Build a To-Do App with Django as Backend

This tutorial is based on the first two steps of the tutorial at https://www.digitalocean.com/community/tutorials/build-a-to-do-application-using-django-and-react

Step 1. Open a folder on VSCode

Step 2. Create a new folder called 'django-todo-backend'

```
mkdir django-todo-backend
```

Step 3. Change into this directory

```
cd django-todo-backend
```

Step 4. Create a virtual environment

```
python -m venv venv
```

Step 5. Activate the virtual environment

`venv/scripts/activate` (Windows)

or

`source venv/bin/activate` (Mac or Codespace)

Step 6. Install Django

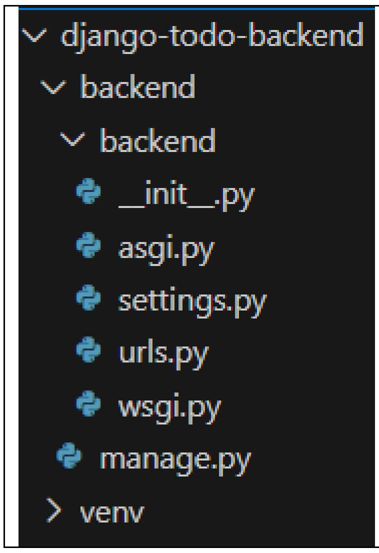
```
pip install Django
```

Step 7. Create a new project in Django

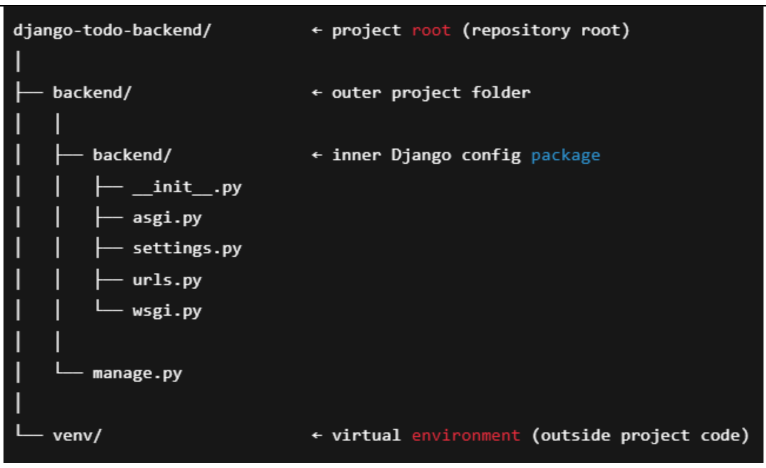
```
django-admin startproject backend
```

This will create a directory called backend (which is the name of our project). Go through the files created in this directory. For more information on the different files go to the official Django tutorials at https://docs.djangoproject.com/en/6.0/intro/tutorial01/

The folder and file structure at this stage should be as shown below



```
django-todo-backend
  backend
    backend
      __init__.py
      asgi.py
      settings.py
      urls.py
      wsgi.py
    manage.py
  venv
```



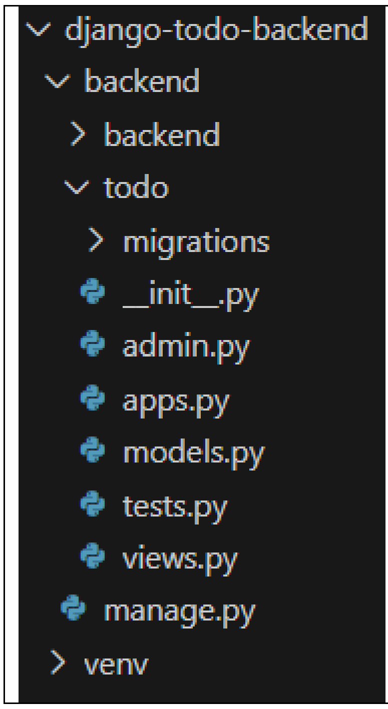
```
django-todo-backend/          ← project root (repository root)
|
├── backend/                  ← outer project folder
|   |
|   ├── backend/              ← inner Django config package
|   |   ├── __init__.py
|   |   ├── asgi.py
|   |   ├── settings.py
|   |   ├── urls.py
|   |   └── wsgi.py
|   |
|   └── manage.py
|
└── venv/                     ← virtual environment (outside project code)
```

You will only be modifying the settings.py and urls.py files.

Step 8. Create a new application (todo) in backend project

cd into the backend folder (cd backend) It is important to do this as we will need the manage.py file to run various commands/operations in Django.

Next run python manage.py startapp todo



```
django-todo-backend
  backend
    backend
    todo
      migrations
      __init__.py
      admin.py
      apps.py
      models.py
      tests.py
      views.py
    manage.py
  venv
```

Following is an extract from https://docs.djangoproject.com/en/6.0/intro/tutorial01/ on what is an application and the difference between projects and apps in Django

“Each application you write in Django consists of a Python package that follows a certain convention. Django comes with a utility that automatically generates the basic

directory structure of an app, so you can focus on writing code rather than creating directories.

Projects vs. apps

What's the difference between a project and an app? An app is a web application that does something – e.g., a blog system, a database of public records or a small poll app. A project is a collection of configuration and apps for a particular website. A project can contain multiple apps. An app can be in multiple projects."

Remember in the SQA module and the Flask todo/blog application, we had to manually create this structure for every component whereas Django creates this structure automatically for us.

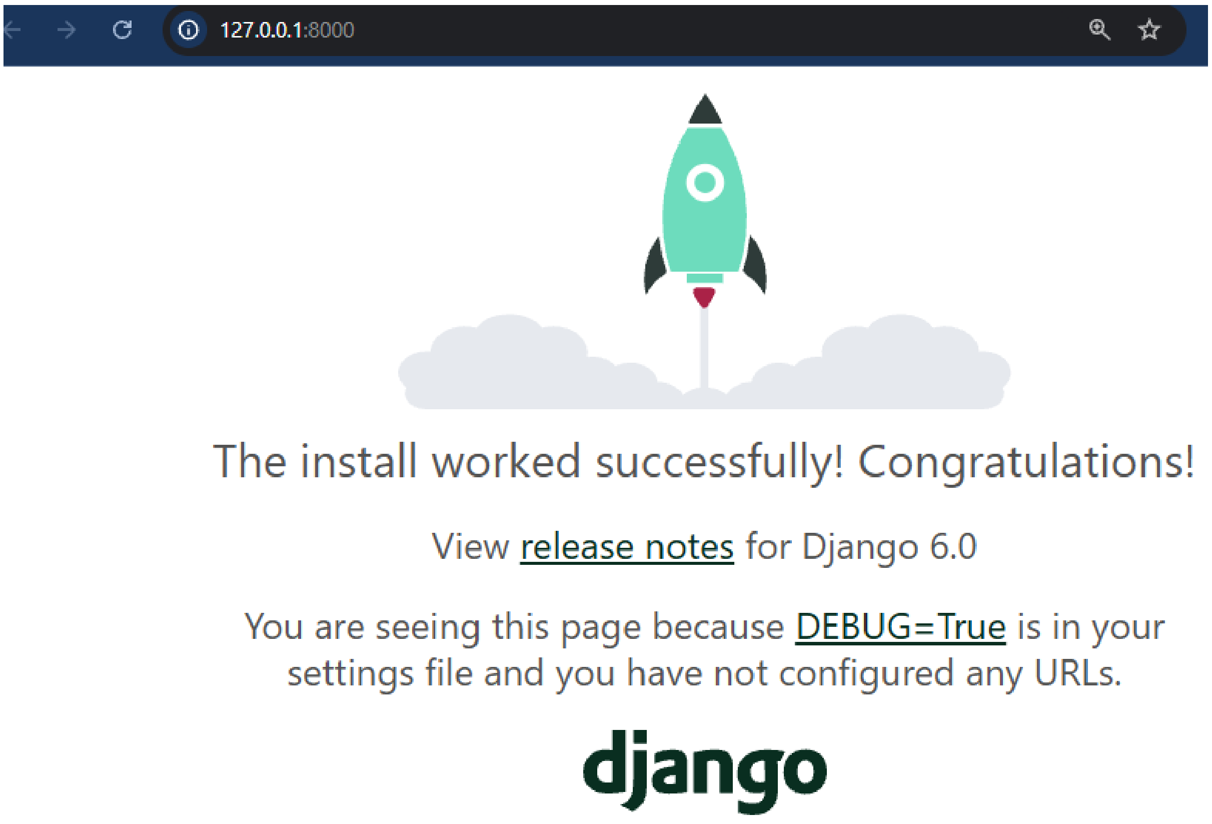
Step 9. Run migrations

```
python manage.py migrate
```

Step 10. Run the server

Python manage.py runserver

http://127.0.0.1:8000/



The install worked successfully! Congratulations!

View release notes for Django 6.0

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Step 11. Register the todo application

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'todo', #add this line
]
```

Step 12. Define the Todo model

Open the todo/models.py file in your code editor and add the following lines of code:

```
from django.db import models

# Create your models here.

class Todo(models.Model):
    title = models.CharField(max_length=120)
    description = models.TextField()
    completed = models.BooleanField(default=False)

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

Every time you update or add a new model, you need to run the following commands

First create the migrations file python manage.py makemigrations todo

Apply the migrations to the database python manage.py migrate todo

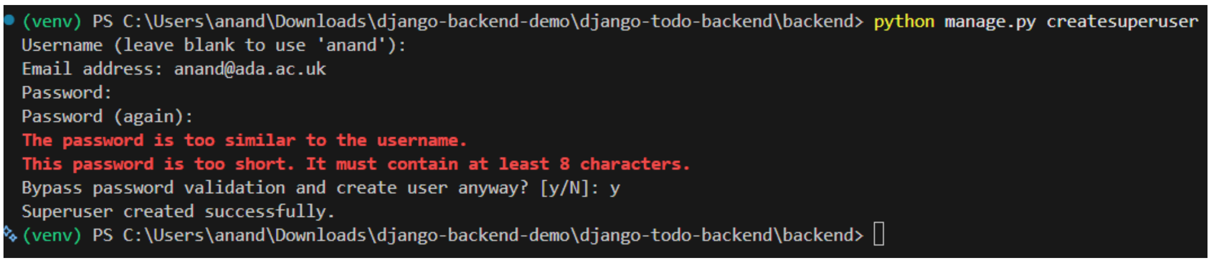
Step 13. Add the Todo model to admin

Open the todo/admin.py file with your code editor and add the following lines of code:

```
from django.contrib import admin
from .models import Todo

class TodoAdmin(admin.ModelAdmin):
    list_display = ('title', 'description', 'completed')

# Register your models here.

admin.site.register(Todo, TodoAdmin)
```

Step 14. Create a superuser account

Specify username, email (optional), and password to create a superuser account. Either make the password easy to remember (this is a local server right now) or remember the password. If you forget it, then just create another superuser.



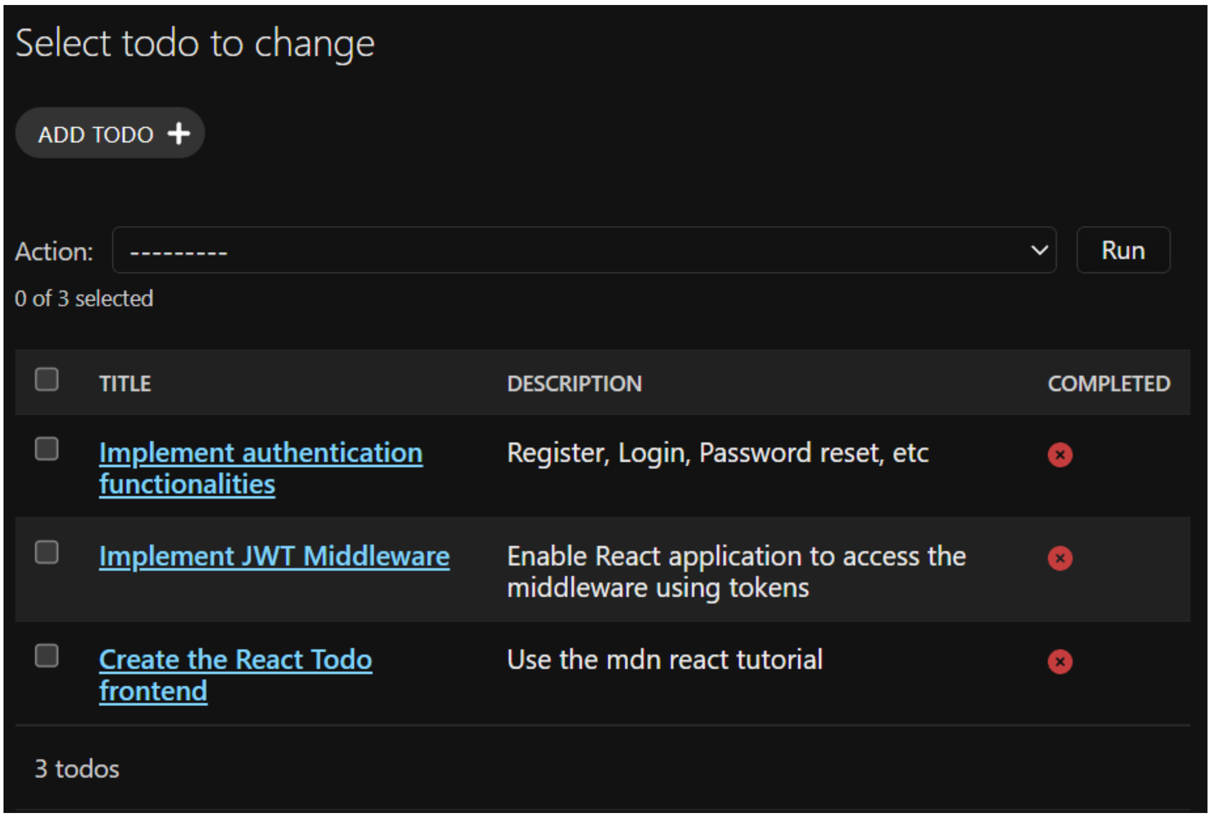
```
(venv) PS C:\Users\anand\Downloads\django-backend-demo\django-todo-backend\backend> python manage.py createsuperuser
Username (leave blank to use 'anand'):
Email address: anand@ada.ac.uk
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
(venv) PS C:\Users\anand\Downloads\django-backend-demo\django-todo-backend\backend>
```

Step 15. Test the Django Admin site

Make sure you run the Django server again `python manage.py runserver`

Access the admin site at http://127.0.0.1:8000/admin

Login with the username and password you used for the superuser. You should be able to add or modify new users/todos



Select todo to change

0 of 3 selected

	TITLE	DESCRIPTION	COMPLETED
☐	Implement authentication functionalities	Register, Login, Password reset, etc	✗
☐	Implement JWT Middleware	Enable React application to access the middleware using tokens	✗
☐	Create the React Todo frontend	Use the mdn react tutorial	✗

3 todos

Step 16. Setting up the APIs

In this step we will install and configure the Django REST framework and Django-cors-headers. Have a look at these links for further information. If you want a quick

understanding of what CORS is then have a look at
https://www.youtube.com/watch?v=4KHiSt0oLJ0

```
pip install djangorestframework django-cors-headers
```

Add the corheaders and rest_framework apps to the list of Installed apps

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'corsheaders', # add this app
    'rest_framework', # add this app
    'todo',
]
```

Add the corsheaders middleware to the list of middlewares

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

Note, **CorsMiddleware should be placed as high as possible**, especially before CommonMiddleware. Middleware executes top to bottom on the request and bottom to top on the response, so CorsMiddleware must run before CommonMiddleware, CsrfViewMiddleware, and any middleware that may return an early response. If it is placed too low in the stack, CORS headers may not be added to error responses such as 404 or 403. This can cause preflight OPTIONS requests to fail and result in browser errors indicating that the Access-Control-Allow-Origin header is missing.

Add the following towards the end of the backend/settings.py file

```
CORS_ORIGIN_WHITELIST = [
     'http://localhost:3000'
]
```

Note, you should replace 'http://localhost:3000' with your actual frontend url whether the frontend is on render or Codespace or local.

If you are working in Codespace, I found it useful to add the following in addition to the above:

```
CORS_ALLOWED_ORIGIN_REGEXES = [
    r"^https:\/\/.*-3000\.app\.github\.dev$",
]
```

This is because

- Codespaces generates dynamic subdomains.
- Hardcoding a single origin will break every time the Codespace URL changes.
- Regex allows all `*-3000.app.github.dev` frontend ports safely.

Step 17. Creating serializers

You will need serializers to convert model instances to JSON so that the frontend can work with the received data.

Create a `todo/serializers.py` file with your code editor. Open the `serializers.py` file and update it with the following lines of code:

```
from rest_framework import serializers
from .models import Todo

class TodoSerializer(serializers.ModelSerializer):
    class Meta:
        model = Todo
        fields = ('id', 'title', 'description', 'completed')
```

This code specifies the model to work with and the fields to be converted to JSON.

Step 18. Creating the View

You will need to create a TodoView class in the todo/views.py file.

Open the todo/views.py file with your code editor and add the following lines of code:

```
from django.shortcuts import render
from rest_framework import viewsets
from .serializers import TodoSerializer
from .models import Todo

# Create your views here.

class TodoView(viewsets.ModelViewSet):
    serializer_class = TodoSerializer
    queryset = Todo.objects.all()
```

Step 19. Set the API routes

Open the backend/urls.py file with your code editor and replace the contents with the following lines of code:

```
from django.contrib import admin
from django.urls import path, include
from rest_framework import routers
from todo import views

router = routers.DefaultRouter()
router.register(r'todos', views.TodoView, 'todo')

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

This code specifies the URL path for the API. This was the final step that completes the building of the API.

You can now perform CRUD operations on the Todo model. The router class allows you to make the following queries:

/todos/ - returns a list of all the Todo items. CREATE and READ operations can be performed here.

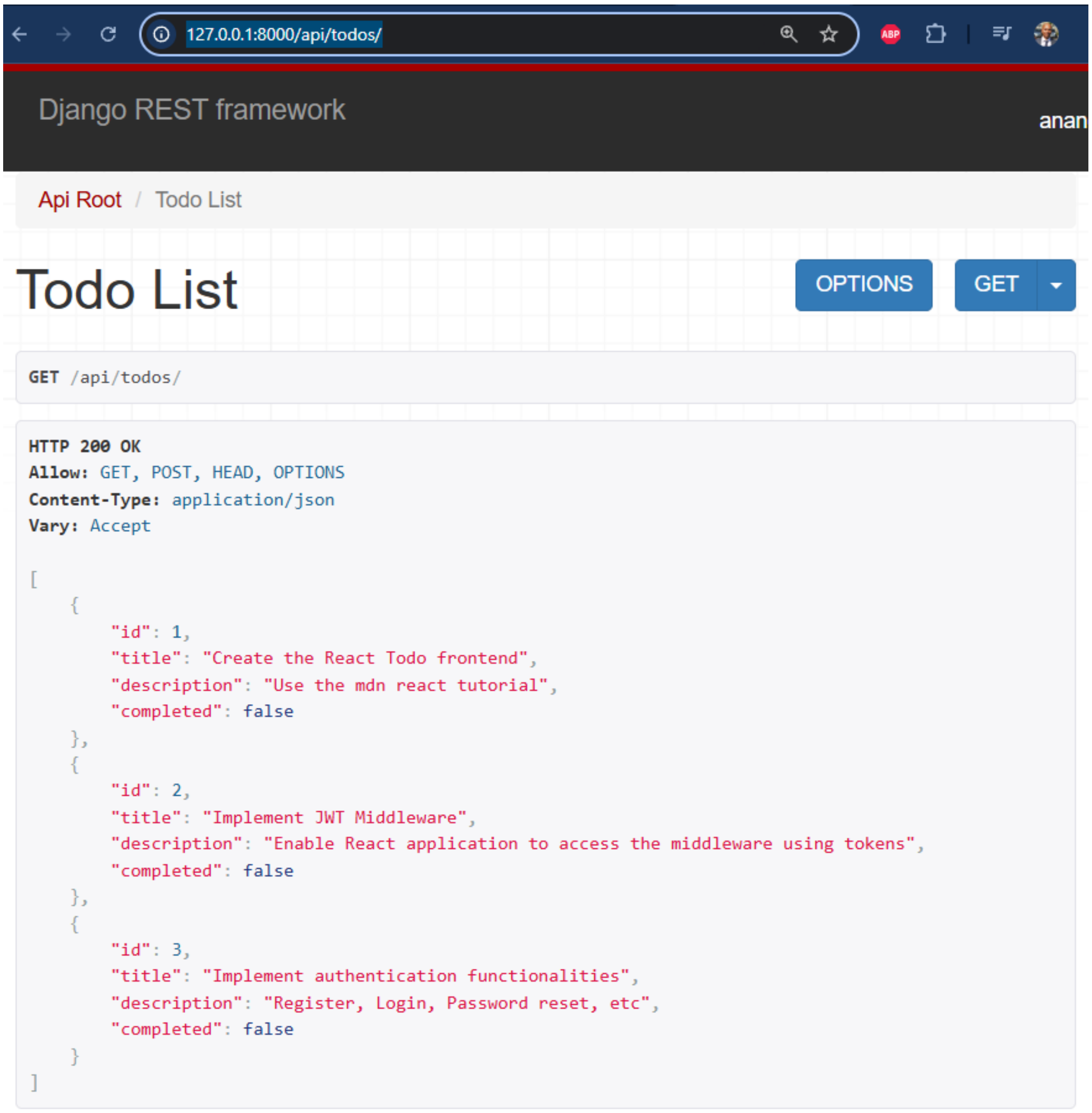
/todos/id - returns a single Todo item using the id primary key. UPDATE and DELETE operations can be performed here.

Let’s restart the server: `python manage.py runserver`

Step 20. Test the REST endpoints

Point your browser to http://127.0.0.1:8000/api/todos/

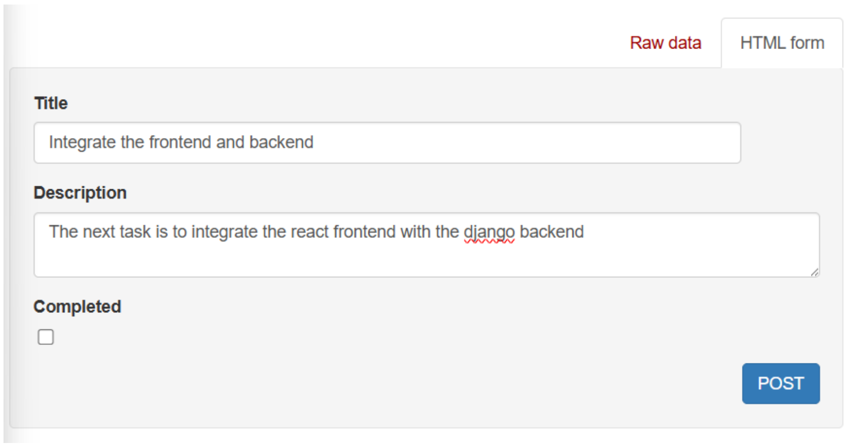
You should be able to see all the todos we created through the admin interface



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

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

[
    {
        "id": 1,
        "title": "Create the React Todo frontend",
        "description": "Use the mdn react tutorial",
        "completed": false
    },
    {
        "id": 2,
        "title": "Implement JWT Middleware",
        "description": "Enable React application to access the middleware using tokens",
        "completed": false
    },
    {
        "id": 3,
        "title": "Implement authentication functionalities",
        "description": "Register, Login, Password reset, etc",
        "completed": false
    }
]
```

You can create a new todo and check if it was added correctly



Title

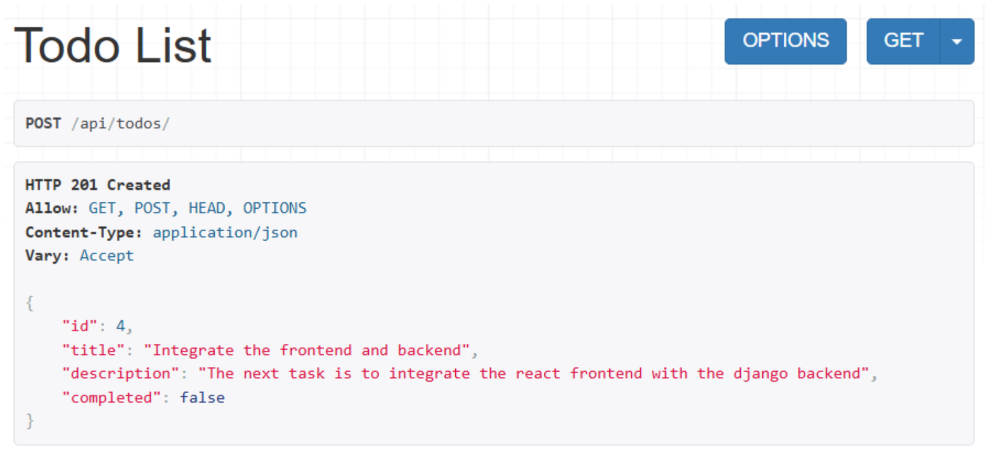
Integrate the frontend and backend

Description

The next task is to integrate the react frontend with the django backend

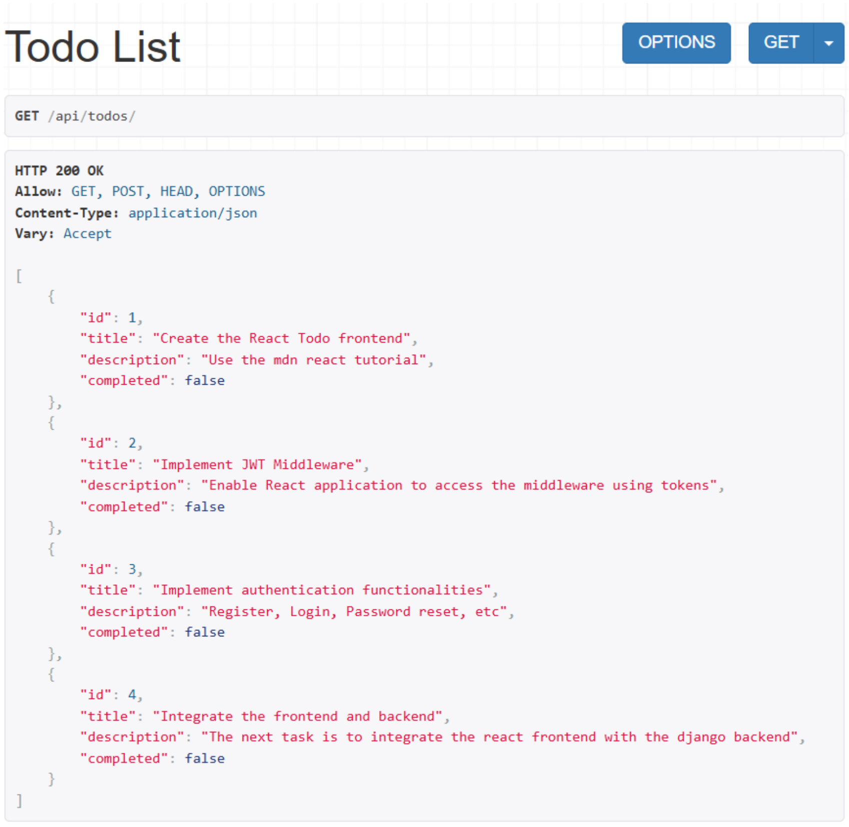
Completed

☐



```
POST /api/todos/
```

```
HTTP 201 Created
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 4,
    "title": "Integrate the frontend and backend",
    "description": "The next task is to integrate the react frontend with the django backend",
    "completed": false
}
```

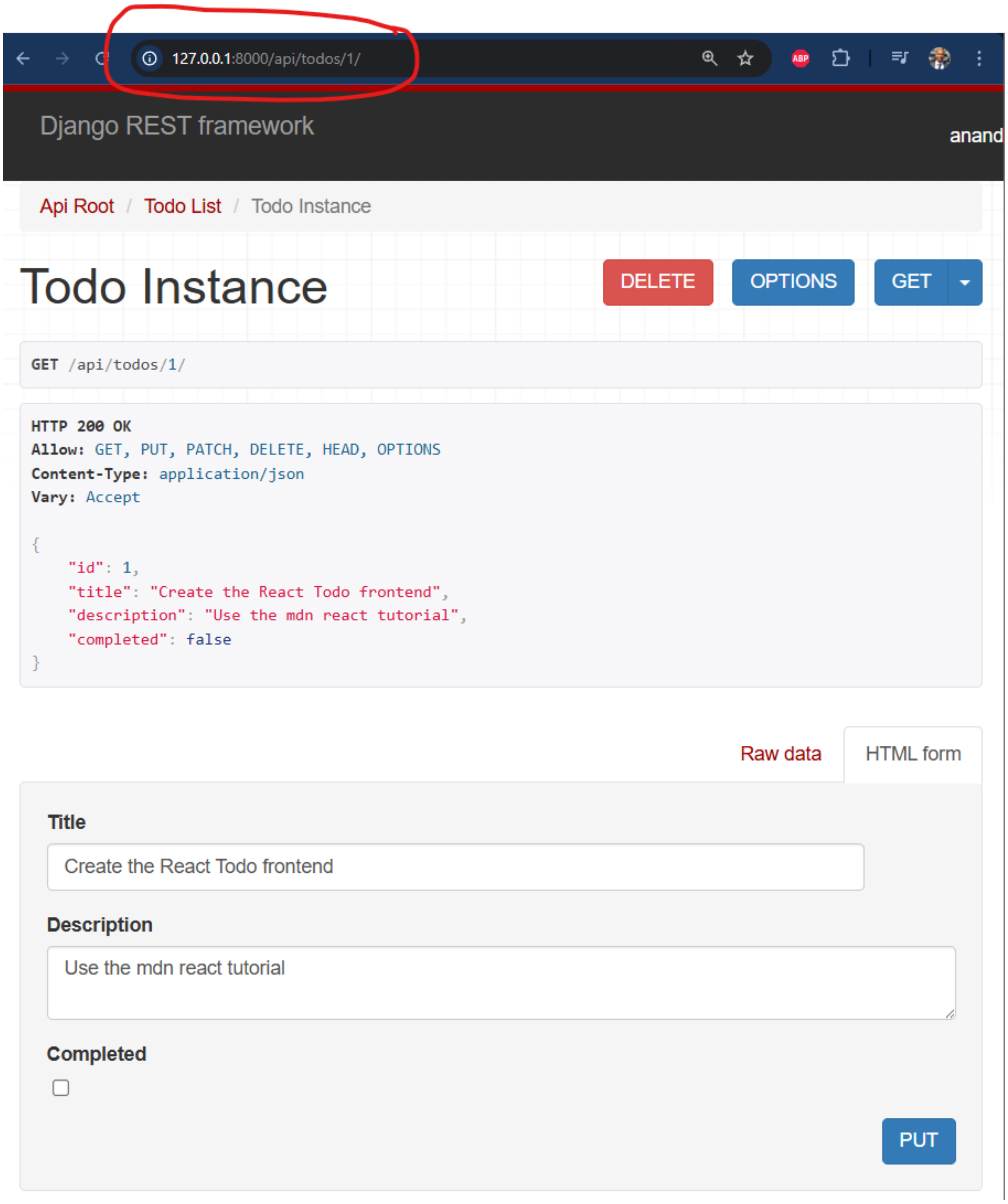


```
GET /api/todos/
```

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

[
    {
        "id": 1,
        "title": "Create the React Todo frontend",
        "description": "Use the mdn react tutorial",
        "completed": false
    },
    {
        "id": 2,
        "title": "Implement JWT Middleware",
        "description": "Enable React application to access the middleware using tokens",
        "completed": false
    },
    {
        "id": 3,
        "title": "Implement authentication functionalities",
        "description": "Register, Login, Password reset, etc",
        "completed": false
    },
    {
        "id": 4,
        "title": "Integrate the frontend and backend",
        "description": "The next task is to integrate the react frontend with the django backend",
        "completed": false
    }
]
```

You can also perform DELETE and UPDATE operations on specific Todo items using the id primary keys. Use the address structure /api/todos/{id} and provide an id.

Add 1 to the URL to examine the Todo item with the id of “1”. Navigate to http://localhost:8000/api/todos/1 in your web browser:



```
GET /api/todos/1/
```

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

{
    "id": 1,
    "title": "Create the React Todo frontend",
    "description": "Use the mdn react tutorial",
    "completed": false
}
```

Title: Create the React Todo frontend

Description: Use the mdn react tutorial

Completed: ☐