# Курсовой проект

## По дисциплине «Технологии программирования»

На тему «Разработка веб-приложения для поиска стажировок для студентов»

Выполнил:

Студент гр. АУБП-21-1

Бахилин В.И.

Проверила:

ст. преп. каф. информатики Волкова Е.А.

Екатеринбург, 2024г.

**Оглавление**

[**Курсовой проект 1**](#_Toc185899835)

[**По дисциплине «Технологии программирования» 1**](#_Toc185899836)

[**1. Постановка задачи 3**](#_Toc185899837)

[**1.1 Характеристика объекта и проблемной области 3**](#_Toc185899838)

[**1.2 Постановка целей и задач автоматизации/информатизации 3**](#_Toc185899839)

[**1.3. Краткое техническое задание 4**](#_Toc185899840)

[**2. Проектирование системы 8**](#_Toc185899841)

[**2.1 Моделирование системы 8**](#_Toc185899842)

[**2.2 Проектирование БД 10**](#_Toc185899843)

[**2.3 Проектирование интерфейса 11**](#_Toc185899844)

[**3. Разработка системы 15**](#_Toc185899845)

[**3.1 Выбор средств реализации 15**](#_Toc185899846)

[**3.2 Структура проекта 15**](#_Toc185899847)

[**3.3 Реализация 20**](#_Toc185899848)

# **Постановка задачи**

## **1.1 Характеристика объекта и проблемной области**

Объектом автоматизации/информатизации является система, предназначенная для автоматизации и упрощения процесса поиска стажировок для студентов и выпускников, а также для облегчения процесса найма стажеров для компаний.

Проблема заключается в том, что поиск стажировок и найм стажеров являются сложными и неэффективными процессами, как для соискателей, так и для компаний. Объявления о стажировках разбросаны по разным сайтам, платформам, социальным сетям, что усложняет поиск и нет единой платформы, где можно было бы найти все подходящие стажировки.

## 1.2 Постановка целей и задач автоматизации/информатизации

**Основная Цель Автоматизации/Информатизации:**

Создать эффективную, удобную и централизованную онлайн-платформу, которая обеспечит эффективный поиск и управление стажировками, а также упростит процесс создания и управления резюме для студентов, тем самым способствуя карьерному росту студентов и удовлетворению потребностей работодателей.

**Задачи, которые необходимо решить для ее достижения**

* Создание базы данных системы
* Разработка функциональности поиска и фильтрации стажировок:
* Разработка структурированного интерфейса для создания резюме:
* Реализация функциональности редактирования и удаления резюме:
* Хранение резюме в базе данных
* Реализация функциональности добавления стажировок в избранное:
* Реализация механизмов отклика на стажировки:
* Реализация системы отзывов о стажировках:
* Реализация механизмов авторизации и аутентификации:
* Выбор и настройка необходимых технологий и инструментов:
* Разработка архитектуры системы

## **1.3. Краткое техническое задание**

**Систему можно логически разделить на следующие подсистемы:**

* Подсистема управления пользователями: Управление учетными записями пользователей (учащихся и компаний), включая регистрацию, аутентификацию, авторизацию, управление профилями и контроль доступа.
* Подсистема поиска и фильтрации соискателей для компаний: обеспечивает поиск соискателей для компаний по различным критериям
* Подсистема управления объявлениями о стажировках: позволяет компаниям создавать, редактировать, публиковать и управлять списками стажировок. Включает функции поиска, фильтрации и сортировки объявлений.
* Подсистема связи: обеспечивает коммуникацию между учащимися и компаниями, включая уведомления, обмен сообщениями и управление приложениями.
* Подсистема хранения и поиска данных: управляет данными системы, включая профили пользователей, списки стажировок и журналы общения. Обычно использует базу данных (например, PostgreSQL, MySQL или SQLite).

**Требования к функциям (задачам), выполняемым АС**

Подсистема управления пользователями:

* Регистрация пользователя: Автоматизированное создание учетных записей пользователей (студентов и компаний).
* Аутентификация: Обеспечьте безопасную аутентификацию пользователей с помощью надежных алгоритмов хеширования паролей и потенциально многофакторной аутентификации.
* Авторизация: Ролевое управление доступом (RBAC) для ограничения доступа к определенным функциям системы на основе ролей пользователей (студент, компания, администратор).
* Управление профилем: Автоматизированное управление профилями пользователей, позволяющее пользователям обновлять личную информацию и предпочтения. Возможна интеграция с внешними сервисами (например, парсинг резюме).
* Сброс пароля: Функция автоматического сброса пароля, обеспечивающая принятие мер безопасности для предотвращения несанкционированного доступа.

Подсистема управления информацией о соискателях для компании:

* Добавление резюме студентом
* Возможность редактировать резюме
* Поиск студентов компаниями

Подсистема управления информацией о стажировках:

Создание объявления: создание и управление списками стажировок с проверкой обязательных полей (описание должности, требования, сроки, реквизиты компании).

* Обновление стажировок: Возможность редактировать и обновлять существующие списки стажировок.
* Публикация стажировок
* Поиск и фильтрация объявлений:  поиск объявлений о стажировках по различным критериям (ключевые слова, местоположение, даты и т.д.).

Подсистема связи:

* Обмен сообщениями в приложении: Автоматизированная система обмена сообщениями для связи между студентами и компаниями. Для этого необходима безопасная передача и хранение сообщений.
* Уведомления по электронной почте: Автоматические уведомления по электронной почте для студентов о новых подходящих стажировках и для компаний о новых заявках.
* SMS-уведомления (опционально): Автоматические SMS-уведомления о срочных обновлениях (в зависимости от требований).
* Обновления статуса программы: Автоматическое обновление статуса заявки (отправлена, получена, собеседования, отклонена, принята).

Подсистема хранения и управления данными:

* Хранение данных: Автоматизированное хранение всех данных системы (профили пользователей, объявления о стажировках, заявки, сообщения) в базе данных. Будет выбран подходящий дизайн базы данных (например, реляционный или NoSQL).
* Резервное копирование и восстановление данных: Автоматизированное регулярное резервное копирование и эффективные механизмы восстановления данных для защиты данных от потери или повреждения.
* Безопасность данных: Шифрование конфиденциальных данных (паролей, личной информации) при хранении и передаче.

**Требования к надёжности**

Состав и количественные значения показателей надежности для системы в целом или ее подсистем:

* Доступность системы: Система должна быть доступна для пользователей не менее 80% времени.
* Среднее время между отказами: для критических подсистем, таких как база данных, среднее время между отказами должен быть не менее 1000 часов. Для менее критических компонентов значения могут быть меньше, но должны быть определены.
* Среднее время восстановления: Время восстановления системы после отказа не должно превышать 30 минут для критических сбоев и 1 часа для менее критических. Должны быть разработаны планы восстановления после сбоев.
* Вероятность безотказной работы: Заданный период безотказной работы (например, 24 часа) должен иметь вероятность не менее 95%.
* Целостность данных: Система должна гарантировать целостность данных о студентах, работодателях и стажировках. Регулярное резервное копирование и механизмы восстановления данных должны быть предусмотрены.

2) Перечень аварийных ситуаций, по которым должны быть регламентированы требования к надежности, и значения соответствующих показателей:

* Отказ базы данных: Должны быть предусмотрены механизмы резервного копирования, автоматического восстановления и аварийного переключения на резервный сервер. MTTR не должен превышать 30 минут.
* Сбои в работе веб-приложения: должен быть предусмотрен мониторинг производительности и автоматическое масштабирование ресурсов для обработки пиковых нагрузок. MTTR не должен превышать 1 часа.
* DDOS-атаки: Система должна быть защищена от DDoS-атак с помощью соответствующих мер безопасности (например, CDN, защита от ботов).
* Потеря данных: Регулярное резервное копирование данных, с проверкой работоспособности восстановления. среднее время для восстановления данных не должен превышать 6 часов.

**Требования к безопасности**

* Несанкционированный доступ: Система должна быть защищена от несанкционированного доступа с помощью многофакторной аутентификации и других мер безопасности.

**Требования к эргономике и технической эстетике**

Доступ к функционалу системы производится через интерфейс системы. Интерфейс системы должен быть таким, чтобы человек с базовыми технологическими знаниями мог ориентироваться в нём и выполнить основную функцию системы, а именно поиск стажировок, за 20 минут.

Все основные разделы системы, такие как чат, резюме, доступ к профилю, должны находится на главном меню и должны быть видны пользователю.

**Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы**

Система должна быть доступна для пользователей 90% времени за исключением моментов технического обслуживания.

Должно проводится обслуживанию программно-аппаратных частей системы раз в неделю. Под этим подразумевается обновление веб-сервера, базы данных, резервное копирование базы данных и других важных данных, проверка состояния серверов, сети, устройств хранения данных.

**Требования к защите информации от несанкционированного доступа**

* Шифрование данных
* Резервное копирование: Регулярное резервное копирование данных, с проверкой целостности и восстановления.
* Контроль версий: Система должна иметь механизмы контроля версий для восстановления предыдущих состояний данных.

**Требования по сохранности информации при авариях;**

**Мониторинг системы:**

* Системы мониторинга: Система должна обеспечивать мониторинг работоспособности оборудования и программного обеспечения, с отправкой уведомлений об аварийных ситуациях.
* Логирование: Подробное логирование всех событий, включая сбои и ошибки, для анализа причин аварий и повышения надежности системы.

**Резервное копирование:**

Должно производится резервное копирование таких важных данных системы, такие как базы данных, где хранится информация о данных системы, файлы, содержащие настройки системы, базы данных, веб-сервера, журнал событий

Резервное копирование производится в хранилище на том же сервере или в той же сети, где работает система

# **2. Проектирование системы**

## **2.1 Моделирование системы**

UML-диаграмма Use Case:

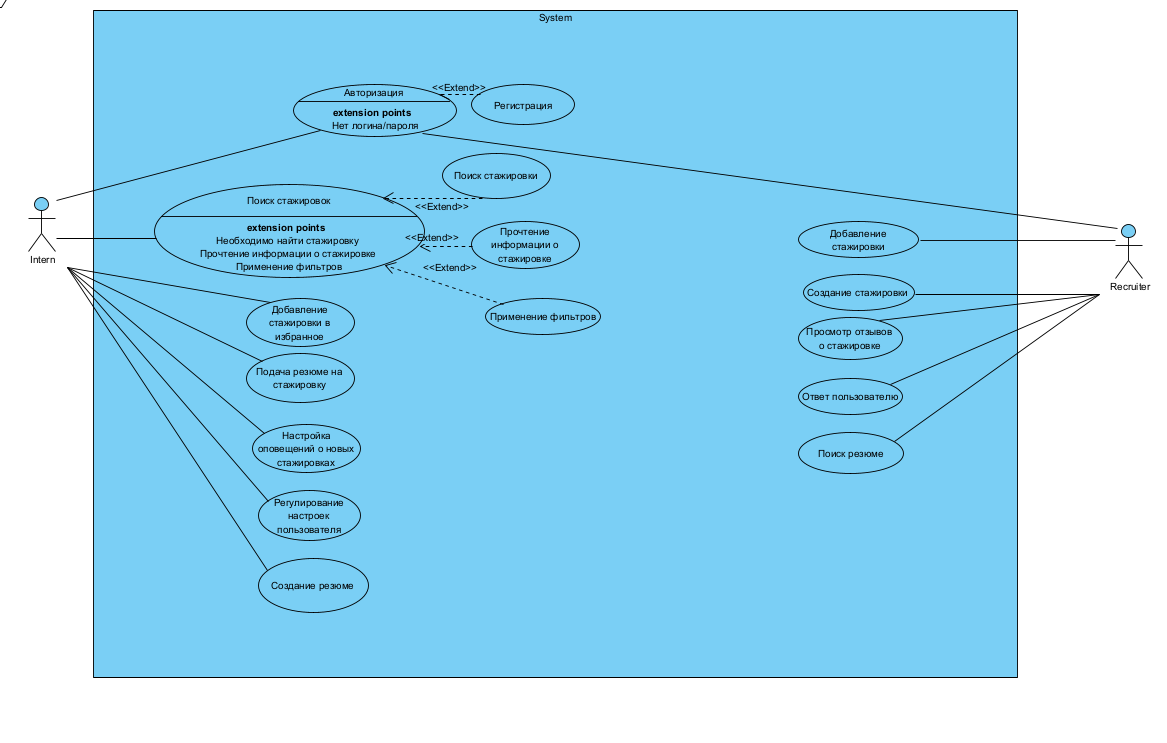


Рис. 2.1 Use Case

DFD-диаграмма (рис. 2.2,2.3):

