**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:

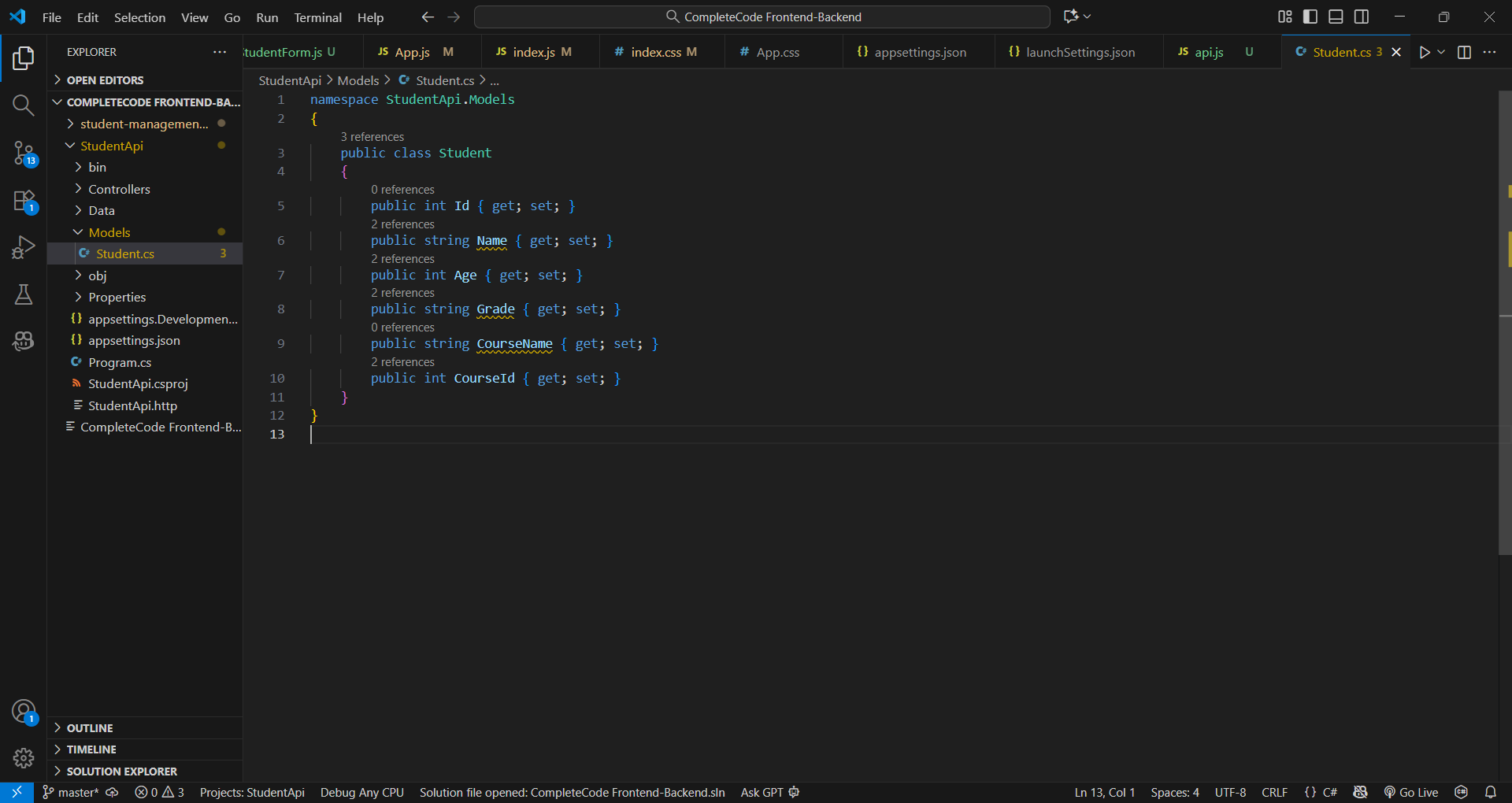
* All CRUD features work without console errors.
* API and React projects are properly organized into separate folders.
* Database backup file (.bak or SQL script) is included.
* Screenshots are captured for demonstration.

**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:**



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

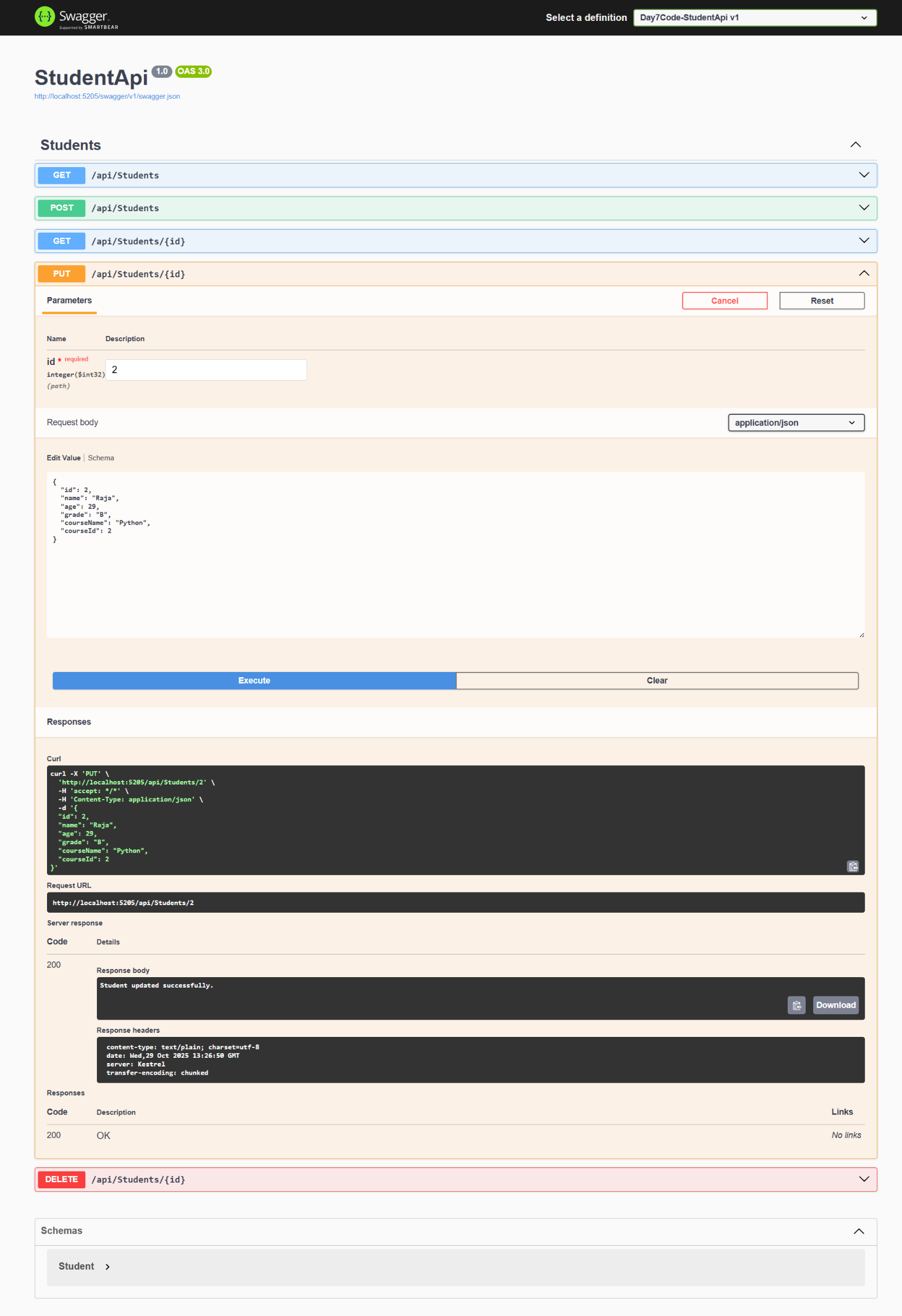
AI-generated content may be incorrect.

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.