ASP .Net Web Forms with C#

Basics and Fundamentals

1. **What is ASP.NET Web Forms?**
    - ASP.NET Web Forms is a part of the ASP.NET framework used to build dynamic web applications. It provides a rich set of server-side controls and components, allowing developers to design web pages using drag-and-drop elements with event-driven programming.
2. **Explain the ASP.NET Page Life Cycle.**
    - The ASP.NET page life cycle consists of stages such as Initialization, Load, PostBack Event Handling, Rendering, and Unload. Each stage provides the opportunity to inject custom logic to handle the page's lifecycle effectively.
3. **What are Postback and ViewState in ASP.NET Web Forms?**
    - A **Postback** occurs when the page is submitted to the server for processing (e.g., when a button is clicked). **ViewState** is used to preserve the values of form controls between postbacks.
4. **What is the difference between Web Forms and MVC?**
    - Web Forms follow an event-driven development model with drag-and-drop controls, whereas MVC follows a separation of concerns, providing more control over HTML and allows for more flexible design patterns.
5. **What is the purpose of Global.asax in ASP.NET?**
    - Global.asax is a file used to handle global application-level events such as Application_Start, Session_Start, Application_End, etc., and for managing HTTP requests.
6. **What is the role of web.config in an ASP.NET Web Forms application?**
    - web.config is used to configure settings for the application such as database connections, authentication, authorization, custom error pages, and more.
7. **What are Server Controls in ASP.NET Web Forms?**
    - Server Controls are controls that run on the server and encapsulate logic for handling HTML rendering and events. Examples include TextBox, Button, DropDownList, etc.
8. **What are the different types of server controls in ASP.NET?**
    - There are three types of server controls: HTML controls, Web server controls, and User controls.

9. What is the difference between Server.Transfer and Response.Redirect?
    - Server.Transfer transfers the execution to another page on the server without changing the URL, while Response.Redirect sends an HTTP redirect to the browser, causing a new request and a visible change in the URL.
10. What is the use of the IsPostBack property?
    - IsPostBack is a property that indicates whether the page is being loaded in response to a client postback (true) or being loaded for the first time (false).

State Management

11. What are the different ways to manage state in ASP.NET Web Forms?
    - State can be managed using ViewState, Session, Cookies, Query Strings, Hidden Fields, Application state, and Cache.
12. What is ViewState, and how does it work in ASP.NET Web Forms?
    - ViewState is used to retain the values of server controls between postbacks. It stores data in a hidden field on the client side and is sent back to the server with each postback.
13. What are the limitations of ViewState?
    - ViewState can increase page size because data is serialized and stored in a hidden field. It is not suitable for storing large amounts of data or sensitive information.
14. How can you disable ViewState for a control?
    - You can disable ViewState for a control by setting the EnableViewState property to false.
15. What is Session State?
    - Session state allows storing user-specific data on the server for the duration of the user's session. It is used for data that needs to persist across different pages of the application.
16. How can you configure Session State in ASP.NET Web Forms?
    - Session state can be configured in the web.config file. You can define its mode (InProc, StateServer, SQLServer), timeout, and whether it is cookieless.

17. What is the difference between Session State and ViewState?
    - Session State stores data on the server and persists across multiple pages, while ViewState stores data on the client side and only lasts during postbacks on the same page.
18. What are Cookies in ASP.NET Web Forms?
    - Cookies are small pieces of data stored on the client side, used to maintain information across multiple page requests.
19. What is Application State in ASP.NET Web Forms?
    - Application State is a global storage mechanism that can store data across all users and sessions for the application. It is stored in memory on the server.
20. What is the difference between Session State and Application State?
    - Session State is specific to a user session, while Application State is shared among all users and sessions across the application.

Controls and Events

21. What is the difference between a HyperLink and a LinkButton?
    - A HyperLink is a simple anchor (<a>) tag that does not trigger a postback, whereas a LinkButton is a server control that triggers a postback when clicked.
22. Explain the difference between AutoPostBack and IsPostBack.
    - AutoPostBack is a property that automatically triggers a postback when an event occurs on a control, while IsPostBack checks if the page is being loaded in response to a postback.
23. How can you implement a Master Page in ASP.NET Web Forms?
    - Master Pages allow consistent layout across multiple pages. You define a Master Page with placeholders, and content pages can fill in the placeholders.
24. What is a User Control in ASP.NET Web Forms?
    - User Controls are reusable components, created using .ascx files, that encapsulate UI and functionality, and can be reused across multiple pages.
25. How do you pass data between different Web Forms?
    - Data can be passed using query strings, session variables, cookies, or server-side techniques like Server.Transfer.

26. What is an UpdatePanel in ASP.NET Web Forms?
    - UpdatePanel is a control used to perform partial page updates asynchronously using AJAX, reducing the need for full-page postbacks.
27. What is the GridView control in ASP.NET Web Forms?
    - GridView is a data-bound control used to display, edit, and sort tabular data. It can be bound to a data source such as a database or an object.
28. What is the Repeater control in ASP.NET?
    - Repeater is a lightweight, data-bound control that allows you to define a custom layout for displaying repeated data. Unlike GridView, it provides more control over the rendering of HTML.
29. What is a DataList control, and how is it different from GridView?
    - DataList is a control similar to Repeater, but it provides more functionality such as editing, updating, and deleting records. It has templates for layout customization.
30. What is the difference between the DataList and Repeater controls?
    - DataList offers more built-in features like editing and layouts, while Repeater is more lightweight and gives more control over HTML generation without extra features like edit and update.

Data Access

31. How do you connect to a database in ASP.NET Web Forms?
    - You connect to a database using ADO.NET objects like SqlConnection, SqlCommand, and SqlDataReader for querying, reading, and manipulating data.
32. What are the main ADO.NET objects used to interact with a database?
    - The primary objects are SqlConnection, SqlCommand, SqlDataAdapter, DataSet, DataReader, and DataTable.
33. What is the use of a SqlDataReader in ASP.NET?
    - SqlDataReader provides a fast, forward-only, read-only cursor for accessing data from a database.
34. What is the difference between SqlDataAdapter and SqlDataReader?
    - SqlDataAdapter is a disconnected, bulk data retrieval method used to fill DataSets and DataTables, while SqlDataReader is a connected, fast, forward-only reader.

35. How do you handle database exceptions in ASP.NET Web Forms?
    - You handle exceptions using try-catch blocks around your database logic and log or display the error message for further analysis.
36. What is Entity Framework, and how does it differ from ADO.NET?
    - Entity Framework is an ORM (Object-Relational Mapper) that simplifies database access by allowing developers to work with data as objects, while ADO.NET requires manual SQL queries and connection management.
37. How can you bind data to a GridView control in ASP.NET Web Forms?
    - You can bind data to a GridView by setting its DataSource property to a data source (such as a DataSet or List) and calling the DataBind method.
38. How do you implement sorting and paging in a GridView?
    - Sorting can be enabled by setting the AllowSorting property to true and handling the Sorting event. Paging can be enabled by setting the AllowPaging property to true and handling the PageIndexChanging event.
39. What is a Connection Pool, and how does it help with performance?
    - A connection pool is a cache of database connections maintained so that they can be reused, reducing the overhead of opening and closing connections frequently.
40. How can you use LINQ with ASP.NET Web Forms?
    - LINQ can be used in ASP.NET Web Forms to query and manipulate data collections such as arrays, lists, and databases (via LINQ to SQL or Entity Framework).

Security and Authentication

41. What are the different types of authentication available in ASP.NET Web Forms?
    - The types of authentication are Windows Authentication, Forms Authentication, and Passport Authentication.
42. How does Forms Authentication work in ASP.NET?
    - Forms Authentication allows users to log in with a username and password. After successful authentication, a cookie is issued to the client to track authenticated requests.
43. What is the role of the FormsAuthentication class?
    - The FormsAuthentication class provides methods for managing forms-based user authentication, including login, logout, and cookie management.

