Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«УНИВЕРСИТЕТ ИТМО»

Факультет: ИКТ

Образовательная программа: Интеллектуальные системы в гуманитарной сфере

Направление подготовки (специальность) 45.03.04 Интеллектуальные системы в гуманитарной сфере

ОТЧЁТ

по курсовой работе

Тема задания: «Реализация web-сервисов средствами Django REST framework, Vue.js, Vuetify»

Обучающийся: Шамахова Александра Сергеевна, группа K3343

Преподаватель дисциплины: Говоров А.И., ассистент кафедры ИТГС Университета ИТМО

Оценка за курсовую \_\_\_\_

Дата \_\_\_\_\_\_\_\_

Санкт-Петербург

2020

# СОДЕРЖАНИЕ

Оглавление

[СОДЕРЖАНИЕ 2](#_Toc55434729)

[ВВЕДЕНИЕ 3](#_Toc55434730)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ, ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ 4](#_Toc55434731)

[1.1. Описание предметной области 4](#_Toc55434732)

[1.2. Функциональные требования 5](#_Toc55434733)

[2. ПРОЕКТИРОВАНИЕ WEB-СЕРВИСА И БАЗЫ ДАННЫХ 6](#_Toc55434734)

[2.1. Средства разработки и проектирование web-сервиса 6](#_Toc55434735)

[2.2. Архитектура базы данных 8](#_Toc55434736)

[3. СЕРВЕРНАЯ ЧАСТЬ СЕРВИСА 9](#_Toc55434737)

[4. КЛИЕНТСКАЯ ЧАСТЬ СЕРВИСА 19](#_Toc55434738)

[4.1. Главная страница 19](#_Toc55434739)

[4.2. Регистрация и авторизация 21](#_Toc55434740)

[4.3. Личный кабинет 23](#_Toc55434741)

[ЗАКЛЮЧЕНИЕ 25](#_Toc55434742)

[СПИСОК ЛИТЕРАТУРЫ 26](#_Toc55434743)

[Приложение 1. Models.py 27](#_Toc55434744)

[Приложение 2. Serializers.py 30](#_Toc55434745)

[Приложение 3. Views.py 34](#_Toc55434746)

[Приложение 4. Vue.js components 38](#_Toc55434747)

# ВВЕДЕНИЕ

В задании 4 варианта было предложено создать программную систему, предназначенную для организаторов ежегодных выставок собак. Организатор собирает информацию об участниках, месте проведения, экспертах и спонсорах, основная цель — учесть взаимосвязи.

В рамках системы владелец собаки должен иметь возможность зарегистрироваться, войти в личный кабинет, работать с информацией о своих собаках и заявках. Аналогично, организатор имеет свой личный кабинет с предоставленной только ему информацией.

Были поставлены цели:

1. Изучить предметную область;
2. Спроектировать БД, соответствующую предметной области;
3. Реализовать серверную часть приложения;
4. Реализовать клиентскую часть приложения.
5. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ, ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ
   1. Описание предметной области

Требуется создать программную систему, предназначенную для организаторов ежегодных выставок собак. Выставки могут быть моно- и полипородные. Она должна обеспечивать хранение сведений о собаках - участниках выставок и экспертах. Участие может быть индивидуальным или от клуба. У выставки могут быть спонсоры, которые могут спонсировать разные выставки.

Для каждой собаки в БД должны храниться сведения, о том, к какому клубу она относится, кличка, порода и возраст, классность, сведения о родословной (номер документа, клички родителей), дата последней прививки, фамилия, имя, отчество и паспортные данные хозяина. Перед соревнованиями собаки должны пройти обязательный медосмотр.

Т.к. участие является платным, то хозяин обязан после регистрации до прохождения медосмотра должен оплатить счет и предоставить его организаторам. Собака допускается до соревнований, если она успешно прошла медосмотр.

Сведения об эксперте должны включать фамилию и имя, номер ринга, который он обслуживает, клуб, название клуба, в котором он состоит. Каждый ринг могут обслуживать несколько экспертов. Каждая порода собак выступает на своем ринге, но на одном и том же ринге в разное время могут выступать разные породы.

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, снять эксперта с судейства, заменив его другим, отстранить собаку от участия в выставке.

