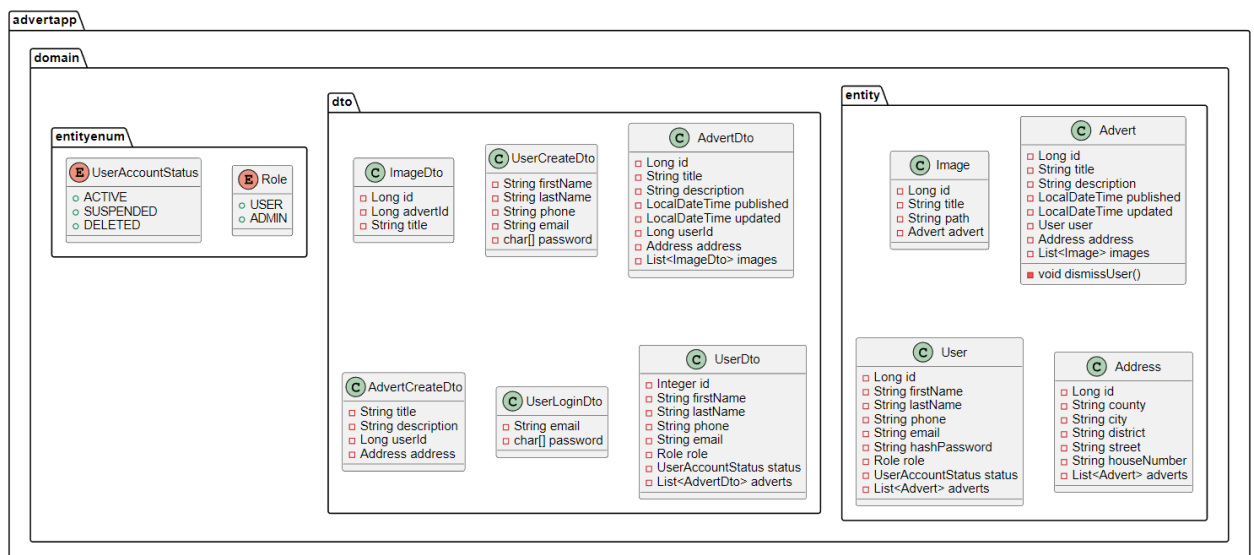


Рисунок 7 – Диаграмма классов слоя домена

Все операции над сущностями на стороне приложения происходят с помощью посредника – интерфейса `CrudRepository`. От него наследуются репозитории для работы с сущностями текущего приложения (рисунок 8), затем каждому из таких интерфейсов сопоставляется имплементация, которая является бином в контексте Spring, и которая сама выполняет отправку необходимых SQL-запросов. Разработчик со своей стороны лишь описывает контракт тех или иных запросов.

Репозитории в данном приложении за неимением сложной логики используются для выполнения базовых CRUD операций, поэтому предопределенная Spring имплементация полностью удовлетворяет требованиям проекта.

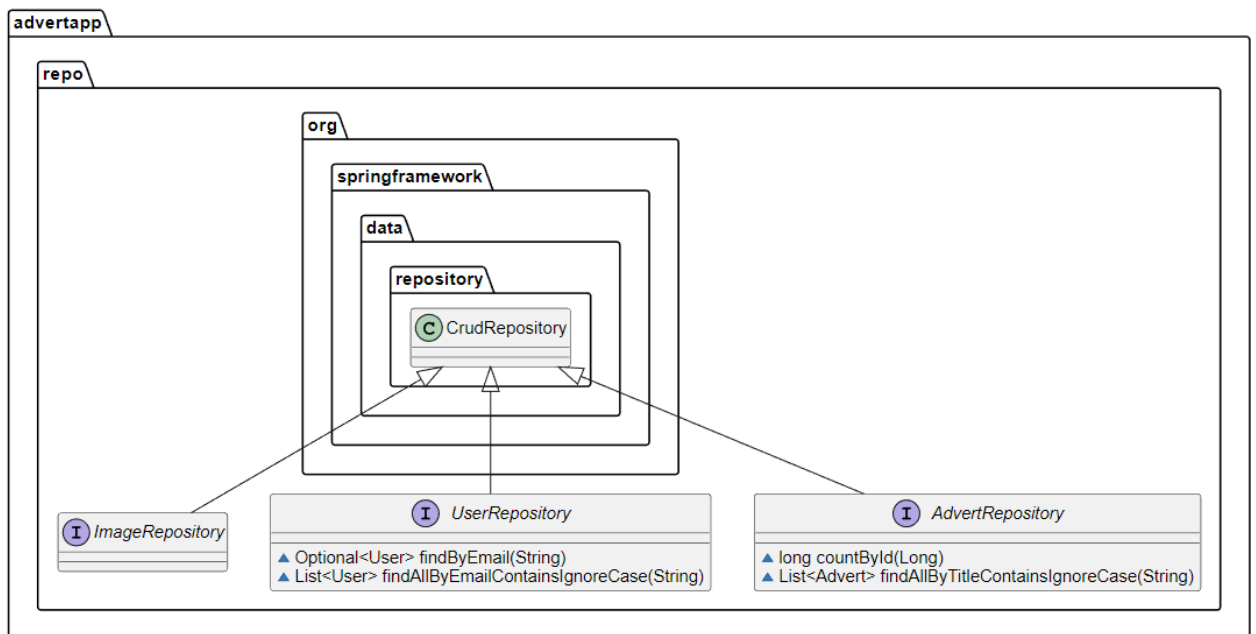


Рисунок 8 – Диаграмма классов репозиториев

3.4.2 Разработка слоя сервисов

Центральным местом всего приложения, реализующим основную бизнес-логику, является слой сервисов. Здесь самым сложным функционалом выступило сохранение изображений на локальной машине. Для этого, как можно наблюдать на листинге 2, в конфигурационном файле задается локальный путь директории для хранения изображений, если такового нет –

то он создается, и уже в нем для каждого объявления создается индивидуальная папка для фотографий, которые относятся конкретно к данному объявлению. Полный путь до изображения при этом сохраняется в поле объекта класса Image и записывается в базу данных.

Аналогично на листинге 3 представлен процесс считывания изображения из файловой системы для отправки на клиент по запросу. Во время всех этих манипуляций изображение проходит нетривиальный процесс сериализации и десериализации, которые заранее реализованы средствами используемых для этих целей библиотек.

С остальными сервисами приложения, а так же их методами можно ознакомиться на диаграмме классов, представленной на рисунке 9.