44. How do you implement Authorization in ASP.NET Web Forms?
    - Authorization can be implemented using the web.config file, by specifying roles or users allowed to access certain parts of the application.
45. How can you secure sensitive data in ASP.NET Web Forms?
    - You can secure sensitive data by using SSL (HTTPS), encrypting sensitive data like passwords, and securing cookies with the Secure and HttpOnly attributes.
46. What is Cross-Site Scripting (XSS), and how do you prevent it in ASP.NET Web Forms?
    - XSS is an attack where malicious scripts are injected into web pages. You can prevent it by encoding user inputs using HttpUtility.HtmlEncode and using validation controls.
47. What is Cross-Site Request Forgery (CSRF), and how do you mitigate it?
    - CSRF is an attack where an attacker tricks a user into submitting a malicious request. You can mitigate it by using anti-CSRF tokens (RequestVerificationToken) and validating them on the server side.
48. What is Role-Based Authentication in ASP.NET Web Forms?
    - Role-based authentication restricts access to parts of the application based on user roles. You define roles and assign users to roles, then restrict access based on those roles using the Authorize attribute or web.config.
49. What is the use of Membership in ASP.NET Web Forms?
    - Membership provides built-in functionality to manage user accounts, including creating, validating, and managing users and roles in the application.
50. How do you implement OAuth or OpenID authentication in ASP.NET Web Forms?
    - OAuth and OpenID authentication can be implemented by integrating external authentication providers like Google, Facebook, or Microsoft. This involves configuring the external provider's authentication middleware in Startup.Auth.cs.

--==========

.NET Core Basics and Fundamentals

1. What is .NET Core? How is it different from .NET Framework?
    - .NET Core is a cross-platform, open-source framework for building modern applications. It's designed to work on Windows, Linux, and macOS, whereas the .NET Framework primarily works only on Windows. .NET Core also has better performance and modularity compared to the .NET Framework.
2. Explain the architecture of .NET Core.
    - .NET Core consists of three components: the **Runtime**, which executes applications; the **Framework Libraries**, which provide APIs and runtime capabilities; and the **SDK**, which contains tools for developing and running applications.
3. What are the key features of .NET Core?
    - Cross-platform support, open-source, high performance, modularity via NuGet packages, improved cloud support, and unified programming model for web, desktop, and mobile apps.
4. What is Kestrel in .NET Core?
    - Kestrel is a cross-platform web server for ASP.NET Core applications. It is lightweight, high-performance, and can be used as a web server to handle HTTP requests directly or behind a reverse proxy.
5. How does .NET Core achieve cross-platform compatibility?
    - .NET Core is designed to be platform-agnostic by running on different runtimes (e.g., CoreCLR for Windows, and the Mono runtime for Linux/macOS). It abstracts OS dependencies and uses platform-specific implementations.
6. What is the .NET Standard? How is it related to .NET Core?
    - .NET Standard is a formal specification of APIs that all .NET platforms must implement, including .NET Core, .NET Framework, and Xamarin. It helps ensure compatibility across different .NET environments.
7. What is the difference between .NET Core and Mono?
    - Mono is an open-source implementation of .NET designed to run on multiple platforms, focusing on mobile and gaming development. .NET Core is a modern, optimized version of the .NET platform designed for cloud, microservices, and cross-platform use.
8. What are the different types of applications that can be built using .NET Core?
    - You can build web apps (using ASP.NET Core), desktop apps (using Windows Forms and WPF), mobile apps (via Xamarin), cloud services, microservices, IoT apps, and more.
9. What is a runtime in .NET Core? How does it differ from a framework?
    - A runtime (like CoreCLR or Mono) is responsible for executing .NET Core applications, including managing memory, handling garbage collection, and JIT compilation. A framework provides libraries and APIs for developers to use, while the runtime manages execution.
10. Explain the role of the SDK in .NET Core.
    - The SDK provides the tools for building, testing, and deploying .NET Core applications, including the dotnet CLI, compilers, and libraries.

ASP.NET Core

11. What is ASP.NET Core? How is it different from ASP.NET?
    - ASP.NET Core is a redesign of ASP.NET, optimized for cloud, cross-platform development. It's modular, faster, and has better support for modern web development features like dependency injection and middleware.
12. Explain Middleware in ASP.NET Core.
    - Middleware are software components used to handle HTTP requests and responses in a pipeline. They can modify requests, responses, or pass them on to the next component in the pipeline.
13. What are Services and how do you use Dependency Injection in ASP.NET Core?
    - Services are reusable components that can be injected into ASP.NET Core classes like controllers. Dependency Injection (DI) is built into ASP.NET Core, and services are registered in the Startup class, typically within the ConfigureServices method.
14. What is the Startup class in ASP.NET Core, and what are its responsibilities?
    - The Startup class configures the application's services and request pipeline. It contains two main methods: ConfigureServices (for setting up DI) and Configure (for setting up middleware).
15. Explain the role of the Configure and ConfigureServices methods in ASP.NET Core.
    - ConfigureServices is used to register services with the DI container. Configure is used to set up the HTTP request pipeline with middleware components.
16. What is the difference between IHostedService and BackgroundService?
    - IHostedService is the base interface for implementing long-running services. BackgroundService is an abstract class that provides a base implementation for background tasks, simplifying the use of IHostedService.
17. How can you enable logging in ASP.NET Core?
    - Logging in ASP.NET Core is enabled by default and can be configured using ILogger<T> and third-party providers such as Serilog or NLog, in the Configure method in Startup or appsettings.json.
18. What is Model Binding in ASP.NET Core?
    - Model Binding is the process of extracting data from HTTP requests and converting it into .NET objects for use in controllers.
19. What are Tag Helpers in ASP.NET Core?
    - Tag Helpers are server-side components that enable server-side rendering in Razor views, making HTML more readable and maintainable by associating C# code with HTML elements.
20. Explain the concept of Routing in ASP.NET Core.
    - Routing is how ASP.NET Core matches incoming HTTP requests to specific controller actions. Routes can be defined using attributes or in Startup via UseRouting and MapControllers.

Performance and Optimization

21. What is Response Caching in ASP.NET Core?
    - Response Caching allows caching HTTP responses to improve performance. You can control it using the [ResponseCache] attribute or middleware to store responses for repeated requests.

22. How can you optimize the performance of an ASP.NET Core application?
    - Techniques include enabling response caching, using asynchronous programming, compressing responses, minimizing resource loading, and optimizing database queries with tools like Entity Framework Core.

23. What is Ahead of Time (AOT) compilation in .NET Core?
    - AOT compilation compiles .NET Core code into machine code before it is executed, reducing the startup time and improving performance in scenarios like microservices or serverless architectures.
24. How do you handle memory management in .NET Core applications?
    - .NET Core uses automatic memory management through garbage collection. Developers can further optimize memory by disposing of objects explicitly using Dispose or using statements and reducing object allocations.
25. Explain how garbage collection works in .NET Core.
    - Garbage collection (GC) in .NET Core is an automatic process that reclaims memory used by objects no longer referenced by the application. It works in generations (0, 1, and 2) to optimize performance.
26. What are some performance profiling tools available for .NET Core?
    - Tools include **dotnet-trace**, **dotnet-counters**, **dotnet-dump**, Visual Studio Profiler, and third-party tools like JetBrains dotTrace and RedGate ANTS.
27. What is in-process hosting, and how is it different from out-of-process hosting?
    - In-process hosting runs the ASP.NET Core app inside the IIS worker process for better performance. Out-of-process hosting runs the app as a separate process (using Kestrel), and IIS acts as a reverse proxy.
28. How can you implement global exception handling in an ASP.NET Core application?
    - You can use middleware such as the ExceptionHandlerMiddleware or custom middleware to catch and log exceptions globally across the application.
29. How do you use asynchronous programming in .NET Core to improve performance?
    - Use the async and await keywords in C# to write non-blocking code, especially for I/O-bound operations like database queries or web service calls, improving scalability and responsiveness.
30. What is the use of the async and await keywords in .NET Core?
    - async defines an asynchronous method, and await is used to asynchronously wait for a task to complete without blocking the thread, improving application performance.

Security in .NET Core

31. How does authentication work in ASP.NET Core?
    - Authentication in ASP.NET Core identifies the user or client making a request, commonly using middleware such as cookie authentication, JWT tokens, or OAuth 2.0.
32. What are the different types of authentication supported in ASP.NET Core?
    - ASP.NET Core supports cookie-based authentication, JWT Bearer Tokens, OAuth, OpenID Connect, API Keys, and custom authentication schemes.
33. Explain how JWT (JSON Web Token) works in ASP.NET Core.
    - JWT is a compact, URL-safe token that represents claims (like user information). ASP.NET Core validates these tokens for stateless authentication in APIs, ensuring secure access to resources.

34. How do you secure an ASP.NET Core Web API using JWT?
    - Use JWT middleware (Microsoft.AspNetCore.Authentication.JwtBearer) to validate tokens in the request header and ensure that only authenticated users can access API endpoints.
35. What is Cross-Site Request Forgery (CSRF)? How is it prevented in ASP.NET Core?
    - CSRF attacks trick users into making unwanted requests. In ASP.NET Core, CSRF prevention is built into the framework through anti-forgery tokens generated with the @Html.AntiForgeryToken() method in Razor views.
36. How do you implement HTTPS redirection in an ASP.NET Core application?
    - Use the UseHttpsRedirection middleware in the Startup.Configure method to automatically redirect HTTP requests to HTTPS.
37. What is Identity in ASP.NET Core?
    - ASP.NET Core Identity is a membership system for handling user authentication and authorization. It provides functionality for managing users, roles, and passwords.
38. How can you enable data protection in .NET Core?
    - .NET Core provides built-in data protection APIs (IDataProtectionProvider) for encrypting sensitive data, like cookies or tokens, to prevent tampering or exposure.
39. Explain how authorization policies work in ASP.NET Core.
    - Authorization policies in ASP.NET Core are a flexible way to implement role-based or claim-based access control. They are configured in Startup and enforced using the [Authorize] attribute.
40. What are CORS (Cross-Origin Resource Sharing), and how do you enable it in ASP.NET Core?
    - CORS is a security feature that allows or restricts web applications from accessing resources from different origins. It is enabled using the UseCors method and configured in Startup.

Deployment and Hosting

41. How do you deploy a .NET Core application to IIS?
    - Publish the .NET Core application using dotnet publish, install the ASP.NET Core hosting bundle on the IIS server, and configure the site in IIS with Kestrel as the backend server.
42. What is the role of the dotnet publish command in .NET Core deployment?
    - The dotnet publish command compiles the application and all its dependencies into a folder, preparing it for deployment on a server.
43. What is Docker, and how can you containerize a .NET Core application?
    - Docker is a containerization platform. To containerize a .NET Core app, create a Dockerfile, specify the base image, copy the app into the container, and define the runtime environment.
44. What is the difference between Framework-Dependent Deployment (FDD) and Self-Contained Deployment (SCD)?
    - FDD relies on the .NET runtime already being installed on the host machine, whereas SCD bundles the runtime with the application, making it self-contained.
45. How do you deploy a .NET Core application to a cloud platform like Azure or AWS?
    - In Azure, you can use Azure App Service or Azure Kubernetes Service (AKS) for deployment. For AWS, you can use AWS Elastic Beanstalk, AWS Lambda, or EC2 for deployment.

46. What is the use of the appsettings.json file in ASP.NET Core?
    - appsettings.json is used to store configuration settings, such as connection strings, API keys, and application settings, which can be accessed at runtime via the IConfiguration interface.
47. Explain the concept of environment-based configuration in .NET Core.
    - Environment-based configuration allows the application to load different settings based on the environment (Development, Staging, Production) using environment-specific JSON files like appsettings.Development.json.
48. What is a reverse proxy, and how is it used with ASP.NET Core?
    - A reverse proxy forwards client requests to backend services. In ASP.NET Core, Kestrel typically runs behind a reverse proxy like IIS, Nginx, or Apache for better security, load balancing, and SSL termination.
49. How can you configure a .NET Core application to run as a Windows Service or a Linux daemon?
    - On Windows, use the Microsoft.Extensions.Hosting.WindowsServices package. On Linux, create a systemd service to manage the .NET Core application as a background service.
50. What are the differences between publishing .NET Core applications on Windows, Linux, and macOS?
    - While the core concepts of publishing remain the same, platform-specific considerations like file paths, case sensitivity (Linux/macOS), and hosting (e.g., IIS for Windows, Nginx/Apache for Linux) need to be accounted for.
51. **What is the difference between IEnumerable and IQueryable in .NET Core?**

- IEnumerable is used for in-memory collections and performs LINQ-to-Objects queries.
- IQueryable is used for querying data from out-of-memory sources (like databases) and can defer execution, thus performing queries on the server-side(LINQ-to-SQL or LINQ-to-Entities).

52. What is Routing in ASP.NET Core?

Routing in ASP.NET Core is the process of matching incoming HTTP requests to controller actions and determining which action should handle the request. It plays a vital role in URL mapping and directing requests to the appropriate endpoints. The ASP.NET Core routing system uses route templates to define patterns for URLs.

53. **Explain the concept of Dependency Injection in .NET Core.**

- Dependency Injection (DI) is a design pattern where the objects or services that a class depends on are injected into the class, rather than the class creating the dependencies itself. DI is natively supported through the **DependencyInjection** library.

54. **What are the different types of DI (Dependency Injection) in .NET Core?**

- **Transient:**

  A new service instance is created for each object in the HTTP request.

  This is a good approach for the multithreading approach because both objects are independent of one another.

  The instance is created every time they will use **more memory** and **resources** and can have a **negative** impact on performance

  Utilize for the **lightweight** service with little or **no state**.

- **Scoped:**

  In this service, with every HTTP request, we get a new instance.

  The same instance is provided for the entire scope of that request.

  eg., if we have a couple of parameter in the controller, both object contains the same instance across the request

  This is a better option when you want to maintain a state within a request.

- **Singleton:**

  Only one service instance was created throughout the lifetime.
  Reused the same instance in future, wherever the service is required
  Since it's a single lifetime service creation, memory leaks in these services will build up over time.
  Also, it has memory efficient as they are created once reused everywhere.

55. What is Middleware in ASP.NET Core?
    Middleware is a piece of code in an application pipeline used to handle requests and responses.
    For example, we may have a middleware component to authenticate a user, another piece of middleware to handle errors, and another middleware to serve static files such as JavaScript files, CSS files, images, etc.
56. How do you configure services and middleware in ASP.NET Core?
    Services are configured in the ConfigureServices method in Startup.cs, while middleware is configured in the Configure method.
57. What is Kestrel?
    Kestrel is a cross-platform web server that is included with ASP.NET Core. It is designed to be lightweight and performant, handling web requests and responses efficiently.
58. How do you implement Global Exception Handling in ASP.NET Core?
    Exception handling can be globally implemented using middleware like UseExceptionHandler or UseDeveloperExceptionPage. Additionally, you can create custom middleware to catch and handle exceptions across the application.
59. Explain how asynchronous programming is implemented in .NET Core.
    Asynchronous programming is implemented using the async and await keywords. This allows I/O-bound operations (like file access or database queries) to run asynchronously, improving application responsiveness and scalability.

