Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ”

Факультет
Инфокоммуникационных технологий

Образовательная программа
Интеллектуальные системы в гуманитарной сфере

Направление подготовки (специальность)
Интеллектуальные системы в гуманитарной сфере

О Т Ч Е Т

по курсовой работе по предмету “Основы web-программирования”

Тема задания: реализация web-сервисов средствами Django REST framework и Vue.js

Обучающийся Евшина Яна Алексеевна, группа К3342

Руководитель курсовой работы: Говоров Антон Игоревич

Санкт-Петербург
2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Django — это программный каркас с богатыми возможностями, подходящий для разработки сложных сайтов и веб-приложений, написанный на языке программирования Python.

Django — фреймворк для веб-приложений на языке Python. Один из основных принципов фреймворка — DRY (don't repeat yourself). Веб-системы на Django строятся из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из заметных архитектурных отличий этого фреймворка от некоторых других. Также, в отличие от многих других фреймворков, обработчики URL в Django конфигурируются явно (при помощи регулярных выражений), а не автоматически задаются из структуры контроллеров.

Первоначально разработка Django велась для обеспечения более удобной работы с новостными ресурсами, что достаточно сильно отразилось на архитектуре: фреймворк предоставляет ряд средств, которые помогают в быстрой разработке веб-сайтов информационного характера. Например, разработчику не требуется создавать контроллеры и страницы для административной части сайта, в Django есть встроенное приложение для управления содержимым, которое можно включить в любой сайт, сделанный на Django, и которое может управлять сразу несколькими сайтами на одном сервере. Административное приложение позволяет создавать, изменять и удалять любые объекты наполнения сайта, протоколируя все совершенные действия, и предоставляет интерфейс для управления пользователями и группами (с пообъектным назначением прав).

Веб-фреймворк Django используется в таких крупных и известных сайтах, как Instagram, Disqus, Mozilla, The Washington Times, Pinterest, lamoda и др.

При выполнении данной курсовой работы основной целью было овладеть практическими навыками и умениями реализации web-сервисов средствами Django REST framework, Vue.js, Muse-UI.

Для этого использовалось следующее программное обеспечение: Python 3.6, Django REST framework, Vue.js, Muse-UI.

Как результат, был реализован сайт, используя вышеуказанные технологии.

Был выбран вариант задания, где нужно было реализовать сайт для организации выставки собак.

Задачи:

1. Реализовать модель данных;
2. Определить список интерфейсов;
3. Реализовать серверную часть;
4. Реализовать клиентскую часть;
5. Выполнить отчет по проделанной работе.

Более подробный функционал реализованного сайта описан в первой главе.

1. Предметная область и функциональные требования

1.1. Описание предметной области и функциональных требований к работе

Выставка собак — публичное мероприятие по оценке соответствия собак стандартам определённой породы, с целью выявления лучшего представителя породы и их дальнейшего разведения. Оценку проводят эксперты, имеющие соответствующую лицензию.

Выставка собак – мероприятие одновременно и зоотехническое и зрелищное. Основное предназначение выставок, конечно, зоотехническое: собрать как можно больше представителей породистого собачьего племени, чтобы получить представление о существующем в данном месте и на данное время поголовье или его части; посмотреть и сравнить уровень разведения различных клубов, питомников; выявить лучших представителей пород.

Также на выставках выявляются племенные производители, дающие наиболее перспективное потомство. Именно на выставках зачастую встречаются специалисты-кинологи из разных городов и стран, здесь завязываются контакты и знакомства, чаще всего перерастающие в крепкую дружбу и сотрудничество.

Другая задача выставок – зрелищная, образовательная. На выставках есть возможность познакомиться с великим множеством пород собак, купить литературу и видеоматериалы об интересующей породе, поговорить со специалистами. На выставку приходят, чтобы лучше узнать выбранную породу перед покупкой собаки, посмотреть выставочные ринги и различные конкурсы, приобщиться к живому миру.

В соответствии с заданием курсовой работы, у нас был апотребность создания сайта, для организаций выставок. Нужно было создать программную систему, предназначенную для организаторов ежегодных выставок собак. Выставки могут быть моно- и полипородные. Система должна обеспечивать хранение сведений о собаках - участниках выставок и экспертах. Для каждой собаки в БД должны храниться сведения, о том, к какому клубу она относится, кличка, порода и возраст, классность, дата последней прививки, фамилия, имя, отчество хозяина. Перед соревнованиями собаки должны пройти обязательный медосмотр. Т.к. участие является платным, то хозяин обязан после регистрации должен оплатить счет и предоставить его организаторам, иначе собака не допускается до выставки.

Каждая собака должна выполнить 3 упражнения, за каждое из которых она получает баллы от каждого эксперта. Итогом выставки является определение медалистов по каждой породе по итоговому рейтингу. Организатор выставки должен иметь возможность добавить в базу нового участника или нового эксперта, подтвердить оплату счета и регистрацию собаки, добавлять оценки экспертов. Необходимо предусмотреть возможность выдачи отчета о результатах заданной выставки.

Для данного функционала была создана и реализована модель данных, представленная на рис.1.

