

Министерство науки высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
(Университет ИТМО)

Факультет инфокоммуникационных технологий

Образовательная программа 45.03.04

Направление подготовки (специальность) Интеллектуальные системы в гуманитарной сфере

О Т Ч Е Т

По курсовой работе по дисциплине «Основы web-программирования»

Тема задания: Реализация web-сервисов средствами Django REST framework, Vue.js, Muse-UI

Обучающийся: Сальникова Надежда Константиновна, К3342

Руководитель: Говоров Антон Игоревич

Оценка за курсовую работу: _____

Подписи членов комиссии:

_____ / Говоров А.И.
(подпись) ФИО

_____ / Чунаев А.В.
(подпись) ФИО

_____ / Антонов М.Б.
(подпись) ФИО

Дата _____

Санкт-Петербург
2020

Оглавление

Введение	3
Глава 1. Описание предметной области и функциональных требований	4
1.1 Описание предметной области.....	4
1.2 Описание функциональных требований	4
Глава 2. Описание серверной части.....	6
2.1 Модель данных	6
2.2 Сериализации.....	7
2.3 Отображения	8
Глава 3. Описание клиентской части	11
3.1 Разработанные интерфейсы клиентской части.....	11
Глава 4. Контейнеризация и оркестрация	17
Заключение.....	19
Список литературы.....	20

Введение

Разработка и использование web-сервисов являются неотъемлемыми частями современных бизнес-процессов, так как подавляющее большинство организаций и людей используют компьютер и интернет. Подобные приложения позволяют создавать для пользователей удобные и функциональные инструменты для взаимодействия с удаленными серверами и решения прикладных задач. Например, с помощью веб-приложения администратор может просматривать и изменять данные в системе, то есть проводить эффективный менеджмент информации.

В современном формате web-сервисов обычно присутствуют клиентская и серверная часть. С первой взаимодействует непосредственно пользователь, поэтому клиентская часть должна обладать визуальным интерфейсом. Серверная часть обычно скрыта от пользователя, но именно она отвечает за принципиальную работу системы, а потому должна быть функциональна.

Есть множество подходов к разработке web-приложений и сервисов, а также языков на которых они пишутся. В данной работе была использована комбинация Django REST фреймворка (Python) с Vue.js, PostgreSQL и Muse-UI. Такая комбинация весьма эффективна и позволяет реализовать весь необходимый функционал.

Целью курсовой работы является разработка web-сервиса для заданной предметной области.

Для достижения этой цели необходимо выполнить следующие задачи:

1. Изучить предметную область
2. Выявить функциональные требования для web-сервиса
3. Разработать серверную и клиентскую части сервиса
4. Провести контейнеризацию и оркестрацию

Глава 1. Описание предметной области и функциональных требований

1.1 Описание предметной области

Создать программную систему, предназначенную для администратора альпинистского клуба. Альпинистский клуб организует восхождения в разных точках мира. Система должна обеспечить сохранение информации о хронике восхождений. Для каждого восхождения формируется группа. В состав группы могут входить альпинисты из других клубов. Поэтому нужно иметь информацию о каждом клубе (название, страна, город, контактное лицо, e-mail, телефон). Необходимо иметь описание маршрута и продолжительность восхождения. Необходимо обеспечить сохранение даты/времени начала и завершения каждого восхождения (планируемого и фактического), имен и адресов участвовавших в нем альпинистов, названия и высоты горы, страны и района, где эта гора расположена. После завершения восхождения фиксируется информация об успешности восхождения для каждого участника и группы в целом. При возникновении нештатных ситуаций необходимо указать для каждого участника, что случилось (травма, пропал без вести, летальный исход) и в пояснении о группе дать подробности.

Администратор должен иметь возможность:

- добавить сведения о новом альпинисте, новой вершине;
- изменить сведения об альпинистах и вершинах;
- формировать новые группы и вносить информацию после завершения восхождения группой.

1.2 Описание функциональных требований

Согласно варианту, существует некоторый альпинистский клуб, у администратора которого должна быть возможность просматривать, редактировать и удалять информацию. Логично, что такая возможность должна быть не у любого пользователя, поэтому необходима регистрация и авторизация на данном сервисе.

Также необходима возможность просмотреть данные в удобной форме, например в таблице. Добавить данные через сервис, например, через форму. Удалить или отредактировать их по какому-либо критерию.

Кроме того, нужен поиск определённой информации по выбранному критерию. Например, по названию горы необходимо определить какие восхождения, кем и когда на нее совершались.

Данные возможности представлены на use-case диаграмме (см. рисунок 1).



Рисунок 1 - use-case диаграмма

Глава 2. Описание серверной части

2.1 Модель данных

Согласно варианту и выявленным функциональным требованиям была спроектирована и реализована следующая модель данным, представленная на рисунке 2.

