Module 18

What to learn

- **Introduction to Entity Framework Core**
  - What is Entity Framework Core?
  - Setting up EF Core in a .NET Core project
  - Creating DbContext and DbSets
  - Hands-on: Add EF Core to the "Books" API and configure a database.
- **Code-First Approach**
  - Code-first migrations
  - Seeding the database
  - Updating the database schema
  - Hands-on: Create and migrate tables for the "Books" API.
- **Querying Data**
  - LINQ basics
  - Async queries with EF Core
  - Query filtering, sorting, and pagination
  - Hands-on: Add filtering, sorting, and pagination to the "Books" API.
- **Repository Pattern**
  - Why use the repository pattern?
  - Creating a repository and unit of work
  - Hands-on: Refactor the "Books" API to use the repository pattern.
- **Error Handling and Logging**
  - Global exception handling
  - Using ILogger for logging
  - Implementing custom error responses
  - Hands-on: Add centralized error handling to the "Books" API.

Practice Exercise

Practice 1

Understanding Entity Framework Core

Explain what Entity Framework Core is and how it differs from previous versions of Entity Framework. Discuss its advantages in modern .NET Core applications.

Practice 2

Setting Up EF Core

Walk through the steps to set up Entity Framework Core in a new .NET Core project. Include details on necessary NuGet packages and configuration settings.

Practice 3

Creating DbContext and DbSets

Define a simple `DbContext` class for a 'Books' API. Include at least two `DbSet` properties for entities such as `Book` and `Author`. Explain the purpose of each.

Practice 4

Code-First Migrations

Describe the process of creating and applying code-first migrations in EF Core. What commands are used, and what files are generated during this process?

Practice 5

LINQ Basics

Provide examples of basic LINQ queries that can be used to retrieve data from a `DbSet` in EF Core. Explain how these queries can be used to filter and sort data.

Practice 6

Implementing the Repository Pattern

Discuss the repository pattern and its benefits in managing data access. Provide a brief outline of how to create a repository for the 'Books' API.

Practice 7

Error Handling and Logging

Explain the importance of error handling and logging in a Web API. Describe how to implement global exception handling and use `ILogger` for logging errors.

Assignment Exercise

Assignment 1

Books API Development Assignment

Develop a 'Books' API that utilizes Entity Framework Core for data access. The API should support the following functionalities:

Entity Framework Core Setup: Set up EF Core in your project and create a `DbContext` for managing books and authors.

Code-First Approach: Implement code-first migrations to create the necessary database schema for books and authors. Seed the database

with initial data.

Querying Data: Implement endpoints for retrieving books with filtering, sorting, and pagination capabilities. Use LINQ to query the data.

Repository Pattern: Refactor your data access code to use the repository pattern, ensuring separation of concerns.

Error Handling and Logging: Implement global exception handling and logging using ILogger. Ensure that meaningful error messages are returned to the client.

Functional Flow: Ensure that all outputs are displayed on the user interface (UI) only, not in the console. The API should be able to handle requests and return appropriate responses based on the operations performed.

Business Logic: Clearly outline any specific rules or conditions for data handling and processing, such as validation rules for book entries and author associations.

Online Reference

No online Reference

.NET Core Web API

WEB API (old)

Authentication And Authorization (WEBAPI)(old)

FullStackDevelopment_WIth_Dotnet_AND_Angular