**DOT NET Training Notes**

**DAY 9**

**TESTING**

**Unit Testing and Test-Driven Development:**

**Introduction to Unit Testing:**

**What is Unit Testing :** Isolation testing of software modules.

**Benefits of Unit testing:**

Bug detection, Code behaivour. Refactoring, overall helps in improving quality of code.

**Basic Terminology:**

**Test Case:** A single scenario to validate piece or functionality.

**Test Suite:** A collection of test cases.

**Test Fixture:** Setups code executed before or after tests. (Biolerplate).

**Test Runner:**  A tool that run test cases.

**Testing Frameworks:**

**Nunit:** Open Source and very popular for .NET

**MSTest:** Microsoft’s own testing framwork.

**xUnit:** Modern unit test framework for .NET Core

**Writing test cases:**

1. **Arrange:** Set up test data.
2. **Act:** Call the method under test.
3. **Assert:** Verify the output.

Day 10

Createing Project:

1. Search **ASP .NET Web App (razor pages)**
2. Choose Framwork then Select authentication type(individual account).

We 3 approaches to connect DB

1. Entity frame code first approach
2. Entity framwork DB first approach
3. Adio dotnet approach

Add Data to DB

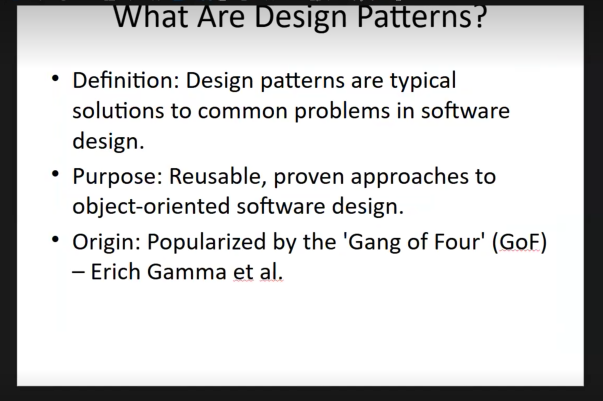
Tool >> NuGET Package manage >> package Manager Console

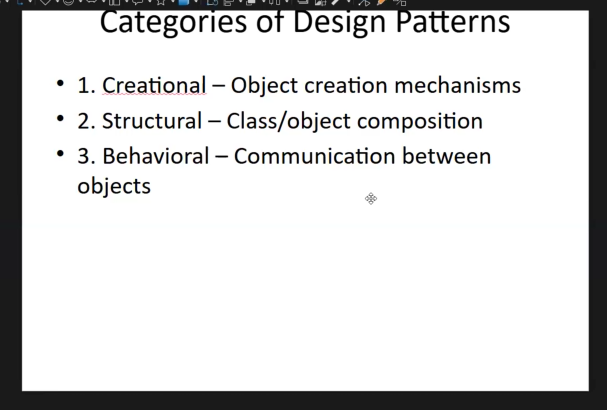
Run migration

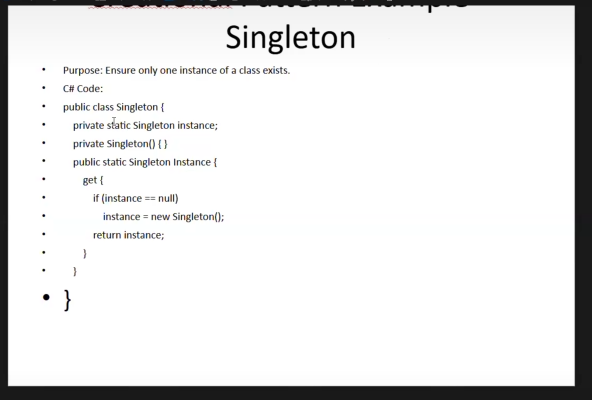
1. add-migration first
2. update-database

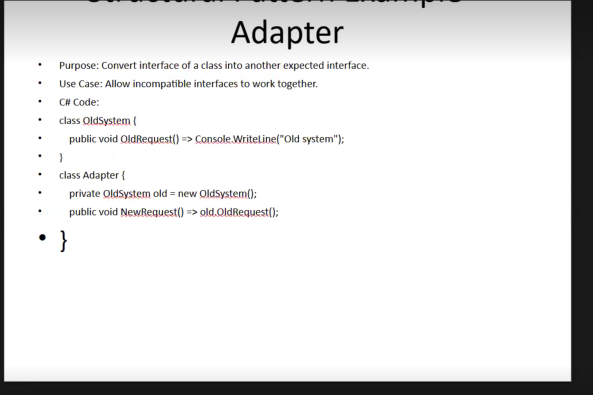
Day 11

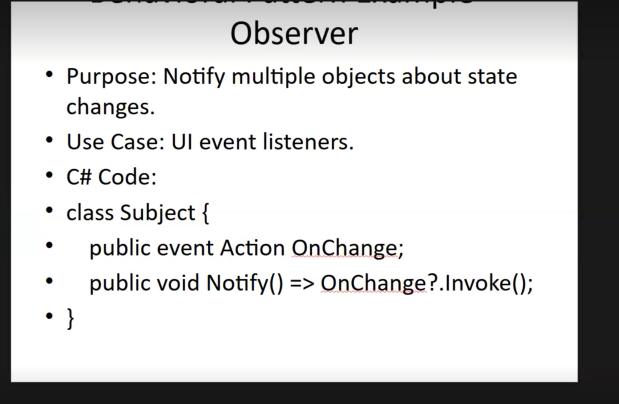
**Design Patterns:**

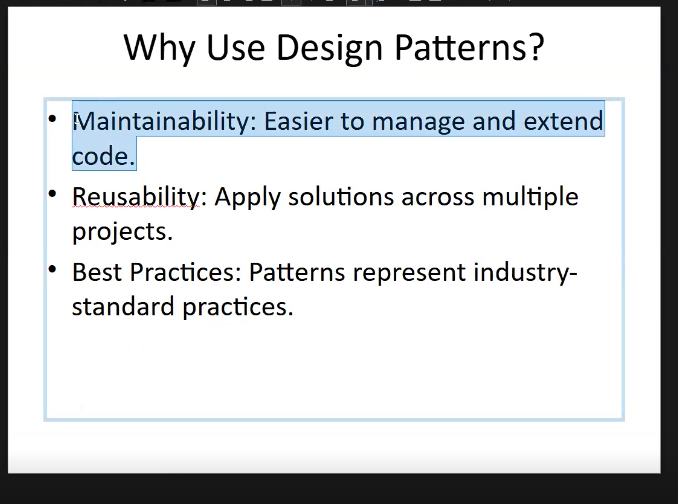
****

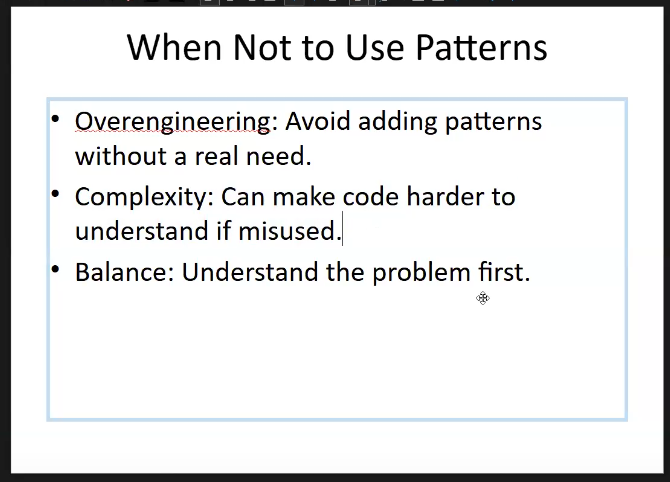
****

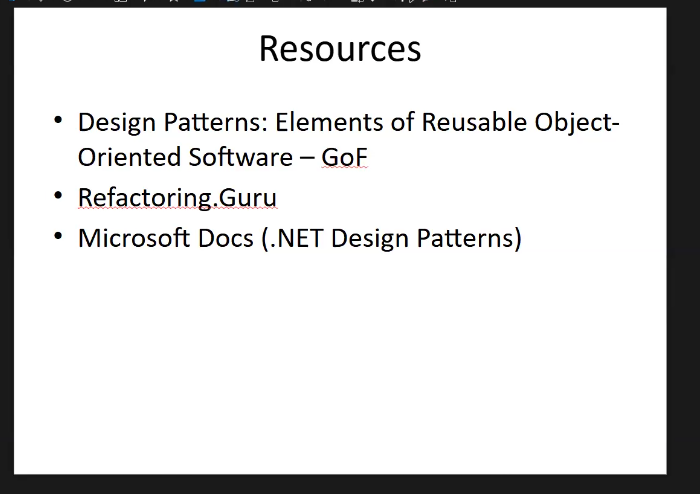
****

****

****

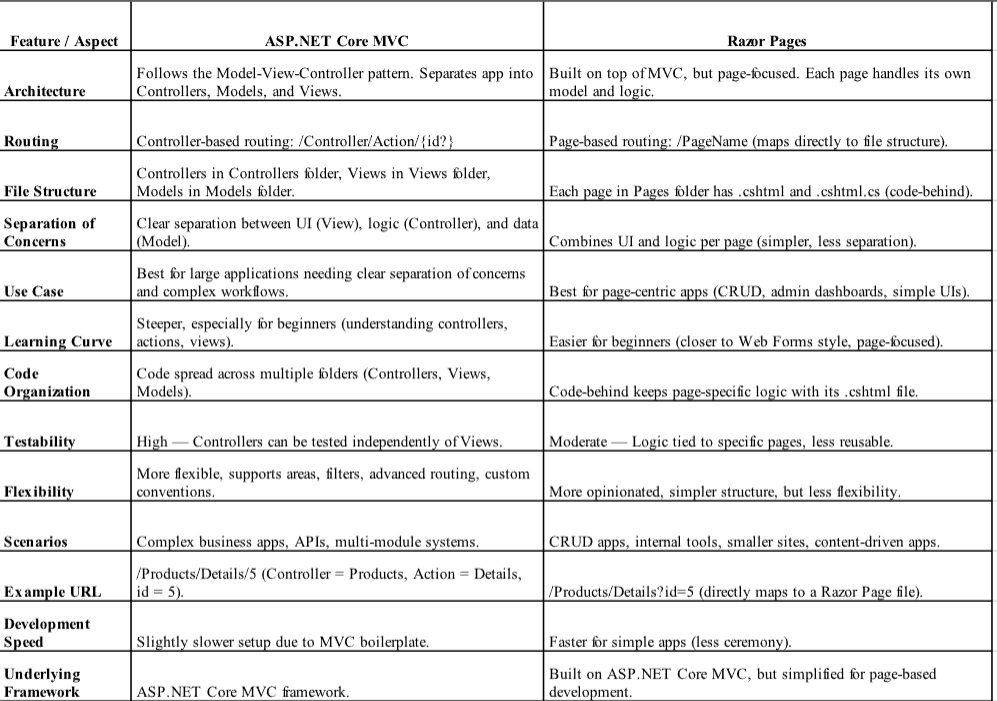
****

****

****

**DAY 25**

**Comparison between Razor Pages and MVC in .NET**

****

| **Aspect** | **Razor Pages** | **MVC** |
| --- | --- | --- |
| Pattern | Page-focused; each page handles its logic and UI together. | Controller-focused; separates concerns into Models, Views, Controllers. |
| Structure | Combines view and page logic in one file with a code-behind file. | Divides application into three components: Model, View, and Controller. |
| Routing | Uses page-based routing, with URLs mapped to pages. | Uses controller and action-based routing. |
| Use Case | Simple web applications or pages with straightforward UI logic. | Complex applications needing full control over UI, behavior, and separation of concerns. |
| Learning Curve | Easier for beginners familiar with traditional web forms or single file structure. | Steeper learning curve due to separation and more concepts involved. |
| Testability | Less testable compared to MVC because logic is tied closer to the UI. | Easier to unit test because of separation of concerns. |
| Code Organization | Less suited for large applications where clear separation is needed. | Better organization for large, complex applications. |
| Handler Methods | Uses handler methods (OnGet, OnPost) within the page model class. | Uses separate action methods across controllers for various HTTP verbs. |
| Performance | Similar performance, as both run on ASP.NET Core runtime. |  |

**When to Use Razor Pages**

* **For simple or small web applications with less complexity.**
* **When building page-centric UI with minimal business logic.**
* **For developers who prefer a more straightforward approach to build pages.**
* **When rapid development with less boilerplate code is desired.**
* **Suitable for scenarios where the app is primarily CRUD and form-based pages.**

**When to Use MVC**

* **For large and complex applications requiring fine separation of concerns.**
* **When you need rich control over routing, behavior, and UI.**
* **When building applications where testability and maintainability are priorities.**
* **For applications that require a clear API and UI separation.**
* **When working in teams where roles are divided among developers for UI, business logic, and data.**

**What is the difference between viewBag, ViewData and TempDAta and to use ??**

**1. ViewBag**

* **Type:** Dynamic property (uses C# dynamic feature).
* **Purpose:** Pass data from controller to view.
* **Lifetime:** Only for the current request.
* **Syntax:** ViewBag.Message = "some message";
* **Use case:** When you want a quick, loosely typed way to send small amounts of data to a view without casting.

Syntax: ViewBag.Message = "Hello from ViewBag";

**2. ViewData**

* **Type:** Dictionary (ViewDataDictionary), stores key-value pairs as object.
* **Purpose:** Pass data from controller to view.
* **Lifetime:** Only for the current request.
* **Syntax:** ViewData["msg"] = "some message";
* **Use case:** When you want strongly typed keys but still need data for one request. Requires casting to original type in view.

Example: ViewData["msg"] = "Hello from ViewData";

**3. TempData**

* **Type:** Dictionary (TempDataDictionary), stores key-value pairs.
* **Purpose:** Pass data from one request to another (e.g., after a redirect).
* **Lifetime:** Maintains data for one subsequent request. After it’s read once, it’s cleared automatically (unless kept).
* **Syntax:** TempData["msg1"] = "some message";
* **Use case:** When you want to pass temporary data (like success/failure messages) across requests, such as after form submission and redirect.

Example: TempData["msg1"] = "Hello from TempData";

Linq Queries

<https://drive.google.com/file/d/1ginn42yDxff6vL1qdSR5aor-NEfw8ri0/view>