

Заведующему кафедрой
инструментального и прикладного
программного обеспечения (ИиППО)
Института информационных технологий (ИТ)

Болбакову Роману Геннадьевичу

От студента Властюка Артёма Валерьевича

ФИО

ИКБО-01-21

группа

3 курс

курс

Контакт: + 7 (929) 928-78-68,

Cvlastyuk@mail.ru

Заявление

Прошу утвердить мне тему курсовой работы по дисциплине «Разработка клиент-серверных приложений» образовательной программы бакалавриата 09.03.04 (Программная инженерия).

Тема: Разработка клиент-серверного фуллстек приложения список случайных пользователей при каждом обновлении.

Подпись студента



подпись

/ Властюк А.В.

ФИО

Дата

26.02.24

Подпись руководителя



подпись

ст.п. Сеницын А.В.

Должность, ФИО

Дата

26.02.24





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: Властюк Артём Валерьевич

Группа: ИКБО-01-21

Срок представления к защите: 22.05.2023

Руководитель: Синицын Анатолий Васильевич, старший преподаватель

Тема: Разработка клиент-серверного фуллстек приложения список случайных пользователей при каждом обновлении

Исходные данные: HTML5, CSS3, JavaScript, React, Node.js, JetBrains WebStorm, SQL СУБД PostgreSQL

Перечень вопросов, подлежащих разработке, и обязательного графического материала: 1. Провести анализ предметной области для выбранной темы, обосновать выбор клиент-серверной архитектуры и дать её детальное описание с помощью UML для разрабатываемого приложения; 2. Выбрать программный стек для реализации фуллстек разработки, ориентируясь на мировой опыт и стандарты в данной области; 4. Реализовать фронтенд и бекенд части клиент-серверного приложения, обеспечивать авторизацию и аутентификацию пользователя; 5. Разместить готовое клиент-серверное приложение в репозитории GitHub с указанием ссылки в тексте отчёта; 6. Развернуть клиент-серверное приложение в облаке. 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: [подпись] / Болбаков Р. Г. /, «26» февраля 2024 г.

Задание на КР выдал: [подпись] / Синицын А.В. /, «26» февраля 2024 г.

Задание на КР получил: [подпись] / Властюк А.В. /, «26» февраля 2024 г.

РЕФЕРАТ

Отчёт 43 с., 33 рис., 18 источн.

SPRING, JAVA, REACT, POSTGRESQL, JWT, ПРИЛОЖЕНИЕ

Объектом разработки является разработка клиент-серверного фуллстек приложения список случайных пользователей при каждом обновлении.

Цель работы – разработка архитектуры, серверной и клиентской части приложения, которое выдаёт список случайных пользователей.

В ходе работы был проведён анализ предметной области и обзор сайтов с аналогичной тематикой. Методом анализа определены необходимые сущности и возможности, которые должны быть включены в приложение.

Рассмотрен процесс разработки архитектуры, выбор необходимых технологий для разработки и процесс создания приложения.

Результатом является развёрнутое на облачном сервисе клиент-серверное приложение по поиску случайных пользователей, готовое для использования.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	5
ВВЕДЕНИЕ	6
1 ОБЩИЕ СВЕДЕНИЯ	7
1.1 Обозначение и наименование веб-приложения.....	7
1.2 Функциональное назначение.....	7
2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ.....	8
3 ВЫБОР И ОБОСНОВАНИЕ ТЕХНОЛОГИЙ	9
3.1 Выбор и обоснование средств разработки	9
3.2 Выбор и обоснование архитектуры	10
4 РАЗРАБОТКА АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ НА ОСНОВЕ ВЫБРАННОГО ПАТТЕРНА.....	11
5 РАЗРАБОТКА СЕРВЕРНОЙ И КЛИЕТСКОЙ ЧАСТИ ВЕБ- ПРИЛОЖЕНИЯ	14
5.1 Разработка слоя моделей	14
5.2 Разработка слоя репозитория.....	17
5.3 Разработка слоя сервисов	17
5.4 Разработка слоя контроллеров	19
5.5 Создание конфигурационных классов	21
5.6 Разработка клиентской части	22
6 РАЗВЕРТЫВАНИЕ ПРИЛОЖЕНИЯ	30
7 ПРОВЕДЕНИЕ ТЕСТИРОВАНИЯ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ	33
ЗАКЛЮЧЕНИЕ.....	41
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	42

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

CRUD	– Create, Read, Update, Delete (Создавать, читать, обновлять, удалять)
DI	– Dependency Injection (Внедрение зависимости)
IDE	– Integrated Development Environment (Интегрированная среда разработки)
JPA	– Java Persistence API (Java API хранения данных)
JSON	– JavaScript Object Notation (Нотация объектов JavaScript)
ORM	– Object-Relational Mapping (Отображение объектно-ориентированных структур на реляционные данные)
HTML	– HyperText Markup Language (стандартизированный язык разметки документов для просмотра веб-страниц в браузере)
IoC	– Inversion of Control (Инверсия управления)
JDK	– Java Development Kit (бесплатно распространяемый компанией Oracle Corporation комплект разработчика приложений на языке Java)
CSS	– Cascading Style Sheets (формальный язык описания внешнего вида документа, написанного с использованием языка разметки)

ВВЕДЕНИЕ

В современном обществе информационные технологии тесно переплетены с сферами государства, общественной жизни и бизнеса, играя важнейшую роль в организации различных процессов. Эффективное функционирование в наши дни становится невозможным без применения передовых технологий, которые значительно упрощают, систематизируют и оптимизируют рабочие процессы. Разработка веб-приложений в данном контексте представляет собой сложную и многогранную задачу, требующую от разработчика глубокого понимания архитектуры, технологий и инструментов.

Для выполнения поставленной цели курсовой работы необходимо выполнить следующие пункты:

- провести анализ предметной области разрабатываемого веб-приложения;
- обосновать выбор технологий разработки веб-приложения;
- разработать архитектуру веб-приложения на основе выбранного паттерна проектирования;
- реализовать слой серверной логики веб-приложения с применением выбранной технологии;
- реализовать слой логики базы данных;
- разработать слой клиентского представления веб-приложения;
- тестирование приложения;
- развёртывание приложения в облаке
- создать презентацию по выполненной курсовой работе;

1 ОБЩИЕ СВЕДЕНИЯ

1.1 Обозначение и наименование веб-приложения

Темой разработанной серверной части веб-приложения является «список случайных пользователей при каждом обновлении». Разработанное приложение получило название – «Friends Findr».

1.2 Функциональное назначение

Функциональное назначение веб-приложения для поиска пользователей заключается в том, чтобы предоставить им возможность найти новых неизвестных людей, просмотреть их профиль, а также создать свой. Кроме того, пользователи могут найти контактную информацию для связи с пользователями.

2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Для анализа предметной области было проведено исследование похожих сайтов. Были выявлены их преимущества и недостатки, представленные в таблице 1. Были взяты сайты, которые позволяют найти новые знакомства, они также предоставляют возможность поиска случайных пользователей. Также было найдено множество других сайтов, которые связаны с loveplanet. Они имеют такой же интерфейс и возможности. Поэтому они не были включены в таблицу.

Таблица 1 – Сравнение аналогичных приложений

Название веб-сервиса	loveplanet	mamba	podbor
Поиск случайных пользователей	+	+	+
Возможность поиска романтических партнёров	+	+	+
Возможность поиска друзей	+	+	+
Возможность регистрации до 18 лет	-	-	-
Возможность отправки сообщений	+	+	+

После анализа всех сайтов с похожей тематикой был сделан вывод, что большая часть сайтов по поиску знакомств направлена на романтические отношения. Найти друзей на таких сайтах будет тяжело. Также на большинстве сайтов необходимо начинать общение в мессенджере сайта, что является не особым удобством. Поэтому было решено было не делать возможность чата. Вместо этого пользователи оставляют в своём профиле ссылки на другие социальные сети.

Главной целью веб-приложения будет иметь возможность поиска различных людей, цель которых будет не поиск романтических отношений. Также будет предоставлена возможность регистрации с 14 лет.

3 ВЫБОР И ОБОСНОВАНИЕ ТЕХНОЛОГИЙ

3.1 Выбор и обоснование средств разработки

Было решено заменить Node Js [1] на Java, а частности был использован фреймворк Spring [2]. Это было обосновано несколькими причинами. Spring – это зрелая платформа, известная своей масштабируемостью. Она хорошо подходит для обработки большого количества запросов и данных, что важно для сайта поиска друзей, где может быть много активных пользователей. Также важным пунктом была безопасность. Spring имеет встроенные механизмы безопасности, которые легко настроить. Spring Security [4] легко интегрируется с существующими приложениями Spring. Он использует конфигурацию на основе аннотаций [5], что делает его более удобным для разработчиков. Безопасность пользователей важный фактор, поэтому было решено писать серверную часть на Java.

Выбор этих технологий был обоснован несколькими причинами. Во-первых, Java является одним из самых популярных языков программирования в мире, и он имеет множество библиотек и фреймворков, которые облегчают разработку приложений. Spring Framework [7] является одним из наиболее популярных фреймворков для разработки веб-приложений на Java, и он обладает широкими возможностями для работы с веб-сервисами и базами данных. Hibernate [6] был выбран для работы с базами данных, так как он предоставляет удобный и простой интерфейс для работы с различными СУБД, включая PostgreSQL [8]. PostgreSQL была выбрана как база данных для приложения из-за её открытости, надежности и возможности масштабирования. HTML и CSS [9] были выбраны для создания интерфейса веб-приложения, так как они являются стандартными языками для разработки веб-страниц и обладают широкой поддержкой в браузерах. Браузер Yandex был использован для тестирования, так как это один из самых популярных браузеров, который обеспечивает высокую скорость работы и большую функциональность для разработчиков. Для удобства создания пользовательского интерфейса и упрощения работы с языками веб-разработки

задействован фреймворк React, который позволяет декомпозировать веб-приложение на компоненты, за счет чего достигается высокий уровень масштабируемости и поддержки кода.

3.2 Выбор и обоснование архитектуры

Сервер построен на монолитной архитектуре с использованием шаблона проектирования MVC. MVC разделяет приложение на три основных компонента: Model (модель), View (представление) и Controller (контроллер). Такое разделение делает код более организованным и понятным.

Модель (Model): Модель представляет собой основные данные приложения и бизнес-логику. Она отвечает за управление данными, их валидацию, сохранение и обновление. Модель не зависит от представления или контроллера, что обеспечивает ее независимость от способа отображения данных и способа пользовательского взаимодействия.

Представление (View): Представление отображает данные пользователю и обрабатывает пользовательский ввод. Оно отвечает за отображение информации пользователю в форме, понятной пользователю. Представление обычно отвечает за форматирование данных и выполняет валидацию пользовательского ввода.

Контроллер (Controller): Контроллер обрабатывает взаимодействие пользователя с приложением. Он принимает пользовательский ввод, обрабатывает запросы и обновляет модель при необходимости. Контроллер также обновляет представление по запросу, чтобы отобразить изменения в данных.

Важным аспектом MVC является разделение ответственности между этими компонентами. Это позволяет легче поддерживать и развивать приложение, так как каждая часть отвечает только за свои задачи. Например, разработчики пользовательского интерфейса могут работать над представлением без участия в бизнес-логике или управлении данными.

4 РАЗРАБОТКА АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ НА ОСНОВЕ ВЫБРАННОГО ПАТТЕРНА

Для разработки архитектуры веб-приложения следует выявить бизнес-правила, на основе которых будет строиться информационная система. На Рисунке 1 представлена диаграмма вариантов использования UML для проектируемой информационной системы:



Рисунок 1 - Диаграмма вариантов использования

Создание веб-приложения было начато с создания моделей, которые описывают основные сущности предметной области проекта. Шаблон «модель —представление —контроллер» (MVC) разделяет функциональность приложения на компоненты трех видов: модель — содержит данные приложения, представление — отображает некоторую часть базовых данных и взаимодействует с пользователем, контроллер — действует в качестве посредника между моделью и представлением и управляет уведомлениями об изменении состояния.

Выделим сущности (Models). Предлагается 2 основных бизнес-объектов: Город и Пользователь. На рисунке 2 представлена схема базы данных, в которой хранятся эти сущности.

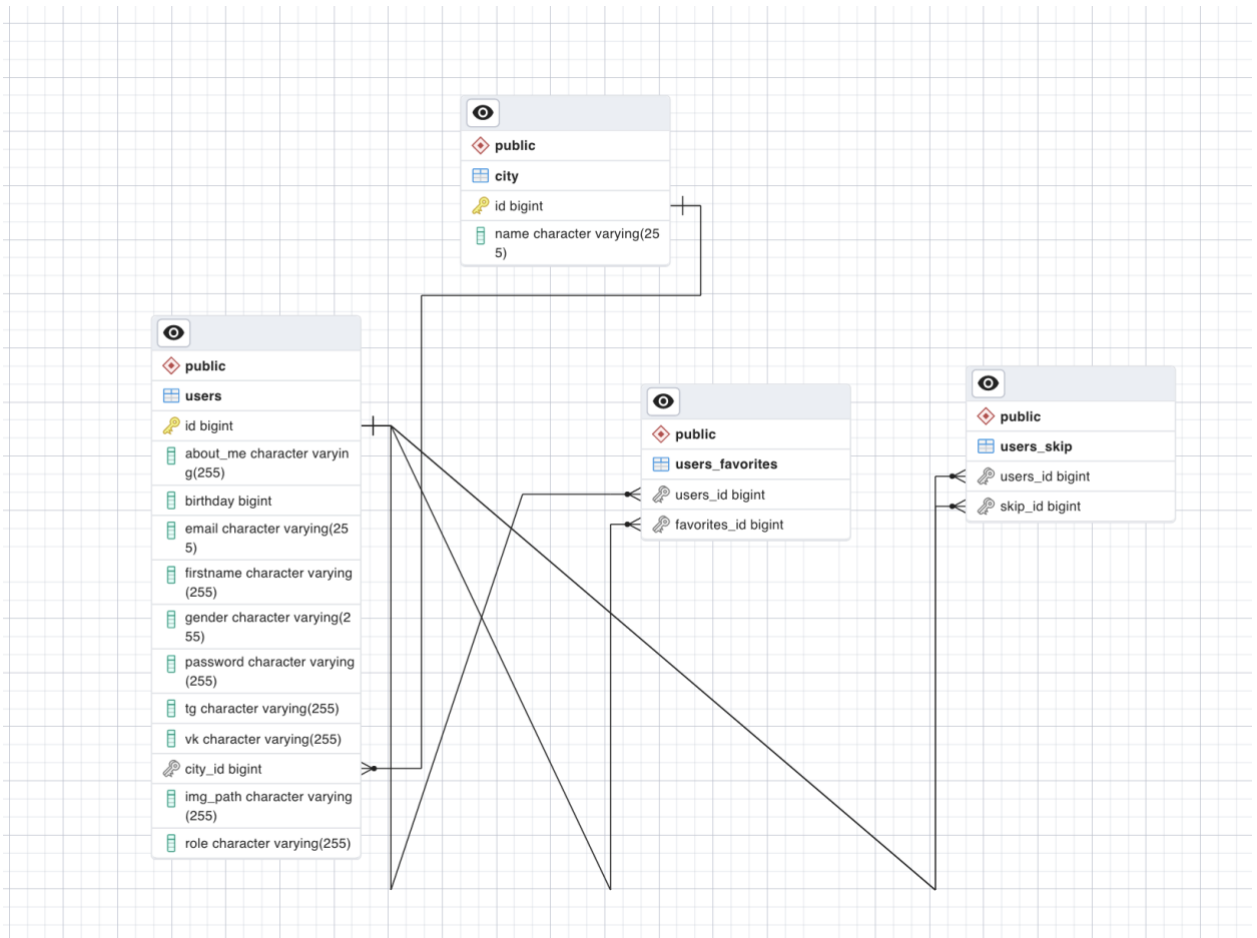


Рисунок 2 – Схема базы данных

Для взаимодействия с базой данных, где будут храниться создадим слой репозитория. Они представляют интерфейс для работы с таблицами и данными в базе данных, а также предоставляют методы для выполнения различных операций, таких как добавление, чтение, обновление и удаление данных.

Для разделения бизнес-логики создадим слой Services [11]. Они будут хранить всю бизнес-логику приложения. Services в MVC используются для организации бизнес-логики приложения и отделения этой логики от контроллеров. Они предоставляют методы для выполнения операций с данными, таких как чтение, создание, обновление и удаление (CRUD). Services также могут содержать дополнительную логику, такую как проверка прав

доступа, валидацию данных и обработку ошибок.

Далее стоит приступить к созданию Контроллеров. Контроллеры в MVC используются для обработки запросов от клиентов и возвращения ответов в виде HTML-страниц, JSON-объектов или других форматов данных. Они являются основным механизмом управления бизнес-логикой приложения и обеспечивают связь между моделью, представлением и пользовательским интерфейсом.

Также последним стоит разработать Представление. Этот компонент отвечает за отображение данных пользователю. Он получает данные от модели и отображает их пользователю в удобной форме, например, в виде веб-страницы или графического интерфейса.

5 РАЗРАБОТКА СЕРВЕРНОЙ И КЛИЕТСКОЙ ЧАСТИ ВЕБ-ПРИЛОЖЕНИЯ

5.1 Разработка слоя моделей

Было реализовано две модели: модель пользователя (User) и модель города (City). Каждая из моделей имеет свои поля и методы, которые описывают ее поведение в системе.

Модель User содержит следующие поля: id – идентификатор с аннотациями @Id и @GeneratedValue с параметром, который позволяет быть id уникальным, email – адрес электронной почты пользователя, который используется при авторизации на сайте, firstname – имя пользователя, password – пароль пользователя для входа в аккаунт, birthday – дата рождения, gender – пол человека, aboutMe – описания пользователя. Также у модели присутствуют методы, которые позволяют установить, редактировать или получить поля. На Листинге 1 представлена модель, реализующая сущность пользователя.

Листинг 1 – Код модели пользователя

```
package org.example.entity;
import jakarta.persistence.*;
import lombok.*;
import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.authority.SimpleGrantedAutho
rity;
import
org.springframework.security.core.userdetails.UserDetails;
import java.util.Collection;
import java.util.List;
@Data
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
@Entity(name="users")
public class User implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column
    private String firstname;
    @Column
    private long birthday;
```

Продолжение Листинга 1

```
@ManyToOne
private City city;
@Column
private String gender;
@Column
private String email;
@Column
private String password;
@Column
private String aboutMe;
@Column
@Enumerated(EnumType.STRING)
private Role role;
@Column
private String vk;
@Column
private String imgPath;
@Column
private String tg;
@ManyToMany
private List<User> favorites;
@ManyToMany
private List<User> skip;
@Override
public String toString() {
    return "User{" +
        "id=" + id +
        ", firstname='" + firstname + '\'' +
        '}';
}
@Override
public Collection<? extends GrantedAuthority>
getAuthorities() {
    return List.of(new
SimpleGrantedAuthority(role.name()));
}
@Override
public String getUsername() {
    return email;
}
@Override
public boolean isAccountNonExpired() {
    return true;
}
@Override
public boolean isAccountNonLocked() {
    return true;
}
}
```

Для хранения города была реализована модель City. Она содержит следующие поля: id – индикатор города, name – название города. Код данного класса представлен на Листинге 2.

Листинг 2 – Код модели города

```
package org.example.entity;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import java.util.List;
@Entity
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class City {
    @Id
    private Long id;
    @Column
    private String name;
    @OneToMany(mappedBy = "city")
    List<User> users;
    public City(Long id, String name) {
        this.id = id;
        this.name = name;
    }
}
```

Также создан класс UserDao, он используется при ответе на запрос информации о пользователе. Код класса представлен на Листинге 3.

Листинг 3 – Код класса UserDao

```
@Setter
@Getter
@Builder
public class UserDao {
    private Long id;
    private String firstname;
    private long birthday;
    private String gender;
    private String email;
    private String aboutMe;
    private String city;
    private String vk;
    private String tg;
    private String img;
}
```


5.2 Разработка слоя репозитория

Репозитории [12] используются для доступа к базе данных. Они представляют интерфейс для работы с таблицами и данными в базе данных, а также предоставляют методы для выполнения различных операций, таких как добавление, чтение, обновление и удаление данных.

Было реализовано два репозитория для каждой модели (Рисунок 3). В каждом интерфейсе были добавлены методы, которые позволяют выполнять все нужные операции.

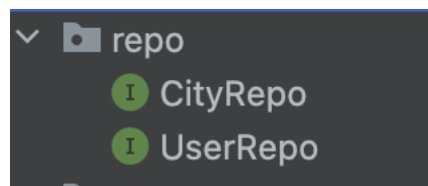


Рисунок 3 – Разработанные репозитории

5.3 Разработка слоя сервисов

Services в Spring MVC [11] используются для организации бизнес-логики приложения и отделения этой логики от контроллеров. Они предоставляют методы для выполнения операций с данными, таких как чтение, создание, обновление и удаление (CRUD). Services также могут содержать дополнительную логику, такую как проверка прав доступа, валидацию данных и обработку ошибок.

Services в Spring MVC обычно связываются с контроллерами через Dependency Injection (DI), что позволяет упростить код контроллеров и сделать его более модульным и легко тестируемым. Кроме того, использование Services позволяет легко изменять бизнес-логику приложения без необходимости изменения контроллеров.

Для реализации бизнес-логики приложения были реализованы классы-сервисы: UserService и CityService.

UserService содержит поля userRepo, passwordEncoder, jwtService, authenticationManager. Поле userRepository предназначено для взаимодействия с базой данных пользователя, а passwordEncoder позволяет кодировать пароль пользователей, jwtService нужен для взаимодействия с токеном пользователя,

authenticationManager для проверки данных при авторизации.

В классе UserService были созданы следующие методы: save() – метод, который сохраняет пользователя в базу данных при помощи метода save() из репозитория пользователя, getUserByToken() позволяет получить пользователя по токену при помощи метода findByEmail() из репозитория пользователя, findUserById() – метод, который возвращает пользователя по идентификатору, auth() – метод, который проверяет данные пользователя и возвращает jwtToken в случае успешной авторизации, change() – изменяет данные пользователя, getRandomUsersByToken() – метод, который возвращает случайных пользователей из базы данных (код этого метода представлен в Листинге 4), skipUser() – записывает в базу данных пропущенных пользователей для того, чтобы они не повторялись, likeUser() – добавляет пользователя в избранное, getFavorites() – получение списка избранных пользователей.

Листинг 4 – Код метода получения случайных пользователей

```
public List<UserDao> getRandomUsersByToken(String token, int
limit) {
    User
user=userRepo.findByEmail(jwtService.extractUserName(token));
    List<User> users=userRepo.findAll();
    Random random = new Random();

    int size=users.size();
    List<User> skip=user.getSkip();
    List<User> favorites=user.getFavorites();
    List<UserDao> randomUsers=new ArrayList<>();
    List<User> viewedUsers=new ArrayList<>();

    for (int i =0; i<limit;i++){

        while (true){
            int randomInt=random.nextInt(size);
            User randomUser=users.get(randomInt);
            if(viewedUsers.size()==size){
                break;
            }
            if(!viewedUsers.contains(randomUser)){
                viewedUsers.add(randomUser);
            }
        }
    }
}
```

Продолжение Листинга 4

```
        else {
            continue;
        }
        if(!skip.contains(randomUser) &&
            !(favorites.contains(randomUser))
            &&(!Objects.equals(randomUser.getId(),
user.getId())))
        ){
            randomUsers.add(UserDao.builder().
firstname(randomUser.getFirstname()).
                aboutMe(randomUser.getAboutMe()).
birthday(randomUser.getBirthday()).
city(randomUser.getCity().getName()).
                img(randomUser.getImgPath()).
                id(randomUser.getId()).build());

            break;
        }
    }
}
return randomUsers;
}
```

В классе `CityService` представленно 3 метода: `setCities()`, `getAll()` и `getById()`. Самый важный метод `setCities()` он делает запрос к `api` и получает список всех городов России и записывает их в базу данных.

5.4 Разработка слоя контроллеров

Контроллеры в `Spring MVC` используются для обработки запросов от клиентов и возвращения ответов в виде `HTML`-страниц, `JSON`-объектов или других форматов данных. Они являются основным механизмом управления бизнес-логикой приложения и обеспечивают связь между моделью, представлением и пользовательским интерфейсом.

Было создано шесть контроллеров: контроллер городов и контроллер пользователя.

`Spring Framework` предоставляет ряд аннотаций для работы с контроллерами. Одной из наиболее часто используемых является `@Controller`,

которая указывает, что класс является контроллером и будет обрабатывать входящие HTTP-запрос. Другие полезные аннотации включают @GetMapping, @PostMapping, @PutMapping и @DeleteMapping, которые указывают тип запроса, который будет обработан методом контроллера. @RequestParam – для чтения параметров запроса HTTP.

Контроллер пользователя хранит методы, которые позволяют зарегистрировать, авторизоваться, получить информацию о различных пользователях, изменить информацию о пользователе, добавить пользователя в избранное.

Контроллер города хранит 2 метода. Метод, который записывает данные о городах в базу данных и метод, который возвращает список всех городов.

Код контроллера представлен на Листинге 5.

Листинг 5 – Контроллер города

```
import org.example.service.CityService;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import
org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@AllArgsConstructor
public class Cont {
    private final CityService cityService;
    @PostMapping("/set-city")
    @CrossOrigin(origins = "http://localhost:3000")
    public ResponseEntity<?> setCity() {
        cityService.setCities();
        return new ResponseEntity<>("OK", HttpStatus.OK);
    }
    @GetMapping("/cities")
    @CrossOrigin(origins = "http://localhost:3000")
    public ResponseEntity<?> getCities() {

        return new ResponseEntity<>(cityService.getAll(),
        HttpStatus.OK);
    }
}
```

5.5 Создание конфигурационных классов

Было создано 5 конфигурационных классов. Класс для Spring Security, JwtService, JwtAuthFilter, CorsConfig, ApplicationConfig.

Класс конфигурации [3] Spring Security хранит метод, который позволяет ограничить доступ к различным страницам. В этом методе были ограничены страницы администратора от обычных пользователей. Также создан метод, который шифрует пароли для безопасного хранения в базе данных. Код класса представлен на Листинге 6.

Листинг 6 – Код конфигурационного класса

```
@EnableWebSecurity
@Configuration
@RequiredArgsConstructor
public class SecurityConfig {
    private final JwtAuthFilter jwtAuthFilter;
    private final AuthenticationProvider
authenticationProvider;
    @Bean
    public SecurityFilterChain
securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf()
            .disable()
            .authorizeHttpRequests()

            .requestMatchers("/api/v1/users/**", "/api/v1/users", "/set-
city", "/cities", "/favorites", "/random-users")
                .authenticated()
                .anyRequest()
                .permitAll()
                .and()
                .sessionManagement()

            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()

            .authenticationProvider(authenticationProvider)
                .addFilterBefore(jwtAuthFilter,
UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }
}
```

Класс JwtService нужен для настройки jwt токена. Он хранит методы для проверки токена, времени его работы и извлечения различных данных.

Класс CorsConfig, данный класс настраивает CORS для приложения. Метод addCorsMappings определяет, какие запросы и методы разрешены для доступа с других доменов. В данном случае, все запросы к приложению будут разрешены с домена "http://localhost:3000" и с использованием методов "GET", "POST", "PUT", "PATCH", "DELETE", "OPTION" и "HEAD". Код метода представлен на Листинге 7.

Листинг 7 – Код конфигурационного класса

```
@Configuration
public class CorsConfig implements WebMvcConfigurer {
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
            .allowedOrigins("http://localhost:3000")
            .allowedMethods("GET", "POST", "PUT", "PATCH",
                "DELETE", "OPTION", "HEAD");
    }
}
```

Этот класс поможет приложению обрабатывать запросы от других доменов с установленными правилами безопасности CORS.

Класс ApplicationConfig хранит методы декодирования паролей, поиска пользователя в базе данных и метод аутентификации.

5.6 Разработка клиентской части

Для создания клиентской части и обеспечения лучшего пользовательского опыта был выбран React.

React - это JavaScript-библиотека для разработки пользовательских интерфейсов, разработанная компанией Facebook. Она позволяет создавать динамические и интерактивные веб-приложения, используя компонентный подход. React использует виртуальный DOM для оптимизации производительности, обновляя только необходимые части страницы при изменении данных. Одной из ключевых особенностей React является возможность создания множества переиспользуемых компонентов, что упрощает разработку и обслуживание сложных интерфейсов. С помощью React разработчики могут строить мощные и интуитивно понятные

пользовательские интерфейсы, обеспечивая при этом хорошую производительность и удобство в разработке.

Было создано множество компонент [17]: компоненты для страницы регистрации, авторизации, страницы профиля, «шапки» и «подвала» сайта и страницы со случайными пользователями. На рисунке 4 представлена структура папки с компонентами.

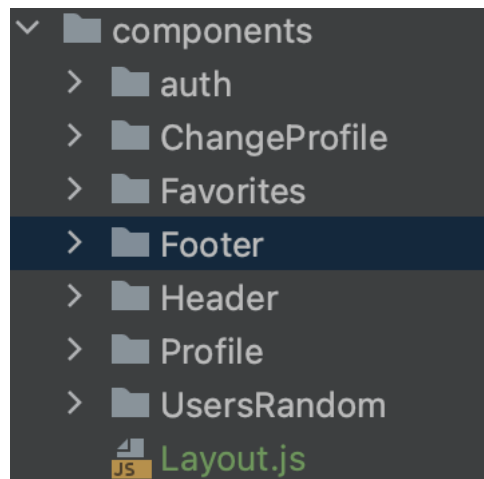


Рисунок 4 – Папка с компонентами

В компоненте App прописаны пути [15] к страницам. Код этой компоненты представлен на Листинге 8.

Листинг 8 – Код компоненты App

```
import {BrowserRouter, Navigate, Route, Routes} from "react-router-dom";
import Login from "../components/auth/Login/Login";
import Registration from
  "../components/auth/registration/Registration";
import {useSelector} from "react-redux";
import Profile from "../components/Profile/Profile";
import RandomUsers from
  "../components/UsersRandom/RandomUsers";
import Favorites from "../components/Favorites/Favorites";
import Layout from "../components/Layout";
import {useState} from "react";
import Logout from "../components/auth/Logout";
import ChangeProfile from
  "../components/ChangeProfile/ChangeProfile";
function App() {
  let exp=useSelector(state => state.auth.exp)
  let token=useSelector(state => state.auth.token)
  if (exp===undefined) {
    exp=0
  }
  let now=Math.floor(Date.now() / 1000)
```

Продолжение Листинга 8

```
let [checkAuth, setCheckAuth]=useState((now<exp) &&
(token!==undefined))
return (
  <>
    <BrowserRouter>
      <Routes>
        <Route path='/profile/change'
element={<ChangeProfile/>}></Route>
        <Route path='/logout' element={checkAuth?<Logout
setCheck={setCheckAuth}/>:<Login
setCheck={setCheckAuth}/>}></Route>

        <Route path='/login'
element={checkAuth?<Navigate to='/'/>:<Login
setCheck={()=>setCheckAuth}/>}></Route>
        <Route path='/registration'
element={checkAuth?<Navigate to='/'/>:<Registration
setCheck={setCheckAuth}/>}></Route>
        <Route path='/'
element={checkAuth?<Layout/>:<Navigate to='/login'/>}>
          <Route index element={<Navigate to='/search-
friends'/>}></Route>
          <Route path='profile' >
            { /*<Route path='/me'/>*/ }
            <Route path=':id' element={<Profile/>}/>
          </Route>

          <Route path='search-friends'
element={<RandomUsers/>}></Route>

          <Route path='favorites'
element={<Favorites></Favorites>}></Route>

        </Route>
      </Routes>
    </BrowserRouter>
  </>
);
}
export default App;
```

Для отправки запросов на сервер была использована библиотека `axios`.

Самая важная компонента это `RandomUsers`, в ней хранится основная логика по запросу и отображению случайных пользователей. В компоненте мы запрашиваем 5 случайных пользователей. И отображаем их по мере

необходимости. После того как эти 5 пользователей заканчиваются мы запрашиваем ещё 5.

Для управления состоянием была подключена библиотека `redux` [15]. В ней хранится токен и время его работы, который берётся из `cookie` файлов. Для хранения информации был создан `reducer`. В нем было создано 2 метода. Для установки нового токена и метод удаления токена. Код этого `reducer`a` представлен на Листинге 9.

Листинг 9 – Код `reducer`, который хранит состояние приложения

```
import {createSlice} from "@reduxjs/toolkit";
import Cookies from 'js-cookie';
import {jwtDecode} from 'jwt-decode';

const slice=createSlice({
  name: 'login',
  initialState:{
    token:Cookies.get('token'),
    exp:Cookies.get('exp')
  },
  reducers:{
    setToken(state, action){
      Cookies.set('token',action.payload.token)
      Cookies.set('exp',
jwtDecode(action.payload.token).exp)
      state.token=action.payload.token
      return state;
    },
    deleteToken(state, action){
      Cookies.remove('token')
      Cookies.remove('exp')
      state.token=null
      state.exp=null

      return state;
    }
  }
})

export const dispatches = slice.actions;
export default slice.reducer;
```

Ниже приведён список страниц, реализованных в приложении:

- 1) Страница регистрации (Рисунок 5) - здесь пользователи могут зарегистрироваться;
- 2) страница входа (Рисунок 6) – на этой странице реализована логика авторизации;
- 3) страница профиля (Рисунок 7) - пользователи могут просмотреть свой и чужой профиль;
- 4) страница избранных пользователей (Рисунок 8) - здесь пользователи могут просмотреть список избранных пользователей;
- 5) страница случайных пользователей (Рисунок 9) – на этой странице предлагаются новые пользователи в случайном порядке;
- 6) страница изменения профиля (Рисунок 10) – здесь можно изменить данные профиля;

Имя:

Дата рождения:

День Месяц Год

Пол:

Город:

Город

Обо мне

cvlastyuk2@mail.ru

Telegram:

VK:

.....

Выберите файл | Файл не выбран

Зарегистрироваться

Рисунок 5 – Страница регистрации

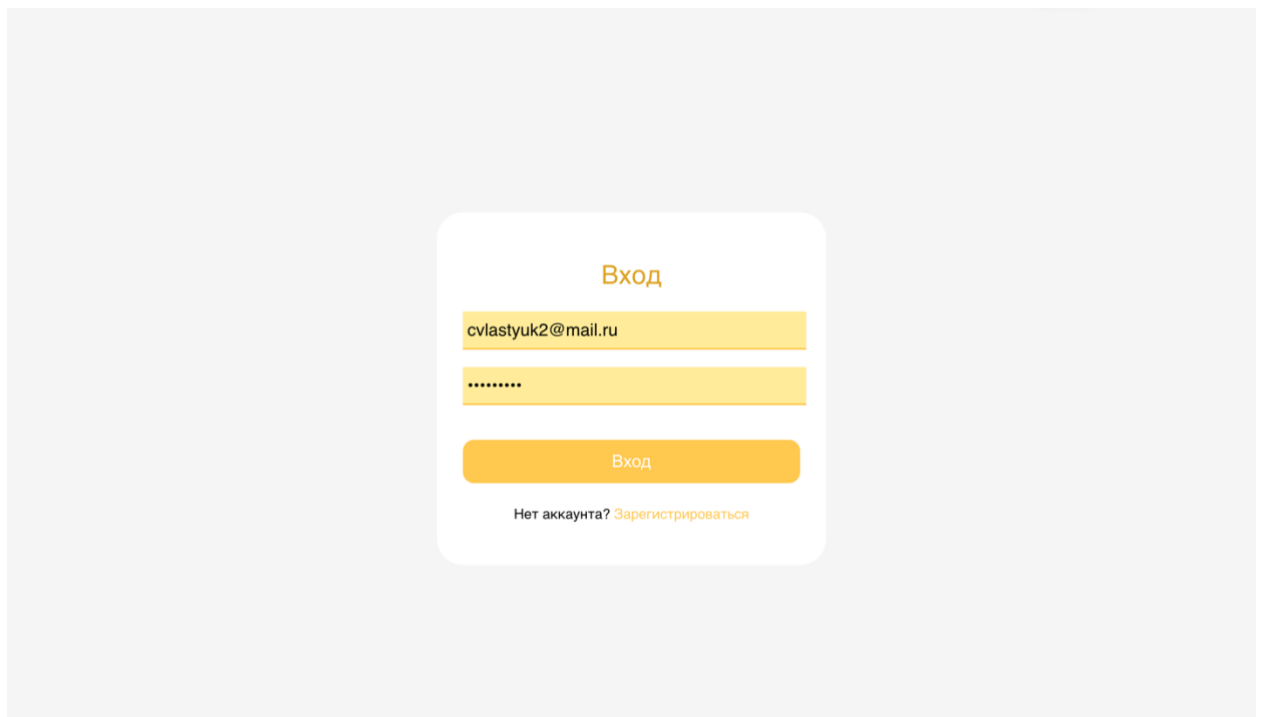


Рисунок 6 – Страница входа

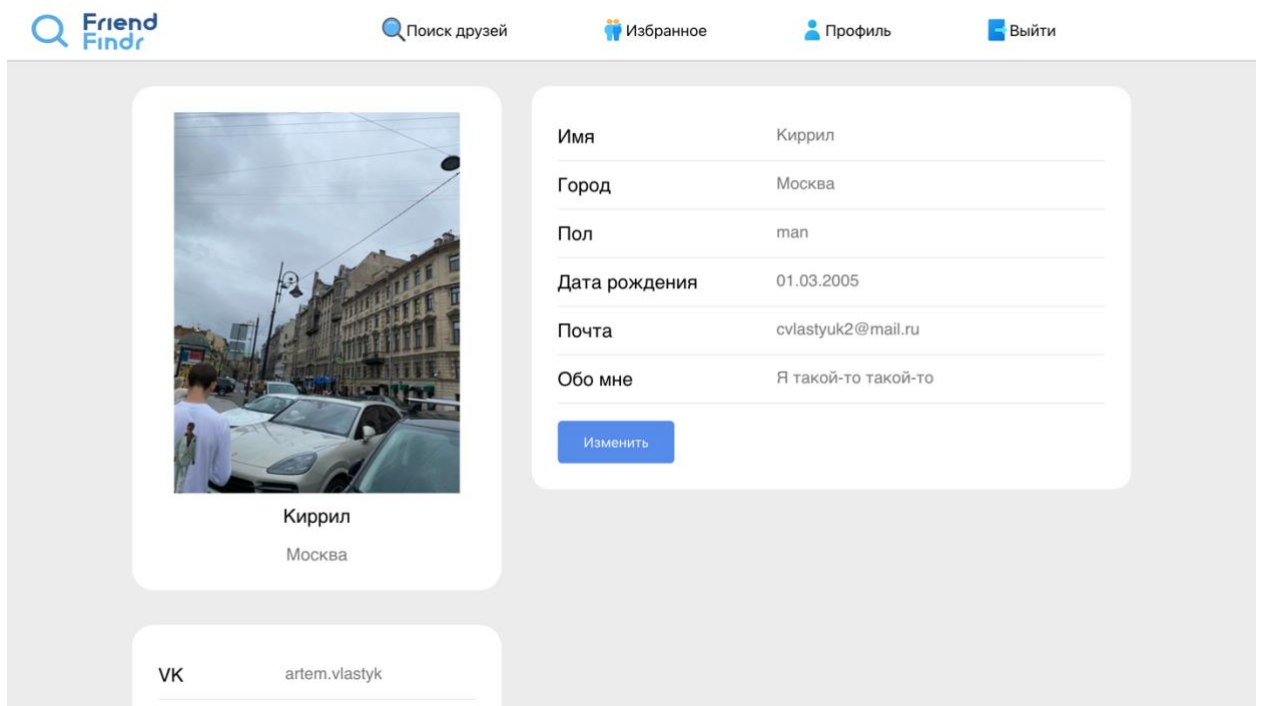


Рисунок 7 – Страница профиля

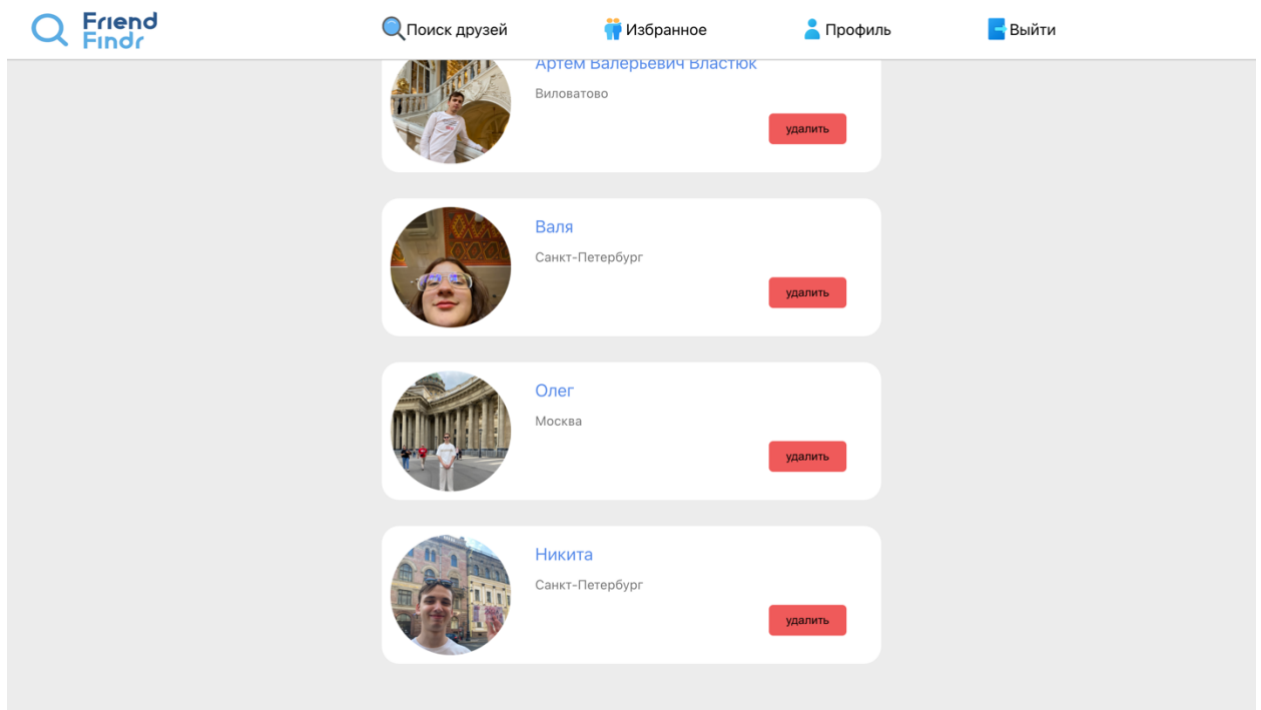


Рисунок 8 – Страница с избранными пользователями

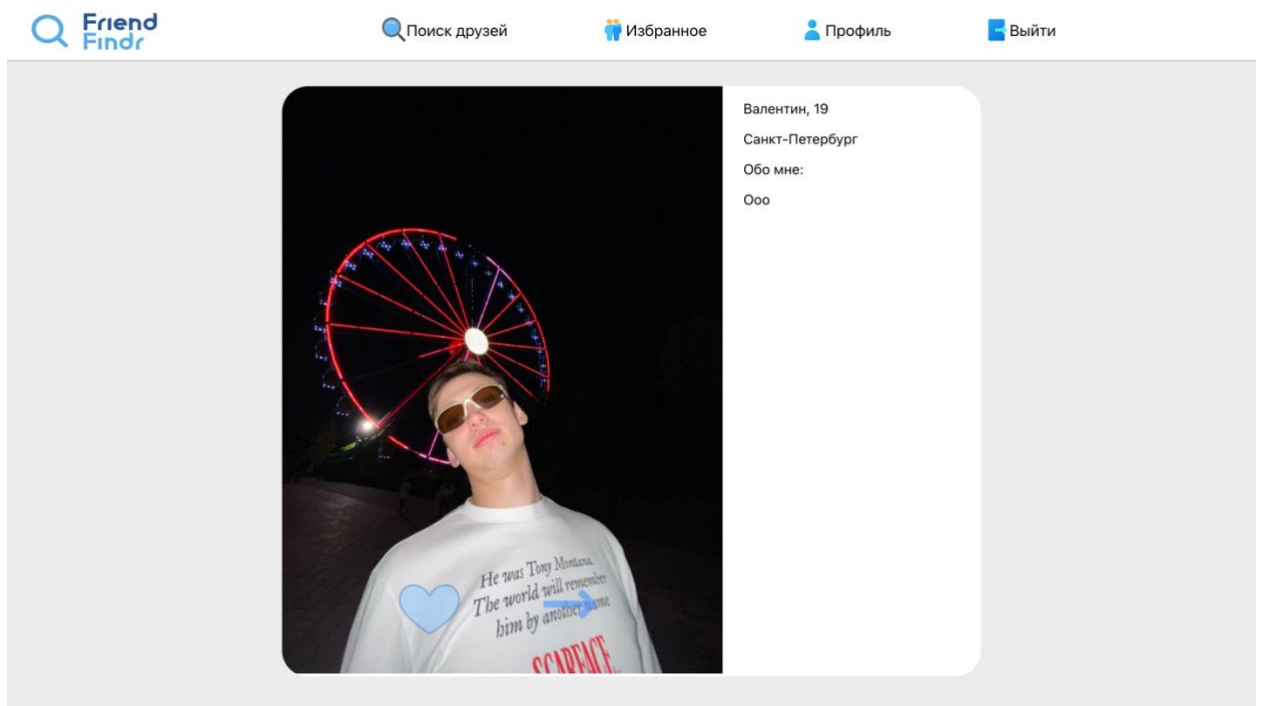


Рисунок 9 – Страница со случайными пользователями

Изменение профиля

Кирилл

Дата рождения:

1

▼

Март

▼

2005

▼

Пол:

M

▼

Город:

Город

▼

Я такой-то такой-то

cvlastyuk2@mail.ru

artem.vlastyuk

artem.vlastyk

Пароль: (укажите, если хотите изменить)

Выберите файл | Файл не выбран

Рисунок 10 – Страница для изменения профиля

6 РАЗВЕРТЫВАНИЕ ПРИЛОЖЕНИЯ

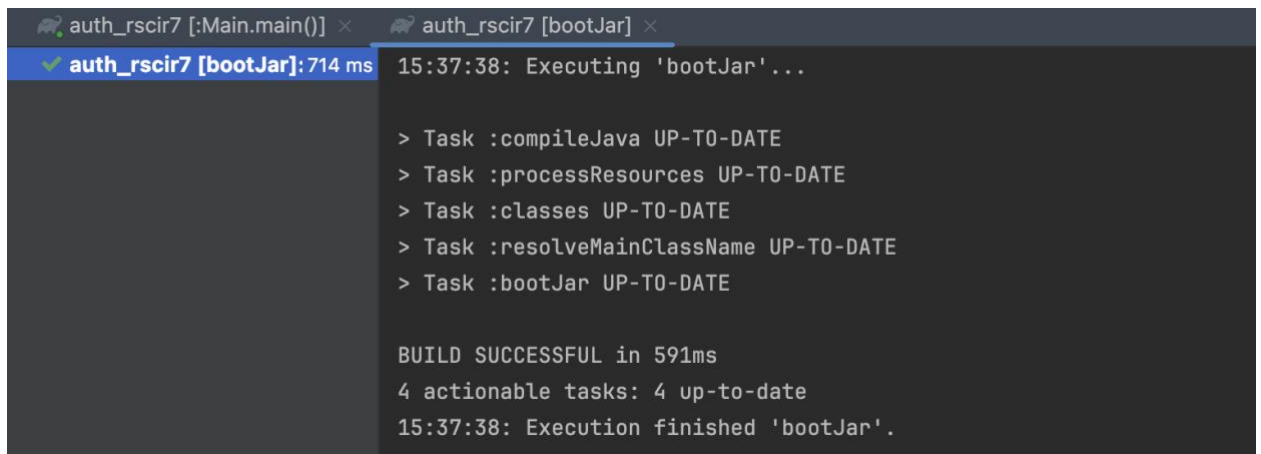
Для развёртывания был выбран vds сервер. VDS (Virtual Dedicated Server) представляет собой виртуальный выделенный сервер, который предоставляет пользователю полный доступ к виртуальной машине с гарантированными ресурсами. VDS обеспечивает большую степень контроля и гибкости, чем обычный виртуальный хостинг, позволяя пользователям настраивать сервер под свои потребности.

Для начала нужно поменять конфигурацию приложения. Конфигурация нашего сервера представлена на Листинге 10.

Листинг 10 – Файл application.properties

```
spring.datasource.url=jdbc:postgresql://localhost:5432/kr
spring.datasource.username=postgres
spring.datasource.password=2003Artem
spring.jpa.database-
platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
spring.mvc.hiddenmethod.filter.enabled=true
spring.servlet.multipart.max-file-size=50MB
spring.servlet.multipart.max-request-size=50MB
#spring.jpa.show-sql=true
#upload-
directory=/Users/artemvolk/Desktop/rschirfull/rschir7/auth_rsc
ir7/src/main/resources/static/
#spring.web.resources.static-
locations=file:/Users/artemvolk/Desktop/rschirfull/rschir7/aut
h_rscir7/src/main/resources/static/
upload-directory=/root/app/
spring.web.resources.static-locations=file:/root/app
spring.servlet.multipart.enabled=true
spring.cors.allowed-origins=http://localhost:3000
spring.cors.allowed-methods=GET, POST, PUT, DELETE, OPTIONS
spring.mvc.converters.preferred-json-mapper=gson
application.security.jwt.secret-
key=404E635266556A586E3272357538782F413F4428472B4B625064536756
6B5970
server.port=8081
host-client=http://localhost:3000
```

После настройки конфигурации соберём наш java проект в jar-файл. Сборка проекта представлена на рисунке 11.



```
auth_rscir7 [:Main.main()] x auth_rscir7 [bootJar] x
✓ auth_rscir7 [bootJar]: 714 ms 15:37:38: Executing 'bootJar'...

> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-DATE
> Task :classes UP-TO-DATE
> Task :resolveMainClassName UP-TO-DATE
> Task :bootJar UP-TO-DATE

BUILD SUCCESSFUL in 591ms
4 actionable tasks: 4 up-to-date
15:37:38: Execution finished 'bootJar'.
```

Рисунок 11 – Сборка проекта

После этого приступим к сборке клиентской части. В проекты был создан файл который хранит url нашего сервера. Больше ничего нет необходимости менять.

Теперь подключимся по ssh нашему vds серверу. Создадим папку для клиентской и серверной части (рисунок 12).

```
root@cvlastyuk1:~# mkdir client
root@cvlastyuk1:~# mkdir server
```


Рисунок 12 – Создание папок

Далее стоит установить базу данных. После настройки базы данных передадим все необходимые файлы на наш сервер (рисунок 13 – 14).



```
artemvolk@MacBook-Air-Artem auth_rscir7 % scp /Users/artemvolk/Desktop/rschirfull/rschir7/auth_rscir7/build/libs/auth_rscir7-0.0.1-SNAPSHOT.jar root@188.120.239.245:/root/s
erver
root@188.120.239.245's password:
auth_rscir7-0.0.1-SNAPSHOT.jar
artemvolk@MacBook-Air-Artem auth_rscir7 % 100% 79MB 10.0MB/s 00:07
```

Рисунок 13 – Передача jar-файла на сервер



```
artemvolk@MacBook-Air-Artem kr % scp -r /Users/artemvolk/Desktop/rksp/kr/ root@188.120.239.245:/root/client
root@188.120.239.245's password:
```

Рисунок 14 – Передача react проекта на сервер

Также стоит создать базу данных (рисунок 15).

```
postgres=# CREATE DATABASE kr
;
CREATE DATABASE
```

Рисунок 15 – Создание базы данных

После настройки запустим наш сервер (рисунок 16).

```
root@cvlastyuk1:~/server# java -jar auth_rscir7-0.0.1-SNAPSHOT.jar --server.port=8082

  ____ _
 / ___ \| | | |
/ /___ \| |_| |
 \___ \|____|_|_|_|
      | |
      | |
      |_|

:: Spring Boot :: (v3.0.6)

2024-05-22T17:54:44.224+03:00 INFO 1315125 --- [ main] org.example.Main : Starting Main v0.0.1-SNAPSHOT using Java 17.0.10 with PID 13151
25 (/root/server/auth_rscir7-0.0.1-SNAPSHOT.jar started by root in /root/server)
2024-05-22T17:54:44.245+03:00 INFO 1315125 --- [ main] org.example.Main : No active profile set, falling back to 1 default profile: "defa
ult"
2024-05-22T17:54:47.792+03:00 INFO 1315125 --- [ main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-05-22T17:54:48.541+03:00 INFO 1315125 --- [ main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 703 ms. Found 2 JPA
repository interfaces.
2024-05-22T17:54:51.544+03:00 INFO 1315125 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8082 (http)
```

Рисунок 16 – Запуск сервера

Теперь запустим клиентскую часть (рисунок 18). Но перед этим скачаем все зависимости (рисунок 17).

```
root@cvlastyuk1:~/client/kr/kr# npm install

npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: '@csstools/postcss-trigonometric-functions@1.0.2',
npm WARN EBADENGINE   required: { node: '^14 || >=16' },
npm WARN EBADENGINE   current: { node: 'v12.22.9', npm: '8.5.1' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: '@csstools/selector-specificity@2.2.0',
npm WARN EBADENGINE   required: { node: '^14 || ^16 || >=18' },
npm WARN EBADENGINE   current: { node: 'v12.22.9', npm: '8.5.1' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: '@jest/expect-utils@29.7.0',
npm WARN EBADENGINE   required: { node: '^14.15.0 || ^16.10.0 || >=18.0.0' },
npm WARN EBADENGINE   current: { node: 'v12.22.9', npm: '8.5.1' }
}
```

Рисунок 17 – Установка зависимостей

```
root@cvlastyuk1:~/client/kr/kr# npm start

> kr@0.1.0 start
> react-scripts start
```

Рисунок 18 – Запуск проекта

7 ПРОВЕДЕНИЕ ТЕСТИРОВАНИЯ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ

Для начала проведём тестирование клиентской части. Доступ к страницам разрешён только после авторизации. Без авторизации мы попадаем на страницу входа (рисунок 19).

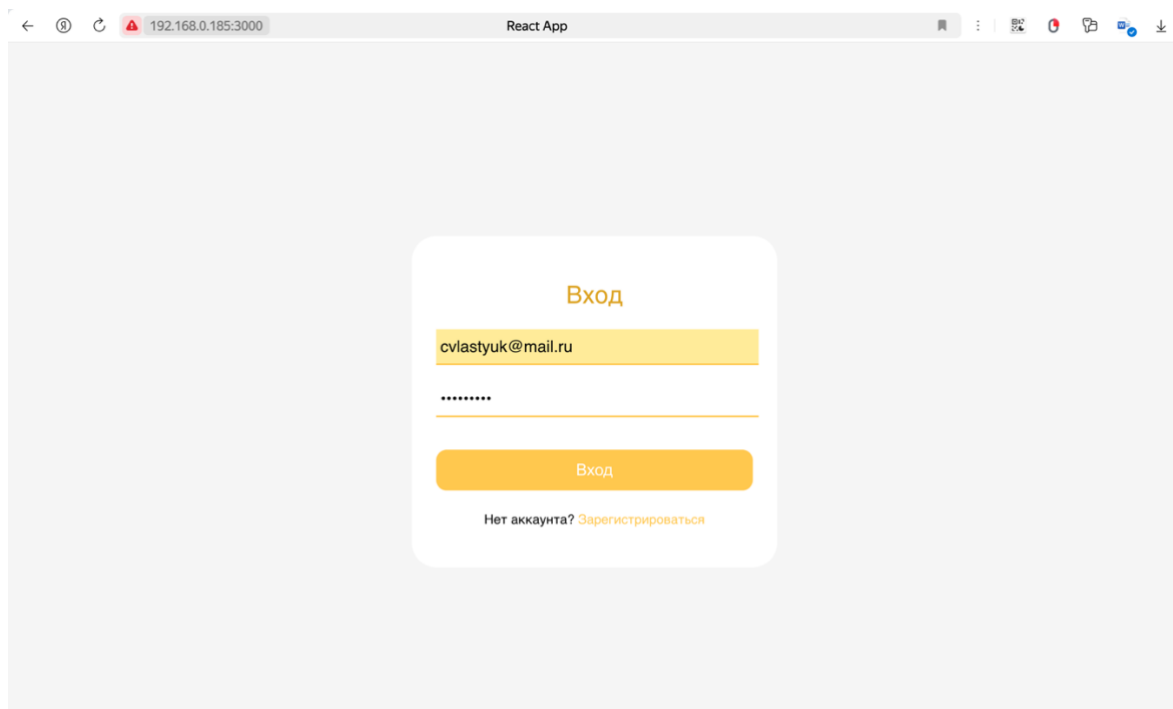


Рисунок 19 – Запуск сайта и переадресация на страницу входа
Попробуем войти в несуществующего пользователя (рисунок 20).

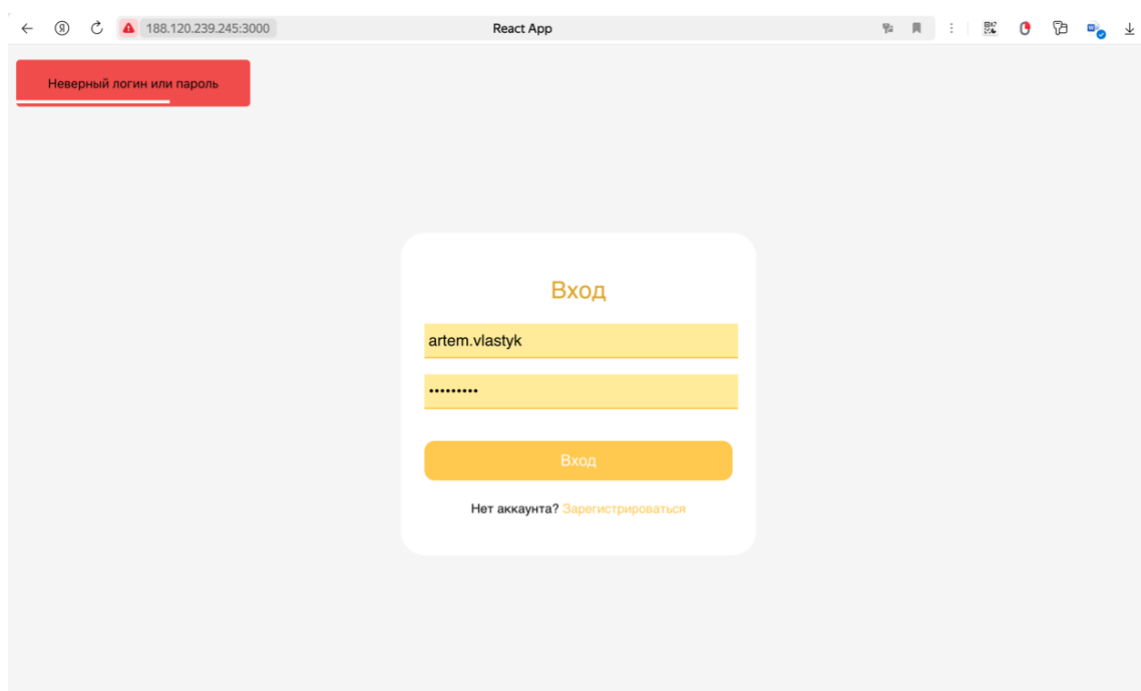


Рисунок 20 – Вход в несуществующего пользователя

Как мы видим, появляется всплывающее окно с ошибкой. Попробуем пройти регистрацию (рисунок 21).

Рисунок 21 – Попытка регистрации

Появилась ошибка. Укажем другую почту. После успешной регистрации мы попадаем на главную страницу веб-приложения (рисунок 22). Появляется случайный пользователь, мы можем добавить его в избранное, нажав на сердечко. Или же пропустить, тогда этот пользователь не появится снова.

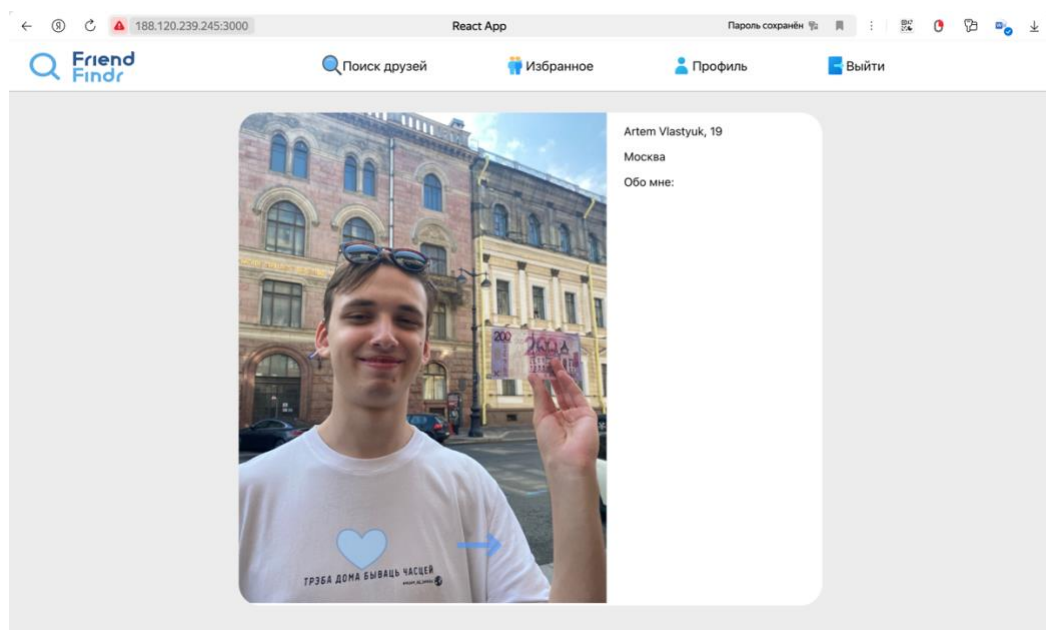


Рисунок 22 – Страница со случайными пользователями

Добавим предложенного пользователя в избранное и проверим советующую страницу (рисунок 23).

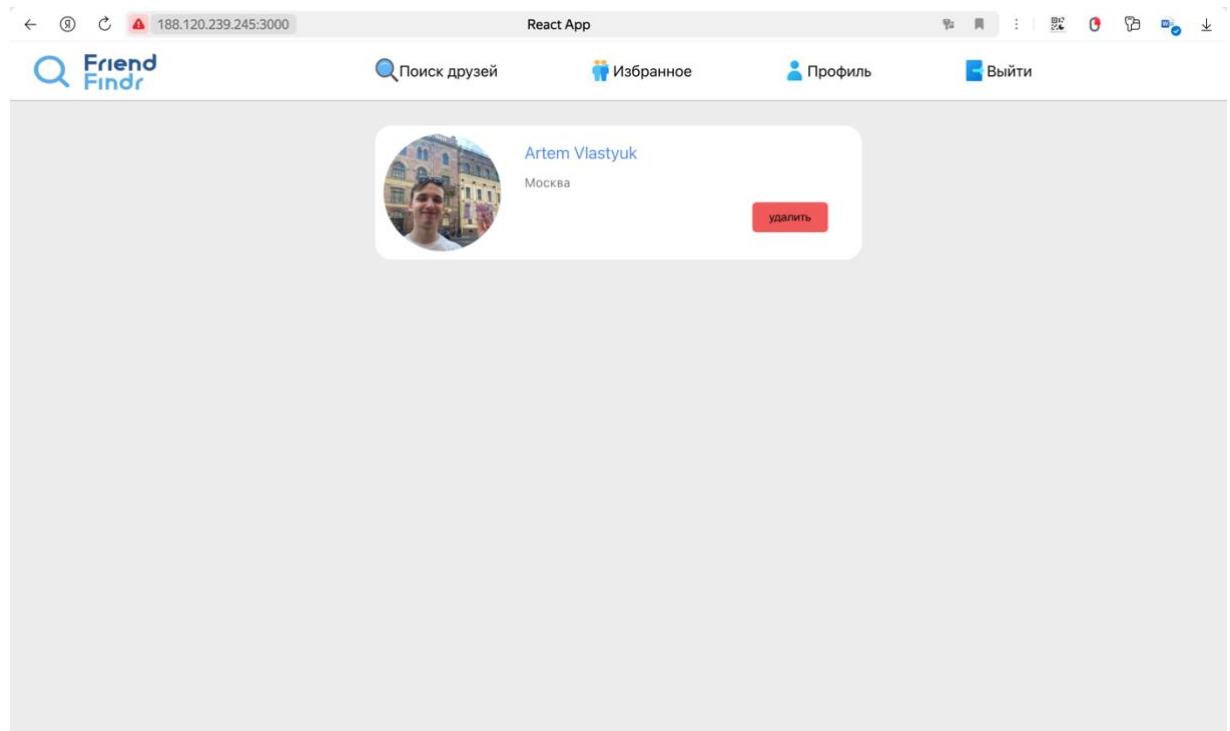


Рисунок 23 – Страница с избранными пользователями

Также проверим страницу профиля (рисунок 24).

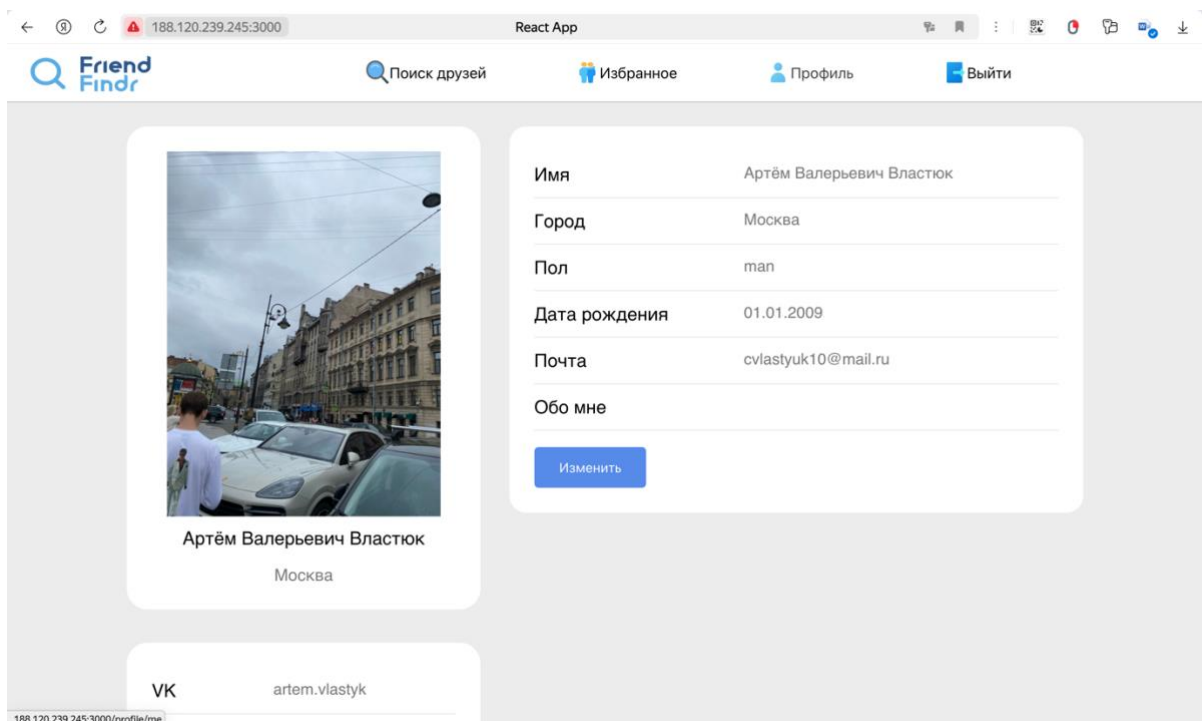


Рисунок 24 – Страница профиля

Теперь проверим как работает изменение профиля. Попробуем изменить имя и добавить описание. Страница после изменения показана на рисунке 25.

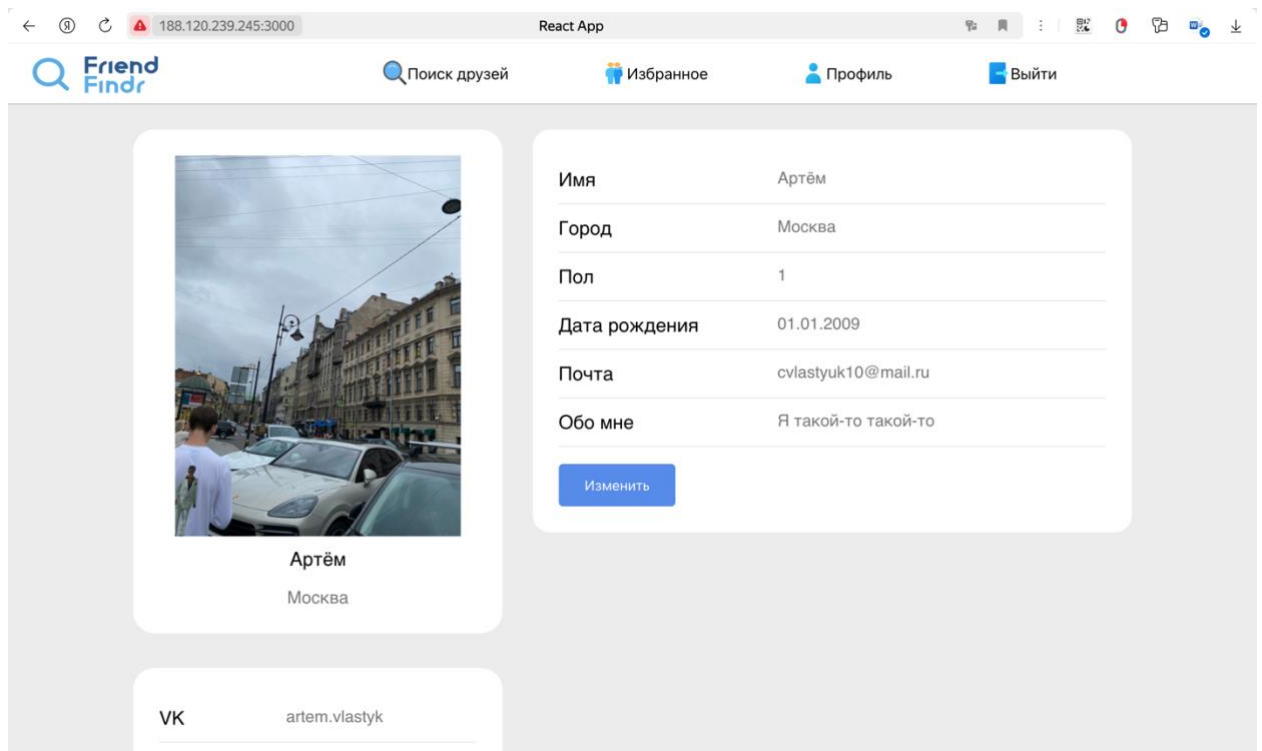


Рисунок 25 – Страница профиля

Также необходимо провести тестирование серверной части. Отправим те же запросы, что и на клиентскую часть, проверив как это работает изнутри. Для отправки запросов был использован Postman.

Для начала попробуем войти в несуществующего пользователя (рисунок 26).

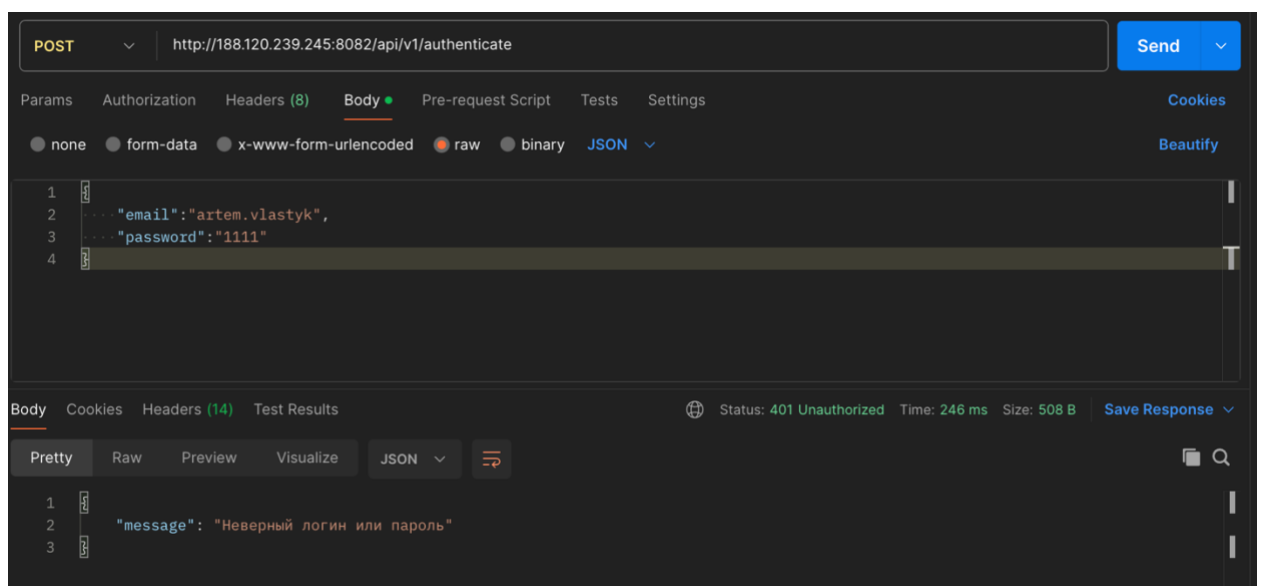


Рисунок 26 – Попытка входа в несуществующего пользователя

Теперь попробуем пройти регистрации. Запрос на регистрацию показан на рисунке 27.

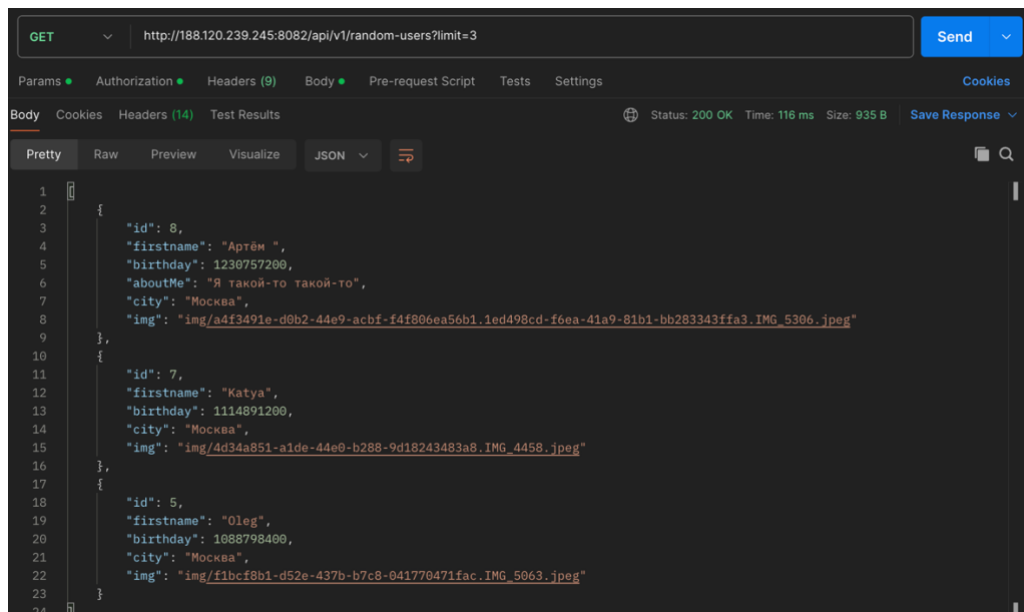


Рисунок 29 – Получение случайных пользователей

Мы видим, что нам выдало каждый раз случайных пользователей. Значит, что наш метод работает корректно.

Далее добавим одно из этих пользователей в избранное (рисунок 30).

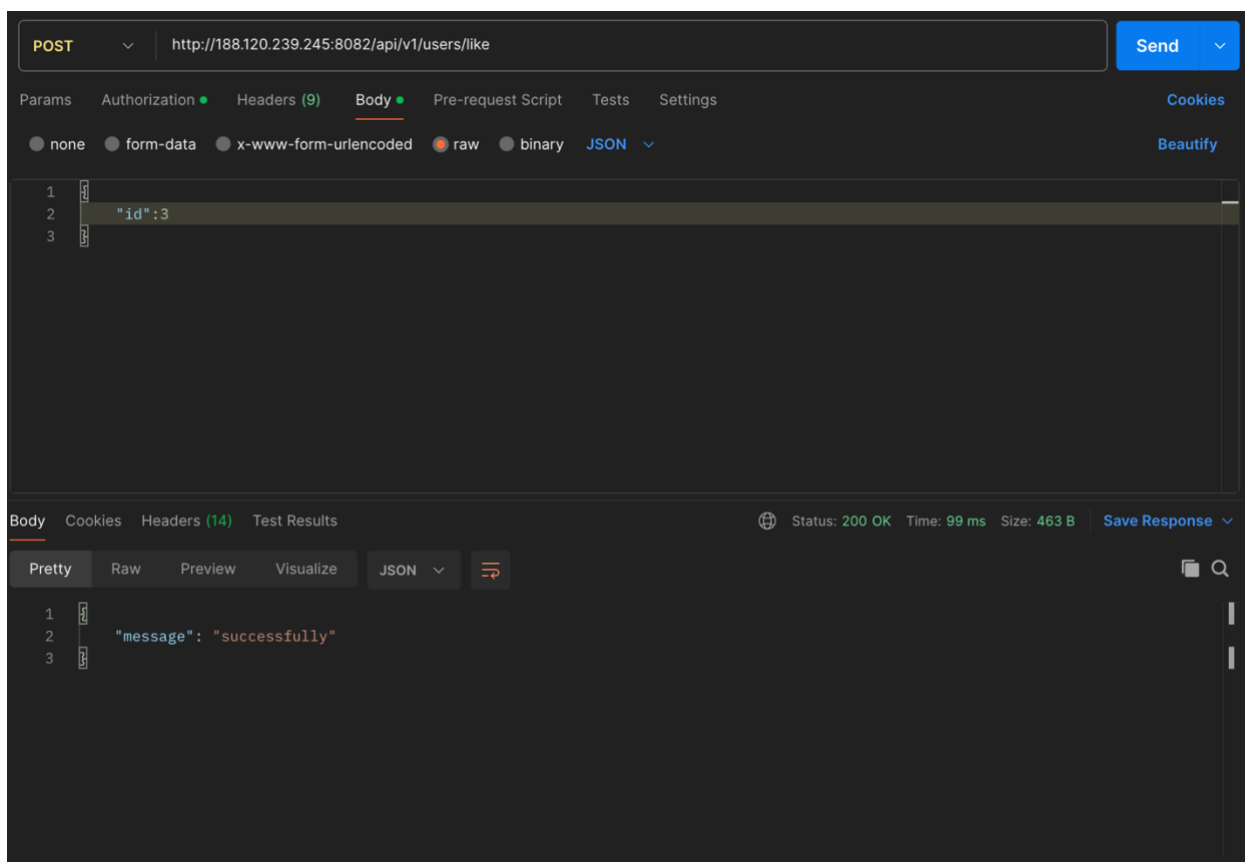


Рисунок 30 – Добавление в избранное

Получаем ответ, что операция успешно выполнена. Проверим список пользователей в избранном (рисунок 31).

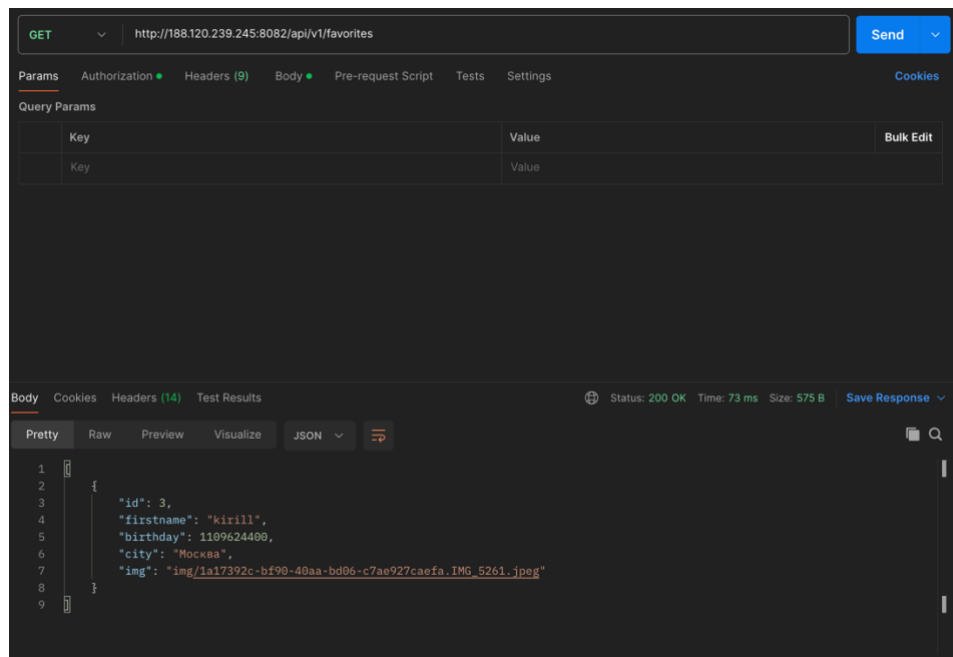


Рисунок 31 – Избранные пользователи

Теперь попробуем удалить данного пользователя из избранного (рисунок 32).

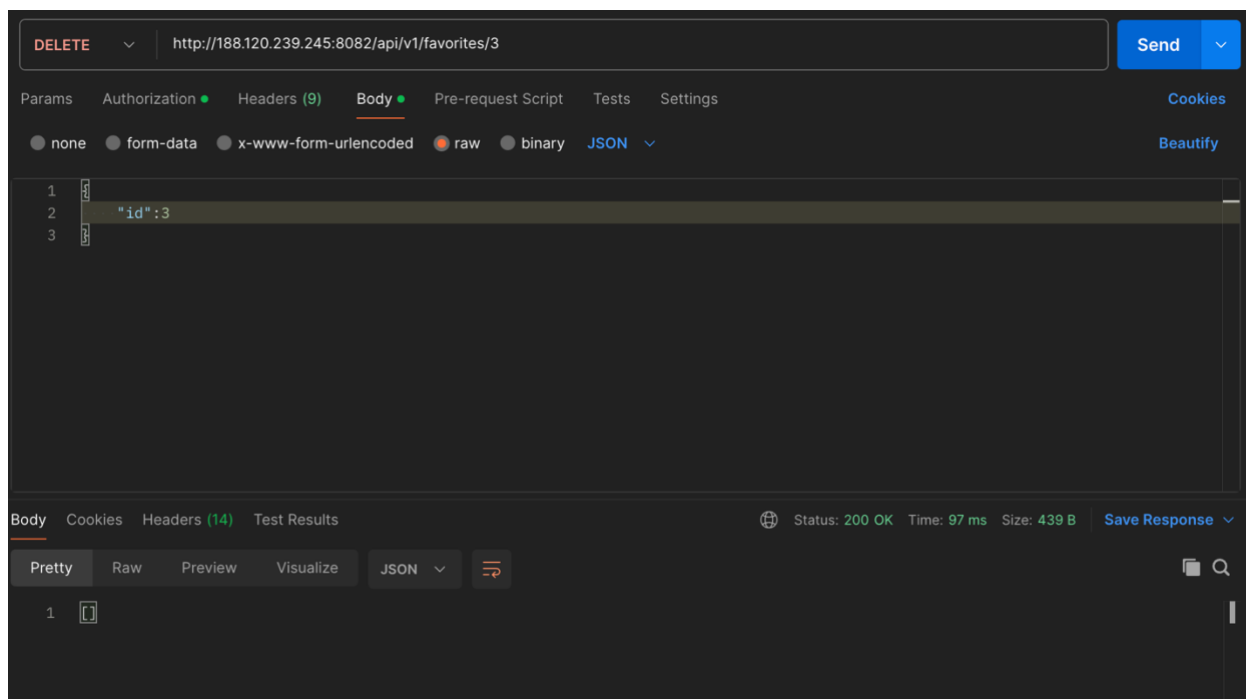


Рисунок 32 – Удаление пользователя из избранного

Далее проверим наш профиль (рисунок 33).

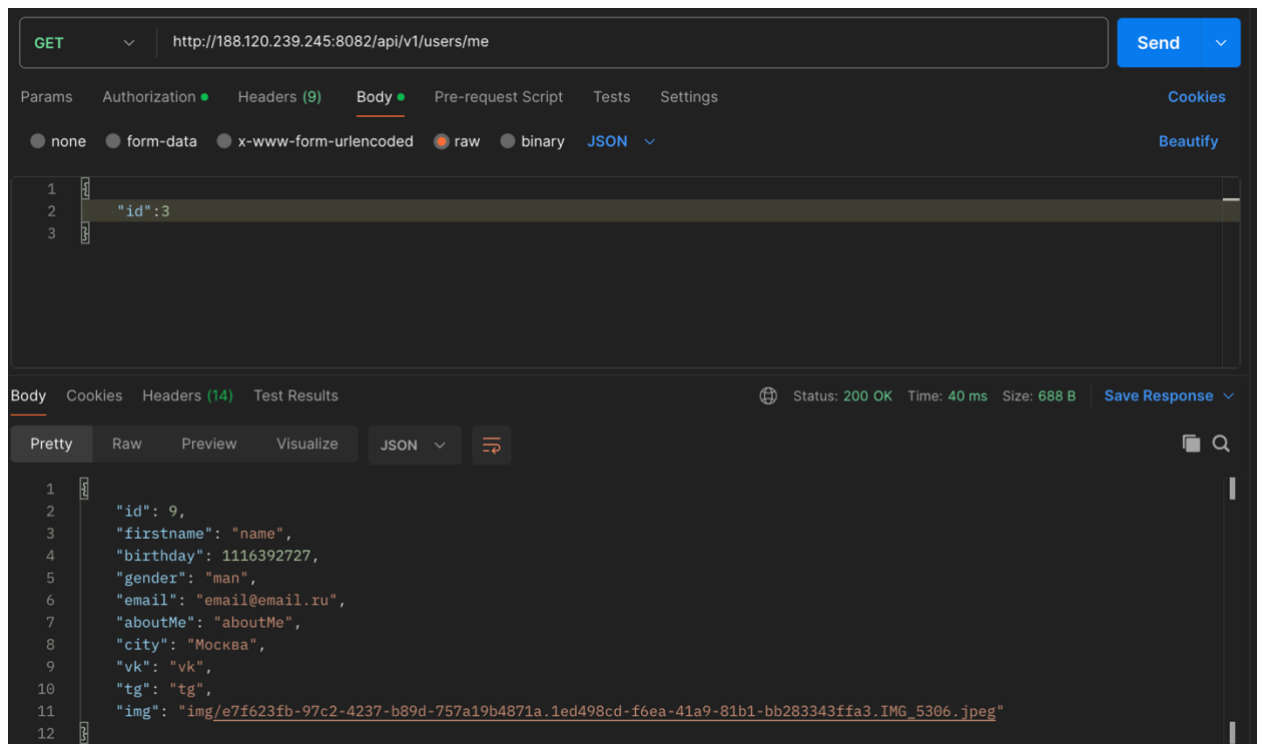


Рисунок 33 – Проверка профиля

После проведённого тестирования мы видим, что у нас получилось развернуть наше приложение без каких-либо ошибок.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было разработано Spring MVC приложение для поиска друзей и знакомых, которое позволяет пользователям просматривать профиль, искать случайных пользователей и добавлять их в избранное.

В ходе работы были изучены принципы работы Spring Framework и Spring MVC, а также использованы различные технологии и инструменты, такие как React, Hibernate и PostgreSQL.

Была реализована логика приложения на стороне сервера и клиента.

В результате работы было получено готовое функциональное приложение, которое может быть использовано в реальных условиях. Оно может быть доработано и расширено в соответствии с потребностями бизнеса.

В целом, выполнение данной курсовой работы позволило получить практические навыки работы с Spring MVC и разработки веб-приложений, что является важным этапом в процессе обучения программированию.

Исходный код серверной части приложения доступен на Github [14] по ссылке – <https://github.com/artemmmvl/serverKR>. Клиентской части приложения по ссылке – <https://github.com/artemmmvl/clientKR>.

Ссылка на приложение, развернутое на хостинге firstVDS [18] – <http://188.120.239.245:3000/>.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Security Best Practices [Электронный ресурс]. – URL: <https://nodejs.org/en/learn/getting-started/security-best-practices> (дата обращения 30.02.2024).
2. Spring Overview [Электронный ресурс]. – URL: <https://docs.spring.io/spring-framework/reference/overview.html> (дата обращения 01.03.2024).
3. Security Configuration [Электронный ресурс]. – URL: <https://docs.spring.io/spring-security/reference/migration-7/configuration.html> (дата обращения 01.03.2024).
4. Security Started [Электронный ресурс]. – URL: <https://docs.spring.io/spring-security/reference/servlet/getting-started.html> (дата обращения 01.03.2024).
5. Java Аннотации [Электронный ресурс]. – URL: <https://javarush.com/groups/posts/1896-java-annotacii-cto-eh-to-i-kak-eh-tim-poljhzovatjhsja> (дата обращения 01.03.2024).
6. Spring и Hibernate [Электронный ресурс]. – URL: <https://habr.com/ru/companies/productstar/articles/774508/> (дата обращения 01.03.2024).
7. Spring Boot – Microsoft [Электронный ресурс] – URL: <https://azure.microsoft.com/ru-ru/resources/cloud-computing-dictionary/what-is-java-spring-boot/> (дата обращения 12.09.2023)
8. PostgreSQL: About [Электронный ресурс] – URL: <https://www.postgresql.org/about/> (дата обращения 15.10.2023).
9. Справочник CSS свойств [Электронный ресурс]. – URL: <http://htmlbook.ru/samcss> (дата обращения 03.11.2023).
10. Spring MVC – Habr [Электронный ресурс] – URL: <https://habr.com/ru/articles/336816/> (дата обращения 30.09.2023)

11. Spring @Service Annotation with Example [Электронный ресурс] – URL: <https://www.geeksforgeeks.org/spring-service-annotation-with-example/> (дата обращения 27.02.2024)
12. Spring Data JPA – devcolibri [Электронный ресурс] – URL: <https://devcolibri.com/spring-data-jpa-работа-с-бд-часть-1/> (дата обращения 04.10.2023)
13. Rohan Vats – Spring Boot Annotations Everyone Should Know [Электронный ресурс] – URL: <https://www.upgrad.com/blog/spring-boot-annotations/> (дата обращения 21.02.2024)
14. Rohan Vats – Spring Boot Annotations Everyone Should Know [Электронный ресурс] – URL: <https://habr.com/ru/articles/498860/> (дата обращения 21.02.2024)
15. Redux [Электронный ресурс] – URL: <https://docs.github.com/ru/repositories/creating-and-managing-repositories> (дата обращения 27.11.2023).
16. React Router [Электронный ресурс] – URL: <https://ru.hexlet.io/blog/posts/react-router-v6> (дата обращения 27.11.2023).
17. React Components [Электронный ресурс] – URL: <https://ru.react.js.org/docs/react-component.html> (дата обращения 27.11.2023).
18. FirstVDS [Электронный ресурс] – URL: <https://firstvds.ru/technology> (дата обращения 27.11.2023).