60. **What are the key features of ASP.NET Core for building microservices?**
    Key features include:
    - Built-in support for **REST APIs** and **gRPC**.
    - **Dependency Injection** for better modularity.
    - **Docker** support for containerized applications.
    - **Health checks** for monitoring microservice health.
    - **Polly** for resilience and fault-tolerant policies like retries, timeouts, and circuit breakers.
61. **What are Entity Framework Core migrations, and why are they used?**
    Migrations in Entity Framework Core are used to apply changes in the database schema as your model changes over time. They are useful for keeping the database in sync with your application's data model.
62. **What is the purpose of the Program.cs file in a .NET Core application?**
    The Program.cs file contains the **Main method**, which is the entry point for the application. It sets up the **Host** and configures services like logging, dependency injection, and configuration settings.
63. **Explain the difference between Build and Publish in .NET Core.**
    Build: Compiles the application and generates the necessary files for running it on the local machine, including assemblies, dependencies, and the compiled code.
    Publish: Compiles the application, and also prepares it for deployment by including all the necessary runtime files and assemblies for a specified target environment.
64. **How do you handle configuration in .NET Core applications?**
    .NET Core uses **appsettings.json** files for configuration. It also supports environment-specific configuration files like **appsettings.Development.json.** The **IConfiguration** interface is used to access configuration data within the application.
65. **What is the difference between IActionResult and ActionResult in ASP.NET Core?**
    - IActionResult is an interface, which allows flexibility in returning various types of results (e.g., views, JSON, files, etc.).
    - ActionResult is a concrete class that implements IActionResult. It provides built-in return types like ViewResult, JsonResult, and FileResult.
66. **What is the .NET Core CLI?**
    The .NET Core Command Line Interface (CLI) is a set of tools for building, running, and managing .NET Core applications from the command line. Some common commands include dotnet new, dotnet build, dotnet run, and dotnet publish.

--==========

MVC framework with RAZOR

Basic Level:

1. What is MVC architecture? Explain its components.
    - MVC (Model-View-Controller) is an architectural pattern used for developing web applications. It divides the application into three interconnected components:
        - Model: Represents the application's data and business logic. It handles data retrieval and manipulation.
        - View: The presentation layer that displays data from the Model to the user. It is responsible for rendering the user interface.
        - Controller: Acts as an intermediary between Model and View. It processes user input, interacts with the Model, and returns the appropriate View.
2. What is ASP.NET MVC? How is it different from ASP.NET Web Forms?
    - ASP.NET MVC is a web application framework developed by Microsoft that implements the MVC architectural pattern. It differs from ASP.NET Web Forms in several ways:
        - Request Handling: MVC uses a request-driven model, while Web Forms is event-driven.
        - Separation of Concerns: MVC has a clearer separation of concerns between data, UI, and control logic, making it more maintainable.
        - Control Over HTML: MVC allows developers to have greater control over HTML output, leading to better performance and SEO.
        - Testability: MVC is more test-friendly because it separates business logic from the UI.
3. What is Razor? How is it different from Web Forms view engine?
    - Razor is a markup syntax used in ASP.NET to create dynamic web pages with C# code embedded within HTML. The differences between Razor and Web Forms view engine include:
        - Syntax: Razor uses @ to switch between HTML and C#, while Web Forms uses <% %>.
        - Simplicity: Razor has a cleaner and more concise syntax compared to Web Forms.
        - No ViewState: Razor does not use ViewState, resulting in reduced page size and improved performance.
4. What is the purpose of a controller in MVC?
    - The controller in MVC serves as an intermediary between the Model and View. Its primary responsibilities include:
        - Receiving user input and actions from the View.
        - Interacting with the Model to retrieve or manipulate data.
        - Selecting the appropriate View to render based on user actions and application logic.
5. Explain the role of the Model in MVC.
    - The Model in MVC is responsible for:
        - Representing the data and business logic of the application.
        - Managing data access, validation, and business rules.
        - Handling the state of the application and notifying the View of changes.
        - It often interacts with the database to perform CRUD (Create, Read, Update, Delete) operations.

6. What are Views in MVC? How are they rendered?
    - Views in MVC are responsible for displaying data to users. They are created using Razor syntax, allowing the integration of C# code within HTML.
    - Rendering Views:
        - When a controller action returns a ViewResult, the associated View is located based on the action name and controller.
        - The View is then processed by the server, where the Razor code is executed to produce HTML, which is sent to the client's browser.
7. What is the difference between ViewData and ViewBag in MVC?
    - Both ViewData and ViewBag are used to pass data from the controller to the View, but they differ in:

        - ViewData: A dictionary of key-value pairs. It is case-sensitive and requires type casting to access values.

```
ViewData["Message"] = "Hello, World!";
```

        - ViewBag: A dynamic property that provides a more flexible way to pass data without needing type casting.

```
ViewBag.Message = "Hello, World!";
```

8. Explain the concept of TempData in MVC. How does it differ from ViewData?
    - TempData is used to store data temporarily for the duration of a single request, primarily for passing data between actions.
    - Differences from ViewData:
        - Lifetime: ViewData exists only for the current request, while TempData persists until it is read or expires after the next request.
        - Use Case: ViewData is used for displaying data in the current view, while TempData is used for passing messages or data that need to persist across redirects.

```
TempData["Message"] = "Your changes have been saved!";
```

9. What is routing in MVC? How does it work?
    - Routing in MVC is the mechanism that maps incoming requests to the appropriate controller actions. It allows for clean, user-friendly URLs.
    - How It Works:
        - The routing table (defined in the RouteConfig.cs file) checks incoming requests against defined routes.
        - A route typically follows the pattern {controller}/{action}/{id}.
        - The routing engine extracts values from the URL and determines the controller and action to invoke.
10. What is the use of Html.RenderPartial vs Html.Partial in Razor?
    - Both Html.RenderPartial and Html.Partial are used to render partial views, but they differ in:
        - Html.Partial: Returns an IHtmlString, which allows capturing the output.

```
@Html.Partial("PartialViewName", model)
```

        - Html.RenderPartial: Writes the rendered output directly to the response stream, making it more efficient for larger content.

```
@Html.RenderPartial("PartialViewName", model)
```

11. What are strongly-typed views in MVC? How do you create one?
    - Strongly-typed views in MVC are views that are bound to a specific model type. This allows for compile-time checking and IntelliSense support in the view.
    - Creating a Strongly-Typed View:
        - In the View, specify the model type at the top of the file using the @model directive.

```
@model YourNamespace.Models.YourModel
```

        - The view can now access properties of the model directly.
12. What is the life cycle of an ASP.NET MVC application?
    - The life cycle of an ASP.NET MVC application consists of several key stages:
        - **Request Handling:** The application receives an HTTP request.
        - **Routing:** The routing engine maps the URL to the appropriate controller and action.
        - **Controller Execution:** The controller processes the request, interacts with the Model, and prepares data for the View.
        - **View Rendering:** The View is generated and rendered to HTML.
        - **Response:** The generated HTML is sent back to the client as an HTTP response.
13. Explain the difference between @Html.ActionLink and <a href> in Razor.
    - @Html.ActionLink is a helper method that generates an HTML anchor (<a>) tag with a link to a specific action method in a controller. It automatically creates the URL based on routing.

```
@Html.ActionLink("Link Text", "ActionName", "ControllerName")
```

    - <a href> is a standard HTML anchor tag where the URL must be specified manually. It does not take advantage of MVC routing.

```
html
<a href="/Controller/Action">Link Text</a>
```

