**1. Understanding API Integration**

ReactJS can communicate with backend services (like ASP.NET Core Web API) using HTTP requests.  
The most common methods used are:

* **GET** → Retrieve data
* **POST** → Add data
* **PUT** → Update data
* **DELETE** → Remove data

We use JavaScript’s fetch() function or the Axios library to send these requests.

**2. Backend API Overview**

From previous weeks, your ASP.NET Core backend exposes endpoints like:

|  |  |  |
| --- | --- | --- |
| **Method** | **Endpoint** | **Description** |
| GET | /api/students | Get all students |
| GET | /api/students/{id} | Get a specific student |
| POST | /api/students | Add a new student |
| PUT | /api/students/{id} | Update a student |
| DELETE | /api/students/{id} | Delete a student |

Your frontend React app will consume these endpoints.

**3. Setting Up API Configuration in React**

Create a file named **api.js** in your React app under src/services/ to define the base URL of your API.

Example:

// src/services/api.js

const API\_BASE\_URL = "http://localhost:5205/api/students";

export default API\_BASE\_URL;

This allows centralized control of the backend URL so that it can be reused in multiple components.

**4. Fetching Data (GET Request)**

To display all students, use the Fetch API inside a React component (like StudentList.js):

Example:

useEffect(() => {

fetch("http://localhost:5205/api/students")

.then(response => response.json())

.then(data => setStudents(data))

.catch(error => console.error("Error fetching students:", error));

}, []);

This code calls your API and sets the response data in a React state variable using setStudents.

**5. Adding Data (POST Request)**

When submitting a form, you can send a POST request to add a new student:

const handleAddStudent = async (student) => {

try {

const response = await fetch("http://localhost:5205/api/students", {

method: "POST",

headers: {

"Content-Type": "application/json"

},

body: JSON.stringify(student)

});

if (response.ok) {

alert("Student added successfully");

}

} catch (error) {

console.error("Error adding student:", error);

}

};

**6. Updating Data (PUT Request)**

To edit student details:

const handleUpdateStudent = async (id, student) => {

try {

const response = await fetch(`http://localhost:5205/api/students/${id}`, {

method: "PUT",

headers: {

"Content-Type": "application/json"

},

body: JSON.stringify(student)

});

if (response.ok) {

alert("Student updated successfully”) }

} catch (error) {

console.error("Error updating student:", error);

}

};

**7. Deleting Data (DELETE Request)**

To remove a student record:

const handleDeleteStudent = async (id) => {

try {

const response = await fetch(`http://localhost:5205/api/students/${id}`, {

method: "DELETE" });

if (response.ok) {

alert("Student deleted successfully”) }

} catch (error) {

console.error("Error deleting student:", error);

} };

**8. Handling API Responses**

When integrating APIs, always handle:

* **Loading state:** Show a loading message or spinner while fetching data.
* **Error state:** Display user-friendly error messages.
* **Success state:** Refresh data after a successful update, addition, or deletion.

**9. Example Folder Structure**

student-management-app/

├── src/

│ ├── components/

│ │ ├── StudentList.js

│ │ ├── AddStudentForm.js

│ │ ├── EditStudentForm.js

│ ├── services/

│ │ └── api.js

│ ├── App.js

│ ├── index.js

│ ├── App.css

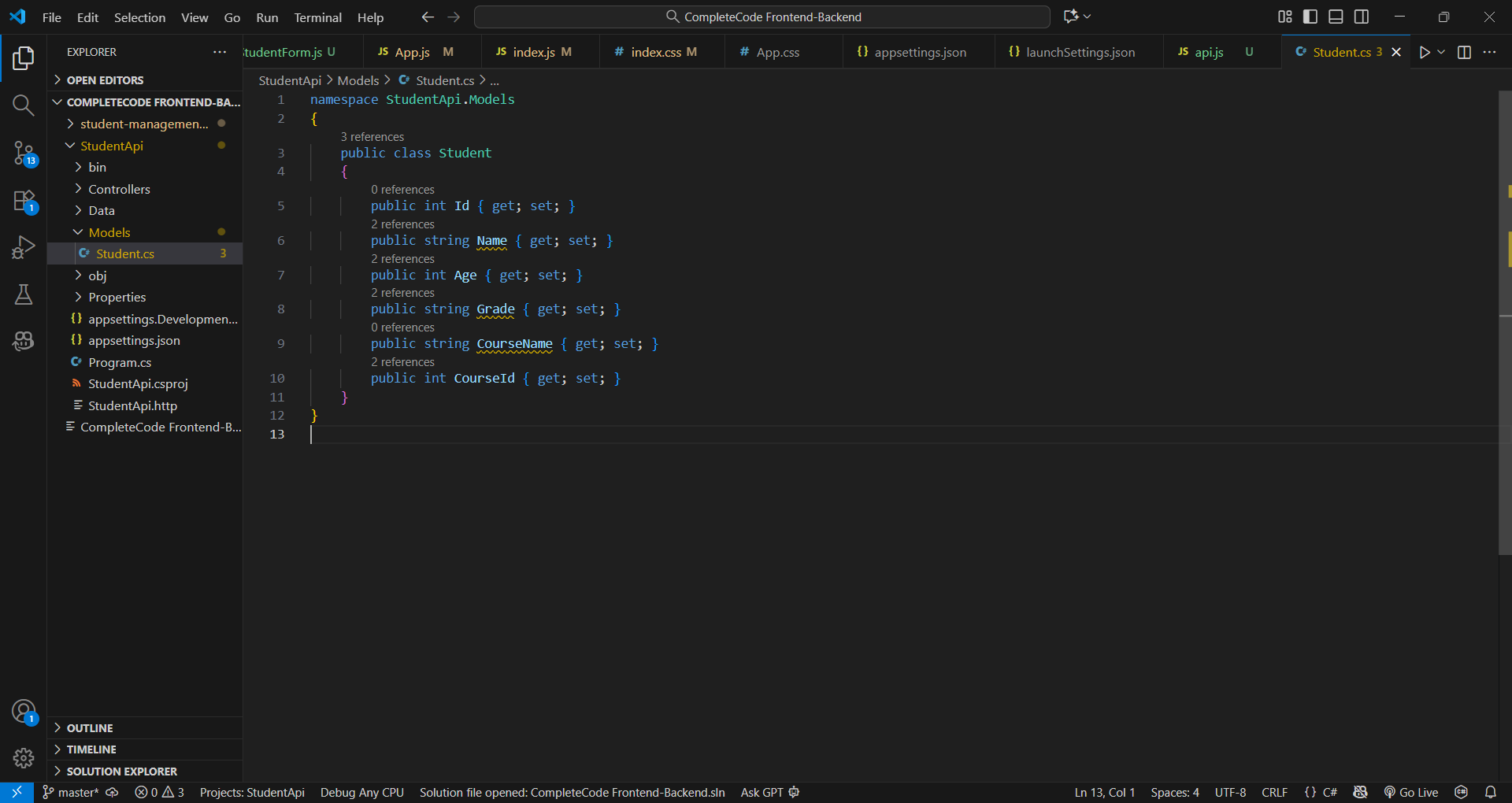
│ └── index.css

**Conclusion**

By the end of this lesson, you should understand how to:

* Connect your ReactJS frontend to a backend API.
* Perform CRUD operations using Fetch.
* Handle user input, loading, and error states properly.
* Prepare for the final integration (Day 7) where the complete frontend and backend will be combined into a functional full-stack application.

**Snapshots:**



Student.cs class with all properties.

A screenshot of a computer

AI-generated content may be incorrect.

Code in ApplicationDbContext.cs linking to Students table.

A screenshot of a computer

AI-generated content may be incorrect.

SQL Server connection string added to appsettings.json.

A screen shot of a computer

AI-generated content may be incorrect.

Middleware, CORS, and EF Core services configured.

A screenshot of a computer

AI-generated content may be incorrect.

Swagger UI running at http://localhost:5205/swagger.

A screenshot of a computer

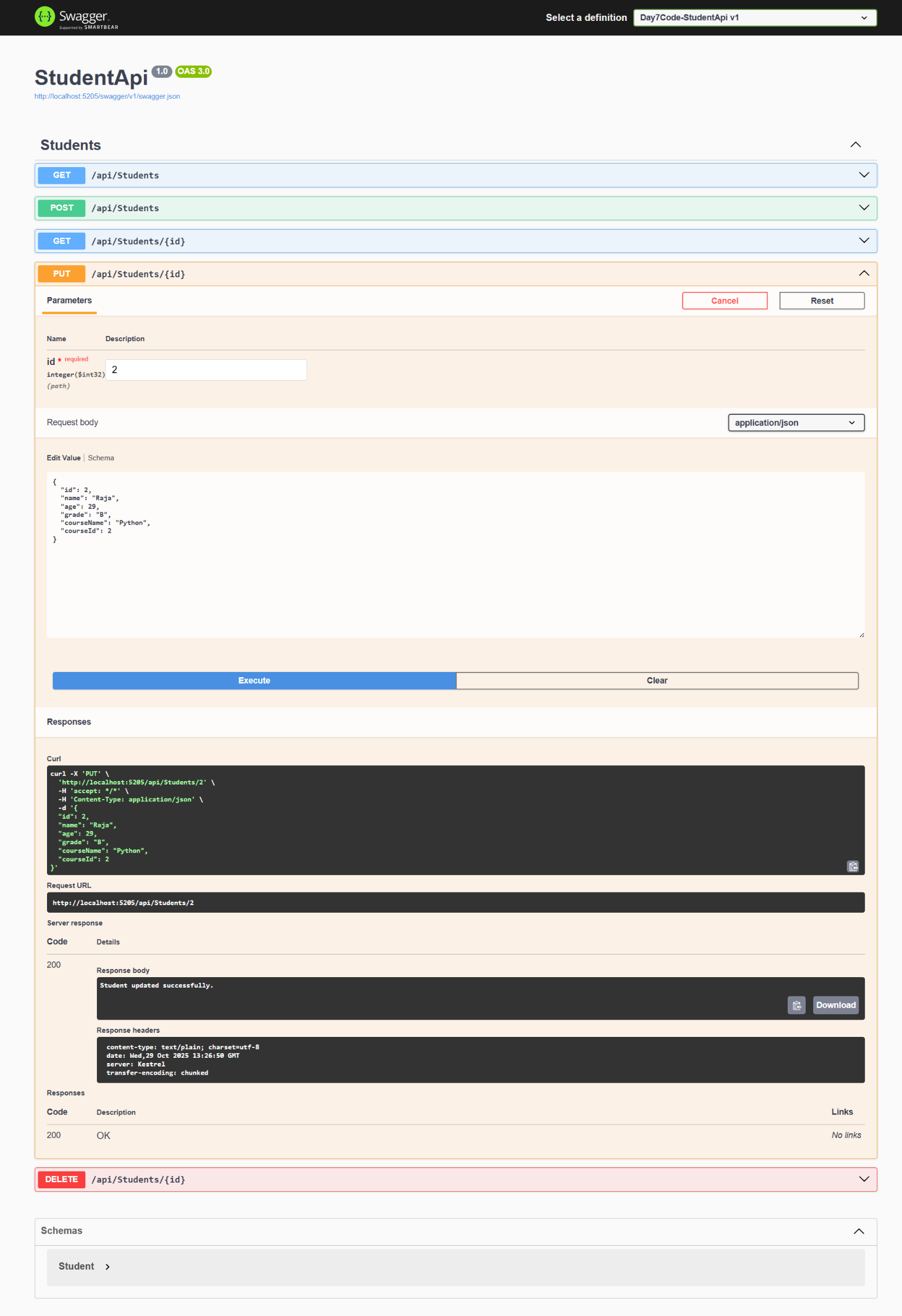
AI-generated content may be incorrect.

Successful response listing students from database.

A screenshot of a computer

AI-generated content may be incorrect.

New student record added through POST endpoint.

Student record updated successfully.

A screenshot of a computer

AI-generated content may be incorrect.

Student entry deleted via DELETE endpoint.

A screen shot of a computer

AI-generated content may be incorrect.

components and services folders created under src/.

A screenshot of a computer

AI-generated content may be incorrect.

|  |
| --- |
|  |

|  |
| --- |
| Code for api.js connecting React to backend API. |

A screenshot of a computer

AI-generated content may be incorrect.

Displays list of students in React UI.

A screenshot of a computer

AI-generated content may be incorrect.

Form UI for adding new student data.

A screenshot of a computer

AI-generated content may be incorrect.

Edit page showing existing data for update.

A screenshot of a computer

AI-generated content may be incorrect.

Edit page showing existing data for delete.