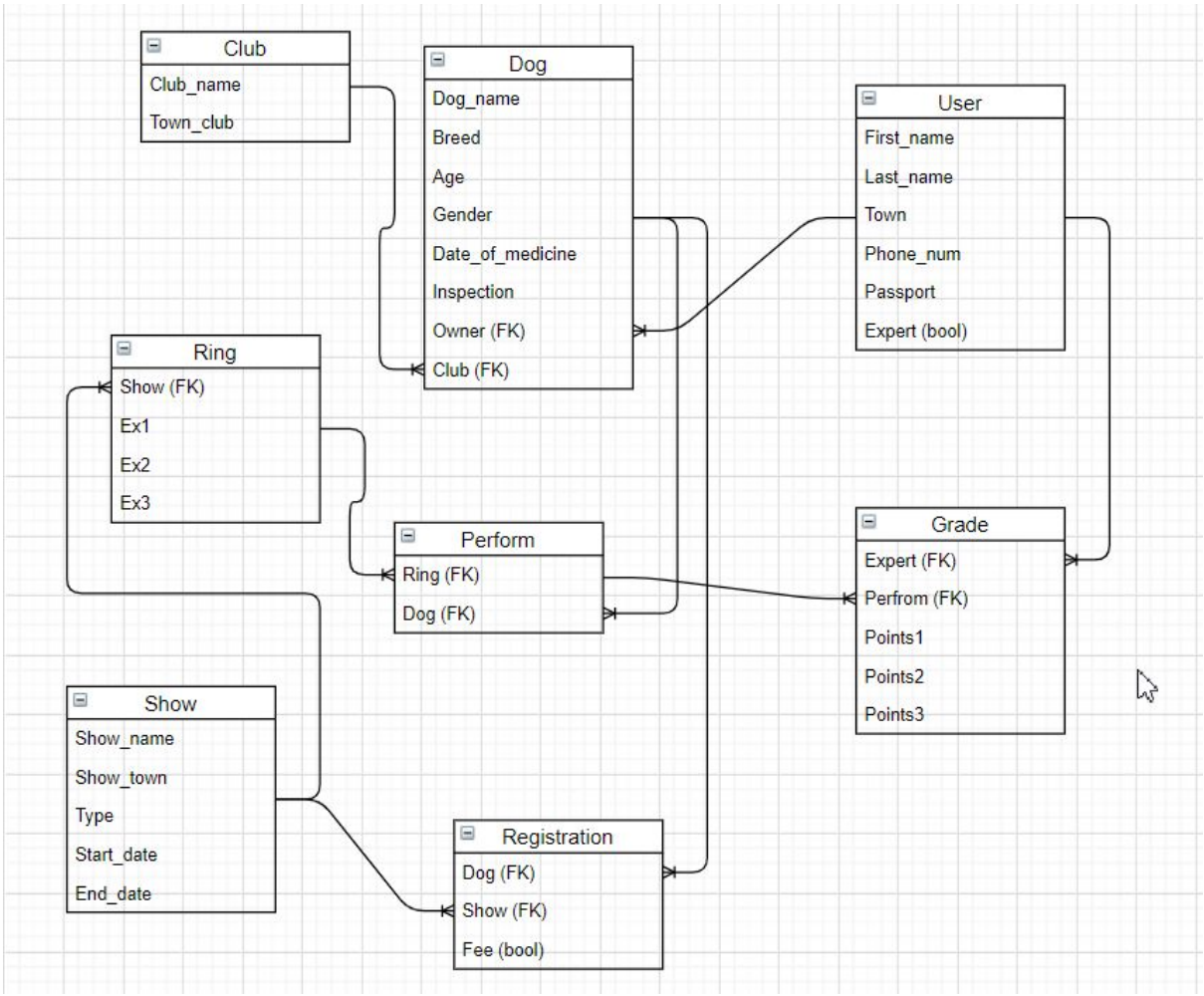

Рисунок 1 - модель данных

На данной модели имеются сущности, описанные в табл.1.

Таблица 1 - Описание сущностей модели данных

Название сущности	Описание сущности
Club	Клуб, его название и город
Dog	Собака, ее кличка, владелец, порода, возраст, пол, дата прививок, клуб
User	Пользователь, с его фамилией, именем, номер телефона, паспортными данными. Также есть поле “эксперт”, если данный пользователь является экспертом
Show	Выставка, ее название, город проведения, моно- или поли породная, даты начала и окончания
Ring	Ринг на выставке, упражнения ринга
Registration	Регистрация на выставку (обязательна для каждой собаки), также есть поле “счет”, которое обязательно для доступа к подробной информации о выставке (со стороны пользователя)
Grade	Оценка экспертом собаки. Выставляется оценка за каждое упражнение. При выводе, считается среднее арифметическое по собаке.
Perform	Выступление собаки на ринге. То есть, на каком ринге выступает собака в данной выставке

Также был определен список необходимых интерфейсов:

1. Главная страница, где выводится список выставок;
2. Для каждой выставки имеется страничка с подробным ее описанием. Для зарегистрированных на выставку пользователей, функционал расширен;
3. Список имеющихся клубов;
4. Список собак пользователя, с возможностью добавить новую собаку;
5. Страницы для входа и регистрации.

1.2 Выводы

В результате данной главы, были выявлены функциональные требования, необходимые к реализации, построена и описана модель данных, представлен список интерфейсов. Следующими частями курсовой работы будут описание реализации данных требований.

2. Серверная часть курсовой работы

2.1. Модели данных

Для серверной части работы мы использовали Django и Django Rest Framework.

Django Rest Framework (DRF) — это библиотека, которая работает со стандартными моделями Django для создания гибкого и мощного API для проекта.

В главе 1 уже была представлена и описана модель данных и сущности, которые будет использоваться для реализации сайта.

