Chapter 5 and 6

ASP.NET VS ASPNET Core

	ASP.NET	ASP.NET Core
Platform	Windows-only; built on the .NET Framework.	Cross-platform; built on the modern .NET Core framework.
Hosting	Can only be hosted on IIS.	Flexible hosting options: IIS, Nginx, Apache, or self-hosted.
Performance	Performance is good but constrained by older architecture.	Higher performance due to lightweight architecture and optimizations.
Architecture	Monolithic; tightly coupled components.	Modular; allows lightweight and flexible configurations.
Open Source	Partially open source.	Fully open source and developed with community input.
Dependency Injection	Requires third-party tools for implementation.	Built-in dependency injection support.
Use Cases	Suitable for legacy applications.	Ideal for modern applications, cloud solutions, and microservices.

ASP.NET (Active Server Pages .NET)

ASP.NET, developed by Microsoft, is a popular framework for building web applications. It provides different models for web development, each suited to various needs. The three main frameworks under the ASP.NET umbrella are:

ASP.NET Web Forms

A traditional event-driven programming model that uses a drag-and-drop approach with server controls.

Features:

- Supports Rapid Application Development (RAD) with a visual designer.
- Uses ViewState to maintain state across postbacks.
- Heavily reliant on server-side controls.
- Ideal for enterprise applications that require a quick UI development approach.

Use Case: Suitable for developers who prefer a Windows Forms-like experience for web development.

ASP.NET (Active Server Pages .NET)

ASP.NET MVC (Model-View-Controller)

A lightweight, testable, and highly structured framework that follows the MVC pattern.

Features:

- Separates application logic into Model (data), View (UI), and Controller (handles requests).
- Promotes clean architecture and testability.
- Uses convention-based routing instead of event-driven development.
- Offers full control over HTML, CSS, and JavaScript.

Use Case: Best suited for complex applications requiring high maintainability, testability, and scalability.

ASP.NET Web Pages

Overview: A lightweight framework for building simple web pages using Razor syntax.

Features:

- Uses Razor markup for embedding C# code into HTML.
- Simplifies development with minimal setup.
- Suitable for small-scale applications and rapid prototyping.

Use Case: Ideal for developers who need to create simple, content-focused web pages quickly.

MVC (Model-View-Controller)

MVC is a software design pattern commonly used in .NET for developing web applications. It helps in separating the concerns of an application by dividing it into three main components: **Model**, **View**, and **Controller**. This architecture makes applications more maintainable and scalable, and it allows multiple developers to work on different aspects of the application independently.

Model

- The **Model** represents the data or business logic of the application.
- It encapsulates the data, typically coming from a database, and the logic for data manipulation, validation, and processing.
- The **Model** is responsible for retrieving data, saving data, and implementing business rules.
- It can be a class or set of classes, representing domain entities like Customer, Order, etc.

MVC (Model-View-Controller)

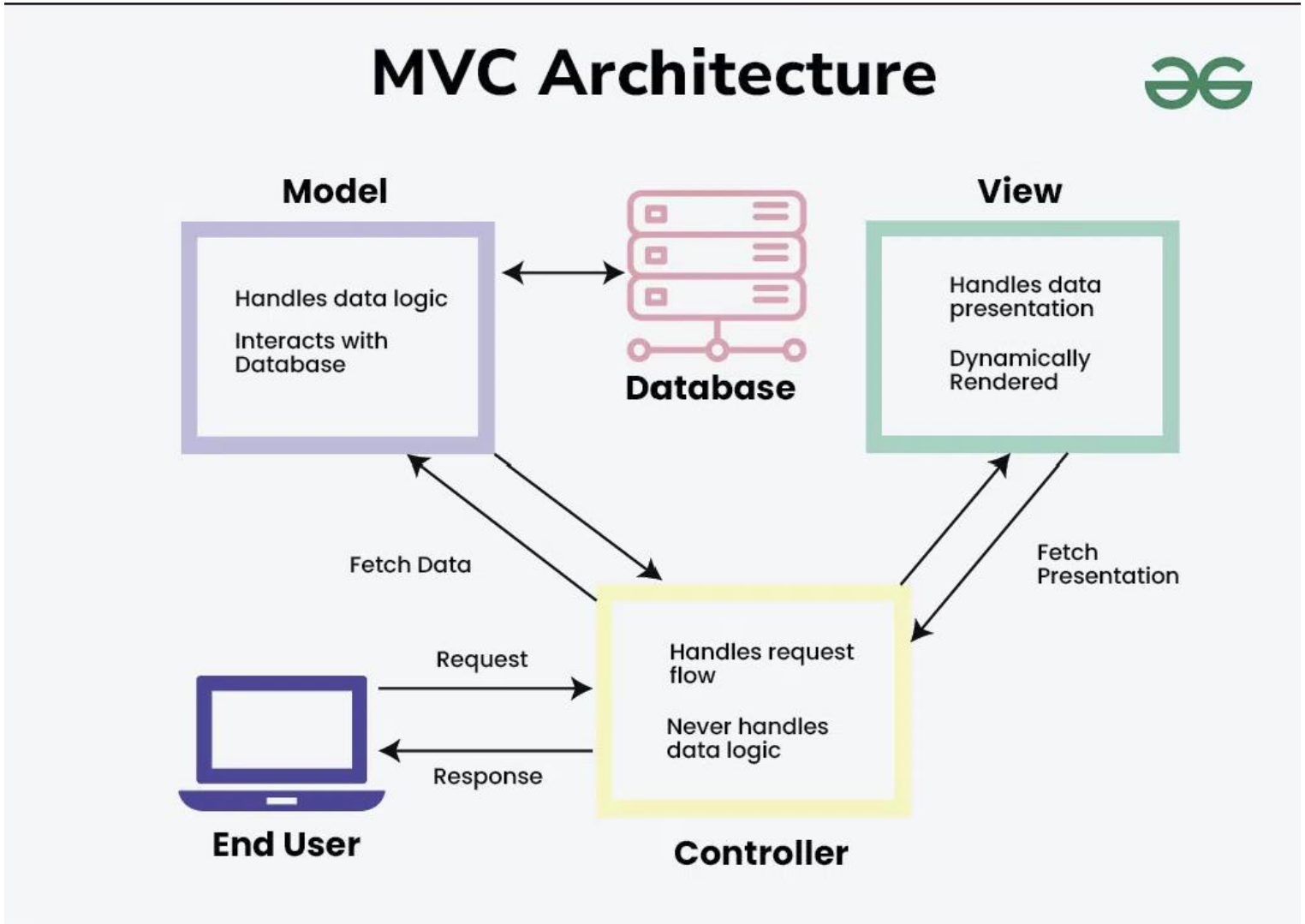
View

- The **View** is responsible for displaying the user interface (UI) of the application.
- It renders the data from the **Model** in a format that is suitable for the user to view.
- In web applications, the **View** could be HTML, CSS, or Razor Views (if using ASP.NET).
- The **View** does not directly contain logic to handle data, but it formats and presents the data provided by the **Controller**.

Controller

- The **Controller** acts as the intermediary between the **Model** and **View**.
- It handles user input (like button clicks, form submissions) and makes decisions based on that input.
- The **Controller** retrieves data from the **Model** and updates the **View** accordingly.
- It contains the application flow logic and communicates with the **Model** to process requests..

MVC Architecture



URL Routing

URL Routing in .NET MVC is the process of mapping incoming requests (URLs) to appropriate controller actions. It defines the pattern in which the URLs should be structured and specifies which **Controller** and **Action** should handle the request.

Key Concepts of URL Routing:

- **Controller**:
  - The **Controller** is responsible for handling incoming HTTP requests.
  - It contains action methods that process the request and return a response.
- **Action**:
  - An **Action** is a method inside the controller that is executed when a particular route is matched.
  - Actions typically return a **View** (HTML content) or data (JSON, XML, etc.).
- **Route**:
  - A **Route** is a URL pattern that the routing engine matches to determine which controller and action to invoke.
  - A route is defined using a pattern with placeholders such as `{controller}`, `{action}`, and `{id}`.
- **Default Route**:
  - The **default route** is the fallback route used if no specific route is matched.
  - It is typically configured to map the URL to a default controller (`HomeController`) and action (`Index`).

URL Routing

Syntax for Defining Routes:

Routes are configured in the `Program.cs` (or `RouteConfig.cs`).

A route is defined using `MapControllerRoute` or `MapDefaultControllerRoute`.

Example of Default Route in **.NET 8**:

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

- `{controller=Home}`: The default controller is `HomeController`.
- `{action=Index}`: The default action is `Index`.
- `{id?}`: `id` is an optional parameter.

URL Routing

Positional Parameters:

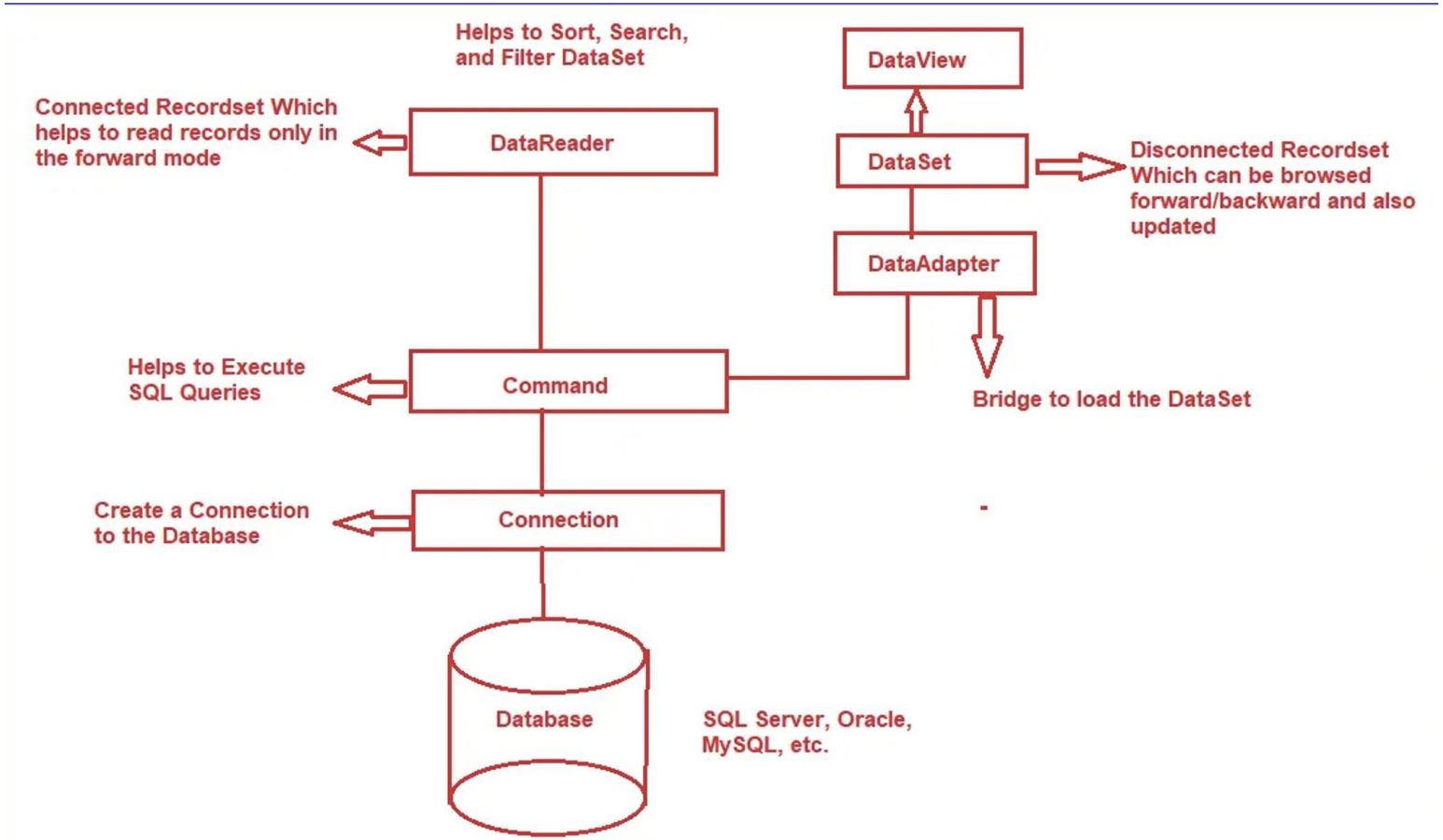
- Parameters are included in the URL as `{parameter}` and mapped to method parameters in the controller action.
- Example: `Products/Details/5` maps to `{controller=Products}/{action=Details}/{id}`

Optional Parameters:

- Parameters can be made optional by using `UrlParameter.Optional`.
- Example: `{id?}` makes the `id` parameter optional.

- **Parameter Constraints**:
  - You can add constraints to parameters to restrict the values (e.g., only digits).
  - Example: `id:int` ensures `id` is an integer.

ADO.NET Architecture



Note

- Remaining theory in pdf and discussed in class code examples , refer pushed code in github