# Short Answer:

Answer the following questions with complete sentences in your own words. You are encouraged to conduct your own research online or through other methods before answering the questions. If you research online, please consult multiple sources before you write down your answers. You are expected to be able to explain your answers in detail (Provide examples to each question).

1. What is Session?

Session

● Session state is an ASP.NET Core scenario for the storage of user data while

the user browses a web app

● Session is accessed via HttpContext.Session

● Session cannot be shared across browsers

A session is a way to store information (in variables) to be used across multiple pages. In web development, a session refers to the period of time a user interacts with a website, from the time the user opens the website to the time the user leaves the website. Each user is assigned a unique session ID, which is stored on the server and used to track the user's activity during the session. A session is typically implemented using cookies, which store the session ID on the user's computer, allowing the server to identify the user with each request.

1. What is Cookie?

Cookie

● Cookie is a small piece of information that is sent by the web server in the

response header and gets stored in the browser cookies

● It is a key value pair sent by the server to the client

● Cookies store data across requests.

● This pair is automatically attached with every request to the server, so their

size should be kept to a minimum

● Cookies are often used for personalization, where content is customized for a

known user

A cookie is a small piece of data that a web server stores on a client's device (usually a web browser) when the user visits a website. Cookies are used to remember user preferences, login information, and other data that the server can access each time the user visits the website. This allows the website to provide a more personalized experience for the user, such as remembering the items in their shopping cart or the pages they have visited. Cookies are typically stored on the client's device as text files and can be accessed by the server when the user makes a request to the website.

1. Difference between Authentication and Authorization
2. Session: A session is a server-side storage of information that is specific to a user session. It allows the web application to remember specific information about a user across multiple requests. The information stored in a session is identified by a session ID, which is typically stored in a cookie on the client-side.
3. Cookie: A cookie is a small text file that is stored on a user's device by a web application. Cookies can be used to remember user-specific information such as login credentials, preferences, or shopping cart contents. They are also used to track user behavior and personalize the user experience. Cookies are sent back and forth between the web application and the user's device with each request.
4. Authentication is the process of verifying a user's identity. It's typically done by asking the user to provide some sort of credentials, such as a username and password, and then checking those credentials against a database of authorized users. Once the user is authenticated, the application knows that the user is who they claim to be.

Authorization is the process of granting or denying access to specific resources or functionality based on the authenticated user's role or permissions. It's typically done by checking the authenticated user's role or permissions against a set of rules or policies. Once the user is authorized, the application knows what the user is allowed to do.

1. Explain Cookie Authentication

Cookie Authentication

If the user information is stored in the cookie, it is not safe, because this

information will be stored on the client side (on the browser).

For security reasons, the user information is saved in the session on the server side,

but the session id will be stored in the cookie so that when the cookie is passed

between the browser and the server, the server can find the corresponding session

through the session id in the cookie, so that the user does not have to log in again

and the website can recognize the user.

● Session cookies are deleted when the browser session ends.

● If a cookie is received for an expired session, a new session is created that uses

the same session cookie

Cookie Authentication

Cookie authentication in ASP.NET Core web application is the popular choice for

developers to implement authentication in most customer-facing web applications and is

also easy to implement in ASP.NET Core as it is provided out of the box without the

need to reference any additional NuGet packages.

ASP.NET Core provides a cookie authentication mechanism which on login serializes

the user details in form of claims into an encrypted cookie and then sends this cookie

back to the server on subsequent requests which gets validated to recreate the user object

from claims and sets this user object in the HttpContext so that it is available & is valid

only for that request

Diagram

Description automatically generated

Cookie authentication is a way of authenticating users by storing a small piece of data, called a cookie, on the user's browser. When the user makes a request to a web application, the server checks for the presence of the cookie and uses its contents to authenticate the user. The cookie typically contains a unique identifier for the user, and may also include other information such as their login status, session data, or permissions. Cookie authentication is often used in conjunction with other forms of authentication such as forms authentication or OAuth. It's a good way to handle session based authentication, where the server doesn't have to keep track of the user's session, it rely on client's cookie. The disadvantage is that it can be easily tampered, and It's not recommended for sensitive information.

1. How to implement Cookie Authentication?

Using Cookie Authentication

1. Add authentication middleware with the AddAuthentication and AddCookie

methods

2. Specify the app must use authentication and authorization

3. Apply [Authorize] attribute on the controllers and actions that require the

cookie authorization

Step 1: Add Authentication Middleware

In your Program.cs file:

Graphical user interface, text

Description automatically generated

On .NET 5.0 or the previous version, we have to do this configuration inside the Startup.cs

and inside its ConfigureServices() method

Cookie Authentication Options

● Cookie.Name – Name of the Cookie

● LoginPath – is used by the handler for the redirection target when handling

ChallengeAsync

● SlidingExpiration – is set to true to instruct the handler to re-issue a new

cookie with a new expiration time any time it processes a request

● AccessDeniedPath – is used by the handler for the redirection target when

handling ForbidAsync

Authentication Properties Options

● IsPersistent – Gets or sets whether the authentication session is persisted

across multiple requests.

● ExpiresUtc – Gets or sets the time at which the authentication ticket

expires.

● AllowRefresh – Gets or sets if refreshing the authentication session should

be allowed.

● IssuedUtc – Gets or sets the time at which the authentication ticket was

issued.

Step 2: UseAuthentication() & UseAuthorization()

In your Program.cs file:

Graphical user interface, text, application, email, website

Description automatically generated

On .NET 5.0 or earlier versions, those will be included within Configure() method of

Startup.cs. This tells the App to use the authentication & authorization

Step 3: Apply [Authorize] attribute

Graphical user interface, text, application, chat or text message

Description automatically generated

Text

Description automatically generated

Claims

● Claims are the foundation behind claims-based authentication (who would have

guessed). A claim is simply a piece of information about a subject. A claim does not

dictate what a subject can, or cannot do.

● The term “subject” is used because claims are not restricted to only describing

users. Claims can be about an application, service, or device.

● Some examples of claims a subject may have are:

○ Username

○ Email

○ IP Address

○ Location

ClaimsIdentity

Claims representing the same subject can be grouped together and placed in a

ClaimsIdentity.

Graphical user interface, text, application

Description automatically generated

ClaimsPrincipal

A principal object represents the security context of the user on whose behalf

the code is running, including that user's identity (IIdentity) and any roles to

which they belong.

By using a ClaimsPrincipal we can group the user identity, and device identity

into one context without having to duplicate any info.

Diagram

Description automatically generated

Logout

Text

Description automatically generated

Login/Logout View

Text

Description automatically generated

In ASP.NET Core 6, the process of implementing cookie authentication has been made simpler and more streamlined. Here is a general outline of the steps to implement cookie authentication:

1. Install the Microsoft.AspNetCore.Authentication.Cookies package.
2. In the Startup.cs file, add the following line in the ConfigureServices method to add the authentication service:

.AddCookie(options => {

options.LoginPath = "/Account/Login";

options.AccessDeniedPath = "/Account/AccessDenied";

});```

1. In the Configure method, add the following line to use the authentication service: **app.UseAuthentication();**
2. Create a Login action in the AccountController, which will handle the login process, including validating the user's credentials and creating the cookie.
3. Create a Logout action in the AccountController that will handle the logout process and delete the cookie.
4. Decorate the controller or action methods that you want to protect with the [Authorize] attribute.
5. On the views that you want to protect, use the built-in AuthorizeTagHelper, which will check if the user is authenticated and redirect him to the login page if not.

Once all these steps have been completed, your application should now have basic cookie authentication implemented. The actual implementation of the login and logout process would be specific to the requirements of the application.

1. What does [Authorize] attribute do?

[Authorize] Attribute

● When we place the Authorize attribute on the controller itself, the authorize

attribute applies to all of the actions inside.

● The MVC framework will not allow a request to reach an action protected by

this attribute unless the user passes an authorization check.

● By default, if you use no other parameters, the only check the Authorize

attribute will make is a check to ensure the user is logged in, so we know

their identity.

● But you can use parameters to specify any fancy custom authorization policy

that you like.

Graphical user interface, text, application

Description automatically generated

Diagram

Description automatically generated

The [Authorize] attribute is a built-in attribute in ASP.NET Core that is used to enforce authentication on an action method or a controller class. When this attribute is applied to an action method or a controller, the application will only allow requests that are authenticated to execute that action method or access that controller. This attribute can be used to restrict access to certain functionality to only authenticated users. It can also be used to specify required roles or other claims that the authenticated user must have in order to access the action method or controller. It is a way to secure your web application by applying the authorization rule on actions or controllers, so that only users with valid claims can access them.

1. What is Filter?

Filter

In MVC, controllers have many action methods and these action methods

generally have a one-to-one relationship with UI controls such as clicking a

button or a link.

But many times we would like to perform some action before or after a

particular operation. For achieving this functionality, ASP.NET MVC provides

feature to add pre and post action behaviors on controller's action methods.

Filter Types

ASP.NET/ASP.NET Core MVC framework supports the following filter types:

● Authorization filters:

○ Are the first filters run in the filter pipeline.

○ Control access to action methods.

○ Have a before method, but no after method.

● Resource filters:

○ Implement either the IResourceFilter or IAsyncResourceFilter interface.

○ Execution wraps most of the filter pipeline.

○ Only Authorization filters run before resource filters.

● Action filters:

○ Implement either the IActionFilter or IAsyncActionFilter interface.

○ Their execution surrounds the execution of action methods.

Filter Types

ASP.NET/ASP.NET Core MVC framework supports the following filter

types:

● Exception filters apply global policies to unhandled exceptions that occur

before the response body has been written to.

● Result filters:

○ Run immediately before and after the execution of action results.

○ Run only when the action method executes successfully.

○ Are useful for logic that must surround view or formatter execution.

Timeline

Description automatically generated with medium confidence

Filter Implementation (Action Filter)

A screenshot of a computer

Description automatically generated with medium confidence

A filter in ASP.NET Core is a component that provides some sort of functionality for certain stages of the request processing pipeline. Filters can be used for tasks such as logging, caching, error handling, and authentication. Filters can be applied globally, to a specific controller, or to a specific action method.

1. What are the types of Filter? What is the execution order of ﬁlters?

Filters are a way to add pre-processing or post-processing logic to actions in an ASP.NET Core MVC application. There are several types of filters, including:

* Action filters: These filters run before and after an action is executed and can modify the action's execution flow.
* Authorization filters: These filters determine whether the user is authorized to perform an action and can short-circuit the action if the user is not authorized.
* Result filters: These filters run before and after the action result (e.g. a view) is executed.
* Exception filters: These filters run if an exception is thrown during the execution of an action or the result.

The order of execution for filters is:

1. Authorization filters
2. Resource filters
3. Action filters
4. Exception filters
5. Result filters

Note that if a filter's behavior is dependent on the order it is executed, you can also specify a filter's ordering explicitly by implementing IAsyncAuthorizationFilter, IAsyncResourceFilter, IAsyncActionFilter, IAsyncResultFilter, IAsyncExceptionFilter and IOrderedFilter respectively.

1. What is Razor? What does it do?

Outline

● Razor

● Razor Syntax

● Razor Directive

● Flow Control in Razor

● Comment in Razor

● Razor Pages

● Razor Pages vs. MVC in ASP.NET

What is Razor

● Razor is a markup syntax that lets you embed server-based code (Visual Basic

and C#) into web pages.

● Server-based code can create dynamic web content on the fly, while a web page

is written to the browser. When a web page is called, the server executes the

server-based code inside the page before it returns the page to the browser. By

running on the server, the code can perform complex tasks, like accessing

databases.

● Razor is based on ASP.NET, and designed for creating web applications.

Razor Syntax

● The Razor syntax consists of Razor markup, C# and HTML.

● Files containing Razor generally have a .cshtml file extension.

● Razor supports C# and uses the @ symbol to transition from HTML to C#.

Razor evaluates C# expressions and render them in the HTML output.

● Razor epressions start with @ followed by C# code

Text

Description automatically generated

Razor Syntax

Razor Blocks

● Razor blocks start with @ and are enclosed by {}

● Unlike Razor expressions, C# code inside the block isn’t rendered.

● Code blocks and expressions in a view share the same scope and are defined in

order.

Graphical user interface, application, Teams

Description automatically generated

Razor Syntax

Razor Blocks

● The default language in a code block is C#, but the Razor Page can transition

back to HTML.

● To render the HTML in a block we can use tags, only the content between

<tag> is rendered; The @: syntax also works

Graphical user interface, application, Teams

Description automatically generated

Razor Directives

Razor directives are represented by implicit expressions with reserved keywords

following the @ symbol. A directive typically changes the way a view is parsed or

enables different functionality.

● @page

The @page directive in a .cshtml file indicates that the file is a Razor page

● @using

The @using directive adds the C# using directive to the generated view

i.e. if you put @using System.IO in your razor page, then you can use all the

methods within System.IO namespace at this page.

Razor Directives

● @model

The @model directive specifies the type of model passed to a view or page, and

Razor exposes a Model property for accessing the model passed to the view

Graphical user interface, text, application

Description automatically generated

Razor is a markup syntax for embedding server-based code into web pages. It allows you to use C# (or Visual Basic) code in an HTML file by prefixing it with the "@" symbol. Razor enables a developer to write server code in HTML and to combine server code with HTML in the same file. This allows for a cleaner separation between the presentation and the logic, and it enables developers to use their existing HTML and CSS skills to create dynamic web applications. Razor is typically used to create views in an ASP.NET MVC or ASP.NET Core application.

1. What does @model mean?

Razor Directives

● @model

The @model directive specifies the type of model passed to a view or page, and

Razor exposes a Model property for accessing the model passed to the view

Graphical user interface, text, application

Description automatically generated

In an ASP.NET Core MVC application, the **@model** directive in a Razor view is used to specify the type of the model object that the view expects to receive. This allows the view to access the properties and methods of the model, and display them in the HTML. The **@model** directive should be placed at the top of the view file and is followed by the fully qualified name of the model class. It is used by the MVC framework to ensure that the correct model object is passed to the view when it is rendered.

1. What are tag helpers in ASP.NET Core? Can you give some examples?

Tag Helper

● In ASP.NET Core, Tag Helpers are a way to create custom HTML tags that can

be used within Razor views.

● They allow developers to write HTML directly in Razor views while providing a

way to include server-side logic that can modify the HTML before it is rendered.

● The Tag Helpers start with an asp- prefix



Tag Helper

● asp-controller and asp-action attributes are used in conjunction with the asproute attribute to generate URLs in Razor views.

● asp-controller specifies the controller’s name that handles the request when the

link is clicked.

● asp-action specifies the name of the action method that is called on the controller

when the link is clicked.



In ASP.NET Core, tag helpers are a way to add server-side logic to HTML elements in Razor views. They allow developers to create custom HTML elements and attributes that are executed on the server before the view is rendered to the client. Tag helpers are similar to HTML helpers in earlier versions of ASP.NET MVC.

Examples of built-in tag helpers in ASP.NET Core include the **<form>**, **<label>**, and **<input>** tag helpers, which are used to create forms and input elements. Here's an example of how the **<form>** tag helper is used:

<form method="post" asp-controller="Account" asp-action="Create">

<!-- input fields here -->

<input type="submit" value="Create Account" />

</form>

In the above example, the **asp-controller** and **asp-action** attributes are used to specify the target controller and action method that the form should be submitted to.

Another example of a built-in tag helper is the **<a>** tag helper, which is used to create hyperlinks. Here's an example of how the **asp-controller** and **asp-action** attributes can be used to create a link to a specific action method:

<a asp-controller="Home" asp-action="About">Learn More</a>

Developers can also create their own custom tag helpers by creating a class that inherits from **TagHelper** and adding the desired functionality.

# Coding Questions:

1. Create a table in your database, **user(id | username | password | email)**
2. Build a Web Application (MVC), implementing the user login function