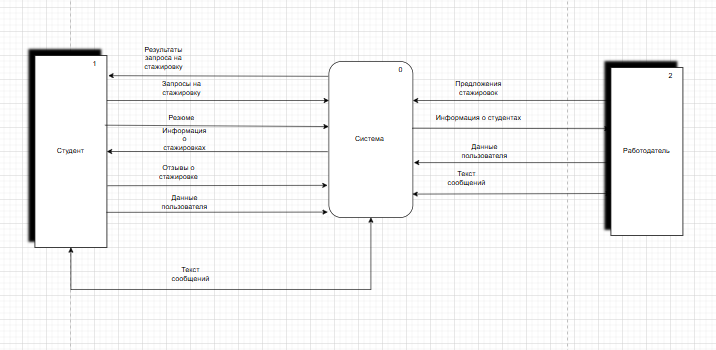


Рис. 2.2 DFD-диаграмма

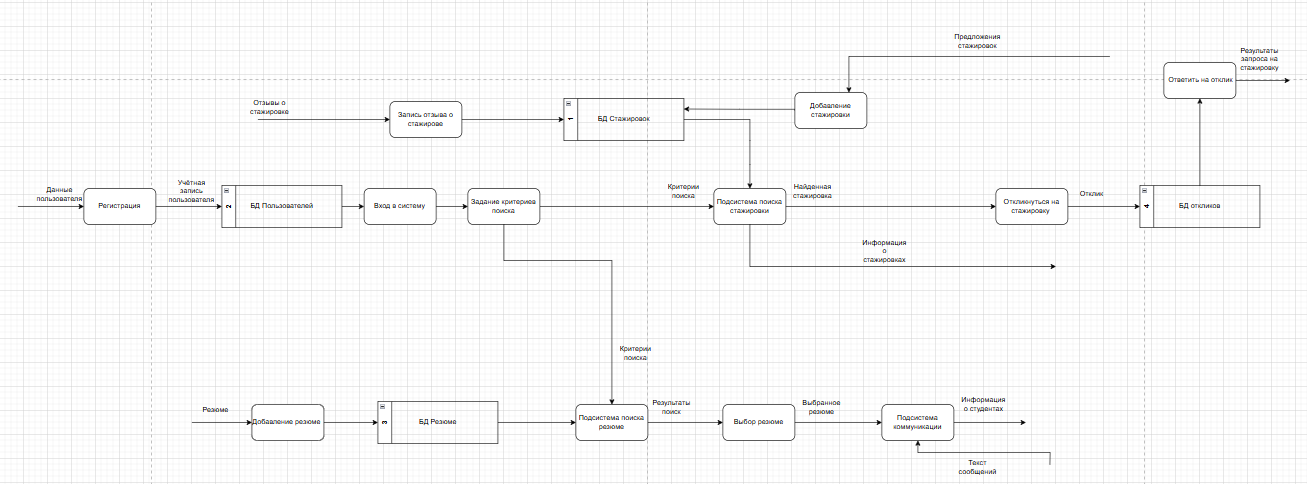


Рис. 2.3 Декомпозиция DFD-диаграммы

Для реализации данной системы будет выбран архитектурный паттерн MVC.

## **2.2 Проектирование БД**

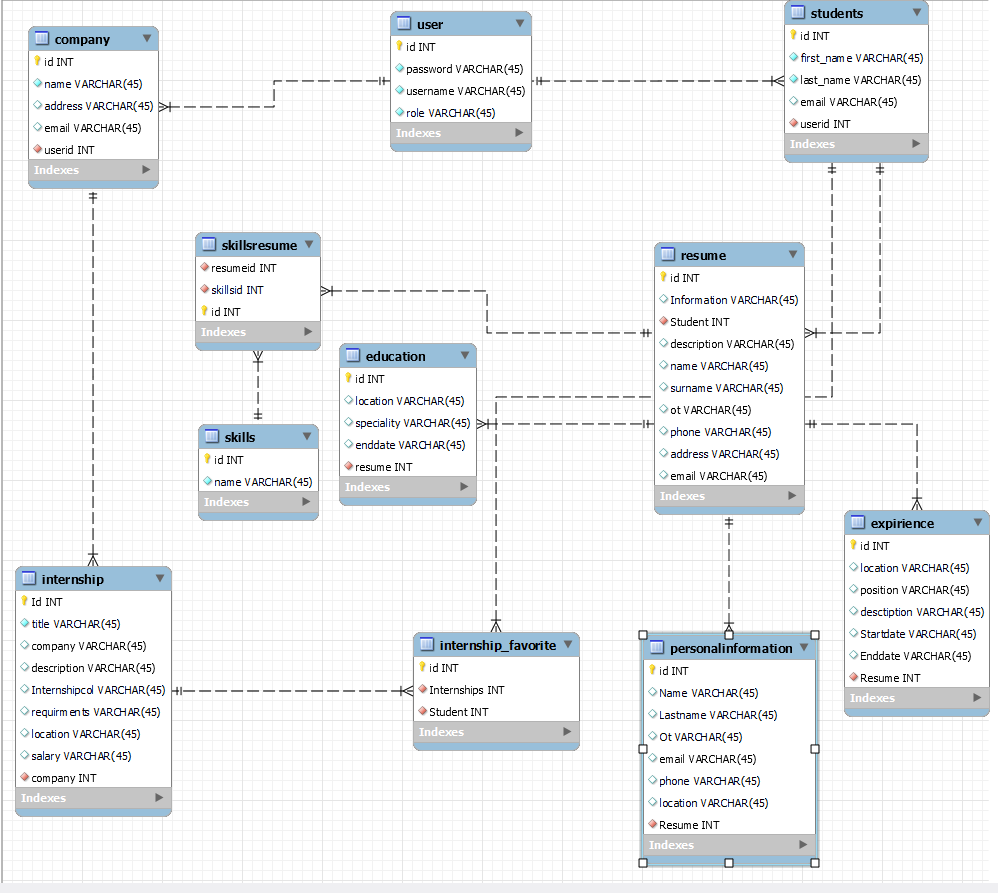
****

Рис. 2.4 ER-диаграмма

Описание основных таблиц:

Таблицы user, student и company отвечают за пользователей системы. В student и company хранится основная информация об студенте или компании, а в таблице user хранятся информация, нужная для входа в систему, а именно username и пароль, и роли пользователей.

В таблице resume хранится основная информация об резюме, а в таблицах experience, skills, skillsresume и education информация, которая заполняется в данном резюме, а именно опыт работы, умения, образование и т.д.

Таблица Internships отвечает за стажировки и информацию про стажировки, а internship\_favorite отвечает за стажировки, которые студент добавил в избранное

## **2.3 Проектирование интерфейса**

Скетчи основных представлений приложений (выполнено в Balsamiq Mockups):

