

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО**

Дисциплина: Основы веб-программирования

Отчет по лабораторной работе №2-3
Вариант 3

**РЕАЛИЗАЦИЯ WEB-СЕРВИСОВ СРЕДСТВАМИ Django
REST framework, Vue.js**

Выполнила:
Бережнова Марина
Группа: К3343

Проверил:
Говоров А. И.

2020 г.

Тема

Создать программную систему, предназначенную для администрации аэропорта некоторой компании-авиаперевозчика.

Рейсы обслуживаются бортами, принадлежащими разным авиаперевозчикам. О каждом самолете необходима следующая минимальная информация: номер самолета, тип, число мест, скорость полета, компания-авиаперевозчик. Один тип самолета может летать на разных маршрутах и по одному маршруту могут летать разные типы самолетов.

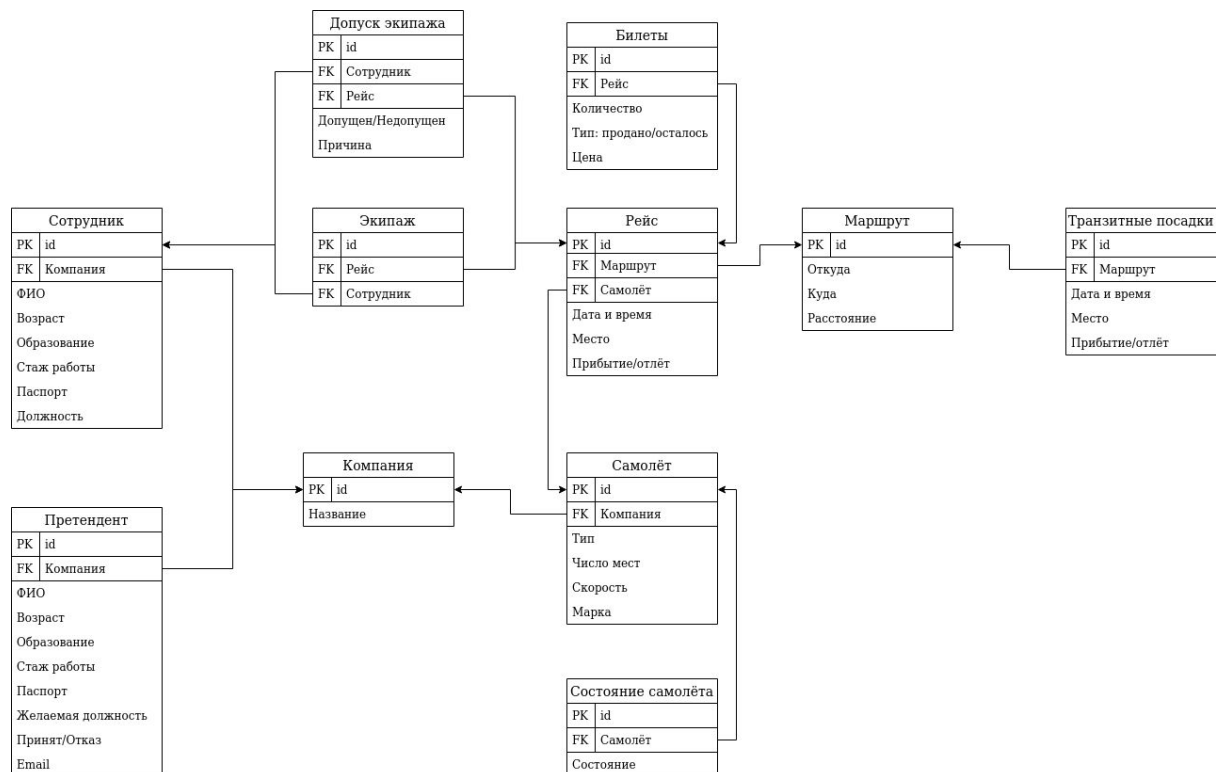
О каждом рейсе необходима следующая информация: номер рейса, расстояние до пункта назначения, пункт вылета, пункт назначения; дата и время вылета, дата и время прилета, транзитные посадки (если есть), пункты посадки, дата и время транзитных посадок и дат и время их вылета, количество проданных билетов. Каждый рейс обслуживается определенным экипажем, в состав которого входят командир корабля, второй пилот, штурман и стюардессы или стюарды. Каждый экипаж может обслуживать разные рейсы на разных самолетах. Необходимо предусмотреть наличие информации о допуске члена экипажа к рейсу. Администрация компании-владельца аэропорта должна иметь возможность принять работника на работу или уволить. При этом необходима следующая информация: ФИО, возраст, образование, стаж работы, паспортные данные. Эта же информация необходима для сотрудников сторонних компаний

Перечень возможных запросов:

- Выбрать марку самолета, которая чаще всего летает по маршруту.
- Выбрать маршрут/маршруты, по которым летают рейсы, заполненные менее чем на 70%.
- Определить наличие свободных мест на заданный рейс.
- Определить количество самолетов, находящихся в ремонте.
- Определить количество работников компания-авиаперевозчика.

Необходимо предусмотреть возможность получения отчета о бортах компании-владельца по маркам с характеристикой марки. Указать общее количество бортов и количество бортов по каждой марке.

Модель БД



- Компания (id, Название)
- Самолёт (id (он же номер), Тип, Число мест, Скорость полета, Компания, Марка)
- Состояние самолёта (id, Самолёт, Тип: в работе/в ремонте)
- Маршрут (id, Откуда, Куда, Расстояние)
- Пересадки/транзитные посадки (id, Маршрут, Дата и время, Место, Тип: прибытие/отлёт)
- Рейсы (id (он же номер рейса), Дата и время, Самолёт, Маршрут)
- Билеты (id, Рейс, Количество, Состояние: продано/осталось, Цена)
- Сотрудник (id, Компания, ФИО, Возраст, Образование, Стаж работы, Паспортные данные, Должность)
- Экипаж рейса (id, Рейс, Сотрудник)

- Может ли быть сотрудник допущен? (id, Сотрудник, Допущен ли? Допущен/Недопущен, Причина)
- Претенденты (id, Желаемая компания, ФИО, Возраст, Образование, Стаж работы, Паспортные данные, Желаемая должность, Состояние: Принят/Отказ, email)

Интерфейсы

Фронтенд:

- Подача резюме на вакансию (форма, с полями: Желаемая компания, ФИО, Возраст, Образование, Стаж работы, Паспортные данные, Желаемая должность, email)
- Просмотр всех рейсов с различной фильтрацией (марка самолёта, количество свободных мест, цена, наличие или отсутствия пересадок)

Django admin:

- Просмотр всех резюме (список "карточек" претендентов с возможностью получить подробную информацию по каждой заявке, принять или отказать)
- Просмотр всех сотрудников компании (список "карточек" сотрудников с возможностью изменения статуса работника: увольнение, недопуск до рейса)
- Добавление рейса (выбор маршрута, самолёта, дата и время, количество билетов)
- Добавление маршрута
- Добавление самолёта (с возможностью изменять в дальнейшем его состояние)
- Просмотр всех самолётов

Выполнение работы

Лабораторная работа 2. Бэкенд.

Создаём проект django в отдельной директории со своим виртуальным окружением, в которое устанавливаем для начала только django и django rest framework.

Настраиваем (кусочек из settings.py):

Application definition

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'rest_framework',  
    'airport'  
]
```

Теперь расписываем models.py и регистрируем все модели в admin.py и проверяем в админке:

Site administration

AIRPORT		
Challengerss	+ Add	✎ Change
Companyss	+ Add	✎ Change
Emloyee states	+ Add	✎ Change
Employeeess	+ Add	✎ Change
Flight teams	+ Add	✎ Change
Flightss	+ Add	✎ Change
Plane states	+ Add	✎ Change
Planes	+ Add	✎ Change
Routes	+ Add	✎ Change
Ticketss	+ Add	✎ Change
Transit pointss	+ Add	✎ Change
AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	✎ Change
Users	+ Add	✎ Change

Создадим view для получения всех полётов:

```
class FlightsView(APIView):
    def get(self, request):
        flights = Flights.objects.all()

        return Response({"flights": flights})
```

Создаём теперь urls.py для нашего приложения:

```
from django.urls import path

from .views import FlightsView
```

```
app_name = "airport"
```

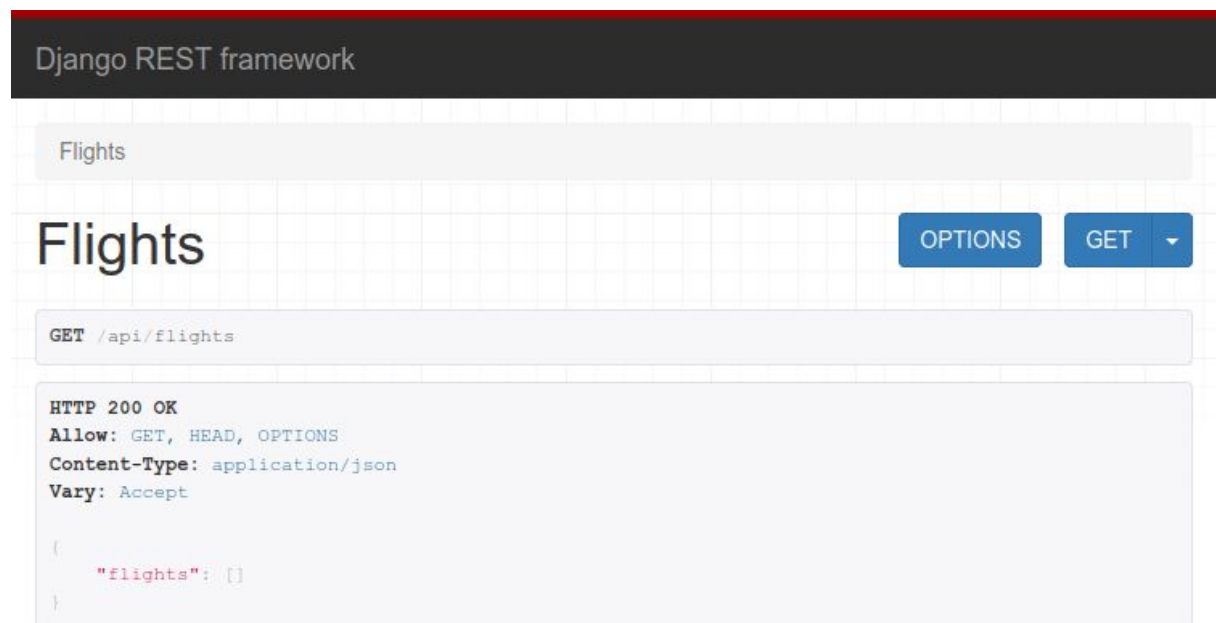
```
urlpatterns = [  
    path('flights', FlightsView.as_view())  
]
```

Теперь нам нужно подключить этот файл в основной urls.py:

```
from django.contrib import admin  
from django.urls import path, include
```

```
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('api/', include('airport.urls'))  
]
```

Проверяем теперь работу:



Всё работает.

Для того, чтобы данные отображались по этому запросу нужно сделать ещё serializer'ы:

```
# airport/serializers.py
from rest_framework import serializers

class FlightsSerializer(serializers.Serializer):
    date = serializers.DateField()
    route = serializers.CharField()
    plane = serializers.CharField()
```

Код views (теперь он такой):

```
# from django.shortcuts import render
from rest_framework.response import Response
from rest_framework.views import APIView

from .models import Flights
from .serializers import FlightsSerializer
# Create your views here.
```

```
class FlightsView(APIView):
    def get(self, request):
        flights = Flights.objects.all()
        serializer = FlightsSerializer(flights, many=True)

        return Response({"flights": serializer.data})
```

Проверяем:

Flights

Flights

OPTIONS

GET

GET /api/flights

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
  "flights": [
    {
      "date": "2020-05-21",
      "route": "From Saint-Petersburg to Kazan. Distance 1500",
      "plane": "S7: 2 plane Airbus-3000"
    }
  ]
}
```

Как видим, информация выводится. Желательно добавить отдельные serializer'ы, чтобы выводилась более подробная информация по маршрутам и самолётам (в плане, чтобы она была разделена по тем же полям, что и в модели).

Обновила:

```
from rest_framework import serializers
from .models import *
```

```
class RouteSerializer(serializers.ModelSerializer):
    class Meta:
        model = Route
        fields = '__all__'
```

```
class PlaneSerializer(serializers.ModelSerializer):  
    class Meta:  
        model = Plane  
        fields = '__all__'
```

```
class FlightsSerializer(serializers.Serializer):  
    id = serializers.IntegerField()  
    date = serializers.DateField()  
    route = RouteSerializer()  
    plane = PlaneSerializer()
```

Теперь ответ приходит вот в таком виде:

Flights

Flights

OPTIONS

GET ▾

GET /api/flights

HTTP 200 OK

Allow: GET, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{
  "flights": [
    {
      "id": 1,
      "date": "2020-05-21",
      "route": {
        "id": 1,
        "from_point": "Saint-Petersburg",
        "to_point": "Kazan",
        "distance": 1500
      },
      "plane": {
        "id": 1,
        "plane_type": "2",
        "seats_amonut": 400,
        "speed": 400,
        "model": "Airbus-3000",
        "company": 1
      }
    }
  ]
}
```

Но мне кажется, что этого всё ещё недостаточно, нужно бы вывести ещё экипаж на данный полёт, количество билетов, все транзитные посадки.

Дописала:

```
class FlightTeamSerializer(serializers.Serializer):
```

```
    id = serializers.IntegerField()
```

```
    employee = EmployeesSerializer()
```

```
    flight = FlightsSerializer()
```

```
class TransitPointsSerializer(serializers.Serializer):
    id = serializers.IntegerField()
    date = serializers.DateField()
    action = serializers.CharField()
    point = serializers.CharField()
    route = RouteSerializer()
```

view переписала вот так:

```
class FlightsView(APIView):
    def get(self, request):
        flights = Flights.objects.all()
        flights_serializer = FlightsSerializer(flights, many=True)

        team = FlightTeam.objects.all()
        team_serializer = FlightTeamSerializer(team, many=True)

        transit_points = TransitPoints.objects.all()
        transit_points_serializer = TransitPointsSerializer(transit_points,
many=True)

        return Response({"flights": flights_serializer.data, "transit_points":
transit_points_serializer.data, "teams": team_serializer.data})
```

Результат:

HTTP 200 OK

Allow: GET, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{
  "flights": [
    {
      "id": 1,
      "date": "2020-05-21",
      "route": {
        "id": 1,
        "from_point": "Saint-Petersburg",
        "to_point": "Kazan",
        "distance": 1500
      },
      "plane": {
        "id": 1,
        "plane_type": "2",
        "seats_amonut": 400,
        "speed": 400,
        "model": "Airbus-3000",
        "company": 1
      }
    }
  ],
  "transit_points": [
    {
      "id": 1,
      "date": "2020-05-21",
      "action": "1",
      "point": "Moscow",
      "route": {
        "id": 1,
        "from_point": "Saint-Petersburg",
        "to_point": "Kazan",
        "distance": 1500
      }
    }
  ]
}
```

```

    {
      "id": 2,
      "date": "2020-05-22",
      "action": "2",
      "point": "Moscow",
      "route": {
        "id": 1,
        "from_point": "Saint-Petersburg",
        "to_point": "Kazan",
        "distance": 1500
      }
    }
  ],
  "teams": [
    {
      "id": 1,
      "employee": {
        "id": 1,
        "first_name": "Ivan",
        "last_name": "Ivanov",
        "middle_name": "Ivanovich",
        "age": 35,
        "experience": 10,
        "passport": 10101010,
        "position": "Captain",
        "company": 1
      },
      "flight": {
        "id": 1,
        "date": "2020-05-21",
        "route": {
          "id": 1,
          "from_point": "Saint-Petersburg",
          "to_point": "Kazan",
          "distance": 1500
        }
      },
      "plane": {
        "id": 1,
        "plane_type": "2",
        "seats_amonut": 400,
        "speed": 400,
        "model": "Airbus-3000",
        "company": 1
      }
    }
  ]
},

```

```

{
  "id": 2,
  "employee": {
    "id": 2,
    "first_name": "Masha",
    "last_name": "Vasileva",
    "middle_name": "Ivanovna",
    "age": 23,
    "experience": 2,
    "passport": 1010010101,
    "position": "Steward",
    "company": 1
  },
  "flight": {
    "id": 1,
    "date": "2020-05-21",
    "route": {
      "id": 1,
      "from_point": "Saint-Petersburg",
      "to_point": "Kazan",
      "distance": 1500
    },
    "plane": {
      "id": 1,
      "plane_type": "2",
      "seats_amonut": 400,
      "speed": 400,
      "model": "Airbus-3000",
      "company": 1
    }
  }
}

```

Теперь мы получаем максимально полную информацию по рейсам.
Осталось только поправить choices и добавить фильтрацию.

