Dhruv Compusoft Consultancy Pvt Ltd

Basaveshwar Nagar, Bengaluru– 560079

Project Report On

"Full-Stack Student Management System"

Submitted in partial fulfilment of the requirements for the .NET Full Stack Developer at Dhruv Compusoft Consultancy Pvt Ltd.

Submitted by

Name: Udaya Kumar Shetty

Batch: 2025 (CSE)

Project Duration: Week 4 (.NET Training)

Submitted to

Technical Team – Dhruv Compusoft Consultancy Pvt Ltd.

1. INTRODUCTION

In today's data-driven world, educational institutions require a systematic way to manage student information effectively. The **Student Management System (SMS)** project is a **full-stack web application** designed to simplify managing student data such as name, age, grade, and course.

This project consolidates all the concepts learned during the .NET training — including backend API development, SQL Server integration, Entity Framework Core, and ReactJS-based frontend design.The main aim of this project is to create a **responsive, interactive, and robust CRUD system** that can handle student records efficiently while maintaining clean architecture and modular code.

The system follows a **three-tier architecture**:

1. **Frontend (ReactJS + Bootstrap)** – Handles user interface and interactions.
2. **Backend (ASP.NET Core Web API)** – Processes business logic and exposes endpoints.
3. **Database (SQL Server)** – Stores and retrieves persistent data through stored procedures.

This project marks the culmination of the training journey, combining **C#, ASP.NET Core, SQL, and ReactJS** into a single integrated product.

---

2. OBJECTIVES

The key objectives of this project are:

1. To develop a **full-stack CRUD application** using ASP.NET Core Web API, SQL Server, and ReactJS.
2. To design and use **stored procedures** for secure and optimized database operations.
3. To implement **Entity Framework Core (Database First Approach)** for database communication.
4. To integrate **ReactJS frontend** with backend API using **Fetch API**.
5. To ensure **input validation**, **error handling**, and **data integrity** throughout the system.
6. To follow clean code practices using modular, reusable, and maintainable components.
7. To apply **Bootstrap and Flexbox** for a responsive user interface.
8. To host and test the API using Swagger and verify data operations.

3. SYSTEM REQUIREMENTS

3.1 Hardware Requirements

- Processor: Intel i5 or higher
- RAM: Minimum 8 GB
- Storage: 20 GB free disk space
- Operating System: Windows 10/11 (64-bit)

3.2 Software Requirements

- Visual Studio 2022 (for API development)
- SQL Server Management Studio (SSMS)
- Node.js and npm (for ReactJS frontend)
- Visual Studio Code (for frontend development)
- Postman / Swagger (for API testing)
- GitHub (for version control)

---

4. TECHNOLOGY STACK

Layer	Technology
Frontend	ReactJS, JavaScript (ES6), Bootstrap 5
Backend	ASP.NET Core Web API (.NET 8/9)
ORM	Entity Framework Core
Database	Microsoft SQL Server
IDE	Visual Studio, VS Code
Testing	Swagger UI, Postman, Browser DevTools

The combination of these technologies allows for a powerful, modern, and scalable web application development environment.

5. SYSTEM DESIGN

The Student Management System uses **three-tier architecture**, ensuring separation of concerns:

5.1 Presentation Layer (Frontend)

- Built using ReactJS.
- Contains reusable components like StudentList, AddStudentForm, EditStudentForm.
- Communicates with backend through HTTP requests using Fetch API.
- Styled using Bootstrap for responsiveness.

5.2 Application Layer (Backend)

- Developed using ASP.NET Core Web API.
- Contains controllers that define endpoints for CRUD operations.
- Uses dependency injection for database context handling.
- Handles business logic and exception management.

5.3 Data Layer (Database)

- SQL Server stores data persistently.
- Data accessed using **Entity Framework Core**.
- Stored procedures used for CRUD to ensure efficiency and security.

---

6. DATABASE DESIGN

6.1 Tables

1. **Courses Table**
    - CourseId (Primary Key)
    - CourseName
2. **Students Table**
    - Id (Primary Key)
    - Name
    - Age
    - Grade

6.2 Relationships

- One-to-Many relationship between **Courses** and **Students**.
- Each course can have multiple students enrolled.

6.3 Stored Procedures

- sp_GetStudents – Retrieves all students with their course details.
- sp_GetStudentById – Fetches a student by ID.
- sp_AddStudent – Inserts a new record.
- sp_UpdateStudent – Updates existing record.
- sp_DeleteStudent – Deletes a record safely.

---

7. BACKEND DEVELOPMENT (ASP.NET CORE WEB API)

The backend of this system is built using **ASP.NET Core Web API**.

7.1 Key Features

- Follows RESTful API design.
- Uses controllers for each entity (StudentsController).
- Implements FromSqlRaw() for executing stored procedures.
- Handles exceptions gracefully with try-catch blocks.
- Returns meaningful HTTP response codes.

7.2 Important API Endpoints

Endpoint	HTTP Method	Description
/api/students	GET	Fetch all students
/api/students/{id}	GET	Fetch student by ID
/api/students	POST	Add new student
/api/students/{id}	PUT	Update existing student
/api/students/{id}	DELETE	Delete student

7.3 Example API Call

POST /api/students

```
{
  "name": "Rahul Kumar",
  "age": 22,
  "grade": "A",
  "courseId": 1
}
```

---

8. FRONTEND DEVELOPMENT (REACTJS)

The frontend is developed using **ReactJS**, enabling fast and dynamic rendering of the user interface.

8.1 Core Components

1. **App.js** – Root component with routing setup.
2. **StudentList.js** – Displays all student records fetched from API.
3. **AddStudentForm.js** – Used to create new student entries.
4. **EditStudentForm.js** – Allows editing of student details.
5. **Navbar.js** – Simple navigation menu using React Router.

8.2 Data Flow

- The frontend communicates with the API using fetch() calls.
- JSON responses are parsed and displayed dynamically.
- State is managed using React hooks like useState and useEffect.

8.3 Styling

- Implemented using Bootstrap 5.
- Used responsive grid layout for consistent alignment.
- Buttons, tables, and forms styled using standard Bootstrap components.

9. SYSTEM FUNCTIONALITY

9.1 CRUD Operations

Create (Add Student)
The user can add a new student record using the form in ReactJS. The form data is sent to the backend API using a POST request.

Read (View Students)
All students are displayed in a table format. The data is fetched from the backend using a GET request.

Update (Edit Student)
When the user clicks "Edit," the existing data is loaded into the form for modification and submitted using a PUT request.

Delete (Remove Student)
User can delete a record with a single click, which triggers a DELETE request.

9.2 Validation and Error Handling

- Form fields validated using React.
- Backend checks for null or invalid data.
- Proper HTTP status codes returned for success or failure.

---

10. TESTING AND DEPLOYMENT

10.1 Testing

- Backend tested using **Swagger UI**.
- Database connectivity verified using SSMS.
- Frontend tested on local browser (http://localhost:3000).
- Manual testing performed for edge cases such as blank inputs, invalid IDs, etc.

10.2 Deployment

- Backend hosted locally using Kestrel server.
- React app run using npm start.
- Could be deployed to Azure App Service and Azure SQL in future.

CHALLENGES FACED

1. Error while executing stored procedure with missing column names.
2. CORS policy issue between frontend and backend.
3. Data not updating due to API routing mismatch.
4. Handling validation errors in React form.
5. Managing multiple hooks and state updates efficiently.

Solutions:

- Adjusted stored procedure output to include required fields.
- Enabled CORS in ASP.NET Core.
- Implemented proper routes and tested in Swagger.
- Added form-level validation in frontend.

FUTURE ENHANCEMENTS

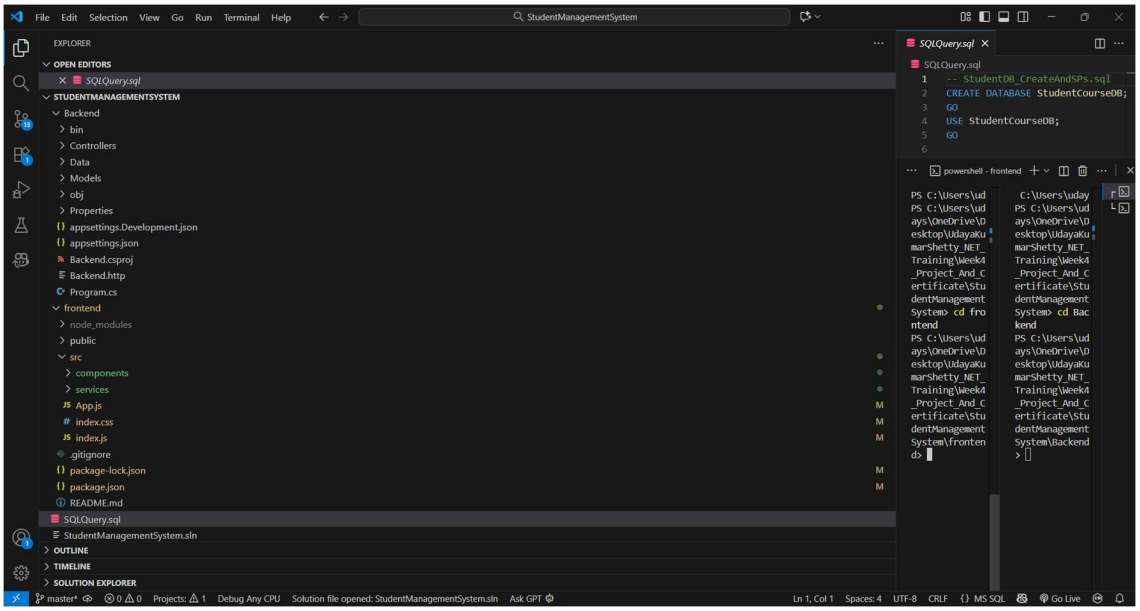
1. Implement user authentication (JWT-based login).
2. Add pagination and search functionality for large datasets.
3. Include course management and teacher modules.
4. Integrate email notification system.
5. Host the project on Azure cloud for real-time access.

CONCLUSION

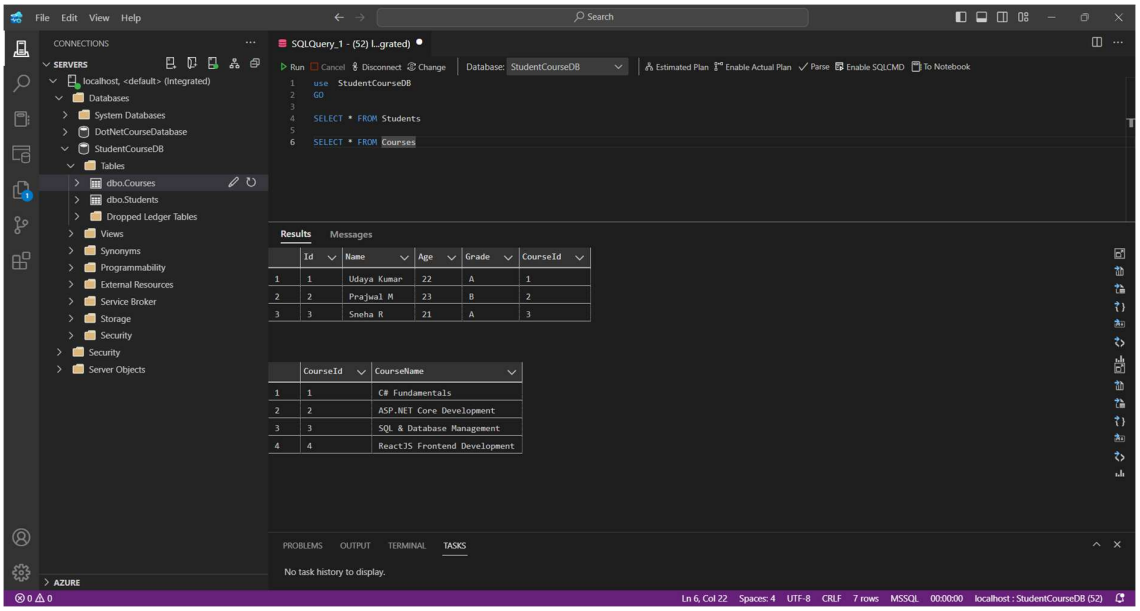
The Full Stack Student Management System successfully demonstrates the integration of frontend, backend, and database technologies learned throughout the .NET training program. It provides an efficient and responsive way to manage student information, leveraging the power of ASP.NET Core Web API and ReactJS.

This project represents the practical application of software development concepts, from database design to UI development, and prepares the developer for real-world full-stack development roles.

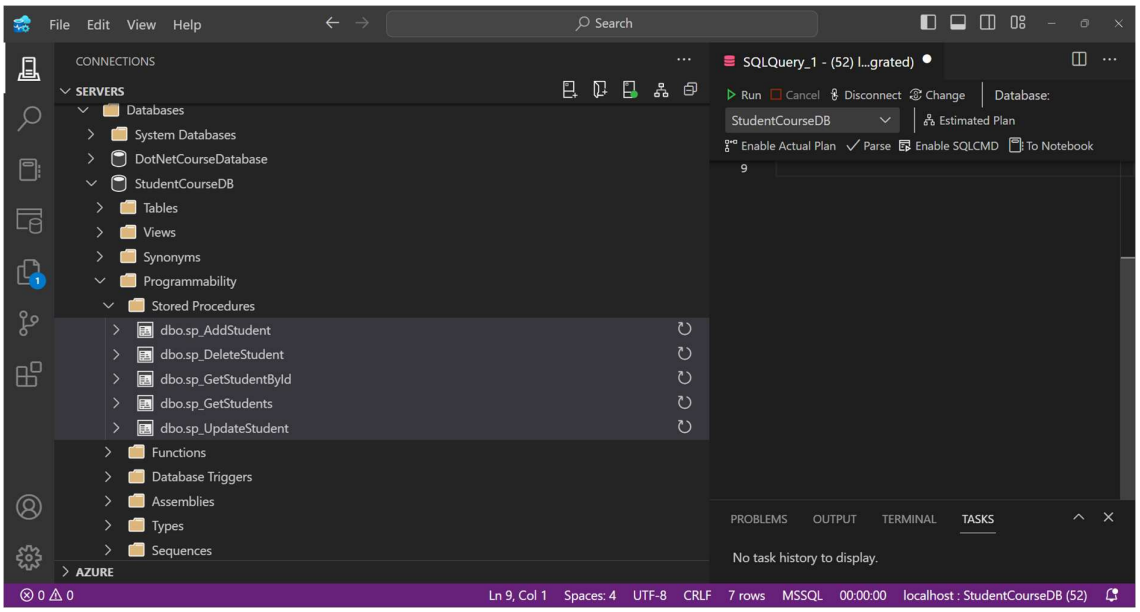
SNAPSHOTS:



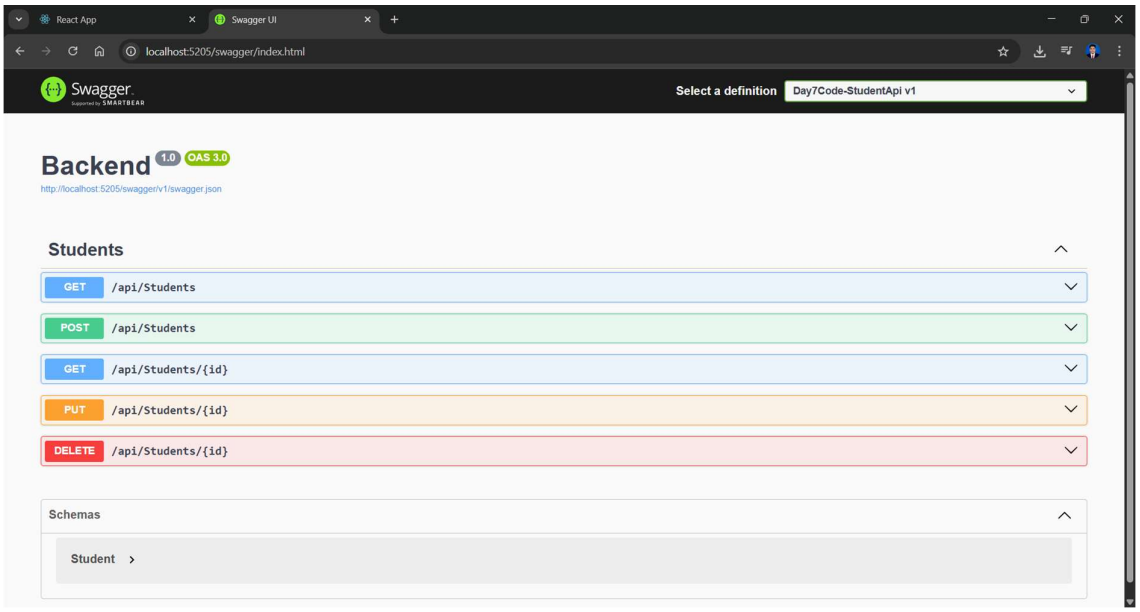
Complete structure of backend and frontend folders



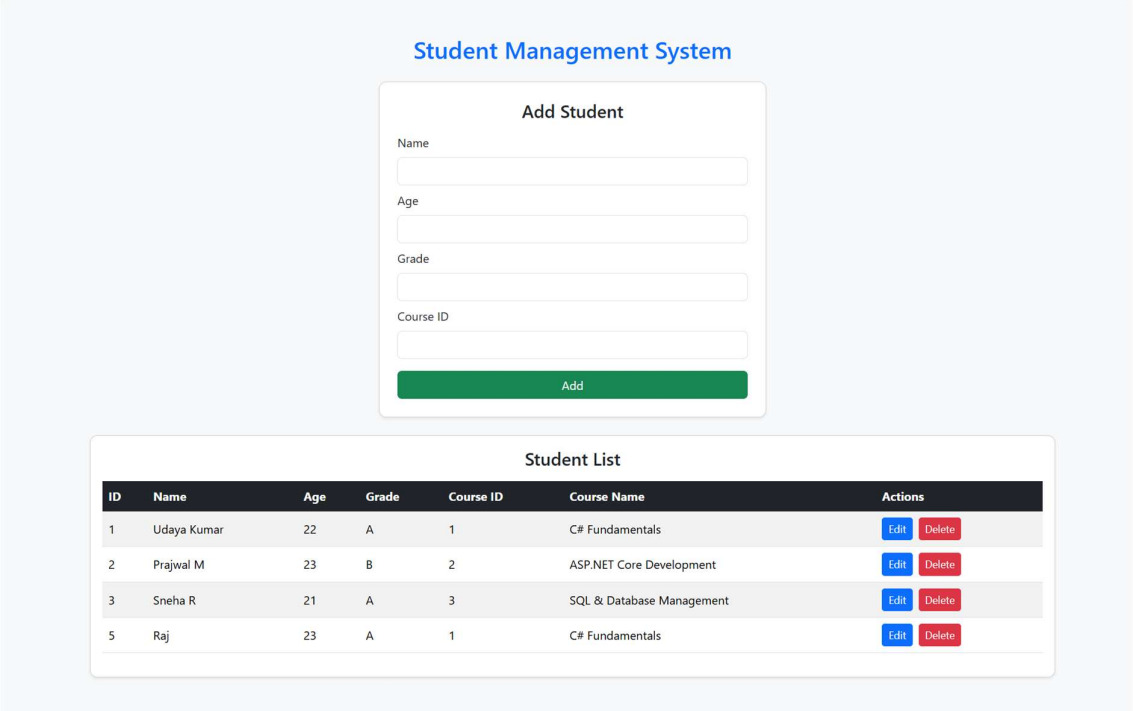
SQL Server tables (Students and Courses)



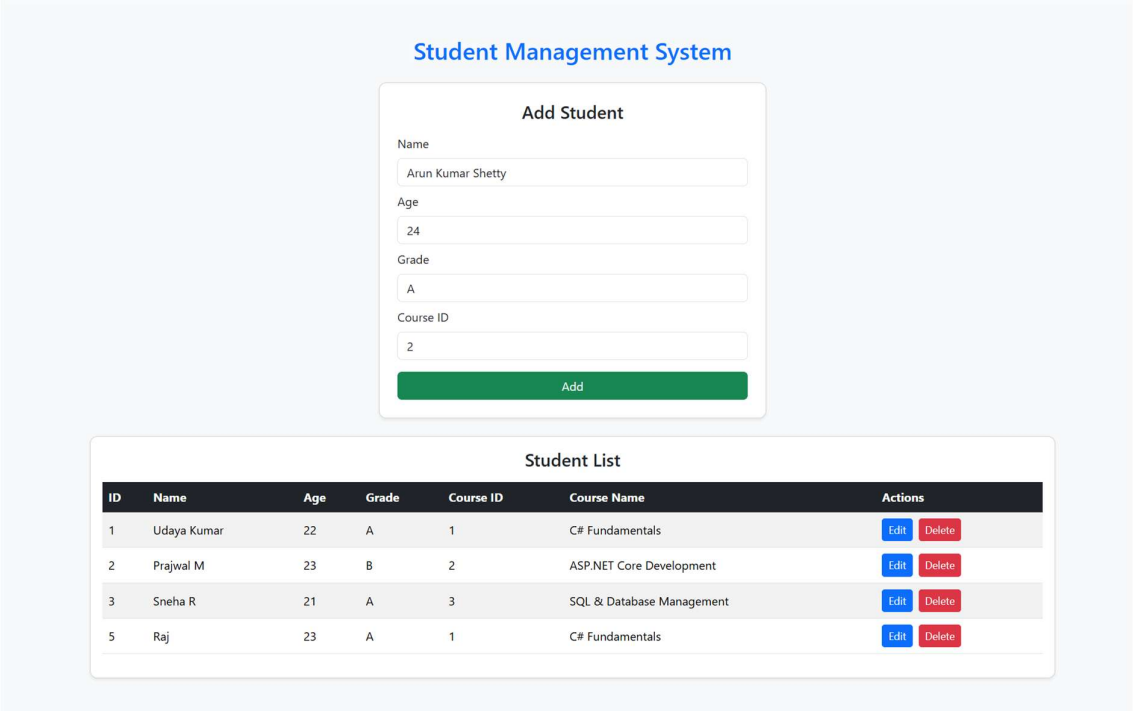
List of stored procedures created



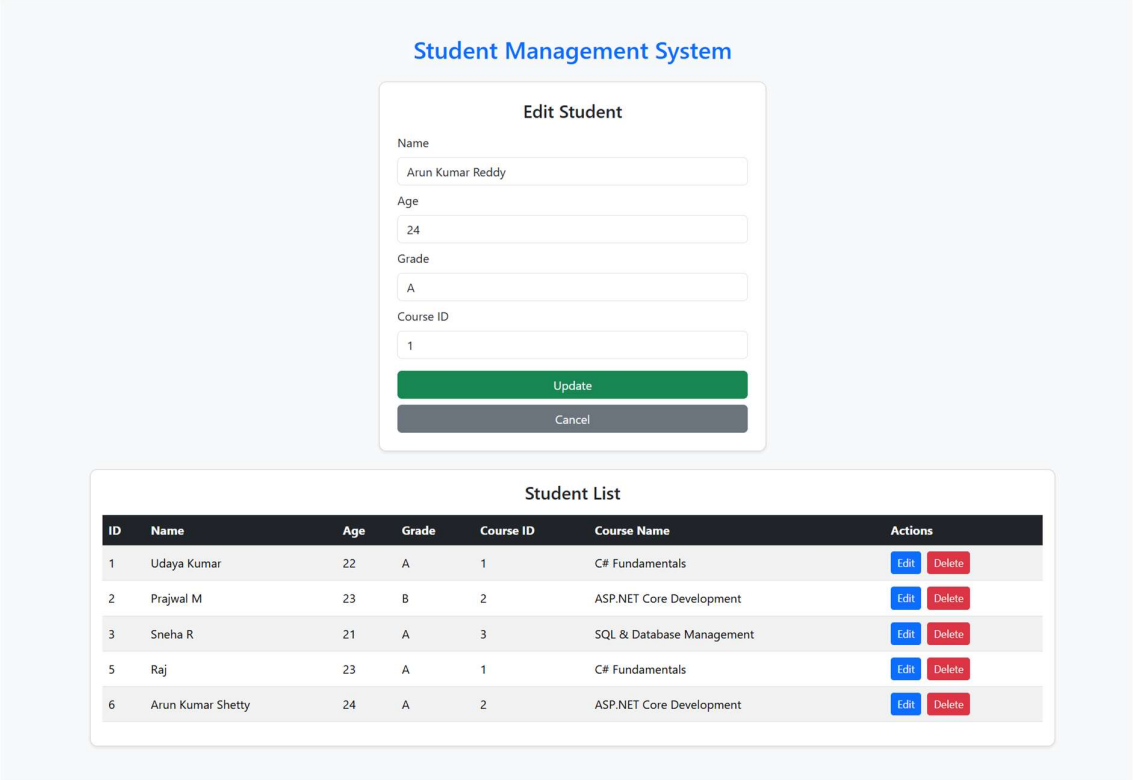
Backend endpoints tested



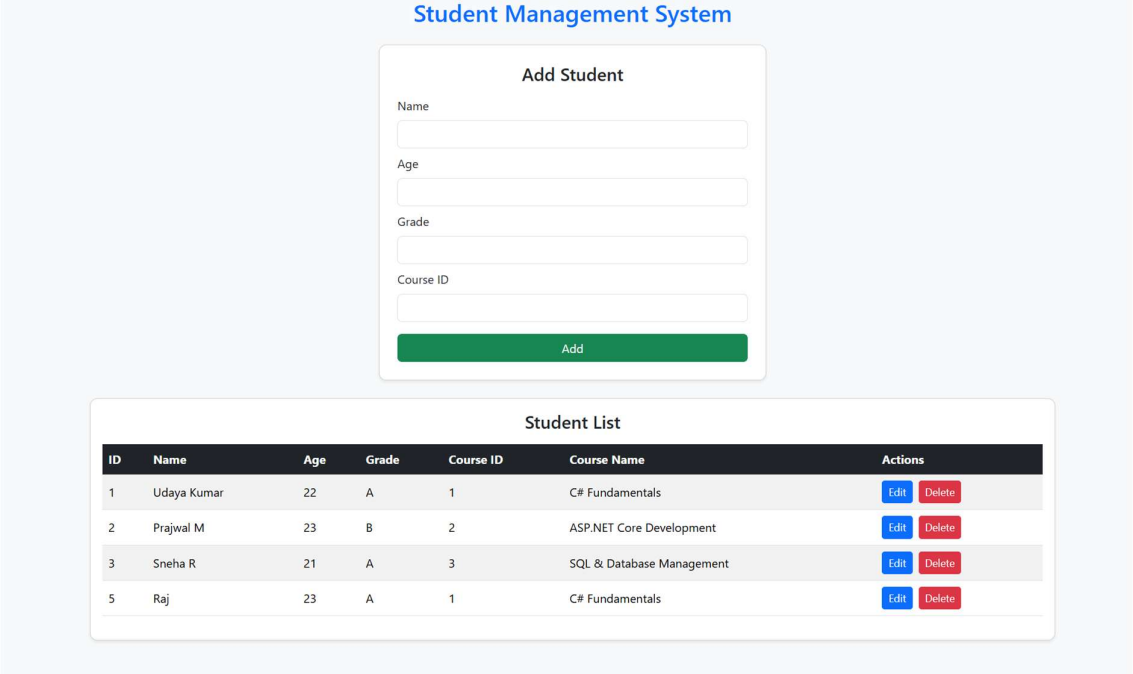
List of students displayed in React app



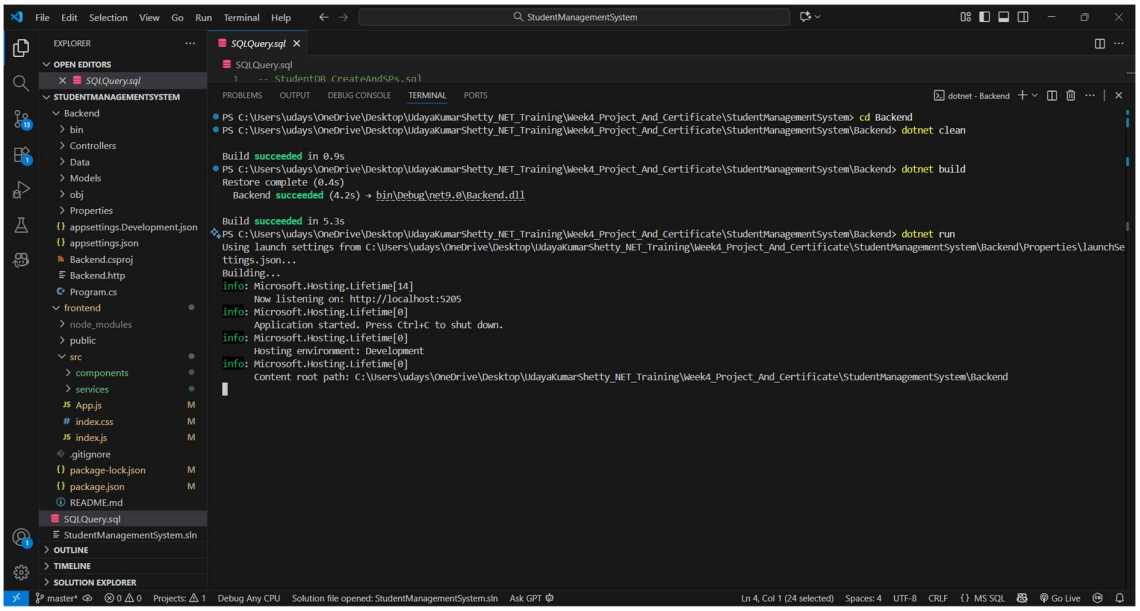
Add new student form



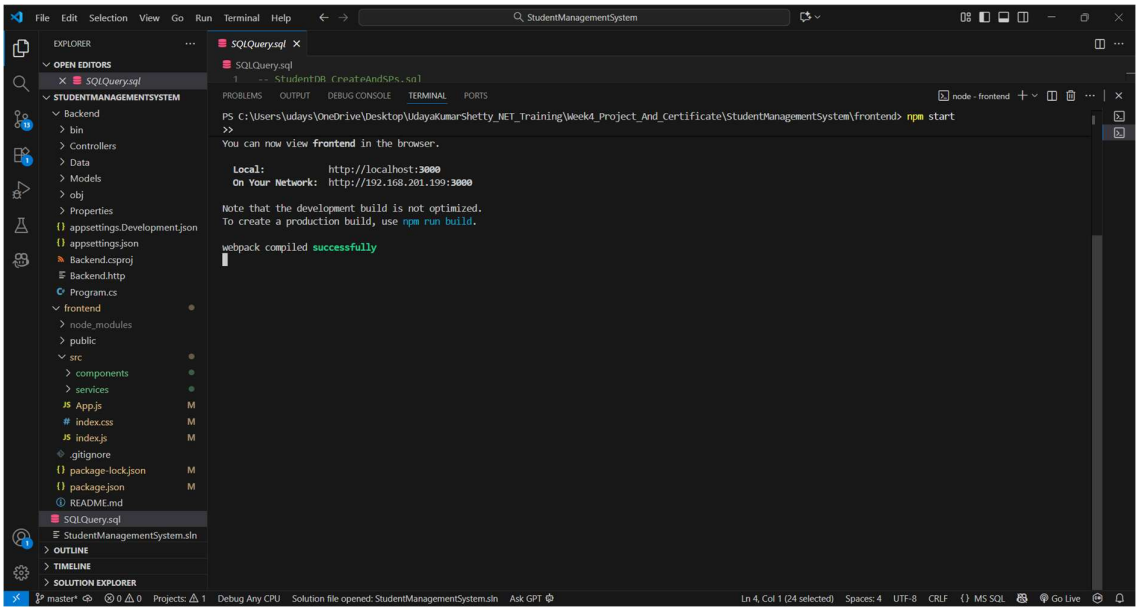
Edit student data



Record deleted successfully



API running in Visual Studio



React frontend running successfully