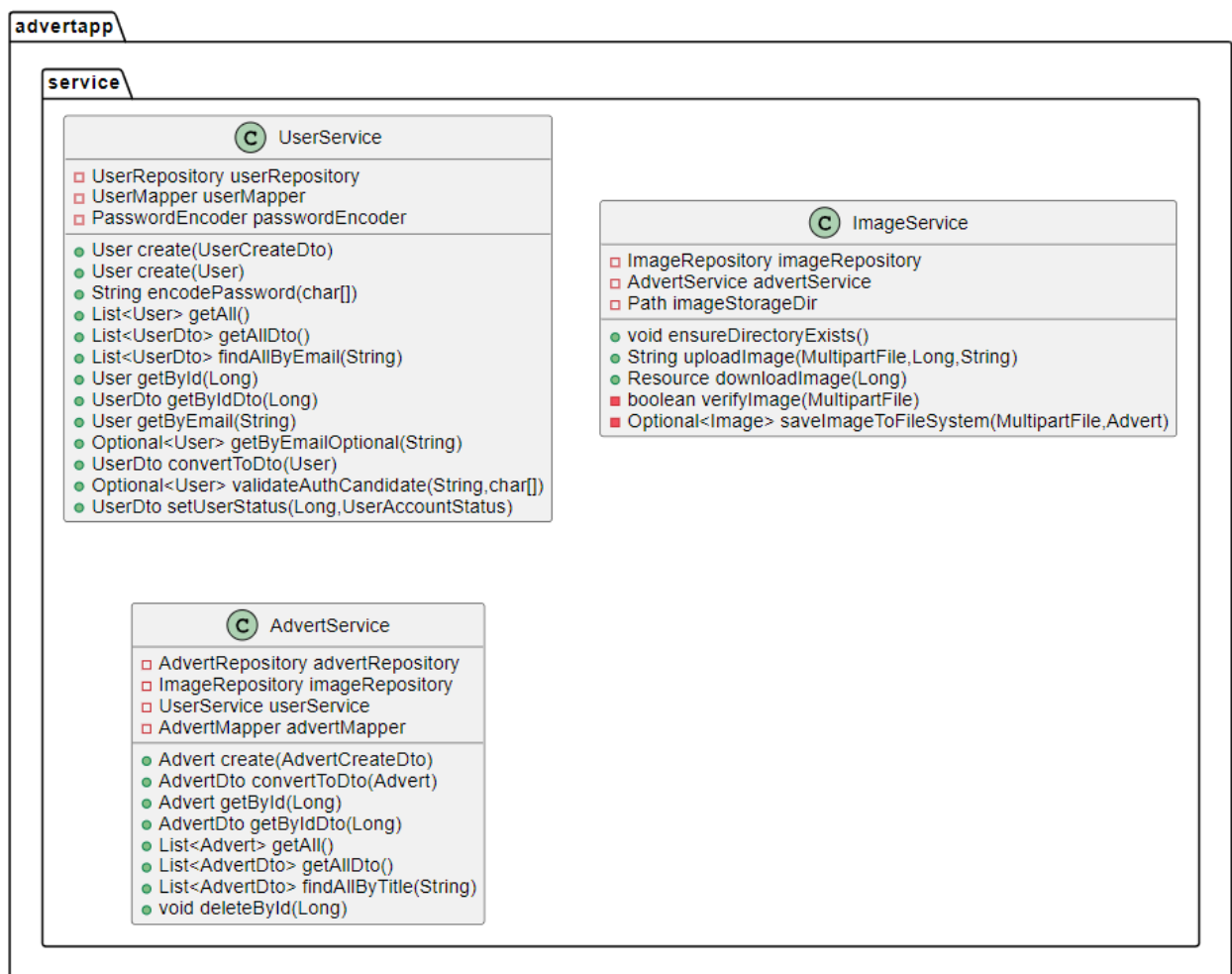


Рисунок 9 – Диаграмма классов слоя сервисов

В соответствии с одним из требований, стоящих перед данной работой, на слое сервисов также реализован компонент, выполняющий инициализацию базы данных тестовыми данными. Всего при старте приложения, в случае если БД пуста, в систему добавляются два пользователя с ролями USER и ADMIN, а так же два полноценных объявления. С отрывком программы, отвечающим за инициализацию базы данных можно ознакомиться на листинге 4.

Листинг 4 – Инициализация базы тестовыми данными

```
@Override
public void run(String... args) {
    if (userService.getByEmailOptional(ADMIN_EMAIL).isPresent() &&
        userService.getByEmailOptional(USER_EMAIL).isPresent()) {
        return;
    }
    getUsers().forEach(userService::create);
    log.info("Database init: users created");
    getAdverts().forEach(advertService::create);
    log.info("Database init: adverts created");
    loadImages();
    log.info("Database init: images created");
}
```

Поскольку были затронуты роли USER и ADMIN, то так же стоит упомянуть о системе авторизации пользователей в системе. Для этих целей был применён компонент Spring Security. Механизм авторизации реализован самый элементарный – Basic Auth [11], при котором на сервер приходят логин и пароль, и в случае их корректности, возвращается статус 200 OK. В дальнейшем, клиент должен сохранить параметры входа в кэше, и добавлять их в заголовок каждого запроса [12].

Когда сервер получает любой такой запрос, то он проходит через фильтр безопасности, в которой происходит авторизация пользователя по переданным в заголовке запроса реквизитам. Когда они считываются – то происходит аутентификация пользователя, а затем сопоставление его роли с ограничениями ресурса, к которому он запрашивает доступ. На листинге 5 представлен отрывок из класса конфигурации безопасности приложения.

3.4.3 Разработка слоя контроллеров

Контроллеры – компоненты системы, помеченные аннотацией *@RestController* и выполняющие связывающую функцию между эндпоинтами API и методами сервисов. В данном приложении представлены пять контроллеров (рисунок 10).

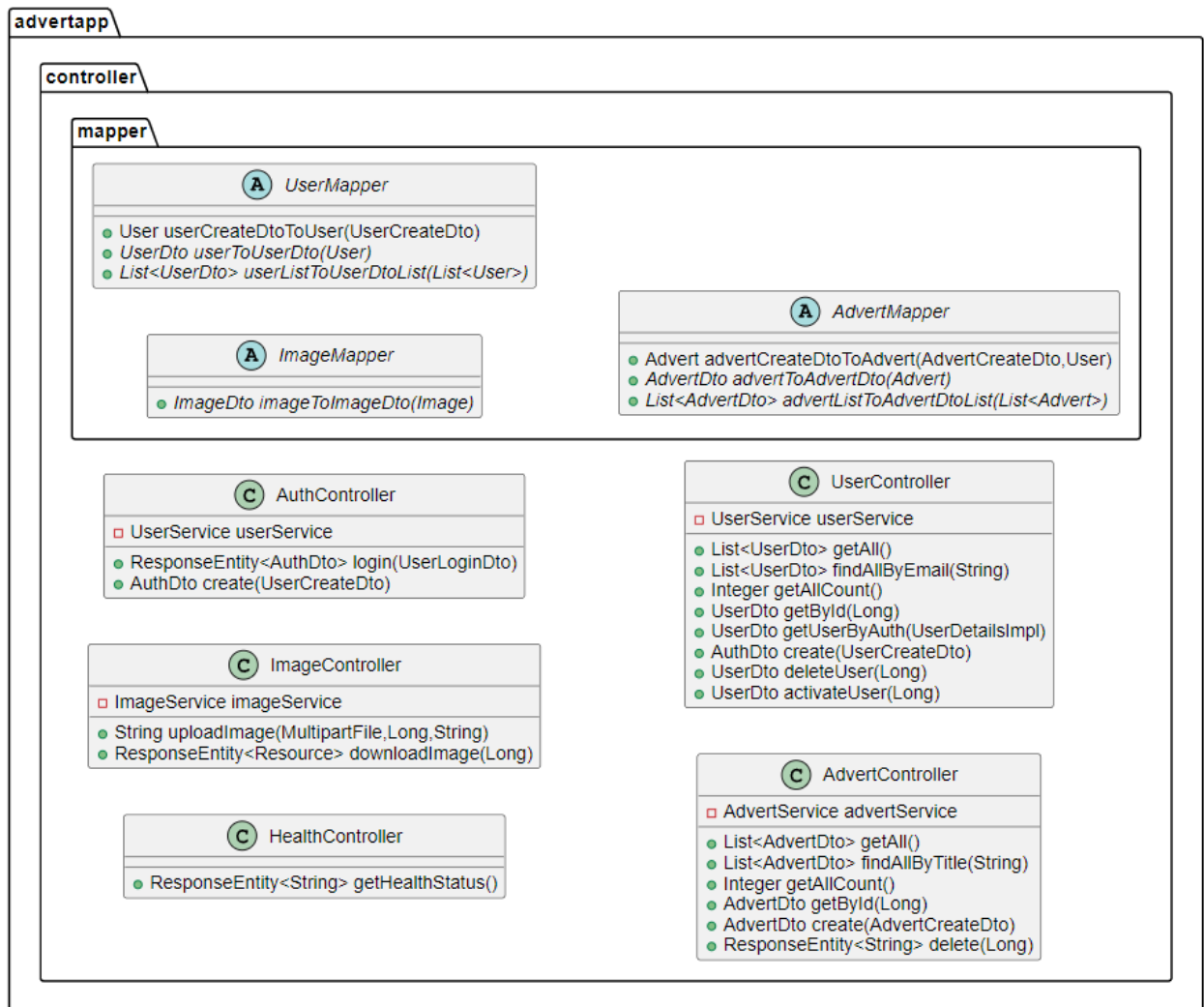


Рисунок 10 – Диаграмма классов слоя контроллеров

Поскольку серверная часть представляет RESTful API и не отдает HTML-страницы непосредственно (что в свою очередь и позволяет построить многоуровневую клиент-серверную систему и уменьшить зацепление между компонентами), то зона ответственности бэкенда заканчивается на

предоставлении ответов на запросы в формате данных JSON, которые получает клиент [13].