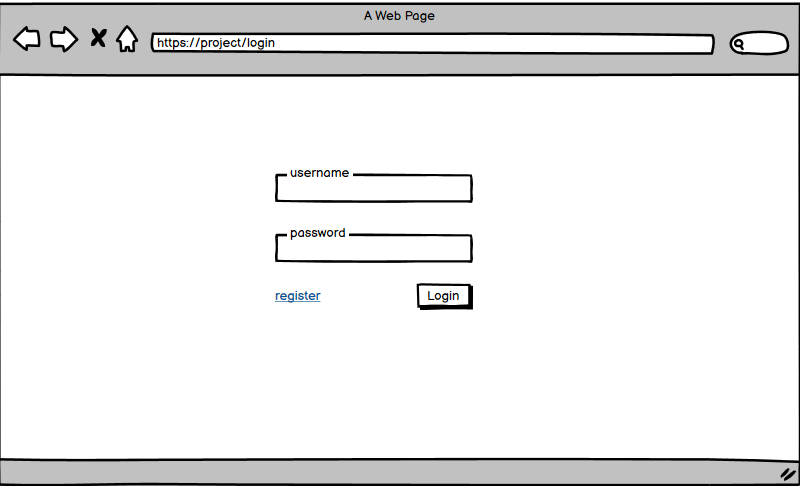


Рис. 2.3.1. Скетч страницы входа

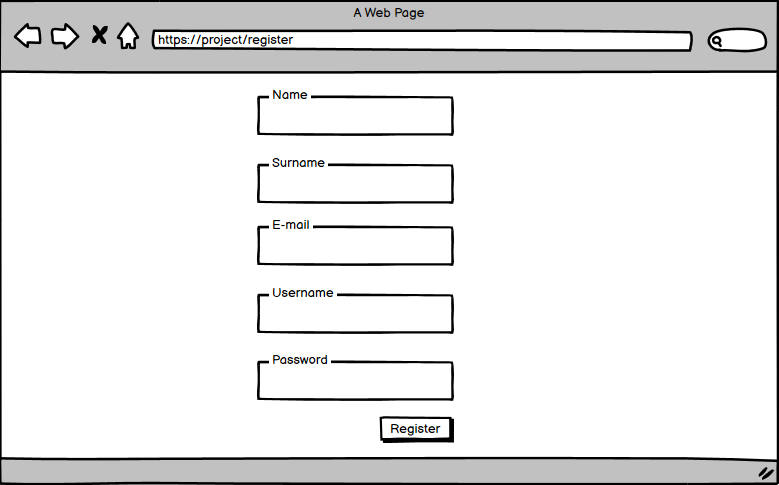


Рис. 2.3.2. Скетч страницы регистрации

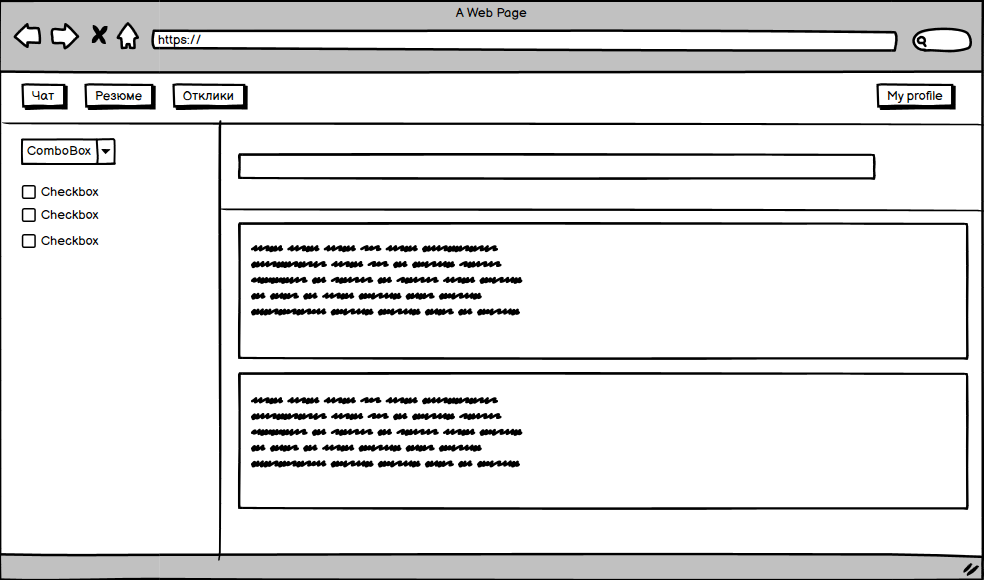


Рис. 2.3.3. Скетч страницы поиска

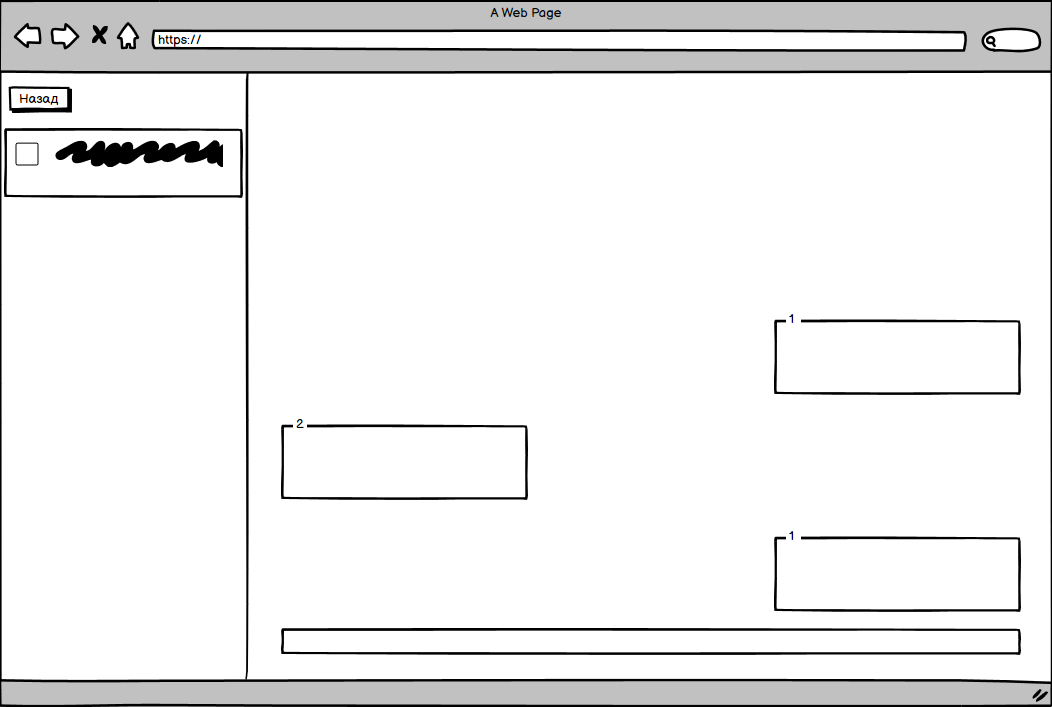


Рис. 2.3.4. Скетч страницы чата

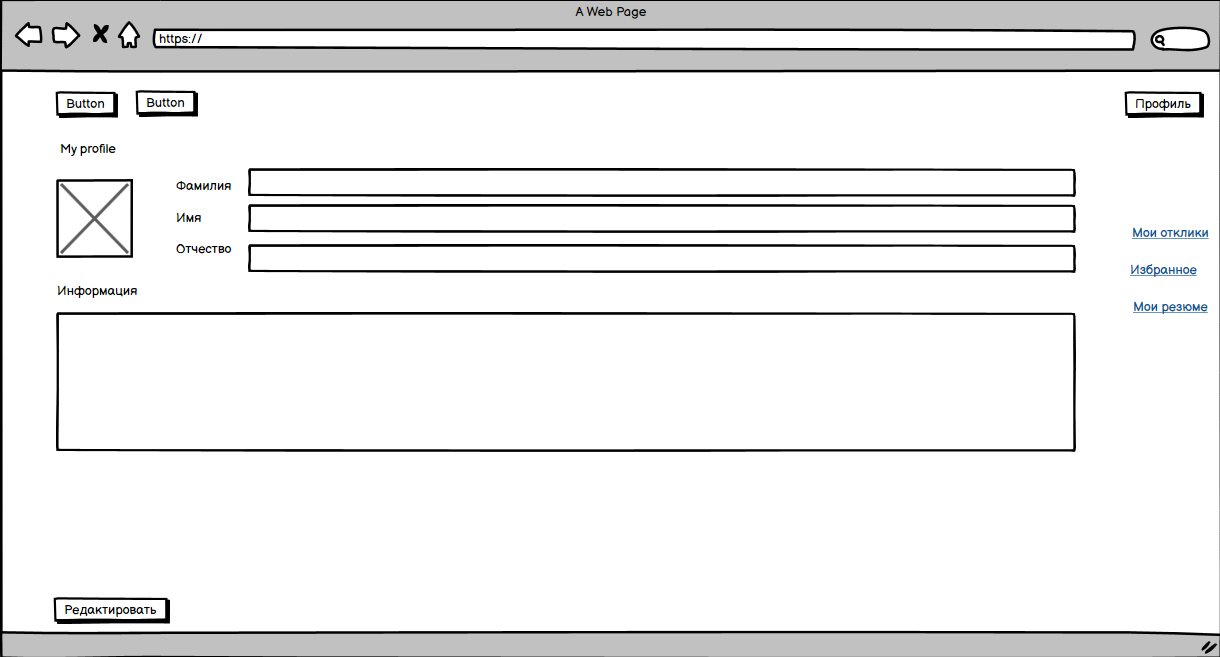


Рис. 2.3.5. Скетч страницы профиля

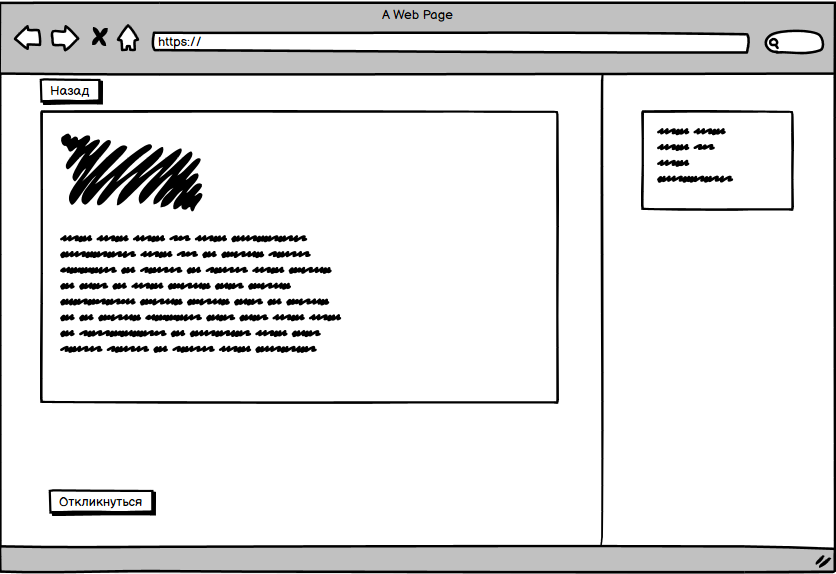


Рис. 2.3.6. Скетч страницы стажировки

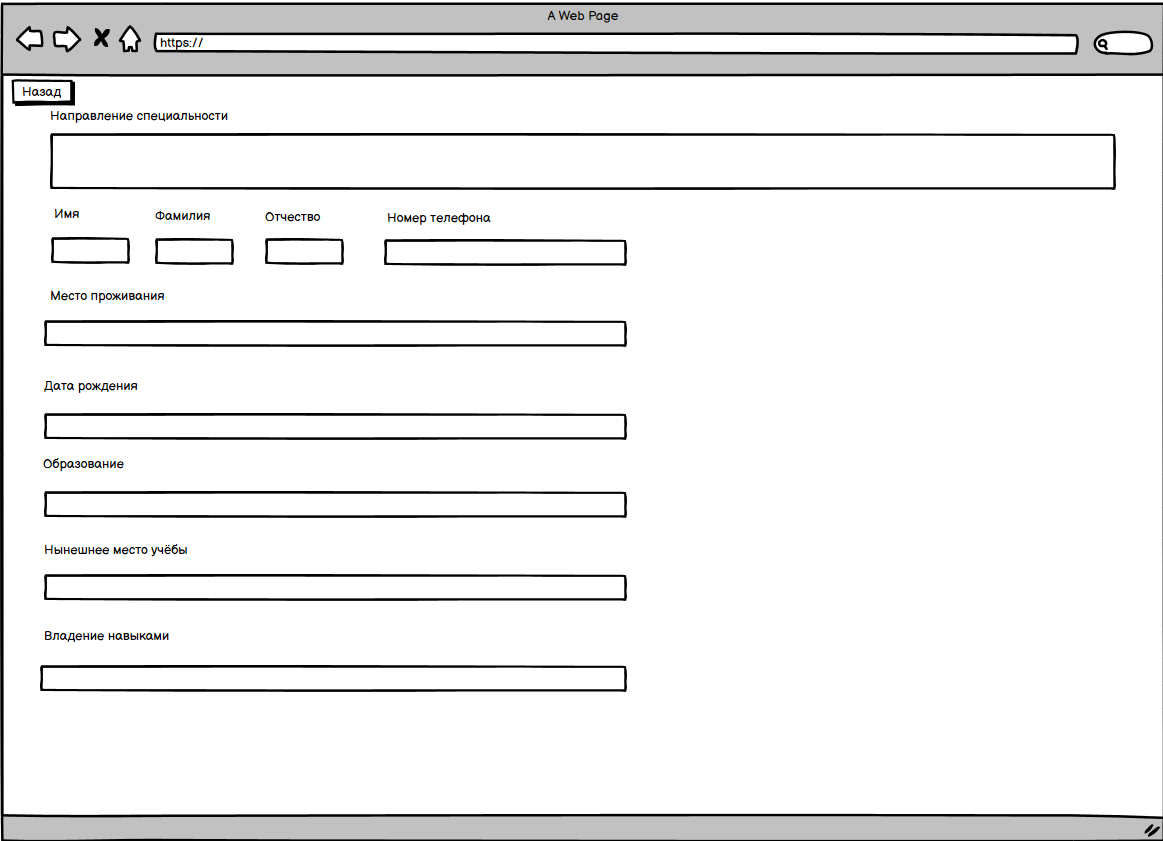


Рис 2.3.7 Скетч создания резюме

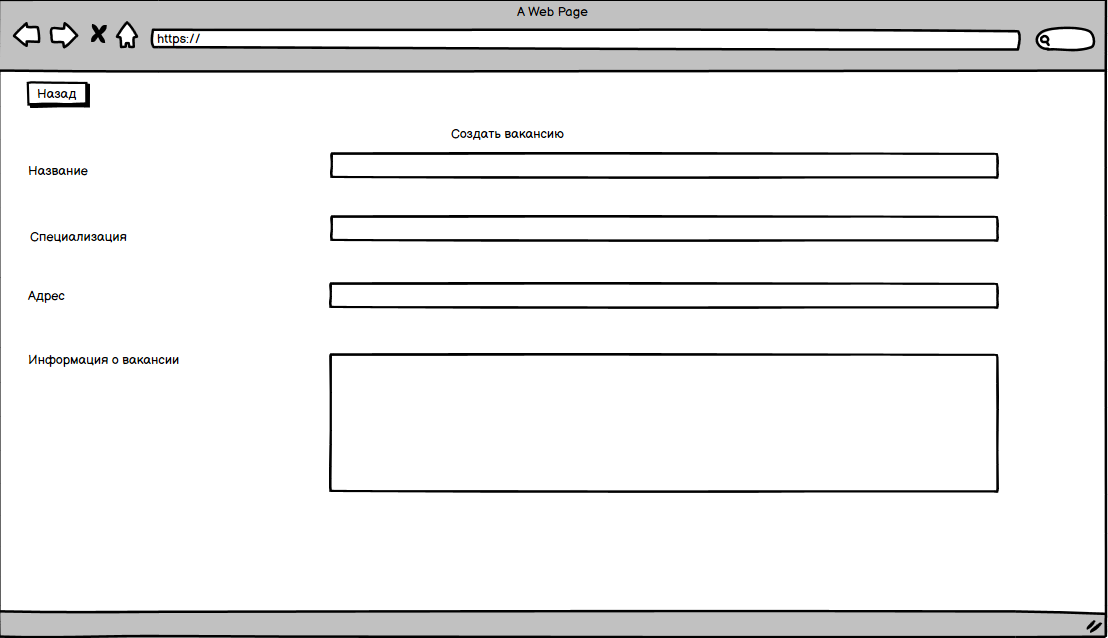


Рис 2.3.8 Скетч создания вакансии

# **3. Разработка системы**

## **3.1 Выбор средств реализации**

Для реализации данной системы я выбрал язык программирования Java. Данный язык программирования знаком мне, в нём есть множество сторонних библиотек для людей задач и приложения, написанные на Java, как правило, отличаются стабильной и надежной работой.