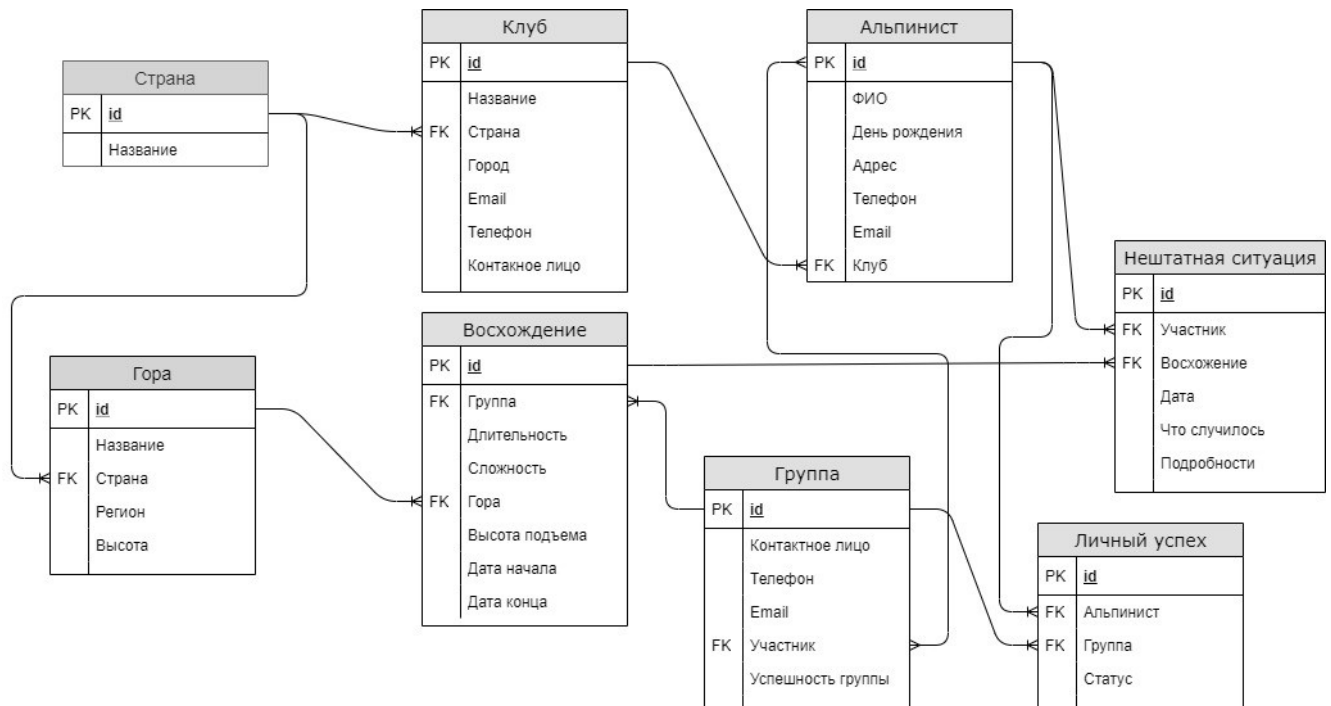


Рисунок 2 - модель данных

Данная модель содержит 8 сущностей, между которыми установлены связи один-ко-многим и многие-ко-многим.

Для управления этой базой данных была выбрана PostgreSQL – объектно-реляционная система управления базами данных [1].

Рассмотрим подробнее реализованные модели (таблицы), все они описаны в файле models.py:

1. Class Country – Страна – содержит уникальный идентификатор и название страны.
2. Class Mountain – Гора – содержит информацию о горах. Созданы поля идентификатора, названия, страны и региона, а также высоты. Страна является внешним ключом.

3. Class Club – Клуб – содержит в себе информацию о клубах. Созданы поля идентификатора, названия, страны (внешний ключ), города, телефона, адреса электронной почты и контактного лица.
4. Class Alpinist – Альпинист – содержит в себе информацию об альпинистах. Созданы поля идентификатора, имени, даты рождения, адреса, телефона и клуба (внешний ключ).
5. Class Group – Группа – содержит в себе информацию о группах. Созданы поля идентификатора, кода группы, участников (внешний ключ, многие-ко-многим), контактного лица, адреса электронной почты и телефона, а также успех группы.
6. Class Ascent – Восхождение – содержит в себе информацию о восхождениях. Созданы поля идентификатора, кода восхождения, группы (внешний ключ), горы (внешний ключ), длительности, сложности, высоты подъема, даты начала и даты конца.
7. Class IndSuccess – Личный успех – содержит в себе информацию о результатах участников групп. Созданы поля идентификатора, группы (внешний ключ), участника (внешний ключ) и статуса.
8. Class Emergency – Нештатная ситуация – содержит в себе информацию о происшествиях. Созданы поля идентификатора, восхождения (внешний ключ), участника (внешний ключ), типа происшествия и комментария.

Все данные модели были зарегистрированы в панели администратора, то есть администратор сервиса (суперпользователь) имеет доступ ко всей перечисленной информации.

2.2 Сериализации

Сериализации Django – это инструмент для перевода моделей в другие форматы. Обычно эти форматы основаны на тексте, но это бывает и иначе. Использование сериализаций необходимо для обмена данными между серверной (Django REST) и клиентской частью (Vue.js). Для большинства моделей было создано по 2 сериализации – для получения и отправления данных, что связано с

обработкой внешних ключей для вывода данных. Все сериализации были описаны с помощью стандартного синтаксиса Django REST Framework [2].

Пример сериализаций для одной из моделей приведен на рисунке 3.

```
class ClubSerializer(serializers.ModelSerializer):
    country = CountrySerializer()

    class Meta:
        model = Club
        fields = ('id', 'club_name', 'country', 'city', 'contact_person', 'email', 'telephone')

class ClubPostSerializer(serializers.ModelSerializer):
    class Meta:
        model = Club
        fields = ('id', 'club_name', 'country', 'city', 'contact_person', 'email', 'telephone')
```

Рисунок 3 - пример сериализации

2.3 Отображения

Для создания отображения данных были использованы встроенные инструменты Django REST, позволяющие сравнительно просто описывать отображения и обрабатывать url-адреса для них.

Все отображения были написаны на основе классов, поскольку такой метод сравнительно прост в реализации и эффективен в использовании [3]. В отображениях на основе классов были реализованы методы get, post, get_one, put и delete. С помощью этих методов происходит получение, добавление, редактирование и удаление данных. Метод get_one – вспомогательный, использован для работы других методов.

Пример отображения для одной из моделей приведен на рисунках 4 и 5.

```
class MountainView(APIView):
    permission_classes = [permissions.AllowAny, ]

    def get(self, request):
        mountains = Mountain.objects.all()
        serializer = MountainSerializer(mountains, many=True)
        return Response({"data": serializer.data})

    def post(self, request):
        mountain = MountainPostSerializer(data=request.data)
        if mountain.is_valid():
            mountain.save()
            return Response(status=201)
        else:
            return Response(status=400)

    def get_one(self, pk):
        try:
            return Mountain.objects.get(pk=pk)
        except Mountain.DoesNotExist:
            raise Http404
```

Рисунок 4 - пример отображения (часть 1)

```
def delete(self, request):
    delete_params = QueryDict(request.body)
    obj = self.get_one(delete_params['pk'])
    obj.delete()
    return Response(status=status.HTTP_204_NO_CONTENT)

def put(self, request):
    put_params = QueryDict(request.body)
    mountain = self.get_one(put_params['id'])
    serializer = GroupPostSerializer(mountain, data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Рисунок 5 - пример отображения (часть 2)

Пример выполнения get запроса на серверной части приведен на рисунке 6.



