React js and django backend connectivity code  notes step by step :-

React js  Frontendapi  :-

Step 1:

Setting up the project First, let's create a new React app.

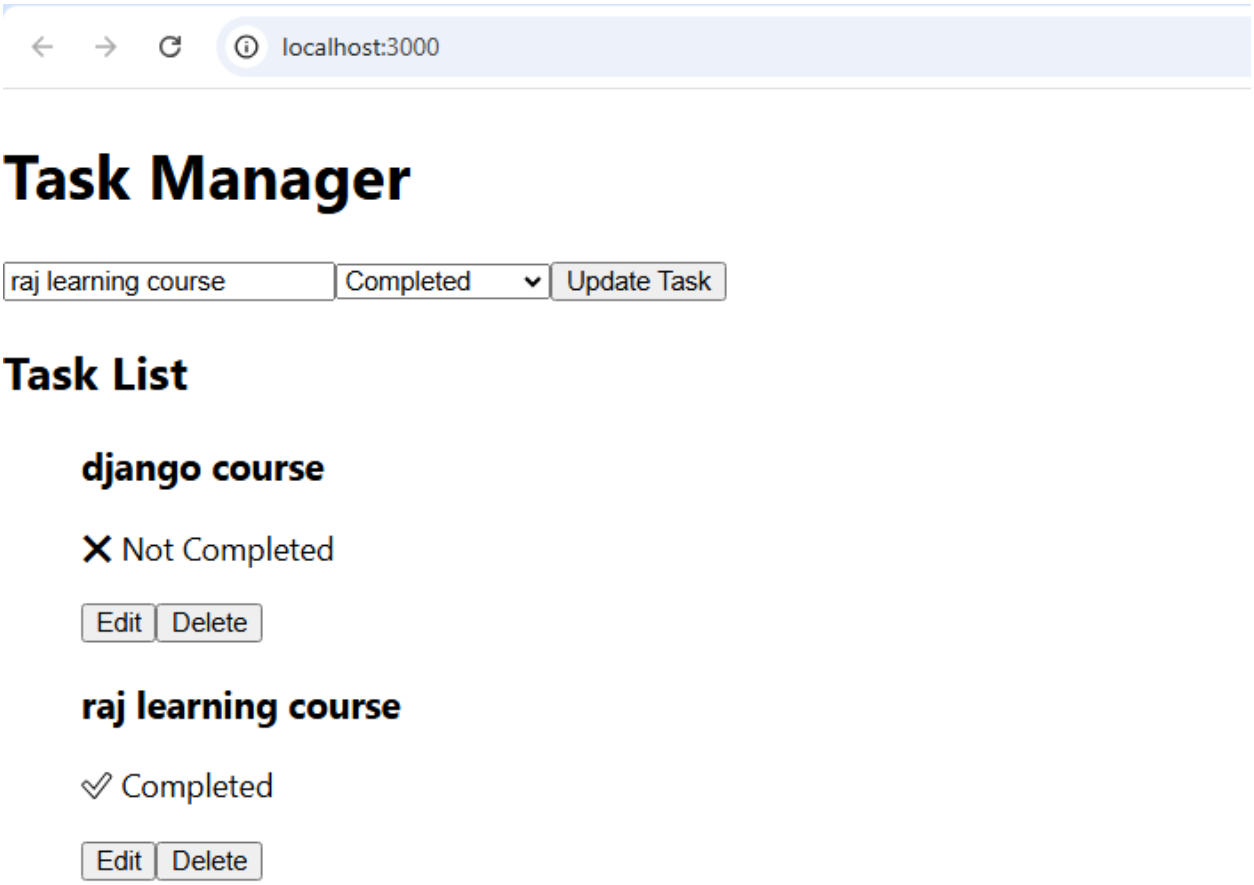
If you haven't already set up React, you can create a new project with the following commands:

Cmd(terminal)  command to type :-

npx create-react-app frontendapi

cd  frontendapi

npm start



Step 2:-

`TaskList.js` with Insert, Update, and Delete:-

```
import { useState, useEffect } from "react";
import axios from "axios";

const API_URL = "http://localhost:8000/api/tasks/";

const TaskList = () => {
  const [data, setData] = useState([]);
  const [task, setTask] = useState({ title: "", completed: false });
  const [editingId, setEditingId] = useState(null);

  useEffect(() => {
    getTasks();
  }, []);

  const getTasks = () => {
    axios.get(API_URL).then((response) => {
      setData(response.data);
    });
  };

  const handleChange = (e) => {
    const { name, value } = e.target;
    setTask((prev) => ({
      ...prev,
      [name]: name === "completed" ? value === "true" : value,
    }));
  };

  const handleSubmit = (e) => {
    e.preventDefault();

    if (editingId !== null) {
      // Update
      axios.put(`${API_URL}${editingId}/`, task).then(() => {
        setTask({ title: "", completed: false });
        setEditingId(null);
        getTasks();
```

```
    });
  } else {
    // Create
    axios.post(API_URL, task).then(() => {
      setTask({ title: "", completed: false });
      getTasks();
    });
  }
};

const handleEdit = (item) => {
  setTask({ title: item.title, completed: item.completed });
  setEditingId(item.id);
};

const handleDelete = (id) => {
  axios.delete(`${API_URL}${id}/`).then(() => {
    getTasks();
  });
};

return (
  <>
    <h1>Task Manager</h1>

    <form onSubmit={handleSubmit}>
      <input
        name="title"
        value={task.title}
        onChange={handleChange}
        placeholder="Title"
        required
      />
      <select
        name="completed"
        value={task.completed}
        onChange={handleChange}
      >
        <option value={false}>Not Completed</option>
        <option value={true}>Completed</option>
      </select>
      <button type="submit">{editingId ? "Update" : "Add"} Task</button>
    </form>

    <h2>Task List</h2>
    <ul>
      {data &&
        data.map((item) => (
```

```
            <div className="row" key={item.id}>
              <div className="col-sm-3">
                <h3>{item.title}</h3>
                <p>{item.completed ? "  Completed": "  Not Completed"}</p>
                <button onClick={() => handleEdit(item)}>Edit</button>
                <button onClick={() => handleDelete(item.id)}>Delete</button>
              </div>
            </div>
          ))}
      </ul>
    </>
  );
};

export default TaskList;
```

App.js code:-

```
import React from 'react';
import TaskList from './TaskList';

function App() {
  return (
    <div className="App">
      <TaskList />
    </div>
  );
}

export default App;
```

Sure! Here's a **detailed explanation** of your React code, section by section. You're building a **Task Manager** that supports:

- **Create** 
- **Read** 
- **Update** ✒
- **Delete** 

(aka **CRUD**) functionality using React + Axios + a Django REST API.

1. **Imports**

**import { useState, useEffect } from "react";**

**import axios from "axios";**

- `useState` – lets you manage state variables (`data`, `task`, `editingId`).
- `useEffect` – used to perform side effects like fetching data when the component mounts.
- `axios` – makes HTTP requests to your Django backend.

## 2. API URL

```
const API_URL = "http://localhost:8000/api/tasks/";
```

This is the base URL to your Django backend (make sure it's running). You're interacting with `/api/tasks/`.

3. **Component & State Setup**

```
const [data, setData] = useState([]);
const [task, setTask] = useState({ title: "", completed: false });
const [editingId, setEditingId] = useState(null);
```

- `data`: Holds the list of tasks fetched from the backend.
- `task`: Holds the current task being added or edited.
- `editingId`: If set, it means you're updating a task (not creating a new one).

4. **Fetch Data on Load**

```
useEffect(() => {
  getTasks();
}, []);
```

- Runs `getTasks()` once when the component mounts.
- `[]` makes it run only once, similar to `componentDidMount()` in class components.

## 5. Get Tasks from Backend

```
const getTasks = () => {
  axios.get(API_URL).then((response) => {
    setData(response.data);
  });
};
```

- Sends a `GET` request to the backend to fetch all tasks.
- The response data (list of tasks) is saved to `data`.

6. **Handle Input Changes**

```
const handleChange = (e) => {

 const { name, value } = e.target;

 setTask((prev) => ({

  ...prev,

  [name]: name === "completed" ? value === "true" : value,

 }));

};
```

- Updates the `task` state when typing in inputs or changing the dropdown.
- Special logic for `completed` because it's a boolean (`true`/`false` from string).

7. **Handle Submit: Create or Update**

```
const handleSubmit = (e) => {

 e.preventDefault();


 if (editingId !== null) {

  axios.put(`${API_URL}${editingId}/`, task).then(() => {

   setTask({ title: "", completed: false });

   setEditingId(null);

   getTasks();

  });
```

```
  } else {
    axios.post(API_URL, task).then(() => {
      setTask({ title: "", completed: false });
      getTasks();
    });
  }
};
```

- If `editingId` is set: sends a `PUT` request to update the task.
- If not: sends a `POST` request to create a new task.
- Then resets the form and reloads the list.

8. **Edit a Task**

```
const handleEdit = (item) => {
  setTask({ title: item.title, completed: item.completed });
  setEditingId(item.id);
};
```

- Prefills the form with the task's data when the "Edit" button is clicked.
- Sets the `editingId` to enable "update" mode.

9. **Delete a Task**

```
const handleDelete = (id) => {
  axios.delete(`${API_URL}${id}/`).then(() => {
    getTasks();
  });
```

};

- Sends a `DELETE` request to the backend.
- Reloads the task list after deletion.

10. **JSX UI Rendering**

```
<form onSubmit={handleSubmit}>
  <input ... />
  <select ...>
    <option value={false}>Not Completed</option>
    <option value={true}>Completed</option>
  </select>
  <button type="submit">{editingId ? "Update" : "Add"} Task</button>
</form>
```

- Renders a form to add or update tasks.
- The button text changes based on whether you're editing.

11. **Render Task List**

```
<ul>
  {data &&
    data.map((item) => (
      <div key={item.id}>
        <h3>{item.title}</h3>
        <p>{item.completed ? "□ Completed" : "□ Not Completed"}</p>
        <button onClick={() => handleEdit(item)}>Edit</button>
        <button onClick={() => handleDelete(item.id)}>Delete</button>
      </div>
```

))}

</ul>

- Loops through the `data` (tasks) and displays them.
- Each task has buttons for **Edit** and **Delete**.

## □ Summary

| Feature | Description |
| --- | --- |
| **Create** | Form + `axios.post()` |
| **Read** | `useEffect()` + `axios.get()` |
| **Update** | Edit form pre-fill + `axios.put()` |
| **Delete** | Delete button + `axios.delete()` |

Step 3:-

Django Rest Api backend code:-

1) pip install django-cors-headers

```
2) pip install djangorestframework
```

and in your project frolder setting.py as show below

'rest_framework',

'corsheaders',

```
# Application definition

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

```
    'rest_framework',
     'todo',
     'corsheaders',


]
```

todo/models.py :-

```
from django.db import models
class Task(models.Model):
 title = models.CharField(max_length=255)
 completed = models.BooleanField(default=False)
 def __str__(self):
  return self.title
```

todo/serializers.py :-

```
from rest_framework import serializers
from .models import Task
class TaskSerializer(serializers.ModelSerializer):
 class Meta:
   model = Task
   fields = '__all__'
```

todo/urls.py :-

```
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import TaskViewSet
router = DefaultRouter()
router.register(r'tasks', TaskViewSet)
urlpatterns = [
path('', include(router.urls)),
]
```

todo/views.py :-