Ответ в JSON формате формируется на основе DTO, которые с помощью соответствующих библиотек проходят процесс конвертации из/в Java-объектов (для этого в данном приложении и используются мапперы на диаграмме выше). Со списком разработанных DTO можно было ознакомиться ранее в пункте, посвященном доменным сущностям.

Все эндпоинты проектировались в соответствие с архитектурным стилем REST. Список всех разработанных эндпоинтов приложения представлен на рисунке 11.

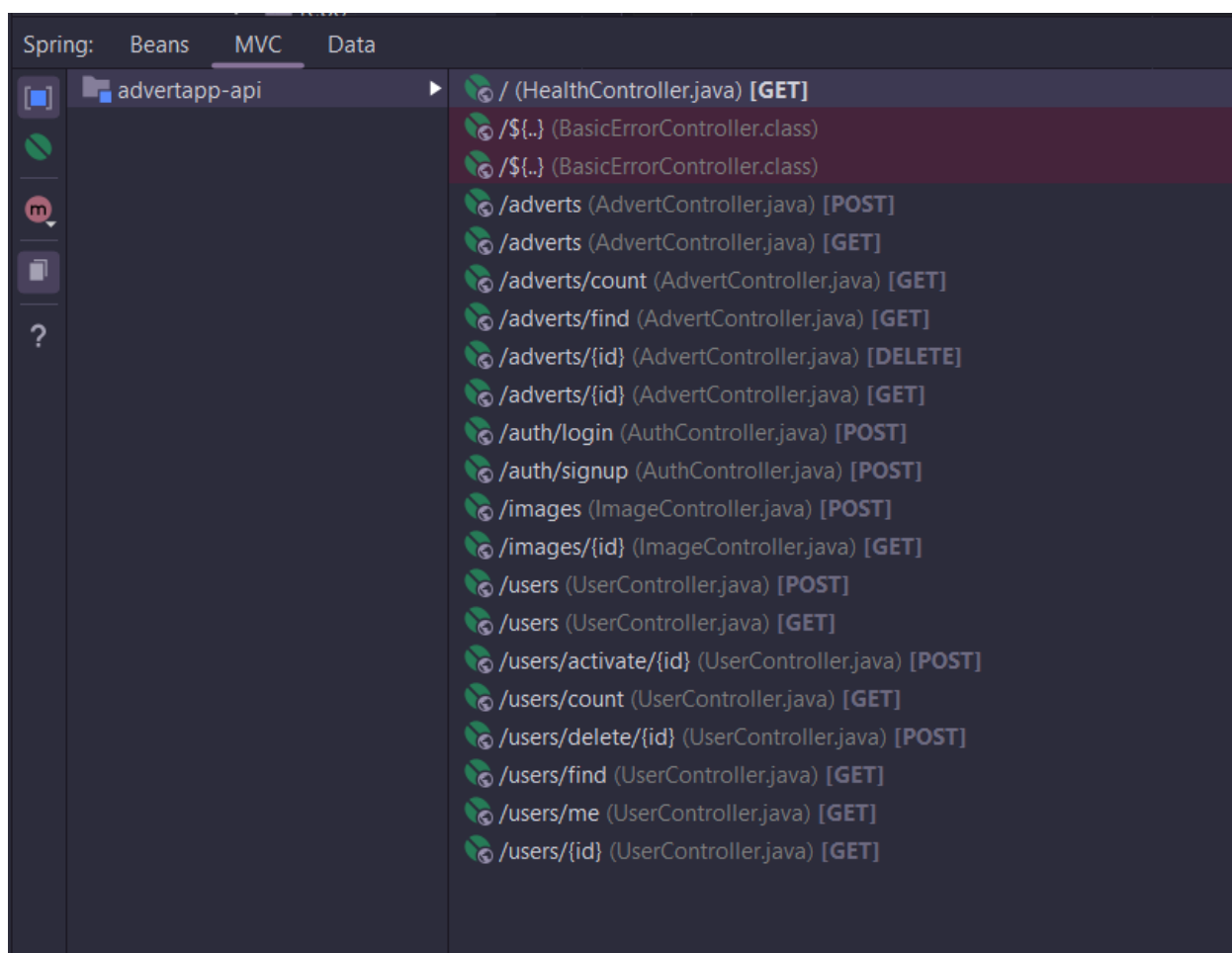


Рисунок 11 – Все эндпоинты серверной части приложения

Для удобства последующего развертывания бэкенд на Spring был контейнеризирован с помощью Docker, для этого в верхнеуровневую директорию добавлен Dockerfile с содержимым, которое представлено на листинге 6. В нем происходит копирование исходного кода внутрь файловой системы контейнера, затем внутри контейнера происходит сборка приложения в jar-архив, который впоследствии и запускается внутри контейнером с предоставлением внешнего порта 8080 из контейнера.

Листинг 6 – Контейнеризация серверной части приложения

```
FROM maven:3.8.3-openjdk-17 AS build
COPY src /home/advertapp-api/src
COPY pom.xml /home/advertapp-api
WORKDIR /home/advertapp-api
RUN mvn clean install -DskipTests=true

FROM openjdk:17-alpine
RUN apk --no-cache add curl
EXPOSE 8080
COPY --from=build /home/advertapp-api/target/advertapp-api-1.0.0.jar
/usr/local/lib/advertapp-api.jar
ENTRYPOINT ["java", "-jar", "/usr/local/lib/advertapp-api.jar"]
```

3.5 Разработка клиентской части веб-приложения

Фронтенд приложения был реализован на библиотеке React для JavaScript, и включает в себя около 15-ти компонентов, необходимых для отображения главной страницы, страниц регистрации и авторизации, просмотра объявлений, создания объявлений, а так же администрирования пользователей и объявлений на UI.

Остановим внимание на компоненте AdvertApi.js, который выполняет отправку и принятие запросов от бэкенда. Для этого была использована библиотека axios [14]. Создается экземпляр класса axios, в который передается URL сервера, к которому будут посылаться запросы. Далее вызывается соответствующий метод, в зависимости от типа запроса, в него передается полезная нагрузка и заголовки. Фрагмент компонента AdvertApi.js, а так же файловая структура клиентской части проекта представлены на рисунке 12.

