django
REST
framework

What is Django?

Django is a free, open source, Python-based web framework that follows the Model-View-Template (MVT) architectural pattern. It reduces the hassle of web development so that you can focus on writing your app instead of reinventing the wheel.

What is a REST API?

A REST API is a popular way for systems to expose useful functions and data. REST, which stands for representational state transfer, can be made up of one or more resources that can be accessed at a given URL and returned in various formats, like JSON, images, HTML, and more.

Why Django REST framework?

Django REST framework (DRF) is a powerful and flexible toolkit for building Web APIs. Its main benefit is that it makes serialization much easier.

Django REST framework is based on Django's class-based views, so it's an excellent option if you're familiar with Django. It adopts implementations like class-based views, forms, model validator, QuerySet, and more.

Setting up Django REST framework

Ideally, you'd want to create a virtual environment to isolate dependencies, however, this is optional. Run the command `python -m venv django_env` from inside your projects folder to create the virtual environment. Then, run `source ./django_env/bin/activate` to turn it on.

Keep in mind that you'll need to reactivate your virtual environment in every new terminal session. You'll know that it is turned on because the environment's name will become part of the shell prompt.

Navigate to an empty folder in your terminal and install Django and Django REST framework in your project with the commands below:

```
pip install django
pip install django_rest_framework
```

Create a Django project called `todo` with the following command:

```
django-admin startproject todo
```

Then, `cd` into the new `todo` folder and create a new app for your API:

```
django-admin startapp todo_api
```

Run your initial migrations of the built-in user model:

```
python manage.py migrate
```

Next, add `rest_framework` and `todo` to the `INSTALLED_APPS` inside the `todo/todo/settings.py` file:

```
# settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'todo_api'
]
```

Create a `serializers.py` and `urls.py` file in `todo/todo_api` and add new files as configured in the directory structure below:

```
├── todo
│   ├──__init__.py
│   ├── settings.py
```

```
|    ├── urls.py
├── db.sqlite3
├── manage.py
└── todo_api
     ├── admin.py
     ├── serializers.py
     ├───__init__.py
     ├── models.py
     ├── urls.py
     └── views.py
```

Be sure to include `rest_framework` and URLs as shown below in your main `urls.py` file:

```
# todo/todo/urls.py : Main urls.py
from django.contrib  import  admin
from django.urls import path, include
from todo_api import urls as todo_urls

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api-auth/', include('rest_framework.urls')),
    path('todos/', include(todo_urls)),
]
```

Next, create a superuser. We'll come back to this later:

```
python manage.py createsuperuser
```

RESTful structure: `GET` , `POST` , `PUT` , and `DELETE` methods

In a RESTful API, endpoints define the structure and usage with the `GET` , `POST` , `PUT` , and `DELETE` HTTP methods. You must organize these methods logically.

Creating models for our Django app

Let’s start by creating the model for our to-do list:

```
# todo/todo_api/models.py
from django.db import models
from django.contrib.auth.models import User

class Todo(models.Model):
    task = models.CharField(max_length = 180)
    timestamp = models.DateTimeField(auto_now_add = True, auto_now = False, blank = T
    completed = models.BooleanField(default = False, blank = True)
    updated = models.DateTimeField(auto_now = True, blank = True)
    user = models.ForeignKey(User, on_delete = models.CASCADE, blank = True, null = T

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

After creating the model, migrate it to the database.

```
python manage.py makemigrations
python manage.py migrate
```

Model serializer

To convert the `Model` object to an API-appropriate format like JSON, Django REST framework uses the `ModelSerializer` class to convert any model to serialized JSON objects:

```
# todo/todo_api/serializers.py
from rest_framework import serializers
from .models import Todo
class TodoSerializer(serializers.ModelSerializer):
    class Meta:
        model = Todo
        fields = ["task", "completed", "timestamp", "updated", "user"]
```

Creating API views in Django

In this section, we'll walk through how to create two API views, list view and detail view.

List view

The first API view class deals with the `todos/api/` endpoint, in which it handles `GET` for listing all to-dos of a given requested user and `POST` for creating a new to-do. Notice that we've added `permission_classes`, which allows authenticated users only:

```
# todo/todo_api/views.py
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from rest_framework import permissions
from .models import Todo
from .serializers import TodoSerializer

class TodoListApiView(APIView):
    # add permission to check if user is authenticated
    permission_classes = [permissions.IsAuthenticated]

    # 1. List all
    def get(self, request, *args, **kwargs):
        '''
        List all the todo items for given requested user
        '''
        todos = Todo.objects.filter(user = request.user.id)
        serializer = TodoSerializer(todos, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)

    # 2. Create
    def post(self, request, *args, **kwargs):
        '''
        Create the Todo with given todo data
        '''
        data = {
            'task': request.data.get('task'),
            'completed': request.data.get('completed'),
            'user': request.user.id
        }
        serializer = TodoSerializer(data=data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)

        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

The `GET()` method first fetches all the objects from the model by filtering with the requested user ID. Then, it serializes from the model object to a JSON serialized object. Next, it returns the response with serialized data and status as `200_OK`.

The `POST()` method fetches the requested data and adds the requested user ID in the `data` dictionary. Next, it creates a serialized object and saves the object if it's valid. If valid, it returns the `serializer.data`, which is a newly created object with status as `201_CREATED`. Otherwise, it returns the `serializer.errors` with status as `400_BAD_REQUEST`.

Create an endpoint for the class-based view above:

```
# todo/todo_api/urls.py : API urls.py
from django.conf.urls  import  url
from django.urls import path, include
from .views import (
    TodoListApiView,
```

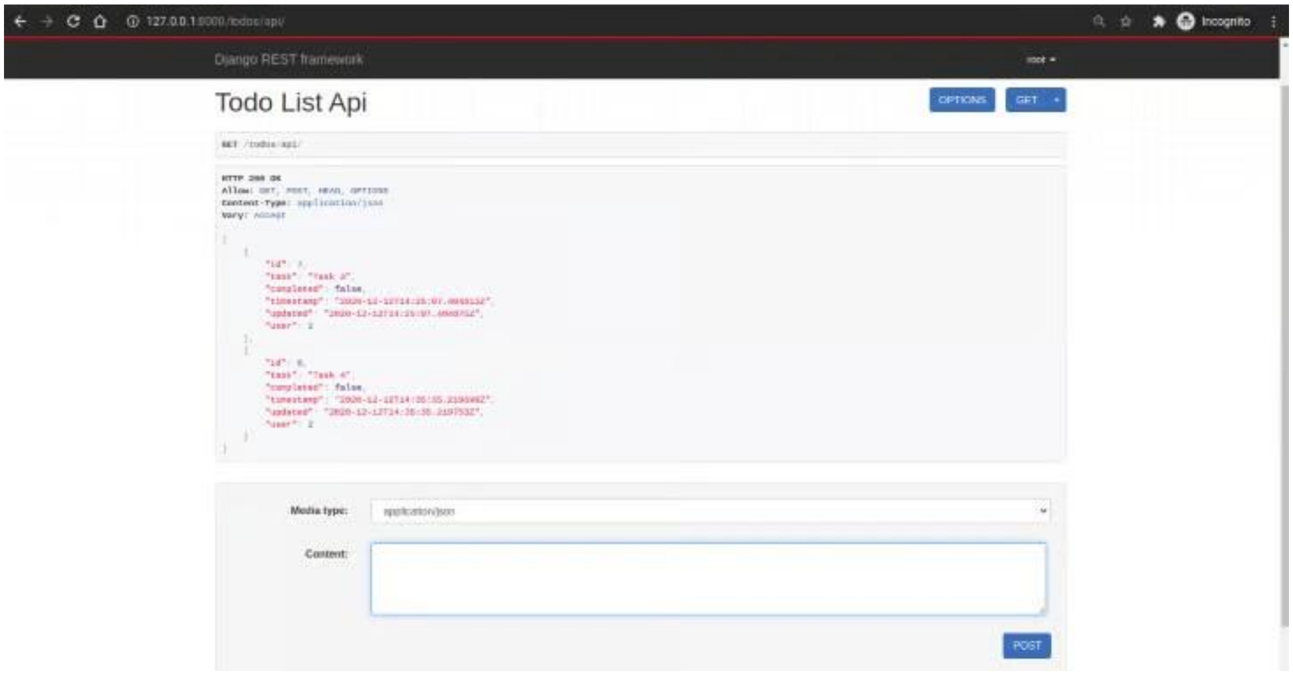
```
)

urlpatterns = [
    path('api', TodoListApiView.as_view()),
]
```