* 1. Функциональные требования

В результате анализа текста задания были сформулированы следующие функциональные требования:

1. Регистрация клиента – хозяина собаки
2. Добавление собаки клиентом
3. Добавление собаки к участию на выставке клиентом
4. Отображение заявок клиента в личном кабинете
5. Кабинет организатора с ограниченным доступом
6. Просмотр всех заявок организатором
7. Возможность руководителем снять собаку или эксперта с выставки
8. Отображение информации о собаках, клубах, породах и экспертах для организатора
9. ПРОЕКТИРОВАНИЕ WEB-СЕРВИСА И БАЗЫ ДАННЫХ
   1. Средства разработки и проектирование web-сервиса

Для приложения была выбрана архитектура «клиент-сервер», где клиент — любое средство, способное отправить HTTP-запрос, а сервер — абстрактная машина в сети, способная получить HTTP-запрос, его обработать и вернуть корректный ответ. Звено архитектура, обеспечивающее сохранность информации, — БД.

Для реализации сервиса использовалось следующее ПО:

БД – PostgreSQL, подключенная в settings.py с помощью psycopg2-binary (рис. 1);

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 — Подключенная БД

Backend – Django (3.1.2) и конкретно Django REST framework (3.12.1), отвечающий за все запросы, требуемые исходя из функциональных требований (отображение, фильтрация, добавление, изменение данных и т. д.). В проекте использовались несколько модулей (рис. 2);

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – requirements.txt

Запросы к серверу разрешены с помощью cors headers (рис. 3);

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 – Разрешение запросов

Для выполнения тестовых запросов в процессе создания API дополнительно использовалось приложение Postman.

Frontend – Vue.js в комбинации с плагином Vuetify для реализации интерфейсов и базового дизайна.

///

* 1. Архитектура базы данных

При проектировании базы данных dogshowdb были выбраны следующие сущности:

* Пользователь (User)
* Собаки (Dogs)
* Спонсоры (Funders)
* Эксперты (Experts)
* Выставки (Shows)
* Ринги (Rings)

А также ассоциативные сущности:

* Запись на выставку (Records)
* Выступление (Acts)
* Спонсирование (Sponsors)
* Судейство (Judgings)

Схема БД представлена на рис. 4:

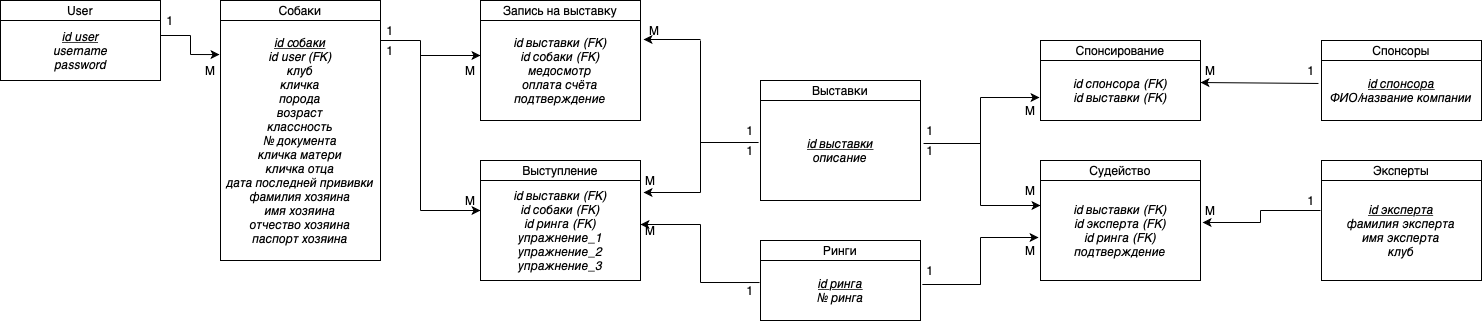


Рисунок 4 – Схема БД

1. СЕРВЕРНАЯ ЧАСТЬ СЕРВИСА

Бэкенд приложения был реализован с помощью Django REST framework (3.12.1).

Rest (сокр. англ. Representational State Transfer, «передача состояния представления») — стиль построения архитектуры рапределённого приложения. Данные в REST передаются в одном из стандартных форматов (например, HTML, XML, JSON).