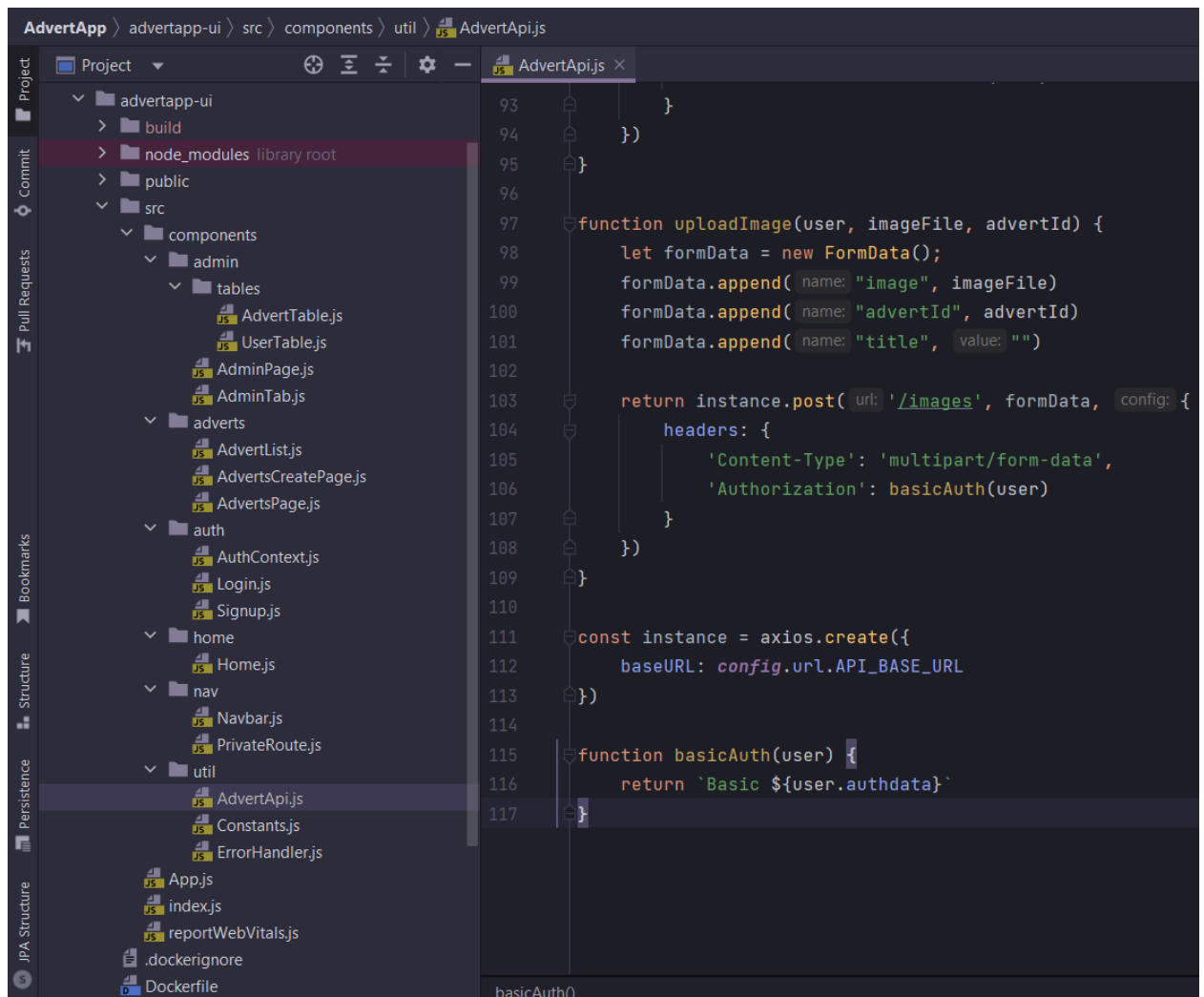


Рисунок 12 – Файловая структура проекта и фрагмент AdvertApi.js

Для дальнейшего развертывания фронтенд на React так же был контейнеризирован с помощью Docker. Содержимое Dockerfile представлено на листинге 7.

Листинг 7 – Контейнеризация клиентской части приложения

```

FROM node:18.14.0

WORKDIR /home/advertpapp-ui
COPY . .
RUN npm ci
RUN npm install -g serve

ARG BACKEND_BASE_URI
ENV REACT_APP_BACKEND_BASE_URI $BACKEND_BASE_URI

RUN npm run build

EXPOSE 3000
CMD [ "npx", "serve", "build" ]

```

3.6 Развертывание приложения с помощью Docker Compose

Для организации взаимодействия между контейнерами был использован плагин для Docker – Docker Compose. В соответствующем `docker-compose.yml` файле (с его содержимым можно ознакомиться на листинге 8) описываются все стартовые параметры такой системы, пробрасываются порты в/из контейнеров, указываются зависимости одних контейнеров от старта других и передаются другие различные настройки для обращения контейнеров друг к другу.

Для развертывания разработанного приложения на промышленном стенде был арендован выделенный сервер на одном из публичных хостингов (рисунок 13).

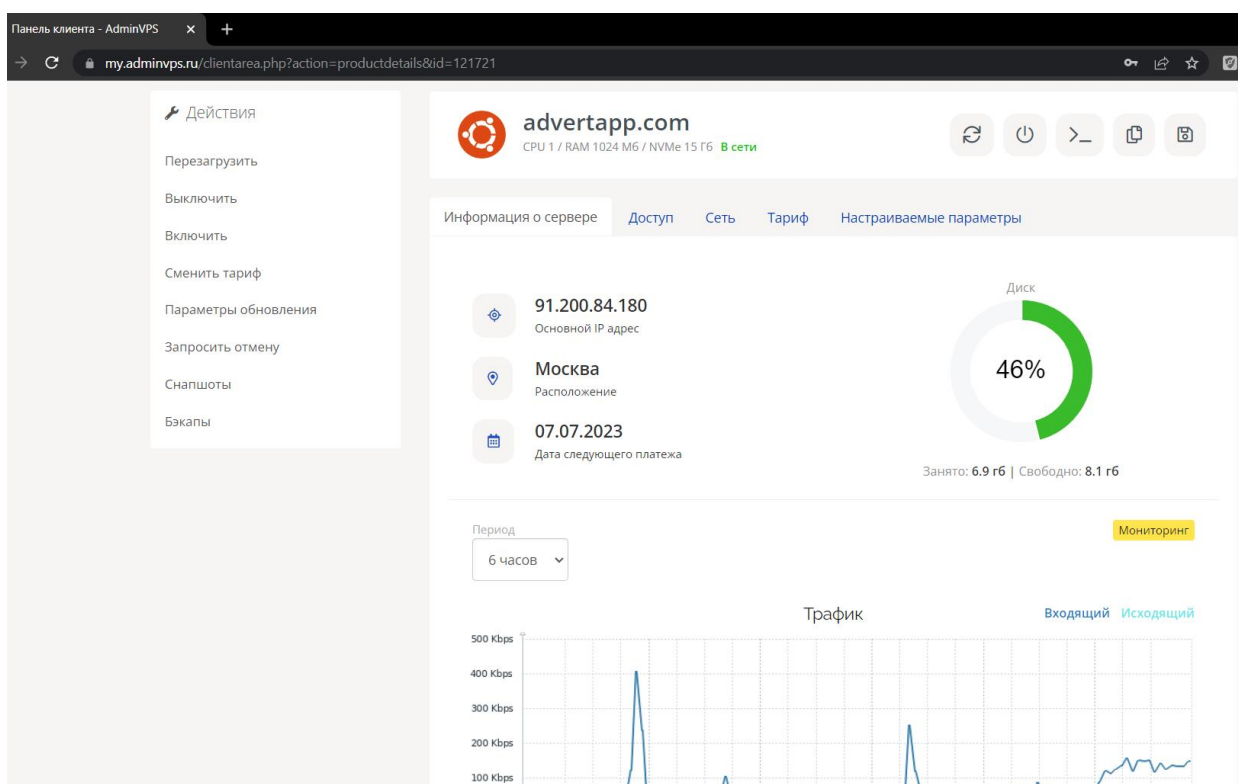


Рисунок 13 – Панель управления персональным сервером

После оплаты сервера на почту пришли логин и пароль, по которым можно выполнить удаленное подключение к серверу по SSH (рисунок 14).

```
root@advertapp: ~
#####
Welcome!

This server is captured by FASTPANEL control panel.

Operation System:      Ubuntu 20.04.6 LTS
=====
IPv4:
91.200.84.180
172.17.0.1
172.18.0.1

=====
By default configuration files can be found in the following directories:

NGINX:      /etc/nginx/fastpanel2-available
APACHE2:    /etc/apache2/fastpanel2-available

Please do not edit configuration files manually.
You may do that in your control panel.

=====

10:53:13 up 13:22,  1 user,  load average: 0.00, 0.00, 0.00
#####
root@advertapp:~#
```

