Module 14

What to learn

Controllers and Routing

Understanding Controllers in Web API
Attribute-based routing
Conventional routing
Parameter binding in routes
Hands-on: Create an API for managing "Books" with CRUD endpoints.

Practice Exercise

Practice 1

Create a Book Controller with Attribute-Based Routing

Objective: Create a controller for managing books using **attribute-based routing**.

Task:

Implement GET, POST, PUT, and DELETE methods with appropriate attribute-based routes ([Route] and [Http*] attributes).
Use routes like /api/books and /api/books/{id}.

Practice 2

Implement CRUD Operations Using Conventional Routing

Objective: Set up **conventional routing** for a BookController.

Task:

Configure the route in Program.cs as api/{controller}/{action}/{id?}.
Implement methods such as GetBooks, GetBookById, AddBook, UpdateBook, and DeleteBook.

Practice 3

Fetch Books by Author Using Parameter Binding

Objective: Fetch books by a specific author using parameter binding in the route.

Task:

Create an endpoint like /api/books/author/{authorName}.
Return a list of books written by the specified author. If no books are found, return an appropriate message.

Practice 4

Implement Pagination for Books

Objective: Implement a GET endpoint for paginated results.

Task:

Add an endpoint like /api/books?pageNumber={pageNumber}&pageSize={pageSize}.

Business Logic: Return the specified page of books, or an error message if the requested page exceeds the available data.

Practice 5

Update a Book's Price with Validation

Objective: Implement a PUT endpoint to update a book's price.

Task:

Endpoint: /api/books/{id}/update-price

Business Logic: Ensure the new price is greater than zero. Return 400 Bad Request if invalid.

Practice 6

Delete a Book with Conditional Logic

Objective: Delete a book by ID with additional validation.

Task:

Prevent deletion if the book has more than 100 copies in stock. Return an appropriate message if deletion is not allowed.

Practice 7

Retrieve Books Published in a Specific Year

Objective: Create a GET endpoint to fetch books published in a specific year.

Task:

Route: /api/books/year/{year}.

Business Logic: If no books are found for the specified year, return a 404 Not Found.

Practice 8

Search Books by Title Using Query Parameters

Objective: Implement a GET endpoint to search for books by title.

Task:

Route: /api/books/search?title={title}.

Business Logic: Perform a case-insensitive search. If no matches are found, return a helpful error message.

Practice 9

Add a New Book with Custom Validation

Objective: Implement a POST endpoint to add a new book.

Task:

Business Logic: Validate that the title is not empty, the author name has at least 3 characters, and the price is greater than 0. Return a 400 Bad Request if any validation fails.

Practice 10

Implement Sorting for Books

Objective: Create an endpoint to fetch all books sorted by title or price.

Task:

Route: /api/books?sortBy={title|price}.

Business Logic: Sort the books in ascending or descending order based on a query parameter like order=asc|desc.

Assignment Exercise

Assignment 1

Create a Restful API to create an employee,get all employees, get an employees,get an employee,update and employee http://localhost:3000/emps • AddressLine1(optional): string • AddressLine2(optional): string • AddressLine3(optional): string • assignments(optional): array • CitizenshipId(optional): integer(int64) • CitizenshipLegislationCode(optional): string • CitizenshipStatus(optional): string • CitizenshipToDate(optional): string(date) • City(optional): string • CorrespondenceLanguage(optional): string • Country(optional): string • CreationDate(optional): string(date-time) • DateOfBirth(optional): string(date) • directReports(optional): array • DisplayName(optional): string • DriversLicenseExpirationDate(optional): string(date) • DriversLicenseId(optional): integer(int64) • DriversLicenseIssuingCountry(optional): string • EffectiveStartDate(optional): string(date) • Ethnicity(optional): string • FirstName(optional): string • Gender(optional): string • HireDate(optional): string(date) • HomeFaxAreaCode(optional): string • HomeFaxCountryCode(optional): string • HomeFaxExtension(optional): string • HomeFaxLegislationCode(optional): string • HomeFaxNumber(optional): string • HomePhoneAreaCode(optional): string • HomePhoneCountryCode(optional): string • HomePhoneExtension(optional): string • HomePhoneLegislationCode(optional): string • HomePhoneNumber(optional): string • Honors(optional): string • LastName(optional): string • LastUpdateDate(optional): string(date-time) • LegalEntityId(optional): integer(int64) • LicenseNumber(optional): string • links(optional): array • MaritalStatus(optional): string • MiddleName(optional): string • MilitaryVetStatus(optional): string • NameSuffix(optional): string • NationalId(optional): string • NationalIdCountry(optional): string Assignments: Assignment Fields • ActionCode(optional): string • ActionReasonCode(optional):

string • ActualTerminationDate(optional): string(date) • AssignmentCategory(optional): string • assignmentDFF(optional): array • assignmentExtraInformation(optional): array • AssignmentId(optional): integer(int64) • AssignmentName(optional): string • AssignmentNumber(optional): string • AssignmentProjectedEndDate(optional): string(date) • AssignmentStatus(optional): string • AssignmentStatusTypeId(optional): integer(int64) • BusinessUnitId(optional): integer(int64) • CreationDate(optional): string(date-time) • DefaultExpenseAccount(optional): string • DepartmentId(optional): integer(int64) • EffectiveEndDate(optional): string(date) • EffectiveStartDate(optional): string(date) • empreps(optional): array • EndTime(optional): string • Frequency(optional): string • FullPartTime(optional): string • GradeId(optional): integer(int64) • GradeLadderId(optional): integer(int64) • JobId(optional): integer(int64) • LastUpdateDate(optional): string(date-time) • LegalEntityId(optional): integer(int64) • links(optional): array • LocationId(optional): integer(int64) • ManagerAssignmentId(optional): integer(int64) • ManagerId(optional): integer(int64) Create an Assignments API http://localhost:3000/emps/{empID}/child/assignments Get All Assignments http://localhost:3000/emps/{empID}/child/assignments Get an Assignment http://localhost:3000/emps/{empID}/child/assignments/{AssignmentID} Update an assignment http://localhost:3000/emps/{empID}/child/assignments/{AssignmentID}
Assignment 2

Assignment: Build a Complete "Library Management" Web API

Objective

Develop a Web API for managing a library system. The API should allow users to perform CRUD operations on books, implement routing strategies, and validate parameters. The assignment should demonstrate understanding of controllers, routing (attribute-based and conventional), parameter binding, and RESTful practices.

Assignment Requirements

You need to create a **Library Management API** that fulfills the following requirements:

1. Create the Project

Task: Create a new .NET Core Web API project named LibraryManagement.
Tools: Use **Visual Studio** or **VS Code**.

Configure the development environment and set up the project.

2. Implement the Book Model

Properties:

Id (integer, auto-increment)
Title (string, required)
Author (string, required)
Price (decimal, must be greater than 0)
YearPublished (integer, must be a valid year)
Stock (integer, default value 0)

3. Create the Controller

Create a BookController with the following endpoints:

3.1. Attribute-Based Routing for CRUD

GET /api/books: Retrieve all books.
GET /api/books/{id}: Retrieve a book by its ID.
Validation: Return a 404 Not Found if the book does not exist.
POST /api/books: Add a new book.
Validation: Ensure Title, Author, Price, and YearPublished are valid. Return 400 Bad Request for invalid data.
PUT /api/books/{id}: Update a book's details by ID.
Validation: Ensure the book exists before updating.
DELETE /api/books/{id}: Delete a book by ID.
Business Logic: Prevent deletion if Stock > 50 and return an appropriate message.

3.2. Custom Routes

GET /api/books/author/{authorName}: Retrieve all books by a specific author.
Validation: Return a helpful message if no books are found.
GET /api/books/year/{year}: Retrieve books published in a specific year.

4. Conventional Routing

Configure **conventional routing** in Program.cs to handle requests as:
bash

Copy code

api/{controller}/{action}/{id?}

Ensure actions in the controller match the route structure:
Example: /api/books/GetByTitle?title={title} should map to GetByTitle().

5. Business Logic

Implement the following:

Pagination: Add a GET /api/books?pageNumber={pageNumber}&pageSize={pageSize} endpoint. Return paginated results.
Sorting: Allow books to be sorted by Title or Price using /api/books?sortBy=title|price&order=asc|desc.
Stock Validation: Only return books with Stock > 0 in /api/books.

6. Error Handling

Handle common errors (e.g., 404 Not Found, 400 Bad Request) with meaningful messages.
Ensure validation messages are clear and descriptive.

7. Testing

Manual Testing:
Use tools like Postman or Swagger to test all endpoints.
Automated Testing:
Write unit tests for key endpoints like AddBook, GetBooksByAuthor, and UpdateBook.

Deliverables

The completed .NET Core Web API project.
A README.md file with:
Instructions on how to run the project.
Sample requests for each endpoint.
Details of the validation rules.
Postman collection (optional).

Online Reference

No online Reference

.NET Core Web API

WEB API (old)

Authentication And Authorization (WEBAPI)(old)

FullStackDevelopment_WIth_Dotnet_AND_Angular