Представленные модели можно увидеть в admin Django.

Рисунок 2 - admin Django

На рис.3 можно увидеть код реализации одной из созданных моделей, а именно, собаки.

```
class Dog(models.Model):
    dog_name = models.CharField(max_length=25)
    breed = models.CharField(max_length=25)
    age = models.IntegerField()
    gender_choice = [
        ('M', 'Male'),
        ('F', 'Female'),
    ]
    gender = models.CharField(max_length=2, choices=gender_choice)
    date_of_medicine = models.DateField()
    inspection = models.BooleanField(
        verbose_name='Medicine inspection done', default=False)
    owner = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.PROTECT,
                              null=True)
    club = models.ForeignKey(Club, on_delete=models.PROTECT,
                             null=True)

    def __str__(self):
        return self.dog_name
```

Рисунок 3 - код модели собаки (Dog)

Для модели пользователя мы использовали фиксированную расширенную модель AbstractUser.

```
class User(AbstractUser):
    first_name = models.CharField(max_length=25)
    last_name = models.CharField(max_length=25)
    phone_num = models.IntegerField(null=True)
    town = models.CharField(max_length=25)
    passport = models.CharField(max_length=25)
    expert = models.BooleanField(verbose_name='Expert', default=False)

    def __str__(self):
        return self.last_name
```

Рисунок 4 - код модели пользователя

Как выглядит пользователь в admin Django можно увидеть на рис. 5. Добавлены дополнительные поля для удобства.

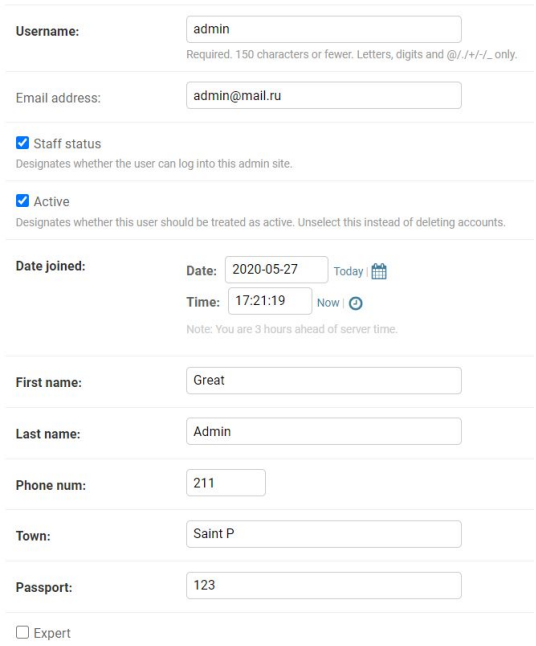

Рисунок 5 - модель пользователя в admin Django

API Django Rest Framework состоит из 3-х слоев: сериализатора, вида и маршрутизатора.

Сериализатор: преобразует информацию, хранящуюся в базе данных и определенную с помощью моделей Django, в формат, который легко и эффективно передается через API.

Вид (ViewSet): определяет функции (чтение, создание, обновление, удаление), которые будут доступны через API.

Маршрутизатор: определяет URL-адреса, которые будут предоставлять доступ к каждому виду.

2.2. Сериализаторы

Модели Django интуитивно представляют данные, хранящиеся в базе, но API должен передавать информацию в менее сложной структуре. Хотя данные будут представлены как экземпляры классов Model, их необходимо перевести в формат JSON для передачи через API.

Сериализатор DRF производит это преобразование. Когда пользователь передает информацию (например, создание нового экземпляра) через API, сериализатор берет данные, проверяет их и преобразует в нечто, что Django может сложить в экземпляр модели. Аналогичным образом, когда пользователь обращается к информации через API, соответствующие экземпляры передаются в сериализатор, который преобразовывает их в формат, который может быть легко передан пользователю как JSON.

Настройки fields позволяют точно указать, какие поля доступны этому сериализатору. В качестве альтернативы, может быть установлен exclude вместо fields, которое будет включать все поля модели, кроме тех, которые указаны в exclude.

Сериализаторы — это невероятно гибкий и мощный компонент DRF. Хотя подключение сериализатора к модели является наиболее распространенным, сериализаторы могут использоваться для создания любой структуры данных Python через API в соответствии с определенными параметрами.

Были реализованы сериализаторы для вывода каждой из моделей данных. Пример кода сериализатора можно увидеть на рис.6.

```
class DogListSerializer(serializers.ModelSerializer):
    class Meta:
        model = Dog
        fields = "__all__"


class ClubListSerializer(serializers.ModelSerializer):
    class Meta:
        model = Club
        fields = "__all__"
```

Рисунок 6 - код сериализаторов вывода всех собак и всех клубов

2.3. Виды (ViewSets)

Сериализатор анализирует информацию в обоих направлениях (чтение и запись), тогда как ViewSet - это тот код, в котором определены доступные операции. Наиболее распространенным ViewSet является ModelViewSet, который имеет следующие встроенные операции:

- Создание экземпляра: create ()
- Получение / чтение экземпляра: retrieve ()
- Обновление экземпляра (все или только выбранные поля): update () или partial_update ()
- Уничтожение / Удаление экземпляра: destroy ()
- Список экземпляров (с разбивкой по страницам по умолчанию): list ()

