**PART A**

**Experiment No. 10**

**A.1 Aim:** Employee Management System using Django REST Framework (DRF)and React

**Objective:** To learn and understand how use CRUD operations in DRF using React

**A.2 Prerequisite:** HTML, CSS, CURD in Django, Postman or Thunder Client and React

**A.3 Outcome:**

After successful completion of these applications students will be able to understand Django DRF Framework and React connectivity

**A.4 Theory**

**CRUD Operations in Django using DRF**

CRUD stands for **Create**, **Read**, **Update**, and **Delete**, which are the four basic operations used to manage data in a database cane be handled by the function in views.py using DRF

**Steps for Setting up Django REST Framework (DRF)**

1. **Install DRF**: pip install django**restframework**
2. **Configure Django Settings**:
   1. Add 'rest\_framework' to your INSTALLED\_APPS list in settings.py:

INSTALLED\_APPS = [

...,

'rest\_framework',

]

1. **Create Serializer for Already created Employee Model:** In myapp/**serializers**.py:

**from rest\_framework import serializers**

from .models import Employee

class EmployeeSerializer(serializers.ModelSerializer):

    class Meta:

        model = Employee

        fields = ['id', 'name', 'salary']

1. **Update Views with API Endpoints:** In myapp/views.py:

from rest\_framework.decorators import api\_view

from rest\_framework.response import Response

from rest\_framework import status

from .models import **Employee**

from .serializers import **EmployeeSerializer**

# List Employees or Create new Employee

@api\_view(['GET', 'POST'])

def employee\_list(request):

if request.method == 'GET':

employees = Employee.objects.all()

serializer = EmployeeSerializer(employees, many=True)

return Response(serializer.data)…..

….

**Scenario Problem statement 1: Employee Management System Using Django DRF**

"Design and implement a Django-based REST API for managing employee records. This API should allow clients to perform CRUD operations on employee data.

Hints:

myproject/ # Root directory of your Django project

│

├── myproject/ # Django settings and configuration files

│ ├── \_\_init\_\_.py

│ ├── settings.py # Project settings

│ ├── urls.py # Root URL configuration

│ ├── asgi.py

│ └── wsgi.py

│

├── myapp3/ # Application directory (the app for employee management)

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── apps.py # Application configuration

│ ├── forms.py # Django forms for Employee model

│ ├── migrations/ # Database migrations for the app

│ │ └── \_\_init\_\_.py

│ ├── models.py # Employee model definition

│ ├── views.py # Application views (CRUD and employee-related operations)

│ ├── urls.py # URL routing for the app (employee CRUD operations)

│ └── templates/

│ ├── employee\_form.html # Template for employee creation and update

│ ├── employee\_list.html # Template to display employee list

│ └── employee\_confirm\_delete.html # Template for employee deletion confirmation

│

├── manage.py

└── db.sqlite3

**Sampe 2. URL Configuration (myapp/urls.py)**

from django.urls import path

from . import views

urlpatterns = [

    path('api/employees/', views.employee\_list, name='employee\_list\_api'),  # List or Create

    path('api/employees/<int:id>/', views.employee\_detail, name='employee\_detail\_api'),  # **Get, Update or Delete**

]

**Problem statement 2: Testing with React so React Frontend Setup need 2 steps**

1. Install axios ( in the cmd of react)
2. Install **cors (3 changes in settings.py of django)**

npx create-react-app **my-app-frontend**

cd my-app-frontend

npm start

<http://localhost:3000>

**Install Axios in cmd of react and check the package.info**

npm install axios

**Handling CORS**

Since React and Django are running on different ports, you need to enable Cross-Origin Resource Sharing (CORS).

**Install django-cors-headers in the cmd of django**

pip install django-cors-headers (Django remains the same )

**Update settings.py**

Add corsheaders to INSTALLED\_APPS and middleware:

INSTALLED\_APPS = [

...,

'corsheaders',

]

MIDDLEWARE = [

'corsheaders.middleware.**CorsMiddleware'**,

...,

]

CORS\_ALLOW\_ALL\_ORIGINS = True

**React Code for Employee CRUD**

Create a basic React app (in src/App.js) to interact with the Django backend:

import './App.css';

import React, { useState, useEffect } from 'react';

import axios from 'axios';

**const API\_URL = 'http://127.0.0.1:8000/myapp4/api/employees/';**

function App() {

    // return (

  //   <div className="App">

  //   Hello Duniya !!