Django REST framework — удобный инструмент для работы с rest, основанный на идеологии фреймворка Django. [1]

Модели таблиц БД из прошлой главы описаны в файле models.py (см. Приложение 1). Далее они были зарегистрированы в админ-панели (рис. 5,6).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 – admin.py

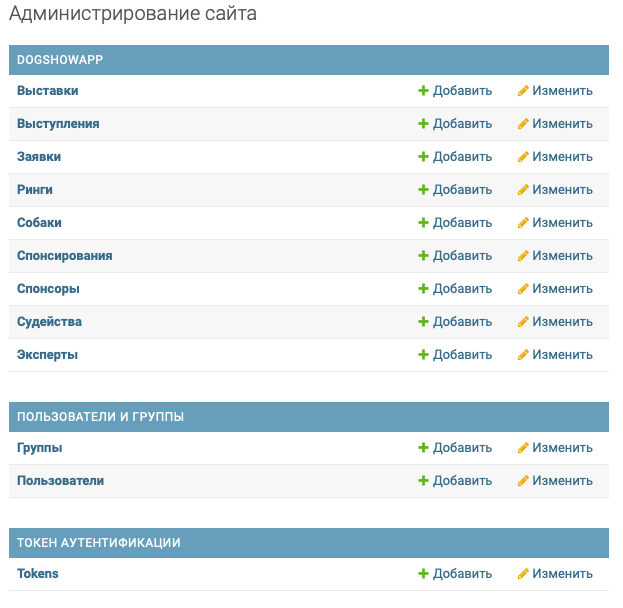


Рисунок 6 – Админ-панель Django

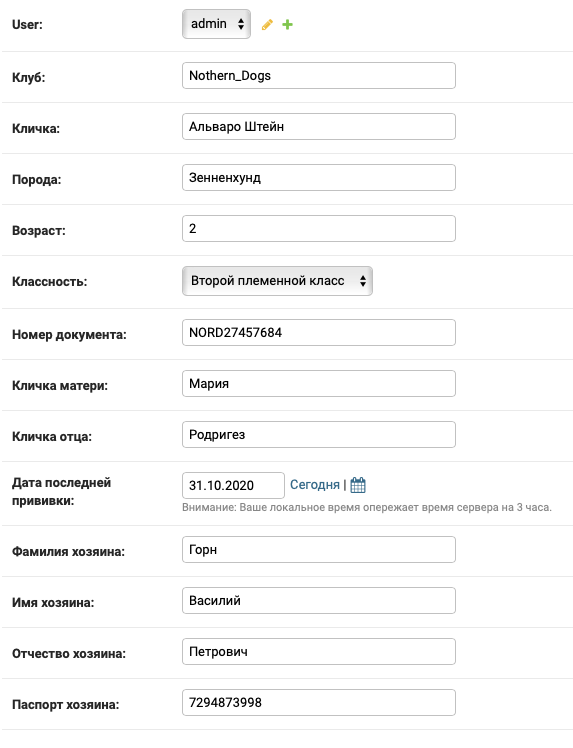


Рисунок 7 – Пример собаки

В serializers.py (см. Приложение 2) прописаны сериализаторы для каждой модели с помощью serializers.ModelSerializer из rest\_framework, представляющие их в формате json. В некоторых случаях понадобилось несколько для одной модели (например, выбраны различные поля или read\_only=True). Также у полей с множественным выбором указан source для корректного отображения.

В файлу views.py с помощью generics views из rest\_framework описаны представления, основанные на классах. Были использованы generics.ListAPIView, CreateAPIView, RetrieveUpdateDestroyAPIView, в зависимости от того, какие функции из CRUD (create, read, update, delete) были необходимы в каждой ситуации. Скриншоты views.py – в Приложении 3.

В некоторых view требовалась фильтрация объектов запроса, она реализуется функцией get\_queryset(self), которая получается все объекты модели, затем параметры запроса, после этого – фильтрует записи, возвращая queryset.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8 – пример представления с фильтрацией

В urls.py (рис. 9) содержатся адреса, соответствующие каждому из представлений:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9 – dogshowapp/urls.py

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 10 – dogshow/urls.py

Если теперь перейти по адресу localhost:8000/api/ отобразится:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 11 – Адреса, относящиеся к API

В этом списке api/auth и api/auth/token – авторизация по токену djoser.

Получившиеся эндпоинты сведены в таблицу 1.