Если необходимы дополнительные настройки, можно использовать общие представления, вместо ModelViewSet или даже отдельные пользовательские представления.

```python
class DogListView(generics.ListAPIView, generics.CreateAPIView):
    queryset = Dog.objects.all()
    serializer_class = DogListSerializer

    def get(self, request):
        dogs = self.queryset.filter(owner=request.user)
        serializer = self.serializer_class(dogs, many=True)

        return Response(serializer.data, status=status.HTTP_200_OK)


class DogDetailView(generics.RetrieveAPIView):
    queryset = Dog.objects.all()
    serializer_class = DogListSerializer


class ClubListView(generics.ListAPIView, generics.CreateAPIView):
    queryset = Club.objects.all()
    serializer_class = ClubListSerializer
```

Рисунок 7 - view для вывода всех собак, детализации собак и списка клубов

Запустив DRF сервер, можно увидеть, что код работает. Также, поскольку мы использовали view, позволяющее сразу вносить дополнительную информацию в базу данных, внизу страницы появляется html форма, позволяющая добавить новых собак/клуб/т.д.

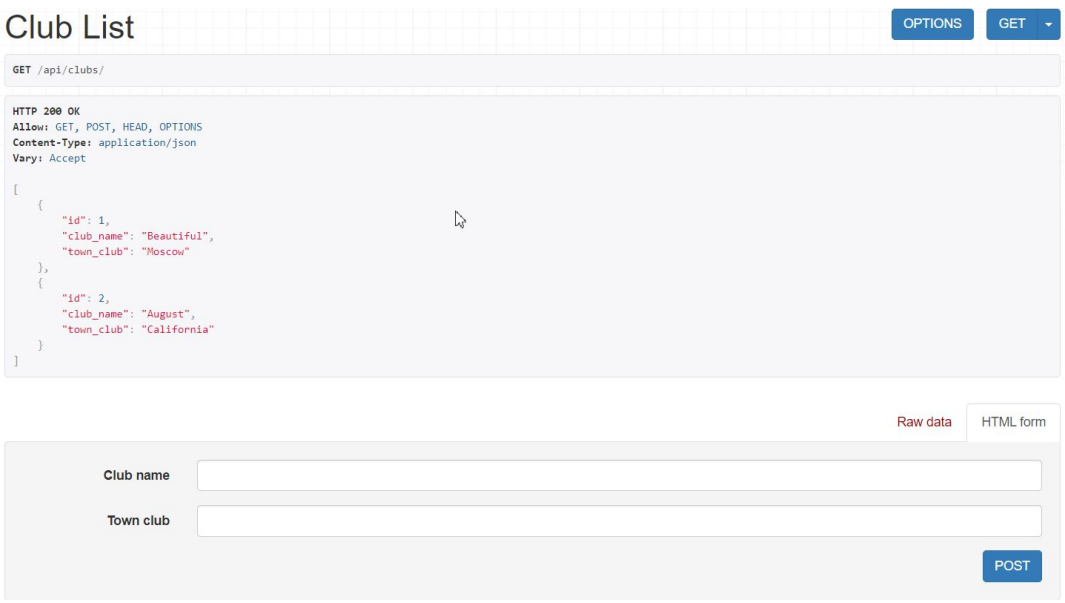

Рисунок 8 - вывод club list в формате json

2.4. Маршрутизаторы (роутеры)

И наконец, маршрутизаторы: они предоставляют верхний уровень API. Чтобы избежать создания бесконечных URL-адресов вида: «списки», «детали» и «изменить», маршрутизаторы DRF объединяют все URL-адреса, необходимые для данного вида в одну строку для каждого ViewSet.

Файл urls.py курсовой работы представлен на рис.9.

```
urlpatterns = [
    path('dogs/', views.DogListView.as_view()),
    path('dogs/<int:pk>/', views.DogDetailView.as_view()),

    path('users/register/', views.RegistrationAPI.as_view(), name='register'),
    path('users/profile-info/', views.UserDetailedView.as_view(), name='profile-info'),

    path('clubs/', views.ClubListView.as_view()),

    path('experts/', views.ExpertListView.as_view()),

    path('shows/', views.ShowListView.as_view()),
    path('shows/<int:pk>/rings/', views.RingListView.as_view(), name='rings'),
    path('shows/<int:pk>/rings/<int:ring_id>/grades/', views.GradeListView.as_view()),

    path('registrations/', views.RegistrationListView.as_view()),

    path('performs/', views.PerformListView.as_view()),
]
```

Рисунок 9 - urls.py

В серверной части работы был полностью реализован back-end, необходимый для сайта с выставками собак. Для работы были использованы Django и Django Rest Framework.

3. Клиентская часть курсовой работы

Клиентская часть курсовой работы представлена библиотекой vue.js. Vue.js — это JavaScript библиотека для создания веб-интерфейсов с использованием шаблона архитектуры MVVM (Model-View-ViewModel).

Поскольку Vue работает только на «уровне представления» и не используется для промежуточного программного обеспечения и бэкэнда, он может легко интегрироваться с другими проектами и библиотеками. Vue.js содержит широкую функциональность для уровня представлений и может использоваться для создания мощных одностраничных веб-приложений.

Функции Vue.js:

- Реактивные интерфейсы;
- Декларативный рендеринг;
- Связывание данных;
- Директивы (все директивы имеют префикс «V-». В директиву передается значение состояния, а в качестве аргументов используются html атрибуты или Vue JS события);
- Логика шаблонов;
- Компоненты;
- Обработка событий;
- Свойства;
- Переходы и анимация CSS;
- Фильтры.

