**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

курсовая работа

**по дисциплине «Введение в нереляционные базы данных»**

Тема: Сервис портфолио для программистов

|  |  |  |
| --- | --- | --- |
| Студент гр. 9303 |  | Дюков В.А. |
| Студент гр. 9303 |  | Лойконен М.Р. |
| Преподаватель |  | Заславский М.М. |

Санкт-Петербург

2022

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студенты  Дюков В.А.  Лойконен М.Р. | | |
| Группа 9303 | | |
| Тема работы: сервис портфолио для программистов | | |
| Исходные данные:  Необходимо реализовать клиентскую и серверную части для сервиса портфолио для программистов с использованием СУБД MongoDb. | | |
| Содержание пояснительной записки:  «Содержание», «Введение», «Качественные требования к решению», «Сценарии использования», «Модель данных», «Разработанное приложение», «Выводы», «Список использованных источников», «Приложения» | | |
| Предполагаемый объем пояснительной записки:  Не менее 15 страниц. | | |
| Дата выдачи задания: 08.09.2022 | | |
| Дата сдачи реферата: 26.12.2022 | | |
| Дата защиты реферата: 26.12.2022 | | |
| Студент |  | Дюков В.А. |
| Студент |  | Лойконен М.Р. |
| Преподаватель |  | Заславский М.М. |

**Аннотация**

В данной работе была разработана система портфолио для программистов. При разработке использовались следующие технологии: основной язык программирования – TypeScript, клиентская часть – React, SCSS, серверная часть – NodeJS, Express, используемая СУБД – MongoDB. Для развертывания системы использовались Docker и Docker-compose. В результате работы было разработано приложение, позволяющее пользователям авторизовываться и просматривать профили других пользователей с помощью системы поиска с фильтрацией результатов.

**Summary**

In this work, a portfolio system for programmers was developed. The following technologies were used in the development: the main programming language is TypeScript, the client part is React, SCSS, the server part is NodeJS, Express, the DBMS used is MongoDB. Docker and Docker-compose were used to deploy the system. As a result of the work, an application was developed that allows users to log in and view the profiles of other users using a search engine with filtered results.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 6 |
| 1. | Качественные требования к решению | 7 |
| 2. | Сценарии использования | 8 |
| 2.1. | Макет UI | 8 |
| 2.2. | Сценарии использования по задачам | 8 |
| 3. | Модель данных | 15 |
| 3.1. | Нереляционная модель данных | 15 |
| 3.2. | Реляционная модель данных | 24 |
| 3.3. | Сравнение моделей | 33 |
| 4. | Разработанное приложение | 34 |
| 4.1 | Краткое описание | 34 |
| 4.2 | Используемые технологии | 34 |
| 4.3 | Ссылки на Приложение | 34 |
|  | Заключение | 35 |
|  | Список использованных источников | 36 |
|  | Приложение А. Документация по сборке и развертыванию приложения | 37 |
|  | Приложение Б. Снимки экранов приложения | 38 |

**введение**

В настоящее время IT-специалистам очень важно при устройстве на работу иметь портфолио, в котором будет содержаться информация о навыках и проектах, в которых работал специалист. Поэтому было бы очень удобно, если специалист будет хранить всю эту информацию в едином месте.

Цель работы – разработка сервиса портфолио для программистов.

**1. качественные требования к решению**

Требуется разработать клиент-серверное приложение, использующее для хранения данных СУБД MongoDB.

**2. сценарии использования**

**2.1. Макет UI**

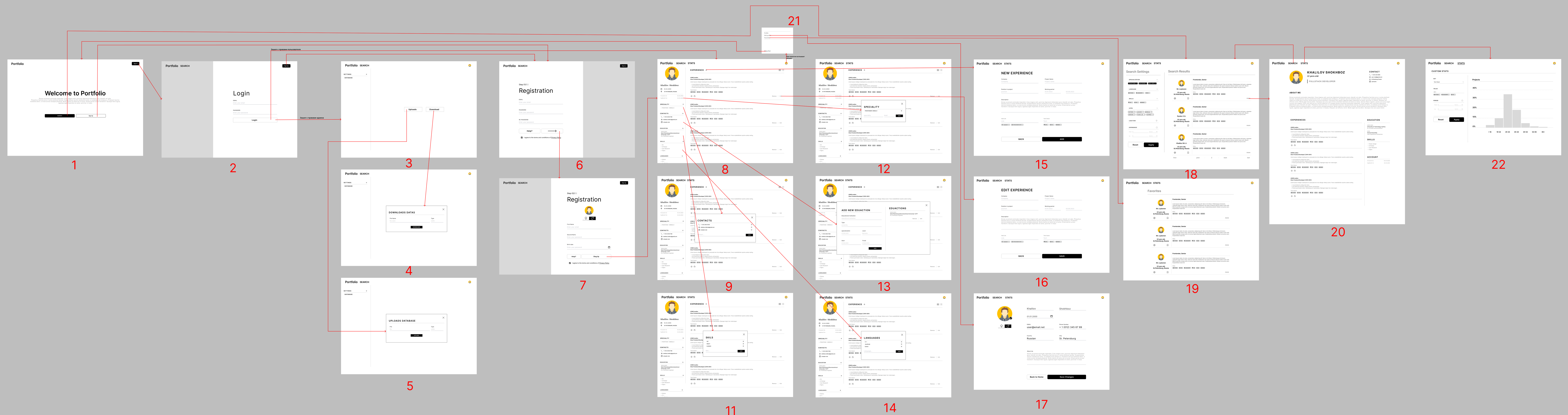
****

Рисунок 1 - Макет пользовательского интерфейса

**2.2. Сценарии использования по задачам**

### I. Сценарий использования - “Авторизация”

Действующее лицо: Пользователь сервиса.

Начальные условия: Пользователь зашел на начальную страницу [1].

Основной сценарий:

1. Пользователь нажимает кнопку ‘Sign in’

2. Пользователь вводит данные, с которыми он регистрировался.

3. Пользователь нажимает кнопку ‘Login’.

4. Перевод пользователя на страницу [8].

Альтернативные сценарии:

Пункт 2.

* Уведомление о неправильном вводе данных, возврат к пункту 1.
* Пользователь авторизовался как администратор, перевод на страницу [3].

### 

### II. Сценарий использования - “Регистрация”

Действующее лицо: Пользователь сервиса.

Начальные условия: Пользователь зашел на начальную страницу [1].

Основной сценарий:

1. Пользователь нажимает кнопку ‘Sign Up’, происходит переход на страницу [6].

2. Пользователь вводит данные для регистрации.

3. Пользователь нажимает кнопку *‘Далее’*, происходит переход пользователя на страницу [7].

4. Пользователь вводит данные профиля (заполнение полей, кнопки с выбором).

5. Пользователь нажимает кнопку ‘Sign Up’, происходит переход пользователя на страницу [8], регистрация успешна.

Альтернативные сценарии:

Пункт 2.

* Уведомление о уже занятом email, возврат к пункту 1.
* Уведомление о неправильном вводе данных, возврат к пункту 1.

Пункт 5.

* Уведомление о неправильном вводе данных, возврат к пункту 1.

### 

### III. Сценарий использования - “Редактирование страницы профиля”

Действующее лицо: Пользователь сервиса.

Начальные условия: Пользователь находится на странице [8].

Основной сценарий:

1. Пользователь нажимает одну из кнопок ‘+’ в сегментах ‘CONTACS’, ‘EDUCATION’, ‘SKILLS’, ‘LANGUAGES’, ‘SPECIALITY’.

2. Открывается модульное окно (МО), соответственно [9], [11], [12], [13] или [14].

3. Пользователь вводит необходимые данные в МО.

4. Пользователь нажимает кнопку ‘ADD’ в МО.

5. Модульное окно закрывается, новые данные отображаются на странице.

Альтернативный сценарий:

Пункт 2.

1. Пользователь нажимает ‘X’ рядом с элементом списка в МО, элемент списка удаляется.

2. Пользователь нажимает ‘X’ сверху слева в МО - кнопка выхода.

3. Модульное окно закрывается, данные о пользователе изменены.

### IV. Сценарий использования - “Редактирование основной информации о пользователе”

Действующее лицо: Пользователь сервиса.

Основной сценарий:

1. Пользователь нажимает иконку в правом верхнем углу экрана

2. В появившемся окне нажимает пункт ‘Edit profile’, происходит переход на страницу [17].

3. Пользователь изменяет данные профиля (заполнение полей, кнопки с выбором).

4. Пользователь нажимает кнопку ‘Save Changes’.

5. Перевод пользователя на окно [8], сохранение измененных данных.

Альтернативный сценарий:

Пункт 4:

1. Пользователь нажимает кнопку ‘Back to Home’.

2. Перевод пользователя на окно [8], измененные данные профиля не применены.

### V. Сценарий использования - “Добавление опыта работы”

Действующее лицо: Пользователь сервиса.

Начальные условия: Пользователь находится на странице [8].

Основной сценарий:

1. Пользователь нажимает кнопку ‘+’ рядом с надписью Experience, происходит переход на страницу [15].

2. Пользователь заполняет данные об опыте работы (заполнение полей).

3. Пользователь нажимает кнопку ‘ADD’.

4. Перевод пользователя на окно [8], добавление нового опыта работы в профиле.

Альтернативный сценарий:

Пункт 3:

1. Пользователь нажимает кнопку ‘BACK’.

2. Перевод пользователя на окно [8], новый опыт работы не создан.

### VI. Сценарий использования - “Редактирование опыта работы”

Действующее лицо: Пользователь сервиса.

Начальные условия: Пользователь находится на странице [8].

Основной сценарий:

1. Пользователь нажимает кнопку ‘edit’ у соответствующей записи в списке, происходит переход на страницу [16].

2. Пользователь редактирует данные об опыте работы (заполнение полей).

3. Пользователь нажимает кнопку ‘SAVE’.

4. Перевод пользователя на окно [8], изменение в существующем опыте работы сохранены.

Альтернативный сценарий:

Пункт 1:

* Пользователь нажимает кнопку ‘remove’, запись удаляется

Пункт 3:

1. Пользователь нажимает кнопку ‘BACK’.

2. Перевод пользователя на окно [8], изменения в опыте работы не применены.

### VII. Сценарий использования - “Поиск”

Действующее лицо: Пользователь сервиса.

Начальные условия: Пользователь находится на странице [18].

Основной сценарий:

1. Пользователь устанавливает характеристики для поиска соответствующих им пользователей (выбор из списка, заполнение полей).

2. Пользователь исключает часть выбранных характеристик нажатием ‘X’ около них.

3. Пользователь нажимает кнопку ‘Apply’

4. Производится автоматическая фильтрация и вывод подходящих результатов в ‘Search Results’.

Альтернативный сценарий (только авторизованный пользователь):

Пункт 3.

* Пользователь нажимает кнопку ‘Reset’, данные поиска сбрасываются

Пункт 4.

* Пользователь нажимает кнопку добавления в закладки у одного из пользователей, пользователь добавляется в избранные. Возврат к пункту 3.

1. Пользователь нажимает кнопку ‘Portfolio’.
2. Перевод пользователя на страницу [8].

### VIII. Сценарий использования - “Просмотр избранных”

Действующее лицо: Пользователь сервиса.

Начальные условия: Пользователь находится на странице [19].

Основной сценарий:

1. Пользователь просматривает избранных пользователей.

2. Пользователь нажимает кнопку ‘more’ у одного из пользователей.

3. Перевод пользователя на страницу [20] (просмотр профиля другого пользователя).

Альтернативный сценарий:

Пункт 1.

* Пользователь нажимает кнопку ‘закладки’ у одного из пользователей, выбранный пользователь удаляется из избранных. Возврат к пункту 1.

### IX. Сценарий использования - “Просмотр профиля другого пользователя”

Действующее лицо: Пользователь сервиса.

Начальные условия: Пользователь находится на странице [18].

Основной сценарий:

1. Пользователь нажимает кнопку ‘more’ у заинтересовавшего его пользователя, происходит переход на страницу [20].

2. Пользователь просматривает профиль пользователя.

### X. Сценарий использования - “Массовый импорт/экспорт данных”

Действующее лицо: Администратор сервиса.

Начальные условия: Администратор находится на странице [3].

Основной сценарий:

1. Администратор нажимает кнопку ‘Upload’.

2. Открывается модульное окно (МО).

3. (В МО) Администратор вводит путь к файлу с данными о пользователях и его тип.

4. (В МО) Администратор нажимает кнопку ‘UPLOAD’.

5. Модульное окно закрывается, данные из файла загружены в БД.

Альтернативный сценарий:

1. Администратор нажимает кнопку ‘Download’.
2. Открывается модульное окно (МО).
3. (В МО) Администратор вводит название файла с данными о пользователях и его тип.
4. (В МО) Администратор нажимает кнопку ‘DOWNLOAD’.
5. Модульное окно закрывается, данные из БД скачиваются в виде файла.

### XI. Сценарий использования - “Просмотр статистики”

Действующее лицо: Пользователь сервиса.

Начальные условия: Пользователь находится на странице [22].

Основной сценарий:

1. Пользователь выбирает интересующие его параметры.

2. Пользователь нажимает кнопку ‘Apply’.

3. Выводится статистика в виде диаграммы.

Альтернативный сценарий:

Пункт 2.

* Пользователь нажимает кнопку ‘Reset’, данные для отбора статистики сбрасываются.

Для данного решения преобладать будут операции записи, так как основной функционал заключается в добавлении различной информации пользователем (добавление информации о проектах, изученных языков программирования, навыков, образования).

**3. модель данных**

**3.1. Нереляционная модель данных**

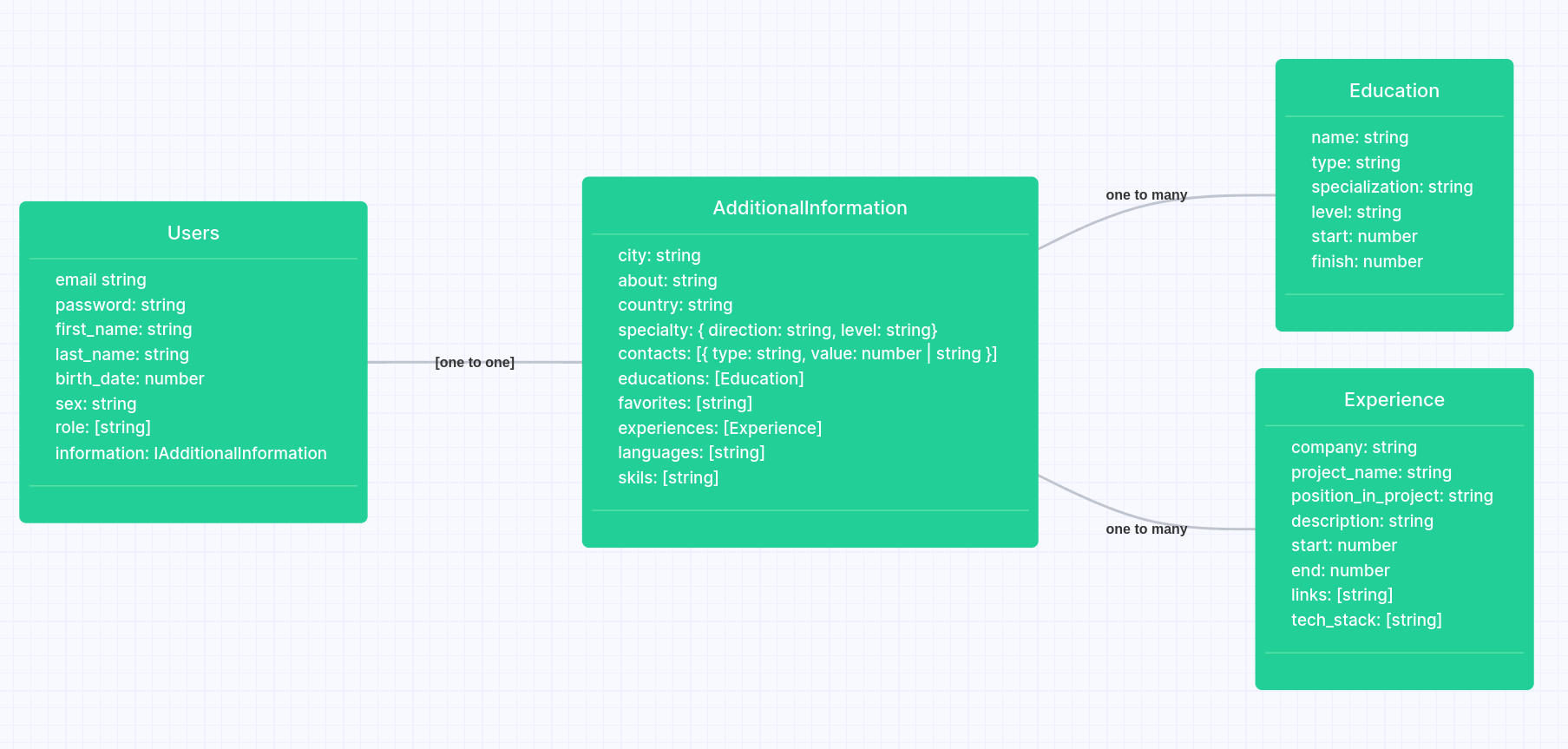


Рисунок 2 - Нереляционная модель данных

## Коллекция Users:

User {  
 email: string // почта пользователя   
 password: string // хэш пароля   
 first\_name: string // имя  
 last\_name: string // фамилия  
 birth\_date: number // дата рождения  
 sex: string // пол  
 role: string[] // роль пользователя  
 information: IAdditionalInformation // ссылка на дополнительную информацию  
}

Фактический размер сущности:

Чистый размер сущности:

## 

## Коллекция AdditionalInformation:

AdditionalInformation { //  
 city: string // город проживания   
 about: string // информация "о себе"  
 country: string // страна проживания  
 specialties: Specialty[] // специализации пользователя  
 contacts: Contacts[] // список контактов  
 educations: Education[] // список образований  
 favorites: string[] // список избранных пользователей  
 experiences: IExperience[] // список проектов (опыт работы)  
 languages: string[] // список языков  
 skills: string[] // список навыков  
}

// sub-document  
Specialty {  
 direction: string // направление специальности  
 level: string // уровень специальности  
}  
  
// sub-document  
Contacts {  
 type: string // тип контакта (телефон, почта, ...)  
 value: string // контакт пользователя  
}

Фактический размер сущности:

Чистый размер сущности:

## Коллекция Experience:

Experience { //  
 company: string // название компании, в которой  
 // работал пользователь (если это не   
 // личный проект)  
 project\_name: string // название проекта  
 position\_in\_project: string // должность пользователя в проекте  
 description: string // описание проекта  
 start: number // дата начала работы над проектом   
 end: number // дата конца работы над проектом   
 links: string[] // ссылки на проект  
 tech\_stack: string[] // технологический стек проекта  
}

Фактический размер сущности:

Чистый размер сущности:

## Коллекция Education:

Education {   
 name: string // название учебного заведения  
 type: string // тип учебного заведения  
 specialization: string // специальность  
 level: string // уровень  
 start: number // дата начало обучения  
 finish: number // дата окончания обучения  
}

Фактический размер сущности:

Чистый размер сущности:

## 

## Оценка удельного объема информации, хранимой в модели

- число пользователей

- число дополнительной информации

- число проектов

- число образований

Зависимость параметров от числа пользователей:

Для расчета будем использовать средние значения переменных:

### Чистый объем

### Фактический объем

## Избыточность модели

## Направление роста модели при увеличении количества объектов каждой сущности

Рост модели линейный.

## Запросы к модели, с помощью которых реализуются сценарии использования

* 1. Поиск людей, у которых в списке языков программирования указан ‘C++’:

await UsersModel.aggregate([  
 {  
 $lookup: {  
 from: "additionalinformations",  
 localField: "information",  
 foreignField: "\_id",  
 as: "information",  
 },  
 },  
 {  
 $match: { "information.languages": { $in: ["C++"] } }  
 },  
 ])

Ответ:

[  
 {  
 "\_id": "6366af384567ca6adf27b838",  
 "email": "test@email.com",  
 "password": "123456789",  
 "first\_name": "иван",  
 "last\_name": "иванов",  
 "birth\_date": 21546464,  
 "sex": "MALE",  
 "role": [  
 "USER"  
 ],  
 "information": [  
 {  
 "\_id": "6366b3964567ca6adf27b841",  
 "country": "Russia",  
 "city": "St.Petersburg",  
 "specialties": [  
 {  
 "direction": "Backend",  
 "level": "senior"  
 }  
 ],  
 "contacts": [  
 {  
 "type": "PHONE",  
 "value": "012 345 67 89"  
 },  
 {  
 "type": "TELEGRAM",  
 "value": "012 345 67 89"  
 }  
 ],  
 "about": "bla bla-bla",  
 "experiences": [  
 "6366b6454567ca6adf27b84b",  
 "6366b6674567ca6adf27b84c"  
 ],  
 "education": [  
 "6366b45d4567ca6adf27b848"  
 ],  
 "languages": [  
 "JavaScript",  
 "TypeScript",  
 "Python",  
 "C++",  
 "Java"  
 ],  
 "skills": [  
 "Git",  
 "express",  
 "mongodb",  
 "MySql",  
 "Nestjs",  
 "CI/CD",  
 "docker",  
 "kubernetes"  
 ]  
 }  
 ]  
 }  
]

Количество запросов: 1

Задействовано коллекций: 2

1. Просмотр статистики - количество человек, обладающих навыком ‘Git’:

await UsersModel.aggregate([  
 {  
 $lookup: {  
 from: "additionalinformations",  
 localField: "information",  
 foreignField: "\_id",  
 as: "information",  
 },  
 },  
 {  
 $match: {  
 "information.skills": { $in: ["Git"] },  
 },  
 },  
 {  
 $count: "count",  
 },  
 ])

Ответ:

[  
 {  
 "count": 2  
 }  
]

Количество запросов: 1

Задействовано коллекций: 2

1. Добавление нового опыта работы

const user = await UsersModel.findOne({  
 email: "test@email.com",  
}).exec();  
  
await AdditionalInformationModel.findById(user.information);  
  
const experience = new ExperiencesModel({  
 company: "mail.ru",  
 project\_name: "аля супер проект",  
 position\_in\_project: "junior frontend developer",  
 start: new Date("01.01.2020").getTime(),  
 end: new Date("01.01.2021").getTime(),  
 description: "...",  
 links: [],  
 tech\_stack: ["React", "TypeScript", "Mongodb", "docker"],  
});  
  
await AdditionalInformationModel.findByIdAndUpdate(user.information, {  
 $push: { experiences: experience.\_id },  
}).exec();

Ответ:

{  
 "company": "mail.ru",  
 "project\_name": "аля супер проект",  
 "position\_in\_project": "junior frontend developer",  
 "description": "...",  
 "start": 1577826000000,  
 "end": 1609448400000,  
 "links": [],  
 "tech\_stack": [  
 "React",  
 "TypeScript",  
 "Mongodb",  
 "docker"  
 ],  
 "\_id": "6366d4c7a54b9069d81fc42d",  
 "\_\_v": 0  
}

Количество запросов: 4

Задействовано коллекций: 2

## Примеры хранения записей для одного пользователя

Коллекция User:



Рисунок 3 - Пример хранения User

Коллекция AdditionalInformation:

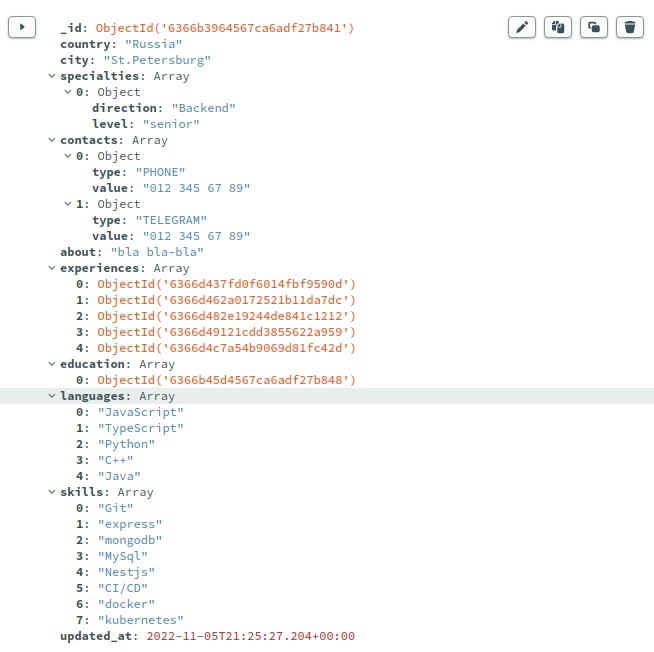


Рисунок 4 - Пример хранения AdditionalInformation

Коллекция Education:



Рисунок 5 - Пример хранения Education

Коллекция Experience:



Рисунок 6 - Пример хранения Experience

**3.2. Реляционная модель данных**

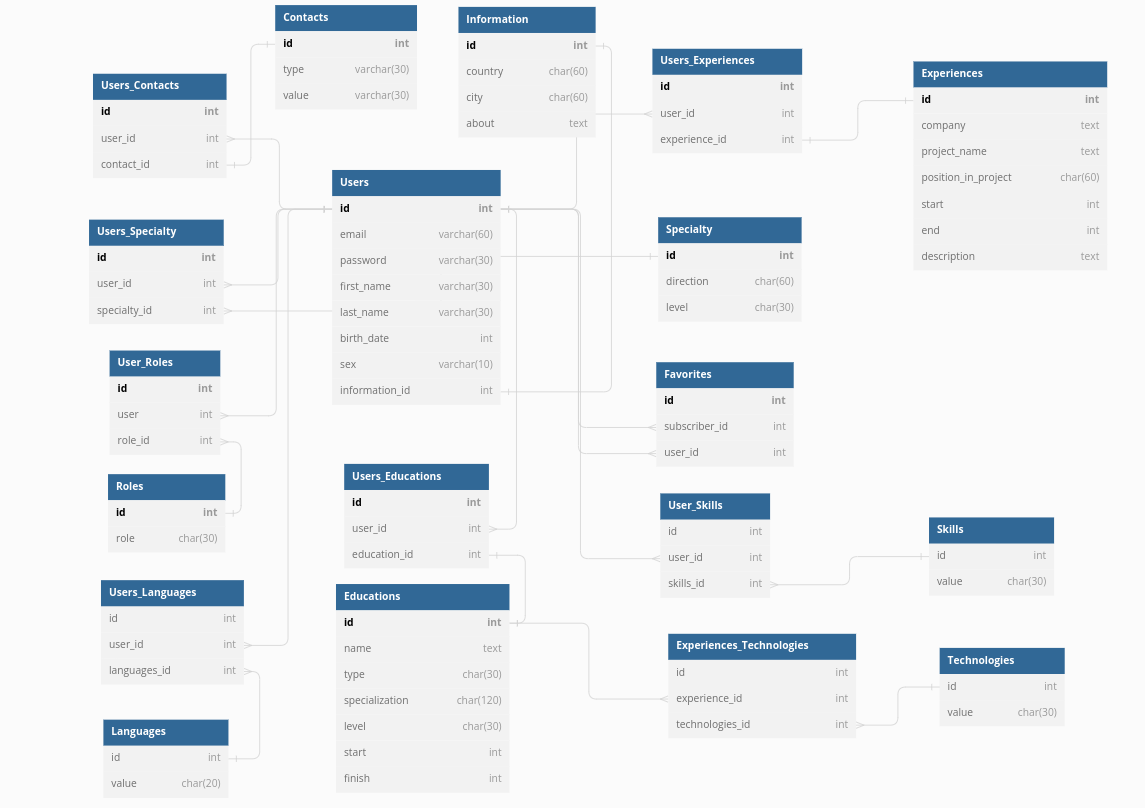


Рисунок 7 - Реляционная модель данных

## Users:

Table Users {  
 id int /\* \*/  
 email varchar(60)   
 password varchar(30)   
 first\_name varchar(30)   
 last\_name varchar(30)  
 birth\_date int  
 sex varchar(10)  
 information\_id int  
}

Фактический размер:

4 + (2 + 60) + (2 + 30) + (2 + 30) + (2 + 30) + 4 + (2 + 10) + 4 = 182Б

Чистый размер:

(2 + 60) + (2 + 30) + (2 + 30) + (2 + 30) + 4 + (2 + 10) = 174Б

## Users\_Contacts

Table Users\_Contacts {  
 id int   
 user\_id int   
 contact\_id int   
}

Фактический размер:

4 + 4 + 4 = 12Б

## Contacts

Table Contacts {  
 id int   
 type varchar(30)  
 value varchar(30)  
}

Фактический размер:

4 + 2 + 30 + 2 + 30 = 68Б

Чистый размер:

2 + 30 + 2 + 30 = 64Б

## Information

Table Information  
{  
 id int   
 country char(60)  
 city char(60)  
 about text  
}

Фактический размер:

4 + 60 + 60 + (255 + 4) = 383Б

Чистый размер:

60 + 60 + (255 + 4) = 379Б

## Experiences

Table Experiences  
{  
 id int   
 company text  
 project\_name text  
 position\_in\_project char(60)  
 start int  
 end int  
 description text  
}

Фактический размер:

4 + (255 + 4)\*3 + 60 + 4 + 4 = 849Б

Чистый размер:

845Б

## Users\_Experiences

Table Users\_Experiences   
{   
 id int   
 user\_id int   
 experience\_id int   
}

Фактический размер:

4 + 4 + 4 = 12Б

## Specialty

Table Specialty  
{  
 id int   
 direction char(60)  
 level char(30)  
}

Фактический размер:

30 + 60 + 4 = 94Б

Чистый размер:

90Б

## Users\_Specialty

Table Users\_Specialty   
{   
 id int   
 user\_id int   
 specialty\_id int   
}

Фактический размер:

4 + 4 + 4 = 12Б

## Educations

Table Educations  
{  
 id int   
 name text  
 type char(30)  
 specialization char(120)   
 level char(30)  
 start int  
 finish int  
}

Фактический размер:

4 + (4 + 255) + 30 + 120 + 30 + 4 + 4 = 451Б

Чистый размер:

448Б

## Users\_Educations

Table Users\_Educations  
{  
 id int   
 user\_id int   
 education\_id int   
}

Фактический размер:

4 + 4 + 4 = 12Б

## Favorites

Table Favorites  
{  
 id int   
 subscriber\_id int   
 user\_id int   
}

Фактический размер:

4 + 4 + 4 = 12Б

## Roles

Table Roles  
{  
 id int   
 role char(30)  
}

Фактический размер:

4 + 30 = 34Б

Чистый размер:

30Б

## User\_Roles

Table User\_Roles  
{  
 id int   
 user int   
 role\_id int  
}

Фактический размер:

4 + 4 + 4 = 12Б

## Skills

Table Skills {  
 id int  
 value char(30)  
}

Фактический размер:

4 + 30 = 34Б

Чистый размер:

30Б

## User\_Skills

Table User\_Skills {  
 id int  
 user\_id int   
 skills\_id int   
}

Фактический размер:

4 + 4 + 4 = 12Б

## Languages

Table Languages {  
 id int  
 value char(20)  
}

Фактический размер:

4 + 20 = 24Б

Чистый размер:

20Б

## User\_Languages

Table User\_Languages {  
 id int  
 user\_id int  
 language\_id int  
}

Фактический размер:

4 + 4 + 4 = 12Б

## Experiences\_Technologies

Table Experiences\_Technologies {  
 id int  
 experience\_id int   
 technologies\_id int  
}

Фактический размер:

4 + 4 + 4 = 12Б

## Technologies

Table Technologies {  
 id int  
 value char(30)  
}

Фактический размер:

4 + 30 = 34Б

Чистый размер:

30Б

## 

## Оценка удельного объема информации, хранимой в модели

- число пользователей

- число дополнительной информации

- число проектов пользователя

- число образований пользователя

- число контактов

- число специальностей пользователя

- число избранных пользователей

- число ролей пользователя

- число языков пользователя

- число навыков пользователя

- число технологий в проекте

Зависимость параметров от числа пользователей:

Следующие параметры не зависят от числа пользователей:

- количество ролей

- количество языков

- количество технологий

- количество навыков

### Чистый объем

### Фактический объем

## Избыточность модели

## Направление роста модели при увеличении количества объектов каждой сущности

Рост модели линейный

## Запросы к модели, с помощью которых реализуются сценарии использования

* 1. Поиск людей, у которых в списке языков программирования указан ‘C++’

SELECT   
 \*   
FROM   
 users   
INNER JOIN   
 users\_information  
 ON users.information = users\_information.information\_id  
 INNER JOIN   
 users\_languages  
 ON users\_languages.user\_id = user.id  
 INNER JOIN   
 languages  
 ON users\_languages.languages\_id = languages.id  
WHERE languages.value like "C++";

Количество запросов: 1

Задействовано таблиц: 4

* 1. Просмотр статистики - количество человек, обладающих навыком ‘Git’

SELECT   
 count(\*)   
FROM   
 users   
INNER JOIN   
 users\_skills  
 ON users\_skills.user\_id = user.id  
 INNER JOIN   
 skills  
 ON users\_skills.skills\_id = skills.id  
WHERE skills.value like "Git";

Количество запросов: 1

Задействовано таблиц: 3

* 1. Добавление нового опыта работы

DECLARE @react\_id INT;  
DECLARE @typescript\_id INT;  
DECLARE @mongodb\_id INT;  
DECLARE @docker\_id INT;  
DECLARE @experience\_id INT;  
DECLARE @user\_id INT;  
  
SELECT @user\_id = (SELECT id FROM Users WHERE mail LIKE "test@email.com")  
SELECT @react\_id = (SELECT id FROM Technologies WHERE Technologies.value LIKE "React");  
SELECT @typescript\_id = (SELECT id FROM Technologies WHERE Technologies.value LIKE "TypeScript");  
SELECT @mongodb\_id = (SELECT id FROM Technologies WHERE Technologies.value LIKE "Mongodb");  
SELECT @docker\_id = (SELECT id FROM Technologies WHERE Technologies.value LIKE "docker");  
INSERT INTO Experiences VALUES (  
 "mail.ru",  
 "аля супер проект",  
 "junior frontend developer",  
 892747298427,  
 891823789793,  
 "..."  
);  
  
SELECT @experience\_id = (SELECT LAST\_INSERT\_ID() FROM Experiences);  
  
INSERT INTO Experiences\_Technologies VALUES (  
 @experience\_id,  
 @react\_id  
);  
  
INSERT INTO Experiences\_Technologies VALUES (  
 @experience\_id,  
 @typescript\_id  
);  
  
INSERT INTO Experiences\_Technologies VALUES (  
 @experience\_id,  
 @mongodb\_id  
);  
  
INSERT INTO Experiences\_Technologies VALUES (  
 @experience\_id,  
 @docker\_id  
);  
  
INSERT INTO Users\_Experiences VALUES (  
 @user\_id,  
 @experience\_id  
);

Количество запросов: 12

Задействовано таблиц: 4

**3.3. Сравнение моделей**

Пусть число пользователей U = 1000.

Реляционная модель:

* Чистый объём:
* Фактический объём:
* Избыточность:

Нереляционная модель:

* Чистый объём:
* Фактический объём:
* Избыточность:

Сравнение показало, что NoSQL модель занимает примерно в 2 раза больше памяти, нежели SQL модель, это происходит из-за дублирования данных. Но при этом избыточность меньше. При запросах в реляционной моделе использовалось большее число операций и задействованных таблиц, что усложняет работу с данной моделью. Исходя из всего этого, можно сделать вывод, что для удобства работы лучше выбрать NoSQL модель.

**4. разработанное приложение**

**4.1. Краткое описание**

В системе предусмотрены 2 роли – пользователь (USER) и администратор (ADMIN). После регистрации человек получает роль USER.

Пользователь может осуществлять поиск других людей при помощи фильтрации по определенным тегам, а также просматривать их профили, чтобы увидеть больше информации.

**4.2. Используемые технологии**

Для серверной части используются следующие технологии:

* Язык программирования TypeScript
* NodeJS
* Express
* СУБД MongoDB

Для клиентской части используются следующие технологии:

* Язык программирования TypeScript
* React
* SCSS

Для сборки и развертывания приложения используются Docker и Docker-compose.

В приложении А представлена документация по сборке и развертывания приложения.

В приложении Б представлены снимки экранов приложения.

**4.3. Ссылки на Приложение**

Ссылка на репозиторий: <https://github.com/moevm/nosql2h22-it-profile>

**заключение**

В ходе выполнения работы был разработан прототип приложения сервиса портфолио для программистов. Приложение позволяет пользователям зарегистрироваться и просмотреть профили других пользователей, используя поиск с фильтрацией.

В дальнейшем планируется расширить функционал приложения, добавив отображения статистики, возможность добавлять и редактировать пользовательские данные через веб-интерфейс.

**список использованных источников**

1. Документация MongoDB: <https://www.mongodb.com/docs/>

**приложение А**

**Документация по сборке и развертыванию приложения**

* + 1. Скачать приложение из репозитория: <https://github.com/moevm/nosql2h22-it-profile>
    2. Выполнить в командной строке команду:

docker-compose -f ./docker-composes/prod.yaml up --build -d

* + 1. Перейти по адресу http://localhost:8080

**приложение Б**

**снимки экранов приложения**

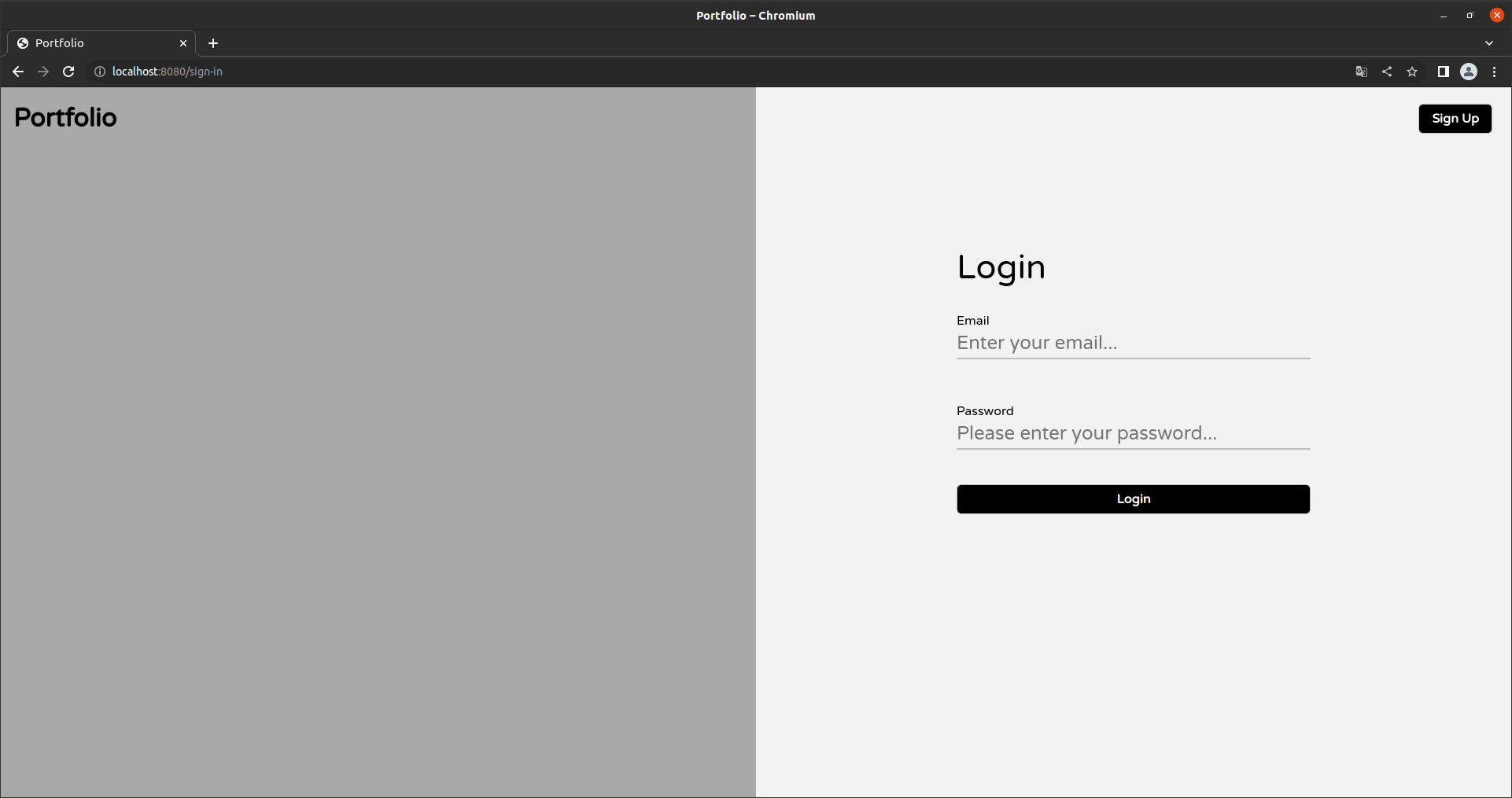


Рисунок 8 - Страница авторизации

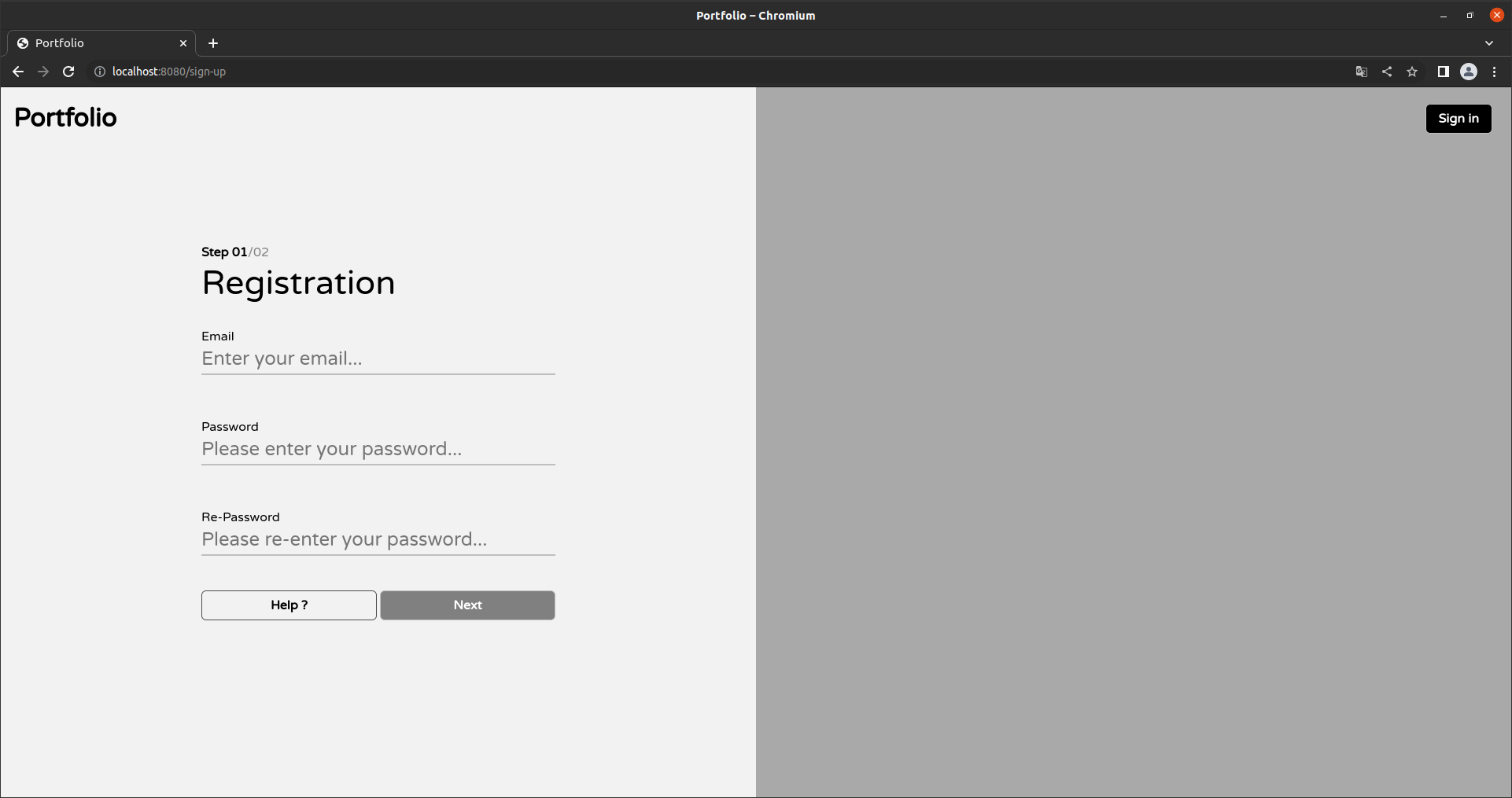


Рисунок 9 - Страница регистрации (1)

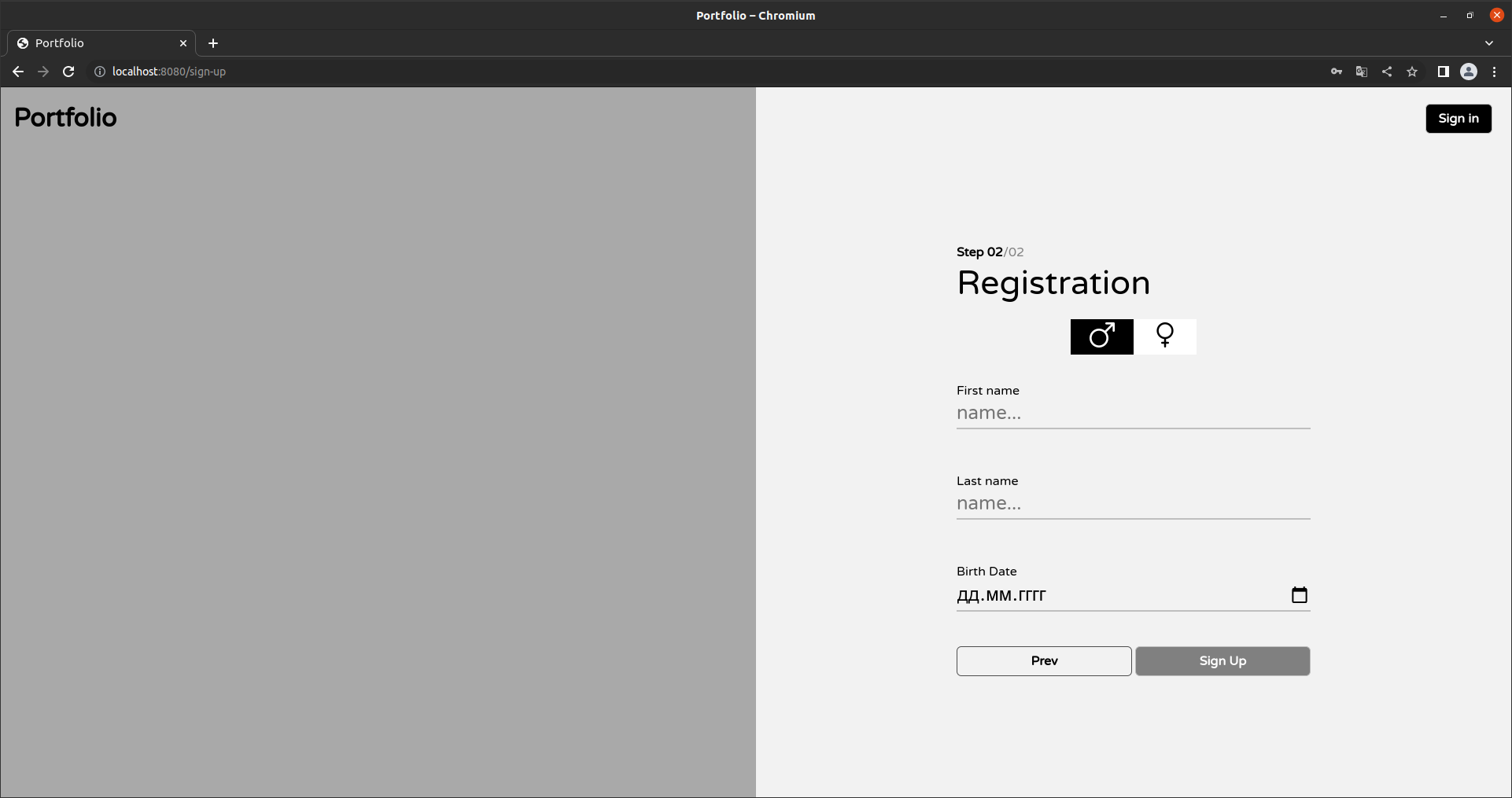


Рисунок 10 - Страница регистрации (2)

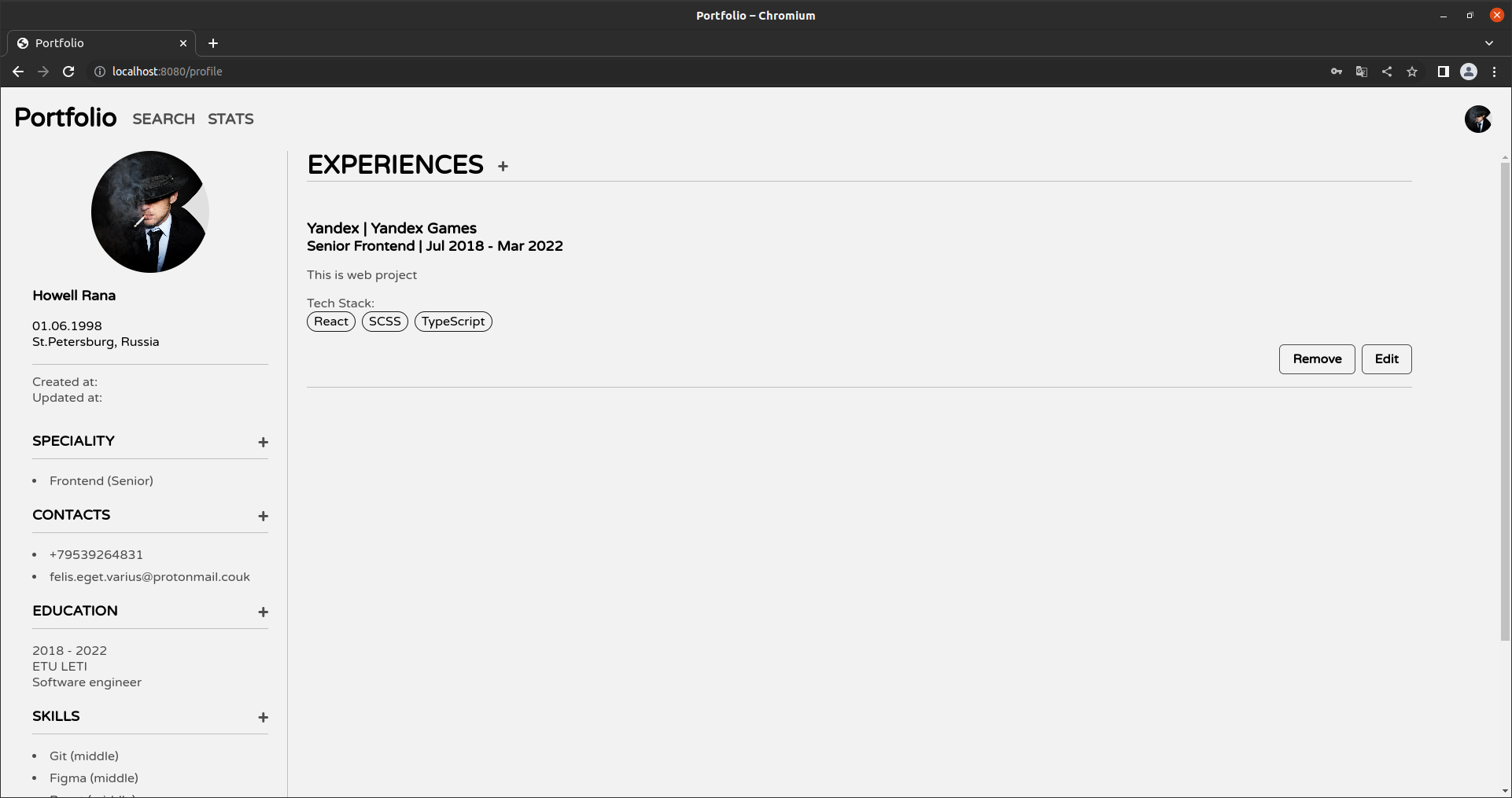


Рисунок 11 - Страница профиля

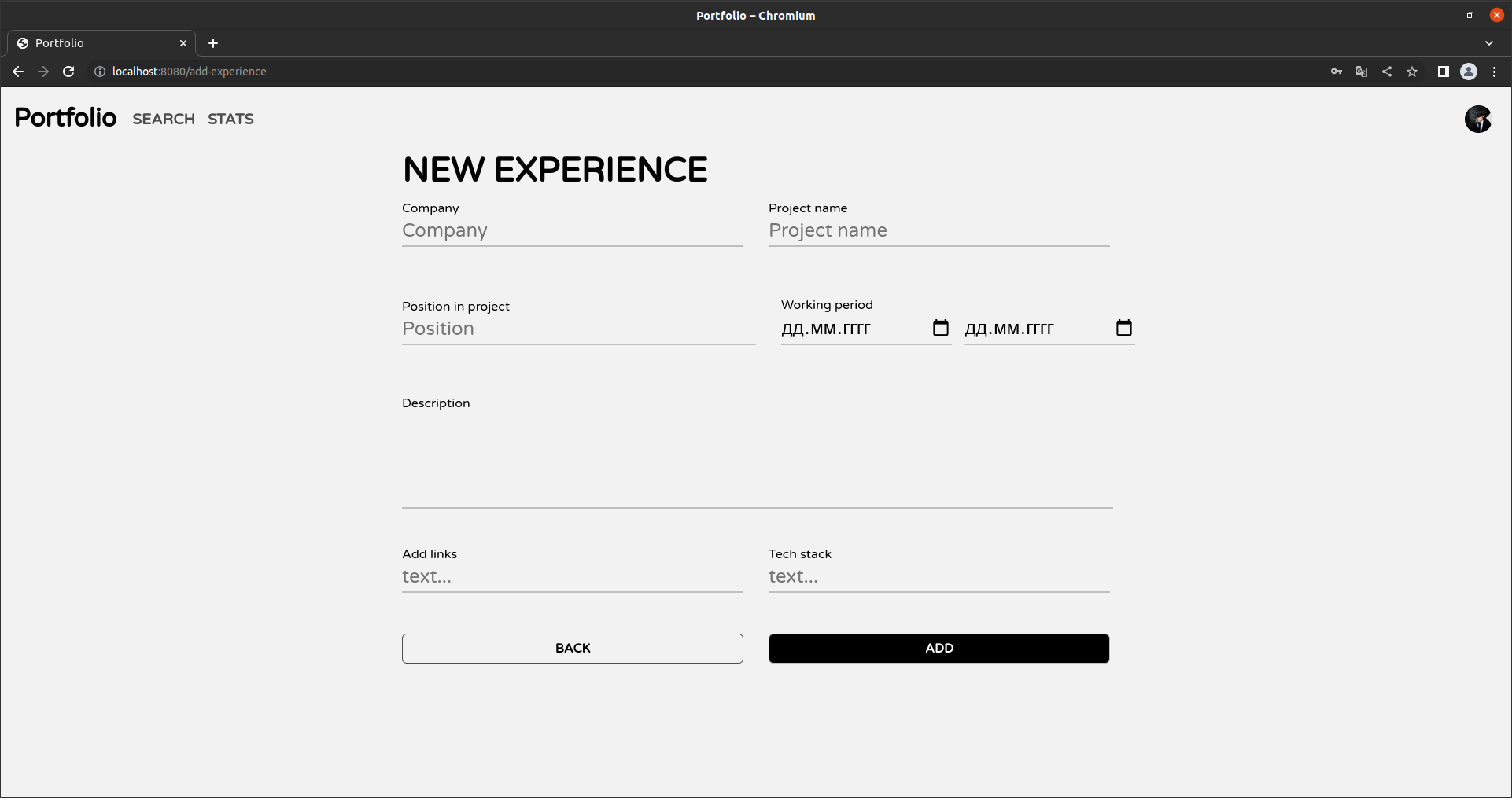


Рисунок 12 - Страница добавления нового проекта в портфолио

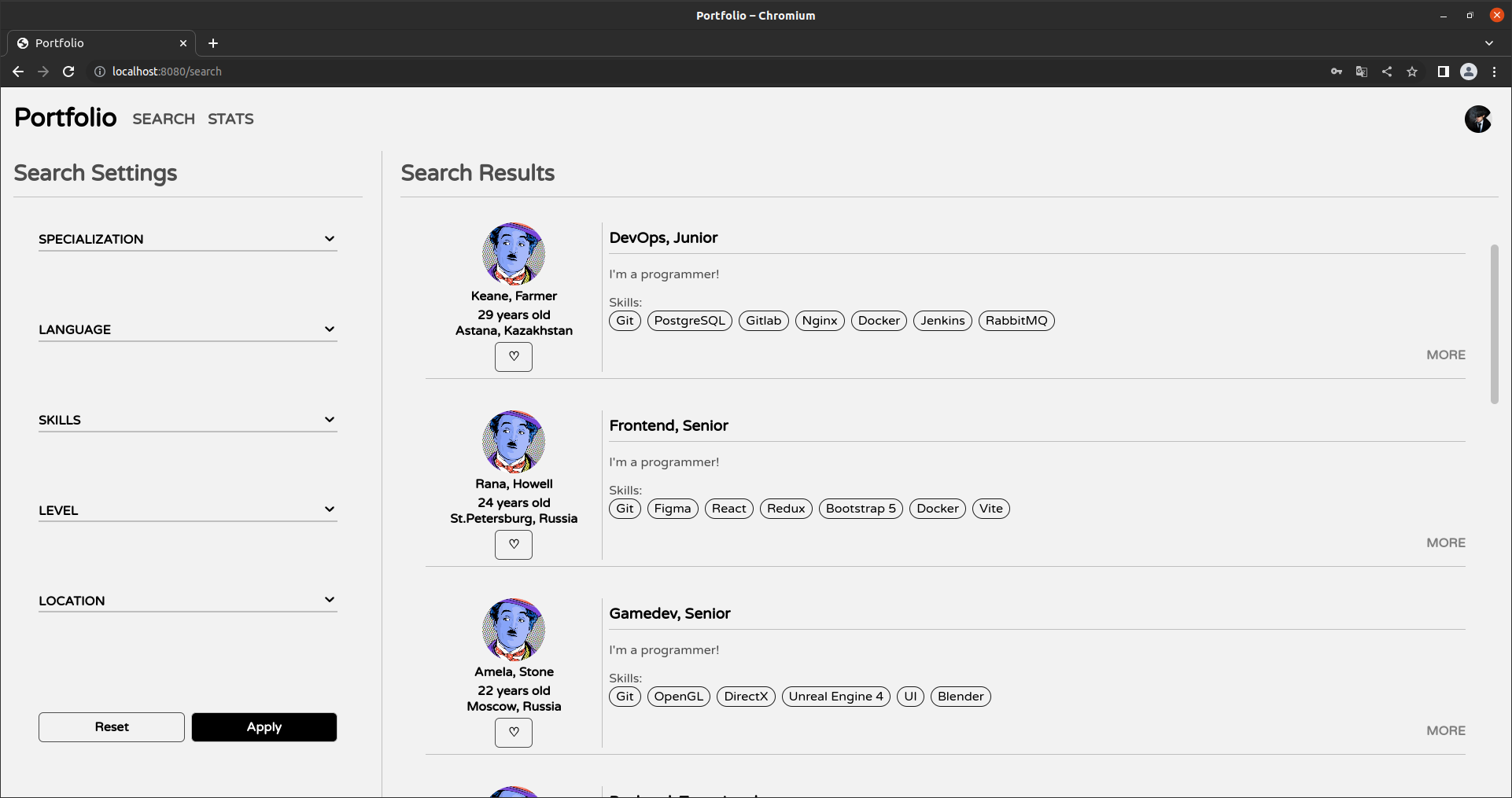


Рисунок 13 - Страница поиска

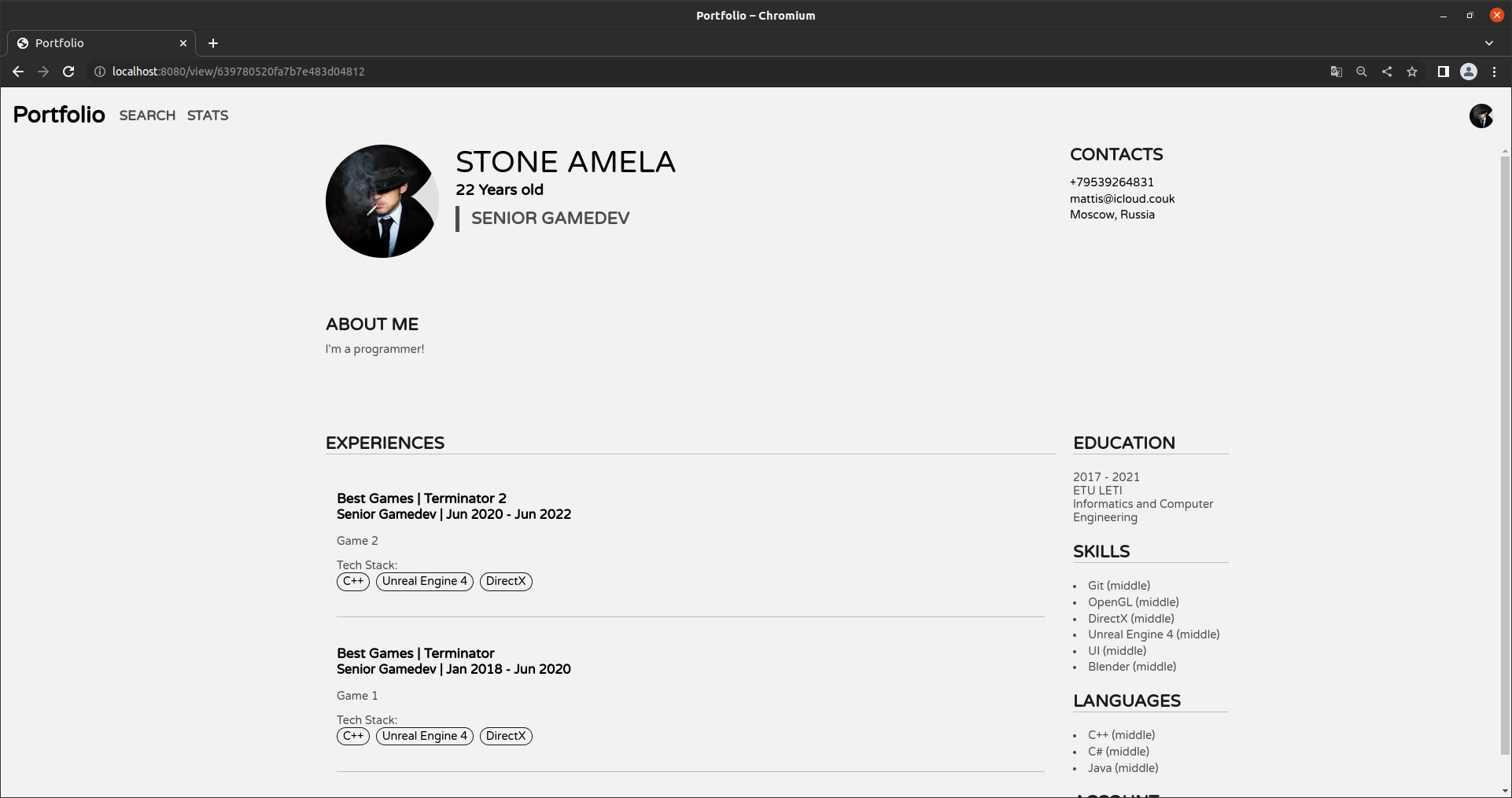


Рисунок 14 - Страница просмотра другого профиля

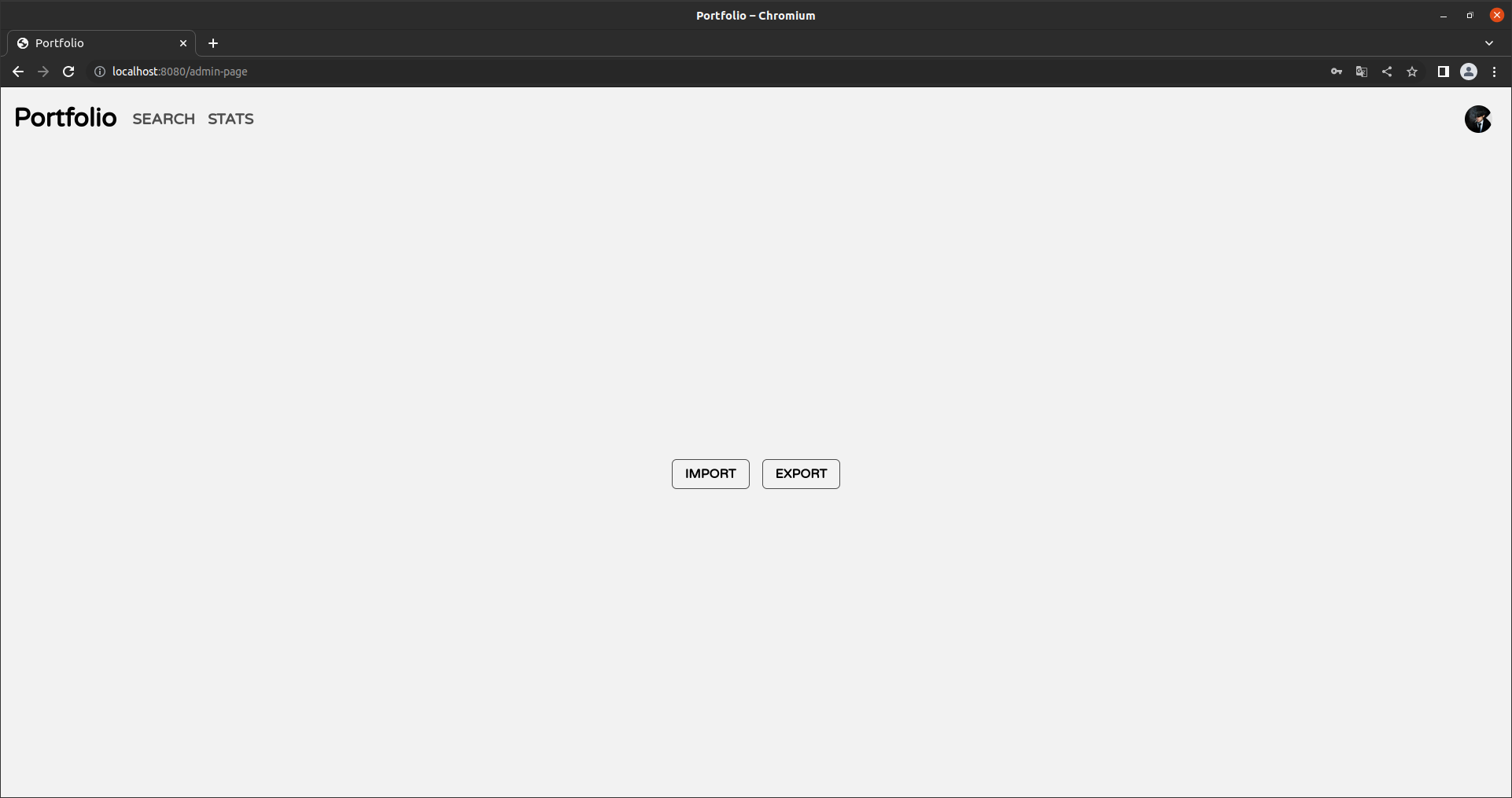


Рисунок 15 - Страница администратора (недоступна обычным пользователям)