Таблица 1 – сводная таблица информации о созданном API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | urls | запросы | назначение | соответствие с views.py |
| 5 | api/dogs/all | GET, HEAD, OPTIONS | вывод всех собак с фильтром по клубу, породе, id\_хозяина | class DogsListView(  generics.ListAPIView) |
| 6 | api/dogs/new/ | POST, OPTIONS | ввод новой собаки | class CreateDogView(  generics.CreateAPIView) |
| 7 | api/dogs/<int:pk> | GET, PUT, PATCH, DELETE, HEAD, OPTIONS | редактирование собаки с определённым id (primary key) | class GetDogView(  generics.  RetrieveUpdateDestroyAPIView) |
| 8 | api/experts/all | GET, HEAD, OPTIONS | вывод всех экспертов | class ExpertsListView(  generics.ListAPIView) |
| 9 | api/experts/new/ | POST, OPTIONS | ввод нового эксперта | class CreateExpertView(  generics.CreateAPIView) |
| 10 | api/experts/<int:pk> | GET, PUT, PATCH, DELETE, HEAD, OPTIONS | редактирование эксперта с определённым id | class GetExpertView(  generics.  RetrieveUpdateDestroyAPIView) |
| 11 | api/funders/all | GET, HEAD, OPTIONS | вывод всех спонсоров | class FundersListView(  generics.ListAPIView) |
| 12 | api/funders/new/ | POST, OPTIONS | ввод нового спонсора | class CreateFunderView(  generics.CreateAPIView) |
| 13 | api/funders/<int:pk> | GET, PUT, PATCH, DELETE, HEAD, OPTIONS | редактирование спонсора с определённым id | class GetFunderView(  generics.  RetrieveUpdateDestroyAPIView) |
| 14 | api/rings/all | GET, HEAD, OPTIONS | вывод всех рингов | class RingsListView(  generics.ListAPIView) |
| 15 | api/rings/new/ | POST, OPTIONS | ввод нового ринга | class CreateRingView(  generics.CreateAPIView) |
| 16 | api/rings/<int:pk> | GET, PUT, PATCH, DELETE, HEAD, OPTIONS | редактирование ринга с определённым id | class GetRingView(  generics.  RetrieveUpdateDestroyAPIView) |
| 17 | api/shows/all | GET, HEAD, OPTIONS | вывод всех выставок | class ShowsListView(  generics.ListAPIView) |
| 18 | api/shows/new/ | POST, OPTIONS | ввод новой выставки | class CreateShowView(  generics.CreateAPIView) |
| 19 | api/shows/<int:pk> | GET, PUT, PATCH, DELETE, HEAD, OPTIONS | редактирование выставки с определённым id | class GetShowView(  generics.  RetrieveUpdateDestroyAPIView) |
| 20 | api/records/all | GET, HEAD, OPTIONS | вывод заявок с фильтром по наличию подтверждения участия, id\_выставки | class RecordsListView(  generics.ListAPIView) |
| 21 | api/records/new/ | POST, OPTIONS | ввод новой заявки (поля выставка и собака) | class CreateRecordView(  generics.CreateAPIView) |
| 22 | api/records/<int:pk> | GET, PUT, PATCH, DELETE, HEAD, OPTIONS | редактирование заявки с определённым id | class GetRecordView(  generics.  RetrieveUpdateDestroyAPIView) |
| 23 | api/acts/all | GET, HEAD, OPTIONS | вывод выступлений с фильтром по id\_собаки | class ActsListView(  generics.ListAPIView) |
| 24 | api/acts/new/ | POST, OPTIONS | ввод нового выступления | class CreateActView(  generics.CreateAPIView) |
| 25 | api/acts/<int:pk> | GET, PUT, PATCH, DELETE, HEAD, OPTIONS | редактирование выступления с определённым id | class GetActView(  generics.  RetrieveUpdateDestroyAPIView) |
| 26 | api/sponsors/all | GET, HEAD, OPTIONS | вывод всех спонсирований | class SponsorsListView(  generics.ListAPIView) |
| 27 | api/sponsors/new/ | POST, OPTIONS | ввод спонсирования | class CreateSponsorView(  generics.CreateAPIView) |
| 28 | api/sponsors/<int:pk> | GET, PUT, PATCH, DELETE, HEAD, OPTIONS | редактирование спонсирования с определённым id | class GetSponsorView(  generics.  RetrieveUpdateDestroyAPIView) |
| 29 | api/judgings/all | GET, HEAD, OPTIONS | вывод всех судейств с фильтром по рингу | class JudgingsListView(  generics.ListAPIView) |
| 30 | api/judgings/new/ | POST, OPTIONS | ввод судейства | class CreateJudgingView(  generics.CreateAPIView) |
| 31 | api/judgings/<int:pk> | GET, PUT, PATCH, DELETE, HEAD, OPTIONS | редактирование судейства с определённым id | class GetJudgingView(  generics.  RetrieveUpdateDestroyAPIView) |