Основная библиотека Vue.js очень маленькая. Это гарантирует, что нагрузка на проект, реализованный с помощью Vue.js, минимальна, а ваш сайт будет быстро загружаться.

Vue подходит для небольших проектов, которым необходимо добавить немного реактивности, представить форму с помощью AJAX, отобразить значения при вводе данных пользователем, авторизация или другие аналогичные задачи. Vue легко масштабируется и хорошо подходит для объемных проектов, поэтому его называют прогрессивным фреймворком.

Все функциональные файлы хранятся в директории components и состоят из трех частей: шаблона, части скриптов и части стилей. В качестве примера рассмотрим файл, выводящий список клубов.

Часть шаблонов ограничена тегом template и описывает внешний вид структурного модального окна.

```
<template>
  <div>
      <h3>Список клубов</h3>
      <mu-expansion-panel v-for="club in clubs" :key="club.id">
          <div slot="header">{{ club.club_name }}</div>
          <div>
              <p><b>Город</b>: {{ club.town_club }}</p>
          </div>
      </mu-expansion-panel>
  </div>
</template>
```

Рисунок 10 - часть шаблонов

Часть скриптов ограничена тегом script и описывает функции, которые срабатывают при нажатии кнопки.

```
<script>
export default {
    name: "ClubsPage",
    computed: {
        clubs() {
            return this.$store.state.club.list;
        }
    }
}
</script>
```

Рисунок 11 - часть скриптов

Часть стилей ограничена тегом style и описывает внешний вид шаблона, как стили css.

```
<style scoped>
.show-page {
  padding: 20px;
}
</style>
```

Рисунок 12 - часть стилей

Перейдем к просмотру интерфейсов самого сайта и их функционалу.

На главной странице сайта выводится список имеющихся выставок, с возможностью посмотреть подробнее о каждой выставке, в формате раскрывающегося списка.

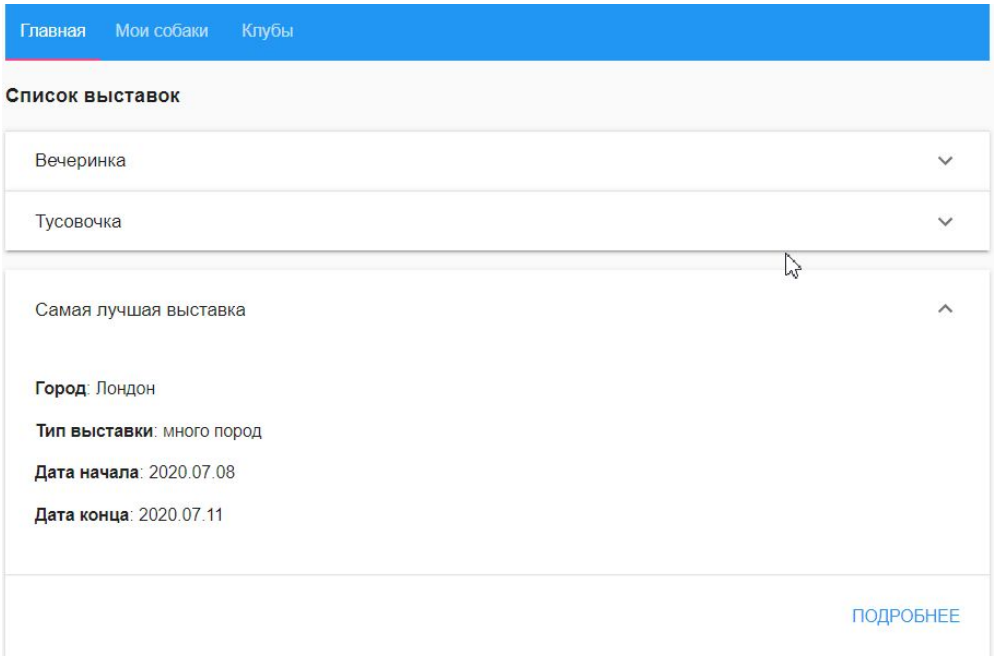

Рисунок 13 - главная страница сайта

При нажатии на кнопку “Подробнее”, можно попасть на страницу выставки, чтобы подробнее ознакомиться с ней и зарегистрировать собаку на участие.

Рисунок 14 - регистрация собаки на выставку

Форма регистрации на выставку на рис.14 доступна только авторизированным пользователям, которые предварительно добавили в личный кабинет свою собаку.

На рис.15 и 16 реализованы авторизация и регистрация пользователей на сайт выставок собак.

