Week 4 – Day 3 Report **Date:** 01-11-2025

Module: Full Stack Project – Student Management System

1. Objective of the Day

The goal of Day 3 was to **implement CRUD (Create, Read, Update, Delete) operations** in the backend using **ASP.NET Core Web API** and **SQL Server (Database First approach)**.
The objective was to connect the API with the database using stored procedures, handle validation, and ensure all operations are working via Swagger.

2. Topics Covered / Tasks Completed

a. Created Student Controller (CRUD operations)

- Implemented endpoints:
  - **GET** → Fetch all students
  - **GET/{id}** → Fetch specific student by ID
  - **POST** → Add a new student
  - **PUT/{id}** → Update an existing student
  - **DELETE/{id}** → Remove a student record

b. Used Stored Procedures for Each Operation

- Each API action was linked to a respective SQL stored procedure.
- Used FromSqlRaw and ExecuteSqlRaw for read/write operations.

3. Student Controller (StudentsController.cs)

```
using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using StudentApi.Models;

namespace StudentApi.Controllers

{

   [Route("api/[controller]")]

   [ApiController]

   public class StudentsController : ControllerBase

   {
```

```
private readonly ApplicationDbContext _context;


public StudentsController(ApplicationDbContext context)
{
   _context = context;
}
// GET: api/students
[HttpGet]
public IActionResult GetStudents()
{
   try
   {
      var students = _context.Students
         .FromSqlRaw("EXEC sp_GetStudents")
         .ToList();
      return Ok(students);
   }
   catch (Exception ex)
   {
      return StatusCode(500, $"Internal server error: {ex.Message}");
   }
}
// GET: api/students/{id}
[HttpGet("{id}")]
public IActionResult GetStudentById(int id)
{
   try
   {
      var student = _context.Students
```

```
            .FromSqlRaw("EXEC sp_GetStudentById @Id={0}", id)
            .AsEnumerable()
            .FirstOrDefault();
        if (student == null)
            return NotFound($"Student with ID {id} not found.");
        return Ok(student);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Internal server error: {ex.Message}");
    }
}
// POST: api/students
[HttpPost]
public IActionResult AddStudent([FromBody] Student student)
{
    if (!ModelState.IsValid)
        return BadRequest(ModelState);
    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}");
    }
```

```
        }
        // PUT: api/students/{id}
        [HttpPut("{id}")]
        public IActionResult UpdateStudent(int id, [FromBody] Student student)
        {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);
            try
            {
                _context.Database.ExecuteSqlRaw(
                    "EXEC sp_UpdateStudent @Id={0}, @Name={1}, @Age={2}, @Grade={3},
    @CourseId={4}",
                    id, student.Name, student.Age, student.Grade, student.CourseId);
                return Ok("Student updated successfully.");
            }
            catch (Exception ex)
            {
                return StatusCode(500, $"Internal server error: {ex.Message}");
            }
        }
        // DELETE: api/students/{id}
        [HttpDelete("{id}")]
        public IActionResult DeleteStudent(int id)
        {
            try
            {
                _context.Database.ExecuteSqlRaw("EXEC sp_DeleteStudent @Id={0}", id);
                return Ok("Student deleted successfully.");
            }
            catch (Exception ex)
            {
                return StatusCode(500, $"Internal server error: {ex.Message}");
            }
        }
    }
}
```

4. Stored Procedures Used

sp_GetStudents

```
CREATE PROCEDURE sp_GetStudents
AS
BEGIN
    SELECT s.Id, s.Name, s.Age, s.Grade, s.CourseId, c.CourseName
    FROM Students s
    JOIN Courses c ON s.CourseId = c.CourseId;
END;
```

sp_GetStudentById

```
CREATE PROCEDURE sp_GetStudentById @Id INT
AS
BEGIN
    SELECT s.Id, s.Name, s.Age, s.Grade, s.CourseId, c.CourseName
    FROM Students s
    JOIN Courses c ON s.CourseId = c.CourseId
    WHERE s.Id = @Id;
END;
```

sp_AddStudent

```
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;
```

sp_UpdateStudent

```
CREATE PROCEDURE sp_UpdateStudent
    @Id INT,
    @Name NVARCHAR(100),
    @Age INT,
    @Grade NVARCHAR(5),
    @CourseId INT
AS
BEGIN
    UPDATE Students
    SET Name = @Name, Age = @Age, Grade = @Grade, CourseId = @CourseId
    WHERE Id = @Id;
END;
```

sp_DeleteStudent

```
CREATE PROCEDURE sp_DeleteStudent
    @Id INT
AS
BEGIN
    DELETE FROM Students WHERE Id = @Id;
END;
```

5. Testing the Endpoints in Swagger

1. **GET /api/students** – Successfully fetched all student records with course details.
2. **POST /api/students** – Inserted new student record using JSON payload.
3. **PUT /api/students/{id}** – Updated existing record fields.
4. **DELETE /api/students/{id}** – Deleted record from database permanently.

6. Challenges Faced

- Encountered "missing CourseId column" issue initially; solved by adding CourseId in the stored procedures.
- Had to verify the data types of parameters to ensure they match table definitions.
- Minor serialization issues were fixed by keeping the property names consistent between the model and the database.