Lab Exercise: Building a Web API Core with GET Endpoints

Objective:

Students will learn how to build a Web API using ASP.NET Core with a focus on implementing GET endpoints. They will create a simple RESTful API that retrieves data from a mock database, simulating a modern real-world scenario of a bookstore's online catalog.

Scenario:

You are working for a tech company that has been hired to create a RESTful API for an online bookstore. The API will allow users to retrieve information about books, authors, and categories. This exercise will help you understand how to design and implement GET endpoints to fetch data from different resources.

Prerequisites:

- Basic understanding of C# and .NET Core.
- Familiarity with RESTful principles.
- Basic knowledge of JSON and HTTP methods.

Tools & Technologies:

- Visual Studio or Visual Studio Code
- .NET Core SDK
- Postman or any API testing tool
- SQL Server (optional, for advanced scenarios)

Lab Steps:

Step 1: Setup the Project

1. **Create a New ASP.NET Core Web API Project:**
    - Open Visual Studio or Visual Studio Code.
    - Create a new project using the ASP.NET Core Web API template.
    - Name the project BookstoreAPI.
2. **Install Required NuGet Packages:**
    - Install any necessary packages, such as Microsoft.EntityFrameworkCore if you are using Entity Framework for data access.

Step 2: Define the Models

1. **Create Models for Book, Author, and Category:**
    - Create a folder named Models.

- Define the following models:

```
public class Book
{
   public int Id { get; set; }
   public string Title { get; set; }
   public string ISBN { get; set; }
   public int AuthorId { get; set; }
   public int CategoryId { get; set; }
   public DateTime PublishedDate { get; set; }
}

public class Author
{
   public int Id { get; set; }
   public string Name { get; set; }
   public string Biography { get; set; }
}

public class Category
{
   public int Id { get; set; }
   public string Name { get; set; }
}
```

Step 3: Create a Mock Data Repository

1. **Implement a Repository Pattern:**
   - Create a folder named Data.
   - Add a class MockBookstoreRepository that will simulate a data source with in-memory data.

```
public class MockBookstoreRepository
{
   private readonly List<Book> _books;
```

```
    private readonly List<Author> _authors;
    private readonly List<Category> _categories;

    public MockBookstoreRepository()
    {
        _authors = new List<Author>
        {
            new Author { Id = 1, Name = "Author 1", Biography = "Biography 1" },
            new Author { Id = 2, Name = "Author 2", Biography = "Biography 2" }
        };

        _categories = new List<Category>
        {
            new Category { Id = 1, Name = "Fiction" },
            new Category { Id = 2, Name = "Science" }
        };

        _books = new List<Book>
        {
            new Book { Id = 1, Title = "Book 1", ISBN = "1111", AuthorId = 1, CategoryId = 1, PublishedDate
= new DateTime(2020, 1, 1) },
            new Book { Id = 2, Title = "Book 2", ISBN = "2222", AuthorId = 2, CategoryId = 2, PublishedDate
= new DateTime(2021, 1, 1) }
        };
    }

    public IEnumerable<Book> GetBooks() => _books;
    public Book GetBookById(int id) => _books.FirstOrDefault(b => b.Id == id);
    public IEnumerable<Author> GetAuthors() => _authors;
    public IEnumerable<Category> GetCategories() => _categories;
}
```

Step 4: Create the Controller

1. **Implement the API Controllers:**
    - Create a folder named Controllers.
    - Add a BooksController, AuthorsController, and CategoriesController.

```
[Route("api/[controller]")]
[ApiController]
public class BooksController : ControllerBase
{
  private readonly MockBookstoreRepository _repository;

  public BooksController()
  {
    _repository = new MockBookstoreRepository();
  }

  [HttpGet]
  public IActionResult GetBooks()
  {
    var books = _repository.GetBooks();
    return Ok(books);
  }

  [HttpGet("{id}")]
  public IActionResult GetBookById(int id)
  {
    var book = _repository.GetBookById(id);
    if (book == null)
    {
      return NotFound();
    }
    return Ok(book);
  }
}
```

Repeat similar steps for AuthorsController and CategoriesController.

Step 5: Test the API Endpoints

1. **Use Postman or Any API Testing Tool:**
   - Test the GET endpoints:
     - GET /api/books to retrieve all books.
     - GET /api/books/{id} to retrieve a specific book by ID.
     - GET /api/authors to retrieve all authors.
     - GET /api/categories to retrieve all categories.
2. **Test with Edge Cases:**
   - Attempt to retrieve a book with an ID that doesn't exist.
   - Ensure the API returns 404 Not Found when appropriate.

Step 6: Add Filtering and Searching (Optional)

1. **Implement Search Functionality:**
   - Modify the GetBooks method to allow searching by title or filtering by category.

```
[HttpGet]
public IActionResult GetBooks([FromQuery] string title, [FromQuery] int? categoryId)
{
  var books = _repository.GetBooks();

  if (!string.IsNullOrEmpty(title))
  {
    books = books.Where(b => b.Title.Contains(title, StringComparison.OrdinalIgnoreCase));
  }

  if (categoryId.HasValue)
  {
    books = books.Where(b => b.CategoryId == categoryId);
  }

  return Ok(books);
}
```

2. **Test the Enhanced Endpoints:**
    - Test the search functionality using different query parameters.

Step 7: Reflection and Discussion

1. **Discuss the RESTful Principles:**
    - How does the API adhere to RESTful principles?
    - What are the benefits of using GET endpoints in an API?
2. **Extend the Exercise:**
    - Add POST, PUT, and DELETE endpoints to the API.
    - Implement a real database using Entity Framework Core.

Submission:

Students should submit:

- The source code of the API.
- A Postman collection or equivalent showing all test cases.
- A brief reflection on what they learned from the exercise.

Evaluation Criteria:

- Correct implementation of GET endpoints.
- Adherence to RESTful principles.
- Code quality and organization.
- Successful testing of all endpoints, including edge cases.
- Thoughtful reflection on the exercise.

This lab exercise will help students solidify their understanding of Web API development, specifically focusing on GET endpoints in a realistic and modern application scenario.