## Day 6 – ReactJS API Integration (Connecting Frontend with Backend) **Date:** 28-10-2025

**Objective:** To learn how to connect a ReactJS frontend with a backend API (ASP.NET Core Web API) using the Fetch API and handle CRUD (Create, Read, Update, Delete) operations.

### 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**

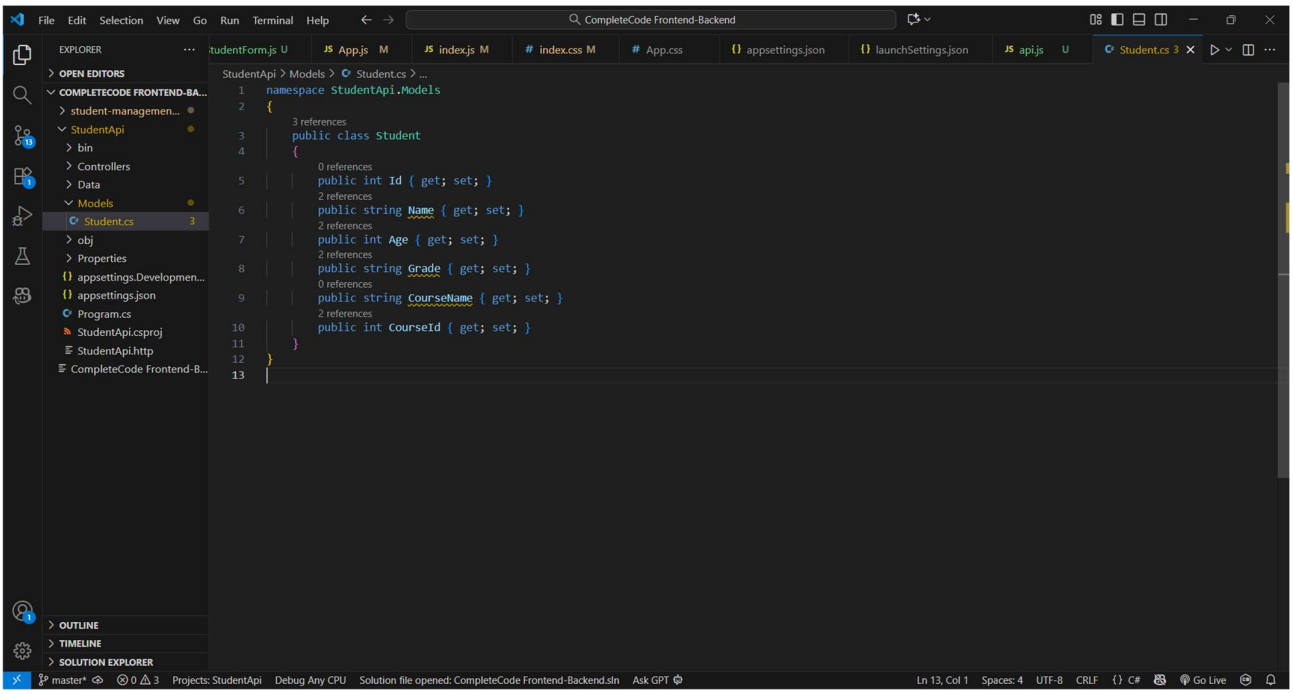
```
student-management-app/
├── src/
│   ├── components/
│   │   ├── StudentList.js
│   │   ├── AddStudentForm.js
│   │   ├── EditStudentForm.js
│   ├── services/
│   │   └── api.js
│   ├── App.js
│   ├── index.js
│   ├── App.css
│   └── index.css
```

**Conclusion**

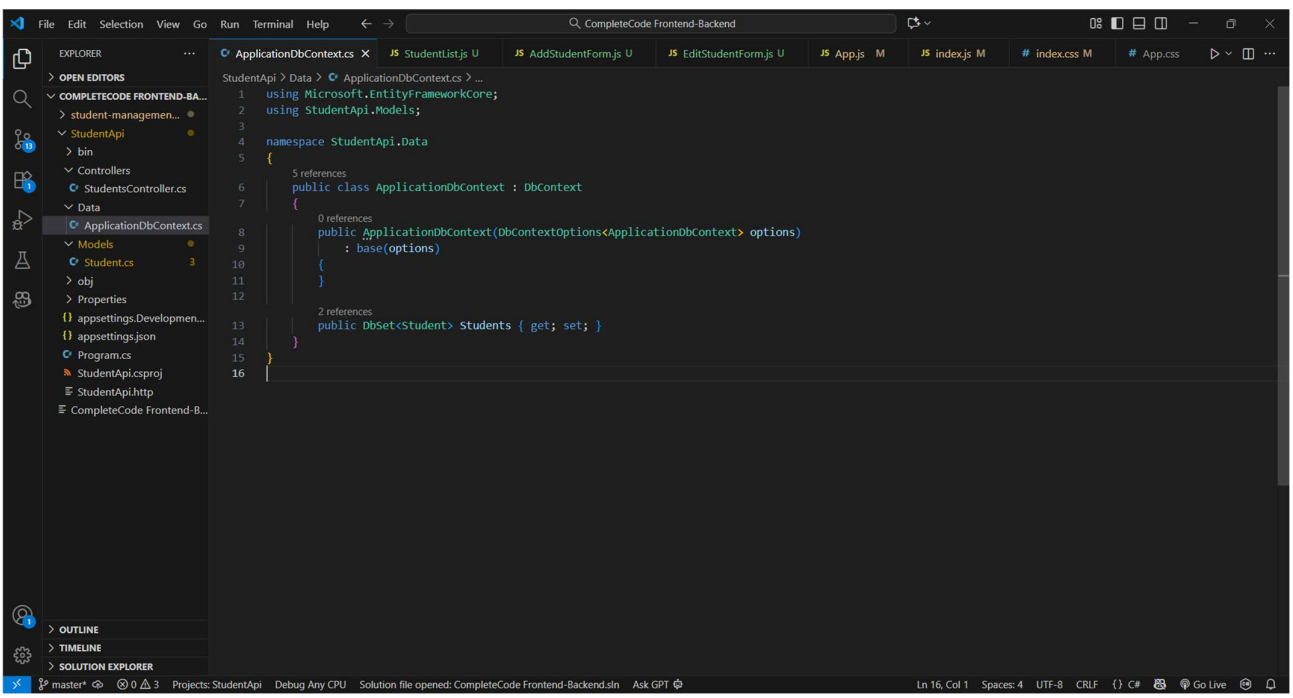
By the end of this lesson, you should understand how to:

- 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.



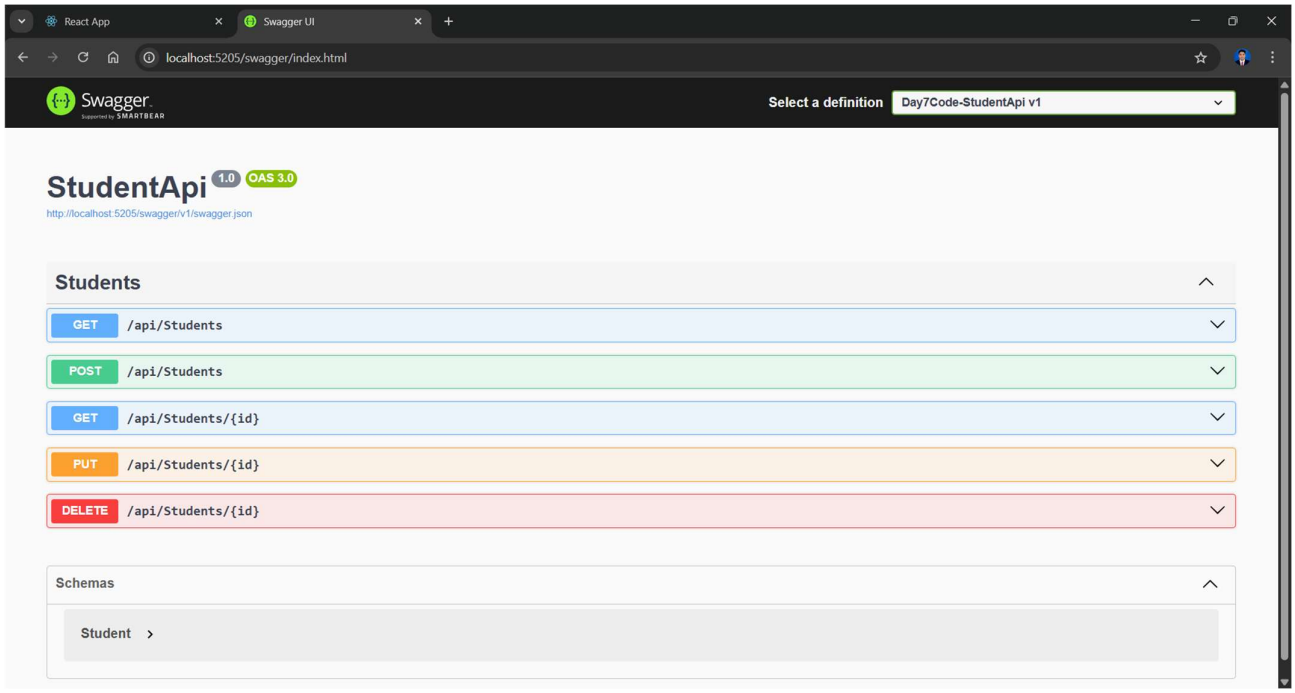

Code in ApplicationDbContext.cs linking to Students table.

```
StudentApi > {} appsettings.json > ...
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost;Database=StudentCourseDB;TrustServerCertificate=true;Trusted_Connection=True;"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

SQL Server connection string added to appsettings.json.

```
StudentApi > Program.cs > Program > <top-level-statements-entry-point>
var builder = WebApplication.CreateBuilder(args);
using Microsoft.EntityFrameworkCore;
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Configure Database Connection
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

// Enable CORS for React
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowReactApp",
        builder => builder.WithOrigins("http://localhost:3000")
                          .AllowAnyMethod()
                          .AllowAnyHeader());
});

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseCors("AllowReactApp");
app.UseAuthorization();
app.MapControllers();
app.Run();
```

Middleware, CORS, and EF Core services configured.




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

Swagger

Select a definition Day7Code-StudentApi v1

# StudentApi 1.0 OAS 3.0

http://localhost:5205/swagger/v1/swagger.json

## Students

GET /api/Students

Parameters Cancel

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5205/api/Students' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:5205/api/Students
```

Server response

| Code | Details |
|---|---|
| 200 | Response body |


```
[
  {
    "id": 2,
    "name": "Sangeeta",
    "age": 23,
    "grade": "B",
    "courseName": "ASP.NET Core Development",
    "courseId": 2
  },
  {
    "id": 3,
    "name": "Sneha R",
    "age": 23,
    "grade": "A",
    "courseName": "SQL & Database Management",
    "courseId": 3
  },
  {
    "id": 4,
    "name": "Nikhil Rao",
    "age": 22,
    "grade": "B",
    "courseName": "ASP.NET Core Development",
    "courseId": 2
  },
  {
    "id": 5,
```


Download

Response headers

```
content-type: application/json; charset=utf-8
date: Wed,29 Oct 2025 13:25:05 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

| Code | Description | Links |
|---|---|---|
| 200 | OK | *No links* |

POST /api/Students

GET /api/Students/{id}

PUT /api/Students/{id}

DELETE /api/Students/{id}

Schemas

Student ›

Successful response listing students from database.

Swagger

Select a definition: Day7Code-StudentApi v1

# StudentApi 1.0 OAS 3.0

http://localhost:5205/swagger/v1/swagger.json

## Students

GET /api/Students

POST /api/Students

Parameters | Cancel | Reset

No parameters

Request body — application/json

Edit Value | Schema

```
{
  "id": 1,
  "name": "Uday",
  "age": 21,
  "grade": "A",
  "courseName": "Java",
  "courseId": 1
}
```

Execute | Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:5205/api/Students' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": 1,
  "name": "Uday",
  "age": 21,
  "grade": "A",
  "courseName": "Java",
  "courseId": 1
}'
```

Request URL

```
http://localhost:5205/api/Students
```

Server response

| Code | Details |
|---|---|
| 200 | Response body: `Student added successfully.` (Download) |

Response headers

```
content-type: text/plain; charset=utf-8
date: Wed,29 Oct 2025 13:25:56 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

| Code | Description | Links |
|---|---|---|
| 200 | OK | No links |

GET /api/Students/{id}

PUT /api/Students/{id}

DELETE /api/Students/{id}

New student record added through POST endpoint.

Swagger

Select a definition: Day7Code-StudentApi v1

# StudentApi 1.0 OAS 3.0

http://localhost:5205/swagger/v1/swagger.json

## Students

GET /api/Students

POST /api/Students

GET /api/Students/{id}

PUT /api/Students/{id}

Parameters

Cancel Reset

| Name | Description |
|---|---|
| id * required<br>integer($int32)<br>(path) | 2 |

Request body

application/json

Edit Value | Schema

```
{
  "id": 2,
  "name": "Raja",
  "age": 29,
  "grade": "B",
  "courseName": "Python",
  "courseId": 2
}
```

Execute Clear

Responses

Curl

```
curl -X 'PUT' \
  'http://localhost:5205/api/Students/2' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": 2,
  "name": "Raja",
  "age": 29,
  "grade": "B",
  "courseName": "Python",
  "courseId": 2
}'
```

Request URL

```
http://localhost:5205/api/Students/2
```

Server response

| Code | Details |
|---|---|
| 200 | Response body<br>`Student updated successfully.`<br>Download<br>Response headers<br>`content-type: text/plain; charset=utf-8`<br>`date: Wed,29 Oct 2025 13:26:50 GMT`<br>`server: Kestrel`<br>`transfer-encoding: chunked` |

Responses

| Code | Description | Links |
|---|---|---|
| 200 | OK | *No links* |

DELETE /api/Students/{id}

Student record updated successfully.

Swagger

Select a definition Day7Code-StudentApi v1

# StudentApi 1.0 OAS 3.0

http://localhost:5205/swagger/v1/swagger.json

## Students

GET /api/Students

POST /api/Students

GET /api/Students/{id}

PUT /api/Students/{id}

DELETE /api/Students/{id}

Parameters

Cancel

| Name | Description |
|---|---|
| id * required<br>integer($int32)<br>(path) | 2 |

Execute Clear

Responses

Curl

```
curl -X 'DELETE' \
  'http://localhost:5205/api/Students/2' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:5205/api/Students/2
```

Server response

| Code | Details |
|---|---|
| 200 | Response body<br>`Student deleted successfully.`<br>Download<br>Response headers<br>`content-type: text/plain; charset=utf-8`<br>`date: Wed,29 Oct 2025 13:27:15 GMT`<br>`server: Kestrel`<br>`transfer-encoding: chunked` |

Responses

| Code | Description | Links |
|---|---|---|
| 200 | OK | *No links* |

Schemas

Student ›

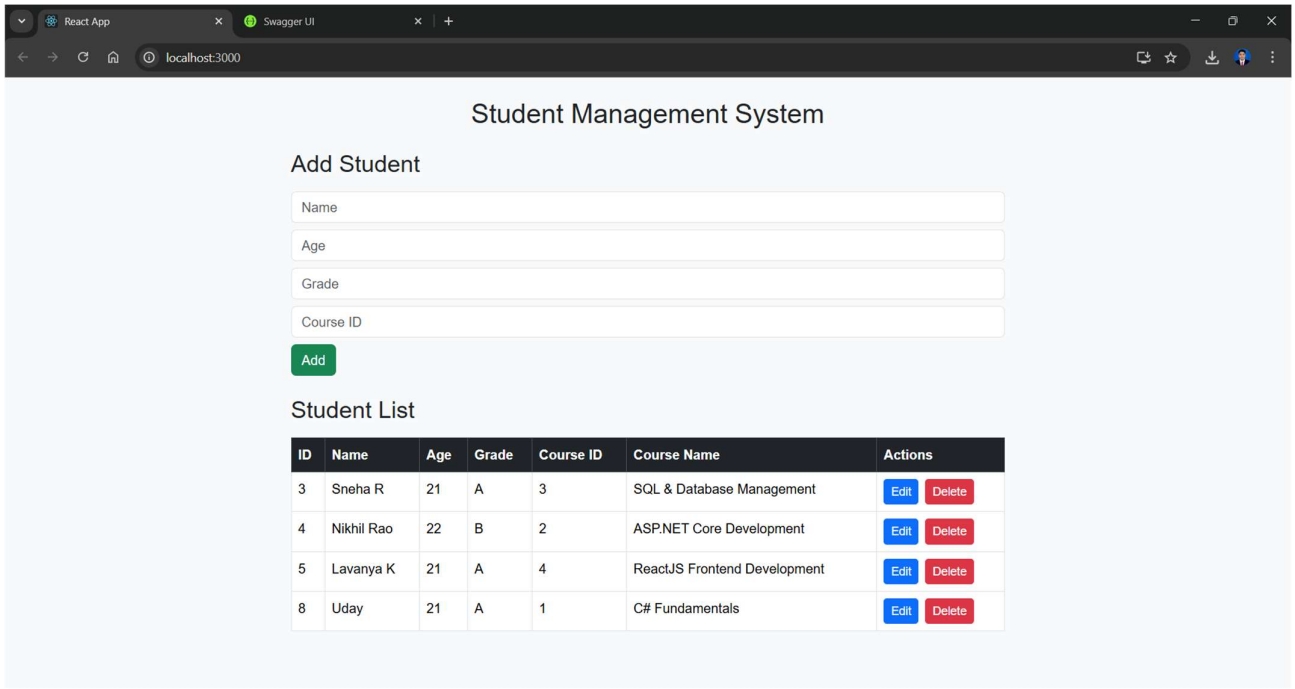
Student entry deleted via DELETE endpoint.

```
import React, { useEffect, useState } from "react";
import API_BASE_URL from "../services/api";

function StudentList({ onEdit }) {
  const [students, setStudents] = useState([]);
  const [loading, setLoading] = useState(true);

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

  const fetchStudents = async () => {
    try {
      const response = await fetch(API_BASE_URL);
      const data = await response.json();
      setStudents(data);
    } catch (error) {
      console.error("Error fetching students:", error);
    } finally {
      setLoading(false);
    }
  };

  const handleDelete = async (id) => {
    if (!window.confirm("Are you sure you want to delete this student?")) return;
    try {
      const response = await fetch(`${API_BASE_URL}/${id}`, {
        method: "DELETE",
      });
      if (response.ok) {
        alert("Student deleted successfully.");
        fetchStudents();
      }
    } catch (error) {
      console.error("Error deleting student:", error);
    }
  };
```

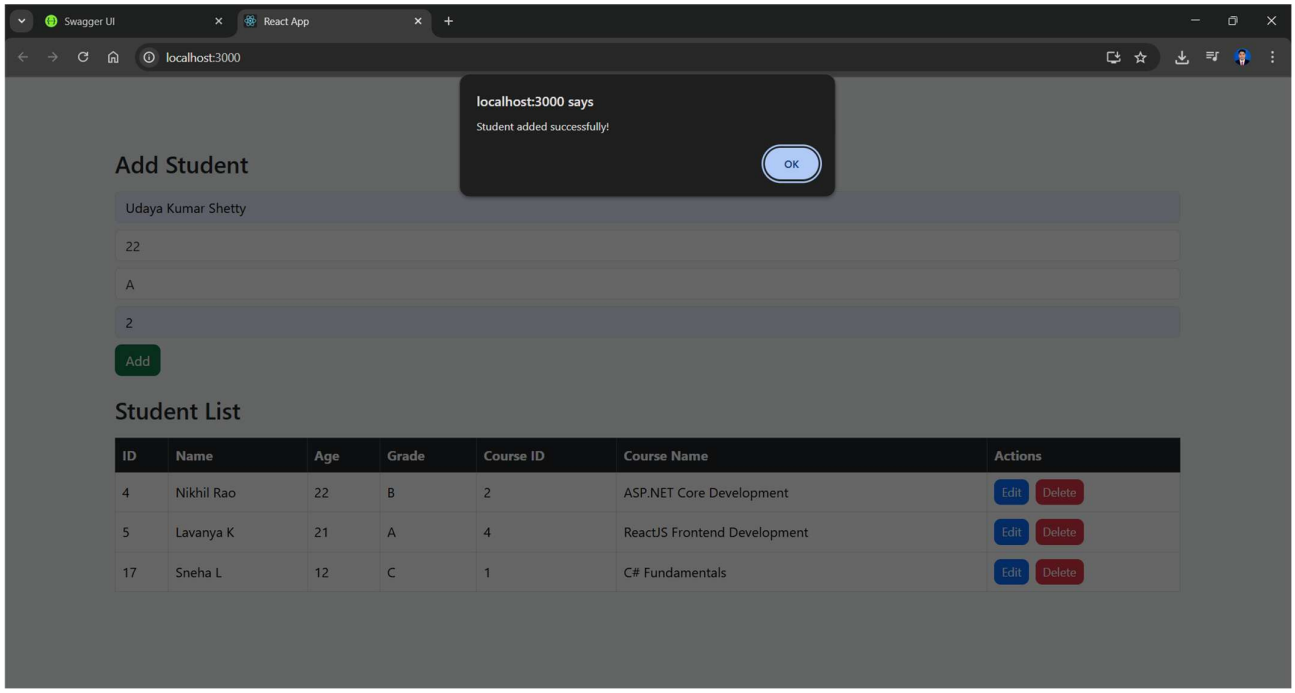
components and services folders created under src/.

```
// src/services/api.js
const API_BASE_URL = "http://localhost:5205/api/students";

export default API_BASE_URL;
```

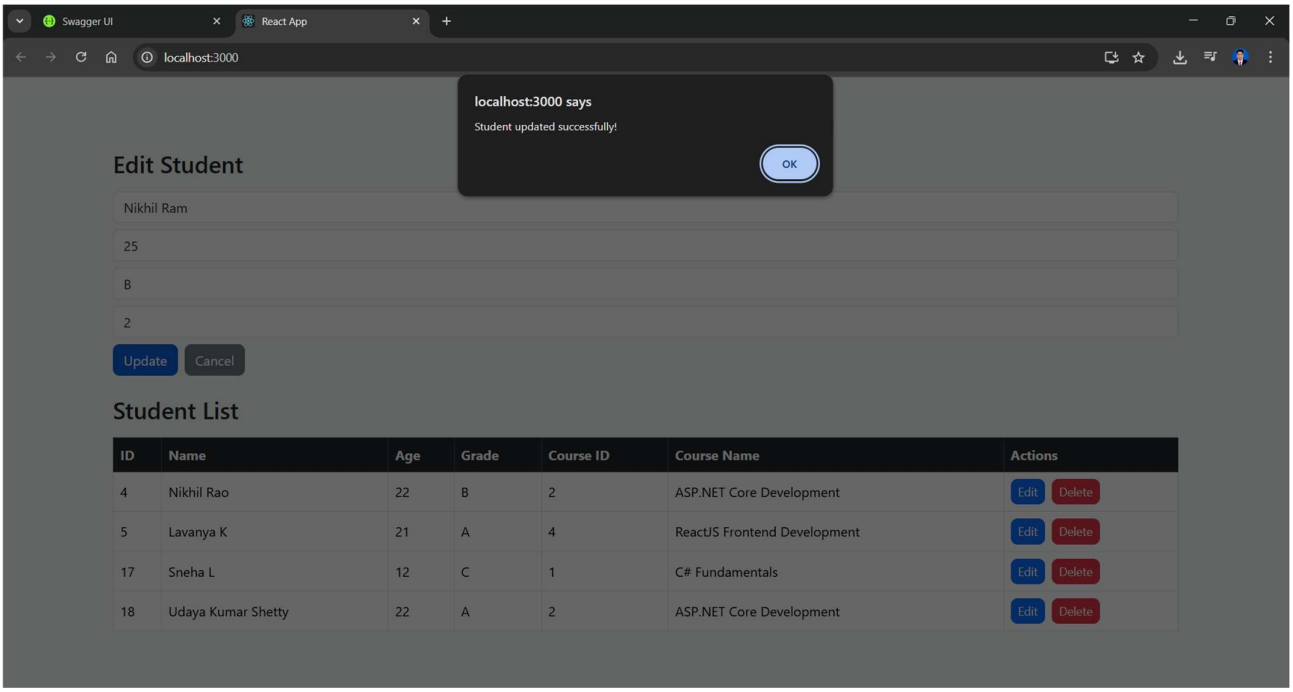
Code for api.js connecting React to backend API.



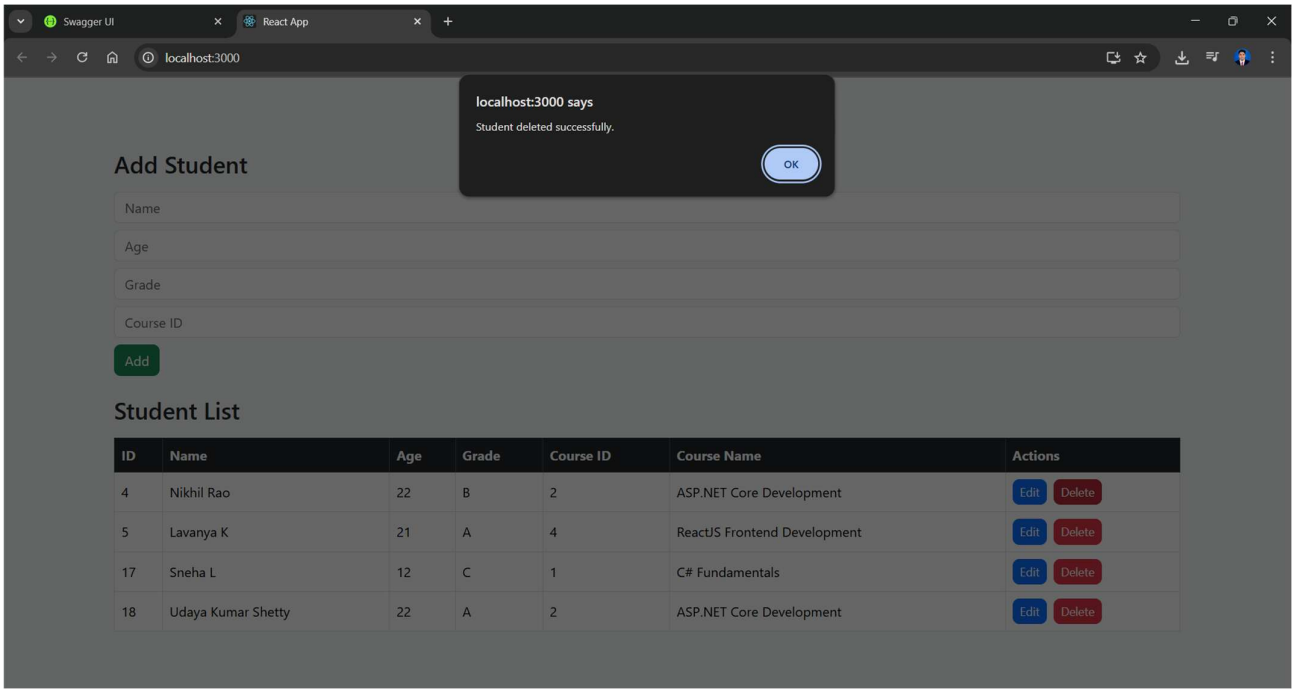

Displays list of students in React UI.




Form UI for adding new student data.




Edit page showing existing data for update.




Edit page showing existing data for delete.