Итоговый код:

```

class FlightsView(APIView):
    def get(self, request):
        params = request.query_params

        flight_id = params.get('id')
        from_point = params.get('from_point', "")
        to_point = params.get('to_point', "")
        plane_model = params.get('plane_model', "")
        price = params.get('price', 0)
        ticket_left = params.get('tickets_left', 0)

```

```
is_transit = params.get('is_transit', 'yes')
```

```
flights = Flights.objects.all()
```

```
team = FlightTeam.objects.all()
```

```
transit_points = TransitPoints.objects.all()
```

```
tickets = Tickets.objects.all()
```

```
if flight_id:
```

```
    flights = flights.filter(id=flight_id)
```

```
    team = team.filter(flight=flight_id)
```

```
    route_id = flights[0].route.id if len(flights) > 0 else 0
```

```
    transit_points = transit_points.filter(route=route_id)
```

```
    tickets = tickets.filter(flight=flight_id)
```

```
if from_point or to_point:
```

```
    routes = Route.objects.all().filter(Q(from_point=from_point) |  
Q(to_point=to_point))
```

```
    route_ids = [route.id for route in routes]
```

```
    flights = flights.filter(route__in=route_ids)
```

```
    flight_ids = [flight.id for flight in flights]
```

```
    team = team.filter(flight__in=flight_ids)
```

```
    transit_points = transit_points.filter(route__in=route_ids)
```

```
    tickets = tickets.filter(flight__in=flight_ids)
```

```
if plane_model:
```

```
    planes = Plane.objects.all().filter(model=plane_model)
```

```
    plane_ids = [plane.id for plane in planes]
```

```
    flights = flights.filter(plane__in=plane_ids)
```

```
    flight_ids = [flight.id for flight in flights]
```

```
    route_ids = [flight.route.id for flight in flights]
```

```
    team = team.filter(flight__in=flight_ids)
```

```
    transit_points = transit_points.filter(route__in=route_ids)
```

```
    tickets = tickets.filter(flight__in=flight_ids)
```

```
if is_transit == 'no':
```



```
route_ids = [point.route.id for point in transit_points]
routes = Route.objects.all().exclude(id__in=route_ids)
flights = flights.exclude(route__in=route_ids)
flight_ids = [flight.id for flight in flights]
team = team.filter(flight__in=flight_ids)
transit_points = transit_points.exclude(route__in=route_ids)
tickets = tickets.filter(flight__in=flight_ids)
```

```
if int(price) > 0:
```

```
    tickets = tickets.filter(Q(price__lte=price) & Q(state='2'))
    flight_ids = [ticket.flight.id for ticket in tickets]
    flights = flights.filter(id__in=flight_ids)
    team = team.filter(flight__in=flight_ids)
    route_ids = [flight.route.id for flight in flights]
    transit_points = transit_points.filter(route__in=route_ids)
```

```
if int(ticket_left) > 0:
```

```
    tickets = tickets.filter(Q(amount__lte=ticket_left) & Q(state='2'))
    flight_ids = [ticket.flight.id for ticket in tickets]
    flights = flights.filter(id__in=flight_ids)
    team = team.filter(flight__in=flight_ids)
    route_ids = [flight.route.id for flight in flights]
    transit_points = transit_points.filter(route__in=route_ids)
```

```
flights_serializer = FlightsSerializer(flights, many=True)
team_serializer = FlightTeamSerializer(team, many=True)
transit_points_serializer = TransitPointsSerializer(transit_points,
many=True)
tickets_serializer = TicketsSerializer(tickets, many=True)
```

```
return Response({
    "flights": flights_serializer.data,
    "transit_points": transit_points_serializer.data,
    "teams": team_serializer.data,
    "tickets": tickets_serializer.data
```

```
}}
```

Так же добавила по аналогии serializer для билетов.

Теперь нужно сделать view для регистрации претендентов.

```
class ChallengersView(APIView):
    def post(self, request):
        challenger = request.get('challenger')

        serializer = ChallengersSerializer(data=challenger)

        if serializer.is_valid(raise_exception=True):
            challenger_saved = serializer.save()

            return Response({"Success": "Challenger '{}' created
succesfully.".format(challenger_saved.last_name)})
```

serializer:

```
class ChallengersSerializer(serializers.ModelSerializer):
    class Meta:
        model = Challengers
        fields = '__all__'

    def create(self, validated_data):
        return Challengers.objects.create(**validated_data)
```

Проверяем:

Django REST frameworkmarina

Challengers

Challengers

OPTIONS

GET /api/challengers

HTTP 405 Method Not Allowed
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "detail": "Method \"GET\" not allowed."
}
```

Media type: application/json

Content:

POST

Попробуем выполнить запрос:

Media type: application/json

Content:

```
{
  "challenger": {
    "company": 1,
    "first_name": "Vasily",
    "last_name": "Petrov",
    "middle_name": "Vladimirovich",
    "age": 28,
    "experience": 3,
    "passport": 4141414141,
    "position": "Captain",
    "state": "2",
    "email": "valisiy@bk.ru"
  }
}
```

POST

Всё работает:

Challengers

Challengers

OPTIONS

POST /api/challengers

```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
  "Success": "Challenger 'Petrov' created succesfully."
}
```

Проверяем в админке:










Select challengers to change

Action: 0 of 1 selected

**CHALLENGERS****S7: Petrov Vasiliy Vladimirovich, Captain | Rejection**

1 challengers

Change challengers

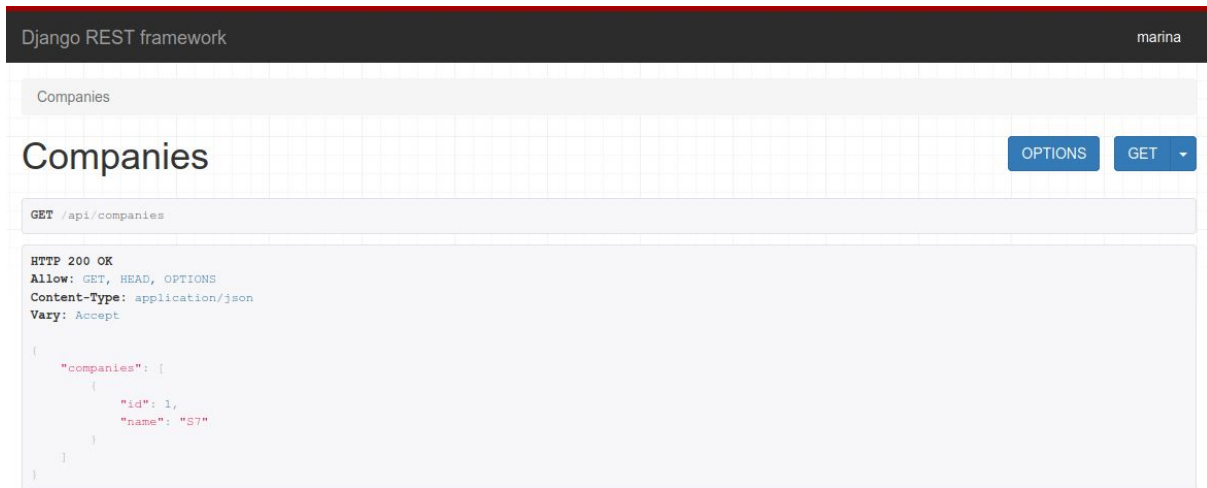
Company:	<div>S7  </div>
First name:	<input type="text" value="Vasiliy"/>
Last name:	<input type="text" value="Petrov"/>
Middle name:	<input type="text" value="Vladimirovich"/>
Age:	<div>28  </div>
Experience:	<div>3  </div>
Passport:	<div>4141414  </div>
Position:	<input type="text" value="Captain"/>
State:	<div>Rejection </div>
Email:	<input type="text" value="valisiy@bk.ru"/>

Delete

Endpoint для получения всех компаний:

```
class CompaniesView(APIView):  
    def get(self, request):  
        companies = Company.objects.all()  
        companies_serializer = CompanySerializer(companies, many=True)  
  
        return Response({"companies": companies_serializer.data})
```

Проверка:



Также нужно разрешить CORS-запросы. Для этого установим библиотеку `django-cors-headers` и пропишем все настройки в `settings.py` согласно их `readme`.

Лабораторная работа 2 на этом выполнена. Реализованы все заявленные интерфейсы, которые будут использоваться на фронтенде.

Приступим к разработке фронтенда.

Лабораторная работа 3. Фронтенд.

В качестве UI-библиотеки я решила использовать `Vuetify`, а для запросов `axios`.

Инициализация проекта:

```
vue init webpack airport
```

```
? Project name airport
```

```
? Project description A Vue.js project
```

```
? Author Marina Berezhnova
```

```
? Vue build standalone
```

? Install vue-router? Yes
? Use ESLint to lint your code? Yes
? Pick an ESLint preset Standard
? Set up unit tests No
? Setup e2e tests with Nightwatch? No
? Should we run `npm install` for you after the project has been created?
(recommended) npm

Проект проинициализирован. Перейдем к установке необходимых зависимостей:

```
npm i axios vue-axios --save-dev
```

```
npm i vuetify --save-dev
```

```
npm i sass sass-loader fibers deepmerge -D
```

В `webpack.base.conf.js` дописываем в массив `rules`:

```
{
  test: /\.s(c|a)ss$/,
  use: [
    'vue-style-loader',
    'css-loader',
    {
      loader: 'sass-loader',
      // Requires sass-loader@^8.0.0
      options: {
        implementation: require('sass'),
        sassOptions: {
          fiber: require('fibers'),
          indentedSyntax: true // optional
        },
      },
    },
  ],
}
```

```
  ],  
}
```

В папке `src/plugins` создаём файл `vuetify.js` и пишем в него следующее:

```
import Vue from 'vue'  
import Vuetify from 'vuetify'  
import 'vuetify/dist/vuetify.min.css'  
import '@mdi/font/css/materialdesignicons.css'
```

```
Vue.use(Vuetify)
```

```
const opts = { icons: { iconfont: 'mdi' } }
```

```
export default new Vuetify(opts)
```

Написала несколько компонент и создала подготовительную разметку:

Откуда Куда Цена билета

Модель самолёта Количество оставшихся билетов ☐ Нет транзитных точек

ОТФИЛЬТРОВАТЬ ОЧИСТИТЬ

S7: Санкт-Петербург - Казань

Осталось 400 билетов

6000 руб./билет

ПОДРОБНЕЕ

Запрос на получение всех полётов:

```
this.axios  
  .get('http://127.0.0.1:8000/api/flights')  
  .then(response => { this.flightsData = response; this.updateCards() })  
  .catch(error => { console.error(error) })
```

Вывела на страницу:


```
{
  "data": {
    "flights": [
      {
        "id": 1,
        "date": "2020-05-21",
        "route": {
          "id": 1,
          "from_point": "Saint-Petersburg",
          "to_point": "Kazan",
          "distance": 1500
        },
        "plane": {
          "id": 1,
          "plane_type": "Passenger",
          "seats_amonut": 400,
          "speed": 400,
          "model": "Airbus-3000",
          "company": 1
        },
        "transit_points": [
          {
            "id": 1,
            "date": "2020-05-21",
            "action": "Arrival",
            "point": "Moscow",
            "route": {
              "id": 1,
              "from_point": "Saint-Petersburg",
              "to_point": "Kazan",
              "distance": 1500
            }
          },
          {
            "id": 2,
            "date": "2020-05-22",
            "action": "Departure",
            "point": "Moscow",
            "route": {
              "id": 1,
              "from_point": "Saint-Petersburg",
              "to_point": "Kazan",
              "distance": 1500
            }
          }
        ],
        "teams": [
          {
            "id": 1,
            "employee": {
              "id": 1,
              "first_name": "Ivan",
              "last_name": "Ivanov",
              "middle_name": "Ivanovich",
              "age": 35,
              "experience": 10,
              "passport": 10101010,
              "position": "Captain",
              "company": 1
            },
            "flight": {
              "id": 1,
              "date": "2020-05-21",
              "route": {
                "id": 1,
                "from_point": "Saint-Petersburg",
                "to_point": "Kazan",
                "distance": 1500
              },
              "plane": {
                "id": 1,
                "plane_type": "Passenger",
                "seats_amonut": 400,
                "speed": 400,
                "model": "Airbus-3000",
                "company": 1
              }
            },
            "id": 2,
            "employee": {
              "id": 2,
              "first_name": "Masha",
              "last_name": "Vasileva",
              "middle_name": "Ivanovna",
              "age": 23,
              "experience": 2,
              "passport": 1010010101,
              "position": "Stuard",
              "company": 1
            },
            "flight": {
              "id": 1,
              "date": "2020-05-21",
              "route": {
                "id": 1,
                "from_point": "Saint-Petersburg",
                "to_point": "Kazan",
                "distance": 1500
              },
              "plane": {
                "id": 1,
                "plane_type": "Passenger",
                "seats_amonut": 400,
                "speed": 400,
                "model": "Airbus-3000",
                "company": 1
              }
            }
          }
        ],
        "tickets": [
          {
            "id": 1,
            "state": "Left",
            "flight": {
              "id": 1,
              "date": "2020-05-21",
              "route": {
                "id": 1,
                "from_point": "Saint-Petersburg",
                "to_point": "Kazan",
                "distance": 1500
              },
              "plane": {
                "id": 1,
                "plane_type": "Passenger",
                "seats_amonut": 400,
                "speed": 400,
                "model": "Airbus-3000",
                "company": 1
              }
            },
            "amount": 400,
            "price": 6000
          }
        ],
        "status": 200,
        "statusText": "OK",
        "headers": {
          "content-type": "application/json"
        },
        "config": {
          "url": "http://127.0.0.1:8000/api/flights",
          "method": "get",
          "headers": {
            "Accept": "application/json, text/plain, */*"
          },
          "transformRequest": [ null ],
          "transformResponse": [ null ],
          "timeout": 0,
          "xsrfCookieName": "XSRF-TOKEN",
          "xsrfHeaderName": "X-XSRF-TOKEN",
          "maxContentLength": -1
        },
        "request": {}
      }
    ]
  }
}
```

Все данные получены успешно. Теперь напишем функцию `updateCards`, чтобы она обновляла всю информацию по каждому рейсу. Результат работы:

S7: Saint-Petersburg - Kazan

Осталось 400 билетов | Есть пересадки

6000 руб./билет

[ПОДРОБНЕЕ](#)

Теперь сделаем фильтрацию.

Пример работы:

Откуда Saint-Petersburg	Куда	Цена билета 0
Модель самолёта	Количество оставшихся билетов 0	<input type="checkbox"/> Нет транзитных точек
<div>▼ ФИЛЬТРОВАТЬ</div> <div>↺ ОЧИСТИТЬ</div>		

S7: Saint-Petersburg - Kazan

Осталось 400 билетов | Есть пересадки

6000 руб./билет

[ПОДРОБНЕЕ](#)

Откуда Saint-Petersburg	Куда	Цена билета 0
Модель самолёта	Количество оставшихся билетов 0	<input checked="" type="checkbox"/> Нет транзитных точек

▼ ОФИЛЬТРОВАТЬ

↺ ОЧИСТИТЬ

Извините, по вашему запросу ничего не найдено...

Осталась только подача заявки.

≡

✈ Airport

Фамилия
Васильев

Имя
Константин

Отчество
Фёдорович

Возраст
45

Стаж работы
20

Паспорт
41145678789

Желаемая должность
Captain

Email
konst@mail.ru










Компания
S7

SUBMIT

CLEAR

Проверяю:

Change challengers

Company:	<div>S7</div>  
First name:	<div>Константин</div>
Last name:	<div>Васильев</div>
Middle name:	<div>Фёдорович</div>
Age:	<div>45</div>  
Experience:	<div>20</div>  
Passport:	<div>4114567</div>  
Position:	<div>Captain</div>
State:	<div>Rejection</div> 
Email:	<div>konst@mail.ru</div>

Delete

На этом лабораторную работу 3 считаю выполненной.

Вывод: выполнив данные лабораторные работы, я научилась создавать REST API, используя DRF и SPA-приложение на Vue.js.