

Here's a **step-by-step beginner-friendly guide** to create a **Django REST API** for a table with:

`id, name, city, phone, email`

We'll use:

- Django
- Django REST Framework

□ **STEP 1: Install Django + DRF**

Create virtual environment in your windows pc

```
python -m venv env
env\Scripts\activate
```

Install packages

```
pip install django djangorestframework
```

□ **STEP 2: Create Project**

```
django-admin startproject myapiproject
cd myapiproject
```

```
(env) D:\django-warmup>django-admin startproject myapiproject
(env) D:\django-warmup>cd myapiproject
(env) D:\django-warmup\myapiproject>_
```

□ STEP 3: Create App

```
python manage.py startapp api
```

```
(env) D:\django-warmup\myapiproject>python manage.py startapp api
```

□ STEP 4: Add Apps in settings.py

Open myproject/settings.py

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
  
    'rest_framework', # DRF  
    'api',             # your app  
]
```

□ STEP 5: Create Model (Database Table)

In api/models.py

```
from django.db import models  
  
class Student(models.Model):  
    name = models.CharField(max_length=100)  
    city = models.CharField(max_length=100)  
    phone = models.CharField(max_length=15)  
    email = models.EmailField()  
  
    def __str__(self):  
        return self.name
```

❑ STEP 6: Run Migrations

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

```
(env) D:\django-warmup\myapiproject>python manage.py makemigrations
Migrations for 'api':
  api\migrations\0001_initial.py
  + Create model Student

(env) D:\django-warmup\myapiproject>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, api, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
```

❑ STEP 7: Create Serializer (VERY IMPORTANT)

Create file:

```
❑ api/serializers.py
```

```
from rest_framework import serializers
from .models import Student

class StudentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Student
        fields = '__all__'
```

```
❑ This converts database ↔ JSON
```

STEP 8: **ViewSet (CORE CONCEPT)**

Instead of writing GET/POST/PUT/DELETE manually, we use:

□ `ModelViewSet`

□ `api/views.py`

```
from rest_framework import viewsets
from .models import Student
from .serializers import StudentSerializer

class StudentViewSet(viewsets.ModelViewSet):
    queryset = Student.objects.all()
    serializer_class = StudentSerializer
```

□ **WHAT THIS AUTOMATICALLY GIVES YOU**

Without writing extra code, you now get:

Action	HTTP Method	URL
List all students	GET	/students/
Create student	POST	/students/
Retrieve student	GET	/students/1/
Update student	PUT	/students/1/
Delete student	DELETE	/students/1/

□ STEP 9: ROUTER (AUTO URL GENERATION)

□ api/urls.py

```
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import StudentViewSet

router = DefaultRouter()
router.register('students', StudentViewSet, basename='student')

urlpatterns = [
    path('', include(router.urls)),
]
```

□ STEP 5: MAIN PROJECT URL

□ myproject/urls.py

```
from django.contrib import admin
from django.urls import path, include

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

□ STEP 6: RUN SERVER

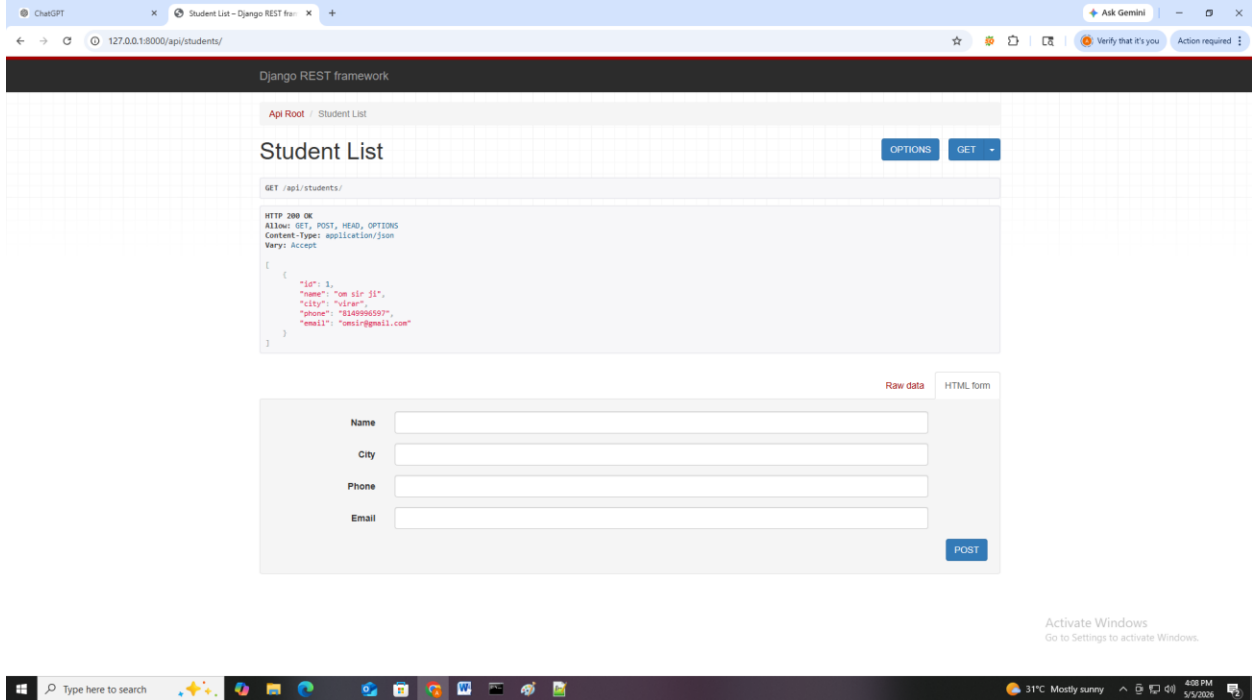
```
python manage.py runserver
```

□ STEP 7: TEST API

Now open browser/Postman:

List Students GET :-

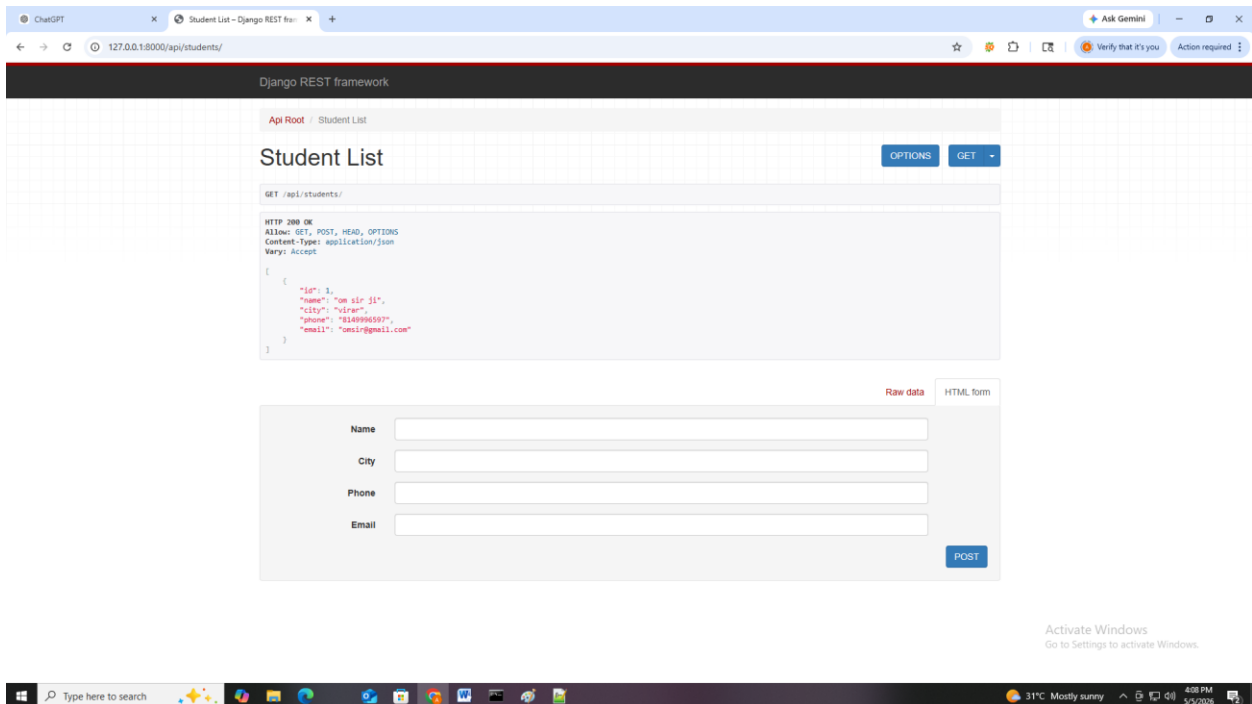
<http://127.0.0.1:8000/api/students/>



Create Student

POST <http://127.0.0.1:8000/api/students/>

```
{  
  "name": "Rahul",  
  "city": "Mumbai",  
  "phone": "9876543210",  
  "email": "rahul@example.com"  
}
```



Get Single Student GET :-

<http://127.0.0.1:8000/api/students/1/>

The screenshot shows a web browser window with the URL 127.0.0.1:8000/api/students/1/. The page displays the Django REST framework interface for the 'Student Instance' endpoint. The breadcrumb navigation shows 'Api Root' > 'Student List' > 'Student Instance'. The endpoint name is 'Student Instance' with buttons for 'DELETE', 'OPTIONS', and 'GET'. The selected method is 'GET /api/students/1/'. The response is shown as 'HTTP 200 OK' with headers: 'Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS', 'Content-Type: application/json', and 'Vary: Accept'. The JSON response is:

```
{ "id": 1, "name": "om sir ji", "city": "virar", "phone": "8149996597", "email": "omsir@gmail.com" }
```

 Below the response, there are tabs for 'Raw data' and 'HTML form'. The 'HTML form' tab is active, showing a form with the following fields: Name (om sir ji), City (virar), Phone (8149996597), and Email (omsir@gmail.com). A 'PUT' button is located at the bottom right of the form. The Windows taskbar at the bottom shows the search bar, taskbar icons, system tray with weather (31°C Mostly sunny), and date/time (4:09 PM 5/3/2025).

Update Student PUT :-

<http://127.0.0.1:8000/api/students/1/>

The screenshot shows the same Django REST framework API browser interface as above, but with the 'PUT' button selected. The breadcrumb navigation and endpoint name remain the same. The selected method is 'PUT /api/students/1/'. The response is shown as 'HTTP 200 OK' with headers: 'Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS', 'Content-Type: application/json', and 'Vary: Accept'. The JSON response is:

```
{ "id": 1, "name": "om sir ji", "city": "virar", "phone": "8149996597", "email": "omsir@gmail.com" }
```

 Below the response, there are tabs for 'Raw data' and 'HTML form'. The 'HTML form' tab is active, showing a form with the following fields: Name (om sir ji), City (virar), Phone (8149996597), and Email (omsir@gmail.com). A 'PUT' button is located at the bottom right of the form. The Windows taskbar at the bottom shows the search bar, taskbar icons, system tray with weather (31°C Mostly sunny), and date/time (4:09 PM 5/3/2025).

Delete Student

DELETE /api/students/1/

The screenshot shows a web browser window displaying the Django REST framework interface. The browser's address bar shows the URL `127.0.0.1:8000/api/students/1/`. The page title is "Django REST framework". The breadcrumb navigation shows "Api Root" > "Student List" > "Student Instance". The main heading is "Student Instance". To the right of the heading are buttons for "DELETE", "OPTIONS", and "GET". Below the heading, the URL `GET /api/students/1/` is shown. The response details are as follows:

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

{
  "id": 1,
  "name": "om sir ji",
  "city": "virar",
  "phone": "8149996597",
  "email": "omsir@gmail.com"
}
```

Below the response details, there are tabs for "Raw data" and "HTML form". The "HTML form" tab is active, showing a form with the following fields:

- Name:
- City:
- Phone:
- Email:

A "PUT" button is located at the bottom right of the form. In the bottom right corner of the browser window, there is a notification that says "Activate Windows Go to Settings to activate Windows." The Windows taskbar is visible at the bottom of the screen, showing the search bar, taskbar icons, and system tray with the date and time (4:09 PM, 5/5/2026).

1. serializers.py (Meaning of each line)

Code:

```
from rest_framework import serializers
from .models import Student

class StudentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Student
        fields = '__all__'
```

Line-by-line meaning

Line 1

```
from rest_framework import serializers
```

Imports Django REST Framework's serializer module.

✓ Serializers convert:

- Python objects ↔ JSON (API format)
-

Line 2

```
from .models import Student
```

Imports your database table (model) named `Student` from the same app.

Line 3

```
class StudentSerializer(serializers.ModelSerializer):
```

Creates a serializer class.

✓ `ModelSerializer` means:

- Django automatically generates fields from model
 - No need to manually define each field
-

Line 4-7

```
class Meta:  
    model = Student  
    fields = '__all__'
```

Meaning:

✓ *class Meta*

Special inner class that gives configuration to serializer

✓ *model = Student*

Tells serializer:

“Use Student model (database table)”

✓ *fields = '__all__'*

Include ALL fields:

- id
- name
- city
- phone
- email

✓ Alternative:

```
fields = ['name', 'email']
```

□ 2. views.py (ModelViewSet)

Code:

```
from rest_framework import viewsets
from .models import Student
from .serializers import StudentSerializer

class StudentViewSet(viewsets.ModelViewSet):
    queryset = Student.objects.all()
    serializer_class = StudentSerializer
```

Line-by-line meaning

□ Line 1

```
from rest_framework import viewsets
```

□ Imports ViewSet system from DRF

✓ ViewSet = combines all CRUD operations:

- GET
 - POST
 - PUT
 - DELETE
-

□ Line 2

```
from .models import Student
```

□ Imports database table (Student model)

□ Line 3

```
from .serializers import StudentSerializer
```

□ Imports serializer to convert:

- Model ↔ JSON

Line 4

```
class StudentViewSet(viewsets.ModelViewSet):
```

Creates a `ViewSet` class

✓ `ModelViewSet` means:

Django automatically gives full CRUD API without writing methods

Line 5

```
queryset = Student.objects.all()
```

Meaning:

- Fetch ALL students from database
- Equivalent to SQL:

```
SELECT * FROM student;
```

Line 6

```
serializer_class = StudentSerializer
```

Tells `ViewSet`:

“Use this serializer to convert data to/from JSON”

What this `ViewSet` automatically gives you:

Action	Method	URL
List	GET	/students/
Create	POST	/students/
Retrieve	GET	/students/1/

Action	Method	URL
Update	PUT	/students/1/
Delete	DELETE	/students/1/

3. urls.py (Router system)

Code:

```
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import StudentViewSet

router = DefaultRouter()
router.register('students', StudentViewSet, basename='student')

urlpatterns = [
    path('', include(router.urls)),
]
```

Line-by-line meaning

Line 1

```
from django.urls import path, include
```

Imports URL tools:

- `path()` → define routes
 - `include()` → connect other URL files
-

Line 2

```
from rest_framework.routers import DefaultRouter
```

Imports Router system from DRF

✓ Router automatically generates URLs for ViewSets

Line 3

```
from .views import StudentViewSet
```

Imports your ViewSet (API logic)

Line 4

```
router = DefaultRouter()
```

Creates a router object

✓ Think of it as:

“Auto URL generator”

Line 5

```
router.register('students', StudentViewSet, basename='student')
```

Meaning:

- 'students' → URL name
- StudentViewSet → logic to handle requests
- basename='student' → name used internally for URL naming

✓ This creates URLs like:

- /students/
 - /students/1/
-

Line 6–8

```
urlpatterns = [  
    path('', include(router.urls)),  
]
```

Meaning:

- `router.urls` contains all auto-generated URLs
- `include()` attaches them to project routing

□ FINAL SIMPLE SUMMARY

serializer.py

- Converts data:

Database ↔ JSON

□ IMPORTANT (CORS ERROR FIX)

If API doesn't work, install Django CORS:

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

Add in Django `settings.py`

```
INSTALLED_APPS = [  
    ...  
    'corsheaders',  
]  
  
MIDDLEWARE = [  
    'corsheaders.middleware.CorsMiddleware',  
    ...  
]  
  
CORS_ALLOWED_ORIGINS = [  
    "http://localhost:5173", # Vite default  
]  
]
```

React + Vite + Django REST API.

We'll extend your current setup using React and Vite.

STEP 1: Update API function

`src/api/studentApi.js`

Great—now you'll connect your Django REST API to a modern React frontend using Vite and React.

We'll build a simple **Student CRUD frontend** for your API:

STEP 1: Create React + Vite Project

```
npm create vite@latest student-frontend
```

Select:

- Framework: **React**
 - Variant: **JavaScript**
-

Go inside project

```
cd student-frontend  
npm install
```

Install Axios (for API calls)

```
npm install axios
```

□ STEP 2: Clean Project Structure

Inside `src/` create:

```
src/
├── api/
│   └── studentApi.js
├── components/
│   ├── StudentForm.jsx
│   └── StudentList.jsx
├── App.jsx
└── main.jsx
```

□ STEP 3: Create API Service

□ `src/api/studentApi.js`

```
import axios from "axios";

const BASE_URL = "http://127.0.0.1:8000/api/students/";

export const getStudents = () => axios.get(BASE_URL);

export const createStudent = (data) => axios.post(BASE_URL, data);

export const updateStudent = (id, data) =>
  axios.put(`${BASE_URL}${id}/`, data);

export const deleteStudent = (id) =>
  axios.delete(`${BASE_URL}${id}/`);
```

STEP 2: Add Edit State in App.jsx

We need to track:

- which student is being edited
 - form pre-filled with old data
-

□ `src/App.jsx`

Update code:

```

import { useEffect, useState } from "react";
import { getStudents } from "../api/studentApi";
import StudentForm from "../components/StudentForm";
import StudentList from "../components/StudentList";

function App() {
  const [students, setStudents] = useState([]);
  const [editStudent, setEditStudent] = useState(null);

  const fetchStudents = async () => {
    const res = await getStudents();
    setStudents(res.data);
  };

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

  return (
    <div>
      <h1>Student Management System</h1>

      <StudentForm
        refresh={fetchStudents}
        editStudent={editStudent}
        setEditStudent={setEditStudent}
      />

      <StudentList
        students={students}
        refresh={fetchStudents}
        setEditStudent={setEditStudent}
      />
    </div>
  );
}

export default App;

```

STEP 3: Update StudentForm (Create + Update)

□ `src/components/StudentForm.jsx`

```

import { useEffect, useState } from "react";
import { createStudent, updateStudent } from "../api/studentApi";

```

```

export default function StudentForm({ refresh, editStudent, setEditStudent })
{
  const [form, setForm] = useState({
    name: "",
    city: "",
    phone: "",
    email: ""
  });

  // Fill form when editing
  useEffect(() => {
    if (editStudent) {
      setForm(editStudent);
    }
  }, [editStudent]);

  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

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

    if (editStudent) {
      // UPDATE
      await updateStudent(editStudent.id, form);
      setEditStudent(null);
    } else {
      // CREATE
      await createStudent(form);
    }

    setForm({ name: "", city: "", phone: "", email: "" });
    refresh();
  };

  return (
    <form onSubmit={handleSubmit}>
      <h2>{editStudent ? "Update Student" : "Add Student"}</h2>

      <input name="name" placeholder="Name" value={form.name}
onChange={handleChange} />
      <input name="city" placeholder="City" value={form.city}
onChange={handleChange} />
      <input name="phone" placeholder="Phone" value={form.phone}
onChange={handleChange} />
      <input name="email" placeholder="Email" value={form.email}
onChange={handleChange} />

      <button type="submit">
        {editStudent ? "Update" : "Add"}
      </button>

      {editStudent && (
        <button type="button" onClick={() => setEditStudent(null)}>
          Cancel
        </button>
      )}
    </form>
  );
}

```

```
    })  
  </form>  
);  
}
```

□ **STEP 4: Add Edit Button in List**

□ `src/components/StudentList.jsx`

```
import { deleteStudent } from "../api/studentApi";  
  
export default function StudentList({ students, refresh, setEditStudent }) {  
  
  const handleDelete = async (id) => {  
    await deleteStudent(id);  
    refresh();  
  };  
  
  return (  
    <div>  
      <h2>Student List</h2>  
  
      {students.map((s) => (  
        <div key={s.id} style={{ border: "1px solid gray", margin: 10,  
padding: 10 }}>  
          <p>Name: {s.name}</p>  
          <p>City: {s.city}</p>  
          <p>Phone: {s.phone}</p>  
          <p>Email: {s.email}</p>  
  
          { /* EDIT BUTTON */ }  
          <button onClick={() => setEditStudent(s)}>  
            Edit  
          </button>  
  
          { /* DELETE BUTTON */ }  
          <button onClick={() => handleDelete(s.id)}>  
            Delete  
          </button>  
        </div>  
      ))}  
    </div>  
  );  
}
```

□ **FINAL FEATURES YOU NOW HAVE**

📄 FULL CRUD SYSTEM

Create

- Add new student

Read

- Show all students

Update (NEW)

- Click Edit → form fills → update data

Delete

- Remove student

Run django api first:-

python manage.py runserver

and then run frontend:-

npm run dev

Student Management System

Update Student

om sir ji | virar | 8149996597 | omsir@gmail.com | Update | Cancel

Student List

Name: om sir ji
City: virar
Phone: 8149996597
Email: omsir@gmail.com
[Edit] [Delete]

Name: raj sir
City: andheri
Phone: 9322437432
Email: rajsingh@gmail.com
[Edit] [Delete]

Activate Windows
Go to Settings to activate Windows.

a **single-file React frontend** (easy to read, easy to teach) that connects to your Django API.

This is perfect for beginners using React with Vite.

□ □ ONE-FILE FRONTEND (FULL CRUD)

□ Replace your `src/App.jsx` with this:

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

const BASE_URL = "http://127.0.0.1:8000/api/students/";

function App() {
  const [students, setStudents] = useState([]);
  const [form, setForm] = useState({
    name: "",
    city: "",
    phone: "",
    email: ""
  });
  const [editId, setEditId] = useState(null);

  // □ Fetch all students
  const fetchStudents = async () => {
    const res = await axios.get(BASE_URL);
    setStudents(res.data);
  };

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

  // □ Handle input change
  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  // □ Create or Update
  const handleSubmit = async (e) => {
    e.preventDefault();

    if (editId) {
      // UPDATE
      await axios.put(`${BASE_URL}${editId}/`, form);
      setEditId(null);
    } else {
      // CREATE
      await axios.post(BASE_URL, form);
    }
  }
}
```

```

    setForm({ name: "", city: "", phone: "", email: "" });
    fetchStudents();
  };

  // □ Delete
  const handleDelete = async (id) => {
    await axios.delete(`${BASE_URL}${id}/`);
    fetchStudents();
  };

  // □ Edit (fill form)
  const handleEdit = (student) => {
    setForm(student);
    setEditId(student.id);
  };

  return (
    <div style={{ padding: 20 }}>
      <h1>Student Management System</h1>

      { /* FORM */ }
      <form onSubmit={handleSubmit}>
        <input name="name" placeholder="Name" value={form.name}
onChange={handleChange} />
        <input name="city" placeholder="City" value={form.city}
onChange={handleChange} />
        <input name="phone" placeholder="Phone" value={form.phone}
onChange={handleChange} />
        <input name="email" placeholder="Email" value={form.email}
onChange={handleChange} />

        <button type="submit">
          {editId ? "Update Student" : "Add Student"}
        </button>

        {editId && (
          <button type="button" onClick={() => {
            setEditId(null);
            setForm({ name: "", city: "", phone: "", email: "" });
          }}>
            Cancel
          </button>
        )}
      </form>

      <hr />

      { /* LIST */ }
      <h2>Student List</h2>

      {students.map((s) => (
        <div key={s.id} style={{ border: "1px solid gray", margin: 10,
padding: 10 }}>
          <p><b>Name:</b> {s.name}</p>
          <p><b>City:</b> {s.city}</p>
          <p><b>Phone:</b> {s.phone}</p>
          <p><b>Email:</b> {s.email}</p>
        </div>
      ))}
    </div>
  );

```

```
        <button onClick={() => handleEdit(s)}>Edit</button>
        <button onClick={() => handleDelete(s.id)}>Delete</button>
      </div>
    )}
  </div>
);
}

export default App;
```

□ HOW TO RUN

1. Install axios

```
npm install axios
```

2. Run React

```
npm run dev
```

3. Make sure Django is running

```
python manage.py runserver
```

□ WHAT THIS SINGLE FILE DOES

🔍 Create

- Add new student

🔍 Read

- Fetch and display all students

🔍 Update

- Click **Edit** → **Update**

🔍 Delete

- Remove student