Многие элементы аналогичны, рассмотрим по одному характерному примеру.

api/dogs/all – вывод всех собак

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

Рисунок 12 – api/dogs/all

Отфильтруем по породе: api/dogs/all?dog\_breed=Зенненхунд

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 13 – Фильтр собак по породе

api/experts/new – поле добавления эксперта

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 14 – api/experts/new

api/acts/new – добавление результатов выступления

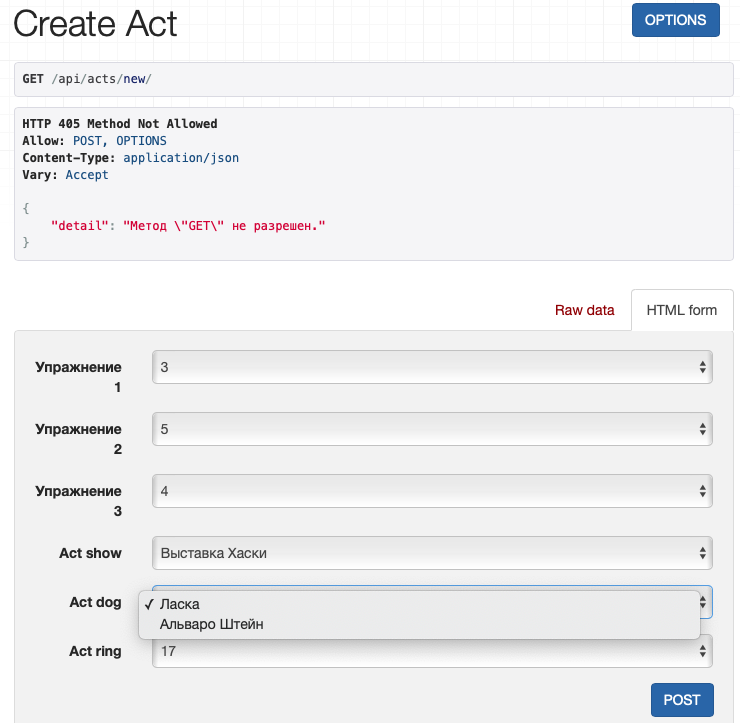


Рисунок 15 – api/acts/new

api/records/1 – работа с заявкой, здесь можно подтвердить прохождение участником медосмотра и оплату чёта, подтвердить участие – или наоборот, снять с выставки