Рисунок 6 - пример выполнения запроса к модели

Таким образом, серверная часть содержит функционал для работы с данными с помощью отображений и запросов, которые приходят с клиентской части. Серверная часть поддерживает различные виды запросов, которые могут прийти с клиентской части, при условии, что эти запросы используют HTTP протокол.

Глава 3. Описание клиентской части

3.1 Разработанные интерфейсы клиентской части

В рамках разработки клиентской части был создан ряд интерфейсов для взаимодействия с веб-сервисом. Все они реализованы с использованием Vue.js, так как данный фреймворк позволяет создавать разметку страницы, методы ее работы и стили в едином формате, который удобно релизуется и модифицируется при необходимости [4].

Также была использована библиотека Muse-UI, которая позволяет организовать и оформить внешний вид страницы с помощью контейнеров, заполненных необходимыми элементами [5]. Например, текстовыми полями, таблицами, кнопками, выпадающими списками.

Первый интерфейс, который использует пользователь это начальная страница – ее вид представлен на рисунке 7. С этой страницы пользователь может перейти к регистрации или войти в систему.

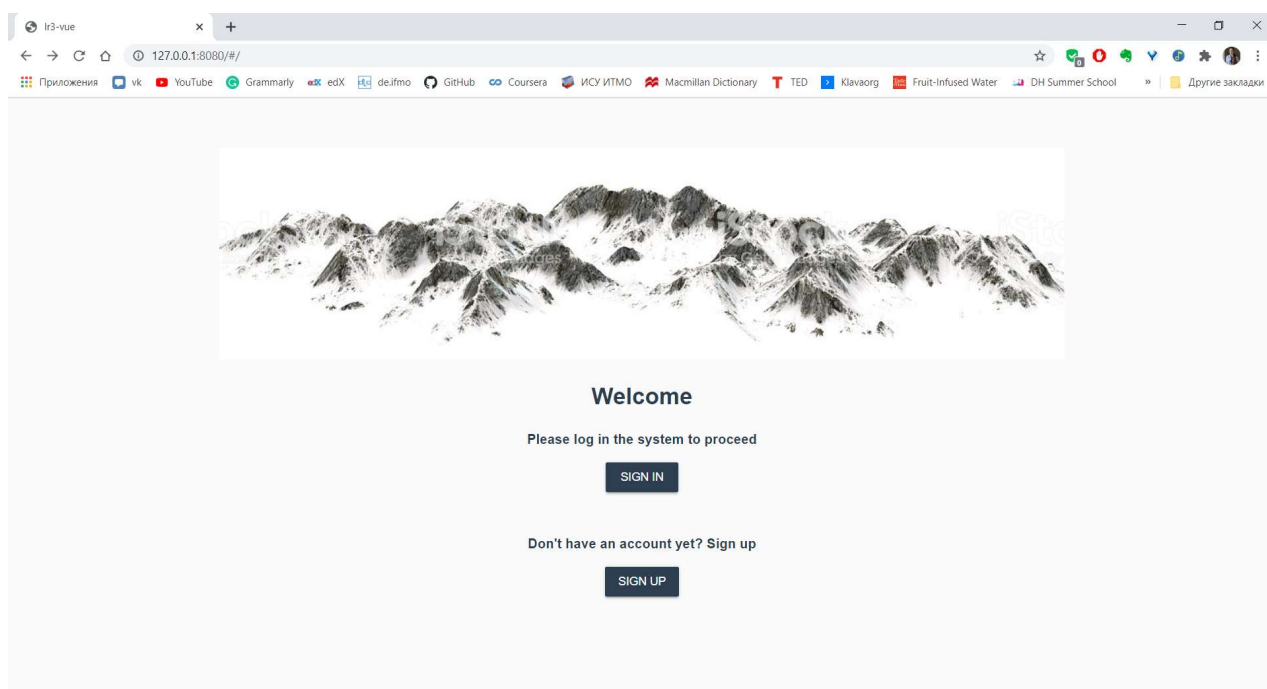


Рисунок 7 - начальная страница

Страница регистрации так же имеет свой интерфейс, он приведен на рисунке 8. Как можно заметить, веб-сервис использует единообразный дизайн на всех своих страницах.

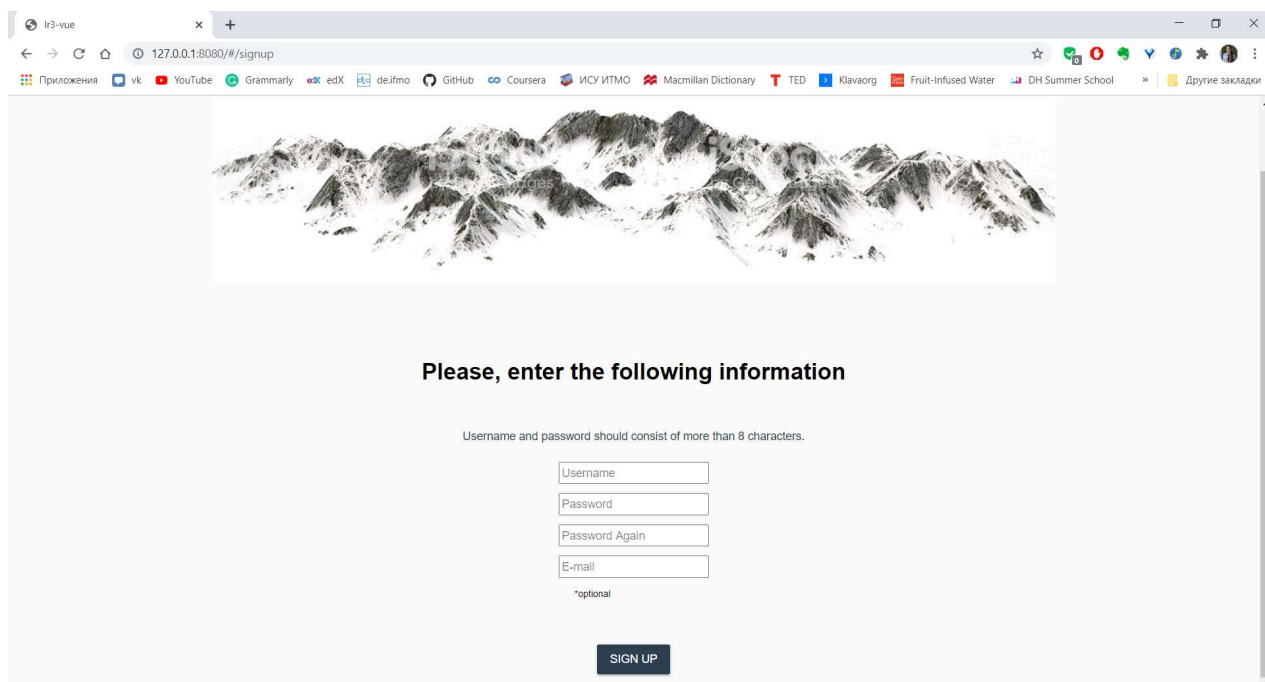


Рисунок 8 - страница регистрации

Страница входа, как и страница регистрации содержит поля для ввода текста и кнопку подтверждения. Вид этой страницы представлен на рисунке 9.

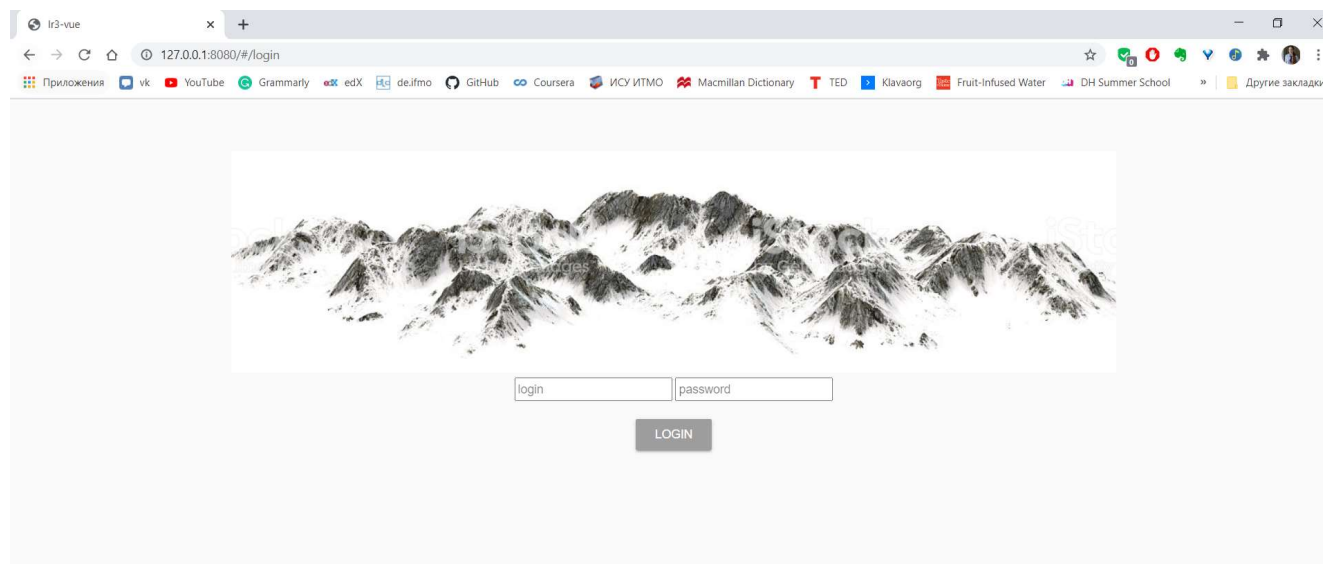


Рисунок 9 - страница входа в систему