  //   </div>

  // );

  const [employees, setEmployees] = useState([]);

  const [name, setName] = useState('');

  const [salary, setSalary] = useState('');

  const [editingId, setEditingId] = useState(null);

  // Fetch employees when the component loads

  useEffect(() => {

    axios.get(API\_URL).then((response) => {

      setEmployees(response.data);

    });

  }, []);

Directory Structure of the Complete Django App

myproject/

│

├── manage.py

├── myproject/ # Project settings directory

│ ├── \_\_init\_\_.py

│ ├── settings.py # Django settings

│ ├── urls.py # Project-level URL routing

│ └── wsgi.py

│

├── myapp/ # Your Django app (e.g., Employee management)

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── apps.py

│ ├── models.py # Define Employee model here

**│ ├── serializers.py # Define Employee serializer here**

│ ├── urls.py # App-level URL routing

│ └── views.py # Define API views for CRUD operations

└── db.sqlite3 # SQLite database file

**2. React Frontend (my-app-frontend) Directory Structure:**

my-app-frontend/

│

├── node\_modules/ # Auto-generated when you run npm install

├── public/

│ ├── index.html # Main HTML file

│ └── ...

├── src/

│ ├── App.js # Main App component with your CRUD logic

│ ├── index.js # Renders App to the DOM

**│ └── App.css # Styling for App component**

├── package.json # Project metadata and dependencies (axios installed here)

└── README.md # Basic project documentation

**Updated settings.py:**

INSTALLED\_APPS = [

...,

**'rest\_framework',**

**'corsheaders',**

]

MIDDLEWARE = [

'corsheaders.middleware.CorsMiddleware', # Already present

...,

]

Note: Total 3 installation

1. on django command prompt> **pip install djangorestframework django-cors-headers**
2. on react command prompt> **npm install axios**

**ART B**

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

|  |  |
| --- | --- |
| Roll No. : S020 | Name: Husain Chhil |
| Class : MBA Tech Data Science | Batch : J1 |
| Date of Experiment : | Date/Time of Submission : |
| Grade : |  |

**B.1 Code:**

**Django**

**/myproject/settings.py:**

INSTALLED\_APPS = [

…

'rest\_framework',

    'corsheaders',

    'myapp',

]

**/myapp/urls.py:**

from django.urls import path

from . import views

urlpatterns = [

    path('api/employees/', views.employee\_list, name='employee\_list\_api'),

    path('api/employees/<int:id>/', views.employee\_detail, name='employee\_detail\_api'),

]

**/myapp/models.py:**

from django.db import models

class **Employee**(**models**.**Model**):

**id** = models.AutoField(primary\_key=True)

    name = models.CharField(max\_length=100)

    salary = models.DecimalField(max\_digits=10, decimal\_places=2)

    def **\_\_str\_\_**(self):

        return self.name

**/myapp/views.py:**

from rest\_framework.decorators import api\_view

from rest\_framework.response import Response

from rest\_framework import status

from .models import Employee

from .serializers import EmployeeSerializer

**@api\_view**(['GET', 'POST'])

def **employee\_list**(request):

    if request.method == 'GET':

        employees = Employee.objects.all()

        serializer = EmployeeSerializer(employees, many=True)

        return Response(serializer.data)

    elif request.method == 'POST':

        serializer = EmployeeSerializer(data=request.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)

**@api\_view**(['GET', 'PUT', 'DELETE'])

def **employee\_detail**(request, id):

    try:

        employee = Employee.objects.get(pk=**id**)

    except Employee.DoesNotExist:

        return Response(status=status.HTTP\_404\_NOT\_FOUND)

    if request.method == 'GET':

        serializer = EmployeeSerializer(employee)

        return Response(serializer.data)

    elif request.method == 'PUT':

        serializer = EmployeeSerializer(employee, data=request.data)

        if serializer.is\_valid():

            serializer.save()

            return Response(serializer.data)

        return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

    elif request.method == 'DELETE':

        employee.delete()

        return Response(status=status.HTTP\_204\_NO\_CONTENT)