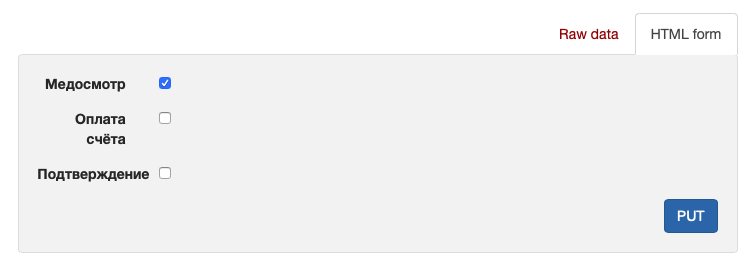


Рисунок 16 – api/records/1

api/dogs/3 – изменение сведений о собаке

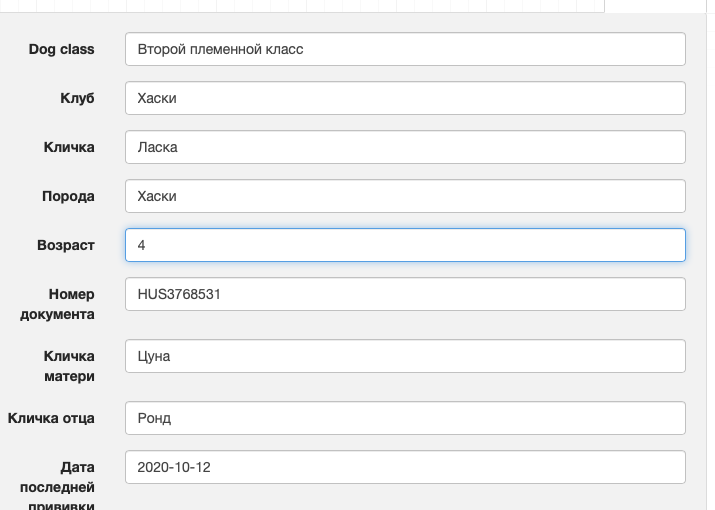


Рисунок 17 – api/dogs/3

1. КЛИЕНТСКАЯ ЧАСТЬ СЕРВИСА
   1. Главная страница

Стартовая страница представлена следующим образом: на ней располагает меню, выдвигающееся по нажатию на кнопку, а основная часть представлена списком выставок, спонсоров и экспертов. Отдельно стоит заметить, что лучшие участники выбираются и сортируются по максимальному количеству баллов, набранных на выставке. Для обращения к серверу использовался axios [4]. Весь код представлен в Приложении 4.