```
from rest_framework import viewsets
```

```
from .models import Task
from .serializers import TaskSerializer
class TaskViewSet(viewsets.ModelViewSet):
 queryset = Task.objects.all()
 serializer_class = TaskSerializer
```

urls.py (project folder):-

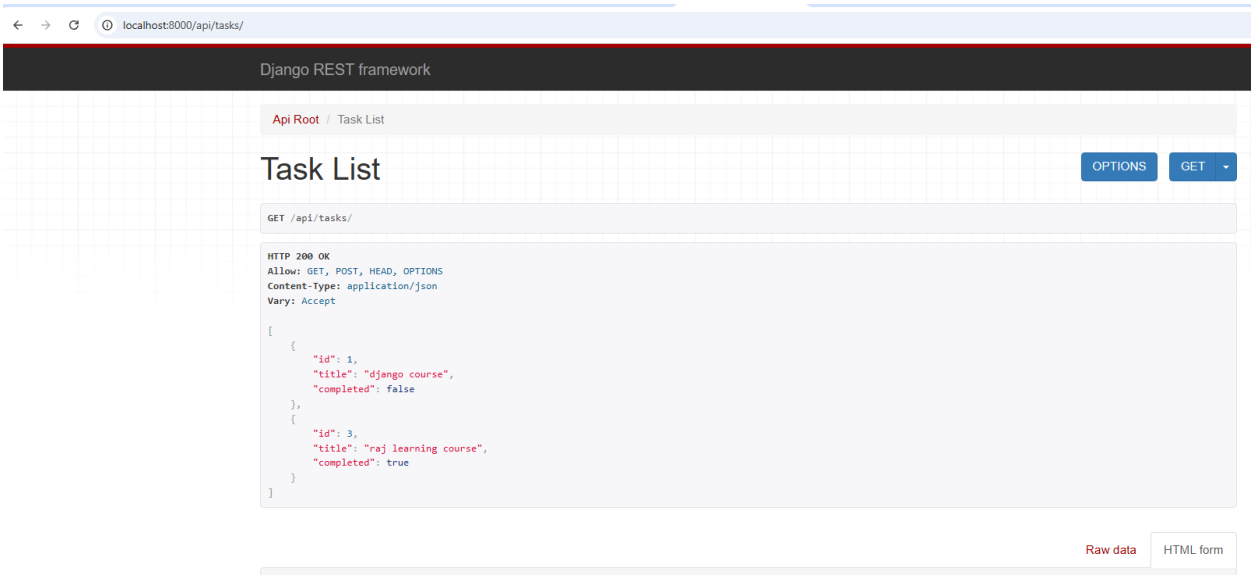
```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
path('admin/', admin.site.urls),
path('api/', include('todo.urls')),
]
```

Finally run backend :-

python manage.py runserver

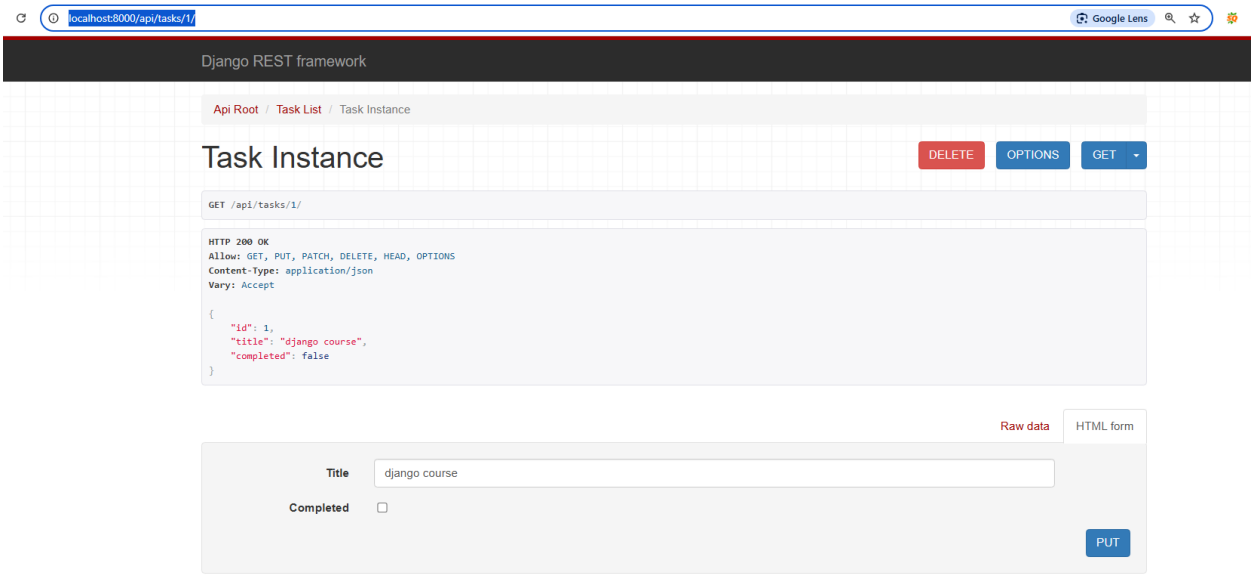
Get and display

http://localhost:8000/api/tasks/




put and delete

http://localhost:8000/api/tasks/1/


localhost:8000/api/tasks/1/
Google Lens
Django REST framework
Api Root / Task List / Task Instance
Task Instance
DELETE
OPTIONS
GET
GET /api/tasks/1/
HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
{
"id": 1,
"title": "django course",
"completed": false
}
Raw data
HTML form
Title
django course
Completed
PUT