**/myapp/serializers.py:**

from rest\_framework import serializers

from .models import Employee

class **EmployeeSerializer**(**serializers**.**ModelSerializer**):

    class **Meta**:

        model = Employee

        fields = ('id', 'name', 'salary')

/myproject/urls.py

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('myapp.urls'))

]

**React:**

**App.js:**

*// src/App.js*

import './App.css';

import React, { useState, useEffect } from 'react';

import {

  fetchEmployees,

  createEmployee,

  updateEmployee,

  deleteEmployee,

} from './services/GlobalAPI.js';

function **App**() {

  const [employees, **setEmployees**] = **useState**([]);

  const [name, **setName**] = **useState**('');

  const [salary, **setSalary**] = **useState**('');

  const [editingId, **setEditingId**] = **useState**(null);

**useEffect**(() => {

    const **getEmployees** = async () => {

      const data = await **fetchEmployees**();

**setEmployees**(data);

    };

**getEmployees**();

  }, []);

  const **handleSubmit** = async (e) => {

    e.**preventDefault**();

    const employeeData = { name, salary };

    if (editingId) {

      const updatedEmployee = await **updateEmployee**(editingId, employeeData);

**setEmployees**((prev) =>

        prev.**map**((emp) => (emp.id === editingId ? { ...emp, ...updatedEmployee } : emp))

      );

**setEditingId**(null);

    } else {

      const newEmployee = await **createEmployee**(employeeData);

**setEmployees**((prev) => [...prev, newEmployee]);

    }

**setName**('');

**setSalary**('');

  };

  const **handleEdit** = (employee) => {

**setName**(employee.name);

**setSalary**(employee.salary);

**setEditingId**(employee.id);

  };

  const **handleDelete** = async (id) => {

    await **deleteEmployee**(id);

**setEmployees**((prev) => prev.**filter**((emp) => emp.id !== id));

  };

  return (

    <div *className*="container mx-auto p-4">

      <h1 *className*="text-2xl font-bold text-center mb-4">Employee Management</h1>

      <form *onSubmit*={**handleSubmit**} *className*="mb-4">

        <div *className*="flex justify-between">

          <input

*type*="text"

*placeholder*="Name"

*value*={name}

*onChange*={(e) => **setName**(e.target.value)}

*className*="border border-gray-300 p-2 w-1/2 mr-2 rounded"

*required*

          />

          <input

*type*="number"

*placeholder*="Salary"

*value*={salary}

*onChange*={(e) => **setSalary**(e.target.value)}

*className*="border border-gray-300 p-2 w-1/4 mr-2 rounded"

*required*

          />

          <button

*type*="submit"

*className*="bg-blue-500 text-white p-2 rounded hover:bg-blue-600 transition duration-300"

          >

            {editingId ? 'Update' : 'Add'}

          </button>

        </div>

      </form>

      <div *className*="overflow-x-auto">

        <table *className*="min-w-full bg-white shadow-md rounded-lg overflow-hidden">

          <thead>

            <tr *className*="bg-gray-800 text-white">

              <th *className*="py-2 px-4">ID</th>

              <th *className*="py-2 px-4">Name</th>

              <th *className*="py-2 px-4">Salary</th>

              <th *className*="py-2 px-4">Actions</th>

            </tr>

          </thead>

          <tbody>

            {employees.**map**((employee) => (

              <tr *key*={employee.id} *className*="hover:bg-gray-100">

                <td *className*="py-2 px-4 border-b">{employee.id}</td>

                <td *className*="py-2 px-4 border-b">{employee.name}</td>

                <td *className*="py-2 px-4 border-b">{employee.salary}</td>

                <td *className*="py-2 px-4 border-b">

                  <button

*onClick*={() => **handleEdit**(employee)}

*className*="bg-yellow-500 text-white p-1 rounded hover:bg-yellow-600 mr-2"