Run the Django server:

```
python manage.py runserver
```

Now, we’re ready for the first test. Navigate to `http://127.0.0.1:8000/todos/api/` . Make sure you’re logged in with your superuser credentials:



```
GET /todos/api/
```

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

[
    {
        "id": 7,
        "task": "Task 2",
        "completed": false,
        "timestamp": "2020-12-12T14:25:07.[illegible]Z",
        "updated": "2020-12-12T14:25:07.[illegible]Z",
        "user": 2
    },
    {
        "id": 8,
        "task": "Task 4",
        "completed": false,
        "timestamp": "2020-12-12T14:35:35.[illegible]Z",
        "updated": "2020-12-12T14:35:35.[illegible]Z",
        "user": 2
    }
]
```

You can create a new to-do by posting the following code:

```
{
    "task": "New Task",
    "completed": false
}
```

Detail view

Now that we’ve successfully created our first endpoint view, let’s create the second endpoint `todos/api/<int:todo_id>` API view.

In this API view class, we need to create three methods for handling the corresponding HTTP methods, `GET` , `PUT` , and `DELETE` , as discussed above:

```
# todo/api/views.py
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from todo.models import Todo
from .serializers import TodoSerializer
from rest_framework import permissions

class TodoDetailApiView(APIView):
    # add permission to check if user is authenticated
    permission_classes = [permissions.IsAuthenticated]

    def get_object(self, todo_id, user_id):
        '''
        Helper method to get the object with given todo_id, and user_id
        '''
        try:
            return Todo.objects.get(id=todo_id, user = user_id)
        except Todo.DoesNotExist:
            return None

    # 3. Retrieve
    def get(self, request, todo_id, *args, **kwargs):
        '''
        Retrieves the Todo with given todo_id
        '''
        todo_instance = self.get_object(todo_id, request.user.id)
        if not todo_instance:
            return Response(
                {"res": "Object with todo id does not exists"},
                status=status.HTTP_400_BAD_REQUEST
            )