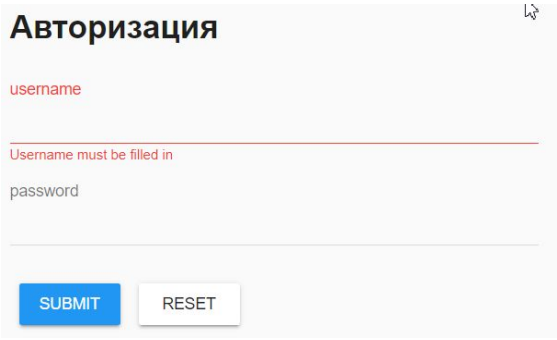

Рисунок 15 - авторизация

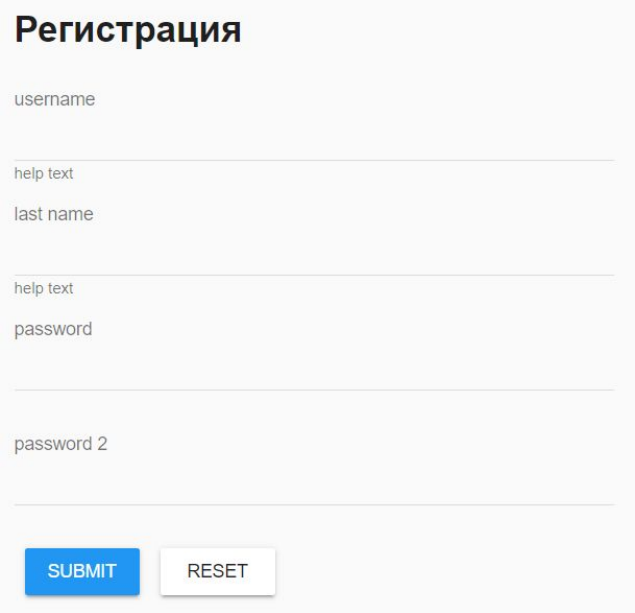

Рисунок 16 - авторизация

На рис.17 представлена форма добавления собаки в личный кабинет пользователя.

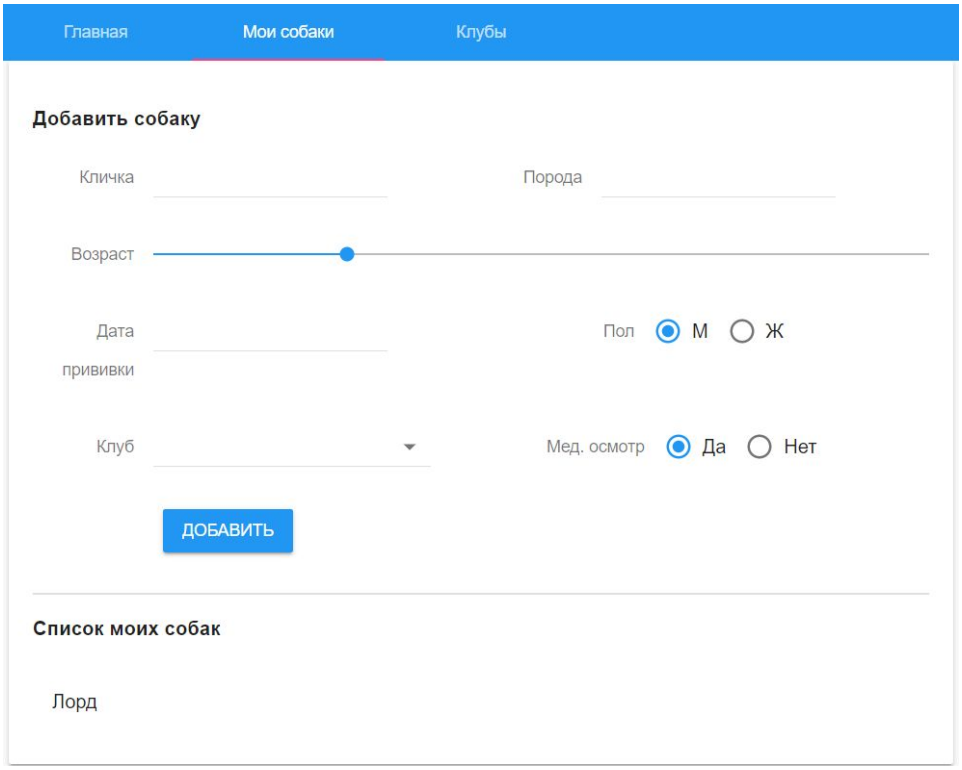

Рисунок 17 - форма для добавления собаки

После добавления открывается функция выбора собаки для регистрации на выставку.

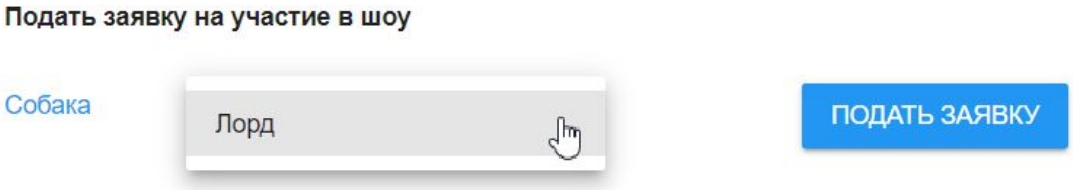

Рисунок 18 - регистрация собаки на выставку, выбор собаки

После того, как администратор одобрит заявку на выставку и распределит на ринг (это делает администратор сайта через admin Django), можно увидеть подробную информацию о выставке.

