REPORT03-DJANGO

On our 6th laboratories we moved on to creating ToDoList web app which was our first approach to create a Django app.

INTRODUCTION

What is Django?

Django is a Python Web framework which is very popular among web-application developers. It's one of the most recognised web development frameworks. It enables clear coding and provides plenty common functionalities for web development which saves from writing code for each function. It just that simplifies common practices in web development. Django is also known for it's simplicity in working with databases.

Django offers variety of patterns: creational, structural and behavioral.?

Pros and Cons of Django

+ implemented in Python

+ clean design?

+ readymade packages

+ offers a flexible way of separating context

+ excellent support for external Python libraries and packages

- problems with handling multilpe requests simultaneously

- not right choice for micro-sites (one-pagers etc.)

What is a Django Model?

A Django model is?the built-in feature that Django uses to create tables, their fields, and various constraints. Models are created in /myapp/models.py file. Django uses SQL as a default tool. Each model represents single database table. Whenever creating or updating a model we need to run command *make migrations?*and?*migrate.*?

What are serializers?

Serializers allow complex data such as querysets and model instances to be converted to native Python datatypes that can then be easily rendered into json or xml?or other content types. The main idea is translate our data into a certain format and to pass that information over the internet and then be consumed by frontend (eg.). Rest Framework provides both, Serializers and ModelSerializers. The difference is simple: Serializer is the normal serializer that will be used when building an API with Django. It simply parses data from complex types into JSON or XML; the Model Serializer is just the same as the? regular one in the sense that it does the same job, only that it builds the serializer based on the model.

Django REST Framework

Django Rest Framework is a specialized, open source Python framework which is used for building Web APIs. It's an easy and powerfull toolkit that is used by web app developers. It's biggest advantage is that it makes model serialization easy. The API provides handling database queries and appropriate formatting of responses, which are returned to a static file. ?REST APIs let us send information back and forth between an interface and a database.

Django REST Framework + REACT

React is a popular JavaScript framework for creating front-end applications, such as user interfaces that allow users to interact with programs. It's way of rendereing is making websites highly dynamic and responsive to user input. It's easy to learn and use beacuse of a documentation, tutorials and training reasources. With Django REST Framework and REACT we were able to launch a simple application on one laboratories.?

We started our laboratories with . We downloaded the repository with backed project - which was already set-up for us. Each application writen in Django consists of a "starting package" that follows a certain convention. This convention is really helpfull beacuse everybody has the same structure of an app. After downloading the repo, we moved on to familiarizing with Django backend. Our project consisted of many files. Some of them are described below:

- manage.py
- __init__.py:
- settings.py: a file that includes default settings for the project,?
- urls.py: a file that stores URL patterns and in which additional links might be included
- wsgi.py:?
- /myapp: a folder in which our source code is stored?

We added one line: 'organizer' to settings.py file in the section of INSTALLED_APPS. Then we run our server with command: *python manage.py runserver*. Our next task was to create a model for our organizer. We defined to models: Author (which contains name and email of an author) and ToDoItem (which contains title, description, athor, deadline and date of creation of a task to do).?

Models.py

```
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=30)
    email = models.EmailField()

    def __str__(self):
        return self.name

class ToDoItem(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    author=models.ForeignKey(Author,on_delete=models.CASCADE,null=True)
    deadline=models.DateField(null=True)
    date_of_creation=models.DateField(null=True)

    def __str__(self):
        """The ToDo item model."""
        return self.title
```

As I mentioned in the introduction, we needed to run command *makemigrations?*and?*migrate* beacuse we created new database's tables. ?To interact with the data we had to modify?*admin.py?*file and import our models.?

Admin.py

```
from django.contrib import admin
from .models import Author,ToDoItem
admin.site.register(Author)
admin.site.register(ToDoItem)
```

We created a superuser account by running a command *python manage.py createsuperuser*. After that we ran our server and navigated to ?http://127.0.0.1:8000/admin/ site. We inserted data into Authors and ToDoItems.

//wrzucic tests.py - zapytac sie

Then we moved on to creating an API. The goal was to be able to send the data so someone else can use it. While we were installing djangorestframework , we added 'rest_framework' to INSTALLED_APPS in settings.py file. Then we updated urls.py file. Since then our list of Tasks To Do will we available at ?http://127.0.0.1:8000/list url.? After that we concetrated on serializers. Once we wanted our URL endpoints to return data in a JSON format, it meant we needed serializers. We created two classes AuthorSerializer and ToDoItemSerializer.?

Serializers.py

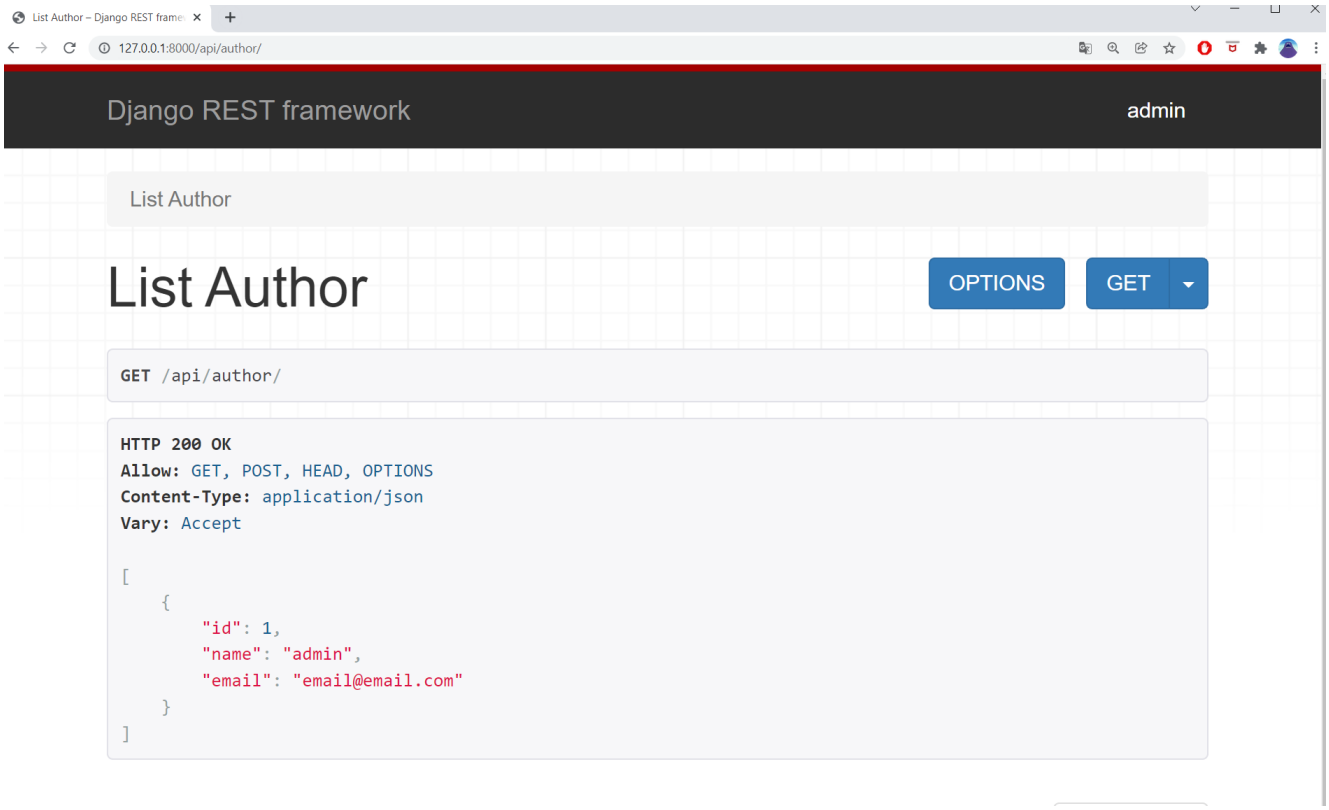
```
from rest_framework import serializers
from .models import Author, ToDoItem
class AuthorSerializer(serializers.ModelSerializer):
    class Meta:
        # Specify which model to use and
        # the specific fields on it we want to expose:
        fields = (
            'id', # created automatically by Django! (Not present in Author model)
            'name',
            'email',
        )
        model = Author
class ToDoItemSerializer(serializers.ModelSerializer):
    class Meta:
        fields=(
            'title',
            'description',
            'author',
            'deadline',
            'date_of_creation',
        )
        model=ToDoItem
```

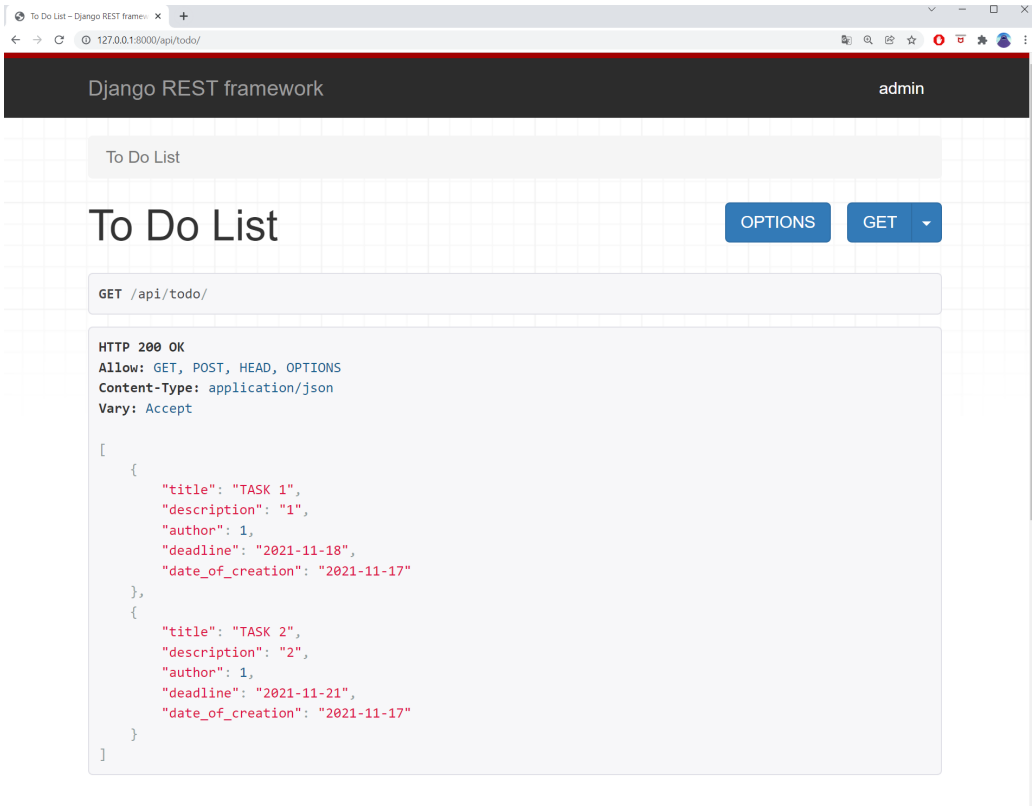
Then we moved to views.py file to create three classes: ListAuthor, DetailAuthor and ToDoList. We did it to customize which serialized data will be send to the templates.

Views.py

```
from django.shortcuts import render

from rest_framework import generics
from .models import Author, ToDoItem
from .serializers import AuthorSerializer, ToDoItemSerializer

class ListAuthor(generics.ListCreateAPIView):
    queryset = Author.objects.all()
    serializer_class = AuthorSerializer

class DetailAuthor(generics.RetrieveUpdateDestroyAPIView):
    queryset = Author.objects.all()
    serializer_class = AuthorSerializer

class ToDoList(generics.ListCreateAPIView):
    queryset = ToDoItem.objects.all()
    serializer_class = ToDoItemSerializer ? ? ?
```

We provided url paths for displaying List of Authors and List of ToDoItems. We ran the local server machine and navigated to the http://127.0.0.1:8000/api/author and http://127.0.0.1:8000/api/todo .?

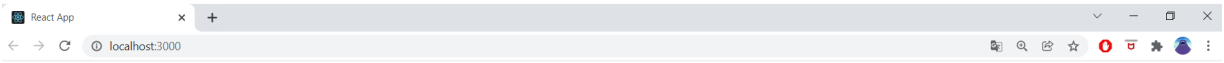




Everything described above is a part of backend. To make our app user-friendly we needed to create a fronted. We downloaded all necessary tools (nodejs, npm) and created a react-app. We started our react app with the command?*npm start*. We modified App.js file that it could display tasks from backend urls. I also created a table in App.css for tasks.

Serializers.py

```
import './App.css';
import React, { Component } from 'react';

class App extends Component {
  state = {
    todos: []
  };

  async componentDidMount() {
    try {
      const res = await fetch('http://127.0.0.1:8000/api/todo');
      const author = await fetch('http://127.0.0.1:8000/api/author');
      const todos = await res.json();
      this.setState({
        todos
      });
    } catch (e) {
      console.log(e);
    }
  }

  render() {
    return (
      <table>
        {this.state.todos.map(item => (
          <tr key={item.id}>
            <tr>{item.title}</tr>
            <th>Author: {item.author}</th>
            <th>Date of creation: {item.date_of_creation}</th>
            <th>Deadline: {item.deadline}</th>
            <th>{item.description}</th>
          </tr>
        )
        )}
      </table>
    );
  }
}

export default App;
```



TASK 1	Author: 1	Date of creation: 2021-11-17	Deadline: 2021-11-18	1
TASK 2	Author: 1	Date of creation: 2021-11-17	Deadline: 2021-11-21	2

BIBLIOGRAPHY:

https://www.django-rest-framework.org/api-guide/serializers/