После авторизации в системе, пользователь автоматически перенаправляется на главную страницу, откуда осуществляется навигация по страницам,

привязанным к моделям данных. Внешний вид главной страницы сервиса представлен на рисунке 10.

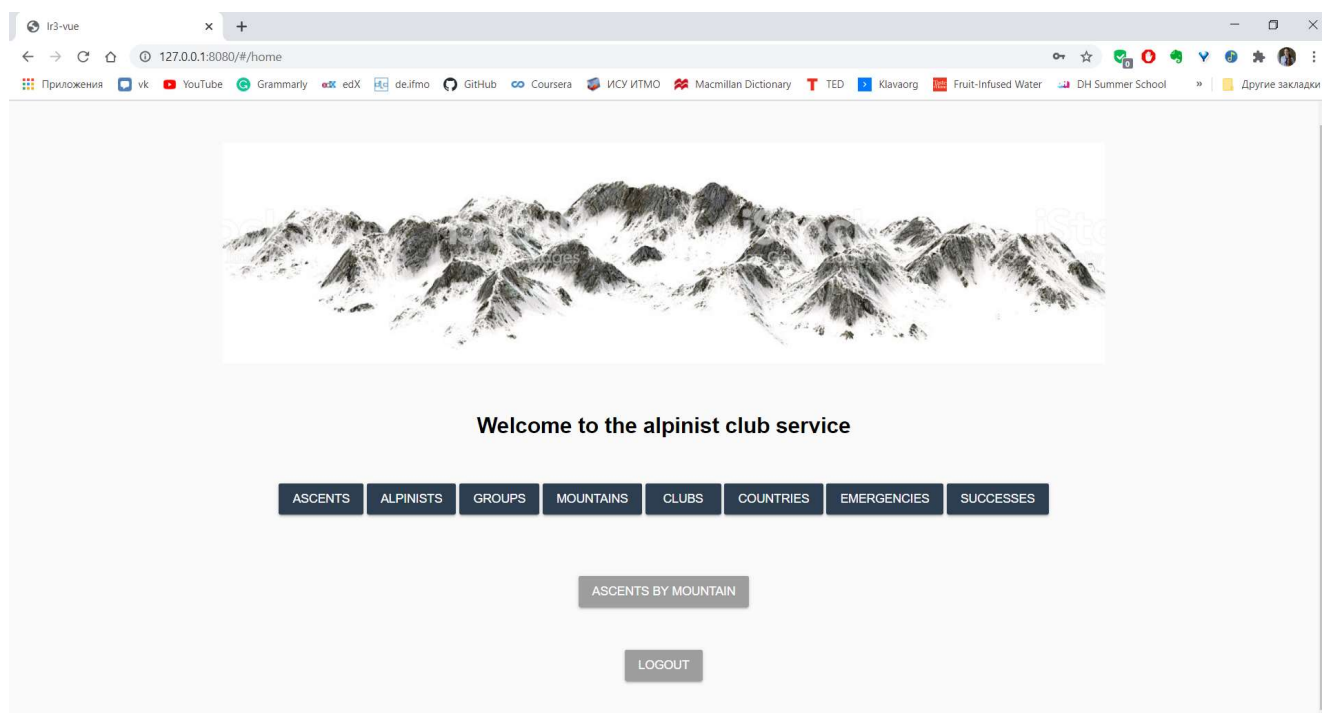


Рисунок 10 - главная страница

Отсюда пользователь может перейти к одной из страниц с данными (синие кнопки), выполнить поиск восхождений по горе или выйти из системы (серые кнопки). Рассмотрим подробнее интерфейс страниц с данными – они типовые. Рассмотрим их на примере страницы с альпинистами (переход по кнопке Alpinists).

Перейдем на страницу по кнопке и рассмотрим ее элементы.

В верхней части страницы находится кнопка «Home», по которой можно вернуться на главную страницу. Далее расположена таблица с данными, где по

строкам можно просмотреть информацию обо всех альпинистах, сохраненных в системе. Это представлено на рисунке 11.

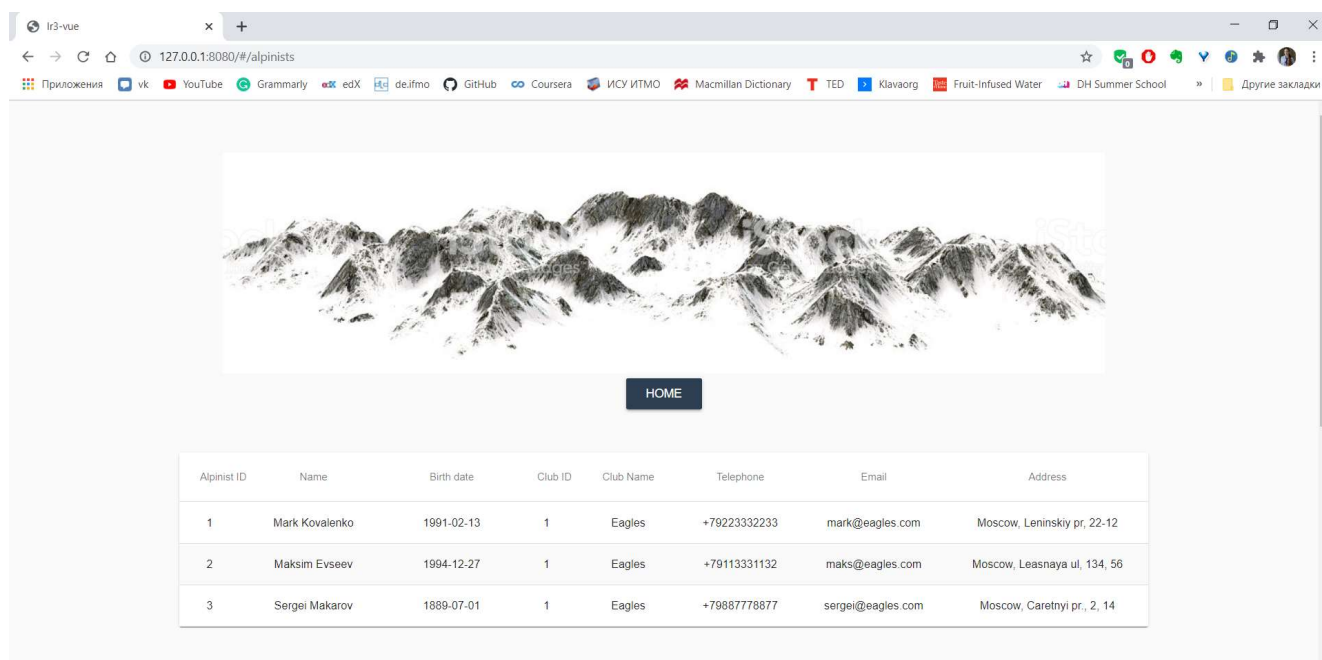


Рисунок 11 - кнопка "домой" и информация об альпинистах

Далее расположены формы для добавления и удаления данных. Они подписаны соответствующими заголовками, а на полях для ввода текста указано какие данные необходимо ввести. Это представлено на рисунке 12. Важно отметить, что при добавлении новой записи значения внешних ключей выбираются из выпадающего списка.

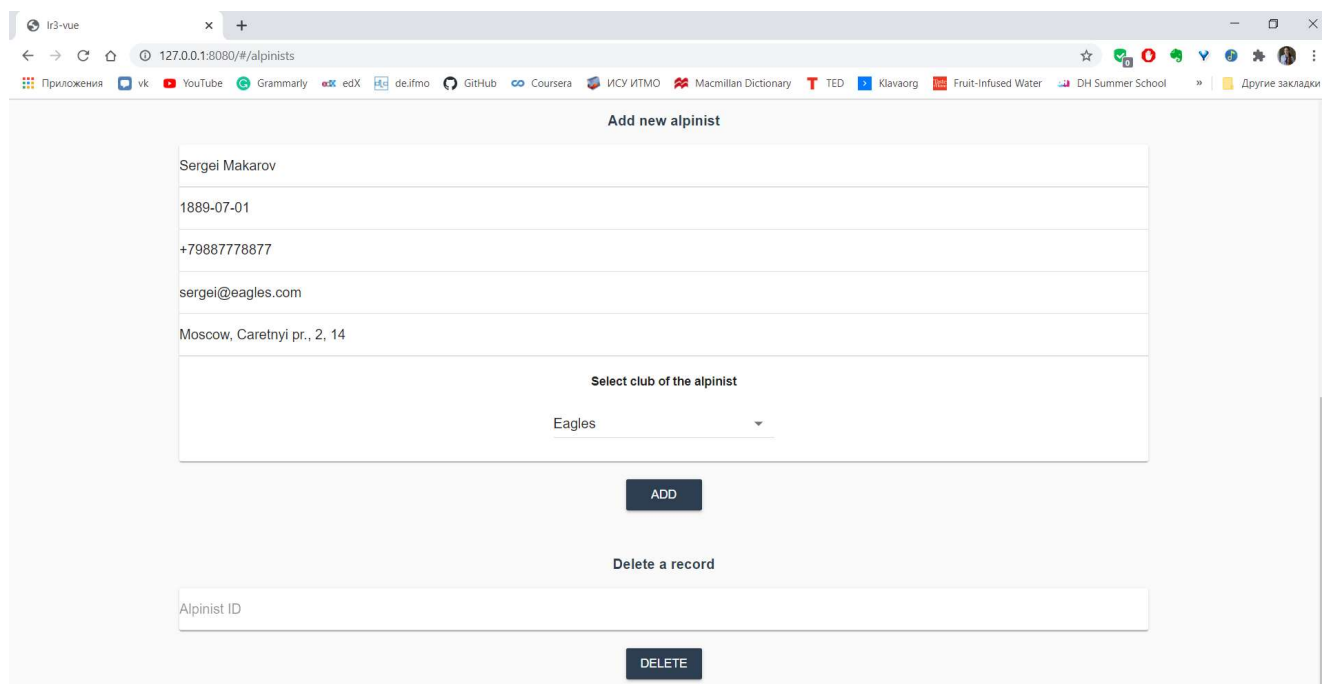


Рисунок 12 - добавление и удаление данных

Последний элемент страницы это форма для редактирования данных. В изначальном варианте она выглядит как одно поле для ввода текста, куда необходимо ввести идентификатор записи. Если запись с таким идентификатором существует, то появится форма для ввода данных, с помощью которой можно отредактировать запись. Это представлено на рисунках 13 и 14.