Рисунок 14 – Подключение к удаленному серверу по SSH

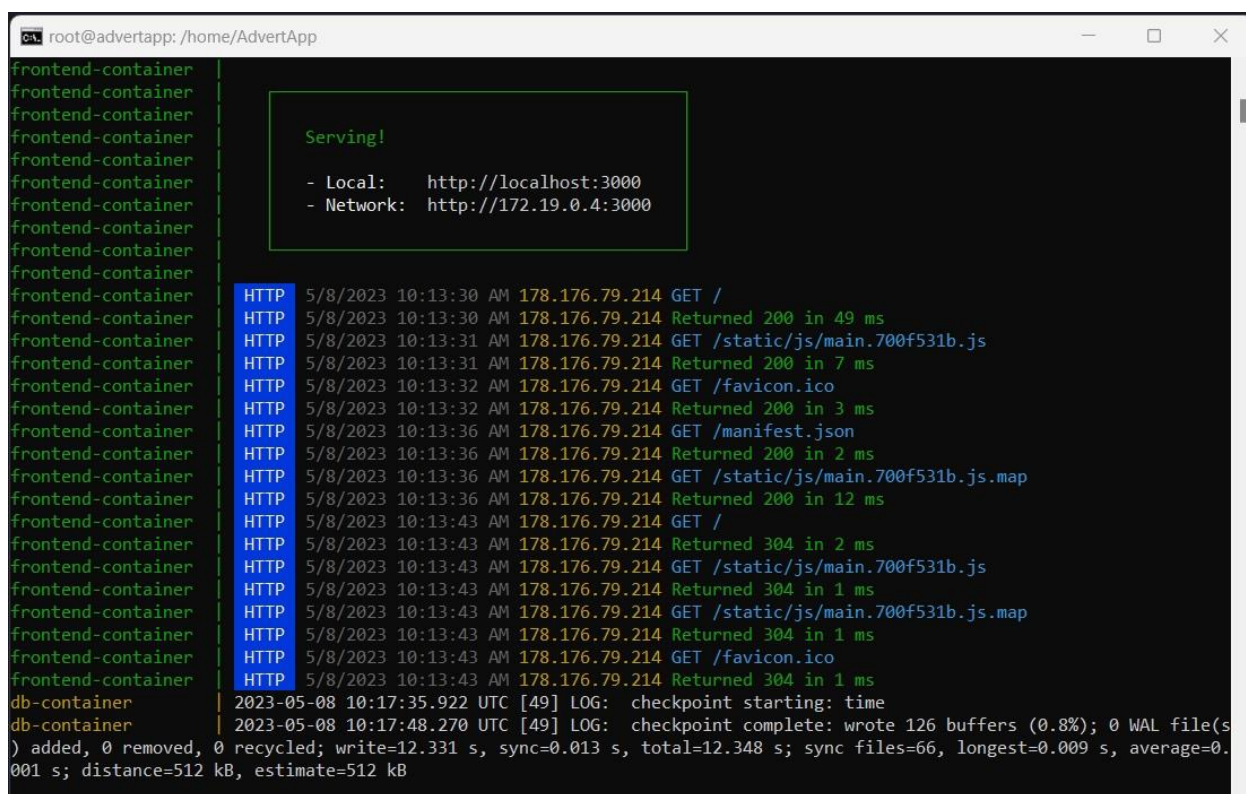
Далее проект был клонирован с репозитория на GitHub и запущен одной командой `docker compose up` в корневой папке проекта (рисунок 15).

```
root@advertapp: /home/AdvertApp
root@advertapp:/home/AdvertApp# docker compose up
[+] Running 9/9
✔ db 8 layers [#####] 0B/0B Pulled 14.7s
✔ f56be85fc22e Pull complete 1.5s
✔ 256414453fba Pull complete 1.6s
✔ f71699d7795a Pull complete 1.6s
✔ 8eff49387ec9 Pull complete 12.2s
✔ 7da7fae4e80a Pull complete 12.3s
✔ f33740282c00 Pull complete 12.3s
✔ b49740a115f2 Pull complete 12.3s
✔ c36da779701e Pull complete 12.4s
[+] Building 0.1s (2/4)
=> [internal] load build definition from Dockerfile 0.1s
=> => transferring dockerfile: 438B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/openjdk:17-alpine 0.0s
=> [internal] load metadata for docker.io/library/maven:3.8.3-openjdk-17 0.0s
```

Рисунок 15 – Старт приложения

Конечно, перед этим была произведена некоторая настройка, в частности на сервер вручную были установлены Docker и Docker Compose [15], поскольку они не предусмотрены хостингом по умолчанию. Также в корневой папке проекта в файле .env указан IP текущего хоста, чтобы фронтенд мог считать URL сервера и отправлять запросы на развернутый на этой же машине бэкенд из браузера пользователя в любой точке мира.

Развертывание приложение было выполнено успешно, что можно видеть в выводе консоли на рисунке 16. Теперь приложение полностью готово к работе с пользователями.



```
root@advertapp: /home/AdvertApp
frontend-container | Serving!
frontend-container | - Local:    http://localhost:3000
frontend-container | - Network:  http://172.19.0.4:3000
frontend-container | HTTP 5/8/2023 10:13:30 AM 178.176.79.214 GET /
frontend-container | HTTP 5/8/2023 10:13:30 AM 178.176.79.214 Returned 200 in 49 ms
frontend-container | HTTP 5/8/2023 10:13:31 AM 178.176.79.214 GET /static/js/main.700f531b.js
frontend-container | HTTP 5/8/2023 10:13:31 AM 178.176.79.214 Returned 200 in 7 ms
frontend-container | HTTP 5/8/2023 10:13:32 AM 178.176.79.214 GET /favicon.ico
frontend-container | HTTP 5/8/2023 10:13:32 AM 178.176.79.214 Returned 200 in 3 ms
frontend-container | HTTP 5/8/2023 10:13:36 AM 178.176.79.214 GET /manifest.json
frontend-container | HTTP 5/8/2023 10:13:36 AM 178.176.79.214 Returned 200 in 2 ms
frontend-container | HTTP 5/8/2023 10:13:36 AM 178.176.79.214 GET /static/js/main.700f531b.js.map
frontend-container | HTTP 5/8/2023 10:13:36 AM 178.176.79.214 Returned 200 in 12 ms
frontend-container | HTTP 5/8/2023 10:13:43 AM 178.176.79.214 GET /
frontend-container | HTTP 5/8/2023 10:13:43 AM 178.176.79.214 Returned 304 in 2 ms
frontend-container | HTTP 5/8/2023 10:13:43 AM 178.176.79.214 GET /static/js/main.700f531b.js
frontend-container | HTTP 5/8/2023 10:13:43 AM 178.176.79.214 Returned 304 in 1 ms
frontend-container | HTTP 5/8/2023 10:13:43 AM 178.176.79.214 GET /static/js/main.700f531b.js.map
frontend-container | HTTP 5/8/2023 10:13:43 AM 178.176.79.214 Returned 304 in 1 ms
frontend-container | HTTP 5/8/2023 10:13:43 AM 178.176.79.214 GET /favicon.ico
frontend-container | HTTP 5/8/2023 10:13:43 AM 178.176.79.214 Returned 304 in 1 ms
db-container       | 2023-05-08 10:17:35.922 UTC [49] LOG: checkpoint starting: time
db-container       | 2023-05-08 10:17:48.270 UTC [49] LOG: checkpoint complete: wrote 126 buffers (0.8%); 0 WAL file(s)
) added, 0 removed, 0 recycled; write=12.331 s, sync=0.013 s, total=12.348 s; sync files=66, longest=0.009 s, average=0.
001 s; distance=512 kB, estimate=512 kB
```

Рисунок 16 – Успешное развертывание приложения на сервере

3.7 Обзор разработанного программного продукта

Проведём обзор разработанного функционала опираясь на основные требования, заявленные на этапе анализа предметной области.

Так, на рисунке 17 можно наблюдать главную страницу, вкладки регистрации и авторизации.