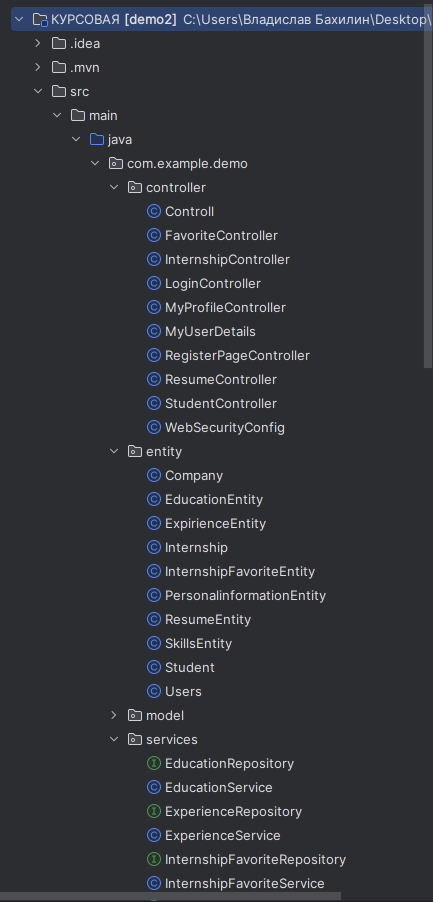
В качестве СУБД я выбрал MySQL. Данная СУБД простая в использовании и хорошо интегрируется с Java и Spring.

Среди фреймворков я выбрал Spring Framework для разработки веб-приложения. Один из его модулей – это Spring Boot, который я использовал, который упрощает разработку Spring-приложений. Он автоматически настраивает компоненты Spring Framework, у него есть встроенный веб-сервер и Spring Boot предоставляет так называемые “starter dependencies”, которые объединяют в себе множество зависимостей для конкретных задач. Spring MVC – компонент Spring Framework, который реализует в моём приложении паттерн MVC. И Spring Data JPA – это модуль, упрощающий работу с базой данных. Также я использовал Thymeleaf. Thymeleaf — это шаблонизатор, упрощающий вёрстку и работу с данными

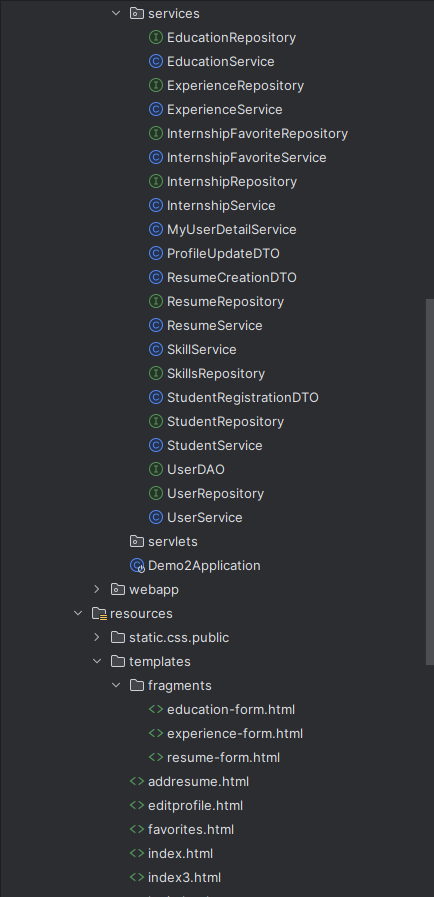
Из библиотек, которые я использовал, я могу выделить Lombok, который я использовал для геттеров/сеттеров и конструкторов.

## **3.2 Структура проекта**

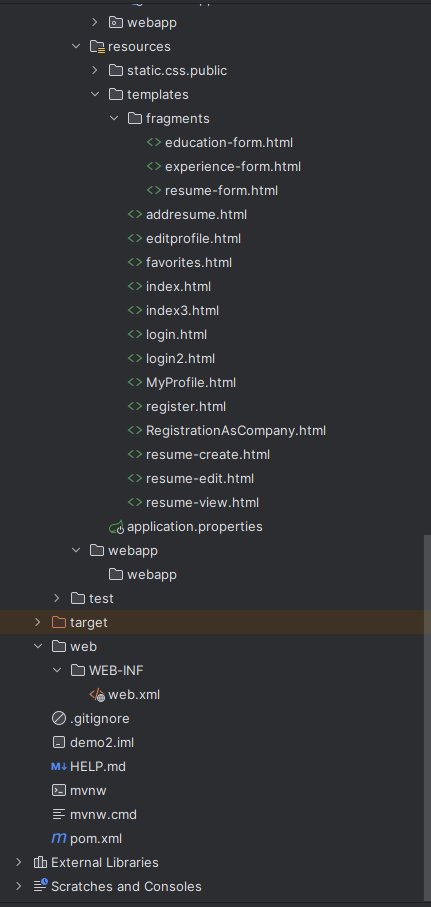
Дерево проекта:



Скриншот 3.2.1 Дерево проекта (1)

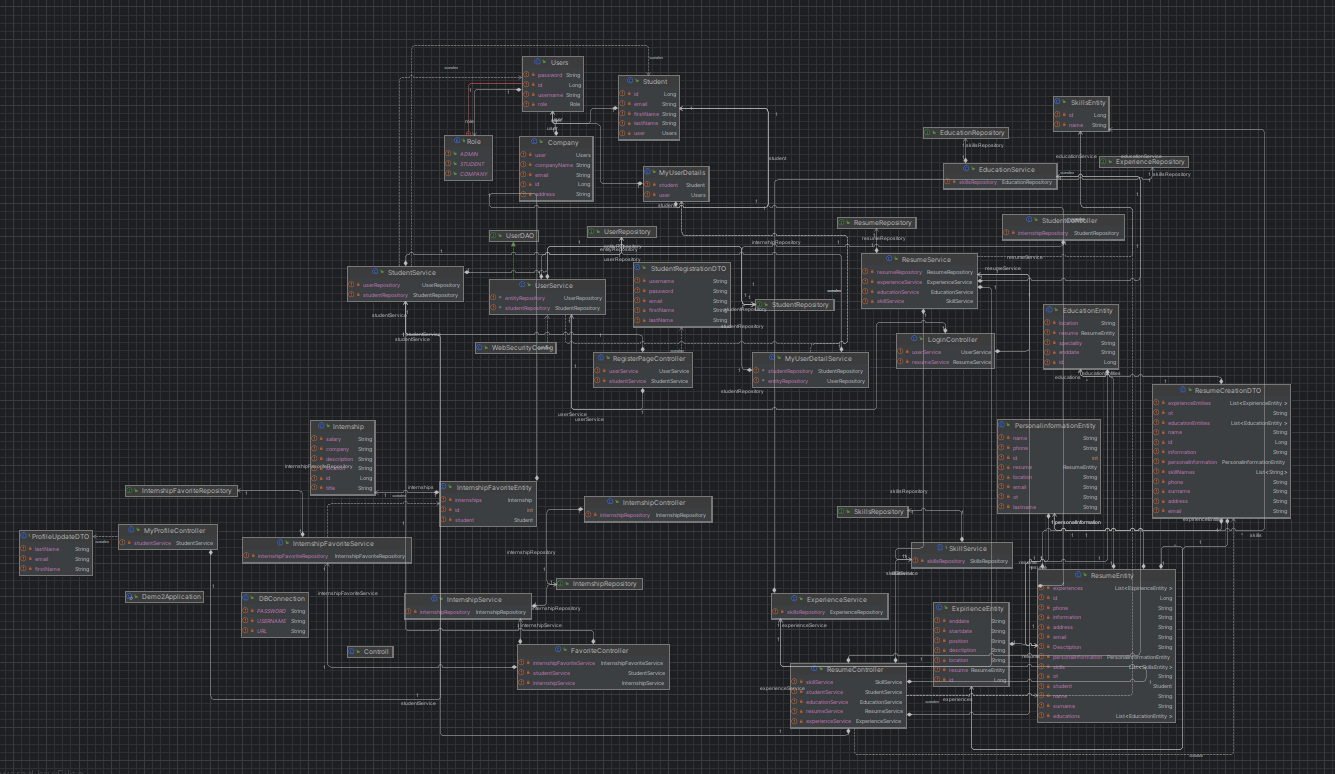


Скриншот 3.2.2 Дерево проекта (2)



Скриншот 3.2.3 Дерево проекта (3)

Дерево классов:



Структура решения:

Приложение построение на основе многослойной архитектуры с использованием паттерна MVC. Приложение построено следующим образом:

Controllers (ResumeController, InternshipFavoriteController, StudentController.). Они обрабатывают HTTP-запросы от пользователя (например, GET, POST), получают данные пользователя, вызывают методы сервисов, передают данные в HTML-шаблоны и перенаправляют пользователя на страницы.

HTML: profile.html, resume-create.html, favorites.html и другие. Они отображают данные пользователю в браузере и содержат формы для ввода данных

Services: (ResumeService, InternshipFavoriteService, StudentService, SkillService, EducationService, ExperienceService и др.). Содержат бизнес-логику приложения и вызывают методы репозиториев для работы с базой данных.

DTOs (Data Transfer Objects): (ResumeCreationDTO и др.). Используются для передачи данных между контроллерами и сервисами.

Entities: (ResumeEntity, Student, InternshipFavoriteEntity, EducationEntity, ExpirienceEntity, SkillsEntity и др.). Представляют данные приложения.

Repositories: (ResumeRepository, InternshipFavoriteRepository, StudentRepository, SkillsRepository и др.). Обеспечивают доступ к данным в базе данных..

Датабаза: MySQL

## **3.3 Реализация**

**Основные функции:**

Следующая функция отвечает за получение данных из формы создания резюме, создание сущности ResumeEntity и заполнение её данными из формы, обработку списков навыков, образования и опыта, сохранение резюме в базу данных через сервис resumeService и перенаправление пользователя на страницу профиля после успешного создания резюме.

