3/10/24

ASSIGNMENT 3

Unit-4,5&6 Web Services, Using Controllers with Views & Working with Model, Form and Other Web Applications

Q.1 What is XML Web Service? Write the full form and purpose of XML, SOAP, UDDI and WSDL.

→ An XML (Extensible Markup Language) Web Service is a platform-independent service that allows applications to communicate with each other over the web using XML-based messages. It enables interoperability between different software applications running on various platforms.

1. XML - Extensible Markup Language.
   It is designed to store and transport data in a format that is both human-readable and machine-readable. It is widely used for data representation in web services.

2. SOAP - Simple Object Access Protocol.
   It is used for exchanging structured information in the implementation of web services. It relies on XML to format the messages and is designed to work over different protocols such as HTTP, SMTP.

3. UDDI - Universal Description, Discovery and Integration.
   It is a platform-independent framework for discovering web services.

It allows businesses to publish information about their services, making it easier for other businesses to find and use them.

4. WSDL - Web Services Description Language.
It is used to describe the functionalities offered by a web service. It provides details like the service's location, the methods it provides and how to invoke them.

Q.2 Define web services. Why web services? Discuss any two reasons.

→ A web service is a standardized way of allowing different applications, typically running on different platforms, to communicate and share data over the internet. It uses open protocols like HTTP and formats like XML or JSON to enable the exchange of data between systems, regardless of the platform or language they are built on.

→ Web services are crucial because they enable interoperability and flexibility in distributed computing.

i) Platform Independence : Web services allows applications developed in different languages (JAVA, .NET) and running on different

operating systems (windows, linux) to communicate and share data seamlessly.

ii) Reusability : web services promote reusability by exposing business logic over the internet, allowing different applications to reuse existing services without the need to redevelop similar functionalities.

Q.3 Explain "Interoperability" and "exposing an existing function on the network" related to web services.

→ Interoperability, refers to the ability of different systems, built on various platforms and using different technologies, to work together and exchange data. It is achieved through the use of open standards like HTTP, XML, SOAP and WSDL, which allow disparate systems to communicate regardless of their underlying software or hardware.

eg: A web service written in Java running on a linux server can interact with a client application built in .NET on windows because both use standard protocols (like SOAP or HTTP) and formats (like XML or JSON).

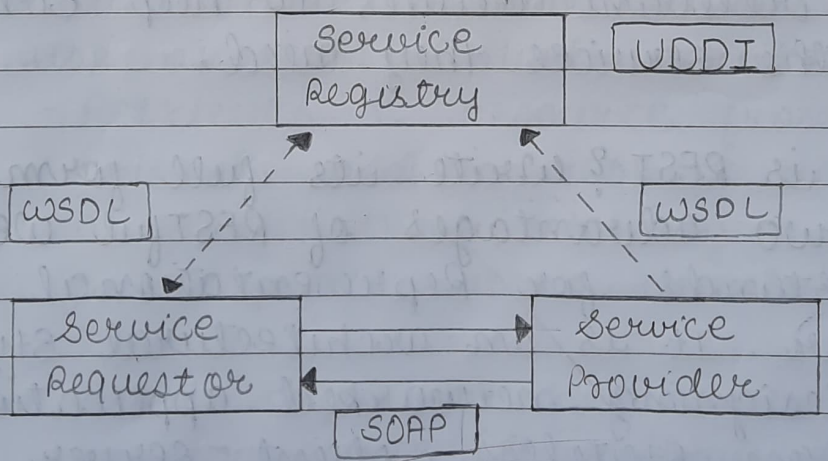
→ Exposing an existing function on the network, means taking an existing

application functionality (such as a business operation, algorithm or database access method) and making it available as a web service so that ~~order~~ other applications can access it over the internet or an intranet.

eg: If a company has a function that retrieves product details from a database, that function can be turned into a web service. Once exposed, clients can call this functions remately over the network using standardized protocols (HTTP) without knowing the underlying implementation.

Q.4 Draw the diagram and discuss the "Service Provider" and "Service Registry" role of web Services.

—ψ



1. Service Provider.

The entity or system that hosts a web service and makes it available for external use. It is responsible for creating

and maintaining the web service, ensuring that it can be accessed over the network by service consumers. The service provider also publishes the service description (typically in WSDL format) so that potential users can understand how to interact with the service.

2. Service Registry.

A centralized directory where web services are listed or registered. It allows service providers to publish their services and service consumers to discover these services. UDDI is often used as the standard for # service registries. The registry contains metadata such as service descriptions, access points (URLs) and other relevant details to help consumers find the services they need.

Q.5 What is REST? Write its full form. Explain any two advantages of RESTful web Services.

—ᵠ REST stands for Representational State Transfer. It is an architectural style used for designing networked applications. REST relies on stateless, client-server communication, primarily over HTTP, where resources are represented as URLs (Uniform Resource Locators) and standard HTTP methods (GET, POST, PUT, DELETE) are used to perform actions on those resources.

i) Scalability : REST is highly scalable due to its stateless nature, which means each request from a client to a server must contain all the information needed to understand and process the request. This allows easy distribution and scalability across multiple servers without maintaining session states.

ii) Flexibility and Simplicity : REST uses standard HTTP methods and can return data in multiple formats like JSON, XML and HTML. This flexibility allows it to interact with a wide range of clients, from web browsers to mobile apps, using simple HTTP requests.

Q.6 Write Five HTTP method names along with its use in RESTful web Services.

→ 1. GET - Retrieves a resource from the server. This is used to request data from a specific URL. (Fetching a list of users or a specific user's details).

2. POST - Sends data to the server to create a new resource (Submitting form data or creating a new user in a database).

3. PUT - Updates or replaces an existing resource with new data. (Updating a user's details by sending new information to replace the old data).

4. DELETE - Deletes a specified resource from the server. (Removing a user or item from a database).

5. PATCH - Partially updates an existing resource. (Modifying specific fields of a user's details without sending the entire resource.)

Q.7 Analyze and explain the following code written in Program.cs file:

```
① using Microsoft.EntityFrameworkCore;
  using WebApp.Models;
② var builder = WebApplication.CreateBuilder(args);
③ builder.Services.AddDbContext<DataContext>(opts => {
      opts.UseMySql(builder.Configuration[
        "ConnectionStrings:ProductConnection"],
      opts.EnableSensitiveDataLogging(true);
  });
④ builder.Services.AddControllers();
⑤ var app = builder.Build();
⑥ app.UseStaticFiles();
⑦ app.MapControllers();
⑧ var content = app.Services.CreateScope().ServiceProvider
      .GetRequiredService<DataContent>();
  SeedData.SeedDatabase(content);
⑨ app.Run();
```

→ The provided code is a basic setup of an ASP.NET Core application using Entity Framework Core with a MySQL Database.

① Using Directives

> The 'Microsoft.EntityFrameworkCore' namespace is included for using Entity Framework Core, which is essential for database operations.
> 'WebApp.Models' is the namespace for the application's Data models.

② WebApplication Builder Setup.

> This line creates a new WebApplicationBuilder instance, which is responsible for configuring services and the application's environment.

③ Registering the DbContext.

> 'AddDbContext<DataContext>' : Registers the application's 'DataContext' for dependency injection.
> 'UseMySql' : Configures Entity Framework to use a MySQL database, getting the connection string from the configuration file (appsettings.json).
> 'EnableSensitiveDataLogging(true)' : Enables logging of sensitive data like parameter values for debugging purposes.

④ Adding Controllers.

> Registers support for MVC controllers, enabling routing and handling of HTTP requests through controllers.

⑤ Building the Application.

> Builds the WebApplication from the configured services.

⑥ Serving Static Files.
> Allow serving static files (images, CSS, JavaScript) from the wwwroot directory.

⑦ Mapping Controllers.
> Maps incoming HTTP requests to the appropriate controller actions.

⑧ Database seeding
> This block creates a scope for the application's services and retrieves the 'DataContext' instance.
> 'SeedData.SeedDatabase(context)' : used for seeding initial data into the database.

⑨ Running the Application.
> Starts the web application and begin listening for incoming HTTP requests.

Q.8 What is "convention routing"? Explain with at least two different examples.

—> Convention Routing is a traditional way of defining routes in an MVC application, where routes are configured globally in the application's 'Startup.cs' or 'Program.cs' file using a specific pattern. These routes follow a convention, usually based on URL segments, such as '{controller}/{action}/{id}',

where each part corresponds to the controller name, action, method and optional parameters.

eg: (i) Basic (Default) Route Example.

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

⇒ If no controller or action is specified, it defaults to the 'HomeController' and its 'Index()' action.

eg: (ii) Custom Route Example

```
app.UseEndpoints(endpoints => {
    endpoints.MapControllerRoute(
        name: "product",
        pattern: "shop/{action=List}/{id?}",
        defaults: new {controller="Product"});
});
```

⇒ If only '/shop' is accessed, it defaults to the 'List' action of 'ProductController'.

⇒ It follows a simple and easy-to-understand structure. You can set default values for controllers and actions, making routing predictable and straightforward.

Q.9 Compare and differentiate following two lines of code :

—	app.MapDefaultControllerRoute();	app.MapControllerRoute("Default", "{controller=Home}/{action=Index}/{id?}");
>	Sets up the default routing convention for controllers.	Defines a custom routing pattern with specific default values for controller and action.
>	Uses the default conventions without explicitly specifying them.	Explicitly sets 'home' and 'index' as default controller & action, while 'id' is optional.
>	Very simple and quick way to setup routing	More detailed and allows for custom naming and explicit definition of route pattern
>	Less flexible since it follows the built-in convention	More flexible, allowing for customization and detailed specification of routes
>	Easier to read and understand for simple routing needs	More descriptive; provides clarity about the intended routing pattern and default values.

Q.10 With appropriate examples, explain "Write()" and "WriteLiteral()" with respect to Razor view.

→ Write() and WriteLiteral() are methods from the 'TextWriter' class that are used to render content into the HTML output, but they handle content differently based on encoding.

1. 'Write()' method.

Used to write a string to the output stream. It automatically HTML-encodes the string, which means it converts characters like <, > and & into their respective HTML entities. This is useful for cross-site Scripting (XSS) attacks by ensuring that any potentially harmful input is safely rendered as text.

eg:
```
@ {
    var content = "<b> Hello World! </b>";
}
@Write(content)

// Output : &lt;b&gt; Hello World! &lt;/b&gt;
```

2. 'WriteLiteral()' method.

Used to write raw text to the output stream without HTML encoding. It is primarily used when you are sure that the content being written does not contain any HTML markup that could be harmful or when

you want to output literal strings without any transformation.

eg:
```
@{
    var htmlString = "<strong>Hello, world!</strong>";
}
@Write Literal (htmlString)

// Output : <strong> Hello, world !. </strong>
```

Q.11 Write code and explain the concept of "View Imports".

→ View Imports (_ViewImports.cshtml) is a special file that allows you to centrally define common directives, namespaces and configurations shared across all Razor views in a folder (and its subfolders). This eliminates the need to repeatedly declare them in individual Razor views files.

eg:
```
@using WebApp.Models
@using Microsoft.AspNetCore.Mvc.Rendering
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

→ @using Directives : Declares namespaces like in the example, WebApp.Models, etc. These namespaces are now available in all views under this folder.

→ @addTagHelper : This line enables tag helpers (form, input, label) provided by Microsoft.AspNetCore.Mvc.TagHelpers.

These tag helpers enhance standard HTML elements with ASP.NET Core features (automatically binding form elements to models)

eg:
```
@model WebApp.Models.Product
<h2> @Model.Name </h2>
<form asp-action = "Save" >
    <label asp-for = "Name" > </label>
    <input asp-for = "Name" />
</form>
```

★ You don't need to include @using WebApp.Models in Index.cshtml because they are already defined in _ViewImports.cshtml.

Q.12 With respect to Razor Views, explain @switch and @foreach with appropriate example and code.

—ᵛ @switch and @foreach are two control structures used for conditional rendering and iteration, respectively. They allow you to create dynamic content based on certain conditions or to loop through collections.

1. Using '@switch'

used for conditional branching, similar to the 'switch' statement in C#. It allows you to execute different code blocks based on the value of an expression.

eg:

```
@{
    var status = "InProgress";
}
@switch (status) {
    case "Completed":
        <p> Status : Task is Completed </p>
        break;
    case "InProgress":
        <p> Status : Task is in progress </p>
        break;
    case "Pending":
        <p> Status : Task is Pending </p>
        break;
    default:
        <p> status : Unknown <p>
        break;
}
```

2. using '@foreach'.

Used to iterate over a collection of items, rendering HTML for each item in the collection.

eg:

```
@ {
    var tasks = new List<string> { "Task 1", "Task 2",
                                   "Task 3" };
}
<ul>
    @foreach (var task in tasks) {
        <li> @task </li>
    }
</ul>
```

Q.13 What is model binding ? Where does the default model binder looks for data values and in which sequence ? Discuss.

-4 Model Binding is the process by which ASP.NET core takes incoming data (such as form data, query strings, route data) and automatically converts it into .NET objects (i.e. models). This allows developers to focus on the logic of the application without manually retrieving or parsing data.

For example, when a form is submitted, the model binder takes the input values from the form and populates the properties of a model, which is passed to the action method.

-4 The default model binder looks for data values in the following sequence (high to low)

i) Form Data : The data sent via HTTP POST, typically from <form> elements (input, select, textarea).

If the form posts data with 'name = "FirstName", the model binder tries to map it to a model property named 'FirstName'.

ii) Route Data: Values provided as part of the URL, defined in the route template.

If the URL is '/Products/Details/5', the value '5' would be mapped to an 'id' parameter in the action method

iii) Query String : Data passed via the URL's query string (for HTTP GET requests).
For ?name = TonyMikaelson, the model binder looks for a matching property 'name' in the model or action parameters.

iv) Header Data : Sometimes, custom headers can be used as part of model binding. Although less common, headers like 'Authorization' or custom headers can be part of the binding process.

v) Cookies : Data from browser cookies can also be included in model binding.

Q.14 Define Model Binding. Explain binding simple data types with an example and code snippet.

→ Model Binding same in Question -13

→ Model Binding can bind simple data types like integers, strings, booleans, etc. from HTTP requests directly to action method parameters or model properties. These values typically come from form fields, route data or query strings.

eg: public IActionResult GetProductInfo(int id, string name) {
return Content($"Product ID : {id},

Product Name : {name}");
}

—> The parameters 'id' (an integer) and 'name' (a string) are simple data types. The model binder looks for these values in the request (query string, route data or form data).
If the URL is : 'https://localhost:5001/Product/GetProductInfo?id=10&name=Laptop'. The 'id' will be bound to "10" and 'name' will be bound to "laptop".

// Output : Product ID : 10, Product Name : Laptop.

Q.15 Explain binding complex data type with an example and code snippet.

—> Model Binding can also handle complex data types, which are typically represented by classes with multiple properties. When binding complex types, the model binder maps incoming request data to the properties of the class based on the names of the properties.

eg: Product.cs :

```
public class Product {
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
}
```

we can bind this 'Product' object directly in an action method in a controller. The model binder will map incoming form data or query string parameters to the properties of this object.

Binding from Form Data:

View (HTML Form):

```
<form method="POST" action="/Product/Create">
    <label for="Id">Product ID:</label>
    <input type="text" id="Id" name="Id" /><br>
    <label for="Name">Name:</label>
    <input type="text" id="Name" name="Name" /><br>
    <label for="Price">Price:</label>
    <input type="text" id="Price" name="Price" /><br>
    <button type="submit">Create Product</button>
</form>
```

Action Method in Controller:

```
[HttpPost]
public IActionResult Create(Product product)
{
    // Model binding automatically maps the form
       values to the Product Object.
    return Content($"Product ID: {product.Id},
    Name: {product.Name}, Price {product.price}");
}
```

// Output: Product ID: 1, Name: Laptop,
Price: 1500.00
(Assuming form submits these values)

Q.16 What is model validation? Why do we need input data validation? Explain with two different examples.

-4 Model validation is a process that checks the incoming data against defined rules or constraints before it is processed in the application. It ensures that the data meet certain criteria, such as format, range or required fields.

-4 Input data validation is crucial for:

i) Data Integrity: Ensures that the data being processed and stored in the app is valid, preventing inconsistencies.

ii) Security: Protects the app from malicious input that could lead to security vulnerabilities like SQL injection or XSS.

iii) User Experience: Provides immediate feedback to users when they submit invalid data, improving overall user experience.

eg: 1) Required Field Validation

Model with validation Attributes:

```
public class UserProfile {
    [Required (ErrorMessage = "Username Required")]
    public string UserName {get; set;}
    [Required (ErrorMessage = "Email Required")]
```

```
    [EmailAddress(ErrorMessage = "Invalid")]
    public string Email {get; set;}
}
```

eg: 2) Range Validation

```
Model with range validation attributes
public class Product {
    public int ProductId {get; set;}
    [Required(ErrorMessage = "Product Name Required")]
    public string ProductName {get; set;}
    [Range(0.01, 10000.00, ErrorMessage = "Invalid")]
    public decimal Price {get; set;}
}

Controller Action
public class ProductsController : Controller {
    [HttpPost]
    public IActionResult Create(Product product) {
        if (!TryValidateModel(Product)) {
            return View(product); // Validation fails
        }
        return RedirectToAction("Index"); // Valid data
    }
}
```

Q.17 What is implicit validation in ASP.NET Core? Explain two basic checks performed by implicit validation.

→ Implicit validation is the automatic validation that occurs without the need for explicit checks

checks in the controller code. It happens when the model binding process takes the incoming request data and binds it to a model. ASP.NET Core automatically validates the model against the validation attributes ([Required], [Range], etc.) applied to its properties. If the validation fails, the errors are stored in the 'ModelState' object

* Examples are same as Question - 16

Q.18 Explain "SetMissingBindRequiredValueAccessor" and "SetValueMustBeANumberAccessor" with appropriate examples.

-> These methods are used ~~to~~ for customizing model binding behaviour. They help define how the model binder should handle certain scenarios, particularly regarding invalid or missing values.

1. "SetMissingBindRequiredValueAccessor"

Used to define a custom accessor that will be called when a required value is missing during model binding. This allows you to specify how to handle cases where a required property is not provided in the incoming request

eg: Model class:

```
public class User {
```

```
    [Required (ErrorMessage = "Name Required")]
    public string Name { get; set; }
}
```

> Custom Model Binder:

```
public class ModelBinder : IModelBinder {
    public Task BindModelAsync (ModelBindingContext
                                bindingContext) {
        var nameValue = bindingContext.ValueProvider.GetValue("Name").FirstValue;
        if (String.IsNullOrEmpty(nameValue)) {
            nameValue = "Default Name";
        }
        bindingContext.Result = ModelBindingResult.
                Success(new User {Name = nameValue});
        return Task.CompletedTask;
    }
}
```

2. "SetValueMustBeANumberAccessor":

Used to define a custom accessor that is invoked when the value for a property must be a number. It allows you to handle cases where the incoming data is expected to be numeric but is not.

eg:

```
Model Class:
public class Product {
    public int ProductId { get; set; }
    [Required (ErrorMessage = "Price must be a Number")]
```

```
    public decimal Price { get; set; }
}
```

Custom Model Binder :

```
// Same code as the previous one except
if (! decimal.TryParse (priceValue, out
                        var price)) {
    bindingContext.ModelState.AddModelError(
    "Price", "Price must be a Number.");
    return Task.CompletedTask;
}
```