                  >

                    Edit

                  </button>

                  <button

*onClick*={() => **handleDelete**(employee.id)}

*className*="bg-red-500 text-white p-1 rounded hover:bg-red-600"

                  >

                    Delete

                  </button>

                </td>

              </tr>

            ))}

          </tbody>

        </table>

      </div>

    </div>

  );

}

export default **App**;

**/src/services/GlobalAPI.js:**

*// src/GlobalAPI.js*

import axios from 'axios';

const API\_URL = 'http://127.0.0.1:8000/api/employees/';

export const **fetchEmployees** = async () => {

  const response = await axios.**get**(API\_URL);

  return response.data;

};

export const **createEmployee** = async (employeeData) => {

  const response = await axios.**post**(API\_URL, employeeData);

  return response.data;

};

export const **updateEmployee** = async (id, employeeData) => {

  const response = await axios.**put**(`${API\_URL}${id}/`, employeeData);

  return response.data;

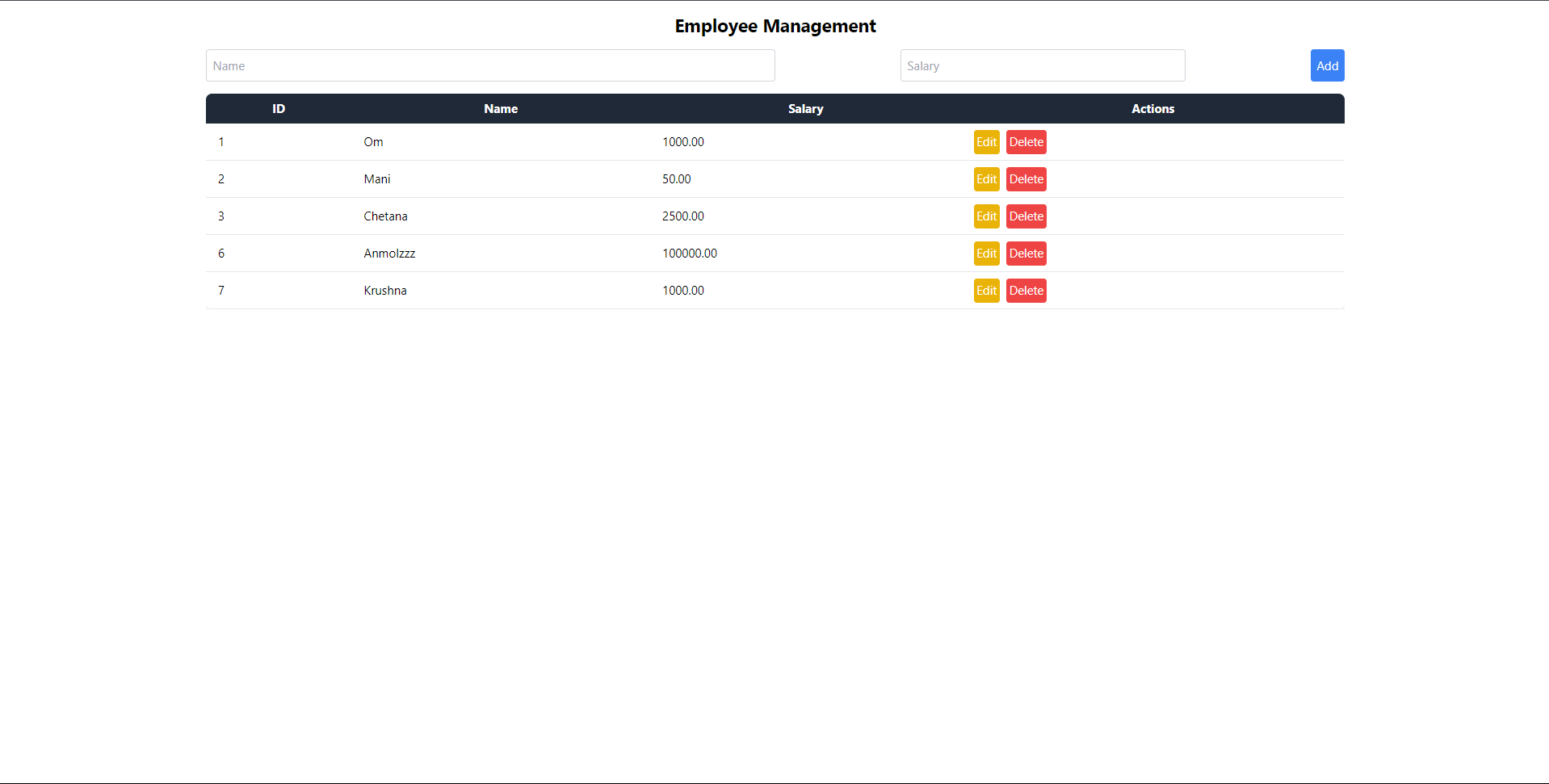
};

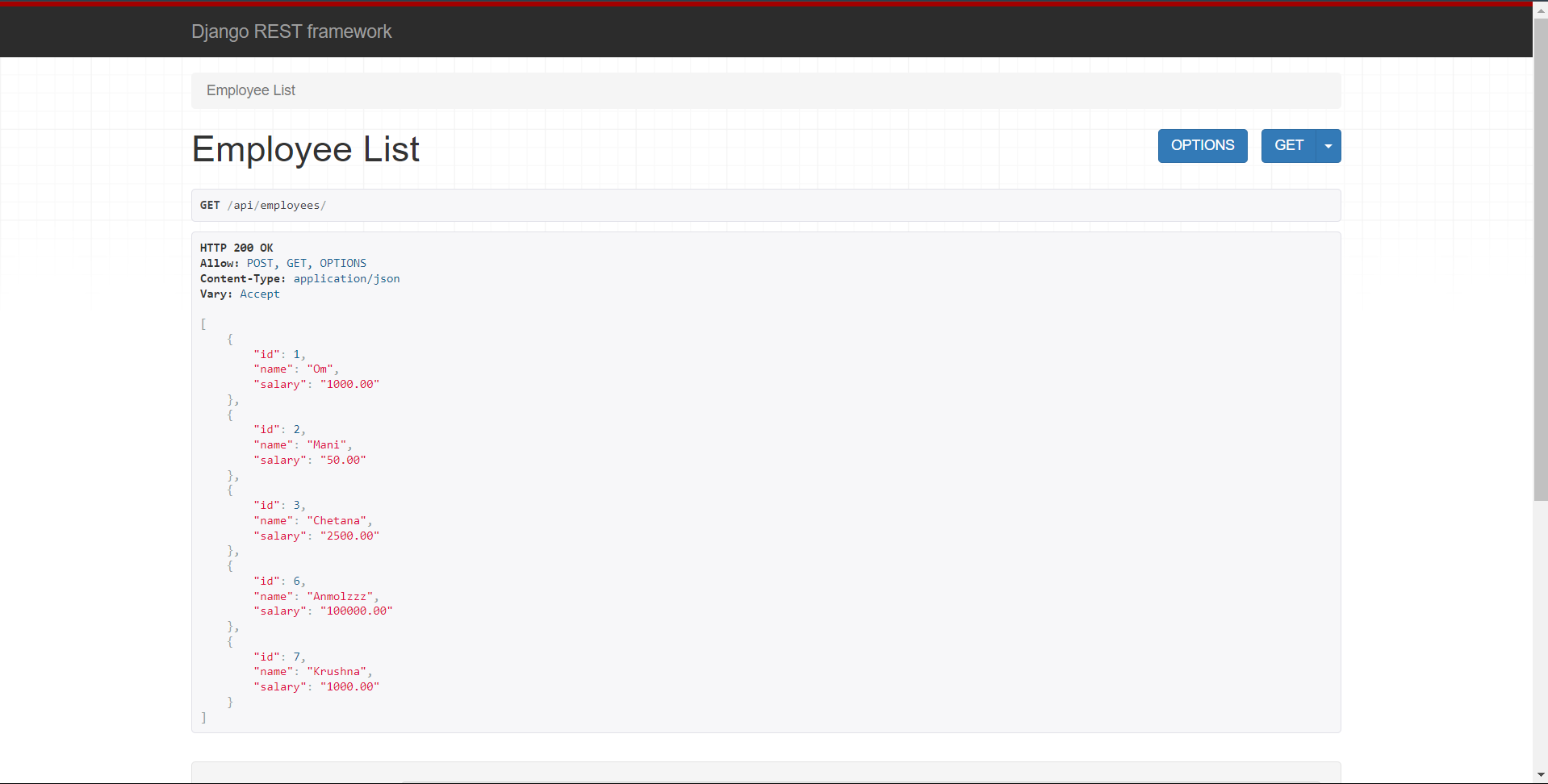
export const **deleteEmployee** = async (id) => {