@GetMapping("/resume/create")  
public String showResumeCreationForm(Model model) {  
 model.addAttribute("resumeCreationDto", new ResumeCreationDTO());  
 return "resume-create";  
}  
  
@PostMapping("/resume/create")  
public String submitResumeCreationForm(@Valid @ModelAttribute("resumeCreationDto") ResumeCreationDTO resumeCreationDto,  
 BindingResult bindingResult, Authentication authentication) {  
 if (bindingResult.hasErrors()) {  
 return "resume-create";  
 }  
  
 MyUserDetails userDetails = (MyUserDetails) authentication.getPrincipal();  
 Long userId = userDetails.getUser().getId();  
  
 ResumeEntity resume = new ResumeEntity();  
 resume.setInformation(resumeCreationDto.getInformation());  
 resume.setName(resumeCreationDto.getName());  
 resume.setSurname(resumeCreationDto.getSurname());  
 resume.setOt(resumeCreationDto.getOt());  
 resume.setPhone(resumeCreationDto.getPhone());  
 resume.setEmail(resumeCreationDto.getEmail());  
 resume.setAddress(resumeCreationDto.getAddress());  
  
  
  
 List<EducationEntity> educationEntities = resumeCreationDto.getEducationEntities();  
 if (educationEntities != null) {  
 for (EducationEntity education : educationEntities) {  
 education.setResume(resume);  
 }  
 resume.setEducations(educationEntities);  
 }  
  
  
 // Set the resume on the experience entities  
 List<ExpirienceEntity> experienceEntities = resumeCreationDto.getExpirienceEntities();  
 if(experienceEntities != null) {  
 for (ExpirienceEntity experience: experienceEntities) {  
 experience.setResume(resume);  
 }  
 resume.setExperiences(experienceEntities);  
 }  
  
 List<String> skillNames = resumeCreationDto.getSkillNames();  
 if(skillNames != null && !skillNames.isEmpty()){  
 List<SkillsEntity> skills = new ArrayList<>();  
 for(String skillName : skillNames){  
 // Check if the skill already exists  
 SkillsEntity existingSkill = skillService.findSkillsEntityByName(skillName);  
 if(existingSkill != null){  
 skills.add(existingSkill);  
 }  
 else {  
 // Create a new skills entity and save it  
 SkillsEntity skillsEntity = new SkillsEntity();  
 skillsEntity.setName(skillName);  
 SkillsEntity savedSkill = skillService.saveSkill(skillsEntity);  
 skills.add(savedSkill);  
 }  
 }  
 resume.setSkills(skills);  
 }  
  
 Student student = studentService.getStudentById(userId);  
 resume.setStudent(student);  
 resumeService.createResume(resume);  
 return "redirect:/profile/profile";  
}

Листинг 3.1 Создание резюме

Следующий метод служит безопасности, и отвечает за то, чтобы обозначать, какие запросы доступны каким ролям

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 return http  
 .csrf(AbstractHttpConfigurer::disable) // Only disable if absolutely necessary  
 .authorizeHttpRequests(auth -> auth  
 .requestMatchers("/login\*", "/register\*", "/main", "/css/\*\*", "/js/\*\*", "/images/\*\*").permitAll()  
 .anyRequest().authenticated() // All other requests require authentication  
 )  
 .formLogin(form -> form  
 .loginPage("/login")  
 .permitAll()  
 .defaultSuccessUrl("/main", true)  
 )  
 .logout(LogoutConfigurer::permitAll) // Add logout functionality  
 .httpBasic(AbstractHttpConfigurer::disable) //Generally recommended to disable basic auth  
 .build();  
 }  
}

Листинг 3.2 Безопасность входа

public class ResumeService {  
  
 private final ResumeRepository resumeRepository;  
  
 private final EducationService educationService;  
 private final ExperienceService experienceService;  
 private final SkillService skillService;  
  
  
  
 @Autowired  
 public ResumeService(ResumeRepository resumeRepository, EducationService educationService, ExperienceService experienceService, SkillService skillService) {  
 this.resumeRepository = resumeRepository;  
 this.educationService = educationService;  
 this.experienceService = experienceService;  
 this.skillService = skillService;  
 }  
  
  
 public void createResume(ResumeEntity resume) {  
 resumeRepository.save(resume);  
 }  
  
  
 public ResumeEntity getResumeById(Long id) {  
 return resumeRepository.findById(id)  
 .orElse(null);  
 }  
  
 public List<ResumeEntity> getResumesByUserId(Long userId) {  
 return resumeRepository.findByStudentId(userId);  
 }  
  
 @Transactional  
 public void deleteResumeById(Long id) {  
 if (!resumeRepository.existsById(id)) {  
 throw new RuntimeException("Resume not found with id: " + id);  
 }  
 resumeRepository.deleteById(id);  
 }  
 public List<ResumeEntity> getAllResume(){  
 return new ArrayList<>(resumeRepository.findAll());  
 }  
  
  
 @Transactional  
 public void updateResume(ResumeEntity resume, ResumeCreationDTO resumeCreationDTO) {  
 ResumeEntity existingResume = resumeRepository.findById(resumeCreationDTO.getId()).orElseThrow(() -> new RuntimeException("Resume not found"));  
 // Check that the user owns this resume  
 if (!Objects.*equals*(existingResume.getStudent().getId(), resume.getStudent().getId())) {  
 throw new RuntimeException("User does not own this resume, and cannot update it");  
 }  
  
 existingResume.setInformation(resumeCreationDTO.getInformation());  
 existingResume.setName(resumeCreationDTO.getName());  
 existingResume.setSurname(resumeCreationDTO.getSurname());  
 existingResume.setOt(resumeCreationDTO.getOt());  
 existingResume.setEmail(resumeCreationDTO.getEmail());  
 existingResume.setAddress(resumeCreationDTO.getAddress());  
 existingResume.setPhone(resumeCreationDTO.getPhone());  
  
  
 // Обновляем experience  
 List<ExpirienceEntity> updatedExperienceEntities = resumeCreationDTO.getExpirienceEntities();  
 if (updatedExperienceEntities != null) {  
 if(existingResume.getExperiences() == null){  
 existingResume.setExperiences(new ArrayList<>());  
 }  
 List<ExpirienceEntity> existingExperienceEntities = new ArrayList<>(existingResume.getExperiences());  
  
 for(ExpirienceEntity updatedExperience : updatedExperienceEntities){  
 Long updatedExperienceId = updatedExperience.getId() != 0 ? updatedExperience.getId() : null;  
 if(updatedExperienceId != null){  
 ExpirienceEntity existingExperience = existingExperienceEntities.stream()  
 .filter(exp -> Long.*valueOf*(exp.getId()).equals(updatedExperienceId))  
 .findFirst()  
 .orElse(null);  
 if(existingExperience != null) {  
 existingExperience.setLocation(updatedExperience.getLocation());  
 existingExperience.setPosition(updatedExperience.getPosition());  
 existingExperience.setDesctiption(updatedExperience.getDesctiption());  
 existingExperience.setStartdate(updatedExperience.getStartdate());  
 existingExperience.setEnddate(updatedExperience.getEnddate());  
 existingExperienceEntities.remove(existingExperience);  
 } else {  
 updatedExperience.setResume(existingResume);  
 existingResume.getExperiences().add(updatedExperience);  
 }  
 } else {  
 updatedExperience.setResume(existingResume);  
 existingResume.getExperiences().add(updatedExperience);  
 }  
 }  
 for(ExpirienceEntity experienceToDelete : existingExperienceEntities){  
 experienceService.deleteExperience(experienceToDelete.getId());  
 existingResume.getExperiences().remove(experienceToDelete);  
 }  
 }  
 else if (existingResume.getExperiences() != null) {  
 // no updated experiences, so we should delete the existing ones  
 for (ExpirienceEntity experience: existingResume.getExperiences()) {  
 experienceService.deleteExperience(experience.getId());  
 }  
 existingResume.setExperiences(null);  
 }  
 //Обновляем education  
 List<EducationEntity> updatedEducations = resumeCreationDTO.getEducationEntities();  
 if (updatedEducations != null) {  
 if (existingResume.getEducations() == null) {  
 existingResume.setEducations(new ArrayList<>());  
 }  
 List<EducationEntity> existingEducationEntities = new ArrayList<>(existingResume.getEducations());  
 for (EducationEntity updatedEducation : updatedEducations) {  
 if (updatedEducation.getId() != null) {  
 EducationEntity existingEducation = existingEducationEntities.stream()  
 .filter(edu -> updatedEducation.getId().equals(edu.getId()))  
 .findFirst().orElse(null);  
  
 if(existingEducation != null){  
 existingEducation.setLocation(updatedEducation.getLocation());  
 existingEducation.setSpeciality(updatedEducation.getSpeciality());  
 existingEducation.setEnddate(updatedEducation.getEnddate());  
 existingEducationEntities.remove(existingEducation);  
 }  
 else {  
 updatedEducation.setResume(existingResume);  
 existingResume.getEducations().add(updatedEducation);  
 }  
 }else {  
 updatedEducation.setResume(existingResume);  
 existingResume.getEducations().add(updatedEducation);  
 }  
 }  
 for (EducationEntity educationToDelete : existingEducationEntities) {  
 educationService.deleteEducation(educationToDelete.getId());  
 existingResume.getEducations().remove(educationToDelete);  
 }  
 }  
 else if (existingResume.getEducations() != null) {  
 // no updated educations, so we should delete the existing ones  
 for (EducationEntity education : existingResume.getEducations()) {  
 educationService.deleteEducation(education.getId());  
 }  
 existingResume.setEducations(null);  
 }  
 //Обновляем skills  
 List<String> updatedSkillsNames = resumeCreationDTO.getSkillNames();  
 if(updatedSkillsNames != null) {  
 List<SkillsEntity> skills = new ArrayList<>();  
 if (existingResume.getSkills() != null) {  
 List<SkillsEntity> existingSkills = new ArrayList<>(existingResume.getSkills());  
 for (String skillName : updatedSkillsNames) {  
 SkillsEntity existingSkill = existingSkills.stream()  
 .filter(skill -> skill.getName().equals(skillName))  
 .findFirst()  
 .orElse(null);  
 if (existingSkill != null) {  
 skills.add(existingSkill);  
 existingSkills.remove(existingSkill);  
 }  
 else {  
 SkillsEntity skillsEntity = new SkillsEntity();  
 skillsEntity.setName(skillName);  
 SkillsEntity savedSkill = skillService.saveSkill(skillsEntity);  
 skills.add(savedSkill);  
 }  
 }  
 for (SkillsEntity skillToDelete : existingSkills) {  
 existingResume.getSkills().remove(skillToDelete);  
 }  
 }  
 else {  
 for (String skillName : updatedSkillsNames) {  
 SkillsEntity existingSkill = skillService.findSkillsEntityByName(skillName);  
 if (existingSkill != null) {  
 skills.add(existingSkill);  
 } else {  
 SkillsEntity skillsEntity = new SkillsEntity();  
 skillsEntity.setName(skillName);  
 SkillsEntity savedSkill = skillService.saveSkill(skillsEntity);  
 skills.add(savedSkill);  
 }  
 }  
 }  
 existingResume.setSkills(skills);  
 }  
 else if (existingResume.getSkills() != null) {  
 existingResume.getSkills().clear();  
 }  
 System.*out*.println(existingResume.getId()+"1");  
 resumeRepository.save(existingResume);  
  
 }  
  
 @Transactional  
 public void deleteResume(Long resumeId) {  
 ResumeEntity resume = resumeRepository.findById(resumeId).orElseThrow(() -> new RuntimeException("Resume not found with id: " + resumeId));  
  
 if (resume.getEducations() != null){  
 for (EducationEntity education : resume.getEducations()) {  
 educationService.deleteEducation(education.getId());  
 }  
 }  
 if(resume.getExperiences() != null) {  
 for(ExpirienceEntity experience : resume.getExperiences()){  
 experienceService.deleteExperience(experience.getId());  
 }  
 }  
 if (resume.getPersonalInformation() != null){  
 // We should also delete the personal information associated with this record.  
 // personalinformationRepository.deleteById(resume.getPersonalInformation().getId());  
 }  
 resumeRepository.delete(resume);  
 }  
  
  
  
}

Листинг 3.3 Класс ResumeService, отвечающий за бизнес-логику резюме

@GetMapping("/resume/update/{id}")  
public String showEditResumeForm(@PathVariable("id") Long resumeId, Model model, Authentication authentication){  
 MyUserDetails userDetails = (MyUserDetails) authentication.getPrincipal();  
 Long userId = userDetails.getUser().getId();  
  
