Day 7 – Integration, Validation, and Finalization **Date:** 29-10-2025

Objective: The main goal of Day 7 is to integrate all components of the full-stack application developed in previous days, test CRUD operations between the frontend (ReactJS) and backend (ASP.NET Core Web API), ensure database consistency, and prepare the system for review or deployment.

1. Overview

By Day 7, both backend (ASP.NET Core Web API) and frontend (ReactJS) applications should be fully functional and connected to a SQL Server database.
This phase ensures that every feature works as intended:

- Fetching data (GET)
- Adding data (POST)
- Updating data (PUT)
- Deleting data (DELETE)

It also focuses on implementing validation, improving UI behavior, and testing data flow between layers.

2. Backend Review

The backend API, built using **ASP.NET Core Web API**, serves as the data access layer for the ReactJS application.
It interacts with the SQL Server database using **Entity Framework Core** and stored procedures.

Core Components

1. **Controller:** Handles API routes and HTTP requests (StudentsController.cs).
2. **Model:** Defines data structure (Student.cs).
3. **Database Context:** Manages database connection and entity mapping (ApplicationDbContext.cs).
4. **Stored Procedures:** SQL Server procedures that perform all CRUD operations efficiently.

API Endpoints

Operation	HTTP Method	Endpoint	Description
Get all students	GET	/api/students	Returns all student records
Get student by ID	GET	/api/students/{id}	Returns a single student
Add student	POST	/api/students	Adds a new student record
Update student	PUT	/api/students/{id}	Updates existing record
Delete student	DELETE	/api/students/{id}	Deletes a student record

3. Frontend Review

The frontend, created using **ReactJS**, provides a user-friendly interface to interact with the API.
It communicates with the backend using fetch() calls and handles user actions through forms and buttons.

Main Components

1. **StudentList.js** – Displays all students with Edit and Delete buttons.
2. **AddStudentForm.js** – Form to add new students.
3. **EditStudentForm.js** – Form to update student details.
4. **App.js** – Main entry point integrating all components and managing routes.

Features Implemented

- Dynamic form inputs using useState().
- Fetch API calls for data transfer.
- Page updates after data insertion or deletion.
- Bootstrap used for responsive styling.

4. Validation and Error Handling

Proper validation ensures that the user inputs are correct before sending them to the backend.

Frontend Validation Example

```
if (!form.name || !form.age || !form.courseId) {
  alert("All fields are required");
  return;
}
```

Backend Error Handling Example

```
try
{
    _context.Database.ExecuteSqlRaw("EXEC sp_AddStudent @Name={0}, @Age={1}, @Grade={2},
@CourseId={3}",
    student.Name, student.Age, student.Grade, student.CourseId);
    return Ok("Student added successfully");
}
catch (Exception ex)
```

```
{
    return StatusCode(500, $"Internal server error: {ex.Message}");
}
```

Common Error Types

- Missing database columns or stored procedures.
- Invalid API endpoint paths.
- React fetch API errors due to wrong URL or missing CORS policy.

5. Database Integration

The backend interacts with SQL Server through stored procedures.
Every operation uses a corresponding SQL procedure for optimized data handling.

Example Stored Procedure

```
CREATE PROCEDURE sp_AddStudent
    @Name NVARCHAR(100),
    @Age INT,
    @Grade NVARCHAR(5),
    @CourseId INT
AS
BEGIN
    INSERT INTO Students (Name, Age, Grade, CourseId)
    VALUES (@Name, @Age, @Grade, @CourseId);
END;
```

Testing SQL Procedures

All stored procedures were executed manually in SQL Server Management Studio (SSMS) to verify functionality.

6. Integration Testing

Both applications (frontend and backend) were tested together to verify full connectivity.

Steps:

1. Run backend API using dotnet run.

2. Run frontend using npm start.
3. Perform operations through the UI:
    - Add new student.
    - Edit existing student details.
    - Delete student record.
    - Refresh to verify data update from the database.

7. Common Debugging Techniques

Issue	**Possible Cause**	**Solution**
Data not inserting	CORS or API URL mismatch	Verify backend API base URL
500 Internal Server Error	Missing database column or stored procedure	Check SQL procedure and model
React not updating list	State not refreshed	Call fetch function after POST/DELETE

8. Deployment Readiness

Before submission or deployment, ensure:

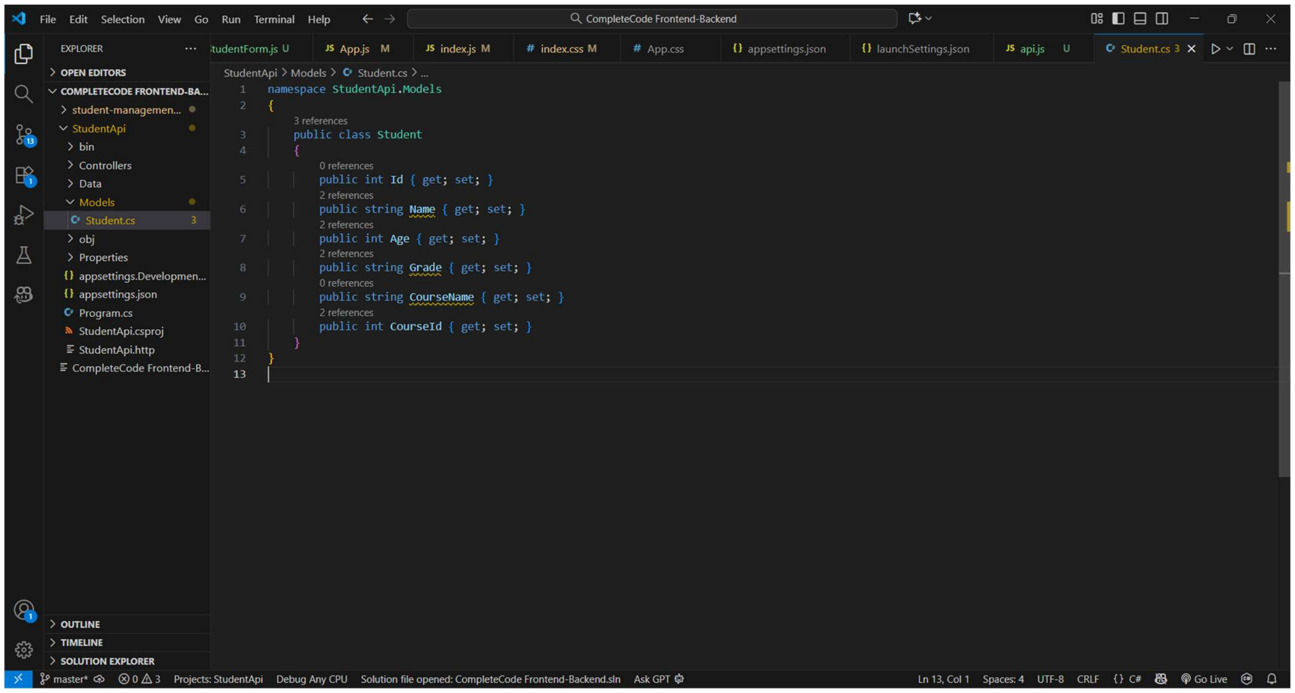
- All CRUD features work without console errors.
- API and React projects are properly organized into separate folders.
- Database backup file (.bak or SQL script) is included.
- Screenshots are captured for demonstration.

Conclusion

The Day 7 activity completes the frontend-backend integration using ReactJS, ASP.NET Core, and SQL Server.
The system is now a functional full-stack project demonstrating all the essential components of modern .NET web development:

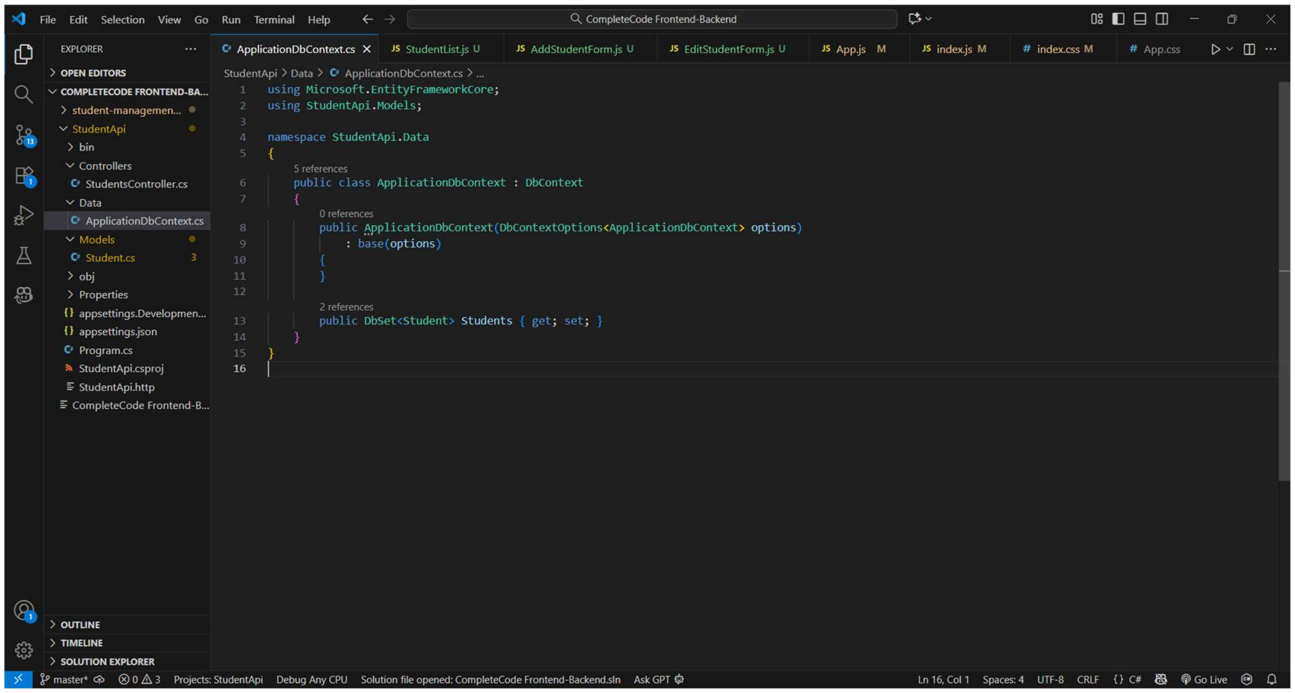
- Backend logic in ASP.NET Core
- Database management in SQL Server
- Responsive UI in ReactJS
- End-to-end data connectivity and CRUD operations

Snapshots:



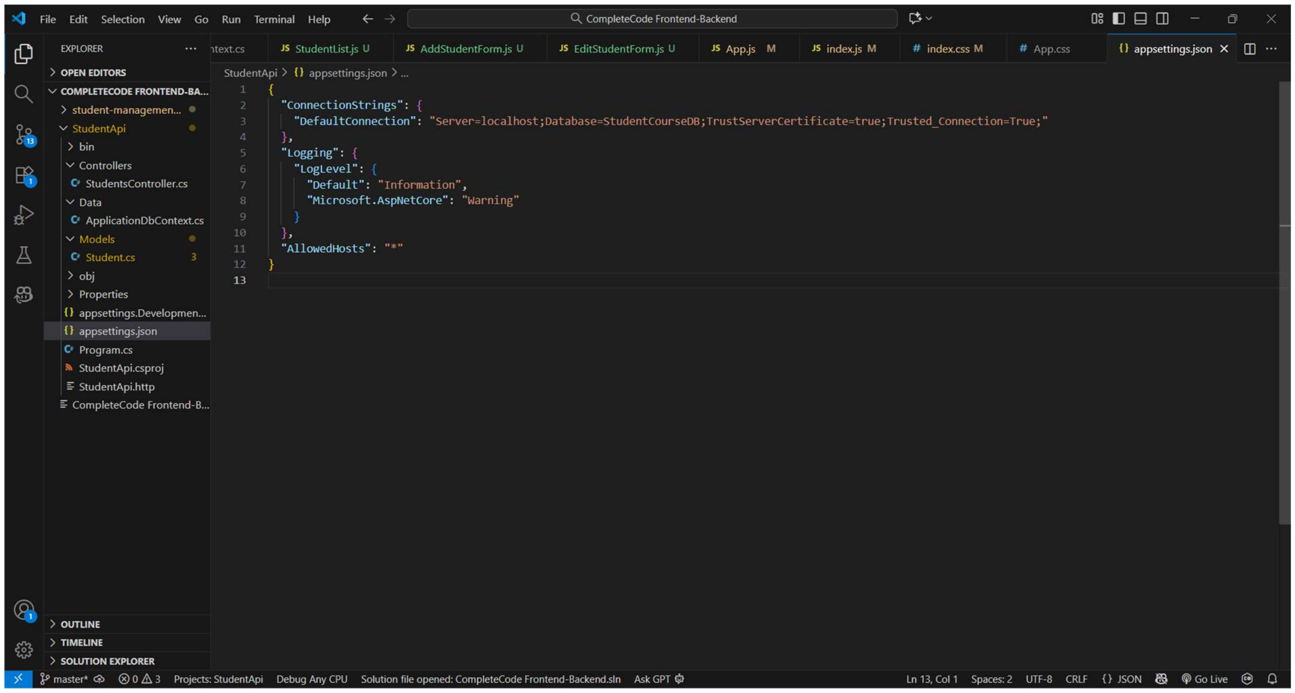
```
namespace StudentApi.Models
{
    public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
        public string Grade { get; set; }
        public string CourseName { get; set; }
        public int CourseId { get; set; }
    }
}
```

Student.cs class with all properties.



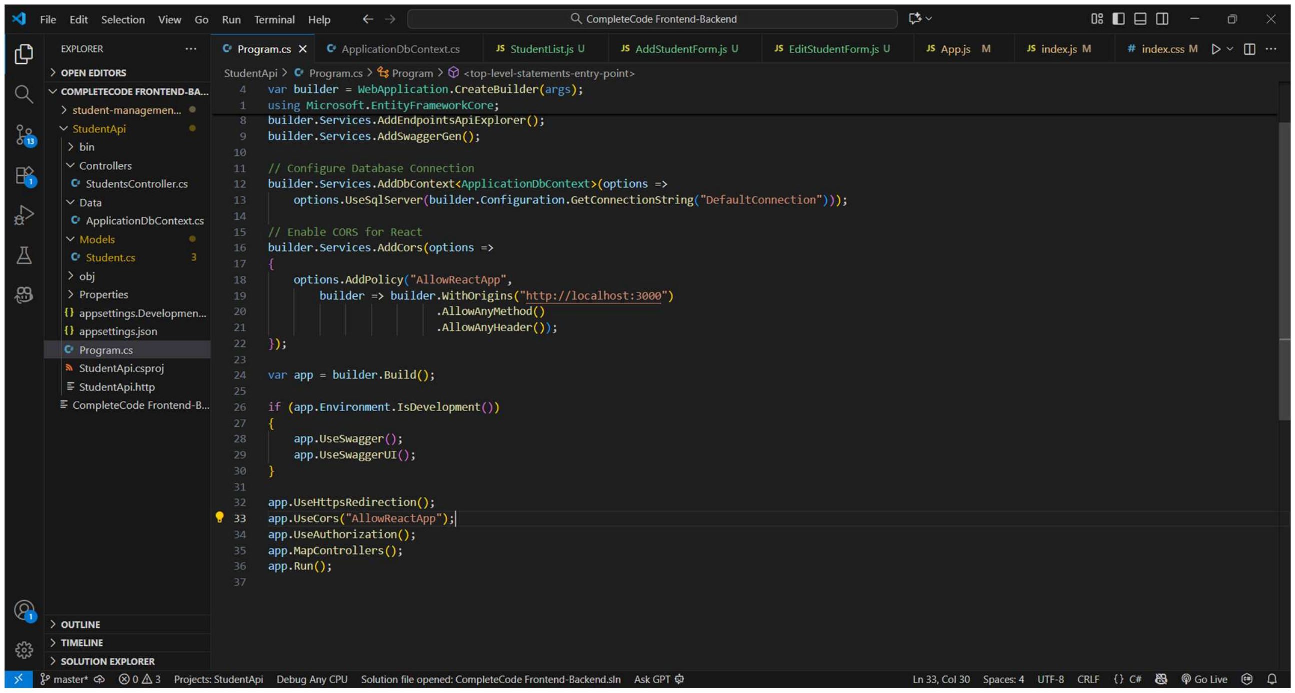
```
using Microsoft.EntityFrameworkCore;
using StudentApi.Models;

namespace StudentApi.Data
{
    public class ApplicationDbContext : DbContext
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }

        public DbSet<Student> Students { get; set; }
    }
}
```

Code in ApplicationDbContext.cs linking to Students table.



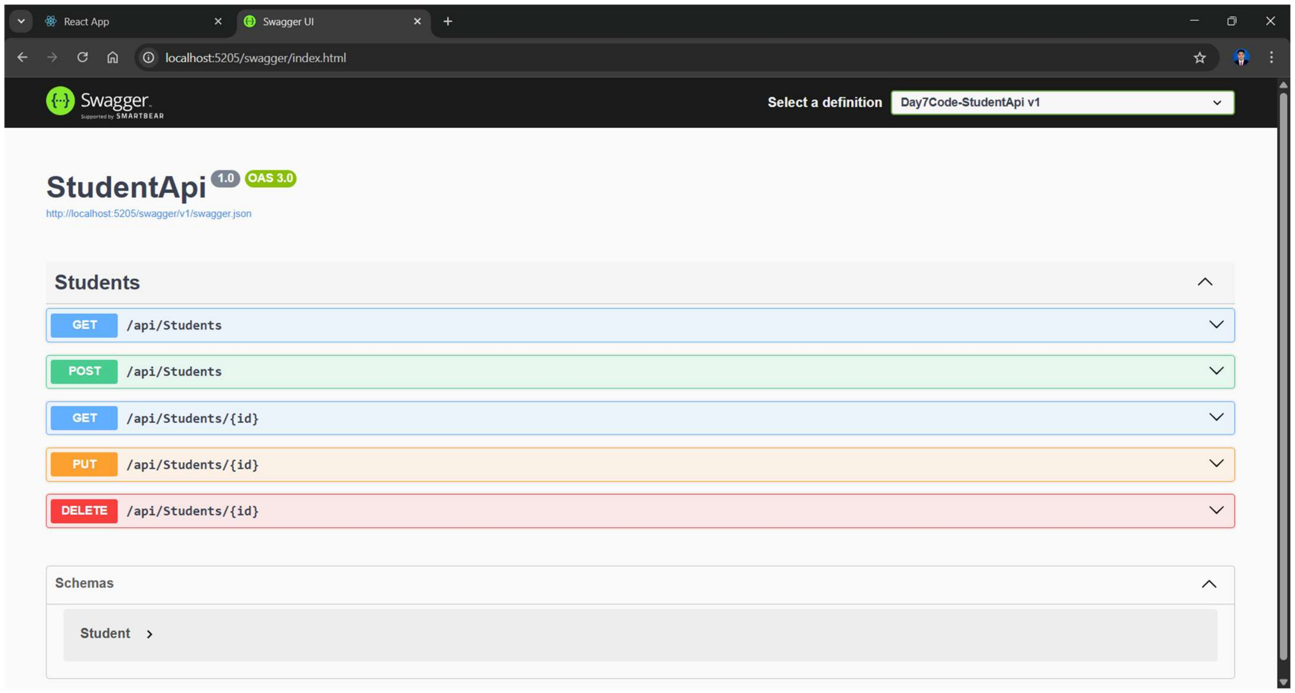
```json
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost;Database=StudentCourseDB;TrustServerCertificate=true;Trusted_Connection=True;"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}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

SQL Server connection string added to appsettings.json.



```csharp
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.



StudentApi 1.0 OAS 3.0

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

Students

GET /api/Students

POST /api/Students

GET /api/Students/{id}

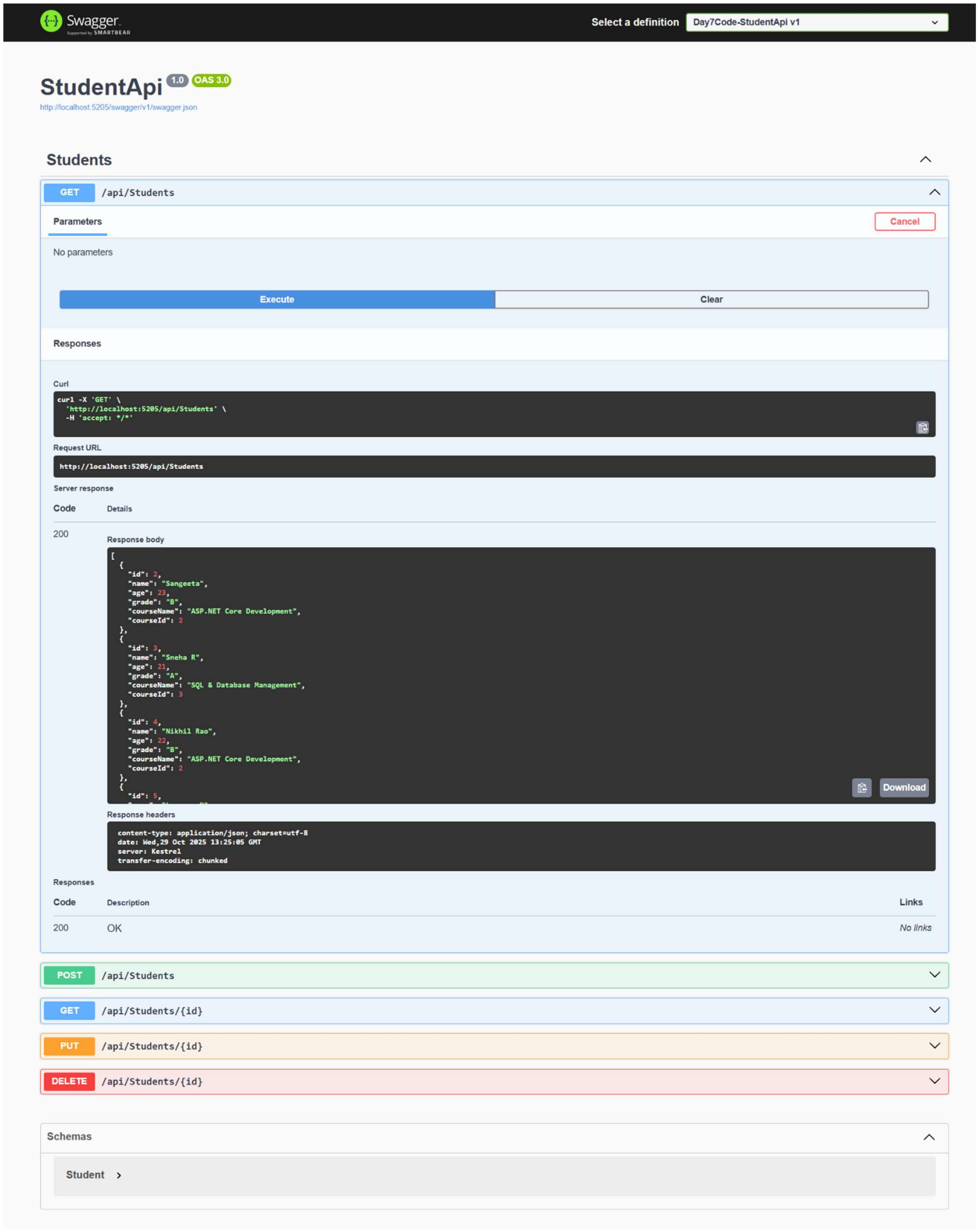
PUT /api/Students/{id}

DELETE /api/Students/{id}

Schemas

Student

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": 21,
    "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}

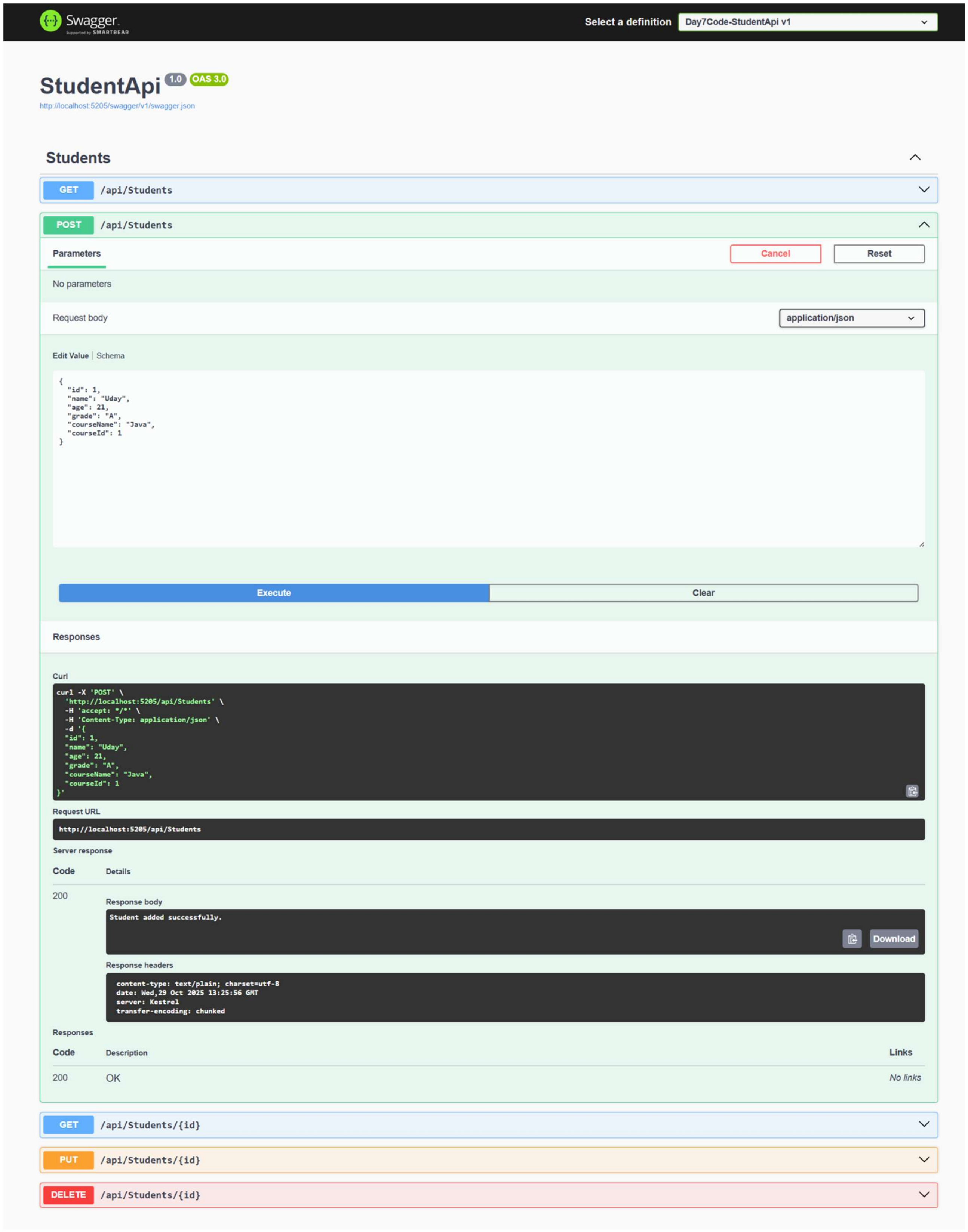
PUT /api/Students/{id}

DELETE /api/Students/{id}

Schemas

Student >

Successful response listing students from database.



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

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.`

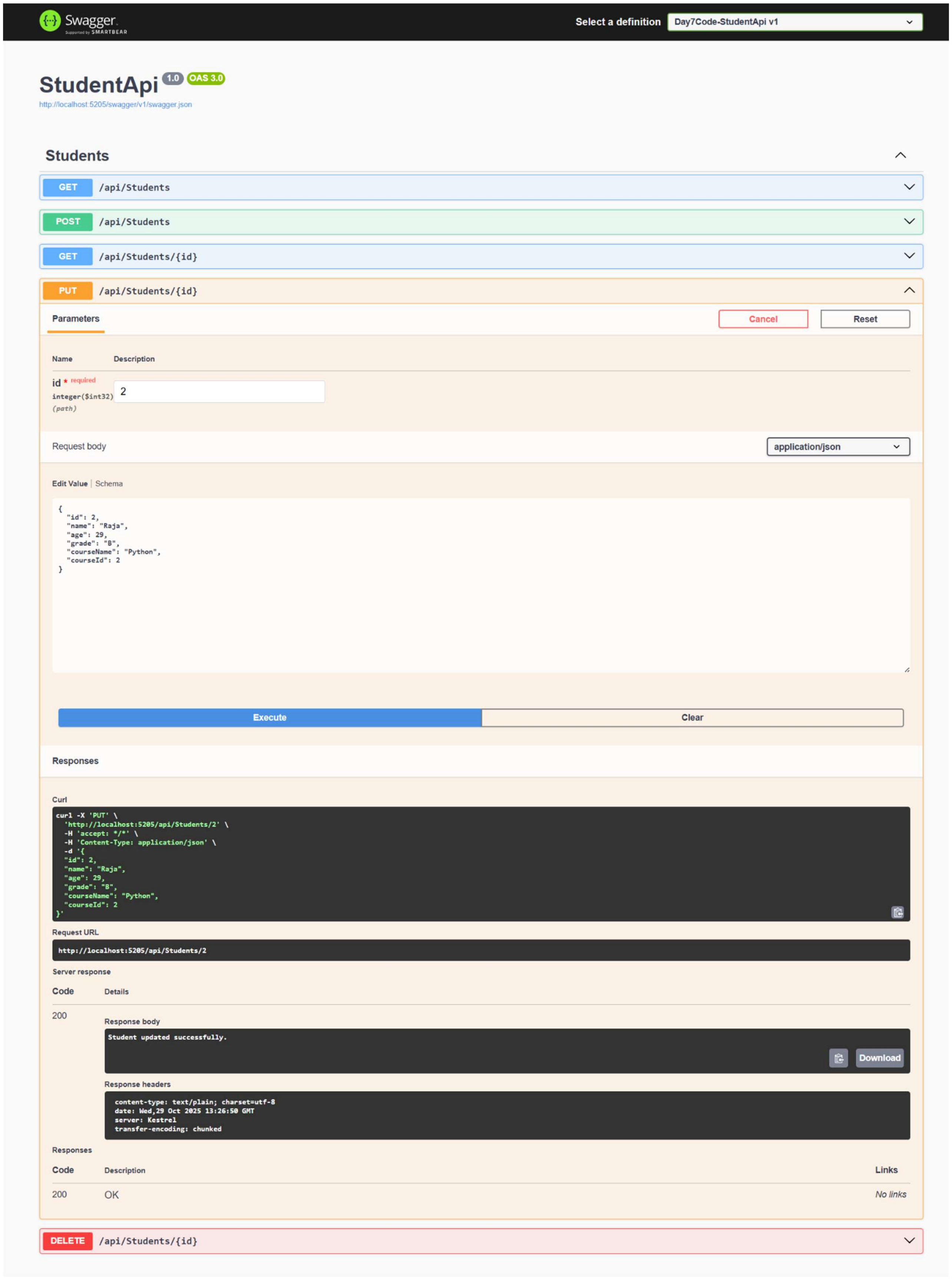
Response headers

```
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*

New student record added through POST endpoint.



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}

Name	Description
id * required integer($int32) (path)	2

Request body: application/json

Edit Value | Schema

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

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: Student updated successfully.

Response body

```
Student updated successfully.
```

Response headers

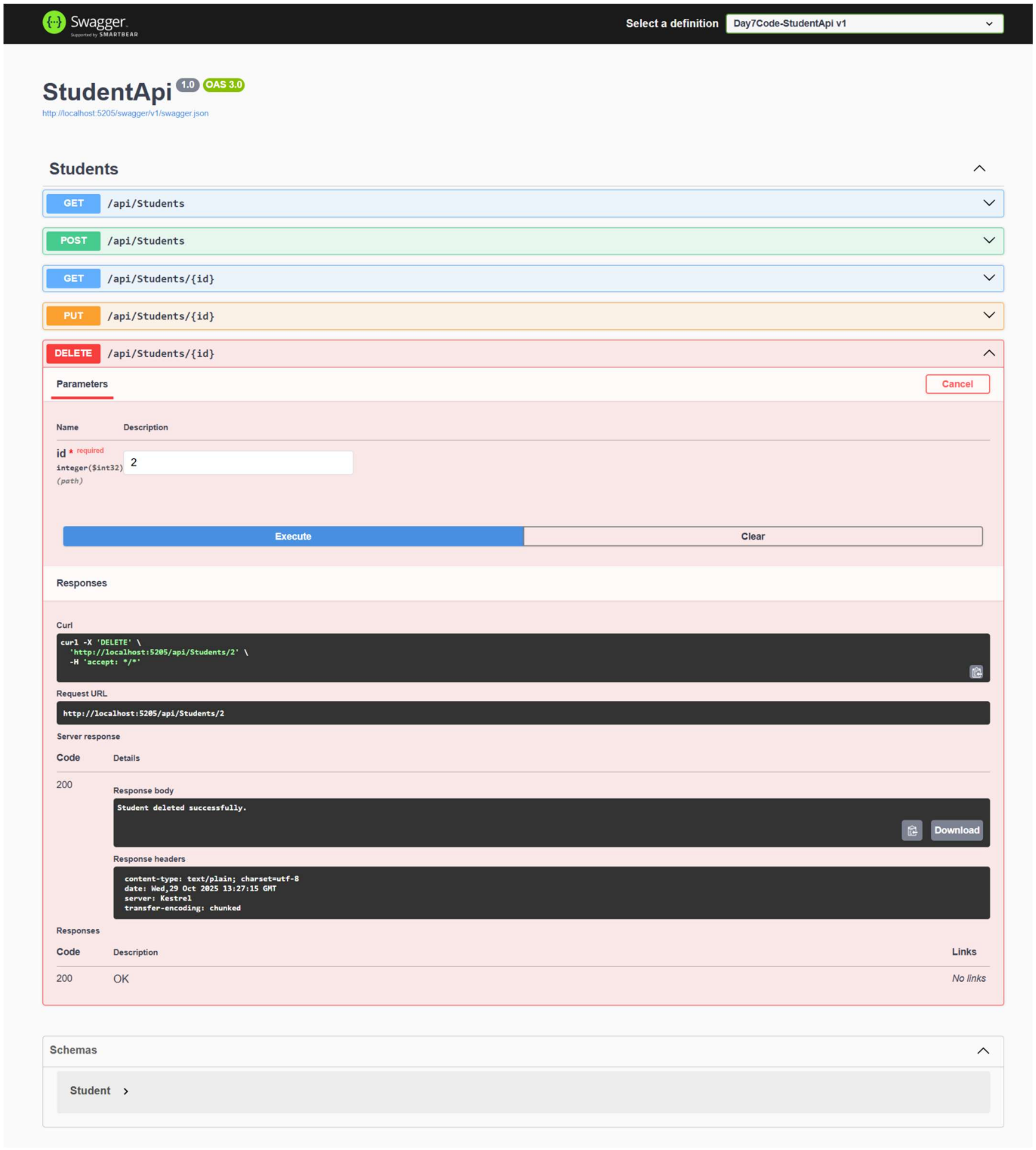
```
 content-type: text/plain; charset=utf-8
 date: Wed,29 Oct 2025 13:26:50 GMT
 server: Kestrel
 transfer-encoding: chunked
```

Responses

Code	Description	Links
200	OK	*No links*

DELETE /api/Students/{id}

Student record updated successfully.



Name	Description
id * required integer($int32) (path)	2

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

```
Student deleted successfully.
```

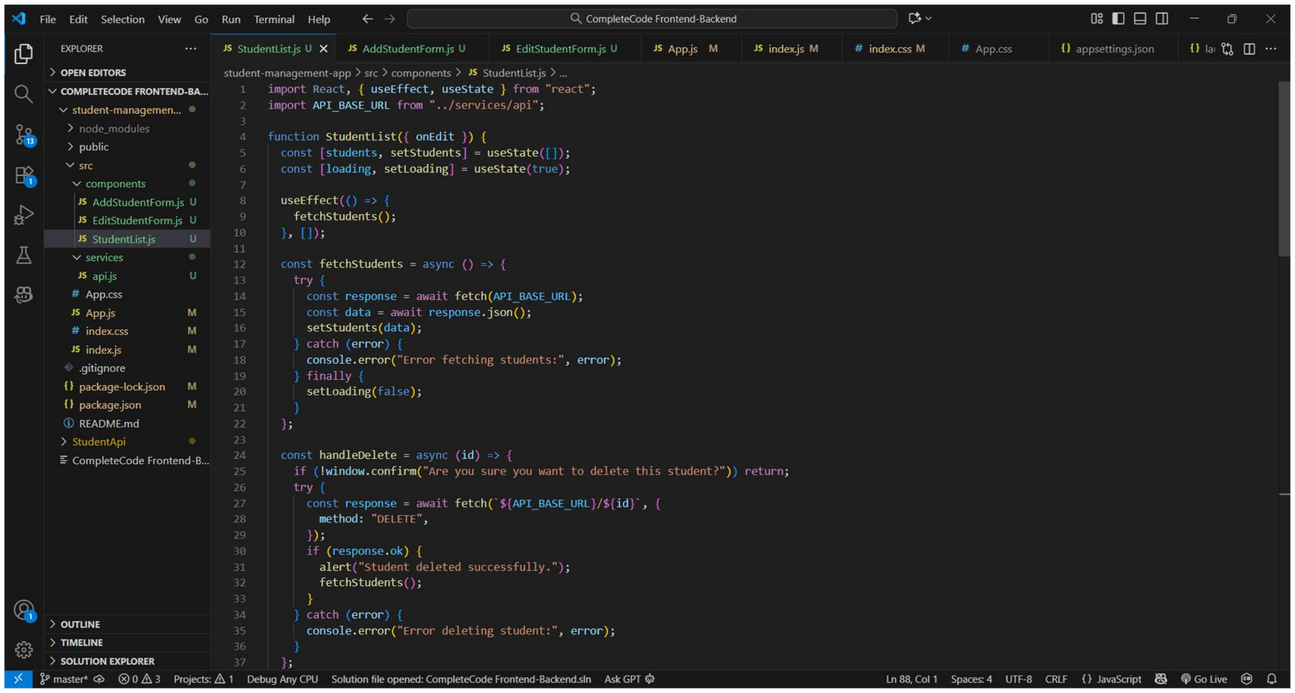
Response headers

```
 content-type: text/plain; charset=utf-8
 date: Wed,29 Oct 2025 13:27:15 GMT
 server: Kestrel
 transfer-encoding: chunked
```

Responses

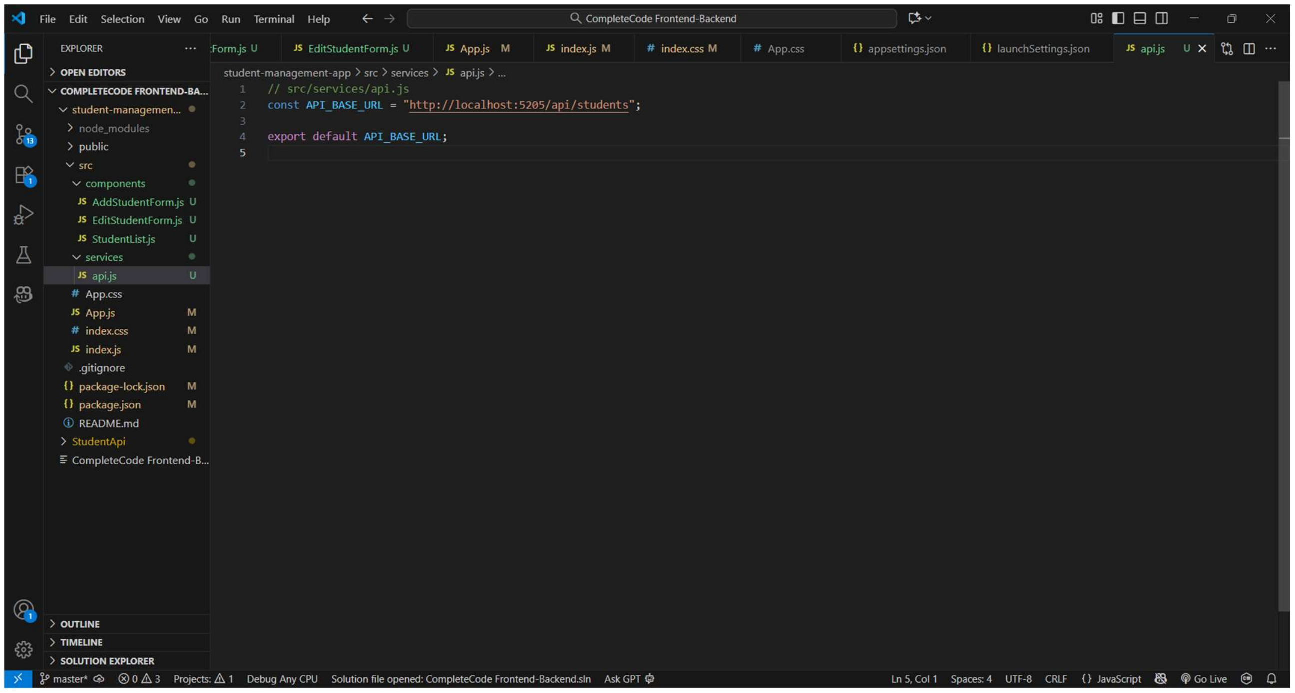
Code	Description	Links
200	OK	No links

Student entry deleted via DELETE endpoint.



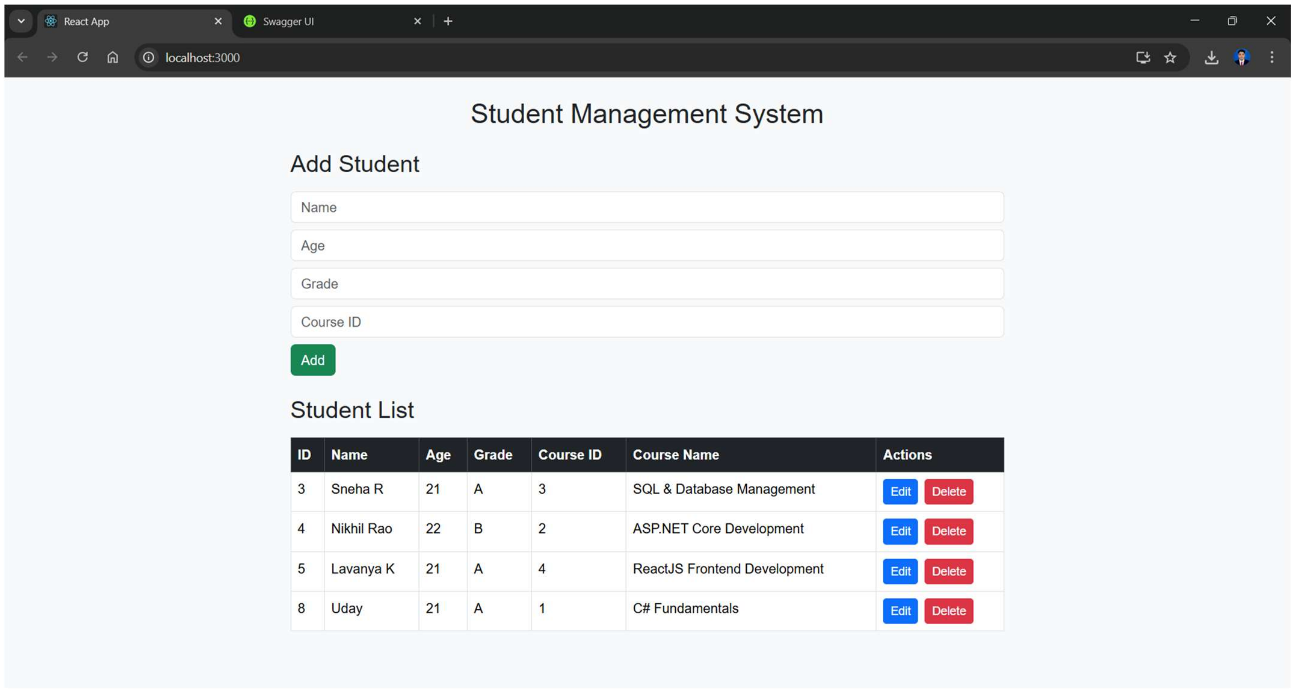
```
import React, { useEffect, useState } from "react";
import API_BASE_URL from "../services/api";

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.



Student Management System

Add Student

Name

Age

Grade

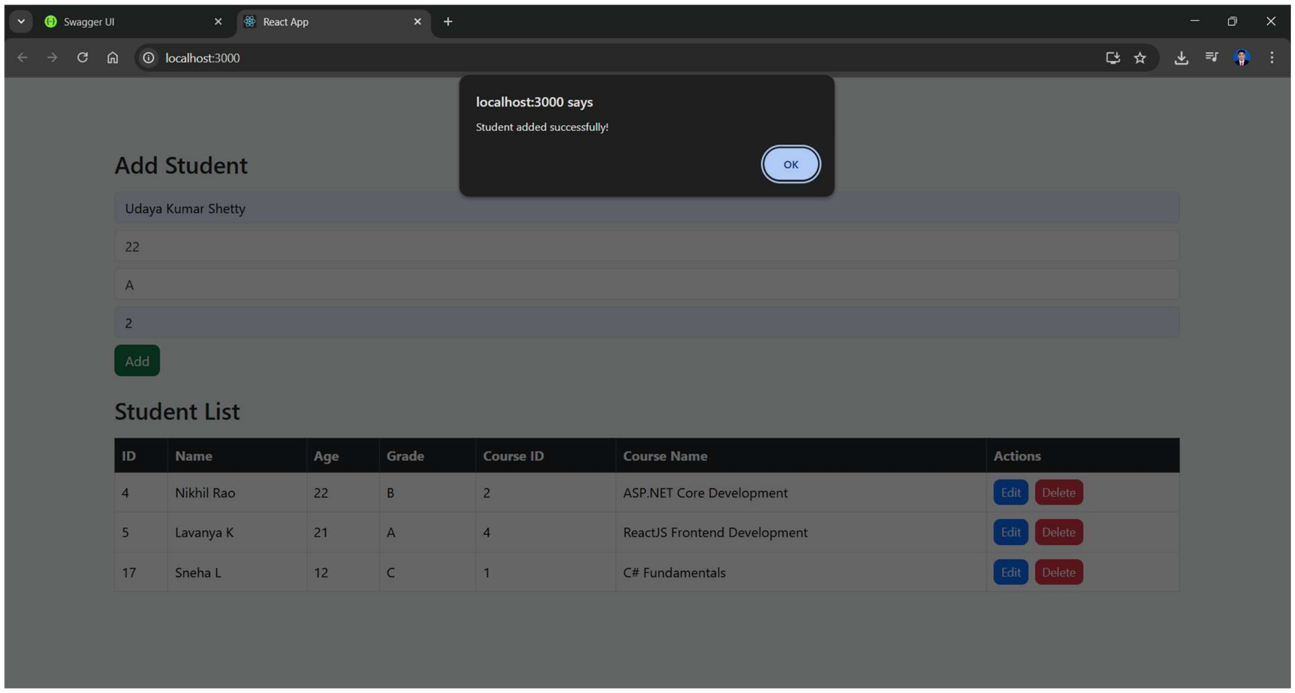
Course ID

Add

Student List

ID	Name	Age	Grade	Course ID	Course Name	Actions
3	Sneha R	21	A	3	SQL & Database Management	Edit Delete
4	Nikhil Rao	22	B	2	ASP.NET Core Development	Edit Delete
5	Lavanya K	21	A	4	ReactJS Frontend Development	Edit Delete
8	Uday	21	A	1	C# Fundamentals	Edit Delete

Displays list of students in React UI.



localhost:3000 says

Student added successfully!

OK

Add Student

Udaya Kumar Shetty

22

A

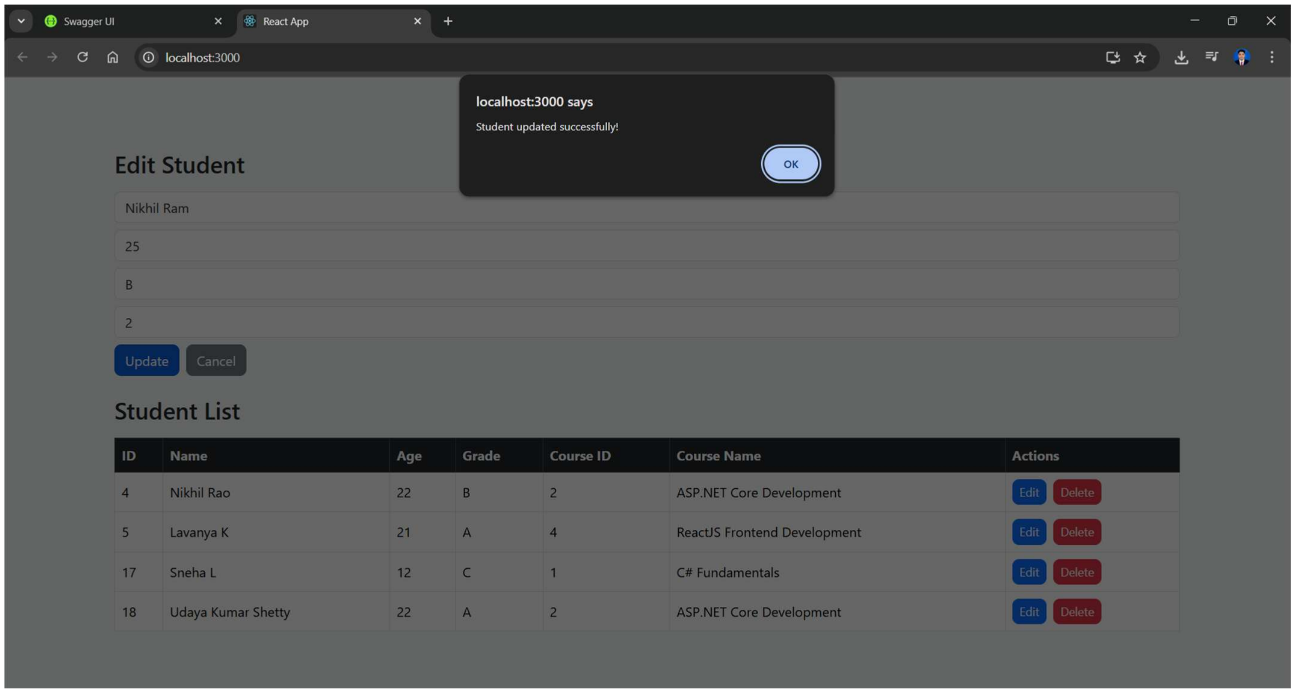
2

Add

Student List

ID	Name	Age	Grade	Course ID	Course Name	Actions
4	Nikhil Rao	22	B	2	ASP.NET Core Development	Edit Delete
5	Lavanya K	21	A	4	ReactJS Frontend Development	Edit Delete
17	Sneha L	12	C	1	C# Fundamentals	Edit Delete

Form UI for adding new student data.



localhost:3000 says

Student updated successfully!

OK

Edit Student

Nikhil Ram

25

B

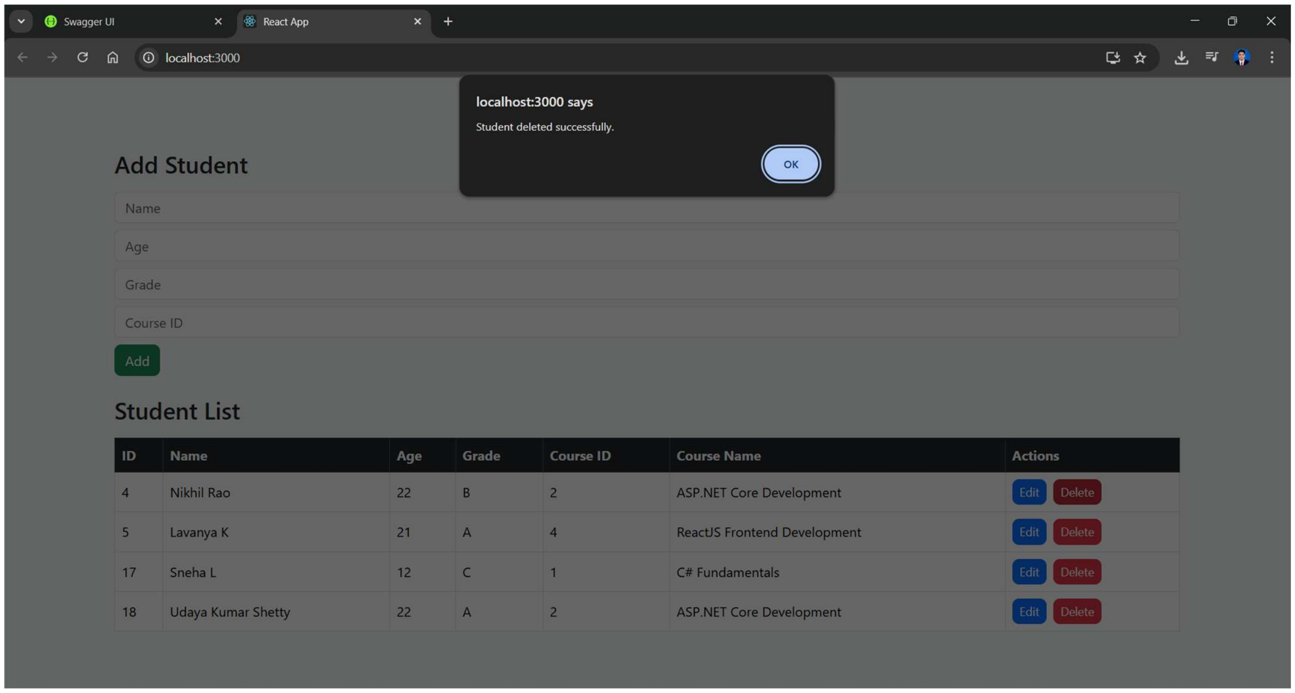
2

Update Cancel

Student List

ID	Name	Age	Grade	Course ID	Course Name	Actions
4	Nikhil Rao	22	B	2	ASP.NET Core Development	Edit Delete
5	Lavanya K	21	A	4	ReactJS Frontend Development	Edit Delete
17	Sneha L	12	C	1	C# Fundamentals	Edit Delete
18	Udaya Kumar Shetty	22	A	2	ASP.NET Core Development	Edit Delete

Edit page showing existing data for update.



localhost:3000 says

Student deleted successfully.

OK

Add Student

Name

Age

Grade

Course ID

Add

Student List

ID	Name	Age	Grade	Course ID	Course Name	Actions
4	Nikhil Rao	22	B	2	ASP.NET Core Development	Edit Delete
5	Lavanya K	21	A	4	ReactJS Frontend Development	Edit Delete
17	Sneha L	12	C	1	C# Fundamentals	Edit Delete
18	Udaya Kumar Shetty	22	A	2	ASP.NET Core Development	Edit Delete

Edit page showing existing data for delete.