  await axios.**delete**(`${API\_URL}${id}/`);

};

**B.2 Output**

****

****

**B.3 Conclusion:**

We performed CRUD operations using the rest-framework in Django and and accessed the API using React library axios.

**B.3 Observations and Learning:**

Learned how to work with APIs in Django and React, and how to integrate these 2 platforms

**B.4 Question of Curiosity**

1. How do you set up communication between a React frontend and a Django REST Framework backend?

Ans: To set up communication between a React frontend and a Django REST Framework (DRF) backend:

* **Create a Django Project**: Start by setting up your Django project and app. Install Django and DRF, and define your models, serializers, and views.
* **Set Up API Endpoints**: Define your API endpoints in urls.py using DRF's routers or manually.
* **Start the Django Server**: Run the Django development server, typically on http://127.0.0.1:8000/.
* **Create a React App**: Use create-react-app to set up your React project.
* **Make API Requests**: Use libraries like Axios or Fetch in your React components to make HTTP requests to the Django API.

1. What role does axios or fetch play in making API requests from React to Django REST Framework?

Ans: Axios and Fetch are JavaScript libraries used to make HTTP requests from the React frontend to the Django REST Framework backend. They help in:

* **Making GET, POST, PUT, DELETE Requests**: You can easily send various types of HTTP requests to interact with the API.
* **Handling Responses**: Both libraries return promises, allowing you to handle asynchronous responses and update the React component state accordingly.
* **Configuring Headers**: They allow you to set headers, such as authorization tokens or content types, necessary for communicating with the backend.

1. How do you handle Cross-Origin Resource Sharing (CORS) issues when connecting React with Django REST Framework?

Ans: CORS issues occur when your React app and Django server are on different origins. To handle CORS in a Django application:

* **Install django-cors-headers**: This package allows you to manage CORS headers in your Django app.
* **Update Django Settings**: Add 'corsheaders' to your INSTALLED\_APPS and add middleware in your settings.py.

1. How can you display a list of records fetched from a Django REST Framework API in a React component?

Ans: To display a list of records fetched from a DRF API in a React component:

* **Use State and Effect Hooks**: Utilize useState to store data and useEffect to fetch data from the API on component mount.
* **Make the API Call**: Use Axios or Fetch to retrieve data from your API.
* **Render the Data**: Map through the fetched data and display it in your component.

import React, { useEffect, useState } from 'react';

import axios from 'axios';

const EmployeeList = () => {

const [employees, setEmployees] = useState([]);

useEffect(() => {

const fetchEmployees = async () => {

const response = await axios.get('http://127.0.0.1:8000/api/employees/');

setEmployees(response.data);

};

fetchEmployees();

}, []);

return (

<ul>

{employees.map(employee => (

<li key={employee.id}>{employee.name} - ${employee.salary}</li>

))}

</ul>

);

};

export default EmployeeList;

1. What is the best way to handle authentication and token management when connecting React with Django REST Framework?

Ans:

1. **Token Authentication**: Use Django REST Framework's JWT for secure API access.
2. **Install Required Packages**: Ensure you have the necessary packages for token authentication in Django.
3. **Update Django Settings**: Configure the REST Framework to use JWT authentication.
4. **Create a Login Endpoint**: Set up an endpoint in Django to obtain tokens when users log in.
5. **Manage Tokens in React**: Store the JWT in local storage and include it in the headers of your API requests to authenticate user actions.

This setup ensures secure communication and proper token management between your React frontend and Django REST backend.