Problem Statement: Building and Testing a RESTful API in ASP.NET Core

---

Objective:

You are tasked with building a RESTful API for an online movie catalog using ASP.NET Core. The API will allow clients to perform CRUD (Create, Read, Update, Delete) operations on movies and their associated directors. You will implement and test the API to ensure it adheres to RESTful principles and functions correctly, using tools like **Postman** and **Fiddler** for debugging and testing.

User Stories:

User Story 1: Implementing a RESTful API with CRUD Operations

As a developer,
I want to implement RESTful API endpoints
So that users can perform CRUD operations on movies and directors.

Acceptance Criteria:

- Create an endpoint for **GET** requests that retrieves a list of all movies.
- Create an endpoint for **GET** requests that retrieves a specific movie by its unique identifier (movie ID).
- Create an endpoint for **POST** requests that allows creating a new movie, with properties such as title, director, and release year.
- Create an endpoint for **PUT** requests that updates an existing movie's details (e.g., title, director, release year).
- Create an endpoint for **DELETE** requests that removes a movie by its ID.
- Ensure that each endpoint adheres to RESTful principles, such as using appropriate HTTP methods (GET, POST, PUT, DELETE) and meaningful URIs.

User Story 2: Using Postman for API Testing and Debugging

As a developer,
I want to test the API endpoints using **Postman**
So that I can verify that all endpoints are working correctly and return the expected results.

Acceptance Criteria:

- Use **Postman** to send requests to the API endpoints (GET, POST, PUT, DELETE) and verify that the responses match the expected output.
- Test all CRUD operations (Create, Read, Update, Delete) to ensure correct functionality, including the appropriate HTTP status codes (e.g., 200 OK, 201 Created, 400 Bad Request, 404 Not Found).
- Verify that the responses from the API include the correct data format (e.g., JSON) and correct status codes.
- Test edge cases, such as sending invalid or missing data, to ensure the API handles errors gracefully with proper error messages.

User Story 3: Using Fiddler for API Debugging

As a developer,
I want to use **Fiddler** to inspect and debug HTTP requests and responses
So that I can identify and resolve any issues with the API communication.

Acceptance Criteria:

- Use **Fiddler** to monitor and inspect the HTTP traffic between the client (Postman) and the API server.
- Ensure that the correct HTTP request methods (GET, POST, PUT, DELETE) are being sent and received.
- Verify that the API responses include correct status codes and appropriate headers (e.g., Content-Type: application/json).
- Use Fiddler to troubleshoot issues like unexpected API responses, incorrect status codes, or missing data.

User Story 4: Implementing Associations and URI Routing in ASP.NET Core

As a developer,
I want to implement associations between resources (movies and directors) and configure URI routing
So that the API can properly handle relationships between entities and provide clean, RESTful URLs.

Acceptance Criteria:

- Implement a **GET** endpoint to retrieve all movies by a specific director, using the director's ID in the URI (e.g., /directors/{directorId}/movies).

- Use **attribute routing** to define custom routes for the movie and director entities. For example, route /movies/{id} to retrieve a specific movie by ID.
- Ensure that the routing configuration follows RESTful principles, making it easy to understand the relationship between resources (movies and directors) and access them via intuitive URLs.
- Test the routing to ensure that the endpoints work as expected, and proper HTTP status codes are returned for invalid requests (e.g., 404 if a director is not found).

User Story 5: Attribute Routing for RESTful APIs

As a developer,
I want to configure **attribute routing**
So that I can map HTTP requests to specific controller actions more flexibly and clearly.

Acceptance Criteria:

- Use **attribute routing** in the MoviesController and DirectorsController to define the routes, such as [Route("api/movies/{id}")] for retrieving a movie by ID.
- Ensure that the routes are clear, concise, and follow RESTful principles.
- Test the API endpoints to verify that the attribute routes are correctly configured and resolve to the correct controller actions.

Additional Requirements:

- Use a simple in-memory database or a lightweight database like SQLite to store movie and director information.
- Implement basic validation for movie and director data, such as required fields (title, director name, etc.).
- Handle errors gracefully, returning appropriate HTTP status codes and error messages for invalid or missing data.
- Document the API endpoints using comments or a basic README to describe the available routes, methods, and expected inputs/outputs.