The image shows two forms stacked vertically. The top form is titled "Delete a record" and contains a single text input field labeled "Alpinist ID". Below this field is a dark blue button with the text "DELETE". The bottom form is titled "Edit alpinist" and contains the instruction "In order to edit a record of an alpinist, please, enter its id." followed by a text input field. Below this field is a dark blue button with the text "EDIT".

Рисунок 13 - изменение данных, ввод идентификатора

The image shows a form titled "Edit alpinist" with the instruction "In order to edit a record of an alpinist, please, enter its id." Below this is a text input field containing the number "3". The form then contains several input fields: "Name", "Birth date (yyyy-mm-dd)", "Telephone", "E-mail", and "Address". Below these is a dropdown menu labeled "Select club of the alpinist". At the bottom of the form is a dark blue button with the text "EDIT".

Рисунок 14 - изменение данных, запись найдена

Последний интерфейс это поиск восхождений по названию горы. На данной страниц пользователь выбирает название горы из выпадающего списка (всех гор, сохраненных в системе) и по нажатию кнопки получает ответ — все восхождения, совершенные на данную гору. Этот интерфейс представлен на рисунке 15.

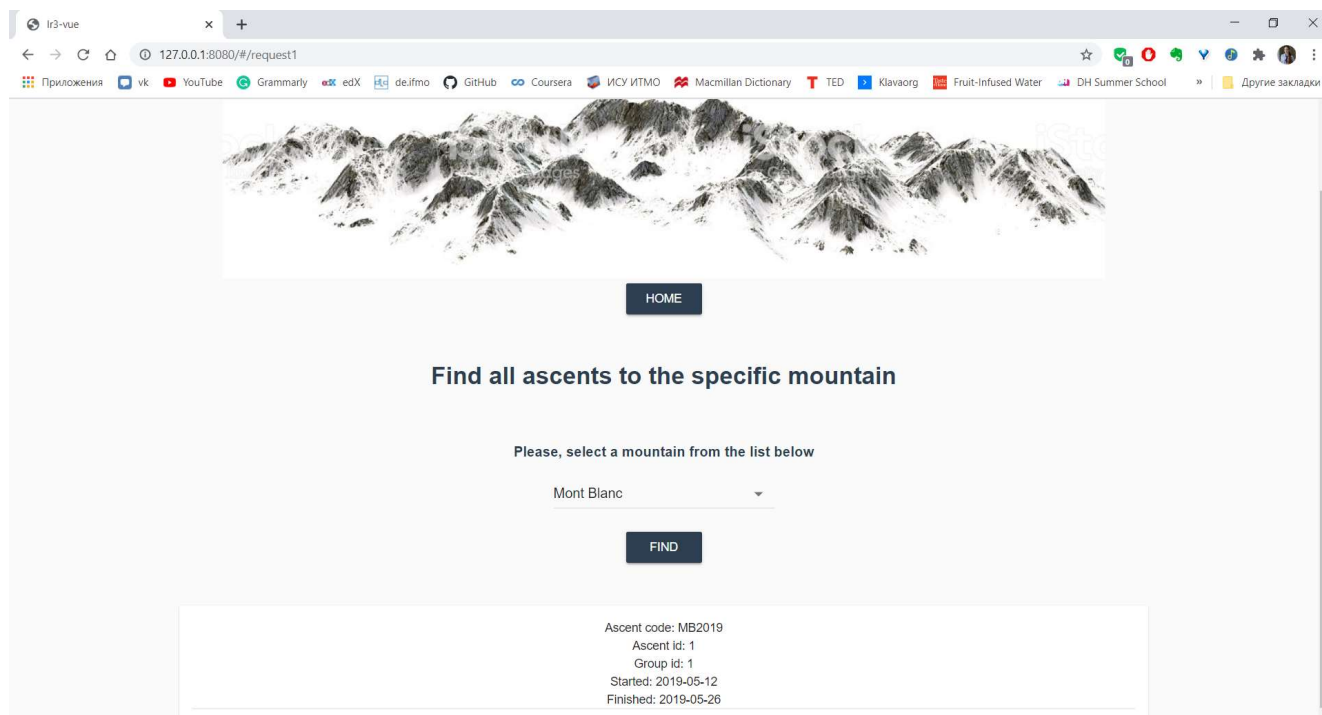


Рисунок 15 - поиск восхождений по названию горы

Клиентская часть сервиса предоставляет пользователю интерфейсы для удобного и понятного взаимодействия с системой, а именно просмотра, добавления, редактирования и удаления данных. Ее интерфейс прост для понимания пользователя, а функционал достаточен для решения необходимых задач.

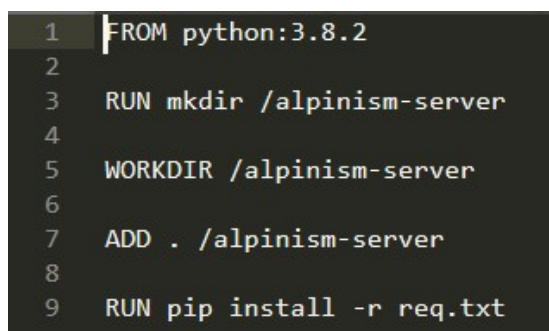
Глава 4. Контейнеризация и оркестрация

Контейнеризация — это подход к разработке программного обеспечения, при котором приложение или служба, их зависимости и конфигурация (абстрактные файлы манифеста развертывания) упаковываются вместе в образ контейнера. Контейнеризованное приложение может быть протестировано как модуль и развернуто в виде экземпляра контейнера в операционной системе (ОС) текущего узла. Docker — это проект с открытым исходным кодом для автоматизации развертывания приложений в виде переносимых, самодостаточных контейнеров, которые могут работать в облаке или локально.

Оркестрация — это координация взаимодействия нескольких контейнеров. Она позволяет строить информационные системы из небольших кирпичиков-контейнеров, каждый из которых ответственен только за одну задачу, а общение осуществляется через сетевые порты и общие директории. При необходимости контейнеры в таком «оркестре» можно заменять на другие: например, чтобы проверить работу приложения на другой версии базы данных.

Для запуска разработанного сервиса с помощью оркестрации докер-контейнеров были созданы 2 папки, одна из которых полностью содержит серверную часть, а вторая — клиентскую. В каждой папке были созданы соответствующие Dockerfile, а также был создан файл docker-compose.yml, с помощью которого осуществляется сборка и запуск проекта.

Содержание этих файлов представлено на рисунках 16, 17 и 18 ниже.

A screenshot of a text editor showing a Dockerfile. The file contains nine lines of code, numbered 1 through 9 on the left margin. The code starts with 'FROM python:3.8.2', followed by 'RUN mkdir /alpinism-server', 'WORKDIR /alpinism-server', 'ADD . /alpinism-server', and 'RUN pip install -r req.txt'.