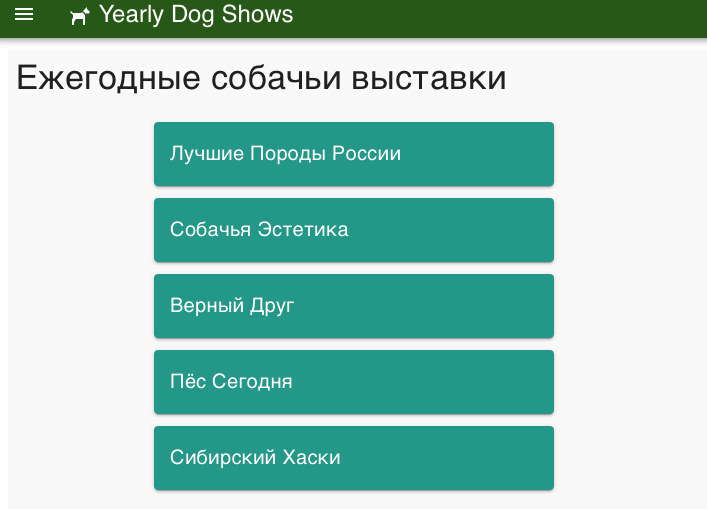


Рисунок 18 – Выставки

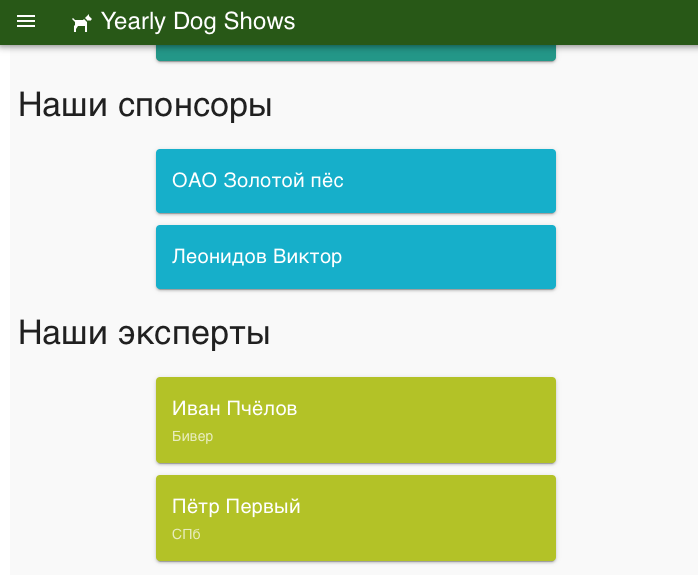


Рисунок 19 – Спонсоры и эксперты

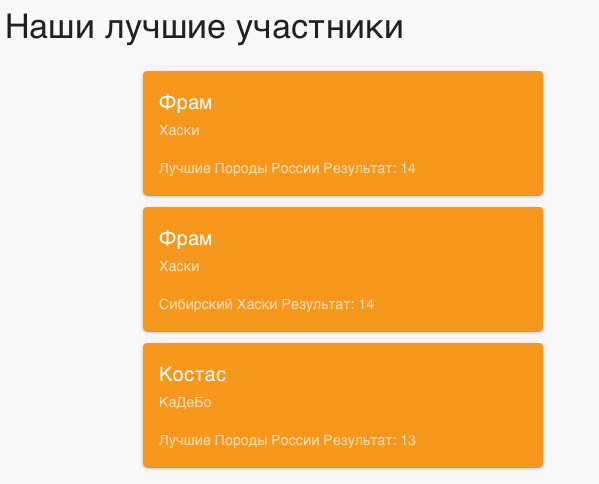


Рисунок 20 – Лучшие участники

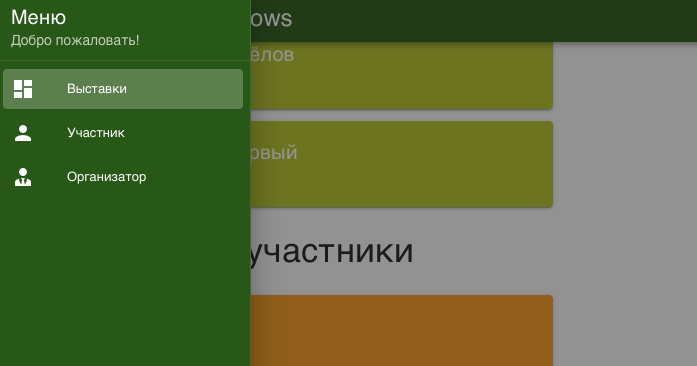


Рисунок 21 – Меню

* 1. Регистрация и авторизация

В зависимости от выбранного пункта меню, пользователь попадает на страницу авторизации либо участника, либо организатора. Следует отметить, что участник не может войти как организатор, так как выполняется проверка с помощью параметра org, принимающий значение True для организатора и False для других.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 22 – Проверка на организатора

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 23 – Авторизация участника

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 24 – Авторизация организатора

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 25 – Регистрация участника

Дополнительно можно отметить наличие кнопок «Есть аккаунт?» и «Нет аккаунта?», переводящего пользователя между страницами.

Сами логин и пароль пользователя передаются в sessionStorage, в последствии их наличие проверяется, так личные кабинеты не будут доступны по api-адресу без авторизации.



Рисунок 26 – Проверка, авторизирован ли пользователь

* 1. Личный кабинет

Участник получает данные, исключительно о своих собаках и заявках, организатор получает данные всех.

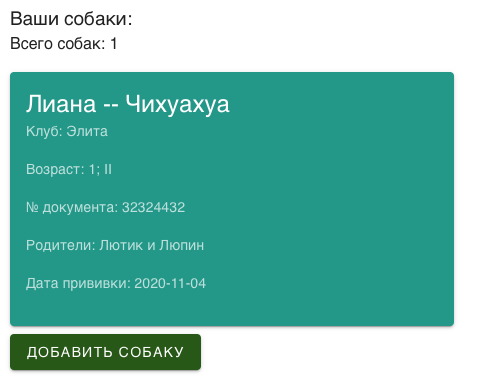


Рисунок 27 – Фрагмент личного кабинета пользователя

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 28 – Форма добавления питомца, аналогична регистрации

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 29 – Редактирование заявки участника

# ЗАКЛЮЧЕНИЕ

Цель курсовой работы состояла в реализации web-сервиса dogshow, а именно организация и упращение проведения выставок собак.

Предложенные для решения задачи технологии, а именно PostgreSQL, Django REST framework и Vue.js с Vuetify.js позволили быстро и относительно просто создать целостное web-приложение, отвечающее сформулированным функциональным требованиям.

Дальнейшее развития заключается в выведении большего количества отчётов и расширении функционала. Может быть, например, реализован чат для общения между всеми категориями пользователей.

# СПИСОК ЛИТЕРАТУРЫ

1. Django Rest Framework. Документация Django Rest Framework [Электронный ресурс]. URL: <https://www.django-rest-framework.org>
2. Vue.js. Документация Vue.js [Электронный ресурс]. URL: <https://vuejs.org>
3. Vuetify. Material Design Framework. [Электронный ресурс]. URL: <https://vuetifyjs.com/en/>
4. Axios. Документация axios. [Электронный ресурс]. URL: <https://github.com/axios/axios>

# Приложение 1. Models.py

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

# Приложение 2. Serializers.py

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

# Приложение 3. Views.py

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

# Приложение 4. Vue.js components

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание