CONTENTS

CLASS 1: GIT

- **Why we need version control?**

Ans: Track changes to files and code over time, Collaborate with others on projects, Recover from mistakes, Experiment with new ideas without fear of breaking things, Deploy software with confidence.

- **One-step version controlling vs Two-step version controlling:**

Feature	One-step version controlling	Two-step version controlling
Definition	Version control system that automatically creates a new version of a file whenever it is saved.	Version control system that requires users to explicitly create a new version of a file by committing it to the repository.
Examples	Subversion	Git, Mercurial (Not popular now)
Advantages	Simple to use, no need to explicitly commit changes.	More powerful and flexible, allows users to stage changes before committing them.
Disadvantages	Can lead to accidental commits, does not provide as much control over the versioning process.	More complex to use, requires users to explicitly commit changes.

- **What is Github?**

Github is a hosting company for version controlling.

- **What is Github? Why we use this?**

Github is a hosting company for version controll. We use github because code safety, sharibility, code availability (we can use it any time from anywhere) and etc.

- **Using git through SSH?**

First go to Puttygen.exe for key and then connect this using pageant.exe.

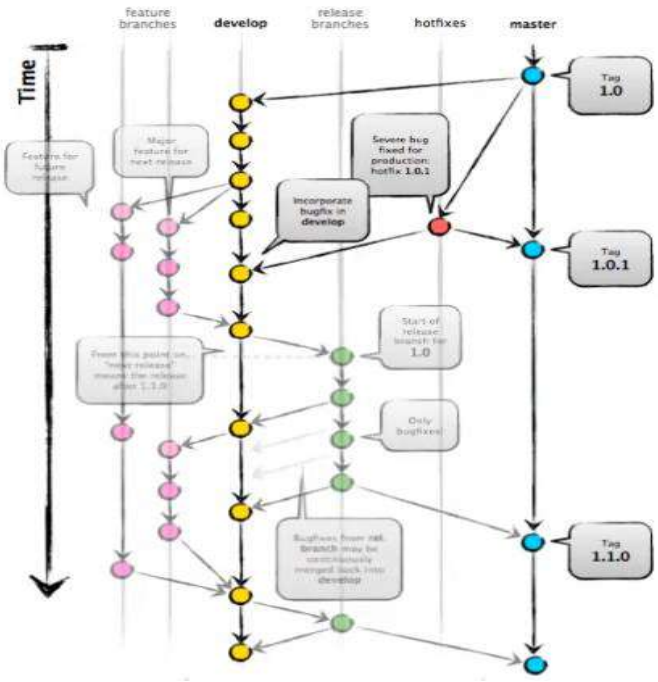
- **Using git through SSH?**

First go to Puttygen

- **Using git commands?**

Git command added top on this book. Check it

- If we work on team then must mind this term -> commit, pull, push
- Conflict two types: Merge conflict (which file will keep is not findout by git) and tree conflict (Folder structure problem. i.e I created a folder but someone delete this folder)
- If work with develop branch. Finally release production level software into master (picture below)



Required Changes

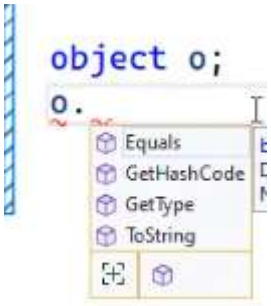
- Regular developers must create feature branch from develop branch with naming convention: feature/{Jira Ticket Number} (i.e. feature/XYZ-333)
- Develop branch can't be modified without Pull Request (PR) by regular developers.
- Develop branch can be changed directly by elevated permissions (Teal lead/Admin)
- Master branch needs to be restricted in the same way.
- Pull request must be reviewed and approved for merge. Prefered, if this can be restricted too.
- Each branch and PR should trigger a CI build (Task for DevOps, developer do not need to worry)
- PR should be rejected if the build fails in CI
- Conflict should be resolved by PR creator by merging develop in their branch.
- PR should be reviewed within 24 hours.

CLASS 3: REFLECTION AND DEPENDENCY INJECTION

- **Reflection**: Reflection is a feature in C# that allows inspection and manipulation of metadata, types, and members of assemblies at runtime. (রিফ্লেকশন জানলে আমরা জানতে পারবো কিভাবে Entity framework work করে, কিভাবে শুধু .dll আপডেট করে পুরো সফটওয়ার আপডেট করা যায়)
- **Why Use Reflection**: Reflection is used to dynamically inspect, invoke, and manipulate types and members, enabling tasks like late binding, creating instances, and accessing private members without knowing them at compile time.
- If we know reflecion then we can understand how software update without new install. We know about .dll
- Reflections has two turning points. Assembly (collection of type) and Type (int, float, class).
- **Why use Assembly:** The use of assemblies in C# is essential for deploying, organizing, and facilitating runtime reflection, enabling code sharing, versioning, and modular development. (Assembly ইউজ করতে using.system.Reflection ব্যবহার করতে হয়। Assembly আসলে .dll ফাইল বা বান্ডেল)

- Object থেকে Type পেতে পারি।

- Type হচ্ছে ডেটাটাইপের টাইপ থেকে আসছে । এই টাইপ লেখতে কোনো রেফারেন্স এড করা লাগে না। একটা এসেম্বলির ভেতর অনেক টাইপ থাকতে পারে। এসেম্বলি হলো কালেকশন অফ টাইপ।
- Object হচ্ছে প্রথম আইটেম বা অব্জেক্ট C# এর




```
Type t2 = typeof(int);
Type t3 = (3).GetType();
```

- JSON deserialization in C# refers to the process of converting a JSON string into an object or a data structure
- Dotnets builtin item or first item is 'object'. If we declare object x; and if we hover on this x then we can see Equals, GetHashCode, GetType and ToString. We can findout Type like this
- We can findout Type x = typeof(int) also
- If we go to defination on int then we can see int actually Int32, string will related with Stirng. Dotnet give this easy name to remember
- Assembly is used as static class. In a Assembly we can see GetType, Load, GetAssembly, LoadFile (we can load file path to load).
- NewtonSoft nuget use for json serialize and deserialize
- Here is config.txt, we want to deserialize this string and use dll (we get dll from anther project)

```
{
    ClassName : "Chart"
}
```

```
// Get the parent directory three levels above the current directory. (we want to come
ReflectionExamples from ReflectionExamples\bin\Debug\net7.0)
DirectoryInfo directory = new DirectoryInfo(Directory.GetCurrentDirectory())
    .Parent.Parent.Parent;

// Read the contents of a file with a name containing "config" from the directory.
string config = File.ReadAllText(directory.GetFiles()
    .Where(x => x.Name.Contains("config")).First().FullName);

// Deserialize the contents of the "config" file into a dynamic object.
dynamic configJson = JsonConvert.DeserializeObject(config);

// Load an assembly (DLL) from the parent directory.
Type t = Assembly.LoadFile(directory.Parent.GetFiles()
    .Where(x => x.Extension == ".dll").First().FullName).GetTypes()
    .Where(x => x.Name == configJson.ClassName.ToString()
    && x.GetInterface("IPlugin") != null).First();

// Get the constructor of the type specified in the "config" file.
ConstructorInfo constructor = t?.GetConstructor(new Type[] { typeof(string) });

// Create an instance of the type using the constructor with a "Demo Report" parameter.
```

```
object o = constructor?.Invoke(new object[] { "Demo Report" });

// Get the "Start" method of the type with specified binding flags.
MethodInfo method = t?.GetMethod("Start", BindingFlags.NonPublic | BindingFlags.Instance,
new Type[] { });

// Invoke the "Start" method on the instance created earlier.
object r = method?.Invoke(o, new object[] { });
```

- If we create two class and pass one class to another class interface is called dependancy injection. Here `(new Class2());` this is a dependency injection

```
Class1 c1 = new Class1(new Class2());
c1.DoSomething();
```

- Dependency Inversion Priciple: এটা একটা অব্জেক্ট অরিয়েন্টেড ডিজাইন প্রিন্সিপাল এটার মেইন কথা হলো Depndency উলটা করতে হবে।
- Is a relationtionship ইনহেরিটেন্স বুঝায়। অন্যদিকে has a relationship ...
- Dependency injection হলো Public Class2(Class3 @class){}. Dependency injection is used to promote loose coupling, improve testability, and enhance maintainability by injecting dependencies into a class rather than letting the class create or manage its dependencies.
- অন্যদিকে Dependency inversion হচ্ছে ধরি Project1 and Project2 তে দুটি ক্লাস আছে লাইক Class1 and Class2। যদি আমি সরাসরি Project2 এর Class2 ব্যবহার করি Project1 এ তাহলে ডিপেন্ডেন্ট হয়ে যাচ্ছি। তাই এটা উল্টানোর জন্য Project1 এ IClass ইন্টারফেস ব্যবহার করবো এবং সেটা ব্যবহার করবো অন্যদিকে Project2 এর Class2 কে বাধ্য করবো যাতে ইন্টারফেস ইনহেরিট করে।

(ইনভারসের আগে Project1 Project2 কে রেফ করতো এখন Project2 Project1 কে রেফ করে)

```
// Project 1
public class Class1
{
    private IClass _class;

    public Class1(IClass @class)
    {
        _class = @class;
    }

    public void DoSomething()
    {
        Console.WriteLine("Doing Something");
        int x = 10;
        _class.Print(x);
    }
}
public interface IClass
{
    void Print(object o);
}
```

```
// Project 2
public class Class2 : IClass
{
    public void Print(object o)
    {
        PrintSomething(o);
    }

    public void PrintSomething(object o)
    {
        Console.WriteLine(o);
    }
}

// Driver
Class1 c1 = new Class1(new Class2());
c1.DoSomething();
```

- Use of GetType():

```
using System;
public class Class1
{
}
class Program
{
    static void Main()
    {
        // Create instances of different types
        string text = "Hello, Reflection!";
        int number = 42;
        double pi = 3.14;
        Class1 c = new Class1();

        // Use GetType() to get the runtime type information
        Type textType = text.GetType();
        Type numberType = number.GetType();
        Type piType = pi.GetType();
        Type cc = c.GetType();

        // Display the runtime types
        Console.WriteLine($"Variable 'text' has type: {textType}");
        Console.WriteLine($"Variable 'number' has type: {numberType}");
        Console.WriteLine($"Variable 'pi' has type: {piType}");
        Console.WriteLine($"Variable 'c' has type: {cc}");
    }
}

// Output:
// Variable 'text' has type: System.String
// Variable 'number' has type: System.Int32
// Variable 'pi' has type: System.Double
// Variable 'c' has type: Class1
```

- Use of GetProperty():

```
using System;
using System.Reflection;

public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
}

class Program
{
    static void Main()
    {
        // Create an instance of the Person class
        Person person = new Person
        {
            FirstName = "John",
            LastName = "Doe",
            Age = 30
        };

        // Get the type of the Person class
        Type personType = person.GetType();

        // Get the public properties of the Person class
        PropertyInfo[] properties = personType.GetProperties();

        // Display property names and values
        Console.WriteLine("Properties of the Person class:");
        foreach (PropertyInfo property in properties)
        {
            object value = property.GetValue(person);
            Console.WriteLine($"{property.Name}: {value}");
        }
    }
}


// Output:
// Properties of the Person class:
// FirstName: John
// LastName: Doe
// Age: 30
```

- Some commonly used method of System.Reflection:

Method	Description	Example

Type.GetType(string typeName)	Gets the Type object with the specified name.	Type myType = Type.GetType("System.String");
Assembly.Load(string assemblyString)	Loads an assembly given its display name.	Assembly myAssembly = Assembly.Load("MyAssembly");
Assembly.GetExecutingAssembly()	Gets the assembly that contains the currently executing code.	Assembly executingAssembly = Assembly.GetExecutingAssembly();
Assembly.GetTypes()	Gets the types defined in an assembly.	Type[] types = myAssembly.GetTypes();
Type.GetMethod(string name)	Gets a MethodInfo representing a specific method.	MethodInfo method = myType.GetMethod("MyMethod");
Type.GetMethods()	Gets an array of all methods defined on the type.	MethodInfo[] methods = myType.GetMethods();
Type.GetProperty(string name)	Gets a PropertyInfo representing a specific property.	PropertyInfo property = myType.GetProperty("MyProperty");
Type.GetProperties()	Gets an array of all properties defined on the type.	PropertyInfo[] properties = myType.GetProperties();
Type.GetField(string name)	Gets a FieldInfo representing a specific field.	FieldInfo field = myType.GetField("MyField");
Type.GetFields()	Gets an array of all fields defined on the type.	FieldInfo[] fields = myType.GetFields();
Type.GetConstructor(Type[] types)	Gets a ConstructorInfo representing a specific constructor.	ConstructorInfo constructor = myType.GetConstructor(new Type[] { typeof(int) });
Type.GetConstructors()	Gets an array of all constructors defined on the type.	ConstructorInfo[] constructors = myType.GetConstructors();
MethodInfo.Invoke(object obj, object[] parameters)	Invokes a method dynamically on an object.	object result = method.Invoke(myInstance, new object[] { arg1, arg2 });
PropertyInfo.GetValue(object obj)	Gets the value of a property on an object.	object value = property.GetValue(myInstance);
FieldInfo.GetValue(object obj)	Gets the value of a field on an object.	object value = field.GetValue(myInstance);
Activator.CreateInstance(Type type)	Creates an instance of a type.	object instance = Activator.CreateInstance(myType);
Attribute.GetCustomAttributes(MemberInfo element, Type attributeType)	Retrieves an array of custom attributes applied to a member.	Attribute[] attributes = Attribute.GetCustomAttributes(myMethod, typeof(MyAttribute));

CLASS 4: ASP.NET

- When we will create a project then we should must select Indivisual Project (If we select this, then ms will auto implement nessessary feature like login, registration, area and etc)
- In asp.net folder structure we can see launchSetting which has http ports, which is help to open project
- Wwwroot is public accessable folder, people can see html js from their browser (We can store people photo and others here)
- What is razor view page:
- In a Model we keep POCO class
- POCO stands for Plain Old CLR (or C#) Object. A POCO class is a simple, lightweight class in C# that does not depend on any specific framework, base class, or library. POCO classes are used to represent data structures or entities in an application, and they typically do not include any behavior or methods beyond simple property accessors. Example:

```
public class Person
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime BirthDate { get; set; }
}
```

- What is CLR: CLR stands for Common Language Runtime. It is a fundamental component of the Microsoft .NET framework and is responsible for executing and managing .NET applications. The CLR provides several key functions, including:

Feature	Description
Just-In-Time Compilation (JIT)	Compiles Intermediate Language (IL) code into native machine code at runtime.
Memory Management	Manages memory allocation and garbage collection to prevent memory leaks.
Security	Enforces code access security and provides various security mechanisms.
Exception Handling	Handles exceptions and supports structured exception handling.
Thread Management	Provides support for multithreading and manages thread execution.
Type Safety	Enforces type safety to reduce runtime errors.
Assembly Loading	Loads and manages assemblies containing code, metadata, and resources.
Interop Services	Supports interaction with code in other languages using P/Invoke and COM interop.
Debugging and Profiling	Provides debugging and profiling capabilities for troubleshooting.
Language Neutrality	Allows the use of multiple programming languages within the .NET framework.

- Here we do dependancy injection in a constractor `ILogger<HomeController> logger`

```
private readonly ILogger<HomeController> _logger;
private readonly IConfiguration _config;

public HomeController(ILogger<HomeController> logger,
IConfiguration config)
{
    logger = logger;
    _config = config;
}
```

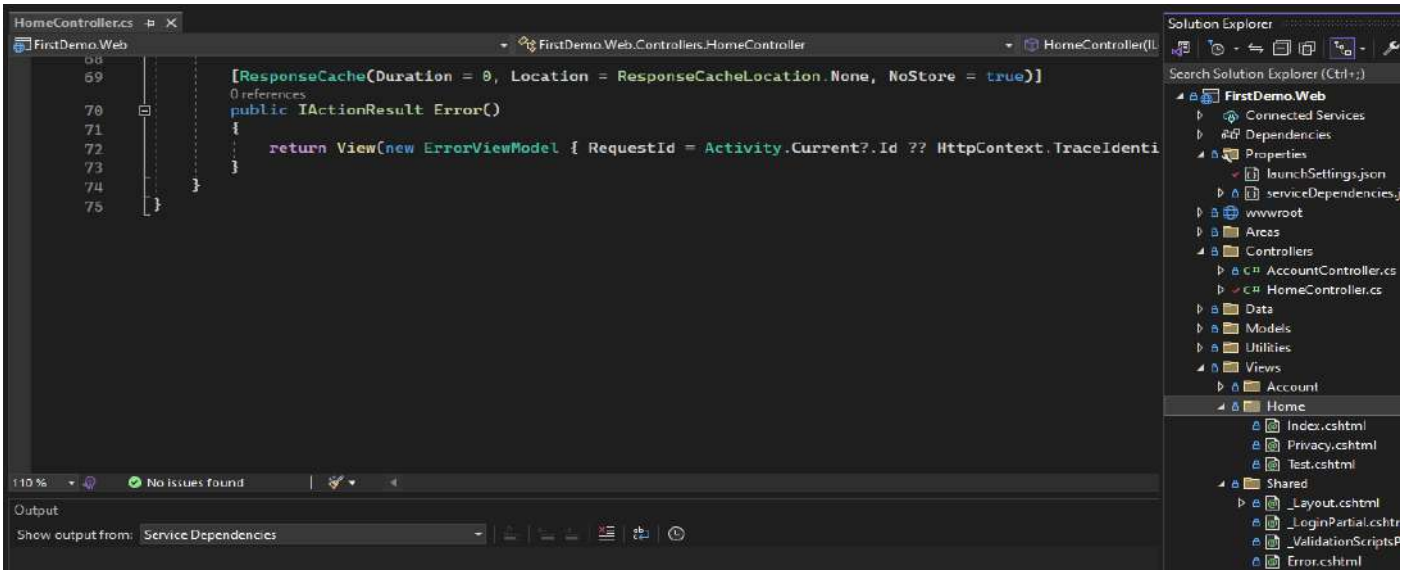
- Why use ILogger

Feature	Description
Purpose	Used for logging in ASP.NET Core applications.
Framework	Part of Microsoft.Extensions.Logging.
Functionality	Records and manages log messages and events.
Output Options	Can write logs to various destinations (e.g., console, files, centralized logging systems).
Importance	Crucial for debugging, monitoring, and diagnosing issues.
Dependency Injection	Often injected into classes that need logging capabilities.

- Why use IConfiguration:

Feature	Description
Purpose	Used for managing configuration settings in ASP.NET Core applications.
Framework	Part of Microsoft.Extensions.Configuration.
Functionality	Reads configuration values from various sources (e.g., JSON files, environment variables).
Configuration Sources	Can read data from multiple sources, providing flexibility in configuration management.
Importance	Essential for separating configuration from code, adapting to different environments, and enhancing maintainability.
Dependency Injection	Often injected into classes that need access to configuration settings.

- If We create any action which razor view created into shared folder. Like here Error() created into shared. How dotnet find it? Why not give error? -> Answer is, Dot net first search Home folder it not found then search it in shared with name convension



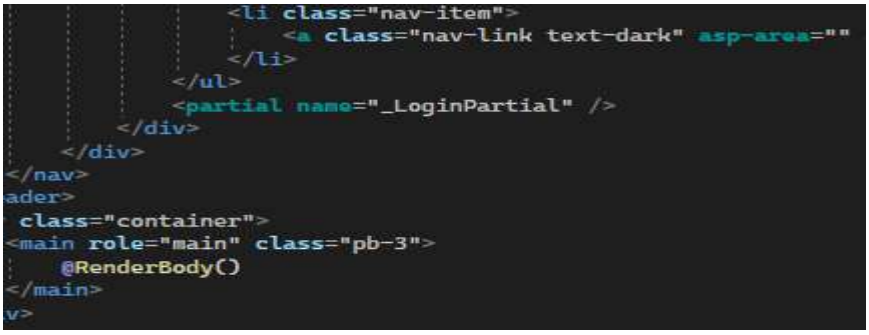
- In a public constructor and public method we always try to keep use Interface like, IActionResult , public HomeController(Ilogger)
- In a _Layout.cshtml file, we keep such code which will be preview all page, and there have a @RenderBody which means when we write body portion for another action's view then it will come on @RenderBody and executed (but @ RenderBody not run directly like abstract class)
- Asp-append-version (Ms tag helper)

```
@* ASP.NET Core appends a version number or hash to the URL of a static file to force
browsers to download the latest version when the file content changes, enabling cache-
busting. *@
<script src="~/js/site.js" asp-append-version="true"></script>
```

- Some middle ware

```
app.UseHttpsRedirection()
    .UseStaticFiles()
    .UseRouting()
    .UseAuthentication()
    .UseAuthorization()
    .UseSession();
```

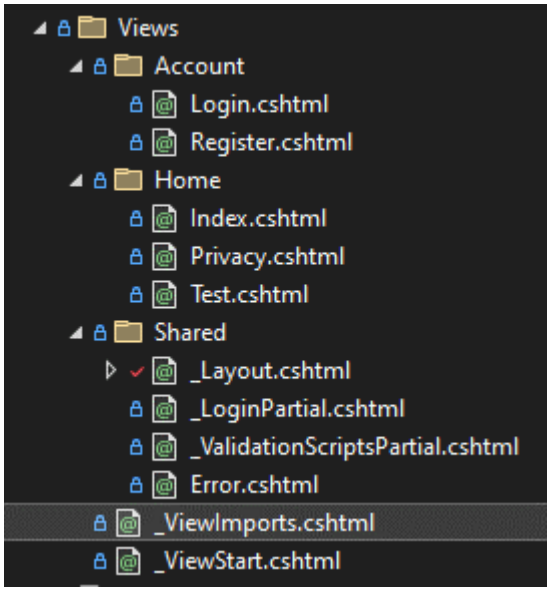
- If we need login registration or other view page (which is not come with controller's action view) then we can create partial view directly _LoginPartial.cshtml and keep path it into _Layout.cshtml
  Which will be reusable.



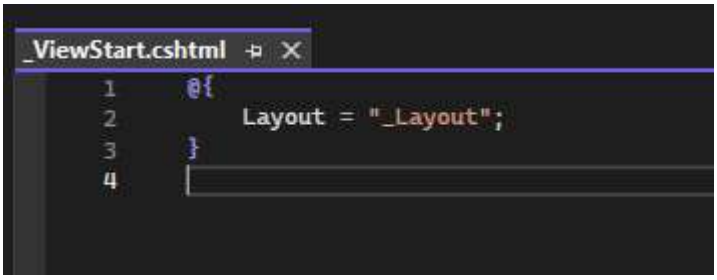
- ViewData["Title"] is a key-value dictionary for storing data to be passed between the controller and the view in ASP.NET Core. We can chage Home Page title anytime. We can add new Layout in every page. If we want to keep _Layout.cshtml in every page then we should remove Layout from this code. If we want to remove default layout then we write Layout = null.

```
@{
    ViewData["Title"] = "Home Page";
    Layout = "_NewLayout";
}
```

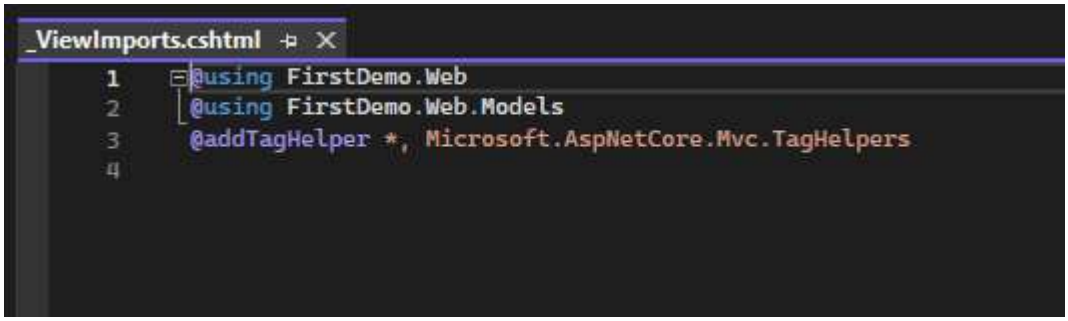
- View file structure



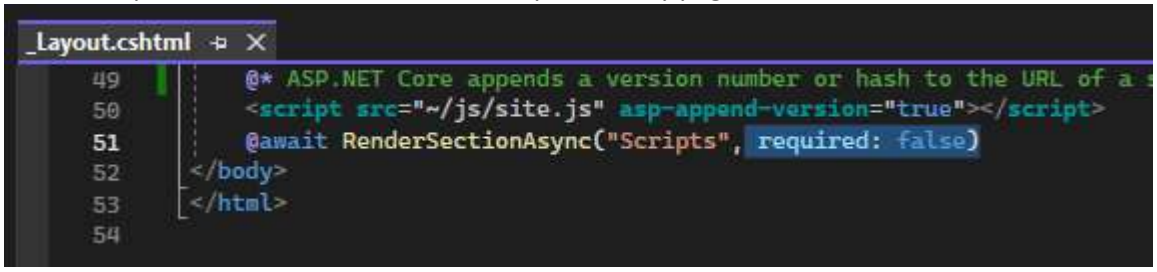
- In _ViewStart.cshtml we set main _Layout



- If we want to use common namespace then we can add it _ViewImports.cshtml. As a result we do no set namespace in every razor page



- If we set required false then we can write script from any page



- Register services with the dependency injection (DI) container

Lifetime	Description	Use When
Transient	A new instance is created for each request or usage.	- Service is lightweight and stateless. - No shared state.

Singleton	A single instance is created and shared across the entire application.	- Service should be shared globally. - Maintains state across multiple requests.
Scoped	A single instance is created and shared within the scope of a single HTTP request.	- Service should be scoped to a specific request. - Per-request data or state management.

- Transient: যতোবার ICourse ইন্টারফেস পাবো ততবার যদি নতুন instance চাই Course ক্লাসের তাহলে এটা ব্যবহার করবো

```
builder.Services.AddTransient<ICourse, Course>()
```

- Singleton: যতোবার ICourse ইন্টারফেস পাবো শুধু একটাই instance পাইতে চাই তবে এটা ব্যবহার করবো (This is rule of singleton design pattern. i.e যদি ক্লাস বানাতে হয় new দিয়ে তবে বার বার কিন্তু ক্লাস বানাতে হবে না)

```
builder.Services.AddSingle<ICourse, Course>()
```

- Scoped: একটা স্কোপের ভিতরে একাধিক বার কল হলেও একটাই instance পাবো (Like foreach loop block or other blocks)

```
builder.Services.AddScoped<ICourse, Course>()
```

- We can use Autofac nuget pakage to avoid ms register services with the dependency injection (DI). We can get extra feature in autofac.
- Mind it:

```
// ASP.NET Core with a different DI container (e.g.,
Microsoft.Extensions.DependencyInjection):
  builder.Services.AddScoped<ICourse, Course>();
  // Autofac:
  builder.RegisterType<Service>().As<IService>().InstancePerLifetimeScope();
```

CLASS 4: CLEAN ARCHITECTURE

- Here is Dotnet CLI to create solution with cmd. Suppose we need to create a dotnet work directory in src folder. Just go to src folder then run cmd and paste this 3 line then it will create src/DigiCV.sln

```
dotnet new mvc -n DigiCV.Web
dotnet new sln -n DigiCV
dotnet sln add DigiCV.Web
```

Clean Architecture is a software architectural approach that emphasizes the separation of concerns, maintainability, and testability of a software system. It provides a structured way to design applications by defining clear boundaries and dependencies between different parts of the system. Clean Architecture, as described by Robert C. Martin (Uncle Bob), is characterized by the following key principles:

1. **Separation of Concerns**: Clean Architecture enforces a clear separation between the core business logic and external concerns such as the user interface, databases, and frameworks. This separation makes it easier to understand, maintain, and extend the system.
2. **Dependency Rule**: The architecture follows the Dependency Rule, which states that dependencies should always point inwards toward the core of the application. In other words, the innermost circle should have no knowledge of the outer circles.
3. **Use of Abstractions:** Interfaces and abstract classes are used to define contracts and abstractions, allowing for interchangeable implementations and testability. This promotes the use of Dependency Injection.
4. **Testability**: Clean Architecture emphasizes testability by isolating the core business logic from external dependencies. This makes it easier to write unit tests and ensure the correctness of the system.
5. **Framework Independence**: The core of the application should not be tightly coupled to any specific framework, technology, or database. This allows for flexibility and adaptability when it comes to changing or upgrading external components.
6. **Screaming Architecture**: Clean Architecture encourages naming conventions that "scream" the intent and purpose of each component. For example, it should be clear from the component's name what its role is in the system.
7. **Adherence to SOLID Principles**: The architecture follows SOLID principles (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion) to promote modularity and maintainability.

Clean Architecture is language-agnostic and can be applied to various programming languages and technologies. It provides a conceptual framework for organizing code in a way that prioritizes the core business logic while keeping external concerns at bay. By adhering to Clean Architecture principles, developers can create maintainable, scalable, and testable software systems that are less prone to becoming tightly coupled or difficult to evolve over time.

Domain-Driven Design (DDD) is a software development approach that focuses on creating a well-structured, maintainable, and effective domain model for a specific problem domain. It provides a set of principles, patterns, and practices for designing and building complex software systems. Here's a specific answer:

Domain-Driven Design (DDD):

1. **Focus**: DDD places a strong emphasis on understanding and modeling the core domain of a software application. The "domain" refers to the specific problem space or subject matter that the software is built to address.
2. **Ubiquitous Language**: DDD promotes the use of a shared and consistent terminology (known as the "ubiquitous language") between developers and domain experts. This language helps bridge the communication gap and ensures a common understanding of the domain's concepts and processes.
3. **Bounded Contexts**: DDD divides a large, complex domain into smaller, more manageable bounded contexts, each with its own distinct domain model. This segmentation helps manage complexity and allows different parts of the application to have their own definitions and interpretations of domain concepts.
4. **Aggregates and Entities**: DDD introduces the concepts of aggregates and entities. Aggregates are clusters of related entities and value objects treated as a single unit for data changes. Entities represent objects with distinct identities and lifecycles within the domain.
5. **Value Objects**: DDD encourages the use of value objects to represent domain concepts that have no distinct identity but are defined by their attributes. Value objects are immutable and can be shared.
6. **Repositories**: DDD uses repositories to abstract the data access layer, allowing the domain to interact with its data without being tightly coupled to specific data storage technologies.
7. **Services**: DDD introduces domain services for operations that don't naturally fit within entities or value objects. Domain services encapsulate domain logic and operations that cross aggregate boundaries.

8. **Event-Driven Architecture**: DDD often employs event-driven architecture to capture and respond to domain events, allowing for loose coupling and scalability.
9. **Testing**: DDD promotes thorough testing, including unit testing of domain logic and behavior using the ubiquitous language of the domain experts.
10. **Continuous Refinement**: DDD acknowledges that domain models evolve over time and supports continuous refinement of the model as the understanding of the domain deepens.
11. **Strategic and Tactical Design**: DDD distinguishes between strategic design (high-level organization of bounded contexts) and tactical design (low-level modeling of aggregates, entities, and value objects).
12. **Collaboration**: DDD encourages close collaboration between domain experts and developers, fostering a shared understanding of the domain and its challenges.

Domain-Driven Design is especially valuable for complex software systems where the understanding of the domain is critical to success. By following DDD principles, developers can create software that aligns closely with the real-world problem it's intended to solve, leading to more effective, maintainable, and adaptable solutions.

- Clean architecture is closly related with DDD. Clean architecture is a architecture of DDD
- Various Scope:

Scope	**Description**	**Use Case**
InstancePerDependency	A new instance is created for each request.	Short-lived, stateless components.
InstancePerLifetimeScope	A new instance is created for each lifetime scope.	Web applications (per-request) and custom scope lifetimes.
InstancePerRequest (Alias of PerLifetimeScope)	A new instance is created for each HTTP request.	Web applications (per-request).
SingleInstance	A single instance is shared across the entire application.	Long-lived, shared services.
InstancePerMatchingLifetimeScope	Created within a specific matching lifetime scope.	Custom control over lifetime scopes.
InstancePerOwned	For owned instances that should be disposed when no longer needed.	Short-lived components with ownership.

Scope	Custom scope with user-defined lifetimes.	Specific requirements with custom scopes.

- MVC originally is a design pattern. এটার উপর বেস করে যে ফ্রেইমোয়ার্ক তৈরি হচ্ছে সেগুলো এটাকে আর্কিটেকচারের মতো ব্যবহার করছে এই কারনে এটাকেও আর্কিটেকচার বলা হয়।
  More details: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

- IQueryble: We use this when we need to get data from database
- IEnamurable: When we want to work with list dictionary then we use this

CLASS 6: SERILOG

- Why use Autofac: Autofac is used for dependency injection, enabling the management and resolution of application components, leading to better maintainability and testability. (We can use RegisterType(WebModule) outside of the program file)
- Why use Serilog: Serilog is used for structured logging, offering flexible and efficient log event capturing and storage for enhanced debugging and monitoring. (We want to hide error from user and want to view what error happen that's why we use this serilog)

- When we use Serilog then serilog replaces Microsoft logger thats why we can use ILogger with Serilog also.

```
builder.Host.UseSerilog((ctx, lc) => lc
    .MinimumLevel.Debug()
    .MinimumLevel.Override("Microsoft", LogEventLevel.Warning)
    .Enrich.FromLogContext()
    .ReadFrom.Configuration(builder.Configuration));
```

- Tag helper vs Html helper: Both Tag Helpers and HTML Helpers serve the purpose of generating HTML elements within ASP.NET Core applications, but Tag Helpers offer a more HTML-like and readable approach, as well as better tooling support. HTML Helpers are still available and can be useful in certain scenarios, but Tag Helpers are the recommended choice for modern ASP.NET Core applications. (Html helper । Html helper এ কোড লেখার চেয়ে Tag Helper এ সুবিধা বেশি এবং সহজ। যদিও দুটোই পিউর HTML কর্নভার্ট হয়ে যায় ব্রাউজারে. তবে যারা ফ্রন্টেন্ড ডেভেলপার তাদের জন্য ট্যাগ হেল্পার বুঝা সুবিধা)

Aspect	**Tag Helpers**	**HTML Helpers**
Language and Syntax	Use HTML-like syntax within Razor views.	Use C# methods to generate HTML elements.
Readability	Enhances readability of Razor views, as they resemble HTML tags.	May result in less readable views due to programmatic HTML generation.
Intellisense Support	Provides excellent Intellisense support as they are written in HTML syntax.	May have limited Intellisense support due to dynamic C# method calls.
Type Safety	Provides compile-time type checking, reducing runtime errors.	May require runtime checks to ensure correctness.
Maintainability	Offers cleaner and more maintainable views by separating markup from C# code.	Can lead to more complex and less maintainable views with intertwined C# and HTML.
Testability	Promotes testability as views are easier to unit test.	May require more effort to test HTML generation.

Extensibility	Easily extendable by creating custom Tag Helpers.	Extending HTML Helpers may be more complex.
Integration with JavaScript	Seamlessly integrates with client-side scripts and libraries.	May require additional work to integrate with JavaScript.
Integration with CSS	Facilitates integration with CSS for styling and layout.	May require additional work for styling integration.
Built-in HTML Encoding	Automatically encodes output, reducing the risk of cross-site scripting (XSS) vulnerabilities.	Requires manual HTML encoding to prevent XSS attacks.

- Some common Microsoft ASP.NET Core MVC tag helpers and their purposes:

Tag Helper	**Purpose**
asp-action	Specifies the action method for generating a URL.
asp-controller	Specifies the controller for generating a URL.
asp-area	Specifies the area for generating a URL.
asp-route	Provides route values for generating a URL.
asp-antiforgery	Generates an anti-forgery token for forms.
asp-validation-for	Generates validation attributes for model properties.
asp-for	Generates HTML form elements based on model properties.
asp-page	Specifies the Razor page for generating a URL.
asp-route-*	Provides values for individual route segments.
asp-all-route-data	Includes all route data when generating a URL.
asp-append-version	Appends a version number to static file URLs for cache-busting.
asp-* (ViewData/TempData)	Displays data from ViewData or TempData in views.

- Some common HTML Helpers in ASP.NET MVC and their purposes

HTML Helper	**Purpose**
Html.ActionLink	Generates an anchor (link) element to an action method.
Html.BeginForm	Renders the opening <form> tag for HTML forms.
Html.CheckBox	Renders an HTML checkbox input.
Html.DropDownList	Creates a dropdown list for selecting a value from a list.
Html.Hidden	Generates a hidden input field.
Html.Label	Generates a label element for a model property.
Html.ListBox	Creates a multi-select list box.
Html.Password	Renders a password input field.
Html.RadioButton	Generates an HTML radio button input.
Html.TextBox	Creates a text input field.
Html.TextArea	Renders a multi-line text input area.
Html.ValidationMessage	Generates a validation message for a model property.
Html.ValidationSummary	Renders a validation summary for the entire form.

- Here is example of Html helper and Tag helper

```
@* HTML Helper *@
@Html.ActionLink("Test Page", "Test", "Home", null, new { style = "background-color:red"})
@* Tag Helper *@
<a asp-action="Test" asp-controller="Home" style="background-color:red">Test Page</a>
```

- HTML Anti-Forgery Tokens is a fundamental security measure to protect web applications from CSRF (Cross-Site Request Forgery.) attacks and maintain data integrity. It is a recommended practice in web development, especially for actions that involve sensitive operations or data modifications.

```
@Html.AntiForgreyToken()
```

- We can create custom HTML helper (LMM says we can create Tag helper)

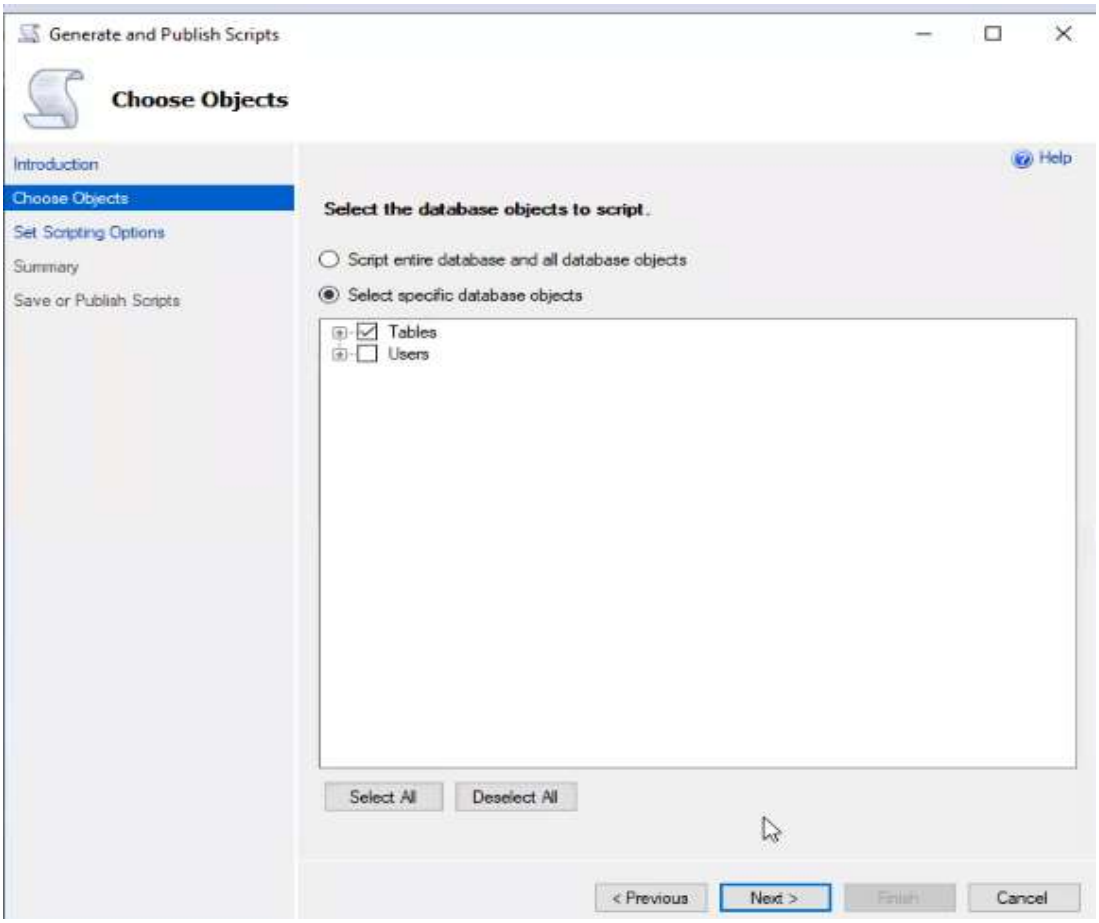
    - **Partial View:**
      A partial view in ASP.NET MVC is a reusable view that can be rendered within other views.

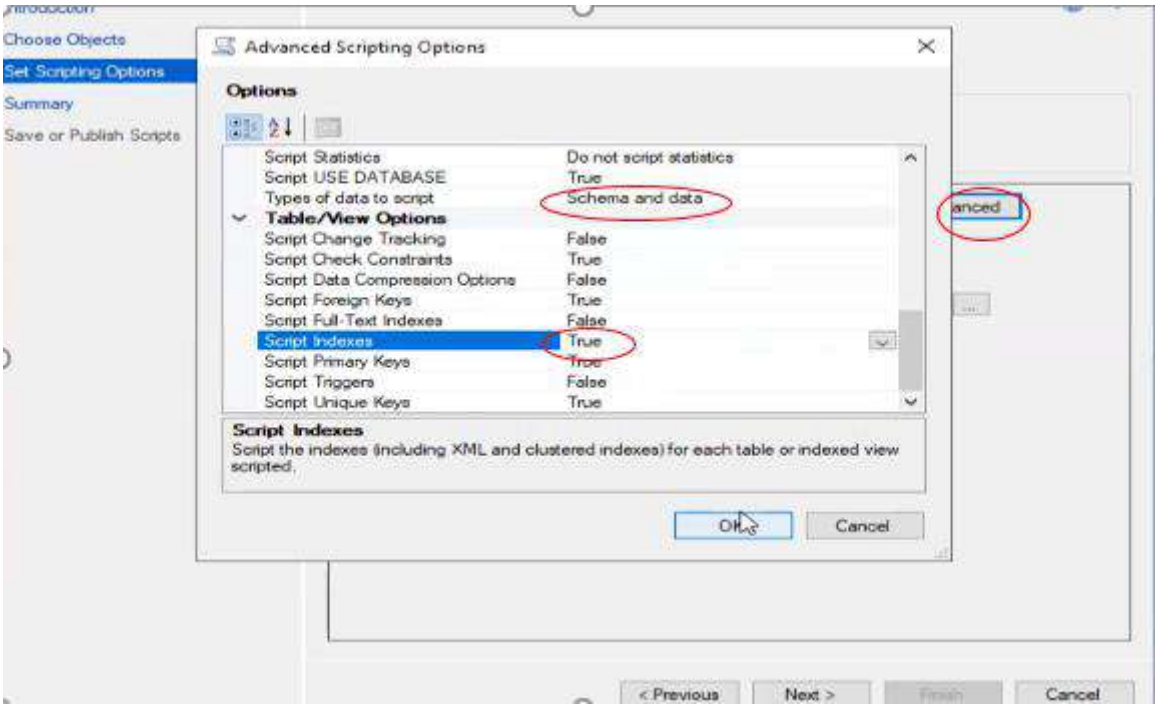
    - **Sections:**
      Sections in ASP.NET MVC allow defining named content areas in a layout file to be populated by content from individual views.

    - Difference and similarites of Clean architecture and DDD:

CLASS 7 : SQL SERVER

- We can use nvarchar instead of nchar. Nvarchar is more efficient
- If I Cascade database table then mean if I delete a row from a table then other row which is related with this table will be also deleted
- If I do database shrink then database will be removed cache
- Normal database backup: Just go to specific database and right click on mouse > select backup
- Database backup with sql for (like php): go to specific database and right click on mouse > select script (check video 1:12:05





- Store procedure vs Function

Stored Procedure (SP)	Function (UDF - User Defined)
SP can return zero, single or multiple values.	Function must return a single value (which may be a scalar or a table).
We can use transaction in SP.	We can't use transaction in UDF.
SP can have input/output parameter.	Only input parameter.
We can call function from SP.	We can't call SP from function.
We can't use SP in SELECT/ WHERE/ HAVING statement.	We can use UDF in SELECT/ WHERE/ HAVING statement.
We can use exception handling using Try-Catch block in SP.	We can't use Try-Catch block in UDF.

- A Basic Store procedure

```
USE [AspnetB8]
GO
/****** Object:  StoredProcedure [dbo].[GetCourses]    Script Date: 2/11/2023 8:03:57 PM ****
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [dbo].[GetCourses]
AS
BEGIN
    declare @count int;
    set @count = 1;

    select @count = count(*) from courses;

    print @count;

END
```

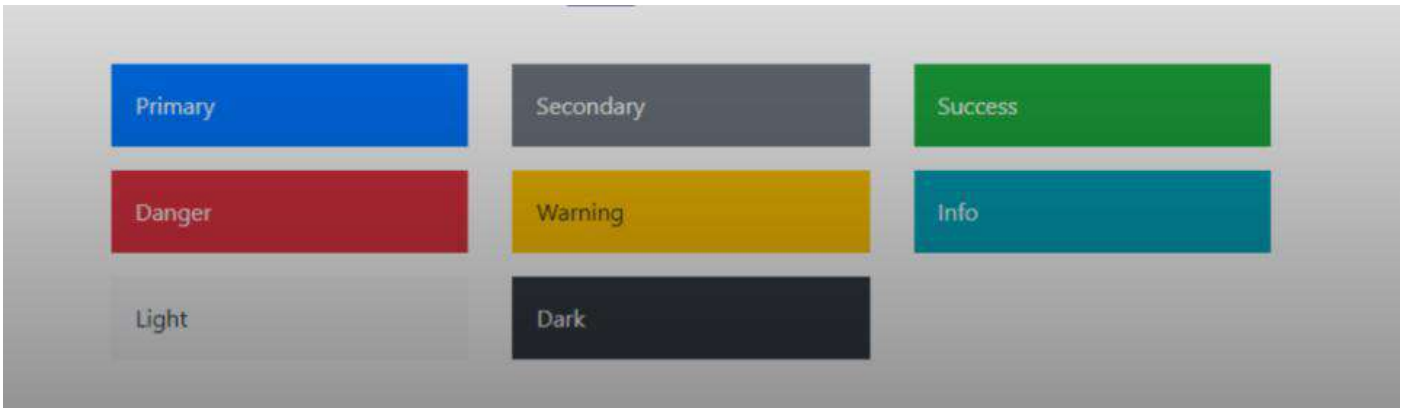
CLASS 7 : ADO.NET

- What is ado.net? Answer: Ado.Net is portion or a library or code of dot net which helps to communication with database. Entity framework is created based on ado.net. Entity framework use ado.net
- ADO.NET Active Data Objects for .NET
- DbParameter is part of the System.Data.Common namespace and is used to represent a parameter associated with a database command (such as a parameterized SQL query) for executing against a database.
- We install nuget System.Data.SqlClient nuget for database (we can install Microsoft.Data.SqlClient, but is lower feature than System.Data.SqlClient)
- We should use 'using' keyword in AdoNetUtility. This keyword help to dispose connection
- If we need to return single data then we can use int, string or other database, if we need to return multiple data then we need to use parameter or dataset
- If we use DbCommand instead of SqlCommand as return type in a signature (function) then if we return data for MySql or php or MSSQL, all will be support
- In an ASP.NET Core `Program.cs` file, the `WebHost.CreateDefaultBuilder(args)` method configures the web host with default settings, including web server configuration and content root location.

```
private DbCommand CreateCommand()
{
    //...
}
```

CLASS 9 : BOOTSTRAP AND ADMINLTE

- Bootstrap color name



CLASS 10 : SCSS

- What is SASS? Answer: Superset of CSS. We can use CSS + Additional feature (We can declare variable, nesting (css not support nesting), we can import file (when create file give _file when import do not give underscore video 1:32:20), mixing (like function but no return type) write function (must have return type), declare parameters)
- If we want to see SCSS file to auto CSS generated file then right click on SCSS file > Web Compiler
- Some example: more details https://sass-lang.com/documentation/at-rules/control/

```
// We can declare Variables
$btnColor : blue;

// Can write functions, called mixin (mixing like a function but no return type)
@mixin someStyle
{}
// for parametarize mixin
@mixin someStyle($color){
}
@include someStyle(red);



// Using default parameters
@mixin identifier($param_1: default_value, $param_2: default_value) {
property: $param_1;
property: $param_2;
}

// Function (Which have return type)
@function function_name($parameters) {
    // code block
    @return value;
}
.header{
    background-color: function_name($user_type);  // called function
}


// ..........................Sir Code.....................

// _logic.scss
@mixin someStyle($divColor, $buttonColor, $hoverColor) {
    div {
        background-color: $divColor;

        button {
            background-color: $buttonColor;

            &:hover {
                background-color: $hoverColor;
            }
        }
    }
}
```

```
// _color.scss
$div: yellow;
$button: purple;
$hover: green;


// demo.scss  (like program.cs, we just compile this file only)
@import "colors";
@import "logic";

@include someStyle($div, $button, $hover);  // here we call someStyle
```

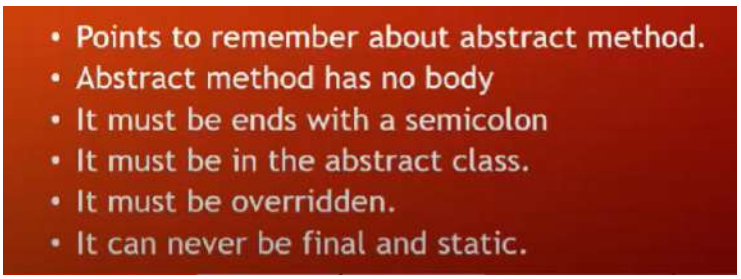
CLASS 11: SOLID

- মনে রাখা দরকার যে virtual এর মেথড থাকে + একে ওভাররাইড করা যায়, অন্যদিকে abstract এর মেথড থাকে না + একে ওভাররাইড করা যায়

- **Programming principle:** (if this 4 principles not exist in a programming language that was not OOPL)

1. **Abstract**:

   Abstract is a keyword in C# used to define abstract classes or abstract members, providing a blueprint for derived classes but cannot be instantiated on its own. Abstraction is the process of hiding the implementation details and showing only the functionality to the user. Example: ATM machine functionality, Messenger interface (এখানে শুধু ফাংশনালিটি দেখা যায়, মূল বিষইয়টা হাইড থাকে। এবস্টাক বানানোর জন্য ইন্টারফেস ব্যবহার করতে হবে। abstract void message(); এই আবস্টাক মেথডে কনো বডি থাকে না। আর কোনো এবস্টাক মেথড ডিক্লেয়ার করি তবে সেটা মাস্ট আবস্টাক ক্লাসে থাকতে হবে তবে এবস্টাক ক্লাসের নন এবস্টাক মেথডও থাকতে পারে (আনিসুল স্যারের জাভা থেকে নেওয়া))




```
using System;

// Abstract class
public abstract class Shape
{
    // Abstract method without implementation
    public abstract double CalculateArea();

    // Regular method with implementation
    public void Display()
    {
        Console.WriteLine("This is a shape.");
    }
}
```

```
// Concrete class inheriting from the abstract class
public class Circle : Shape
{
    // Fields specific to Circle
    private double radius;

    // Constructor
    public Circle(double radius)
    {
        this.radius = radius;
    }

    // Implementation of the abstract method
    public override double CalculateArea()
    {
        return Math.PI * radius * radius;
    }
}

class Program
{
    static void Main()
    {
        // Creating an instance of the concrete class
        Circle circle = new Circle(5.0);

        // Calling abstract method and regular method
        double area = circle.CalculateArea();
        Console.WriteLine($"Area of the circle: {area}");

        circle.Display();
    }
}
```

2. **Inheritance**:

   Inheritance is a fundamental OOP concept in C# where a class (derived or child class) inherits properties and behaviors from another class (base or parent class), promoting code reuse.

3. **Polymorphism**:

   Polymorphism is a feature in C# allowing objects of different types to be treated as objects of a common base type, enabling flexibility and extensibility through method overriding and interfaces.

4. **Encapsulation**:

   Encapsulation is an OOP principle in C# that involves bundling data (fields) and methods that operate on the data into a single unit (class), restricting direct access to the internal details and promoting modularity and information hiding. (কোনো একটি নির্দিষ্ট ক্লাসের মেডথ বা পুরো ক্লাসকেই বাইরের কোনো ক্লাস বা একই সলিউশনের অন্য কোনো প্রজেক্টে যাতে ইচ্ছামতো ব্যবহার করতে না পারে সেজন্য public, private, internal, protected ইত্যাদি সেট করে লিমিট করে দেওয়াই হচ্ছে এনক্যাপ্সুলেশন)

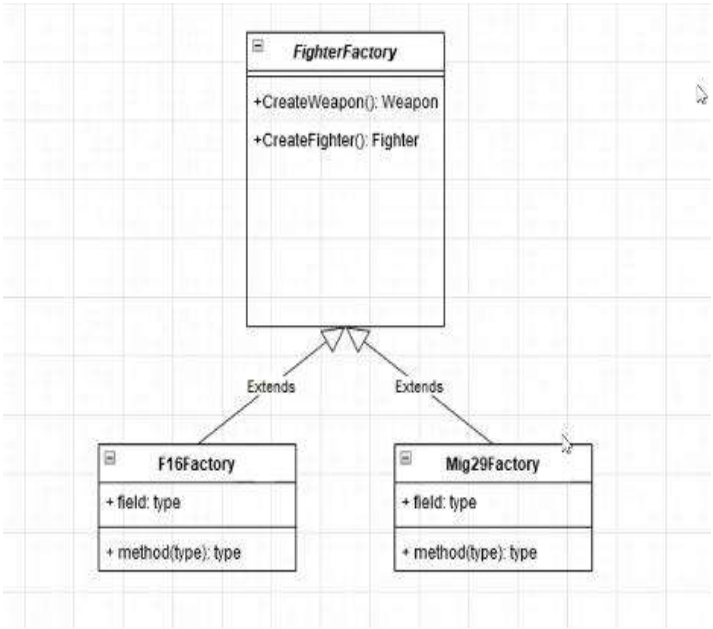
- Object Oriented Design Principle: এটা আমাদের কোডকে গুড মেনইটেবিলিটির জন্য লাগে।

CLASS 12 : PRINCIPLE & PATTERN

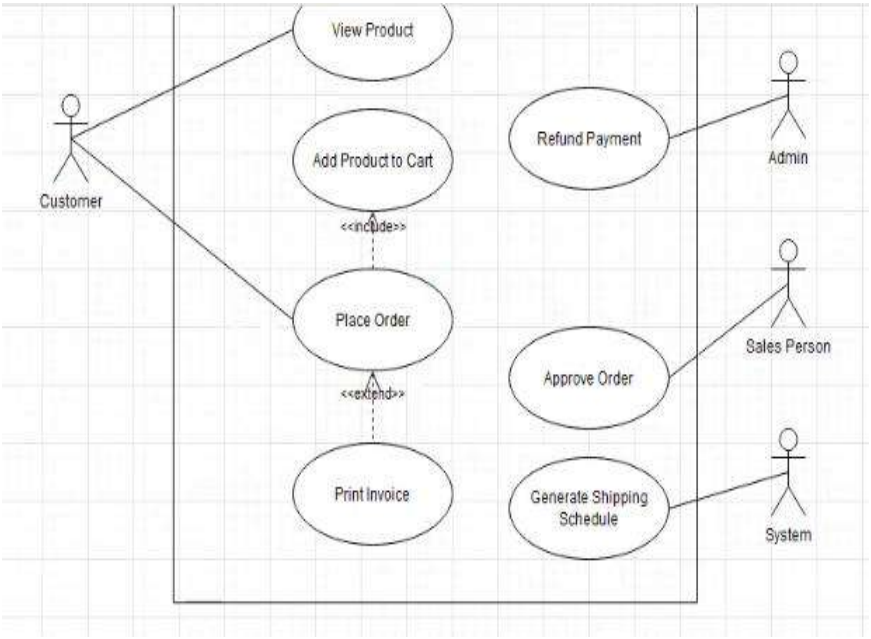
- We can't do dependancy injection easily with static (static is not good). We can call direct a method without creating instance

CLASS 13 AND 14: UML CLASS DIAGRAM, USE CASE DIAGRAM, CLASS DIAGRAM

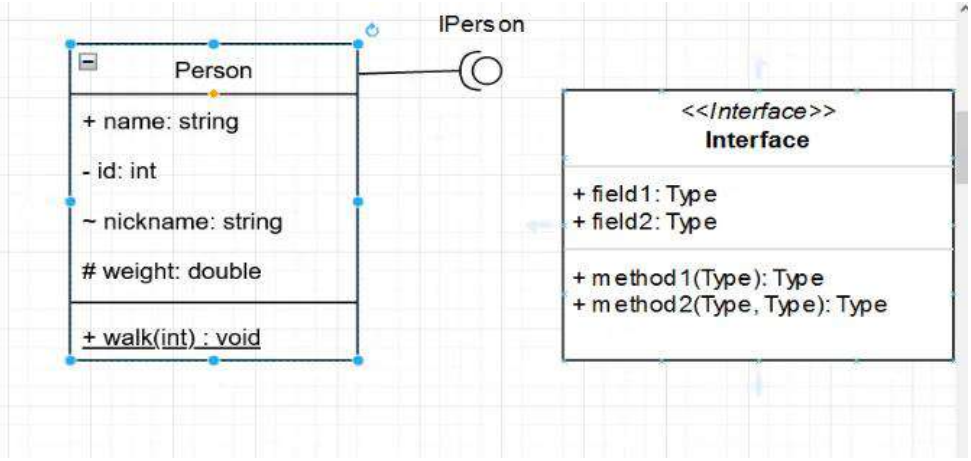
- A UML (Unified Modeling Language) Class Diagram is a visual representation of the structure and relationships of classes in a system, used for modeling and understanding software systems and their components. (Here extend arrow used for inheritance method)

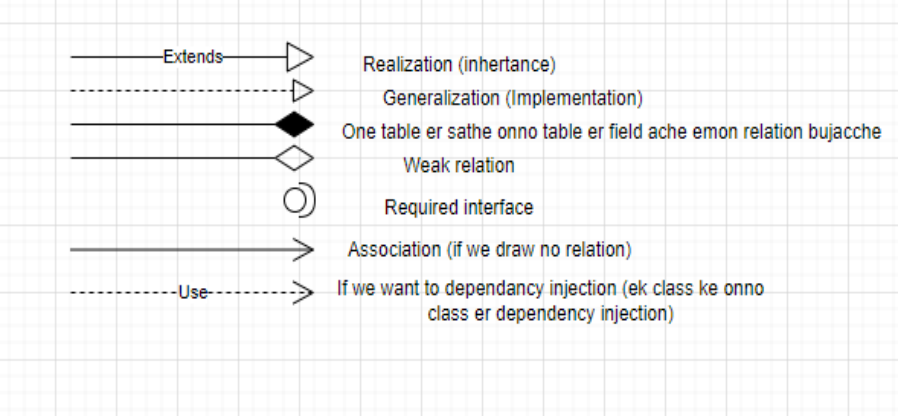


- Use case diagram mainly use for requirement analysis (Here include use for include operation (some operation can not be completed without another operation, in that case we use this include), extend used for when we need to do addtion feature (like I want to payment, I can do this using card or cash))
- In Use case diagram actor inherit actor (Customer and Bulk customer (paikari customer))

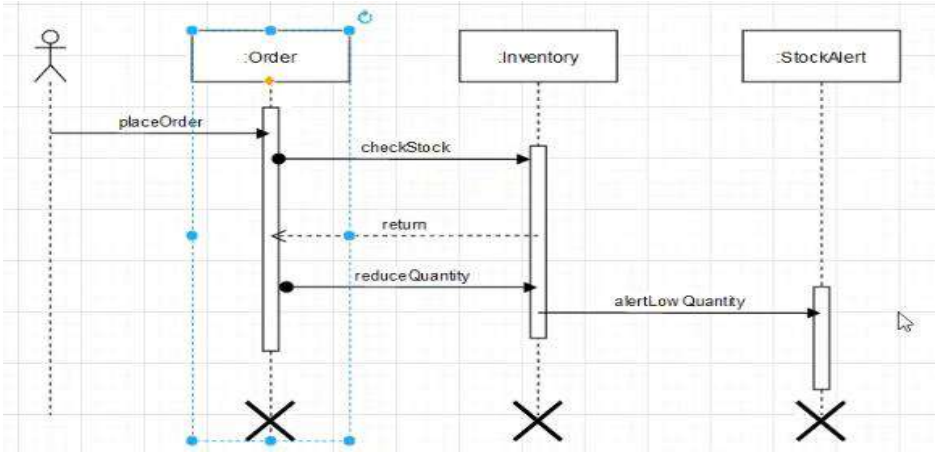


- Class diagram is a visual representation of the structure and relationships of classes in object-oriented programming, used to plan, document, and communicate the design of a software system (Here, + public, - private, # protected, ~ internal)
- In Class diagram o) symbol means required interface
- If in class diagram table name italic that means it is a abstract class
- If in class diagram table or method is static then we use underline
-





- Sequence diagram:



CLASS 15: CLASS DIAGRAM

- UML diagram's decrease day by day
- Class diagram and uml diagram is wastage
- But use case and activity diagram should use
- Here have lifeline, initialize (vertical box is initialize)
- UML diagram's decrease day by day

- Here sir discuss 5 Creational design pattern: Singleton design pattern, Prototype pattern, Builder pattern, Factory pattern, Abstract factory pattern
- Sir discuss 1 Enterprise design pattern: Repository pattern

CLASS 16: UNIT OF WORK

- Why use sql transaction into unit of work?
  Answer: Using SQL transactions within a unit of work is essential to ensure the consistency, reliability, and integrity of a database, allowing multiple related database operations to be treated as a single, atomic, and all-or-nothing operation, which is crucial in situations like complex database updates, where data must remain consistent even if an error occurs during the process. (If we use transaction feature (commit all at once, delete all at once) while unit of work then if some data missing while I insert into database then no data will be inserted)

- ORM – Object Relational Mapper (C# er object er sathe database er relation er mapping kore)
- Entity framwwork, Nhibernet are also ORM
- Aggregate root:

An aggregate root is a concept from domain-driven design (DDD), a software development methodology that focuses on creating a shared understanding of complex problem domains and on aligning software systems with those domains. In DDD, an aggregate is a group of related entities and value objects treated as a single unit. Among these, one entity is designated as the "aggregate root," and it is the only entry point to the aggregate from the outside world.

Here are some key characteristics of an aggregate root:

1. Encapsulation: The aggregate root is responsible for ensuring the consistency and integrity of the entire aggregate. It encapsulates the internal state and business rules of the aggregate, and all changes to the aggregate's state should go through the aggregate root.
2. Consistency Boundary: The aggregate root defines a boundary within which all invariants (business rules) must be maintained. The aggregate enforces these rules to ensure that the data within it is always in a valid and consistent state.
3. Identity: Every aggregate root has a globally unique identity that distinguishes it from other aggregates. This identity is used for referencing the aggregate in the application.
4. Transactions: Changes to the aggregate, including creating, modifying, or deleting entities and value objects within the aggregate, should be treated as a single transaction. This ensures that the aggregate is always in a consistent state.
5. Access Control: The aggregate root controls access to its internal components, and it may expose methods or operations that allow for making changes to its state while maintaining consistency.
6. Concurrency Control: The aggregate root is responsible for handling concurrency issues, such as preventing conflicting updates to the aggregate by multiple users.

In a software application, aggregates are often used to model complex, interrelated data structures or business processes. By designating an aggregate root, you create a clear entry point for interacting with and maintaining the aggregate's integrity. This helps in managing and organizing the complexity of the domain model in a more understandable and maintainable way, as well as ensuring data consistency in multi-user or distributed systems.

CLASS 16: EF

- Data Anotation – [key], [Required]
- FluentAPI Approch–

```
modelBuilder.Entity<Topic>().ToTable("Course");
```

- Below code is use for relationship not directly generated table. It make relation and then migration generated table and others

```
            modelBuilder.Entity<CourseStudent>()
                .HasOne(x => x.Course)
                .WithMany(y => y.Students)
                .HasForeignKey(z => z.CourseId);

            modelBuilder.Entity<CourseStudent>()
                .HasOne(x => x.Student)
                .WithMany(y => y.Courses)
                .HasForeignKey(z => z.StudentId);

            base.OnModelCreating(modelBuilder);
        public DbSet<Course> Courses { get; set; }
        public DbSet<Student> Students { get; set; }
```

CLASS 17: REPOSITORY PATTERN + UNIT OF WORK + EF

- Why we use repository pattern: We use repo pattern (add update delete) to use database easily (remove complexity)
- Why we use UoW: Here we implement Commit() method where have repositories add update delete method, when I call commit it add all data or all delete all data at a time

- Why we use persistence layer? Answer: We don't want to loss our data while we close our application. So we should use persistence layer to connect with database

CLASS 18: CLEAN ARCH.

- Why use web, persistence, domain, application,
- RenderAction will call an action method of the current controller and render a result inline. RenderPartial will render the specified view inline without calling any action method.

```
@{
    Html.RenderAction("Add");
    Html.RenderPartial("Add");
}
```

- We have two class, ICourseRepo.cs and CourseRepo.cs. Among them CourseRepo.cs implementation exist on which path we bind (Register) on that project module
- Normally database related material exist on Infrastructure layer, but sir keep that on persistence

CLASS 21: CLEAN ARCH.

- Castrol server use asp.net by default. If we donot use IIS then we can see that

CLASS 23: CLEAN ARCH.

- In the Domain Layer, we keep such as file which is common for company
- Web project is lower level
- Application and Domain's implementation either exist in persistence or Infrastructure
- Clean arch. comes to base on Onion arch.

CLASS 24: CLEAN ARCH.

- We can use fluentValidation nuget to validate instead of [Required] in Model.cs
- Sometimes we face a little problem to generate razor view page, to avoid such as problem we need to install Microsoft.VisualStudio.Web.CodeGeneration.Utils
- We must use ValidateAntiForgeryToken when attribute HttpPost, if we work js or ajax whatwever
  `[HttpPost, ValidateAntiForgeryToken]`
- If we want to add jquery and client validation then add a _ValidationScriptsPartial.cshtml in shared folder

CLASS 25-27: CLEAN ARCH.

- From view bag data never deleted automatically that's why we need to delete data using temp data.
- Here in _ResponsePartial.cshtml first we peek if there any message exixt or not, if exist then we use Tempdata to get message, as a result after printing result it will be delete message automatically (after reload or we if go another page then tempdata will be removed, if tempdata not read once then it never removed (like session)).

```
@if(TempData.Peek<ResponseModel>("ResponseMessage") != null)
{
    var response = TempData.Get<ResponseModel>("ResponseMessage");
    <div class="alert alert-@(response.Type.ToString().ToLower())" role="alert">
        @response.Message
    </div>
}
```

- View Bag vs ViewData vs TempData vs Session

Aspect	ViewBag	ViewData	TempData	Session
Type	Dynamic property bag (dynamic)	Dictionary<string, object>	Dictionary<string, object>	Dictionary<string, object>
Strongly Typed	Not strongly typed.	Not strongly typed.	Not strongly typed.	Not strongly typed.
Data Sharing	Shares data between controller and view within a single request.	Shares data between controller and view within a single request.	Shares data between controller and view for a single request.	Shares data between controller and view across multiple requests.
Scope	Per request (short-lived).	Per request (short-lived).	Per request (short-lived).	Across sessions (long-lived).
Performance	Slightly faster due to dynamic nature.	Slightly slower due to casting to access data.	Slightly slower due to casting to access data.	Can be slower if using an out-of-process session storage.

Syntax	ViewBag.PropertyName = value;	ViewData["PropertyName"] = value;	TempData["PropertyName"] = value;	Session["PropertyName"] = value;
Error Handling	No compile-time error checking.	No compile-time error checking.	No compile-time error checking.	No compile-time error checking.
Data Retention	No data type information; may lead to runtime errors if not used correctly.	Stores data with data type; can result in runtime errors if not used incorrectly.	Stores data with data type; can result in runtime errors if not used incorrectly.	Stores data with data type; can result in runtime errors if not used incorrectly.
Preferred Usage	When dynamic, loosely typed data sharing is needed within a single request.	When data sharing with a dictionary is preferred within a single request.	When passing data between actions within a single request.	For storing user-specific data across multiple requests.

- **ViewBag:** VewBag never pass data one controller to another controller. This will pass data same controller to same view page.
- **ViewData:** ViewData never pass data one controller to another controller. This will pass data same controller to same view page.
- **TempData:** TempData can pass data one controller to another controller. This a special variable for dot net. Which is exist on controller. If I not read tempdata message, then message never removed.
- **Session:** Session can pass data one controller to another controller. If we not delete session data manually then it will never deleted.

- If we need more optimize then use decimal if not need to optimization then use double

- Lazy loading vs Eager loading:

Aspect	**Lazy Loading**	**Eager Loading**
Loading Strategy	Data is loaded on-demand, typically when accessing a navigation property.	Data is loaded upfront along with the main entity, using techniques like .Include().
Query Execution	Generates additional queries to the database as related data is accessed.	Generates a single query with JOINs or additional queries for related data, depending on how it's configured.
Control	Provides finer control over which related data is loaded, reducing over-fetching.	Loads all specified related data, which may lead to over-fetching if not used carefully.
Performance	Reduces the amount of data fetched initially, potentially improving performance when only a subset of related data is needed.	Fetches all related data at once, which can be more efficient when you know you'll need most of it.
Usage Scenario	Useful when you need to minimize initial data transfer or load related data conditionally.	Suitable when you know you'll use a significant portion of the related data, reducing the need for additional round-trips to the database.
Potential Issues	May lead to the N+1 query problem, resulting in multiple queries for related data in a loop.	Reduces the risk of the N+1 query problem but can fetch more data than needed, leading to over-fetching.

- **Lazy Loading:** In dotnet, by default lazy loading turned off. Suppose we have two data table name, Course and Topic (Both are in relation). If I want to search data for Course table then lazy loading only show me Course table data, though they are relations with Topic table. If I search CourseTable data then lazy loading give us Course and Table both data. Lazy loading has a big problem that is, if we search data where have thousand of

data and datatable then lazy loading send thousand of query request to fetch data (which is horrible, but lazy loading is easy to use). Example: Social media, Ecommerce

- **Eager Loading:** On ther other hand If I call a table then all table data comes at once with eager loding. (This is just opposite of lazy loading). Eager loading fetch all data at once and one query as a result there are no need to send thousand of database query request. But problem is that we no need whole database data at a time what we get from eager loading. Example: Blog, Inventory

- If there have more than 5000 lines code then rid it – Martin Fowler

- If we want to do apply autoMapper then we need to install AutoMapperDependencyInjection nuget then, we setup in program.cs and Create a class name WebProfile.cs

```
builder.Services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies()); // Current project er assembly theke WebProfile.cs khuje ber korbe (like autofac WebModule.cs)
```

```
public class WebProfile : Profile
{
    public WebProfile()
    {
        CreateMap<CourseUpdateModel, Course>()
            .ReverseMap(); //if we use reverseMap this will be two way (other wise one way). CourseUpdateModel theke course e copy kora jabe and Course theke CourseUpdateModel eo copy kora jabe
    }
}
```

- Something in automapper give false result like where we need value automapper gives us null like that (This is not caught into simple case). We can handle this type of problem.

```
// before appling automapper
internal void Load(Guid id)
{
    Course course = _courseService.GetCourse(id);
    Id = course.Id;
    Name = course.Name;
    Fees = course.Fees;
}

// after appling automapper
internal void Load(Guid id)
{
    Course course = _courseService.GetCourse(id);
    if(course != null)
    {
        _mapper.Map(course, this); //here 'this' is instance of model
    }
}
```

- Bonus: Microservice vs monolithic project

CLEAN ARCH. EXTRA

Just Read comment all of the picture:

API :

```
namespace FirstDemo.API.Controllers
{
    [ApiController]
    [Route("v3/[controller]")]
    [EnableCors("AllowSites")]
    1 reference
    public class EnrollmentController : ControllerBase
    {
        // indicates that the class relies on dependency injection and is likely using an IoC container to manage the lifetime and resolution
        //          of its dependencies. This promotes a more modular, testable, and maintainable codebase.
        private readonly ILifetimeScope _scope; // Life cycle by Autofac

        // Ekhane CourseController ke ms er Logger Injection kora hoyeche karon amra eitar executional log dekhte chai, ei injection ta kind of
        //        generics use kore kora hoyeche
        // Ar finally dependency injection o kora hoyeche ekhane
        private readonly ILogger<CourseController> _logger; // Logger by microsoft

        0 references
        public EnrollmentController(ILogger<CourseController> logger,
            ILifetimeScope scope)
        {
            _logger = logger;
            _scope = scope;
        }
```

```
[HttpPost()]
// Ekhane formbody na dile 400 Bad request ashe postman e data insert ba update e
// Ekhane view page theke data anar jonno model binding use kora hoyeche (jodi aro onek doroner binding ache like FormCollection, Parameterize
0 references
public IActionResult Post([FromBody] CourseModel model)
{
    try
    {
        model.ResolveDependency(_scope);
        model.CreateCourse();

        return Ok();
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Couldn't delete course");
        return BadRequest();
    }
}
```

(Note: Remember Postman methods Get, Put (Update), Post (Create), Delete)

- Why use AutoMapper: AutoMapper simplifies the process of mapping data between objects in .NET applications, reducing boilerplate code, improving development speed, and supporting convention-based mapping. It enhances maintainability, testability, and decouples layers in your application.
- Example without using of AutoMapper:

```
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
}

public class PersonDto
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
```

```
//'StringValues' is used instead of 'string' in ASP.NET Core to handle scenarios where a parameter can have multiple
//values(e.g., in query strings or form data) and to avoid null checks by representing an empty collection when no value
//is provided.
var StringValues enrollmentDateFrom = Request.Query["SearchItem[EnrollmentDateFrom]"];
var StringValues enrollmentDateTo = Request.Query["SearchItem[EnrollmentDateTo]"];
```

```
    public int Age { get; set; }
}

// Without AutoMapper
public class ManualMappingExample
{
    public static PersonDto MapPersonToDto(Person person)
    {
        return new PersonDto
        {
            FirstName = person.FirstName,
            LastName = person.LastName,
            Age = person.Age
        };
    }

    public static Person MapDtoToPerson(PersonDto dto)
    {
        return new Person
        {
            FirstName = dto.FirstName,
            LastName = dto.LastName,
            Age = dto.Age
        };
    }
}
```

- Example with using of AutoMapper:

```
using AutoMapper;

public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
}

public class PersonDto
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
}

// With AutoMapper
public class AutoMapperExample
{
    private readonly IMapper _mapper;

    public AutoMapperExample()
    {
```

```
        // Configure AutoMapper (usually done during application startup)
        var configuration = new MapperConfiguration(cfg =>
        {
            cfg.CreateMap<Person, PersonDto>();
            cfg.CreateMap<PersonDto, Person>();
        });

        _mapper = configuration.CreateMapper();
    }

    public PersonDto MapPersonToDto(Person person)
    {
        return _mapper.Map<PersonDto>(person);
    }

    public Person MapDtoToPerson(PersonDto dto)
    {
        return _mapper.Map<Person>(dto);
    }
}
```

- Using a service in ASP.NET is a best practice that provides several benefits, including separation of concerns, reusability, testability, maintainability, and flexibility. It contributes to a clean and organized codebase, making it easier to develop, maintain, and extend your application.
- JWT Token generate for postman token genetation

Worker:

```
{
    Log.Information("Application Starting up");

    IHost host = Host.CreateDefaultBuilder(args)
        .UseWindowsService() //this is for windows, if i want to run this background process in linux then use daemon (use dll in daemon) here
        .UseServiceProviderFactory(new AutofacServiceProviderFactory())
        .UseSerilog()
        .ConfigureContainer<ContainerBuilder>(ContainerBuilder builder =>
        {
            builder.RegisterModule(new WorkerModule());
        })
    .ConfigureServices(IServiceCollection services =>
    {
        //This line registers the Worker class as a hosted service in the dependency injection container, allowing it to be automatically
        //      managed by the ASP.NET Core host.
            services.AddHostedService<Worker>();
    })
        .Build();
    //'await host.RunAsync();' is used to asynchronously start and run an ASP.NET Core host, allowing the application to listen for
    //      incoming requests and handle them.
    await host.RunAsync();
}
catch (Exception ex)
```

```
5 references
public class Worker : BackgroundService
{
    private readonly ILogger<Worker> _logger;

    0 references
    public Worker(ILogger<Worker> logger)
    {
        _logger = logger;
    }
    // when we stop service from task manager then CancellationToken get stop value
    0 references
    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            _logger.LogInformation("Worker running at: {time}", DateTimeOffset.Now);
            await Task.Delay(1000, stoppingToken); // It write log after every one second
        }
    }
}
```

Runtime template (.tt) will auto convert C# files

Persistence :

```
    {
        services.AddIdentity<ApplicationUser, ApplicationRole>() // Configure Identity services with custom user and role types
        .AddEntityFrameworkStores<ApplicationDbContext>() // Use Entity Framework as the storage provider for user and role data
        .AddUserManager<ApplicationUserManager>() // Add a custom user manager for ApplicationUser
        .AddRoleManager<ApplicationRoleManager>() // Add a custom role manager for ApplicationRole
        .AddSignInManager<ApplicationSignInManager>() // Add a custom sign-in manager for user authentication
        .AddDefaultTokenProviders(); // Add default token providers for features like password reset, email confirmation, etc.

        services.Configure<IdentityOptions>(IdentityOptions options =>
        {
            // Password settings.
            options.Password.RequireDigit = true;
            options.Password.RequireLowercase = false;
            options.Password.RequireNonAlphanumeric = false;
            options.Password.RequireUppercase = false;
            options.Password.RequiredLength = 6;
            options.Password.RequiredUniqueChars = 0;

            // Lockout settings. (5 barer beshi bhul dile baad)
            options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
            options.Lockout.MaxFailedAccessAttempts = 5;
            options.Lockout.AllowedForNewUsers = true;

            // User settings.
            options.User.AllowedUserNameCharacters =
            "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-._@+";
            options.User.RequireUniqueEmail = true;
        });
        //'services.AddRazorPages()' configures and adds Razor Pages as a feature to your ASP.NET Core web application, enabling you to create dynamic web pages.
        services.AddRazorPages();
    }
}
```

```
namespace FirstDemo.Persistence.Features.Training.Repositories
{
    // we write specific repo for specific service. We wantn't not add repositories code in the service class that's why we keep here
    2 references
    public class CourseRepository : Repository<Course, Guid>, ICourseRepository
    {
        // Here we work autofac binding against interface thats why if I do only ApplicationDbContext instead of IApplicationDbContext then works but give some
        // IApplicationDbContext er 'context' er value pabo hocche base er context theke (ekhane 'DbContext' ke typecast kora hoyeche,
        //      karon base context er parent class DbContext)
        0 references
        public CourseRepository(IApplicationDbContext context) : base((DbContext)context)
        {
        }

        public async Task<(IList<Course> records, int total, int totalDisplay)>
            2 references
            GetTableDataAsync(Expression<Func<Course, bool>> expression,
            string orderBy, int pageIndex, int pageSize)
        {
            return await GetDynamicAsync(expression, orderBy, null,
                pageIndex, pageSize, true);
        }
```

```
{
    //IdentityDbContext is using instead of DbContext here to use Identity framework
    11 references
    public class ApplicationDbContext : IdentityDbContext<ApplicationUser,
        ApplicationRole, Guid,
        ApplicationUserClaim, ApplicationUserRole,
        ApplicationUserLogin, ApplicationRoleClaim,
        ApplicationUserToken>,
        IApplicationDbContext
    {
        private readonly string _connectionString;
        private readonly string _migrationAssembly; // all file, class, type khuje ber kore

        0 references
        public ApplicationDbContext(string connectionString, string migrationAssembly)
        {
            _connectionString = connectionString;
            _migrationAssembly = migrationAssembly;
        }

        //'OnConfiguring' is used to configure the database connection and other options for the DbContext in Entity Framework Core.
        0 references
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            if (!optionsBuilder.IsConfigured)
            {
                optionsBuilder.UseSqlServer(_connectionString,
                    SqlServerDbContextOptionsBuilder x => x.MigrationsAssembly(_migrationAssembly));
            }

            base.OnConfiguring(optionsBuilder);
        }

        // 'OnModelCreating' is used to configure the model (entity relationships, constraints, etc.) in Entity Framework, providing customization of the
        //      database schema generation and mapping between the database and the application's domain model.
        0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Student>().HasData(StudentSeed.Students);
```

```
public class Repository<TEntity, TKey> : IRepository<TEntity, TKey> where TKey : IComparable where TEntity : class, IEntity<TKey>
{
    7 references
    protected DbContext _dbContext { get; set; }
    33 references
    protected DbSet<TEntity> _dbSet { get; set; }
    2 references
    protected int CommandTimeout { get; set; }
    1 reference
    protected Repository(DbContext dbContext)
    {
        ≤ 1ms elapsed
        CommandTimeout = 300; // Set a default command timeout of 300 seconds
        _dbContext = dbContext; // Store the provided DbContext
        _dbSet = _dbContext.Set<TEntity>(); // Set the DbSet property to the DbSet of the specified entity type (TEntity)
    }
```

- `DbSet<Course>` represents a collection of entities of type `Course` in Entity Framework, providing a way to query and interact with the corresponding database table.

CLASS 28: SECURITY

- Async: we use async in file, db context and networking and etc (mainly when we add async before some of method then all async method start at a time. When we need start such as group of mehod which are start at a time and more faster then we use async)
- In MVC sometime we wrongly say viewmodel this is not correct but not worng also (viewmodel keyword come from MVVC). We should say razor view instead view model
- DTO – Data Transfer Object: We use dto to transfer data to one layer to another layer. We already use dto with store procedure

- SQL injection: We must use parameterized query to prevent sql injection. Enitity framework is by default parameterized query which is prevent sql injection.
- We should keep server side validation beside Javascript client side validation. Otherwise if hacker want then they turn of js from browser and then access.
- HTML Anti-Forgery Tokens is a fundamental security measure to protect web applications from CSRF (Cross-Site Request Forgery.) attacks and maintain data integrity. It is a recommended practice in web development, especially for actions that involve sensitive operations or data modifications.

  We add in crontroller and razor view's html input page or form like:

```
        [HttpPost, ValidateAntiForgeryToken]
        public IActionResult Create(CourseCreateModel model)
```

```
@Html.AntiForgreyToken()
```

which generated hash value in browser as a result if hacker want to temper html input which hash value will not match with serverside hash. So tempering html not work. Here is server side hash (here full form page is hased),

This is the main reason to use `ValidateAntiForgeryToken.`

- Script injection: If I take input something and want to show it in browser then we infected like this. If we use `HttpUtility.HtmlEncode` then we can avoid

```
return new
{
    recordsTotal = data.total,
    recordsFiltered = data.totalDisplay,
    data = (from record in data.records
            select new string[]
            {
                    HttpUtility.HtmlEncode(record.Name),
                    record.Fees.ToString(),
                    record.Id.ToString()
            }
        ).ToArray()
};
```

- DDoS (Distributed Denial of Service) : If we block some ip from where we face DDoS attact then we can prevent DDoS attact
- Http – pc to server connection will be happen will plain text (anyone can view site data, login info)
- One way encription- If i encrypt once we can not decript
- Two way encription- We ca encrypt and also decrypt

CLASS 29-33: IDENTITY FRAMEWORK AND AUTHENTICAITON AND AUTHORIZATION

- Markup language vs Markdown language

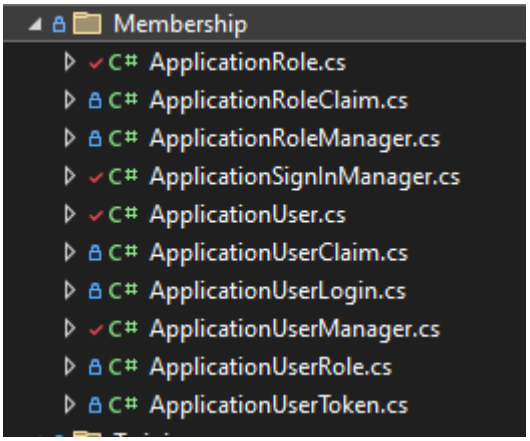
Aspect	Markup Language	Markdown Language

Syntax Complexity	Generally more complex, using tags, attributes, and often requiring a deeper understanding of the language.	Simpler and more lightweight, using plain text with simple formatting rules and conventions.
Learning Curve	May have a steeper learning curve, particularly for beginners and non-technical users.	Easier for most users to pick up quickly due to its simplicity.
Readability	Markup code can be less readable because of the presence of tags, attributes, and potentially complex structures.	Markdown is more human-readable because it uses plain text with minimal formatting elements.
Widespread Use	Commonly used in web development (HTML), document formatting (e.g., LaTeX), and various programming languages (e.g., XML).	Widely used for creating documentation, README files, and online communication platforms (e.g., GitHub, Reddit).
Extensibility	Markup languages are highly extensible and can be customized for specific needs.	Markdown is less extensible by design but can be extended using custom parsers or converters.
Use Cases	Suitable for creating complex web pages, structured documents, and data representation.	Ideal for simple text formatting, documentation, and quick note-taking.
Examples	HTML, XML, LaTeX, JSON, RTF	Markdown, Markdown-based languages (e.g., CommonMark), reStructuredText
Integration	May require more effort to integrate into certain systems and applications.	Easily integrated into various platforms, services, and tools due to its plain text nature.

- Markup language: Use Tiny.Cloud (wiz-ee-wig উইজ-ই-উইগ)
- Markdown language: We can use this on Readme.md on github

- In ApplicationDbContext we should rename DbContext to IdentityDbContext to access feature of database

- In Persistence layer we use create 10 class like Role, signIn, User class



- Authentication: Authentication check is password, username valid or not. In below code example though written Authorize attribute but it works like Authentication

```
[Area("Admin"), Authorize]
```

- Authorization: Authorization check is user have permission or not. There are tree type of authorization.
    1. Role Base : If we set Admin or Customer role for specific task then they did the job (We can set multiple role for specific work)

```
[Area("Admin"), Authorize(Roles = "Admin")]
```

    2. Policy Base: If I want to give permission someone as HR department and Admin controll

```
[Authorize(Policy = "StudentView")]
```

```
options.AddPolicy("StudentView", policy =>
{
    policy.RequireAuthenticatedUser();
    policy.RequireAssertion(context =>
    {
        return context.User.IsInRole("Admin") ||
               context.User.IsInRole("Client") ||
               context.User.IsInRole("Employee") ||
               context.User.IsInRole("User");
    });
});
```

3. Claim Base: If I want to set permission for every specific work like view, add, delete

```
[Authorize(Policy = "StudentDeleteRequirementPolicy")]
```

```
options.AddPolicy("StudentDelete", policy =>            //StudentDelete is policy name
{
    policy.RequireAuthenticatedUser();
    policy.RequireClaim("StudentDeleteClaim", "true");  //StudentDeleteClaim is claim name
});
options.AddPolicy("StudentDeleteRequirementPolicy", policy =>
{
    policy.RequireAuthenticatedUser();
    policy.Requirements.Add(new StudentDeleteRequirement());
});

options.AddPolicy("StudentMarksheetView", policy =>           // Policy name
{
    policy.RequireAuthenticatedUser();
    policy.RequireClaim("StudentMarksheetViewClaim", "true");  // Claim name
});
options.AddPolicy("StudentMarksheetViewRequirementPolicy", policy =>
{
    policy.RequireAuthenticatedUser();
    policy.Requirements.Add(new StudentMarksheetViewRequirement());
});
```

- Session is exist on server side and cookies is exist in browser

```
var builder = WebApplication.CreateBuilder(args);

// Serilog (When we use serilog then serilog replace microsoft logger thats why we can use
ILogger with serilog also
// This 5 line of code collected from Serilog documentation
builder.Host.UseSerilog((ctx, lc) => lc
    .MinimumLevel.Debug()
    .MinimumLevel.Override("Microsoft", LogEventLevel.Warning)
    .Enrich.FromLogContext()
```

```
    .ReadFrom.Configuration(builder.Configuration));

// Add services to the container.

try
{
    var connectionString = builder.Configuration.GetConnectionString("DefaultConnection")
?? throw new InvalidOperationException("Connection string 'DefaultConnection' not
found.");
    // This line added later (not come with initialization).
    // This line gives FirstDemo.API, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null result while debug point set on this line.
    // `Assembly.GetExecutingAssembly()` returns the assembly containing the code
currently being executed.
    var migrationAssembly = Assembly.GetExecutingAssembly().FullName;

    builder.Host.UseServiceProviderFactory(new AutofacServiceProviderFactory());
    builder.Host.ConfigureContainer<ContainerBuilder>(containerBuilder =>
    {
        containerBuilder.RegisterModule(new ApplicationModule());
        containerBuilder.RegisterModule(new InfrastructureModule());
        containerBuilder.RegisterModule(new PersistenceModule(connectionString,
            migrationAssembly));
        containerBuilder.RegisterModule(new ApiModule());
    });

    builder.Services.AddDatabaseDeveloperPageExceptionFilter();
    builder.Services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());
    builder.Services.AddIdentity();

    //JWT Token generate for postman token genetation
    builder.Services.AddAuthentication()
        .AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, x =>
        {
            x.RequireHttpsMetadata = false;
            x.SaveToken = true;
            x.TokenValidationParameters = new TokenValidationParameters
            {
                ValidateIssuerSigningKey = true,
                IssuerSigningKey = new
SymmetricSecurityKey(Encoding.ASCII.GetBytes(builder.Configuration["Jwt:Key"])),
                ValidateIssuer = true,
                ValidateAudience = true,
                ValidIssuer = builder.Configuration["Jwt:Issuer"],
                ValidAudience = builder.Configuration["Jwt:Audience"],
            };
        });


    // ekhane clear add na korle token dhore rakhe authorization e tai advance search kaj
kore na
    builder.Services.AddAuthorization(options =>
    {
```

```
        options.AddPolicy("CourseViewRequirementPolicy", policy =>
        {
            policy.AuthenticationSchemes.Clear();
            policy.AuthenticationSchemes.Add(JwtBearerDefaults.AuthenticationScheme);
            policy.RequireAuthenticatedUser();
            policy.Requirements.Add(new CourseViewRequirement());
        });
    });

    // Here use cors to give permission of communication between two project
    builder.Services.AddCors(options =>
    {
        options.AddPolicy("AllowSites",
            builder =>
            {
                //builder.AllowAnyOrigin()
                //    .AllowAnyMethod()
                //    .AllowAnyHeader();

                builder.WithOrigins("http://localhost:4200", "https://localhost:7307",
"https://localhost:9510")
                    .AllowAnyMethod()
                    .AllowAnyHeader();
            });
    });

    // Using AddSingleton for the registration of an IAuthorizationHandler like
CourseViewRequirementHandler indicates that only one instance
    //      of CourseViewRequirementHandler will be created and shared throughout the
entire lifetime of the application.
    builder.Services.AddSingleton<IAuthorizationHandler, CourseViewRequirementHandler>();

    builder.Services.AddControllers(); // Add controllers to the services collection for
MVC and API handling
    builder.Services.AddEndpointsApiExplorer(); // Add API Explorer to generate
OpenAPI/Swagger specifications
    builder.Services.AddSwaggerGen(); // Add SwaggerGen to generate OpenAPI/Swagger
documents for API documentation

    var app = builder.Build();

    Log.Information("Application Starting...");

    // Configure the HTTP request pipeline.
    if (app.Environment.IsDevelopment())
    {
        app.UseSwagger();
        app.UseSwaggerUI();
    }

    app.UseCors(); // Enable Cross-Origin Resource Sharing (CORS) for handling requests
from different origins
    app.UseHttpsRedirection(); // Redirect HTTP requests to HTTPS for secure communication
```

```
    app.UseAuthorization(); // Authorize and validate incoming requests based on policies
and roles
    app.MapControllers(); // Map and handle HTTP requests to controllers
    app.Run(); // Execute the final middleware to end the request pipeline
}
catch (Exception ex)
{
    Log.Fatal(ex, "Application start-up failed");
}
finally
{
    //we flush and close logger so that no cache store here (if we flush then try-catch
logger msg removed from logger)
    Log.CloseAndFlush();
}
```

- DTOs (Data Transfer Objects) are used to encapsulate and transfer data between different layers of an application, promoting loose coupling and facilitating data exchange.

CLASS 34-37: WEB API

- There are two type of service are available: i. Windows service, ii. Web service

- Windows Service:
- Web Service: We should keep web service like web application in a server. Web service is not for human directly. Web service communicate with another application. Example: WebAPI and Angular (We keep angular project outside of our project, this is a frontend framework) will be communicate if they need (web service communicate with json, xml etc format). Web service is a generalized term.

- API (Application Programming Interface): Interface provide guideline to us. API can fetch data from one application to another application (Like external login)
- WebAPI: Microsoft announce their web api name is Web API (Why ms give this generalized name we don't know. Already web api is subset of webservice. So here if we create web api here something it maybe webapi and sometimes it will be service but we alltime say ms given name WebAPI)
- Web Application or Website: Human use this. Here have many content like login, registransion and many type UI to interaction with Human

- যদি ইউজার ইন্টারফেস থাকে, ইউজারের সাথে Human Interaction করে তবে সেটা ওয়েব এপ্লিকেশন বা ওয়েব সাইট
- আর যদি মেশিন টু মেশিন communication হয়, যেখানে Human Interaction দরকার পড়েনা সেটা হচ্ছে web service
- Static website is never will be web application.
- Web Application maybe dynamic application

- OpenAPI is a protocol of API creation (but we not apply this in our University project)

- Minimal API: This is mainly use microservice or other type of project (where we need to reduce load where we want make more lightweight thats why we remove controller and then we can apply route and say what will be method (like lambda expression)
- A monolithic project is a software application that is developed as a single, self-contained unit, typically with all components tightly integrated into a single codebase and deployed as a single executable.
- Restful is a convension (As sir's opinion. But someone say it architecture or Protocol or Framework)

- For this two middleware, Swagger install at initilized

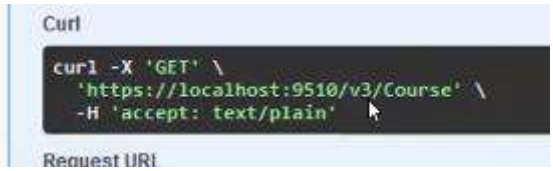
```
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

This create default documentation of Swagger

```
builder.Services.AddSwaggerGen();
```

- Put used for update, Post used for create, Delete used for delete and Get used for get
- WebSocket is a communication protocol that provides full-duplex, bidirectional communication channels over a single TCP connection, often used for real-time web applications, such as chat, online gaming, and live updates on websites.
- Our connection between database and DbContext is used TCP connection (http connection is not remembering anything after one transaction but TCP can remember everything.)
- Hard delete is permanently delete. Soft delete is not permanently delete but use will see delete.
- Http method

- Here Curl software is to call data



- When we send get, delete, post continously request then we need to set jwt token base authorization for specific user, so that not everyone cannot send request

CLASS 37: WORKER SERVICE

- Worker service: A worker service is a long-running background service that performs tasks asynchronously and independently of user interaction.
- If I want to make a reliable software then we need to work with various part of module like video processing, mail sending, image cropping

- When we stop service from task manager then CancellationToken get stop value

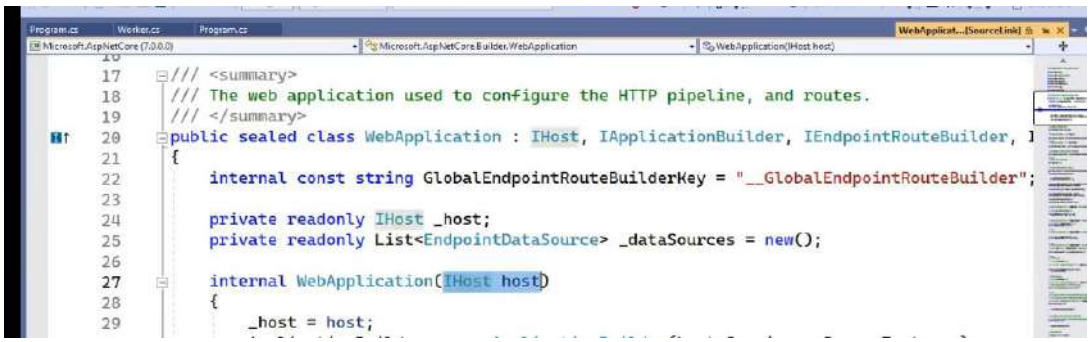
```
0 references
protected override async Task ExecuteAsync(CancellationToken stoppingToken)
{
```

- IHost Come from IDisposal (IHost located in program.cs)

```
public interface IHost : IDisposable
{}
```

```
IHost host = Host.CreateDefaultBuilder(args)
```

- Even IHost use on Web Application also (we can find this in demo.web's program.cs)



- IHost is used in ASP.NET to provide a unified way to configure and manage the lifetime of an application, including starting, stopping, and running background tasks.

- Interfaces are a valuable tool for developers who are writing ASP.NET applications. They can help to improve the quality, maintainability, and reusability of code.

Interface	Description	Usage
IDisposable	An interface that indicates that an object can be safely disposed of.	Used to release resources that are no longer needed, which can help to improve the performance of an application.
IEnumerator	An interface that allows an object to be iterated over one item at a time.	Used to iterate over the items in a collection, such as an array or a list.
IComparable	An interface that allows an object to be compared to another object of the same type.	Used to sort a collection of objects.
ICloneable	An interface that allows an object to be cloned.	Used to create a new instance of an object that is an exact copy of the original object.

ICollection	An interface that represents a collection of objects.	Used to store and manage a collection of objects.
IEnumerable<T>	An extension of the IEnumerator interface that allows an object to be iterated over one item at a time of a specific type.	Used to iterate over the items in a collection of objects of a specific type.
IServiceProvider	An interface that provides access to services that are registered with a service provider.	Used to retrieve services that are required by an application.
IAsyncDisposable	An interface that is similar to the IDisposable interface, but it allows an object to be disposed of asynchronously.	Used to release resources that are no longer needed in an asynchronous way.

- We use `Microsoft.Extensions.Hosting.WindowsServices` Nuget to work with worker service

```
IHost host = Host.CreateDefaultBuilder(args)
    .UseWindowsService() //this is for windows, if i want to run this background process
in linux then use daemon (use dll in daemon) here
```

- At first select worker project then publish it. After publishing done of worker service some command: (Go to publish folder and copy path and exe file name)

```
// For Creating
sc.exe create MsaService binpath=D:\Users\msash\Desktop\aspnet-
b8\src\FirstDemo\FirstDemo.EmailWorker\bin\Debug\net7.0\publish\FirstDemo.EmailWorker.exe
start=auto

// For Delete
sc.exe delete MsaService
```

- For web scraping we should install HtmlAgailityPack nuget

CLASS 38-39: UNIT TEST

- A unit test is a type of software testing that focuses on verifying the correctness of individual components or units of code in isolation.
- Characteristics of unit tests include:
    1. **Isolation**: Unit tests focus on testing a single component or function in isolation, excluding external dependencies.
    2. **Deterministic**: Unit tests should produce consistent results, giving the same output for the same input.
    3. **Fast Execution**: Unit tests should run quickly, making them suitable for frequent execution during development.
    4. **Automated**: Unit tests are typically automated, allowing for easy and repeatable testing.
    5. **Independence**: Unit tests should not rely on the success of other tests and should be able to run independently.
    6. **Purpose-Specific**: Unit tests are designed to validate specific, small units of code, ensuring their correctness.
    7. **White-Box Testing**: Unit tests often have knowledge of the code's internal structure, enabling testing of individual code paths.
    8. **No User Interface**: Unit tests do not involve user interfaces and primarily deal with program logic and functions.

9. **Reproducibility**: Unit tests should be reproducible on different development environments.
10. **Minimal External Dependencies**: Unit tests minimize reliance on external systems or services to maintain consistency and reliability.

- Characteristics of unit tests (by sir):
    1. আমি যে মেথডটাকে টেস্ট করছি সেটা এই মেথডের বাইরে অন্য কোথাও কল যেতে পারবে না। এই মেথডের ভিতর থেকে যদি অন্য কোনো প্রাইভেট মেথডকে কল করা হয় যেটা এই ক্লাসেই আছে তাহলে সেখানে কল যেতে পারবে কিন্তু অন্য কোনো মেথডে কল যেতে পারবে না।
       কেনো এই বৈশিষ্ট?
       -> ধরি, আমি একটি মেথড A কে টেস্ট করছি, সেই মেথডের ভিতরে আরেকটি মেথড B কল করা আছে, যদি টেস্ট ফেল করে তাহলে বুঝা যাবে না কোন মেথড ফেইল হয়েছে। এইজন্য প্রত্যেক মেথডের জন্য আলাদা আলাদা টেস্ট করা দরকার। যদি আমরা জোর করেও এই A - মেথডের ভিতরে B কল করতে দেই তাহলে এটা ইউনিট টেস্ট হবে না (সেটা হবে ইন্টিগ্রেশন টেস্ট)
       -> ইউনিট টেস্ট, ইন্ট্রিগেশন টেস্ট এবং সিস্টেম টেস্টের মধ্যে কোডিং গত কোনো পার্থক্য নেই। শুধুমাত্র কল আমরা কতদূর যেতে দিবো সেটার উপর বেইজ করে কাজ করে। যদি একই মেথডে থাকে তবে ইউনিট টেস্ট , যদি কল এক মেথডের থেকে আরেক মেথডে যায় যা অন্য ক্লাসে আছে তাহলে ইন্ট্রিগ্রেশন টেস্ট (দুটি মেথডের ইন্ট্রিগ্রেশন ঠিক আছে কিনা চেক করে), কন্ট্রোলার থেকে ডেটাবেইজ বা অন্যান্য ফিচার চেক করে ফেললে সেটা সিস্টেম টেস্টিং।
       -> আমরা এখানে automated testing করবো। Manual tesing is not a good testing process
    2. External Resource এ কল যেতে পারবে না (ডেটাবেইজ, নেটওয়ার্ক, ফাইল সিস্টেমে কল যেতে পারবে না)।
    3. কোড এক ক্লিকে রান হতে হতে (মান কোনো কনফিগারেশন লাগে না যেনো, ১ সেকেন্ডে ১০০০ টা ইউনিট টেস্ট রান করে ফেলতে হবে, সুপার ফাস্ট হতে হবে)।

- We need to work with NUnit Test Project (This is free opensource testing framework for unit test. Another free unit test framework is XUnit Test (xunit almost similar with NUnit)). Beside this there have another test framework by microsoft which is MS Test, Visual studio test (Currently free)
- Setup method (which contain initialization) are run before every Test case run
- We should install Autofac.Extras.Moq nuget because we need to work moq like autofac
- "Setup" in unit testing is code that runs before each test, while "OnetimeSetup" typically runs once for a group of tests.
- "Teardown" in unit testing is code that runs after each test, while "OnetimeTeardown" typically runs once for a group of tests to clean up resources or perform finalization.
- Here IApplicationUnitOfWork give actual instance with dependency injection (like Resolve) and Mock use for virtual dependancy binding

```
_applicationtUnitOfWork = _mock.Mock<IApplicationUnitOfWork>();
```

- AAA: Arange (Initialize), Act (Execution), Assert (Verification)
- Static, global variable, inheritance is make hard to write unit test
- A project which is not have any test code which called lagecy project (লেগেসি মানে পুরাতন হয়ে গেছে বা বাতিল হয়ে গেছে)
- In MVC, we call enitity instead of model. Our model is Model folder's ViewModel (like course view model) and Model is subclass (we can findout model courseview model and other things also)
- Middle wire chain or pipeline:

```
// just middle wire (not chaining)
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();
app.UseSession();

// this is middle wire chain or pipeline
app.UseHttpsRedirection()
    .UseStaticFiles()
```

```
    .UseRouting()
    .UseAuthentication()
    .UseAuthorization()
    .UseSession();
```

- We say url is a endpoint (though url is not direcly endpoint, but we use this that's why this is endpoint)

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

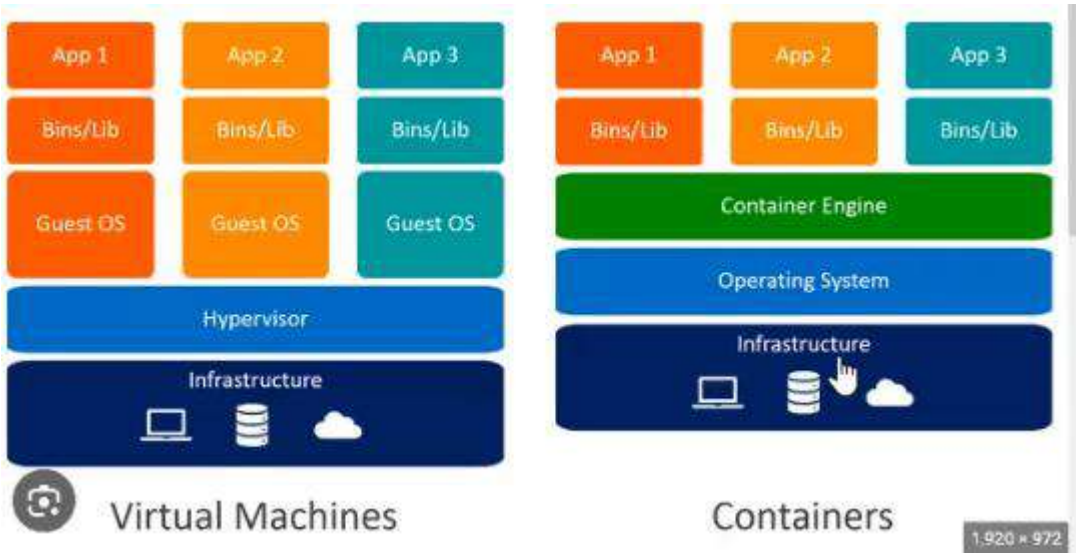
- Test-Driven Design (TDD) is a software development approach where tests are written before writing the actual code to guide development and ensure code correctness. (আগে টেস্ট কোড লিখবো, পরে একচুয়াল কোড লিখবো)
- Shouldly is a NuGet package that provides more expressive and readable assertions in unit tests, improving the clarity of test failure messages.
- We can write like this then test case run for 3 time under a Test case for a single test case (this run parallel so not sequence which test run first)

```
[Test]
[TestCase("")]
[TestCase(null)]
[TestCase(" ")]
```

- Crystal Reports is a business intelligence tool used to design and generate interactive and feature-rich reports from various data sources.
- Best pdf generate for nuget DinkToPDF

CLASS 40-43: DOCKER

- Here, VM and Container bother have insfrastructure.
- VM create OS using GuestOS and VM has hypervisor which can create and run multiple virtual machine on a single computer. On the other hand container have only one OS and container engine which can create multiple container from a single OS, as a result container is so much lightweight.



- Bin/Lib in VM is the layer that shares binaries and libraries between applications. Which have runtime, jdk, sdk and others
- Dockerfile have no extension

```
# Use the official .NET SDK image version 7.0 as the base for building the application.
FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build

# Specify the maintainer of the Dockerfile, for documentation purposes.
MAINTAINER Dev Skill

# Set the DEBIAN_FRONTEND variable to noninteractive to prevent interactive prompts during package installation.
ARG DEBIAN_FRONTEND=noninteractive

# Update the package repository to ensure the latest packages are available for installation.
RUN apt-get update
# Install the Apache web server.
RUN apt-get install -y apache2

# Set the working directory to /var for subsequent commands.
WORKDIR /var

# Expose port 80 to allow external access to the Apache web server.
EXPOSE 80

# Start the Apache web server in the foreground when the container is run.
CMD apachectl -D FOREGROUND
```

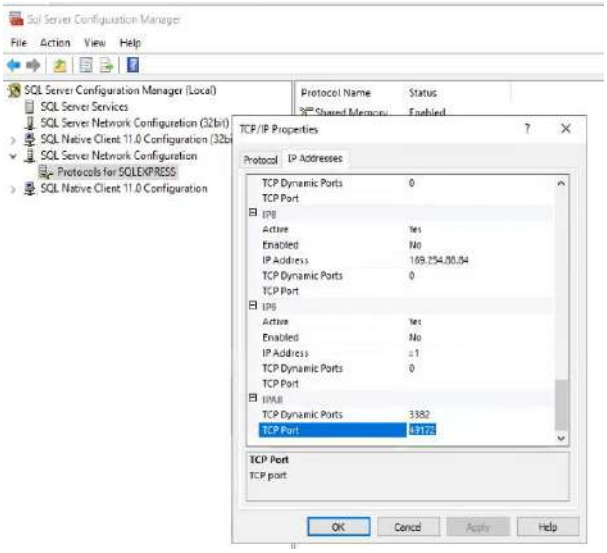
- Some command:

Category	Command	Description
Images	docker build -t test -f "D:\Users\msash\Desktop\MyCode\ASP.NET\Recap ASP.NET\Dockerfile" .	Build a Docker image named "test" using the specified Dockerfile.
	docker images	Display a list of all Docker images.

	docker rmi -f test	Delete the Docker image named "test".
	docker run test	Run the "test" image, creating a container. Use -d to run in detached mode (in the background).
Container	docker ps	Show all running containers.
	docker ps -a	Show all containers, including those that are stopped.
	docker run -d -it test	Run the "test" image in interactive mode and detached mode, allowing interaction with the terminal. After entering the container, run commands like apt-get update and apt-get install -y apache2.
	docker stop f39fc84feb38 05baeb405c86	Stop two containers with the specified container IDs.
	docker rm -f f39fc84feb38	Remove a container with the specified container ID.
	docker run -d -p 8000:80 test	Run the "test" image, mapping port 8000 on the host to port 80 in the container. Use -d to keep the command running in the background.
	docker container prune	Stop and remove all stopped containers.
Volume	docker volume ls	List all Docker volumes.
	docker volume create --driver local --opt type=none --opt device="D:\Users\msash\Desktop\MyCode\ASP.NET\Recap ASP.NET\shared" --opt o=bind test-shared	Create a Docker volume named "test-shared" with specified options for local binding.
	docker run -d -v test-shared:/var/www/html -p 8000:80 test	Run the "test" image, creating a new container with a volume named "test-shared" mapped to /var/www/html.
	docker volume rm test-shared	Remove the Docker volume named "test-shared".
	docker exec 3b60ce04778a cat /var/www/html/home.html	Execute a command (cat /var/www/html/home.html) inside the specified container, displaying the contents of the specified file.

- **Image**: A lightweight, standalone, and executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, and system tools. (Image is like a template)
- **Container**: A runnable instance of a Docker image, encapsulating the application and its dependencies, isolated from the host system and other containers. (যেটা চলে সেটা কন্টেইনার। যেটা থেকে কন্টেইনার বানানো হয় সেটা ইমেজ)
- **Volume**: A persistent data storage mechanism in Docker, allowing data to be shared and preserved between containers and the host machine.

- Must setup this feature:



- Docker file command:

```
# Use the .NET SDK 7.0 as the base image for building the application.

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build


# Set the working directory to /src.

WORKDIR /src


# Update package repository and install Node.js (we install node for future typescript),
which may be required for certain build processes.

RUN apt-get update && apt-get install -y nodejs


# Copy the Worker project files (University.Worker/*.csproj) to the container
(University.Worker/).

COPY ["University.Worker/*.csproj", "University.Worker/"]

# Copy the Infrastructure project files.

COPY ["University.Infrastructure/*.csproj", "University.Infrastructure/"]

# Copy the Persistence project files.

COPY ["University.Persistence/*.csproj", "University.Persistence/"]

# Copy the Application project files.

COPY ["University.Application/*.csproj", "University.Application/"]

# Copy the Domain project files.

COPY ["University.Domain/*.csproj", "University.Domain/"]


# Restore the dependencies for the Worker project.

RUN dotnet restore "University.Worker/University.Worker.csproj"
```

```
# Copy the entire application to the container.
COPY . .

# Change the working directory to the Worker project.
WORKDIR "/src/University.Worker"

# Build the Worker project in Release mode. (In dotnet we use Debug mode which copy extra information what we not need that why we use Release here, -o means output what keep into app folder)
RUN dotnet build "University.Worker.csproj" -c Release -o /app

# Create a new build stage named "publish" based on the previous "build" stage.
FROM build AS publish

# Publish the Worker project to the /app directory.
RUN dotnet publish "University.Worker.csproj" -c Release -o /app

# Create a new build stage named "final" based on the "build" stage.
FROM build AS final

# Set the working directory to /app.
WORKDIR /app

# Copy the published output from the "publish" stage to the final stage.
COPY --from=publish /app .

# Set the entry point command for running the application when the container starts. (dotnet is tool)
ENTRYPOINT ["dotnet", "University.Worker.dll"]
```

- Here is YML

```
version: "3"
services:
    worker:
        build:
```

```
            # This context means from where building will be running (this can not be
point one level upper directory)
            context: .
            dockerfile: University.Worker\Dockerfile
        image: university-worker-image
        env_file:
            - Env/worker.env
        container_name: university-worker-container
        volumes:
         - university-worker-volume:/app/Logs
        depends_on:
            - api
            - web
        entrypoint: ["dotnet", "University.Worker.dll"]



volumes:
    university-worker-volume:
       external: true
```

- Evn file

```
ConnectionStrings:DefaultConnection="Server=192.168.43.29,49172\\SQLEXPRESS;Database=StudentDB;User
Id=msa;Password=123456;Trust Server Certificate=True;"
```

- If keep feature wise Service name instead entity wise service name
- cd ~ (home folder of ubuntu)
- ls
- cd / (root)
- ls
- exit
- CI and CD stand for continuous integration and continuous delivery/continuous deployment. In very simple terms, CI is a modern software development practice in which incremental code changes are made frequently and reliably.
- Runtime Text Templates are used to generate dynamic code or text during runtime, allowing for flexible and customizable code generation within an application.
- If we want to push docker then first we set tag in the docker.

```
docker tag firstdemob8 devskill/aspnetb8:v1
docker login
docker push devskill/aspnetb8:v1
```

CLASS 44-47: ADVANCE SEARCH AND MAIL SENDING

- Mailkit nuget for send mail
- **SMTP** (Simple Mail Transfer Protocol) is used to send emails. It is a standard protocol that is used by all email servers. SMTP is not a secure protocol, so it is not recommended for sending sensitive emails. POP3 and IMAP can be secured using SSL or TLS encryption, which helps to protect the privacy of your emails.
- **POP3** (Post Office Protocol 3) is used to receive emails. It downloads emails from the server to your computer. POP3 is generally easier to use than IMAP, but IMAP offers more flexibility.
- **IMAP** (Internet Message Access Protocol) is also used to receive emails, but it leaves the emails on the server. This allows you to access your emails from multiple devices. IMAP offers more features than POP3, such as the ability to create and manage folders, search for emails, and mark emails as read or unread.
- CQRS (Command Query Responsibility Segregation) is a design pattern that separates read and write operations in a system, using different models for querying and updating data.
- A mediator is a behavioral design pattern that defines an object to centralize communication between components, promoting loose coupling by ensuring objects communicate only through the mediator.

```
-- Ensures that NULLs are treated as equal during comparisons
SET ANSI_NULLS ON

-- Specifies that double quotation marks can be used to define identifiers
SET QUOTED_IDENTIFIER ON

-- Modifies the stored procedure [dbo].[GetCourseEnrollments]
ALTER PROCEDURE [dbo].[GetCourseEnrollments]
@PageIndex int,
@PageSize int ,
@OrderBy nvarchar(50),
@CourseName nvarchar(250) = '%',
@StudentName nvarchar(250) = '%',
@EnrollmentDateFrom datetime = null,
@EnrollmentDateTo datetime = null,
@Total int output,
@TotalDisplay int output
AS
BEGIN
    -- Declare variables for dynamic SQL
    Declare @sql nvarchar(2000);
    Declare @countsql nvarchar(2000);
    Declare @paramList nvarchar(MAX);
    Declare @countparamList nvarchar(MAX);

    -- Disables the sending of DONE_IN_PROC messages to the client
    SET NOCOUNT ON;

    -- Counts the total number of records in CourseStudent table
    Select @Total = count(*) from CourseStudent;

    -- Constructs the dynamic SQL for counting records with filtering conditions (Here 1=1
meaning we need to make this true forever (this is easy way to make true))
    SET @countsql = 'select @TotalDisplay = count(*) from CourseStudent cs inner join
```

```
                    Courses c on cs.CourseId = c.Id inner join
                    Students s on cs.StudentId = s.Id  where 1 = 1 ';

    -- Appends filtering conditions for counting records
    IF @CourseName IS NOT NULL
    SET @countsql = @countsql + ' AND c.Name LIKE ''%'' + @xCourseName + ''%'''

    IF @StudentName IS NOT NULL
    SET @countsql = @countsql + ' AND s.Name LIKE ''%'' + @xStudentName + ''%'''

    IF @EnrollmentDateFrom IS NOT NULL
    SET @countsql = @countsql + ' AND EnrollDate >= @xEnrollmentDateFrom'

    IF @EnrollmentDateTo IS NOT NULL
    SET @countsql = @countsql + ' AND EnrollDate <= @xEnrollmentDateTo'

    -- Constructs the dynamic SQL for fetching paginated records with filtering conditions
    SET @sql = 'select c.Name as CourseName, s.Name as StudentName, EnrollDate from
CourseStudent cs inner join
                Courses c on cs.CourseId = c.Id inner join
                Students s on cs.StudentId = s.Id where 1 = 1 ';

    -- Appends filtering conditions for fetching records
    IF @CourseName IS NOT NULL
    SET @sql = @sql + ' AND c.Name LIKE ''%'' + @xCourseName + ''%'''

    IF @StudentName IS NOT NULL
    SET @sql = @sql + ' AND s.Name LIKE ''%'' + @xStudentName + ''%'''

    IF @EnrollmentDateFrom IS NOT NULL
    SET @sql = @sql + ' AND EnrollDate >= @xEnrollmentDateFrom'

    IF @EnrollmentDateTo IS NOT NULL
    SET @sql = @sql + ' AND EnrollDate <= @xEnrollmentDateTo'

    -- Adds the pagination and ordering clauses to the dynamic SQL
    SET @sql = @sql + ' Order by '+@OrderBy+' OFFSET @PageSize * (@PageIndex - 1)
    ROWS FETCH NEXT @PageSize ROWS ONLY';

    -- Defines parameter list for executing the count SQL
    SELECT @countparamlist = '@xCourseName nvarchar(250),
        @xStudentName nvarchar(250),
        @xEnrollmentDateFrom datetime,
        @xEnrollmentDateTo datetime,
        @TotalDisplay int output' ;

    -- Executes the count SQL and captures the total count in @TotalDisplay output
parameter
    exec sp_executesql @countsql , @countparamlist ,
        @CourseName,
        @StudentName,
        @EnrollmentDateFrom,
        @EnrollmentDateTo,
```

```
        @TotalDisplay = @TotalDisplay output;

    -- Defines parameter list for executing the main SQL
    SELECT @paramlist = '@xCourseName nvarchar(250),
        @xStudentName nvarchar(250),
        @xEnrollmentDateFrom datetime,
        @xEnrollmentDateTo datetime,
        @PageIndex int,
        @PageSize int';

    -- Executes the main SQL with pagination and filtering conditions
    exec sp_executesql @sql , @paramlist ,
        @CourseName,
        @StudentName,
        @EnrollmentDateFrom,
        @EnrollmentDateTo,
        @PageIndex,
        @PageSize;

    -- Prints the count SQL and main SQL for debugging purposes
    print @countsql;
    print @sql;

END
```

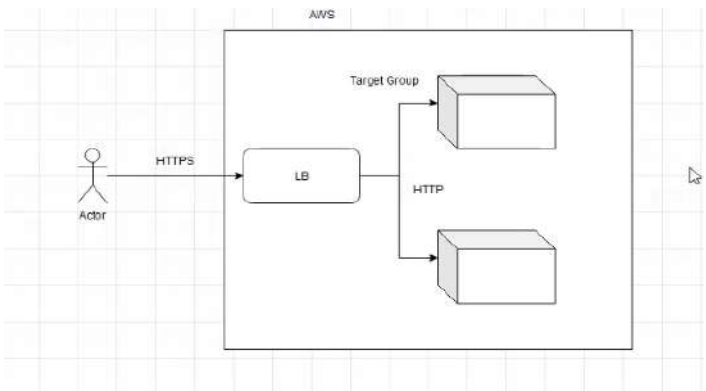
CLASS 48-53: AWS

- .t2 micro is free
- Here RDP use for allow machine server

- Subnet help to connect with different server (if some server off then other server will connected)
- Some command for linux:

```
sudo su                          (We can see all directory using this)
apt-get update                   (Using this commad we update linux)
apt-get install -y apache2       (Using this commad we install apache)
cd var/www/html                  (We go to html directory)
ls                               (We can see existing html here)

vim hello.html                   (We create a file using vim)
i                                (press i for insert text)
Press Esc  and write :wq         (For exit vim)
```

- We can communication with two instance using private ip (We can not connect from outside of aws using public ip)
- Vertical scaling (y axis) which is incresing machine power. Another is horizontal scaling (x axis) which means increase machine number (horizontal scaling is cost effective).
- If we delete server then must check volume deleted or not
- If we delete image (AMI) then must check snapshot deleted or not
- Application Load Balancer: This is use for Http/ Https connection
- Network Load Balancer: This is use for TCP/ UDP connection
- Gateway Load Balancer: If we do load balance outsite Load balancing
- We can add design aws CloudFormation to design of Load Balancing
- VPC use for set multiple subnet
- Target group (While creating Load Balancing) use for, How machine will be connected with Load Balancer which configuration.
- We can connect with Load Balancing and Instance via Target group and HTTP



- What is web socket:
- If we want to use auto scaling then we delete all instance (Just keep AMI(image)). When we create lauch template then set image here (from where instance will create). When we create auto scaling then we set our load balancer + lauch template
- After configure auto scaling we must delete auto scale group first (if we just delete instance, then it will recreate this again again). If we delete auto scaling first then template then ami then instance (We we do like this, then auto intance creating will stop).

Storage Class	Description	Use Case
Standard	The default storage class with high durability and availability.	Frequently accessed data
Intelligent-Tiering	Automatically moves objects between frequent and infrequent access tiers based on access patterns.	Variable or unknown access patterns
Standard-IA	Lower-cost option for infrequently accessed data.	Data that is accessed less frequently but requires rapid access
One Zone-IA	Similar to Standard-IA but stores data in a single availability zone, reducing costs.	Infrequently accessed data that can be easily recreated if lost
Glacier	Suitable for archiving data with long retrieval times (minutes to hours).	Archival data with retrieval time flexibility
Glacier Deep Archive	Lowest-cost option for archiving data with the longest retrieval times (12 hours).	Rarely accessed data that can tolerate long retrieval times
Outposts	Designed for use with AWS Outposts, providing low-latency access to data stored on Outposts.	Storage for AWS Outposts installations

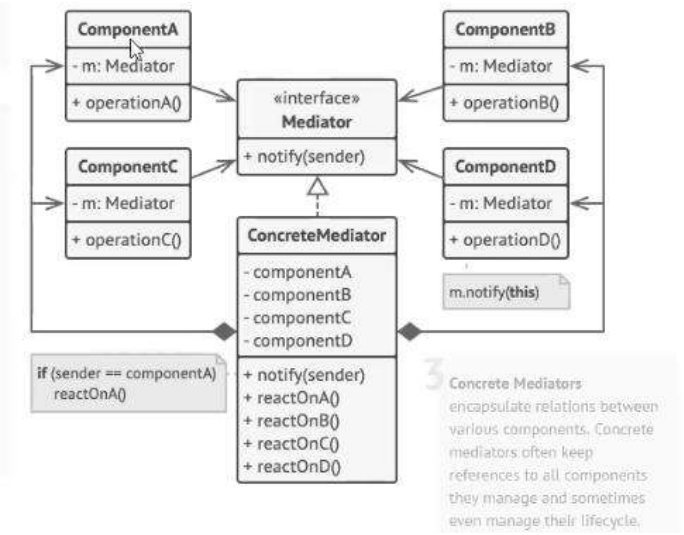
- A worker is process from a queue, that's why we use worker service
- Standard is fast but not maintain sequence, Fifo maintain sequence (worker service)
- Worker service work based on http service
- Dead letter queue is back if queue failed to send
- SQL vs NoSQL:

Aspect	SQL (Relational Databases)	NoSQL (Non-Relational Databases)
Structure	Tables with structured schema	No fixed structure, dynamic schema
Schema	Fixed schema	Dynamic schema, no predefined schema
Schema Changes	Schema changes may be complex	Easily adaptable to changes
ACID/BASE Properties	ACID properties (Strict consistency)	BASE properties (Relaxed consistency)
Joins/Relations	Supports complex joins and relationships	Typically no support for complex joins
Normalization	Normalization often used	Denormalization is common
Scalability	Vertical scaling more common	Horizontal scaling is emphasized

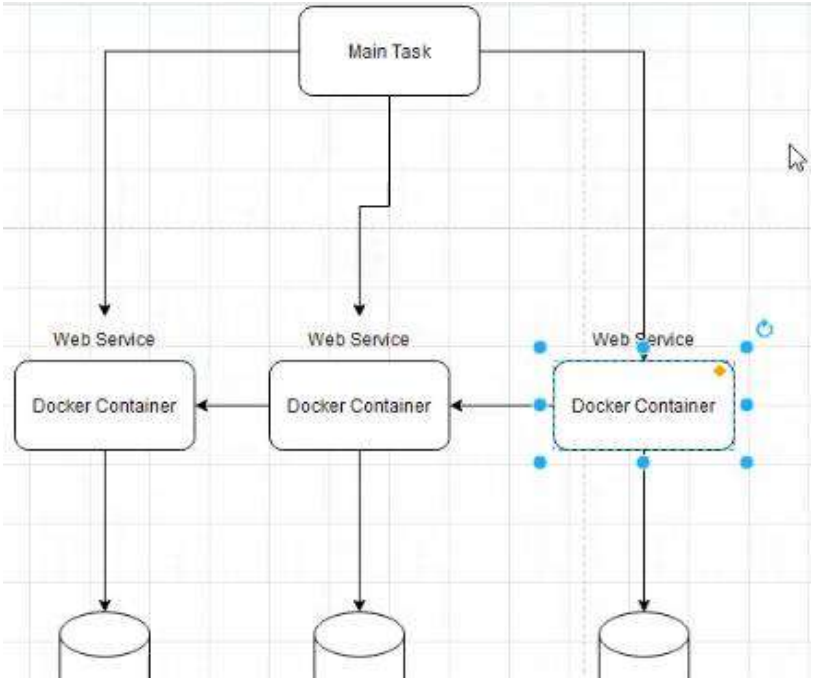
- Relational database cannot take huge pressure
- We can use Nosql when our project database when: unstructure, high velocity, high volume
- We can use nosql with relational database (suppose: 80% relational and 20% nosql)
- Nosql support only binary, string, number
- Patition key like primary id but not primary key (it accept duplicate unique key as a result we can find actual value because nosql support have id=1, id=1), we can use sort key to findout nosql data more specific
- Scan query is slow and dengerage (it check every data, every row), Query is first and findout specific data
- In nosql we have no attribute limit (suppose we have Id, Name attribute but we can but we add 3 attribute or more)
- If there have two aggregate root then we findout first aggregate root id then we use it first repositories. We see that there have only id relation with two aggregate roots
- Must see assignment of aws

CLASS 54-57: TYPESCRIPT AND ANGULAR

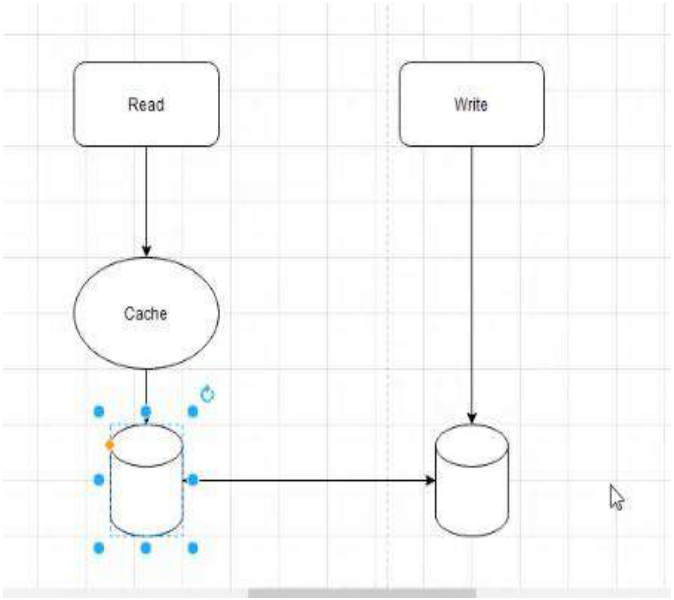
- A mediator is a behavioral design pattern that defines an object to encapsulate communication between components or objects, promoting loose coupling by centralizing communication logic. (মেডিয়েটর প্যাটার্ন হচ্ছে, অবজেক্ট এ একটা আরেকটির সাথে ডিপেন্ডেন্সি থাকে বা কল গুলো হয়ে সে কল গুলোকে ডিরেক্ট কল না দিয়ে ভায়া হয়ে কল দেওয়া। অন্য ভাবে বলতে গেলে, আমরা যখন ডিপেন্ডেন্সি ইঞ্জেকশন করবো তখনও কিন্তু ডিরেক্ট কল হচ্ছে (ইন্টারফেসকে ডিরেক্ট কল করছে, মাঝে কিন্তু মিডল ম্যান নাই)। সো আমরা চাই এমন কিছু যেটার মাঝে মিডল ম্যান থাকুক। মেডিয়েটরের কাজ হচ্ছে মাঝখানে ভায়া হয়ে বাইপাস করে নিয়ে যাওয়া। নিচের চিত্রে componentA মিডেলম্যান হিসেবে কাজ করে। আর if(sender==componentA){} ocp violation করে)



- Microservice work as parallel. একটি আস্ত প্রজেক্টকে অনেকগুলো টুকরো টুকরো করে ফেলবো। যেমন কয়েকটি ক্রুডের জন্য কয়েকটি সার্ভিস বানাবো বা কন্টেইনার বানাবো একটি। This is need for hugely loaded service. Netflix microservice is iconic microservice (They work unbelieveable work. Netflix kill there service by own to check work or not)



- CQRS (Command query responsibility segregation) : যদি অনেক বড় প্রজেক্টের জন্য অনেক read and write অপারেশন হয় তবে আমরা রিড এবং রাইটের জন্য আলাদা আলাদা কমান্ড কুয়েরি লেখতে পারি যার ফলে চাপ কমে যাবে। নিচের চিত্রের মতোঃ



CQRS stands for Command Query Responsibility Segregation, and it is a software architectural pattern that suggests separating the responsibilities for reading and writing data in a system. In a CQRS architecture:

1. Command Side (Write): Handles operations that modify data. It involves commands, which are requests to change the state of the system.

2. Query Side (Read): Handles operations that retrieve data. It involves queries, which are requests to get information from the system.

By segregating the read and write operations, CQRS aims to improve scalability, performance, and flexibility in designing complex systems. It allows optimization of the read and write paths independently, enabling the use of different models for reading and writing data. This pattern is often used in conjunction with event sourcing, where changes to the state of an application are captured as a series of events.

CQRS is especially beneficial in scenarios where the read and write patterns of an application differ significantly, and optimizing for one does not necessarily optimize for the other. While CQRS introduces additional complexity, it can be a powerful pattern for certain types of applications, such as those with high scalability and diverse querying requirements.

- TypeScript provides static typing for JavaScript, enhancing code reliability and maintainability in large-scale applications.
- Any vs Object is TS: (Here ‘any’ datatype have a similaties like C# dynamic)

Feature	Object Type	any Type

Type Inference	Provides type information for non-primitive types.	No type information is enforced; allows any type.
Type Checking	Provides type checking for non-primitive types, but limits access to specific properties/methods without additional type assertions or checks.	No type checking; allows any operations on the variable without type restrictions.
Type Safety	Offers some level of type safety for non-primitive types, but may require additional type assertions or checks.	Lacks type safety; provides maximum flexibility but at the cost of potential runtime errors.
Example	let prettySure: Object = "34";	let notSure: any = "34";
Common Use Cases	Working with non-primitive types when a more specific type is not known or important.	When maximum flexibility is needed, or when interfacing with dynamic/unknown data.

- We can declare void value in TS

```
let unusable: void = undefined;
unusable = null; // OK if `--strictNullChecks` is not given
```

- Here is in enum in TS, we can findout value like array which maybe not have in c#

```
enum Color{
  Red = 1,
  Green,
  Blue
}
let colorName: string = Color[2];
```

- `never` is used to represent values that never occur, such as functions that always throw exceptions or never return.
- C# এর ফাংশনও একেকটি datatype.
- Type assertions in TypeScript are used to tell the compiler to treat a particular expression as a different type, providing flexibility when the actual type is more specific than the inferred or declared type. (typecasting maybe)
- JS have not any oop concept but TS have this
- We can use lambda, inline interface and etc in TS
- We can create class as interface in TS (Reverse method of C#)
- Aliases in TypeScript provide a concise and more readable way to represent complex type annotations
- Using ES5 in TypeScript allows broader compatibility for targeting older browsers and environments.
- Decorator: এটা experimental. C# এর উপরে যেমন attribute লাগাতে পারি ঠিক তেমন
- Union: এটা and এর মতো কাজ করে। (এর আগে ফাংশনে অর use করেছি)
- What is deference between c# vs ts vs angular:

Aspect	C#	TypeScript (TS)	Angular
Type System	Static	Static	Static
Primary Use	General-purpose	Frontend language	Frontend framework
Platform	.NET framework	Any	Web (runs in browsers)
Compiled to	Intermediate Language (IL)	JavaScript	JavaScript
Object-Oriented	Yes	Yes	Yes
Superset of	-	JavaScript	TypeScript
Framework	.NET	-	Angular
Developed by	Microsoft	Microsoft	Google
Main IDEs	Visual Studio	Visual Studio Code	Visual Studio Code
Module System	Common Language Runtime (CLR)	CommonJS, AMD, ES6	Angular Modules

Concurrency	Supports multi-threading	Async/await, Promises	Reactive Extensions (RxJS)
Language Level	High-level	High-level	High-level
Usage	Backend development	Frontend development	Frontend development

- WebSocket is a communication protocol that provides full-duplex communication channels over a single, long-lived connection, allowing for real-time data transfer between a client and a server. It enables bidirectional communication, making it well-suited for applications requiring low-latency and real-time updates, such as chat applications, online gaming, and financial platforms.
- At first we go to wwwroot and Add New Item > Select Typescript file & install nuget pakage *Microsoft.Typescript.MSBuild* + (must delete filterizr plugin from AdminLTE)
- If we need to move generated js from ts in any specific folder then:

```
  "compilerOptions": {
    // Disallow implicit 'any' types
    "noImplicitAny": false,

    // Halt compilation on any error
    "noEmitOnError": true,

    // Preserve comments in the generated JavaScript
    "removeComments": false,

    // Generate source maps to enable debugging in the original TypeScript code
    "sourceMap": true,

    // Output directory for the compiled JavaScript files
    "outDir": "wwwroot/js",

    // Specify the ECMAScript target version for the generated JavaScript
    "target": "es5"
  }
}
```

- Typescript ‘Any’ datatype have similarities with C# dynamic datatype.

- Angular is frontend framework.
- Angular is a TypeScript-based open-source web application framework developed and maintained by Google. It is a comprehensive front-end framework used for building dynamic, single-page web applications (SPAs). Here are some key aspects and reasons to use Angular:

  1. Declarative UI: Angular uses declarative templates with HTML to define the structure of the user interface, making it easier to understand and maintain.

  2. Two-Way Data Binding: Angular provides two-way data binding, allowing automatic synchronization between the model (business logic) and the view (UI). Changes in one are reflected in the other, simplifying development.

3. Modular Architecture: Angular promotes a modular and component-based architecture, making it easier to organize and maintain code. Components encapsulate specific functionality and can be reused across the application.

4. Dependency Injection: Angular's dependency injection system helps manage component dependencies, making it easier to develop, test, and maintain code.

5. TypeScript Language: Angular is built with TypeScript, a superset of JavaScript that adds static typing. This enhances development productivity by catching errors at compile-time and providing better tooling support.

6. Cross-Platform Development: Angular supports cross-platform development, enabling the creation of web applications as well as mobile applications using tools like Ionic and NativeScript.

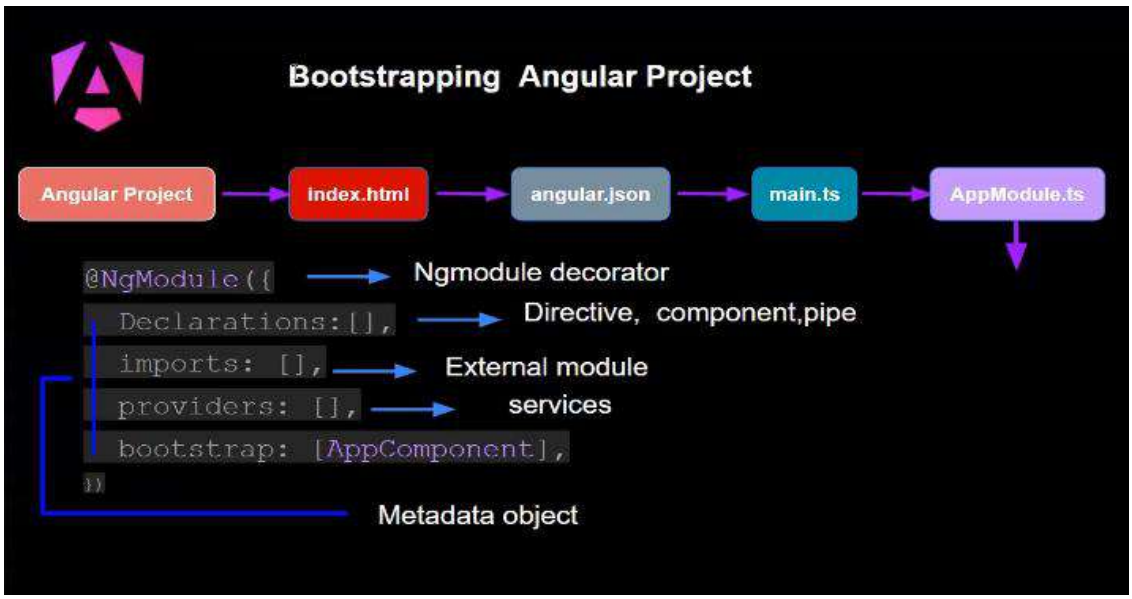
7. Rich Ecosystem: Angular has a vast ecosystem with a rich set of libraries, tools, and extensions that can be leveraged to enhance development.

8. Official Support and Community: Being developed and maintained by Google, Angular has strong official support, regular updates, and an active community. This ensures that developers have access to resources, documentation, and solutions to common issues.

9. Testing Support: Angular is designed with testability in mind, and it comes with tools for unit testing, end-to-end testing, and integration testing.

10. Scalability: Angular is well-suited for building large and scalable applications due to its modular architecture and the ability to manage complex state and data flow.

In summary, developers choose Angular for its powerful features, comprehensive tools, and a structured approach to building modern web applications. It's particularly well-suited for projects that require a robust framework, scalability, and a rich ecosystem.



- When we work with react, vue, angular then we can not use MVC with that. If we want to see view then we write angular and when we want to see C# then work work with API
- Now I open vs code and open a terminal and run command:

```
// Move directory to D
cd "D:\Users\msash\Desktop\MyCode\ASP.NET\Recap ASP.NET\src"
// If there haven’t any permission then set it
Set-ExecutionPolicy -ExecutionPolicy Unrestricted
// Now Create a project
```

```
ng new university-front --no-standalone //enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? : NO
// Open visual studio code
code .
// Now start angular. Go to one step down to start this, cd university-front
ng serve
```

- Angular Signals can be used to manage user profile data and e-commerce cart updates. With Signals, user profile updates are instantaneously reflected. This avoids the need for manual subscription management or usage of async pipe. It also ensures the UI remains in sync with the profile data.
- Angular folder structure:

1. **.angular**: Angular configuration folder, storing project-specific configuration files.
2. **.vscode**: VSCode settings folder, holding project-specific Visual Studio Code settings and configurations.
3. **node_modules**: Folder where npm packages and dependencies are installed. (It takes 2-3 minutes to download)
4. **src**: Folder containing the source code of the Angular application.
    - **app**:Folder for application-specific components, modules, and services. (We will code here)
    - **assets**: Folder for static assets like images and configuration files.
    - Here also have, index.html, styles.css, main.ts (main.ts এটা অনেকটা program.cs এর মতো, এটা ইনিশিয়াল কিছু কোড রান করে। এখান থেকেই প্রগ্রাম শুরু হয়), favicon.ico
5. .**editorconfig**: Configuration file for code editors.
6. .**gitignore**: Configuration file specifying files and directories to be ignored by Git.
7. **angular.json**: Angular CLI configuration file for project settings. (অনেকটা appsetting.json এর মতো। ভবিষ্যতে এখানে কিছু চেঞ্জ করবো)
8. **package-lock.json**: Locks down exact versions of npm package dependencies for consistent, reproducible builds across different environments.
9. **package.json**: Configuration file for npm packages, scripts, and project metadata. (যা যা প্যাকেজ আর ভার্সন ইউজ হচ্ছে সেগুলো থাকে)
10. .**tsconfig.app.json**: TypeScript configuration file specifically for the app directory.
11. **tsconfig.json**: TypeScript configuration file. (এখানে TypeScript এর কনফিগ থাকে)
12. .**tsconfig.spec.json**: TypeScript configuration file specifically for the app's unit tests.

- Here main.ts, AppModule is tutti module. From this angular start (like C# program.cs)

```
platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

- Angular Module vs Angular Component:

Aspect	Angular Module	Angular Component
Definition	A logical grouping of components, services, directives, etc.	A fundamental building block encapsulating view and logic
Purpose	Organizes and manages related features of an application	Represents a part of the user interface and its behavior
File Structure	Typically defined in a separate TypeScript file (*.module.ts)	Typically defined in a separate TypeScript file (*.component.ts)
Imported By	Imported by other modules to use their declared components and services	Declared within modules and can be used by other components
Declaration	Uses @NgModule decorator for declaration and configuration	Uses @Component decorator for declaration and configuration

Metadata	Contains metadata such as declarations, imports, exports, and providers	Contains metadata such as template, styles, selector, and more
Encapsulation	Provides encapsulation by creating a separate namespace for components	Encapsulates its own view, data, and behavior
Dependencies	Can have dependencies on other modules	Can depend on services, other components, or modules
Communication	Communication between modules is typically achieved through services	Communication between components is achieved through inputs, outputs, and services
Example	@NgModule({ declarations: [AppComponent], imports: [CommonModule] })	@Component({ selector: 'app-root', template: ' Hello World ' })

- Module: এটা হচ্ছে অনেক বড় প্যাকেজ বা লাইব্রেরীর মতো। এটার মধ্যে আমরা কম্পোনেন্ট তৈরি করতে পারি। আমরা যদি কোনো কিছু প্যাকেজ আকারে রিলিজ করতে চাই তখন আমরা মডিউল আকারে বানাতে পারি।
- Component: আমরা যখন কোনো ভিজিয়াল ইমেজ তৈরি করি সেটা কম্পোনেন্ট আকারে তৈরি করি। একটা কম্পোনেন্ট মূলত চারটি ফাইল নিয়ে গঠিত হয়। src > app এর মধ্যে এসব ফাইল পাওয়া যাবে (যদিও app-routing.module.ts ফাইল আছে যা রাউটিং সংক্রান্ত)
- **app.component.css**: Contains component-specific styles to define the appearance and layout of the Angular component.
- **app.component.html**: Holds the template or view for the Angular component, defining its structure and content.
- **app.component.spec.ts**: Provides unit tests for the Angular component, ensuring its behavior meets expectations.(এটা app.component.ts এর সাথে ম্যাপিং এর জন্য ইউজ হয়)
- **app.component.ts**: Contains the TypeScript class definition for the Angular component, including logic, properties, and methods. (এখানে কোডিং লজিক রাখবো)
- In angular we can pass data from parent to child using Input event and child to parent using Output event. Here app.component is parent and button.component is child
- How we add component:

```
// Now we add component
ng generate component components/button

// First we go app.component.html
<div>
  <app-button></app-button>
</div>

// then we go button.component.html
<p>button works!</p>

// Finally we start angular
ng serve

[Note: Use command "npm config set legacy-peer-deps true" and "npm install" while want to  ng serve git project, because git ignore big node-module folder while commiting ]
```

- Field add into component: আমরা মডেলের মধ্যে asp for দিয়ে যেমন বাইন্ড করি। ঠিক তেমনি আমরা এখানে ফিল্ড ইউজ করবো কারন আমরা যেনো customize করতে পারি component কে + component গুলো যেনো static করতে না হয়।
- To adding field we write, button.component.ts

```
// The `import` statement is used to bring in Angular symbols (`Component`, `OnInit`, `Input`, `Output`, `EventEmitter`) from the `@angular/core` module, which is part of the Angular framework. These are imported locally from the Angular framework installed in your project, not from the internet.

// `export` is used to make the `ButtonComponent` class available for use in other files within the same project (locally), not from the internet.

// `Component`: Represents an Angular component, encapsulating the component's logic, view, and data. In easy word, Components in Angular provide a way to create reusable and encapsulated UI elements

//`OnInit`: `OnInit` is used to implement the `ngOnInit` lifecycle hook in Angular components, allowing you to perform component initialization logic when the component is instantiated.
//`Input`: Declares an input property, allowing data binding to pass data from parent to child components. In easy word, we can pass data parent to child using Input

//`Output`: Declares an output property, allowing child components to emit events to be captured by parent components. In easy word, we can pass data child to parent using Input

//`EventEmitter`: Emits events that can be subscribed to, facilitating communication between components. In easy word, is used to emit (প্রেরণ করা) custom events from a child component to its parent component in Angular.

// `ngOnInit` is used to perform initialization logic for an Angular component, such as initializing properties or making asynchronous calls, when the component is being created.


//.............. First we go child: button.component.ts..............(here OnInit is a event)

import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
@Component({
  selector: 'app-button',
  templateUrl: './button.component.html',
  styleUrls: ['./button.component.css']
})
export class ButtonComponent implements OnInit {
  @Input() text:string= "";
  @Input() color:string= "";
  @Input() cssClass:string= "";
  @Output() btnClick = new EventEmitter();
  constructor() { }
  ngOnInit(): void {
  }
  onClick(){
    this.btnClick.emit(); // emit is like C# invoke
    }
}

//.................. then we go child: button.component.html.............
<!-- this is static, we dont use this -->
```

```
<!-- <input type="button" style="background-color: yellow;" value="Button" /> -->

<!-- We will use like this -->
<input type="button" [ngClass]="cssClass" [ngStyle]="{'background-color':color}"
value="{{text}}" (click)="onClick()"/>

//.................. Now we go parent: app.component.ts.............
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'university-front';

  onClick1(){
    alert("Button 1 clicked!");
    }

  onClick2(){
      alert("Button 2 clicked!");
      }
}

//.................. we add styles.css (this is for global css, we can also use this css
button.component.css).....................

.blueBorder{
    border: 2px rgb(0, 0, 0) dashed;
    }


//.................. finally we go parent: app.component.html.............
<div>
  <app-button color="red" text="Button1" (click)="onClick1()"></app-button> // this click
is js click
  <br>
  <br>
  <app-button color="green" cssClass="blueBorder" text="Button2"
(btnClick)="onClick2()"></app-button> // we have customized event: named btnClick
</div>
```

- We can create composit component (Suppose we create a text box for login page and a button for login page. If we use both in login page then we can say this is composit component)
- How to get static data:

```
// First we create service class using cmd
ng generate service services/student
// Now we add component
ng generate component components/student
```

```
// Now add a folder named ‘data’ and create Student and IStudent
// ..............................Student.............................
import { IStudent } from './IStudent';
export class Student implements IStudent {
  public name: string = '';
  public roll: number = 0;
  public dateOfBirth: Date = new Date();

  // constructor syntax to easily initialize current object
  public constructor(init?: Partial<Student>) {
    Object.assign(this, init);
  }
}
// ..............................IStudent.............................
export interface IStudent
{
    name : string;
    roll : number;
    dateOfBirth : Date;
}

//.................. student.component.ts.............
import { Component, Input } from '@angular/core';
import { IStudent } from 'src/app/data/IStudent';

@Component({
  selector: 'app-student',
  templateUrl: './student.component.html',
  styleUrls: ['./student.component.css']
})
// This Angular component named `StudentComponent` has an input property `students` of
type `IStudent[]`, allowing external components to pass an array of student data to it.
The default value for `students` is an empty array.
export class StudentComponent {
  @Input() students : IStudent[] = [];
}

//.................. student.component.html.............
<p>student works!</p>

<!-- <div *ngFor="let s of students">
    {{s.name}} {{s.roll}} {{s.dateOfBirth}}
</div> -->

<table border="1">
    <thead>
      <tr>
        <th>Name</th>
        <th>Roll</th>
        <th>Date of Birth</th>
      </tr>
    </thead>
    <tbody>
```

```
      <tr *ngFor="let s of students">
        <td>{{ s.name }}</td>
        <td>{{ s.roll }}</td>
        <td>{{ s.dateOfBirth }}</td>
      </tr>
    </tbody>
  </table>

//.................. student.service.ts.............
import { Injectable } from '@angular/core';  // `@Injectable` is used to allow Angular to
inject dependencies into a service, making it available for dependency injection
throughout the application.
import { IStudent } from '../data/IStudent';
import { Student } from '../data/Student';

// `@Injectable({ providedIn: 'root' })` is used to register the service at the root
level, making it a singleton service instance shared across the entire Angular
application.
@Injectable({
  providedIn: 'root'
})
export class StudentService {

  constructor() { }

  getStudents() : IStudent[]{
    return [
    new Student({ name : "Meem",  roll: 1, dateOfBirth : new Date(2000, 11, 18) }),
    new Student({ name : "Anika", roll : 2, dateOfBirth : new Date(2000, 11, 19) })
    ];
    }
}


//.................. app.component.ts.............
import { Component } from '@angular/core';
import { IStudent } from './data/IStudent';
import { StudentService } from './services/student.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'university-front';

  students: IStudent[] = [];
  constructor(private studentService: StudentService) {}

  update(){
    this.students = this.studentService.getStudents();
  }
```

```
  onClick1(){
    alert("Button 1 clicked!");
    }

  onClick2(){
      alert("Button 2 clicked!");
      }
}
import { Observable, of } from 'rxjs'; import { HttpClient } from '@angular/common/http';
//.................. finally we go app.component.html.............
<div>
  <app-button color="red" text="Button1" (click)="onClick1()"></app-button><br><br>

  <app-button color="green" cssClass="blueBorder" text="Button2"
(btnClick)="onClick2()"></app-button><br><br>

  <app-button color="white" text="Show Static Result" (btnClick)="update()"></app-
button><br><br>
// This Angular template syntax binds the local variable `”students”` from the component
to the input property `[students]` of the `<app-student>` component. Binding allows data
to be passed from the parent component to the child component.
  <app-student [students] = "students"></app-student>
</div>
```

- This code is used to import the Observable and of classes from the RxJS library and the HttpClient class from Angular's @angular/common/http module, enabling the use of observables for asynchronous operations and making HTTP requests in an Angular application.

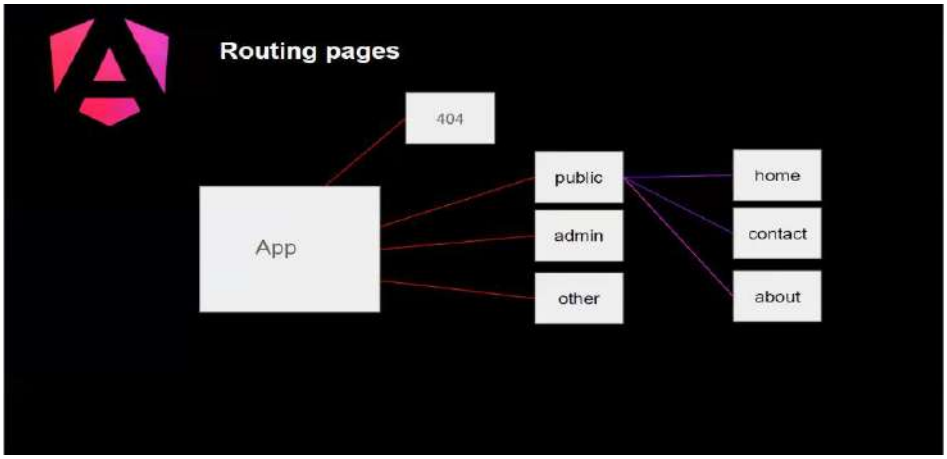
```
import { Observable, of } from 'rxjs';
import { HttpClient } from '@angular/common/http';
```

CLASS (INTERNSHIP SESSION): ANGULAR EXTRA

- Single line component: (If there have error in code then we can not caught it)

```
.........................home.component.ts.............
import { Component } from "@angular/core";

@Component({
    selector: 'app-home',
    template: '<h2> Hi </h2>',
    styles: 'h2[font-size: 20px]'
})
export class HomeComponent{
}

.........................app.component.html.............
<!-- <app-home></app-home> -->
<app-home/> <!-- this is work on angular 17 -->

.........................app.module.ts.............
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

- Routing:



- Creating component using command:

```
// Internship class command (here we skip unit test, and create standalone স্বতন্ত্র component)
ng generate component components/header --skip-tests=true --standalone=true
// sir generate
ng generate component components/student
```

- If we want to add a routing module in a component then:

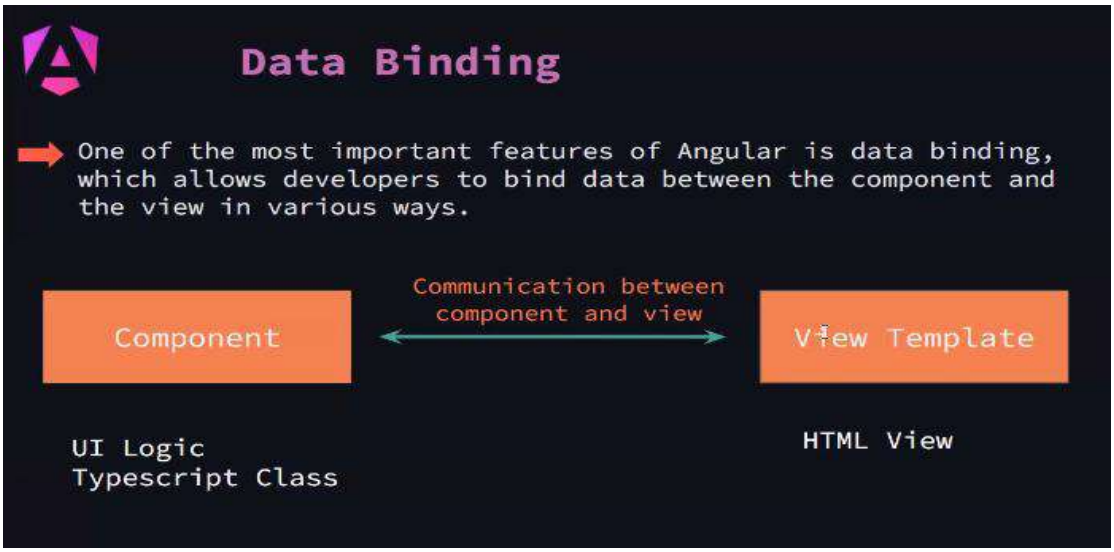
```
// First create a component in any name (here I give component name ‘Public’)
ng generate component areas/public

// Routing in Angular is used to navigate between different views or components in a
single-page application, enabling a seamless and dynamic user experience.
// now we add routing module like app.module.ts (go to areas directory and then apply cmd)
ng generate module public --routing          or        ng g m public --routing
```

- If we want to use simple routing just http://localhost:4200/footer or header or normal (this routing work even path: ‘’, this routing work also standalone and normal component also)

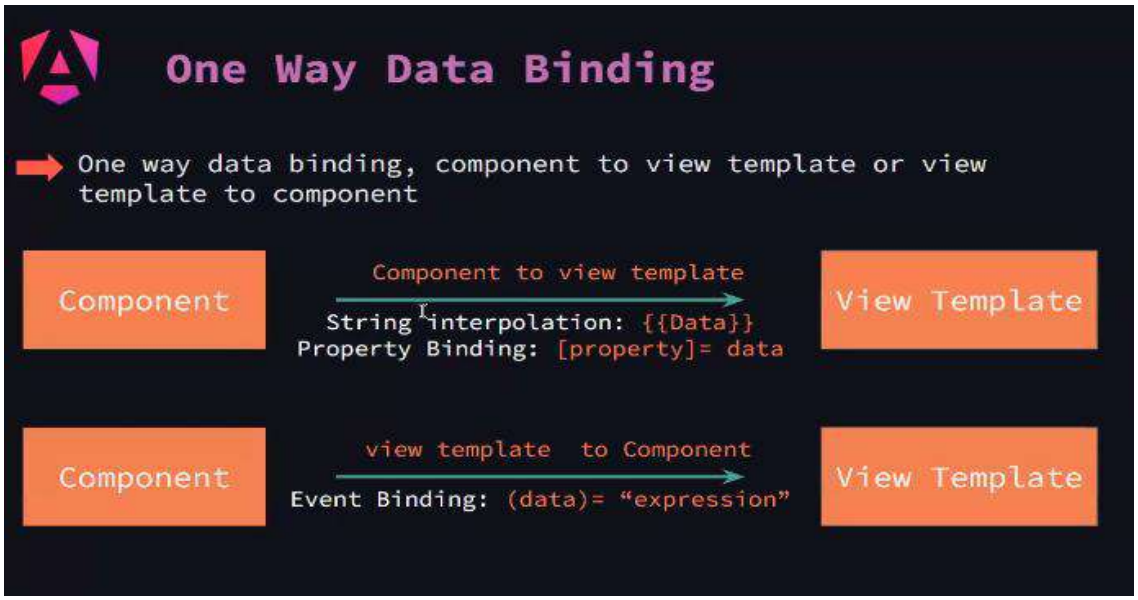
```
.........................app-routing.module.ts.............

const routes: Routes = [
  {
    path: 'normal',
    component: NormalComponent
  },
  {
    path: 'header',
    component: HeaderComponent
  },
  {
    path: 'footer',
    component: FooterComponent
  },
  {
    path: '**',
    component: NotfoundComponent
  }
];

.........................app.component.ts.............
<router-outlet></router-outlet>
```

- Difference between standalone component vs normal componet
    1. In standalone component we not add it in ngModule directory and we can use it directly
    2. We use standalone component for quick and small works, single page, where lazy loading not required

- **Data Binding:** One of the most important features of Angular is data binding, which allows developers to bind data between the component and the view in various ways.



- **One way data binding:**



- **One way data binding:**