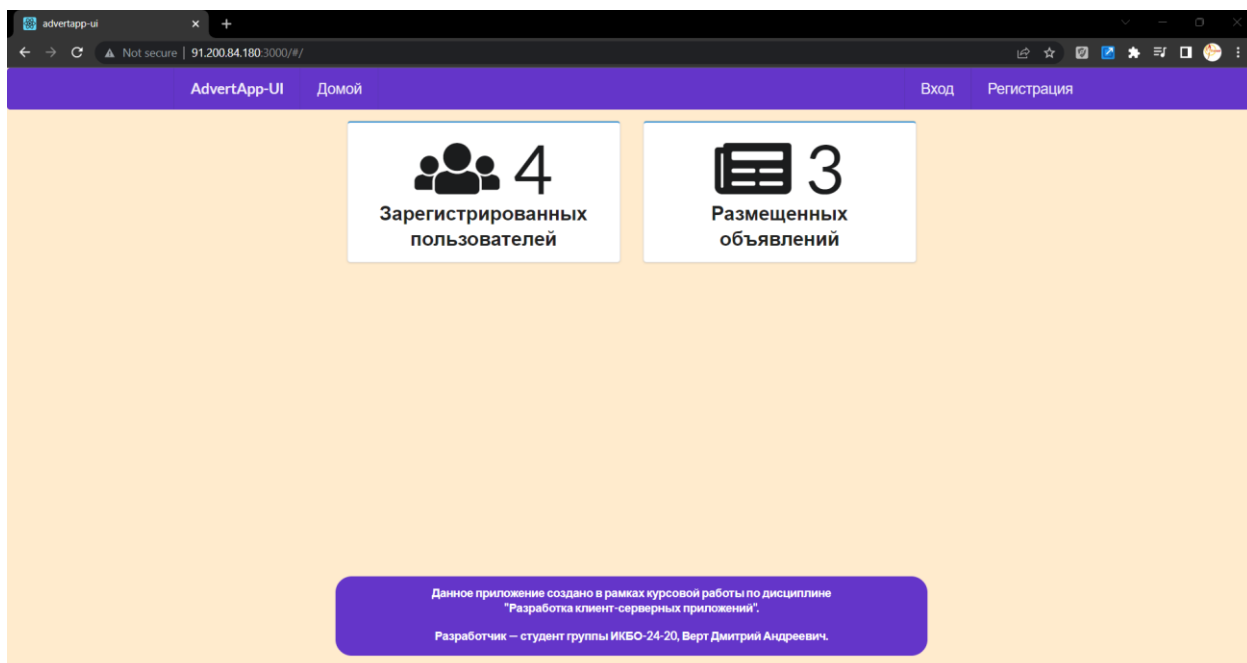


Рисунок 17 – Главная страница, вкладки входа и регистрации

После авторизации под пользователем с ролью USER становятся доступны страницы списка с функцией поиска по заголовку объявления (рисунок 18), а также форма создания своего объявления (рисунок 19).

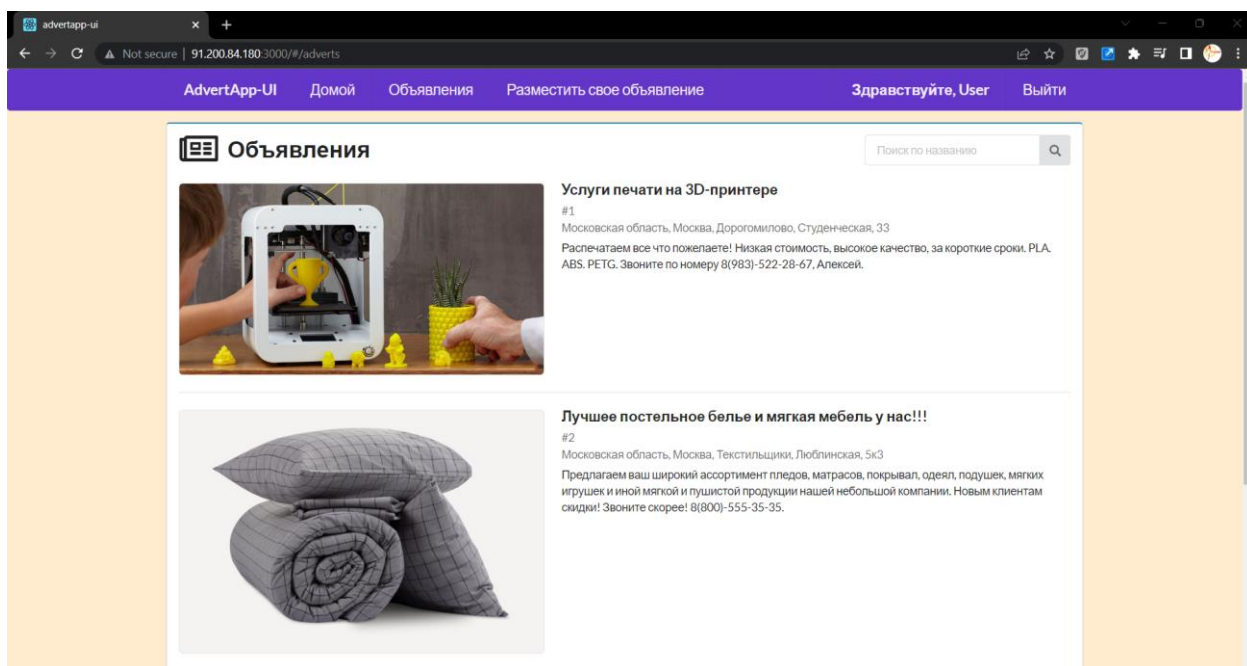


Рисунок 18 – Список объявлений

Форма создания объявления

Заголовок объявления

Область

Город

Район

Улица

Дом

Описание

Выбрать фотографию

Создать объявление

Рисунок 19 – Форма создания своего объявления

При авторизации под пользователем с ролью ADMIN становятся доступны вкладки администрирования пользователей с поиском по почте (логину) и функцией блокировки/разблокировки, а также управления объявлениями с опцией поиска по заголовку и их удалению (рисунки 20 и 21).

advertapp-ui

Not secure | 91.200.84.180:3000/#/admin

🏠

🔍

📄

🔧

📧

👤

АдvertApp-UI

Домой

Администрирование

Здравствуйте, Admin

Выйти

👤 Пользователи

📄 Объявления

Поиск по почте

🔍

	ID	First Name	Last Name	Email	Role	Status
<div><div>🚫</div><div>✅</div></div>	1	Admin		admin	ADMIN	ACTIVE
<div><div>🚫</div><div>✅</div></div>	2	User		user@mail.ru	USER	ACTIVE
<div><div>🚫</div><div>✅</div></div>	3	Алексей	Решетняк	mister-alekcei@mail.ru	USER	ACTIVE
<div><div>🚫</div><div>✅</div></div>	4	Алина	М	alinamardanova5@gmail.com	USER	ACTIVE

Рисунок 20 – Управление пользователями

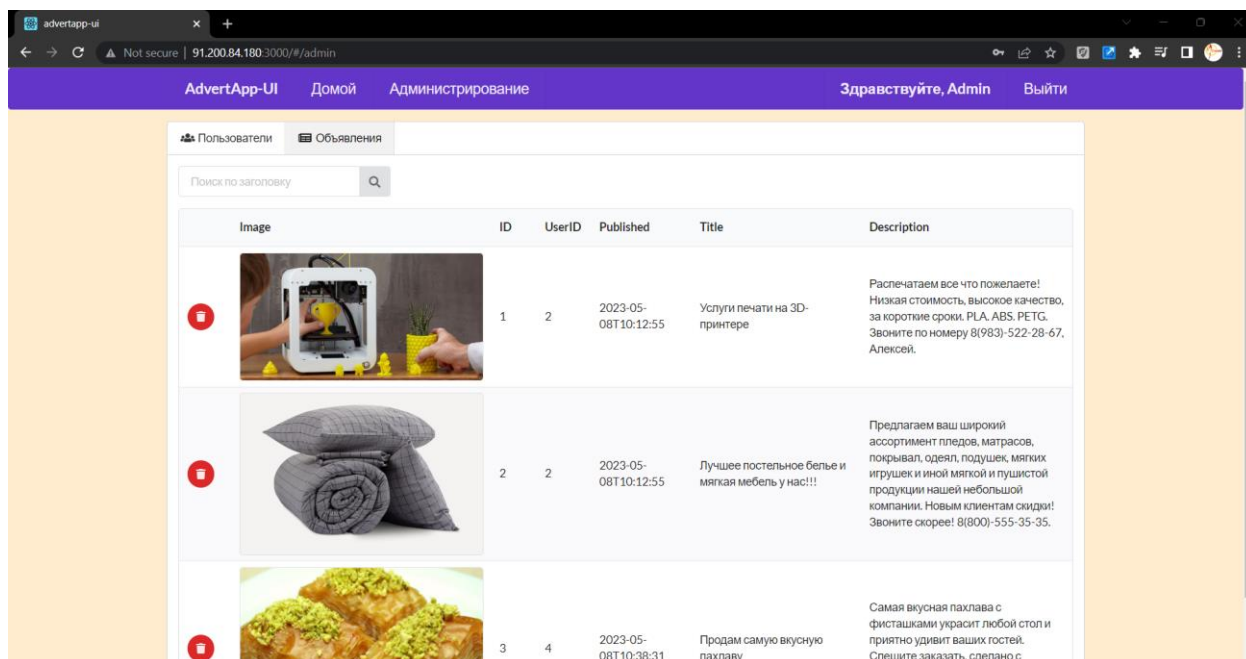


Рисунок 21 – Управление объявлениями

3.8 Подготовка к защите курсовой работы

Исходный код разработанного в рамках курсовой работы приложения загружен в сеть Интернет (рисунок 22), развертывание сервиса на выделенном сервере также выполнено. Оба ресурса доступны по соответствующим ссылкам для публичного взаимодействия (указаны в приложении А).

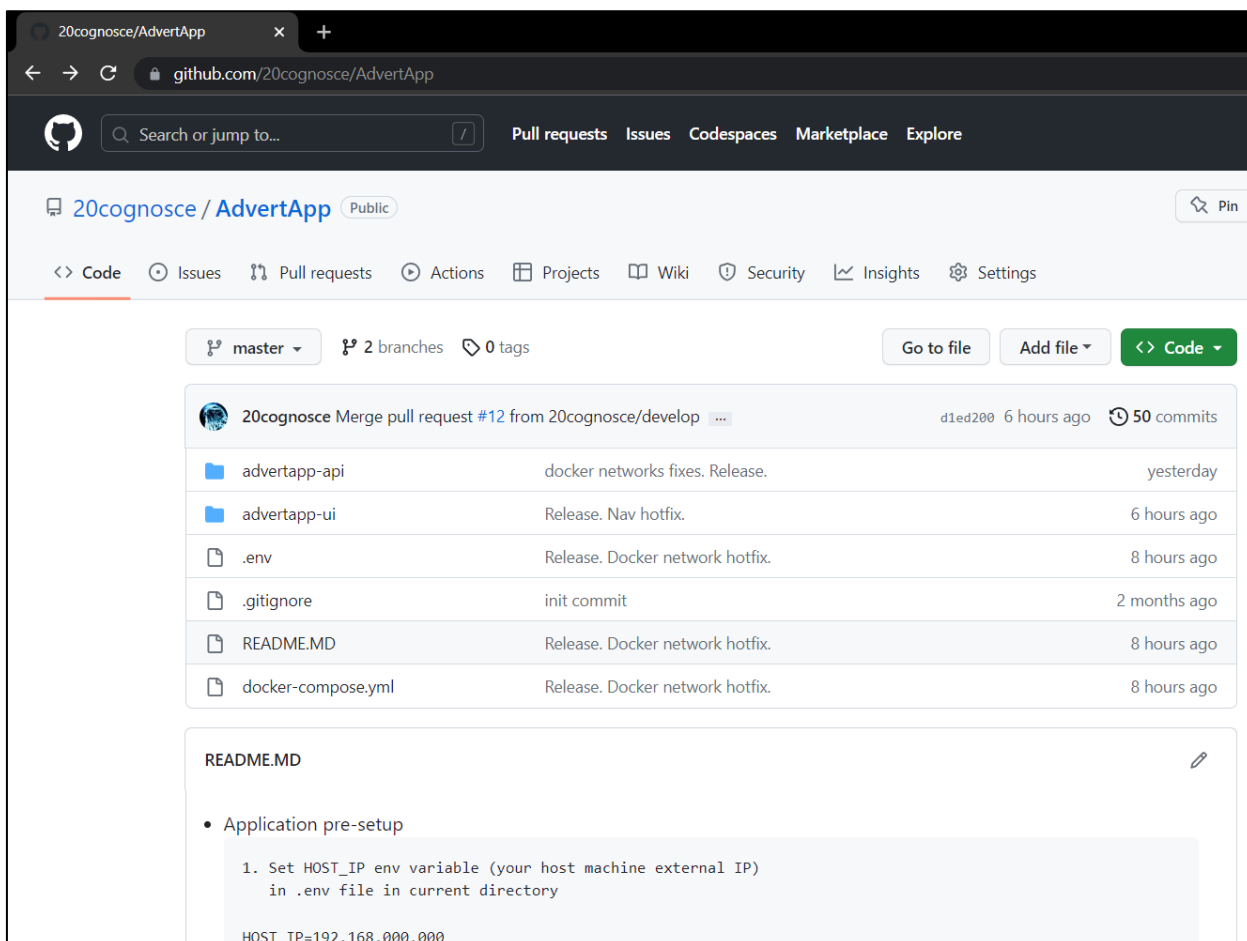


Рисунок 22 – Репозиторий проекта

ЗАКЛЮЧЕНИЕ

В результате выполнения данной курсовой работы было разработано клиент-серверное фуллстек-приложение для размещения объявлений, использующее стек технологий, основанный на Spring и React. В процессе разработки было проведено проектирование архитектуры приложения, интеграция с базой данных, настройка и развертывание приложения на выделенном сервере. Также были реализованы функциональные требования, включая возможность создания, просмотра, поиска и удаления объявлений, а также система аутентификации и авторизации пользователей посредством Basic Auth и защита маршрутов на основе Spring Security.

Был проведен анализ предметной области разработанного веб-приложения. Итоговый проект полностью работоспособен, соответствует заявленной тематике и реализует заявленный на этапе анализа функционал сервиса для размещения объявлений. Финальный продукт предоставляет возможность пользователям и администраторам просматривать, создавать и управлять объявлениями через удобный интерфейс. Таким образом, разработанный программный продукт соответствует заявленной тематике и выполняет свою основную функцию работоспособного MVP с возможностью дальнейшего развития и масштабирования для обслуживания большого количества пользователей и новых потребностей бизнеса.

По факту выполнения работы были освоены компетенции по проектированию и разработке клиент-серверных веб-приложений, интеграции с базой данных, обеспечения безопасности приложения, контейнеризации программных модулей и настройке сетевого взаимодействия между ними, а так же развертыванию системы на промышленном сервере.

Все цели и задачи, поставленные в листе задания на курсовую работу, были выполнены.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Хортон, А. Разработка веб-приложений в ReactJS / А. Хортон, Р. Вайс ; перевод с английского Р. Н. Рагимова. — Москва : ДМК Пресс, 2016. — 254 с. — ISBN 978-5-94074-819-9. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/97339> (дата обращения: 24.04.2023). — Режим доступа: для авториз. пользователей.
2. JDK 17 – OpenJDK [Электронный ресурс]. – URL: <https://openjdk.org/projects/jdk/17> (дата обращения 25.04.2023).
3. Level up your Java code and explore what Spring can do for you. [Электронный ресурс]. – URL: <https://spring.io> (дата обращения 25.04.2023).
4. PostgreSQL: The world's most advanced open source database [Электронный ресурс]. – URL: <https://www.postgresql.org> (дата обращения 25.04.2023).
5. Maven – Welcome to Apache Maven [Электронный ресурс]. – URL: <https://maven.apache.org> (дата обращения 25.04.2023).
6. React – JavaScript-библиотека для создания пользовательских интерфейсов [Электронный ресурс]. – URL: <https://ru.legacy.reactjs.org> (дата обращения 28.04.2023).
7. Docker: Accelerated, Containerized Application Development [Электронный ресурс]. – URL: <https://www.docker.com> (дата обращения 30.04.2023).
8. Уоллс, К. Spring в действии : руководство / К. Уоллс ; перевод с английского А. Н. Киселева. — 6-е изд. — Москва : ДМК Пресс, 2022. — 544 с. — ISBN 978-5-93700-112-2. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/314828> (дата обращения: 30.05.2023). — Режим доступа: для авториз. пользователей.
9. Шёниг, Г. -. PostgreSQL 11. Мастерство разработки / Г. -. Шёниг ; перевод с английского А. А. Слинкина. — Москва : ДМК Пресс, 2020. — 352 с. — ISBN 978-5-97060-671-1. — Текст : электронный // Лань : электронно-

библиотечная система. — URL: <https://e.lanbook.com/book/131714> (дата обращения: 30.05.2023). — Режим доступа: для авториз. пользователей.

10. Иванова, И. А. Основы построения контейнерной архитектуры современных приложений : учебно-методическое пособие / И. А. Иванова, И. Д. Котилевец, И. С. Хаханов. — Москва : РТУ МИРЭА, 2022. — 117 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/240035> (дата обращения: 01.05.2023). — Режим доступа: для авториз. пользователей.

11. Хоффман Эндрю X85 Безопасность веб-приложений. — СПб.: Питер, 2021. — 336 с.: ил. — (Серия «Бестселлеры O'Reilly»).

12. Тузовский, А. Ф. Проектирование и разработка web-приложений : учебное пособие для вузов / А. Ф. Тузовский. — Москва : Издательство Юрайт, 2021. — 218 с. — (Высшее образование). — ISBN 978-5-534-00515-8. — Текст : электронный // Образовательная платформа Юрайт [Сайт]. — URL: <https://urait.ru/bcode/469982> (дата обращения: 02.05.2023).

13. Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. — СПб. : Питер, 2021. — 352 с.

14. Янцев, В. В. JavaScript. Как писать программы / В. В. Янцев. — 2-е изд., стер. — Санкт-Петербург : Лань, 2023. — 200 с. — ISBN 978-5-507-47050-1. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/322520> (дата обращения: 03.05.2023). — Режим доступа: для авториз. пользователей.

15. Колисниченко, Д. Н. LINUX. Полное руководство По работе и администрированию : руководство / Д. Н. Колисниченко. — Санкт-Петербург : Наука и Техника, 2021. — 480 с. — ISBN 978-5-94387-608-0. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/175386> (дата обращения: 05.05.2023). — Режим доступа: для авториз. пользователей.

ПРИЛОЖЕНИЕ А

Исходный код клиент-серверного веб-приложения доступен по ссылке – <https://github.com/20cognosce/AdvertApp>.

URL промышленного стенда с развернутым сервисом, доступным публично для всех пользователей сети Интернет – <http://91.200.84.180:3000> (сервер оплачен на период до 07.07.2023 включительно).

Листинг 1 – schema.sql

```
CREATE TABLE IF NOT EXISTS users
(
    id            SERIAL PRIMARY KEY,
    first_name    VARCHAR(100)      NOT NULL,
    last_name     VARCHAR(100)      NOT NULL,
    phone         VARCHAR(25) UNIQUE,
    email         VARCHAR(50) UNIQUE NOT NULL,
    hash_password VARCHAR,
    role          VARCHAR(50),
    status        VARCHAR(50)
);

create table IF NOT EXISTS address
(
    id            SERIAL PRIMARY KEY,

    county        VARCHAR(100),
    city          VARCHAR(50),
    district      VARCHAR(50),
    street        VARCHAR(100),
    house_number  VARCHAR(10)
);

CREATE TABLE IF NOT EXISTS advert
(
    id            SERIAL PRIMARY KEY,
    title         VARCHAR(255) NOT NULL,
    description    TEXT         NOT NULL,
    published     TIMESTAMP     NOT NULL,
    updated       TIMESTAMP,
    address_id    INT,
    user_id       INT,
    FOREIGN KEY (address_id) REFERENCES address (id),
    FOREIGN KEY (user_id) REFERENCES users (id)
);

CREATE TABLE IF NOT EXISTS image
(
    id            SERIAL PRIMARY KEY,
    title         VARCHAR(255),
    path          VARCHAR(255) NOT NULL,
    advert_id     INT,
    FOREIGN KEY (advert_id) REFERENCES advert (id)
);
```

Листинг 2 – Логика сохранения изображения в файловой системе

```
@Autowired
public ImageService(@Value("${image-storage-dir}") Path imageStorageDir,
                    ImageRepository imageRepository,
                    AdvertService advertService) {
    this.imageStorageDir = imageStorageDir;
    this.imageRepository = imageRepository;
    this.advertService = advertService;
}

@PostConstruct
public void ensureDirectoryExists() throws IOException {
    if (!Files.exists(this.imageStorageDir)) {
        Files.createDirectories(this.imageStorageDir);
    }
}

public String uploadImage(MultipartFile image, Long advertId, String title)
throws IllegalArgumentException {
    Advert advert = advertService.getById(advertId);

    if (!verifyImage(image)) {
        throw new IllegalArgumentException("File extension is not
supported");
    }

    Image imageEntity = saveImageToFileSystem(image, advert)
        .orElseThrow(() -> new IllegalArgumentException("Failed to save
image"));
    imageEntity.setTitle(title);
    imageEntity.setAdvert(advert);

    imageRepository.save(imageEntity);
    return imageEntity.getPath();
}

private Optional<Image> saveImageToFileSystem(MultipartFile image, Advert
advert) {
    File folder = new File(imageStorageDir + "/" + advert.getId());
    folder.mkdir();

    String targetFileName = image.getOriginalFilename();
    Path targetPath =
folder.toPath().resolve(Objects.requireNonNull(targetFileName));

    try (InputStream in = image.getInputStream()) {
        try (OutputStream out = Files.newOutputStream(targetPath,
StandardOpenOption.CREATE)) {
            in.transferTo(out);
        }
    } catch (IOException e) {
        return Optional.empty();
    }

    return Optional.of(
        Image.builder()
            .path(targetPath.toString())
            .build());
}
```

Листинг 3 – Метод загрузки изображения из файловой системы

```
public Resource downloadImage(@PathVariable("id") Long id) {
    Image image = imageRepository.findById(id)
        .orElseThrow(() -> new EntityNotFoundException("Image with id = "
+ id + " not found"));

    File foundFile = new File(image.getPath());

    return new PathResource(foundFile.getPath());
}
```

Листинг 5 – Фильтр безопасности для эндпоинтов приложения

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
    http.authorizeHttpRequests()
        .requestMatchers(HttpMethod.POST, "/auth/**").permitAll()
        .requestMatchers(HttpMethod.GET, "/adverts/count",
"/users/count", "/images/**").permitAll()
        .requestMatchers("/", "/error").permitAll()

        .requestMatchers(HttpMethod.POST,
"/adverts/**").hasAnyAuthority(ADMIN.name(), USER.name())
        .requestMatchers(HttpMethod.GET,
"/adverts/**").hasAnyAuthority(ADMIN.name(), USER.name())

        .requestMatchers(HttpMethod.POST, "/users/**").hasAuthority(ADMIN.name())
        .requestMatchers(HttpMethod.GET, "/users/**").hasAuthority(ADMIN.name())
        .requestMatchers(HttpMethod.DELETE, "/adverts/**").hasAuthority(ADMIN.name())

        .anyRequest().authenticated()
        .and()
        .cors()
        .and()
        .csrf().disable() //TODO: should be fixed in the future
        .httpBasic();

    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    return http.build();
}
```

Листинг 8 – docker-compose.yml

```
version: '3.8'

services:
  db:
    image: postgres:alpine
    container_name: db-container
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
      POSTGRES_DB: advert-db
    ports:
      - '5432:5432'
```

```
backend:
  build: ./advertapp-api
  container_name: backend-container
  restart: always
  environment:
    SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/advert-db
    SPRING_DATASOURCE_USERNAME: postgres
    SPRING_DATASOURCE_PASSWORD: password
    SPRING_DATASOURCE_DRIVER_CLASS_NAME: org.postgresql.Driver
    APP_CORS_ALLOWED_ORIGINS: http://frontend:3000 # is ignored for now
  depends_on:
    - db
  ports:
    - '8080:8080'
  healthcheck:
    test: [ "CMD", "curl", "-f", "http://localhost:8080" ]
    interval: 30s
    timeout: 10s
    retries: 5

frontend:
  build:
    context: ./advertapp-ui
    args:
      BACKEND_BASE_URI: http://${HOST_IP}:8080
  container_name: frontend-container
  restart: on-failure
  stdin_open: true
  tty: true
  depends_on:
    backend:
      condition: service_healthy
  ports:
    - '3000:3000'
```