 ResumeEntity resume = resumeService.getResumeById(resumeId);  
 if (resume == null || !Objects.*equals*(resume.getStudent().getId(), userId)) {  
 return "redirect:/profile/profile?error=resumeNotFound";  
 }  
  
 ResumeCreationDTO resumeCreationDTO = new ResumeCreationDTO();  
 resumeCreationDTO.setId(resume.getId());  
 resumeCreationDTO.setInformation(resume.getInformation());  
  
 if(resume.getPersonalInformation() != null){  
 resumeCreationDTO.setPersonalInformation(resume.getPersonalInformation());  
 }  
  
 if(resume.getEducations() != null){  
 List<EducationEntity> educationEntities = new ArrayList<>();  
 for(EducationEntity education : resume.getEducations()){  
 educationEntities.add(education);  
 }  
 resumeCreationDTO.setEducationEntities(educationEntities);  
 }  
  
 if(resume.getExperiences() != null){  
 List<ExpirienceEntity> experienceEntities = new ArrayList<>();  
 for(ExpirienceEntity experience : resume.getExperiences()){  
 experienceEntities.add(experience);  
 }  
 resumeCreationDTO.setExpirienceEntities(experienceEntities);  
 }  
  
 if (resume.getSkills() != null){  
 List<String> skillNames = new ArrayList<>();  
 for(SkillsEntity skill: resume.getSkills()){  
 skillNames.add(skill.getName());  
 }  
 resumeCreationDTO.setSkillNames(skillNames);  
 }  
 model.addAttribute("resumeCreationDto", resumeCreationDTO);  
 model.addAttribute("resumeId", resumeId);  
 return "resume-edit";  
}  
  
  
@PostMapping("/resume/update/{id}")  
public String updateResume(@PathVariable("id") Long resumeId,  
 @Valid @ModelAttribute("resumeCreationDto") ResumeCreationDTO resumeCreationDto,  
 BindingResult bindingResult, Authentication authentication) {  
 if (bindingResult.hasErrors()) {  
 return "resume-edit";  
 }  
  
 MyUserDetails userDetails = (MyUserDetails) authentication.getPrincipal();  
 Long userId = userDetails.getUser().getId();  
 try {  
  
 ResumeEntity resume = resumeService.getResumeById(resumeId);  
 if (resume == null || !Objects.*equals*(resume.getStudent().getId(), userId)) {  
  
 return "redirect:/profile/profile?error=resumeNotFound";  
 }  
 resumeService.updateResume(resume, resumeCreationDto);  
  
 } catch (RuntimeException e){  
 return "redirect:/profile/profile?error=userDoesNotOwnResume";  
 }  
  
 return "redirect:/profile/profile";  
}

Листинг 3.4 Запросы GET и POST на обновление резюме

@GetMapping("/resume/{id}")  
public String showResumeDetails(@PathVariable Long id, Model model) {  
 ResumeEntity resume = resumeService.getResumeById(id);  
 if (resume == null) {  
 return "error";  
 }  
 model.addAttribute("resume", resume);  
 return "resume-view";  
}  
  
@PostMapping("/resume/delete/{id}")  
public String deleteResume(@PathVariable("id") Long resumeId, Authentication authentication){  
 MyUserDetails userDetails = (MyUserDetails) authentication.getPrincipal();  
 Long userId = userDetails.getUser().getId();  
 try {  
 ResumeEntity resume = resumeService.getResumeById(resumeId);  
 if (resume == null || !Objects.*equals*(resume.getStudent().getId(), userId)) {  
 return "redirect:/profile/profile?error=resumeNotFound";  
 }  
 resumeService.deleteResume(resumeId);  
 }  
 catch (RuntimeException e){  
 return "redirect:/profile/profile?error=userDoesNotOwnResume";  
 }  
 return "redirect:/profile/profile";  
}

Листинг 3.5 Запросы на удаление и просмотр резюме

@Entity  
@Table(name = "resume", schema = "internshipproject")  
public class ResumeEntity {  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Id  
 @Column(name = "id")  
 private Long id;  
 @Basic  
 @Column(name = "Information")  
 private String information;  
  
 @Getter  
 @Setter  
 @OneToMany(mappedBy = "resume", cascade = CascadeType.*ALL*, fetch = FetchType.*LAZY*, orphanRemoval = true)  
 private List<EducationEntity> educations;  
 @Getter  
 @Setter  
 @OneToMany(mappedBy = "resume", cascade = CascadeType.*ALL*, fetch = FetchType.*LAZY*, orphanRemoval = true)  
 private List<ExpirienceEntity> experiences;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "Student", nullable = false)  
 private Student student;  
  
 @ManyToMany(fetch = FetchType.*LAZY*)  
 @JoinTable(  
 name = "skillsresume",  
 joinColumns = @JoinColumn(name = "resumeid"),  
 inverseJoinColumns = @JoinColumn(name = "skillsid")  
 )  
 @Getter  
 @Setter  
 private List<SkillsEntity> skills;  
  
  
 @Getter  
 @Setter  
 @OneToOne(mappedBy = "resume", fetch = FetchType.*LAZY*)  
 private PersonalinformationEntity personalInformation;  
 @Getter  
 @Setter  
 @Basic  
 @Column(name = "description")  
 private String Desctiption;  
 @Getter  
 @Setter  
 @Basic  
 @Column(name = "name")  
 private String name;  
 @Getter  
 @Setter  
 @Basic  
 @Column(name = "surname")  
 private String surname;  
 @Getter  
 @Setter  
 @Basic  
 @Column(name = "ot")  
 private String ot;  
 @Getter  
 @Setter  
 @Basic  
 @Column(name = "phone")  
 private String phone;  
 @Getter  
 @Setter  
  
 @Basic  
 @Column(name = "address")  
 private String address;  
 @Getter  
 @Setter  
  
 @Basic  
 @Column(name = "email")  
 private String email;  
  
  
 public ResumeEntity() {  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(long id) {  
 this.id = id;  
 }  
  
 public String getInformation() {  
 return information;  
 }  
  
 public void setInformation(String information) {  
 this.information = information;  
 }  
  
 public Student getStudent() {  
 return student;  
 }  
  
 public void setStudent(Student student) {  
 this.student = student;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 ResumeEntity that = (ResumeEntity) o;  
 return id == that.id && student == that.student && Objects.*equals*(information, that.information);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(id, information, student);  
 }  
  
}

Листинг 3.6 Класс для сущности Resume

**Создание БД**

DROP TABLE IF EXISTS `company`;

CREATE TABLE `company` (

`id` int NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL,

`address` varchar(45) DEFAULT NULL,

`email` varchar(45) DEFAULT NULL,

`userid` int NOT NULL,

PRIMARY KEY (`id`),

KEY `userid\_idx` (`userid`),

CONSTRAINT `userid` FOREIGN KEY (`userid`) REFERENCES `user` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8mb3;

LOCK TABLES `company` WRITE;

INSERT INTO `company` VALUES (1,'gh','gh','gh@mail.ru',2);

UNLOCK TABLES;

DROP TABLE IF EXISTS `education`;

CREATE TABLE `education` (

`id` int NOT NULL AUTO\_INCREMENT,

`location` varchar(45) DEFAULT NULL,

`speciality` varchar(45) DEFAULT NULL,

`enddate` varchar(45) DEFAULT NULL,

`resume` int NOT NULL,

PRIMARY KEY (`id`),

KEY `resume\_idx` (`resume`),

CONSTRAINT `fk\_education\_resume` FOREIGN KEY (`resume`) REFERENCES `resume` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8mb3;

LOCK TABLES `education` WRITE;

INSERT INTO `education` VALUES (1,'1','1','1',18),(2,'2','2','2',21),(3,'2','2','2',22);

UNLOCK TABLES;

DROP TABLE IF EXISTS `expirience`;

CREATE TABLE `expirience` (

`id` int NOT NULL AUTO\_INCREMENT,

`location` varchar(45) DEFAULT NULL,

`position` varchar(45) DEFAULT NULL,

`desctiption` varchar(45) DEFAULT NULL,

`Startdate` varchar(45) DEFAULT NULL,

`Enddate` varchar(45) DEFAULT NULL,

`Resume` int NOT NULL,

PRIMARY KEY (`id`),

KEY `Resume\_idx` (`Resume`),

CONSTRAINT `Resume` FOREIGN KEY (`Resume`) REFERENCES `resume` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8mb3;

LOCK TABLES `expirience` WRITE;

INSERT INTO `expirience` VALUES (1,NULL,'1',NULL,NULL,NULL,17),(2,NULL,'1',NULL,NULL,NULL,18);

UNLOCK TABLES;

DROP TABLE IF EXISTS `internship`;

CREATE TABLE `internship` (

`Id` int NOT NULL AUTO\_INCREMENT,

`title` varchar(45) NOT NULL,

`company` varchar(45) DEFAULT NULL,

`description` varchar(45) DEFAULT NULL,

`Internshipcol` varchar(45) DEFAULT NULL,

`requirments` varchar(45) DEFAULT NULL,

`location` varchar(45) DEFAULT NULL,

`salary` varchar(45) DEFAULT NULL,

PRIMARY KEY (`Id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8mb3;

LOCK TABLES `internship` WRITE;

INSERT INTO `internship` VALUES (1,'Программист-стажёр','comany',NULL,NULL,NULL,NULL,NULL),(2,'Стажёр 2','company2',NULL,NULL,NULL,NULL,NULL);

UNLOCK TABLES;

DROP TABLE IF EXISTS `internship\_favorite`;

CREATE TABLE `internship\_favorite` (

`id` int NOT NULL AUTO\_INCREMENT,

`Internships` int NOT NULL,

`Student` int NOT NULL,

PRIMARY KEY (`id`),

KEY `Internships\_idx` (`Internships`),

KEY `Student\_idx` (`Student`),

CONSTRAINT `Internships` FOREIGN KEY (`Internships`) REFERENCES `internship` (`Id`),

CONSTRAINT `Student` FOREIGN KEY (`Student`) REFERENCES `user` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8mb3;

LOCK TABLES `internship\_favorite` WRITE;

INSERT INTO `internship\_favorite` VALUES (1,1,1);

UNLOCK TABLES;

DROP TABLE IF EXISTS `personalinformation`;

CREATE TABLE `personalinformation` (

`id` int NOT NULL AUTO\_INCREMENT,

`Name` varchar(45) DEFAULT NULL,

`Lastname` varchar(45) DEFAULT NULL,

`Ot` varchar(45) DEFAULT NULL,

`email` varchar(45) DEFAULT NULL,

`phone` varchar(45) DEFAULT NULL,

`location` varchar(45) DEFAULT NULL,

`Resume` int NOT NULL,

PRIMARY KEY (`id`),

KEY `Resume\_idx` (`Resume`),

CONSTRAINT `fk\_PersonalInformation\_resume` FOREIGN KEY (`Resume`) REFERENCES `resume` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;

LOCK TABLES `personalinformation` WRITE;

UNLOCK TABLES;

DROP TABLE IF EXISTS `resume`;

CREATE TABLE `resume` (

`id` int NOT NULL AUTO\_INCREMENT,

`Information` varchar(45) DEFAULT NULL,

`Student` int NOT NULL,

`description` varchar(45) DEFAULT NULL,

`name` varchar(45) DEFAULT NULL,

`surname` varchar(45) DEFAULT NULL,

`ot` varchar(45) DEFAULT NULL,

`phone` varchar(45) DEFAULT NULL,

`address` varchar(45) DEFAULT NULL,

`email` varchar(45) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `Student\_idx` (`Student`),

CONSTRAINT `fk\_resume\_student` FOREIGN KEY (`Student`) REFERENCES `user` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=46 DEFAULT CHARSET=utf8mb3;

LOCK TABLES `resume` WRITE;

INSERT INTO `resume` VALUES (7,'1',1,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(9,'1',1,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(17,'1',1,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(18,'1',1,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(19,'2',1,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(20,'1',1,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(21,'2',1,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(22,'2',1,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(25,'1',1,NULL,'Владислав','Бахилин',NULL,NULL,NULL,'vladislav25702@gmail.com'),(26,'3',1,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(27,'5',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(28,'5',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(29,'7',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(30,'9',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(31,'9',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(32,'6',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(33,'9',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(34,'0',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(35,'1',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(36,'36',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(37,'8',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(38,'9',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(39,'18',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(40,'20',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(41,'20',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(42,'22',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(43,'30',3,NULL,NULL,NULL,NULL,NULL,NULL,NULL),(44,'1',3,NULL,'1','1','1','1','1','1'),(45,'fd',3,NULL,'df','df','df','df','df','df');

UNLOCK TABLES;

DROP TABLE IF EXISTS `skills`;

CREATE TABLE `skills` (

`id` int NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=13 DEFAULT CHARSET=utf8mb3;

LOCK TABLES `skills` WRITE;

INSERT INTO `skills` VALUES (1,'JAVA'),(2,'PYTHON'),(3,'html'),(4,'1'),(5,'2'),(6,'3'),(7,'5'),(8,'7'),(9,'9'),(10,'0'),(11,'12-1'),(12,'13-3');

UNLOCK TABLES;

DROP TABLE IF EXISTS `skillsresume`;

CREATE TABLE `skillsresume` (

`resumeid` int NOT NULL,

`skillsid` int NOT NULL,

`id` int NOT NULL AUTO\_INCREMENT,

PRIMARY KEY (`id`),

KEY `resumeid\_idx` (`resumeid`),

KEY `skillsid\_idx` (`skillsid`),

CONSTRAINT `resumeid` FOREIGN KEY (`resumeid`) REFERENCES `resume` (`id`),

CONSTRAINT `skillsid` FOREIGN KEY (`skillsid`) REFERENCES `skills` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=30 DEFAULT CHARSET=utf8mb3;

LOCK TABLES `skillsresume` WRITE;

INSERT INTO `skillsresume` VALUES (20,1,1),(20,2,2),(21,1,3),(21,2,4),(22,1,5),(22,3,6),(25,4,7),(26,6,8),(27,7,9),(28,8,10),(29,8,11),(30,8,12),(31,9,13),(32,8,14),(33,8,15),(34,10,16),(35,10,17),(37,10,19),(38,10,20),(39,10,21),(40,10,22),(41,10,23),(42,10,24),(43,10,25),(36,10,26),(44,4,27),(45,11,28),(45,12,29);

UNLOCK TABLES;

DROP TABLE IF EXISTS `students`;

CREATE TABLE `students` (

`id` int NOT NULL AUTO\_INCREMENT,

`first\_name` varchar(45) NOT NULL,

`last\_name` varchar(45) NOT NULL,

`email` varchar(45) DEFAULT NULL,

`userid` int NOT NULL,

PRIMARY KEY (`id`),

KEY `userid\_idx` (`userid`),

CONSTRAINT `useridstudent` FOREIGN KEY (`userid`) REFERENCES `user` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8mb3;

LOCK TABLES `students` WRITE;

INSERT INTO `students` VALUES (1,'Владислав','Бахилин','vladislav25702@gmail.com',1),(2,'1','1','1@mail.ru',3),(3,'2','2','1@mail.ru',4);

UNLOCK TABLES;

DROP TABLE IF EXISTS `user`;

CREATE TABLE `user` (

`id` int NOT NULL AUTO\_INCREMENT,

`password` varchar(45) NOT NULL,

`username` varchar(45) NOT NULL,

`role` varchar(45) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8mb3;

LOCK TABLES `user` WRITE;

INSERT INTO `user` VALUES (1,'1','vladislav25702','STUDENT'),(2,'2','company1','COMPANY'),(3,'1','1','STUDENT'),(4,'1','2','STUDENT');

UNLOCK TABLES;