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

    # 4. Update
    def put(self, request, todo_id, *args, **kwargs):
        '''
        Updates the todo item with given todo_id if exists
        '''
        todo_instance = self.get_object(todo_id, request.user.id)
        if not todo_instance:
            return Response(
                {"res": "Object with todo id does not exists"},
                status=status.HTTP_400_BAD_REQUEST
            )
        data = {
            'task': request.data.get('task'),
            'completed': request.data.get('completed'),
            'user': request.user.id
        }
        serializer = TodoSerializer(instance = todo_instance, data=data, partial = Tr
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_200_OK)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    # 5. Delete
    def delete(self, request, todo_id, *args, **kwargs):
        '''
        Deletes the todo item with given todo_id if exists
        '''
        todo_instance = self.get_object(todo_id, request.user.id)
        if not todo_instance:
            return Response(
                {"res": "Object with todo id does not exists"},
                status=status.HTTP_400_BAD_REQUEST
            )
        todo_instance.delete()
        return Response(
            {"res": "Object deleted!"},
            status=status.HTTP_200_OK
        )
```

The `GET()` method first fetches the object with the ID `todo_id` and user as request user from the to-do model. If the requested object is not available, it returns the response with the status as `400_BAD_REQUEST`. Otherwise, it serializes the model object to a JSON serialized object and returns the response with `serializer.data` and status as `200_OK`.

The `PUT()` method fetches the to-do object if it is available in the database, updates its data with requested data, and saves the updated data in the database.

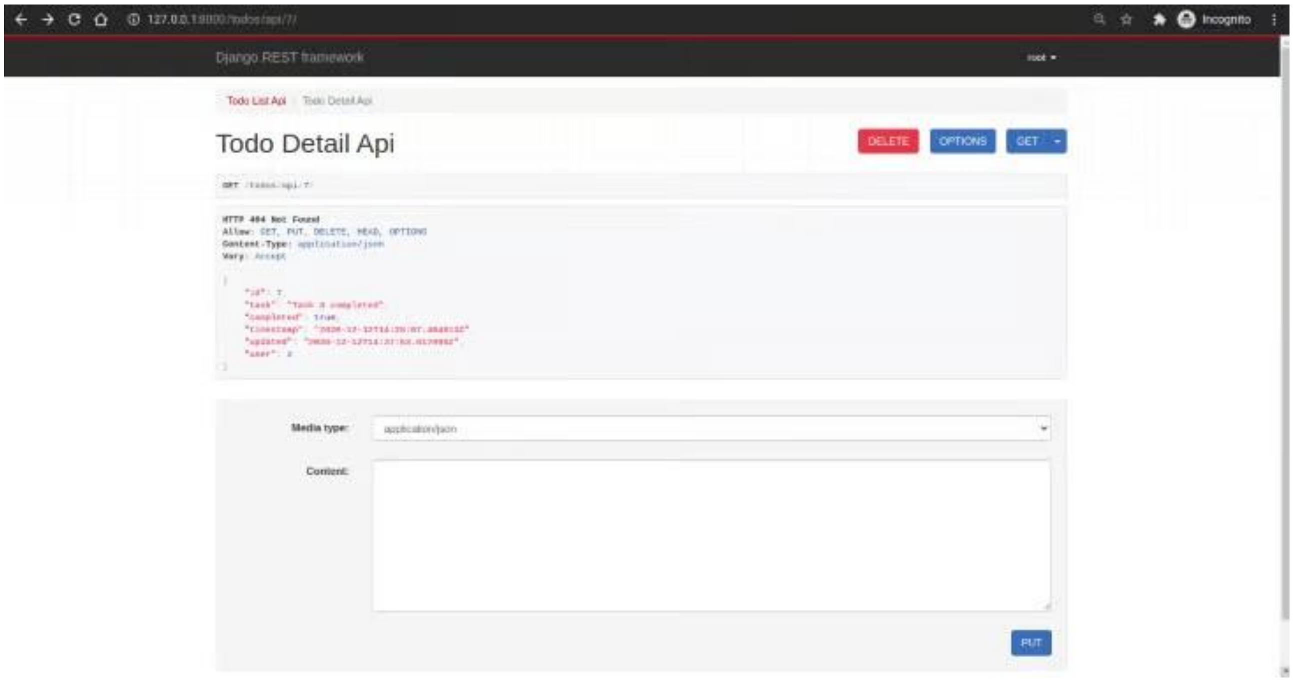
The `DELETE()` method fetches the to-do object if is available in the database, deletes it, and responds with a response.

Update the API `urls.py` as demonstrated below:

```python
# todo/api/urls.py : API urls.py
from django.conf.urls  import  url
from django.urls import path, include
from .views import (
    TodoListApiView,
    TodoDetailApiView
)

urlpatterns = [
    path('api', TodoListApiView.as_view()),
    path('api/<int:todo_id>/', TodoDetailApiView.as_view()),
]
```

Now, if you navigate to `http://127.0.0.1:8000/todos/api/<id>/` , it will show the detail API view page. Notice that you correctly navigate to a valid ID. In the screenshot below, I used `7` as the ID:



Todo Detail Api

```
GET /todos/api/7/
```

```
HTTP 404 Not Found
Allow: GET, PUT, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 7,
    "task": "Task 3 completed",
    "completed": true,
    "timestamp": "[illegible]",
    "updated": "[illegible]",
    "user": 2
}
```

Media type: application/json

Content: