ASP.NET Core MVC - Roadmap

1. Getting Started with MVC Projects

- **Directory Structure**: Detailed explanation of the MVC directory (Controllers, Views, Models, wwwroot) and the role of Program.cs in configuration.
- **Routing and Controllers**: Custom routing, attribute routing, and handling various HTTP requests.
- **Actions in Controllers**: Exploring different return types and handling different HTTP requests.

---

2. Controllers and Routing

- **Routing Basics**: Attribute-based routing and custom route parameters.
- **Controller Actions**: How to handle different return types like JSON, HTML, and redirect results.

---

3. Views in MVC

- **Creating and Organizing Views**: Using view models and best practices for organizing views.
- **_Layout.cshtml**: Creating reusable layouts for consistent page structure.
- **Bootstrap Integration**: Introduction to Bootstrap and using its grid system, forms, and navigation.
- **Razor View Engine**: Using Razor for conditional content, loops, and strongly-typed views.
- **HTML Helpers and Tag Helpers**: Leveraging helpers to generate forms, links, and other elements in a clean way.

---

4. Working with Models

- **Models and Data Annotations**: Detailed use of data annotations like `[Required]`, `[StringLength]`, and creating custom validators.
- **View Models and Data Models**: Mapping between data models and view models to separate business logic from UI.
- **Model Binding**: Automatic model binding from form data to controller actions.

---

5. Forms and Input

- **Form Creation with HTML Helpers**: Build forms with `@Html.BeginForm()` and generate form elements like text boxes and dropdowns.
- **Form Validation**: Server-side validation with data annotations and client-side validation using jQuery Validation.

---

6. Dependency Injection and Repository Pattern

- **Service Lifetimes**: Discuss how to choose between `Scoped`, `Transient`, and `Singleton` in dependency injection.
- **Repository Pattern**: Abstraction of database operations with repositories to separate business logic from data access.
- **Unit of Work Pattern**: Manage transactions across multiple repository operations.

---

7. Database and Entity Framework (EF Core)

- **EF Core Setup**: How to install EF Core and configure a connection string for SQL Server.
- **Code First Migrations**: Applying migrations to manage database schema updates.

- **Seeding the Database**: Automatically populate the database with initial data.
- **Entity Relationships**: Define one-to-many and many-to-many relationships using EF Core.
- **LINQ**: Use LINQ for querying data in a readable way.
- **Tracking and Detaching Entities**: Optimizing performance by detaching entities when needed.

---

8. CRUD Operations

- **Full CRUD Cycle**: Create, Read, Update, and Delete operations for managing records.
- **Pagination and Filtering**: Enable pagination and filtering for large data sets.

---

9. REST API Development

- **What is a REST API?**: Introduce REST principles and why APIs are important in web development.
- **Creating a REST API**: Use ASP.NET Core to create simple APIs using controllers and routing.
- **Returning JSON**: Show how to return JSON responses from API endpoints.
- **Model Binding in APIs**: Handle POST, PUT, and DELETE requests, including validation.
- **Swagger Integration**: Demonstrate how to use Swagger for API documentation and testing.
- **API Versioning**: Explain the need for versioning in APIs and how to implement it.

---

10. Authentication and Authorization

- **ASP.NET Core Identity**: User registration, login, and managing user profiles.
- **Role-Based Authorization**: Restrict access to parts of the application based on user roles.
- **External Authentication**: Implement third-party login with Google, Facebook, etc.
- **JWT Authentication (Optional)**: For building APIs with token-based authentication.

---

11. Advanced Views and AJAX

- **Partial Views and View Components**: Reusable components for modularizing your views.
- **AJAX in ASP.NET Core**: How to make asynchronous requests using AJAX without refreshing the page, and handling responses dynamically.
- **JavaScript Integration**: Use JavaScript and jQuery for front-end interactivity.