Листинг 3.6 создание БД

**Скриншоты приложения:**

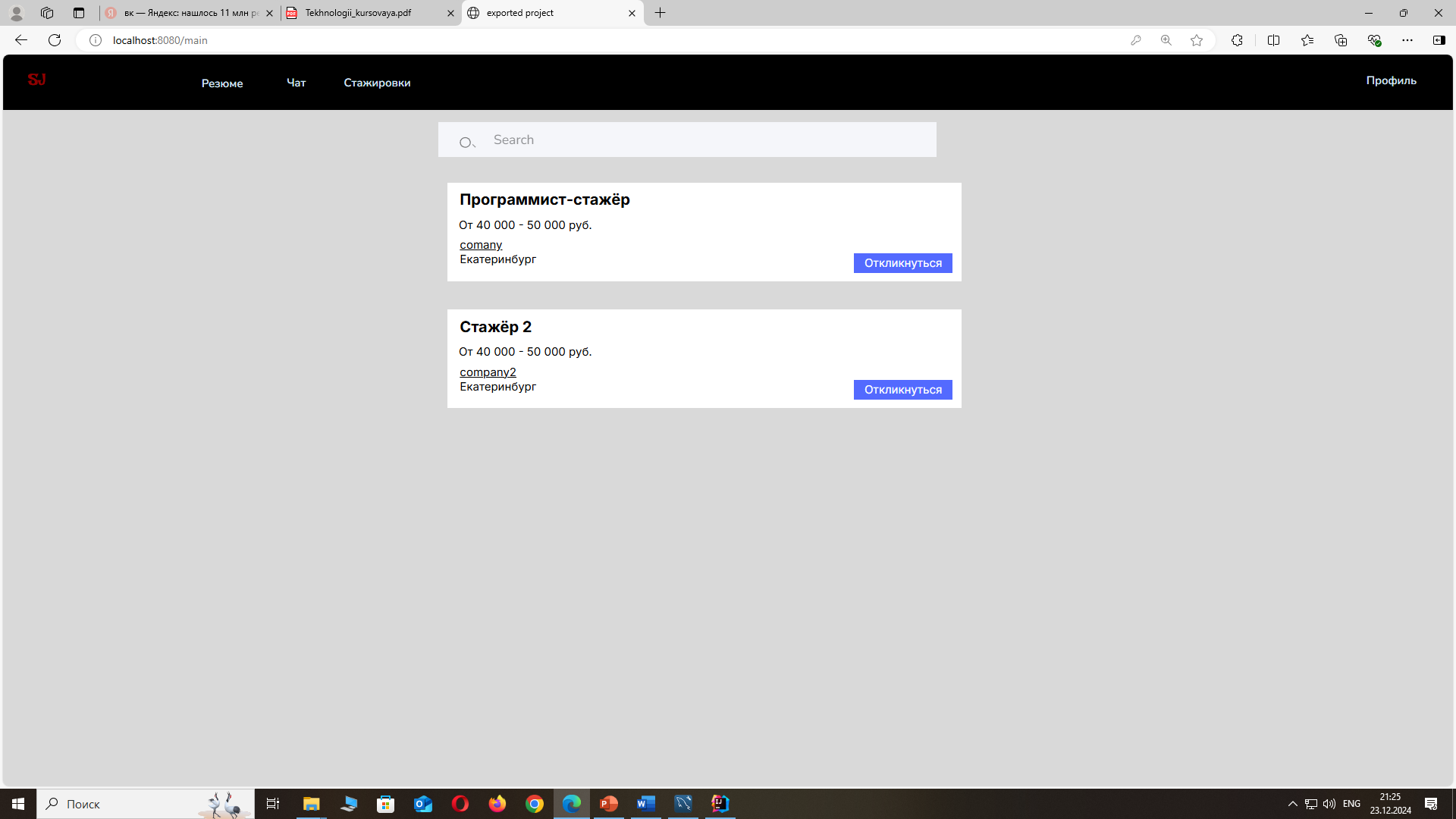


Рис. 3.1 Главная страница

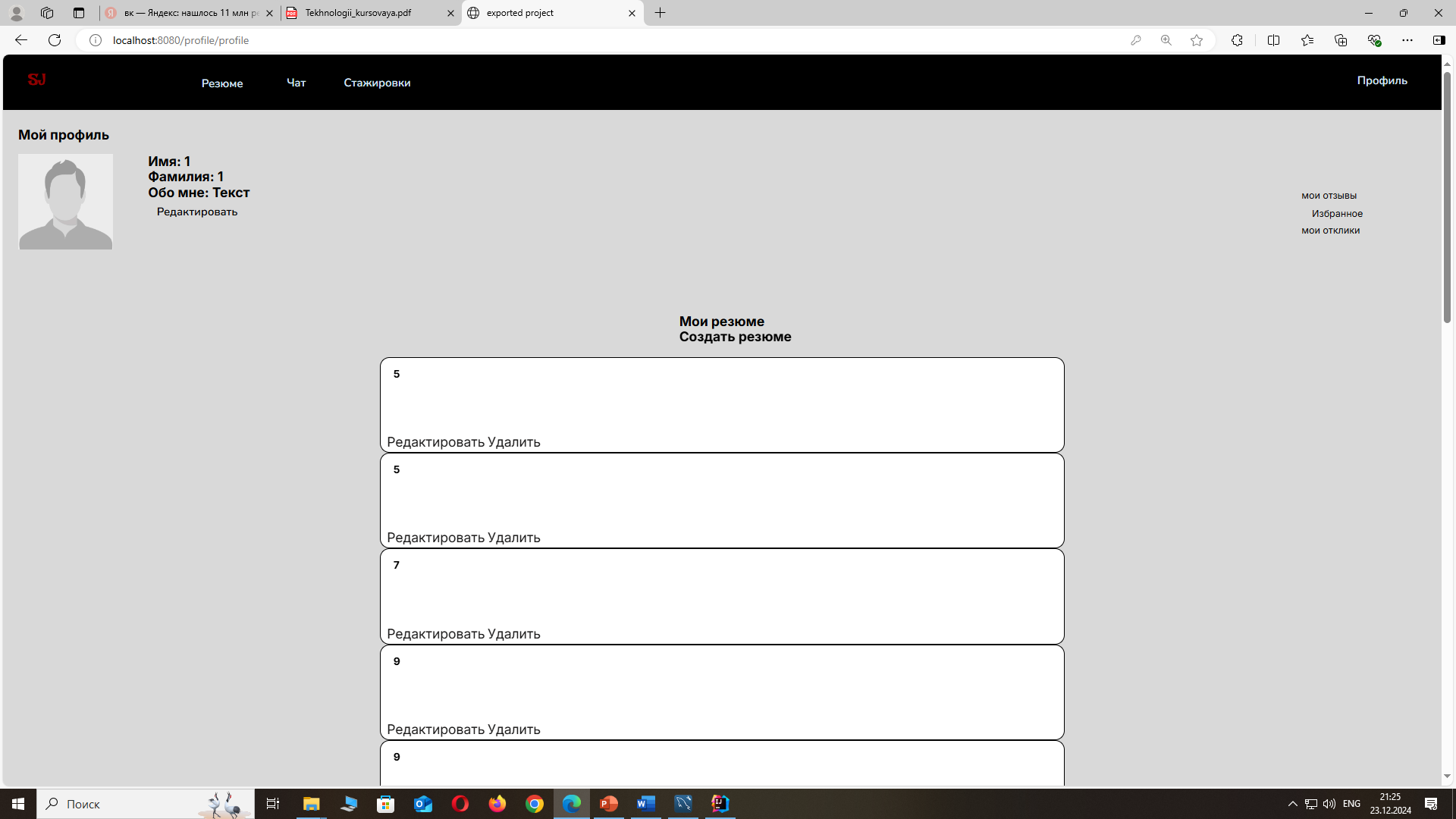


Рис. 3.2 Страница профиля

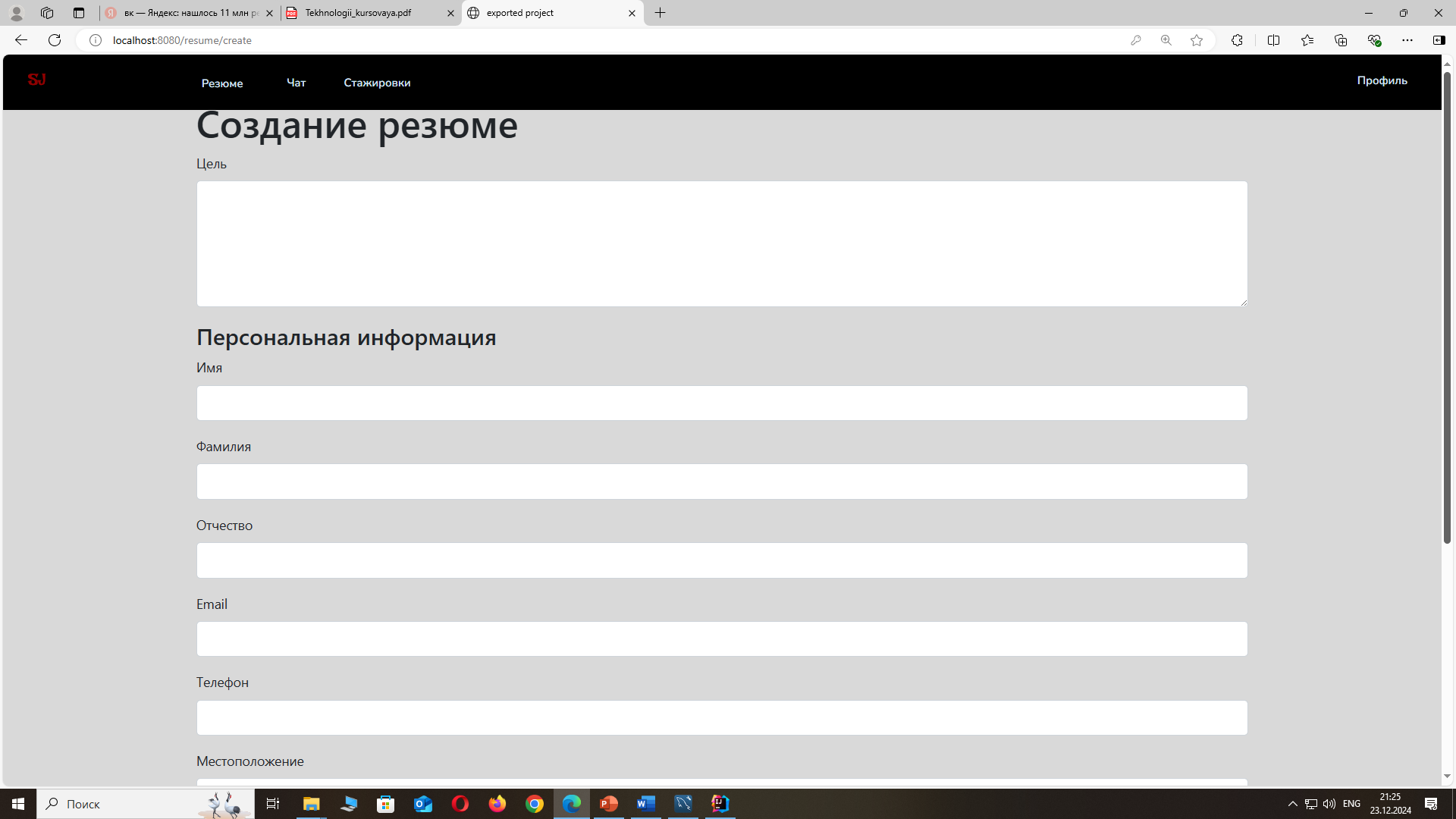


Рис. 3.3 Страница создания резюме

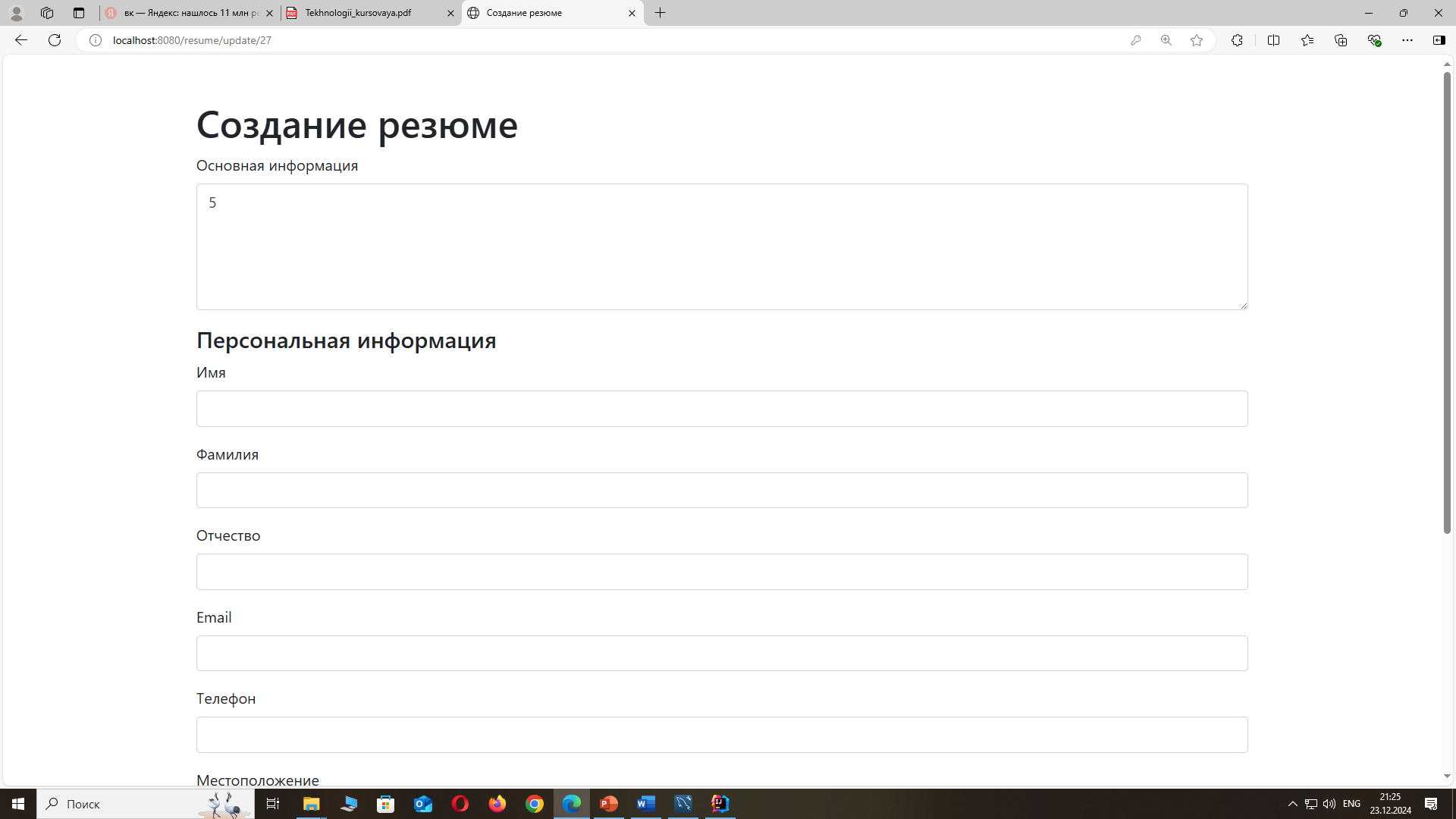


Рис. 3.4 Страница редактирования резюме

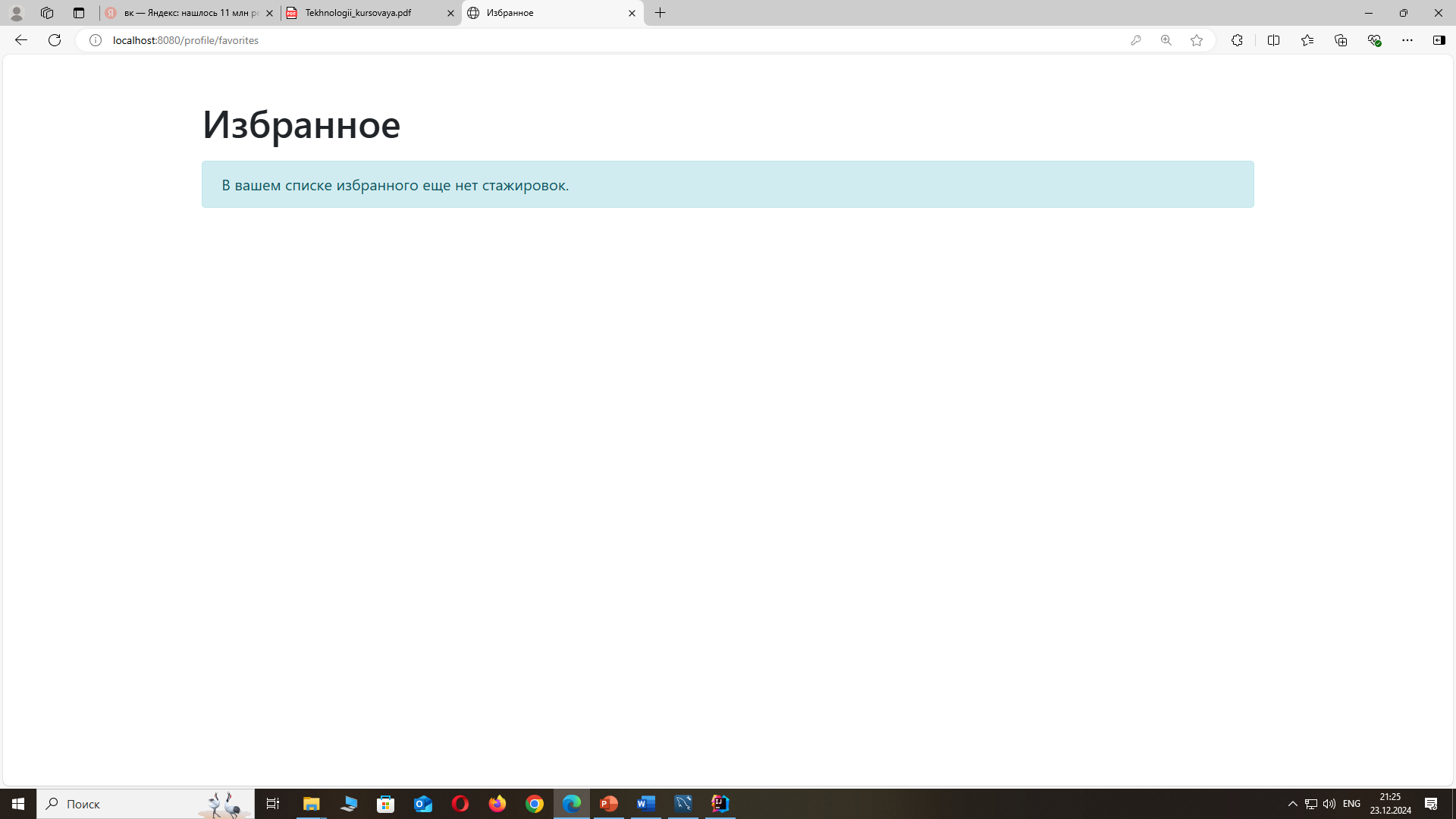


Рис. 3.5 Страница просмотра избранного