Module 13

What to learn

- .NET Core vs. .NET Framework
- Key features of .NET Core
- Setting up the development environment (Visual Studio/VS Code)
- Introduction to RESTful APIs
    - Creating HTTP methods: GET, POST, PUT, DELETE
    - Retrieve Headers Information
- Creating a new .NET Core Web API project
    - Hands-on: Build a basic "Hello World" Web API.
    - Hands-on: Build a in Memory .NET Core WebAPI with GET/PUT/POST/PATCH/DELETE

Practice Exercise

Practice 1

Create PPT and Explain learned Topics in this module

Practice 2

Create WebAPI for day13 task

Practice 3

Exercise: Building a Simple API

Create a RESTful API with one endpoint /greet that returns "Hello, World!".

Business Logic: Extend it to accept a query parameter name and return "Hello, [Name]!".

Outcome: Understand the basics of creating an API.

Practice 4

Exercise: Statelessness in APIs

Create a RESTful API that calculates the factorial of a number passed as a query parameter (e.g., /factorial?number=5).

Business Logic: Return a 400 Bad Request if the number is not a positive integer.

Outcome: Understand stateless behavior of REST APIs.

Practice 5

Exercise: REST Constraints

Create a RESTful API with two endpoints:

- /uppercase?text=example to convert the text to uppercase.
- /reverse?text=example to reverse the text.

Business Logic: Validate that the text parameter is not empty.

Outcome: Understand RESTful principles.

Practice 6

HTTP Methods (GET, POST, PUT, DELETE)

Exercise: CRUD API

Build a RESTful API for managing a ToDo list with GET, POST, PUT, and DELETE methods.

Business Logic: Add validation for POST and PUT to ensure the title field is not empty.

Outcome: Learn how to implement CRUD operations.

Exercise: Resource Validation

Create a Books API with:

- GET /books: Returns all books.
- POST /books: Adds a new book.

Business Logic: Return a 409 Conflict if a book with the same title already exists.

Outcome: Practice HTTP method implementation.

Exercise: HTTP Method Error Handling

Extend the ToDo API to handle cases where:

- A PUT request tries to update a non-existent task.
- A DELETE request tries to delete a non-existent task.

Business Logic: Return 404 Not Found with meaningful error messages.

Outcome: Improve error handling in REST APIs.

Practice 7

Exercise: First API with Swagger

Set up a new .NET Core Web API project in Visual Studio/VS Code.

Business Logic: Add Swagger for API documentation and test a simple "Ping" endpoint.

Outcome: Learn the setup process and Swagger integration.

Exercise: Debugging

Create a simple Web API with one endpoint and set breakpoints to debug it.

Business Logic: Add logic to calculate the square of a number and debug step-by-step.
Outcome: Learn debugging basics.

Exercise: File Structure

Create a Web API project and reorganize the default folder structure:
- Move models, services, and controllers into their respective folders.

Business Logic: Add a simple service that returns a hardcoded message.
Outcome: Understand the structure of a .NET Core project.

Practice 8

Exercise: Dynamic Hello World

Create a Web API with an endpoint /hello that accepts a query parameter name and returns "Hello, [name]!".
Business Logic: If no name is provided, default to "World".
Outcome: Practice dynamic responses.

Exercise: Logging in Hello World

Add logging to the "Hello World" API to log every request and its parameters.
Business Logic: Log the current time and user agent for each request.
Outcome: Learn basic logging.

Exercise: API Versioning

Add versioning to the "Hello World" API (/v1/hello and /v2/hello).
Business Logic: In version 2, return the greeting in uppercase.
Outcome: Practice API versioning.

Assignment Exercise

Assignment 1

Assignment: Build a Mini Student Management System with .NET Core Web API

Objective

To combine knowledge of .NET Core Web API, RESTful principles, HTTP methods, and basic features of .NET Core into a practical assignment.

Problem Statement

Develop a **Student Management System API** using **.NET Core Web API** that allows users to perform CRUD operations on student data. The system should provide endpoints for managing student information such as name, age, and grade. Additionally, implement error handling, validations, and logging.

Requirements

Part 1: Project Setup

- Set up a new .NET Core Web API project in **Visual Studio** or **VS Code**.
- Organize the project into folders for Models, Controllers, and Services.

Part 2: Features and Endpoints

Get All Students

- **Endpoint**: GET /students
- **Description**: Returns a list of all students.
- **Business Logic**: Return an appropriate message if no students exist.

Get Student by ID

- **Endpoint**: GET /students/{id}
- **Description**: Fetches the details of a specific student by their ID.
- **Business Logic**: If the ID does not exist, return a 404 Not Found with an error message.

Add a New Student

- **Endpoint**: POST /students
- **Description**: Adds a new student to the system.
- **Business Logic**:
  - Validate that the name is not empty, age is between 5 and 100, and grade is a valid letter (A–F).
  - Return a 400 Bad Request for invalid inputs.

Update Student Details

- **Endpoint**: PUT /students/{id}
- **Description**: Updates the details of an existing student.
- **Business Logic**:
  - Check if the student exists before updating.
  - Only update fields that are provided in the request.

Delete a Student

- **Endpoint**: DELETE /students/{id}
- **Description**: Deletes a student by their ID.
- **Business Logic**:
  - Prevent deletion if the student is in grade A.
  - Return a 400 Bad Request with a message like "Cannot delete top-performing students".

Part 3: Additional Requirements

Logging

Implement logging for every API call with details like the endpoint, HTTP method, and timestamp.

Configuration

Use appsettings.json to store application settings such as the maximum number of students allowed (e.g., 100).

Error Handling

Implement global error handling using middleware to catch unhandled exceptions and return meaningful error messages.

Asynchronous Programming

Use async/await for all database or data layer operations.

Expected Output

A functional RESTful API that meets all requirements.
Well-structured code with proper error handling and logging.
Validation messages and meaningful error responses for invalid inputs.

Evaluation Criteria

Code Quality: Clean and modular code with proper folder structure.
Functionality: All endpoints work as expected with valid and invalid inputs.
RESTful Principles: Proper use of HTTP methods and status codes.
Error Handling and Logging: Clear error messages and logged API activities.
Completion: Ability to complete the assignment within the time limit.

Online Reference

No online Reference

.NET Core Web API

WEB API (old)

Authentication And Authorization (WEBAPI)(old)

FullStackDevelopment_WIth_Dotnet_AND_Angular