```
1 FROM python:3.8.2
2
3 RUN mkdir /alpinism-server
4
5 WORKDIR /alpinism-server
6
7 ADD . /alpinism-server
8
9 RUN pip install -r req.txt
```

Рисунок 16 - Dockerfile для серверной части

```

1 FROM node:12.16.3
2
3 WORKDIR /alpinism-vue
4
5 COPY package*.json ./
6 CMD npm install
7 CMD npm start
8
9 COPY . .
10

```

Рисунок 17 - Dockerfile для клиентской части

```

1  version: '3'
2
3  services:
4    alpinism_db:
5      image: postgres
6      ports:
7        - "5432:5432"
8      environment:
9        - POSTGRES_DB=alpinism
10       - POSTGRES_USER=uninadia
11       - POSTGRES_PASSWORD=kitten
12
13    backend:
14      container_name: alpinism_backend
15      build:
16        context: ./alpinism-server
17        dockerfile: Dockerfile
18
19      command: bash -c "sleep 3 &&
20        python manage.py makemigrations && python manage.py migrate &&
21        python manage.py runserver 0.0.0.0:8000";
22      volumes:
23        - ./alpinism-server:/alpinism-server
24      ports:
25        - "8000:8000"
26      depends_on:
27        - alpinism_db
28
29    frontend:
30      container_name: alpinism_frontend
31      build:
32        context: ./alpinism-vue
33        dockerfile: Dockerfile
34
35      command: npm run dev
36      volumes:
37        - ./alpinism-vue:/alpinism-vue
38      ports:
39        - "8080:8080"
40      depends_on:
41        - backend

```

Рисунок 18 - файл оркестрации Docker-compose

В данном варианте сервис собирается и запускается с помощью команд `docker-compose build` и `docker-compose up`.

Заключение

В ходе проделанной работы были продемонстрированы практические навыки, полученные в рамках обучения на курсе «Основы web-программирования». С помощью Django REST фреймворка и Vue.js был разработан полноценный web-сервис.

Была создана система, которая позволяет пользователю создать аккаунт в системе и авторизоваться для работы с данным сервисом. Были разработаны отображения и интерфейсы для взаимодействия с данными, а именно их просмотра, добавления, редактирования и удаления.

Функциональные требования, выявленные в ходе подготовки к разработке web-сервиса, были удовлетворены в полном объеме, включая контейнеризацию и оркестрацию сервиса для удобства его дальнейшего использования.

Список литературы

1. Документация PostgreSQL [Электронный ресурс]. – URL: <https://postgrespro.ru/docs/postgresql/12/ddl-basics> (дата обращения: 28.05.2020)
2. Сериализации (Django REST Framework) [Электронный ресурс]. – URL: <https://www.django-rest-framework.org/api-guide/serializers/> (дата обращения: 29.05.2020)
3. Introduction to class-based views (Django) [Электронный ресурс]. – URL: <https://docs.djangoproject.com/en/3.0/topics/class-based-views/intro/> (дата обращения: 25.06.2020)
4. Документация Vue.js [Электронный ресурс]. – URL: <https://ru.vuejs.org/v2/guide/index.html> (дата обращения: 02.06.2020)
5. Документация Muse-UI [Электронный ресурс]. – URL: <https://muse-ui.org/#/en-US/installation> (дата обращения: 05.06.2020)