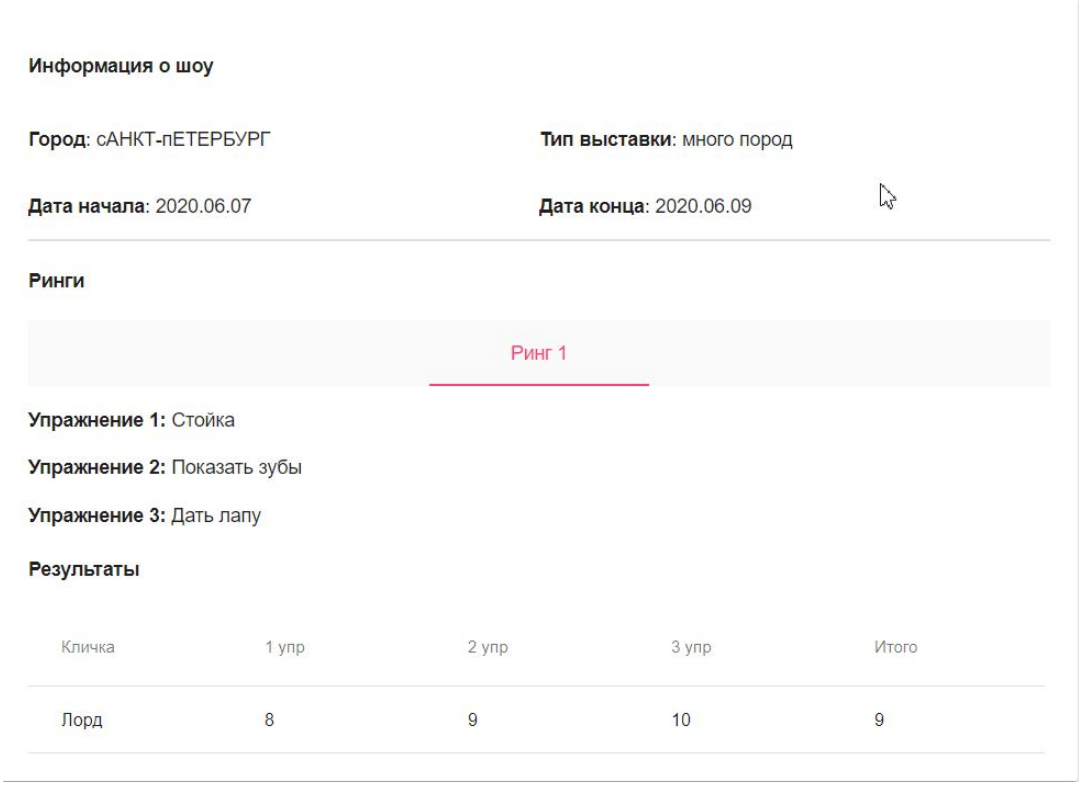

Рисунок 19 - информация о выставке, которая доступна зарегистрированному на выставку пользователю

Доступна общая информация, список рингов, имеющихся на выставке. Также задания для каждого ринга и оценка экспертами каждой собаки по упражнениям. Оценка считается, как среднее арифметическое и сортируется в порядке убывания.

Также присутствует интерфейс со списком клубов (рис.20).

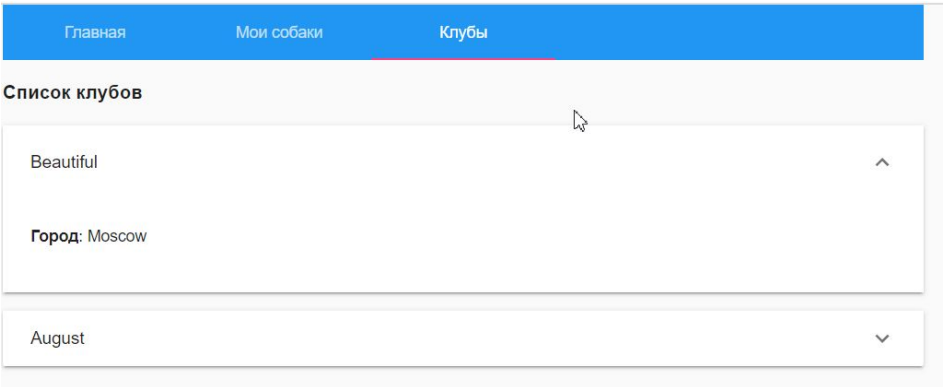

Рисунок 20 - список клубов

Таким образом, была реализована клиентская часть.

4. Использование Docker для развертывания веб-приложения

Докер — это открытая платформа для разработки, доставки и эксплуатации приложений. Docker разработан для более быстрого выкладывания ваших приложений. С помощью docker вы можете отделить ваше приложение от вашей инфраструктуры и обращаться с инфраструктурой как управляемым приложением. Docker помогает выкладывать ваш код быстрее, быстрее тестировать, быстрее выкладывать приложения и уменьшить время между написанием кода и запуска кода. Docker делает это с помощью легковесной платформы контейнерной виртуализации, используя процессы и утилиты, которые помогают управлять и выкладывать ваши приложения.

В своем ядре docker позволяет запускать практически любое приложение, безопасно изолированное в контейнере. Безопасная изоляция позволяет вам запускать на одном хосте много контейнеров одновременно. Легковесная природа контейнера, который запускается без дополнительной нагрузки гипервизора, позволяет вам добиваться больше от вашего железа.

Платформа и средства контейнерной виртуализации могут быть полезны в следующих случаях:

- упаковывание вашего приложения (и так же используемых компонент) в docker контейнеры;

- раздача и доставка этих контейнеров вашим командам для разработки и тестирования;

- выкладывания этих контейнеров на ваши продакшены, как в дата центры так и в облака.

Для создания и запуска контейнера с помощью Docker важны два файла: dockerfile и docker-compose. Dockerfile – это сценарий, который состоит из последовательности команд и аргументов, необходимых для создания образа. Такие сценарии упрощают развёртывание и процесс подготовки приложения к запуску.

```
FROM python:3

ENV PYTHONUNBUFFERED 1

RUN mkdir /application

WORKDIR /application

COPY . /application

RUN pip install -r requirements.txt
```

Рисунок 21 - Dockerfile

Docker compose - инструмент для создания и запуска многоконтейнерных Docker приложений. В Compose, вы используете специальный файл для конфигурирования ваших сервисов приложения. Затем, используется простая команда, для создания и запуска всех сервисов из конфигурационного файла.

```
version: '3'

services:
  server:
    build: .
    container_name: django
    command: bash -c "sleep 3 && python manage.py makemigrations && python manage.py migrate && python manage.py runserver 0.0.0.0:8000"
    volumes:
     - .:/application
    ports:
     - "8000:8000"

  client:
    build: ./vuedog
    container_name: front
    command: npm run serve
    volumes:
     - ./vuedog:/application
      - /application/node_modules
    ports:
     - "8080:8080"
```

Рисунок 22 - Docker compose

5. Дополнение. Помощь в реализации сервиса онлайн-тетради для факультета ИКТ

На факультете инфокоммуникационных технологий планируется создание сервиса для реализации публикации и проверки домашнего задания “Онлайн-тетрадь”.

В качестве реализованных мной задач были следующие:

1. Реализация фильтра среди студентов - возможность поиска студента по фамилии;

```python
class StudentListView(generics.ListAPIView):
    serializer_class = StudentSerializer
    permission_classes = [permissions.AllowAny]

    def get_queryset(self):
        queryset = User.objects.filter(role='student')
        last_name = self.request.query_params.get('last_name', None)
        group_id = self.request.query_params.get('group_id', None)
        stream_id = self.request.query_params.get('stream_id', None)

        if last_name is not None:
            queryset = queryset.filter(last_name__istartswith=last_name)

        if group_id is not None:
            queryset = queryset.filter(group_id=group_id)

        if stream_id is not None:
            queryset = queryset.filter(group__streams__id=stream_id)

        return queryset
```

Рисунок 23 - реализация поиска среди студентов

2. Создание учебных групп, потоков, студентов

```python
class StudentStreamListCreateView(generics.ListCreateAPIView):
    queryset = StudentStream.objects.all()
    serializer_class = StudentStreamSerializer
    permission_classes = [permissions.AllowAny]


class StudentStreamRetrieveUpdateView(generics.RetrieveUpdateAPIView):
    queryset = StudentStream.objects.all()
    serializer_class = StudentStreamSerializer
    permission_classes = [permissions.AllowAny]


class GroupInStreamListCreateView(generics.ListCreateAPIView):
    queryset = StudentStream.objects.all()
    serializer_class = StudentGroupSerializer
    permission_classes = [permissions.AllowAny]

    def get(self, request, pk):
        stream = get_object_or_none(StudentStream, {'id': pk})

        if stream is None:
            return Response(status=status.HTTP_404_NOT_FOUND)

        serializer = self.serializer_class(stream.groups, many=True)

        return Response(serializer.data, status=status.HTTP_200_OK)
```

```python
class StudentGroupMembersListCreateView(generics.ListCreateAPIView):
    queryset = StudentGroup.objects.all()
    serializer_class = GroupMemberSerializer
    permission_classes = [permissions.AllowAny]

    def get(self, request, pk):
        student_group = self.queryset.get(id=pk)
        members = student_group.members.all()
        serializer = self.serializer_class(members, many=True)

        return Response(serializer.data, status=status.HTTP_200_OK)

    def post(self, request, pk):
        student_group = self.queryset.get(id=pk)

        if 'user_id' not in request.data:
            return Response({'user_id': ['Отсутствует обязательное поле user_id']},
                            status=status.HTTP_400_BAD_REQUEST)

        try:
            student_group.add_member(request.data['user_id'])
        except ObjectDoesNotExist:
            return Response({'user_id': ['Пользователь с указанным user_id не был найден']},
                            status=status.HTTP_404_NOT_FOUND)
```

```python
def post(self, request, pk):
    stream = get_object_or_none(StudentStream, {'id': pk})

    if stream is None:
        return Response(status=status.HTTP_404_NOT_FOUND)

    if 'group_id' not in request.data:
        return Response({'group_id': ['Отсутствует обязательное поле group_id']},
                        status=status.HTTP_400_BAD_REQUEST)

    group = get_object_or_none(StudentGroup, {'id': request.data['group_id']})

    if group is None:
        return Response({'group_id': ['Группа с указанным group_id не была найдена']},
                        status=status.HTTP_404_NOT_FOUND)

    group_in_stream = GroupInStream(group=group, stream=stream)
    group_in_stream.save()

    serializer = self.serializer_class(stream.groups, many=True)

    return Response(serializer.data, status=status.HTTP_200_OK)
```

Рисунок 24 - добавление студентов, групп, потоков

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была реализована система для организации выставки собак с полным функционалом, который требовался. Все задачи были реализованы: выполнена модель данных ,представлен список интерфейсов для сайта, а также реализованы клиентская и серверная части системы.

Поставленная цель была выполнена: навыки и умения реализации web-сервисов приобретены, могут впоследствие успешно использоваться в будущем.

СПИСОК ЛИТЕРАТУРЫ

1. Django documentation. – Электрон. текстовые дан.: – Режим доступа: https://docs.djangoproject.com/en/3.0/, свободный.
2. Django Rest Framework documentation. – Электрон. текстовые дан.: – Режим доступа: https://www.django-rest-framework.org/topics/documenting-your-api/, свободный.
3. Vue.js documentation. – Электрон. текстовые дан.: – Режим доступа: https://ru.vuejs.org/v2/guide/index.html, свободный.