14. What is the default route in MVC?
    - The default route in MVC is usually defined in the RouteConfig.cs file and typically follows the pattern:

```
routes.MapRoute(
name: "Default",
url: "{controller}/{action}/{id}",
defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

    - This route maps requests to the Home controller's Index action by default.
15. What is the purpose of ActionResult in ASP.NET MVC?
    - ActionResult is a base class that represents the result of an action method in a controller. It provides a way to return different types of responses, such as:
        - **ViewResult:** Renders a View.
        - **JsonResult:** Returns JSON data.
        - **RedirectResult:** Redirects to a different URL.
        - **ContentResult:** Returns plain text or HTML.

16. Explain the HttpGet and HttpPost attributes.
    - HttpGet and HttpPost are attributes used to specify the HTTP methods that a controller action can respond to.
        - **HttpGet:** Indicates that the action should be invoked for HTTP GET requests, typically used for retrieving data.
        - **HttpPost:** Indicates that the action should be invoked for HTTP POST requests, commonly used for submitting data (e.g., form submissions).
17. How do you handle exceptions in MVC applications?
    - Exceptions in MVC can be handled using:
        - **Try-Catch Blocks:** Surround code with try-catch to manage specific exceptions.
        - **Global Exception Handling:** Implement a custom HandleErrorAttribute or override the OnException method in the controller to handle exceptions globally.
        - **Custom Error Pages:** Configure custom error pages in the web.config file for specific HTTP status codes.
18. What is Scaffold in ASP.NET MVC?
    - **Scaffolding** is a code generation framework in ASP.NET MVC that automates the creation of controller and view code based on a specified model. It helps developers quickly generate CRUD operations for a model, saving time during development.
19. What are Areas in ASP.NET MVC?
    - **Areas** are a way to partition a large ASP.NET MVC application into smaller, manageable sections. Each Area can have its own controllers, views, and models, helping to organize related functionality and improve maintainability.
20. What is the purpose of Layout.cshtml in Razor?
    - Layout.cshtml serves as a master page for Razor views, providing a common structure for the layout of multiple views. It typically includes shared components like headers, footers, and navigation menus. Views can specify a layout using the Layout property at the top of the file.

```
@{
Layout = "~/Views/Shared/_Layout.cshtml";
}
```

Intermediate Level:

21. How do you implement custom routing in ASP.NET MVC?
    - To implement custom routing, define a new route in the RouteConfig.cs file before the default route:

```
routes.MapRoute(
name: "CustomRoute",
url: "products/{category}/{id}",
defaults: new { controller = "Products", action = "Index", id = UrlParameter.Optional }
);
```

22. What is Partial View in MVC and how is it different from View?
    - A **Partial View** is a reusable component that renders a portion of the view, often used for rendering common UI elements. It differs from a regular View in that it does not have a full layout or be rendered as a standalone page. It can be included in other views using Html.Partial or Html.RenderPartial.
23. How can you maintain session state in MVC applications?
    - Session state in MVC applications can be maintained using:
        - **Session Variable:** Store user-specific data in the Session object.
        - **Cookies:** Use cookies to store small amounts of data on the client-side.
        - **Database or Cache:** Store session data in a database or in-memory cache for larger amounts of data or distributed applications.
24. What are filters in MVC? Explain types like authorization, action, and exception filters.
    - **Filters** are used to execute code before or after an action method runs. Types include:
        - **Authorization Filters:** Check if a user is authorized to access an action.
        - **Action Filters:** Run code before or after an action executes, often used for logging or modifying the action parameters.
        - **Exception Filters:** Handle exceptions thrown during action execution and provide a way to log errors or redirect users to error pages.
25. Explain the difference between View() and RedirectToAction() in ASP.NET MVC.
    - **View():** Returns a ViewResult that renders a specific view directly without changing the URL.

```
return View("MyView");
```

    - **RedirectToAction():** Redirects the user to another action method, changing the URL in the browser.

```
return RedirectToAction("Index", "Home");
```

26. What is Bundling and Minification in MVC?
    - **Bundling and Minification** is a technique used to improve the performance of web applications by reducing the number of HTTP requests and minimizing file sizes.
        - **Bundling:** Groups multiple CSS or JavaScript files into a single file.
        - **Minification:** Removes unnecessary characters (like whitespace and comments) from files to reduce their size.
27. How can you implement AJAX in MVC applications?
    - AJAX can be implemented in MVC using:
        - **jQuery AJAX methods:** Use jQuery to make asynchronous calls to controller actions.
        - **AjaxHelper methods:** Use built-in helpers like @Ajax.ActionLink to create AJAX-enabled links or forms.
        - **Return JSON:** Create actions that return JSON data, which can be consumed by JavaScript on the client-side.
28. How can you validate user input in MVC applications?
    - Input validation can be achieved using:
        - **Data Annotations:** Apply validation attributes (e.g., [Required], [StringLength]) to model properties.
        - **Fluent Validation:** Use third-party libraries like FluentValidation for complex validation rules.
        - **ModelState Validation:** Check ModelState.IsValid in controller actions to ensure data meets validation requirements before processing.

29. What is Razor syntax and how does it work in ASP.NET MVC?
    - Razor syntax is a markup syntax used for embedding C# code in HTML. It begins with the @ symbol and allows seamless integration of C# logic and HTML markup. Razor is processed on the server to produce HTML output, making it easier to create dynamic web pages.
30. What is AntiForgeryToken and why is it used?
    - AntiForgeryToken is a security feature used to prevent Cross-Site Request Forgery (CSRF) attacks. It generates a unique token that is sent with forms and validated on the server to ensure the request comes from a trusted source. It is implemented using the @Html.AntiForgeryToken() helper method in forms.
31. How can you use @Html.EditorFor and @Html.DisplayFor helpers in Razor?
    - @Html.EditorFor(model => model.Property) generates an HTML input element for editing the specified model property, including appropriate validation attributes.
    - @Html.DisplayFor(model => model.Property) generates HTML to display the specified model property without editing capabilities, using the display templates if available.
32. Explain Model Binding in ASP.NET MVC.
    - Model Binding is the process of mapping form data from HTTP requests to action method parameters or model properties. It automatically converts data types and validates input. ASP.NET MVC uses built-in model binders, but custom model binders can also be created to handle specific scenarios.
33. What is the difference between @Html.DropDownList and @Html.DropDownListFor?
    - @Html.DropDownList generates a drop-down list without binding it to a model property.

    @Html.DropDownList("Category", new SelectList(categories, "Id", "Name"))
    - @Html.DropDownListFor generates a drop-down list and binds it to a specific model property, automatically managing the selected value.

    @Html.DropDownListFor(model => model.SelectedCategory, new SelectList(categories, "Id", "Name"))
34. What is Dependency Injection (DI)? How do you implement it in MVC?
    - Dependency Injection (DI) is a design pattern used to achieve Inversion of Control (IoC) by injecting dependencies into a class rather than having the class instantiate them itself. In MVC, DI can be implemented using:
        - Constructor Injection: Pass dependencies through the constructor of the controller.
        - Service Locator: Use a service locator pattern to resolve dependencies.
        - IoC Containers: Utilize libraries like Autofac, Ninject, or Unity to manage and resolve dependencies automatically.
35. Explain the difference between server-side and client-side validation in MVC.
    - Server-Side Validation: Validation occurs on the server after the form is submitted. It is more secure as it cannot be bypassed by users but may lead to a round-trip to the server if validation fails.
    - Client-Side Validation: Validation occurs in the browser before the form is submitted, providing immediate feedback to users. However, it can be bypassed, so server-side validation is also necessary for security.

36. What is the use of @RenderSection in Razor views?
    - @RenderSection allows you to define optional sections in a Razor layout that child views can fill. It helps create flexible layouts where child views can provide additional content.

@RenderSection("Scripts", required: false)

    - In a child view, you can define the section:

```
@section Scripts {
<script src="..."></script>
}
```

37. What are HTML Helpers in Razor? Provide examples.
    - HTML Helpers are methods that return HTML markup in Razor views, simplifying the process of creating common HTML elements. Examples include:
        - @Html.TextBoxFor(model => model.Property) to create a text input.
        - @Html.CheckBox("IsAccepted") to create a checkbox.
        - @Html.ValidationMessageFor(model => model.Property) to display validation messages.
38. What is the role of the RouteConfig file in ASP.NET MVC?
    - The RouteConfig file is used to define the application's routing rules. It specifies how URLs are mapped to controllers and actions, helping the framework determine which code to execute for incoming requests. The routing configuration is typically found in App_Start\RouteConfig.cs.
39. What is the use of AsyncController in ASP.NET MVC?
    - AsyncController is used to create asynchronous action methods in MVC, allowing the application to handle multiple requests efficiently without blocking threads. However, in modern ASP.NET MVC, it is recommended to use the async and await keywords with Task<ActionResult> as the return type instead of the older AsyncController pattern.
40. How do you handle multiple forms on a single page in MVC using Razor?
    - Multiple forms can be handled on a single page by giving each form a unique action method and ensuring that each form's inputs are properly named to avoid conflicts. Use different view models for each form if necessary, and ensure that the correct model binding occurs in the action methods.

Advanced Level:

41. How do you create a custom Action Filter in ASP.NET MVC?
    - To create a custom Action Filter, derive from ActionFilterAttribute and override the desired methods (e.g., OnActionExecuting, OnActionExecuted). Register the filter globally or on specific controllers/actions.

```
public class CustomActionFilter : ActionFilterAttribute {
public override void OnActionExecuting(ActionExecutingContext filterContext) {
// Custom logic before the action executes
}
}
```

42. What are Child Actions in MVC? How do you use them?
    - Child Actions are action methods that can be invoked from within views. They allow for rendering reusable components with their own models. Use Html.Action("ActionName", "ControllerName") to call a child action from a view.

```
public ActionResult RecentPosts() {
// Return a partial view with recent posts
return PartialView();
}
```

43. How can you implement RESTful services using ASP.NET MVC?
    - RESTful services can be implemented by creating actions that correspond to HTTP verbs (GET, POST, PUT, DELETE) in controllers. Use attribute routing to map URLs to these actions, and return appropriate HTTP status codes and data formats (JSON or XML) based on the request.

```
[HttpGet]
public JsonResult GetProduct(int id) {
// Return product data as JSON
return Json(product, JsonRequestBehavior.AllowGet);
}
```

44. What is @section in Razor? How do you use it in views?
    - @section is used to define a named section in a Razor view that can be rendered in a layout. This allows child views to inject content into specific parts of the layout.

```
@section Head {
<link rel="stylesheet" href="styles.css" />
}
```

45. How can you implement caching in an ASP.NET MVC application?
    - Caching can be implemented using:
        - Output Caching: Use the [OutputCache] attribute to cache the output of controller actions.
        - Data Caching: Use HttpRuntime.Cache or a distributed cache like Redis for caching data across requests.
        - Response Caching Middleware: In ASP.NET Core, use response caching middleware for advanced caching strategies.
46. What is the role of ViewStart.cshtml in an MVC application?
    - ViewStart.cshtml is a special file that runs before any view is rendered. It allows you to define common settings, such as the layout, that apply to multiple views in the application, reducing duplication.

```
@{
Layout = "~/Views/Shared/_Layout.cshtml";
}
```

47. How do you optimize the performance of an MVC application?
    - Performance optimization can include:
        - **Minification and Bundling:** Reduce file sizes and HTTP requests.
        - **Caching:** Use output caching, data caching, and distributed caching.
        - **Asynchronous Programming:** Use async and await for I/O-bound operations.
        - **Database Optimization:** Use efficient queries and indexes.
        - **Profiling:** Analyze application performance with tools like MiniProfiler or Application Insights.
48. What is OWIN and how does it work with ASP.NET MVC?
    - **OWIN (Open Web Interface for .NET)** is a standard that decouples the web server from the application framework. It allows for middleware components to be added in the request pipeline. ASP.NET MVC applications can use OWIN to add features like authentication, CORS, and other middleware components.
49. Explain how Web APIs are integrated with ASP.NET MVC.
    - Web APIs can be integrated into ASP.NET MVC applications by adding API controllers that return JSON or XML responses. These controllers can coexist with MVC controllers in the same project, allowing for a unified application structure. Routing for APIs is typically defined separately.
50. What are Tag Helpers in ASP.NET Core MVC and how do they work with Razor views?
    - **Tag Helpers** are a feature in ASP.NET Core that allows developers to create custom HTML elements with server-side functionality. They enable the use of C# code in Razor views to generate dynamic content. Tag Helpers can be used to replace traditional HTML helpers, making markup cleaner and easier to read.

    html

    ```
    <form asp-controller="Home" asp-action="Index" method="post">
    <input asp-for="Name" />
    <button type="submit">Submit</button>
    </form>
    ```

--==========

Entity Framework

1. What is Entity Framework (EF)?

Answer: Entity Framework is an open-source object-relational mapper (ORM) for .NET applications. It allows developers to work with databases using .NET objects, eliminating the need for most of the data-access code that developers typically need to write.

2. What are the different approaches to using Entity Framework?

Answer: There are three main approaches:

- **Database-First:** Generates EF model classes from an existing database.
- **Model-First:** Creates a model using the EF designer, which generates the database.
- **Code-First:** Developers write classes and EF generates the database schema from these classes.

3. What is Code-First approach in EF?

Answer: In the Code-First approach, you define your domain model using .NET classes. EF uses these classes to generate the database schema and manage the database through migrations.

4. Explain the Entity Framework DbContext.

Answer: DbContext is a class that represents a session with the database. It is used to query and save instances of entities. It manages the entity lifecycle and tracks changes to entities.

5. What are DbSet and its role in EF?

Answer: DbSet represents a collection of entities of a specific type. It allows you to perform CRUD (Create, Read, Update, Delete) operations on the entity type it represents.

6. How do you perform a query using Entity Framework?

Answer: You can perform queries using LINQ (Language Integrated Query). For example:

```
var products = context.Products.Where(p => p.Price > 100).ToList();
```

7. What is Entity Framework Migration?

Answer: Migrations are a way to update the database schema without losing data. They allow you to version control your database schema changes and apply them incrementally.

8. How do you create a migration in EF?

Answer: You can create a migration using the Package Manager Console with the command:

sql

Add-Migration MigrationName

Then update the database using:

mathematica

Update-Database

9. What is the difference between eager loading, lazy loading, and explicit loading?

Answer:

- Eager Loading: All related data is loaded at the same time using Include().
- Lazy Loading: Related data is loaded on demand, i.e., when you access the property.
- Explicit Loading: You load related data explicitly after the initial query using the Load() method.

10. What are Navigation Properties in EF?

Answer: Navigation properties are properties defined in an entity that allow navigation to related entities. They represent relationships between entities, such as one-to-many or many-to-many.

11. How do you handle concurrency in EF?

Answer: You can handle concurrency by using optimistic concurrency control. This can be done by adding a RowVersion column to your entity and checking for conflicts when saving changes.

12. What are the different types of relationships in EF?

Answer: The main types of relationships are:

- One-to-One: Each entity in the relationship corresponds to one entity.
- One-to-Many: One entity can have multiple related entities.
- Many-to-Many: Multiple entities can relate to multiple entities.

13. What is the purpose of the AsNoTracking method?

Answer: AsNoTracking is used to query entities without tracking them in the context. This can improve performance for read-only scenarios because it reduces memory usage.

14. How do you perform transactions in EF?

Answer: You can use the DbContext.Database.BeginTransaction() method to manage transactions. This allows you to group multiple operations and commit or roll back as needed.

15. What is the difference between FirstOrDefault and SingleOrDefault?

Answer:

- FirstOrDefault: Returns the first element of a sequence or a default value if no element is found. It can return multiple results.
- SingleOrDefault: Returns the only element of a sequence or a default value. It throws an exception if there is more than one element.

16. How do you optimize performance in Entity Framework?

Answer: Performance can be optimized by:

- Using AsNoTracking for read-only queries.
- Avoiding N+1 query problems by using eager loading.
- Using compiled queries for repetitive queries.
- Minimizing the amount of data retrieved.

17. What are complex types in EF?

Answer: Complex types are types that do not have a primary key and are used to define properties of an entity. They can encapsulate multiple properties into a single property.

18. Explain the use of the Include method in EF.

Answer: The Include method is used to specify related entities to include in the query results. It allows eager loading of navigation properties to reduce the number of database calls.

19. How do you handle entity validation in EF?

Answer: Entity validation can be handled using Data Annotations or Fluent API. Data Annotations are attributes applied to model properties, while Fluent API provides a more flexible way to configure validation in the OnModelCreating method.

20. What is the purpose of the OnModelCreating method?

Answer: The OnModelCreating method is overridden in the DbContext class to configure the model using Fluent API. It allows you to specify configurations such as relationships, constraints, and table mappings.

--==========

SSRS:

1. What is SSRS?

Answer: SSRS stands for SQL Server Reporting Services. It is a server-based report generating software system from Microsoft that allows users to create, manage, and deliver reports in various formats, such as PDF, Excel, Word, and HTML.

2. What are the key components of SSRS?

Answer: The key components of SSRS include:

- Report Builder: A tool for creating reports.
- Report Server: The server that hosts and processes the reports.
- Report Manager: A web-based interface for managing reports.
- Data Sources: Connections to the databases that provide data for the reports.

3. Explain the types of reports in SSRS.

Answer: SSRS supports three main types of reports:

- Tabular Reports: Display data in a straightforward table format.
- Matrix Reports: Similar to pivot tables, these allow for dynamic data organization.
- Graphical Reports: Use charts and graphs for visual representation of data.

4. What is a dataset in SSRS?

Answer: A dataset is a collection of data that a report uses. It can be created using a query, stored procedure, or through a connection to a data source. Each dataset can have multiple fields that correspond to the report's data.

5. How do you deploy reports in SSRS?

Answer: Reports can be deployed in SSRS by:

- Using the Report Manager to upload reports directly.
- Using SQL Server Data Tools (SSDT) to publish reports from Visual Studio.
- Using the SSRS Web Service API for programmatic deployments.

6. What is a data source in SSRS?

Answer: A data source in SSRS defines the connection information to retrieve data for reports. It can be a shared data source (available to multiple reports) or a embedded data source (specific to a single report).

7. How do you handle parameters in SSRS?

Answer: Parameters in SSRS can be created to filter data or provide dynamic values. They can be set up in the report designer, and users can input or select parameter values when running the report.

8. What are expressions in SSRS?

Answer: Expressions in SSRS are used to perform calculations, format data, or control report behavior dynamically. They can be created using the Expression Builder and can include various functions and operators.

9. How can you schedule report execution in SSRS?

Answer: Reports can be scheduled for execution using the SQL Server Agent. You can create a job to run a report at specific intervals, such as daily, weekly, or monthly.

10. What are subscriptions in SSRS?

Answer: Subscriptions allow users to receive reports automatically via email or to save them in a specific location at scheduled times. There are two types of subscriptions: standard and data-driven subscriptions.

11. What is a snapshot in SSRS?

Answer: A snapshot is a static view of a report at a certain point in time. It captures the report's data and layout, allowing users to view the report without executing the query again.

12. How do you implement security in SSRS?

Answer: Security in SSRS can be implemented using role-based security, where users are assigned roles that dictate what they can do (view, edit, manage) with reports and data sources. This can be configured in Report Manager or through the SSRS Web Service.

13. What are the different rendering formats available in SSRS?

Answer: SSRS supports several rendering formats, including:

- HTML
- PDF
- Excel
- Word
- CSV
- XML
- Image formats (JPEG, PNG, etc.)

14. What is the difference between a stored procedure and a view?

Answer: A **stored procedure** is a precompiled collection of SQL statements that can perform actions on the database, while a **view** is a virtual table based on the result set of a SQL query. Views can be used to simplify complex queries.

15. How can you optimize SSRS reports?

Answer: Optimization strategies for SSRS reports include:

- Using stored procedures for complex queries.
- Limiting the amount of data retrieved.
- Using appropriate indexes on the database tables.
- Avoiding unnecessary calculations in the report.

16. What are custom code and custom assemblies in SSRS?

Answer: Custom code allows you to add VB.NET code to a report to perform complex calculations or operations. Custom assemblies can be referenced in a report to utilize external .NET libraries for additional functionality.

17. How can you integrate SSRS with SharePoint?

Answer: SSRS can be integrated with SharePoint by using the SharePoint Integrated Mode, allowing users to manage and view reports directly within SharePoint sites. This integration provides a seamless experience for users.

18. What is the difference between Report Manager and Report Server?

Answer: Report Manager is the web-based application used for managing and accessing reports, while the **Report Server** is the backend server that processes reports, handles data connections, and manages report execution.

19. How do you troubleshoot SSRS report issues?

Answer: Troubleshooting SSRS issues can involve:

- Checking the report server logs for error messages.
- Verifying data source connections.
- Ensuring parameters are set correctly.
- Testing queries used in datasets.

20. Can you explain the difference between a standard subscription and a data-driven subscription?

Answer: A standard subscription allows users to receive the same report at scheduled intervals, while a data-driven subscription dynamically determines the recipients and parameters based on data queries at the time of report execution.

---==========

JAVASCRIPT:

1. What is JavaScript?

Answer: JavaScript is a lightweight, interpreted programming language that is primarily used for enhancing web pages to provide interactive features. It can be run in the browser or on a server using environments like Node.js.

2. What are the data types supported by JavaScript?

Answer: JavaScript supports several data types, including:

- Primitive Types: Number, String, Boolean, Null, Undefined, Symbol, BigInt
- Reference Types: Object, Array, Function, Date, RegExp

3. What is the difference between == and ===?

Answer: == is the equality operator that performs type coercion, meaning it converts the operands to the same type before making the comparison. === is the strict equality operator that checks both value and type, without performing type coercion.

4. What is a closure in JavaScript?

Answer: A closure is a function that retains access to its lexical scope even when the function is executed outside that scope. Closures allow for data encapsulation and private variables.

5. Explain the concept of hoisting in JavaScript.

Answer: Hoisting is a behavior in JavaScript where variable and function declarations are moved to the top of their containing scope during compilation. However, only declarations are hoisted, not initializations.

6. What are promises in JavaScript?

Answer: Promises are objects that represent the eventual completion (or failure) of an asynchronous operation and its resulting value. They allow for better handling of asynchronous code using methods like .then(), .catch(), and .finally().

7. What is the difference between let, const, and var?

Answer:

- var: Function-scoped or globally scoped, can be redeclared and updated.
- let: Block-scoped, can be updated but not redeclared in the same scope.
- const: Block-scoped, cannot be updated or redeclared, but the object it refers to can be mutated.

8. What is event delegation in JavaScript?

Answer: Event delegation is a technique that involves attaching a single event listener to a parent element rather than to individual child elements. This improves performance and allows for dynamically added elements to respond to events.

9. Explain the this keyword in JavaScript.

Answer: The this keyword refers to the object that is executing the current function. Its value can vary depending on the context in which a function is called (e.g., as a method of an object, in a constructor, etc.).

10. What are template literals in JavaScript?

Answer: Template literals are string literals that allow embedded expressions, multi-line strings, and string interpolation. They are enclosed by backticks (`) instead of single or double quotes.

11. What is the purpose of the bind(), call(), and apply() methods?

Answer:

- bind(): Creates a new function that, when called, has its this keyword set to a specified value.
- call(): Calls a function with a given this value and arguments provided individually.
- apply(): Calls a function with a given this value and arguments provided as an array.

12. What is the difference between synchronous and asynchronous programming?

Answer: Synchronous programming executes tasks sequentially, meaning each task must complete before the next one starts. Asynchronous programming allows tasks to be executed concurrently, enabling other code to run while waiting for a task to complete.

13. What are arrow functions in JavaScript?

Answer: Arrow functions are a shorter syntax for writing function expressions. They do not bind their own this, which means they inherit this from the surrounding lexical context.

14. What is the event loop in JavaScript?

Answer: The event loop is a mechanism that allows JavaScript to perform non-blocking I/O operations. It continuously checks the call stack and the message queue, executing tasks from the queue when the stack is empty.

15. What is JSON and how is it used in JavaScript?

Answer: JSON (JavaScript Object Notation) is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. It is commonly used to transmit data between a server and a client.

16. What is the difference between Object.keys() and Object.entries()?

Answer:

- **Object.keys():** Returns an array of a given object's own property names.
- **Object.entries():** Returns an array of a given object's own enumerable string-keyed property [key, value] pairs.

17. What are the advantages of using async/await?

Answer: async/await simplifies working with promises by allowing asynchronous code to be written in a more synchronous fashion, making it easier to read and maintain. It also allows for better error handling using try/catch blocks.

18. What is the purpose of try/catch in JavaScript?

Answer: The try/catch statement is used to handle exceptions. Code that may throw an error is placed in the try block, while the catch block contains the code to handle the error if one occurs.

19. Explain the concept of prototypal inheritance.

Answer: Prototypal inheritance is a feature in JavaScript where objects can inherit properties and methods from other objects. This is achieved through the prototype chain, allowing for shared behavior and properties.

20. What is a module in JavaScript?

Answer: A module is a reusable piece of code that encapsulates functionality. JavaScript modules can export variables and functions so they can be used in other files, promoting modular programming and separation of concerns. ES6 introduced native module support with import and export statements.

JQuery:

1. What is jQuery?

Answer: jQuery is a fast, lightweight, and feature-rich JavaScript library. It simplifies HTML document traversal and manipulation, event handling, animation, and Ajax interactions for rapid web development.

2. How do you include jQuery in a project?

Answer: jQuery can be included in a project by downloading the jQuery library and linking it in your HTML file or by using a Content Delivery Network (CDN). For example:

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

3. What is the purpose of the $ in jQuery?

Answer: The $ is an alias for the jQuery function. It is used to access jQuery features and methods. For example, $(document).ready(function() {...}) ensures the DOM is fully loaded before executing the function.

4. Explain the difference between $(document).ready() and $(window).load().

Answer: $(document).ready() executes when the DOM is fully loaded, while $(window).load() waits for all content (including images, iframes, etc.) to load completely.

5. What are jQuery selectors?

Answer: jQuery selectors are patterns used to select and manipulate HTML elements. Common selectors include ID selectors (#id), class selectors (.class), and element selectors (element).

6. How do you handle events in jQuery?

Answer: You can handle events in jQuery using methods like .click(), .on(), or .bind(). For example:

$('#myButton').click(function() {

alert('Button clicked!');

});

7. What is the purpose of the .each() method?

Answer: The .each() method iterates over a jQuery object and executes a function for each matched element. It allows you to perform operations on each element in the set.

8. How can you make an Ajax request using jQuery?

Answer: You can make an Ajax request using the .ajax() method or shorthand methods like .get(), .post(). For example:

```
$.ajax({

url: 'https://api.example.com/data',

method: 'GET',

success: function(response) {

console.log(response);

}

});
```

9. What is the difference between $.get() and $.ajax()?

Answer: $.get() is a shorthand method for making a GET request, while $.ajax() is a more versatile method that allows you to specify the request type, data, headers, and more.

10. What is the purpose of the .css() method in jQuery?

Answer: The .css() method in jQuery is used to get or set the CSS properties of the selected elements. For example:

```
$('#myElement').css('color', 'red'); // Sets the color to red
```

11. How can you hide or show elements in jQuery?

Answer: You can hide elements using .hide() and show them using .show(). You can also toggle visibility with .toggle(). For example:

```
$('#myElement').hide(); // Hides the element

$('#myElement').show(); // Shows the element

$('#myElement').toggle(); // Toggles visibility
```

12. Explain how to manipulate the DOM using jQuery.

Answer: jQuery allows you to manipulate the DOM through methods like .append(), .prepend(), .remove(), and .empty(). For example:

```
$('#myList').append('<li>New Item</li>'); // Adds a new list item
```

13. What is chaining in jQuery?

Answer: Chaining in jQuery allows you to perform multiple operations on the same jQuery object in a single statement by returning the jQuery object itself. For example:

```
$('#myElement').css('color', 'red').fadeIn(2000).slideUp();
```

14. How do you prevent the default action of an event in jQuery?

Answer: You can prevent the default action of an event using the event.preventDefault() method within the event handler. For example:

```
$('form').submit(function(event) {

event.preventDefault(); // Prevents form submission

});
```

15. What are jQuery plugins, and how do you use them?

Answer: jQuery plugins are reusable pieces of code that extend jQuery's functionality. To use a plugin, include its script file after the jQuery library and call it on a jQuery object. For example:

```
$(document).ready(function() {

$('#myElement').myPlugin();

});
```

16. How can you animate elements in jQuery?

Answer: You can animate elements using methods like .fadeIn(), .fadeOut(), .slideUp(), and .slideDown(), as well as custom animations using the .animate() method. For example:

```
$('#myElement').fadeOut(1000); // Fades out over 1 second
```

17. What is the $.noConflict() method?

Answer: The $.noConflict() method is used to release the $ variable to other libraries (like Prototype) that may also use it. After calling this method, you can still use jQuery by referencing jQuery instead of $.

18. Explain the data- attribute and how to use it in jQuery.

Answer: The data- attribute is used to store custom data on HTML elements. You can access these data attributes using the .data() method. For example:

```
<div id="myElement" data-info="123"></div>



var info = $('#myElement').data('info'); // Retrieves the value 123
```

19. How can you serialize form data in jQuery?

Answer: You can serialize form data using the .serialize() method, which converts form inputs into a query string format. For example:

javascript

var formData = $('form').serialize(); // Serializes form data

20. What is the difference between stop() and stop(true, true) in jQuery?

Answer: The stop() method stops the currently running animations on an element. stop(true, true) not only stops the current animation but also clears the animation queue, ensuring that all queued animations are removed.

LINQ:

1. What is LINQ?

Answer: LINQ (Language Integrated Query) is a Microsoft .NET Framework component that adds native data querying capabilities to .NET languages. It allows developers to write queries directly in C# or VB.NET using a syntax similar to SQL.

2. What are the types of LINQ?

Answer: There are several types of LINQ:

- LINQ to Objects: Queries against in-memory collections (like arrays and lists).
- LINQ to SQL: Queries against SQL Server databases.
- LINQ to XML: Queries against XML data.
- LINQ to Entities: Queries against Entity Framework models.
- LINQ to DataSet: Queries against ADO.NET DataSets.

3. What is the difference between deferred execution and immediate execution in LINQ?

Answer:

- Deferred Execution: Queries are not executed until the results are enumerated. This allows for query composition.
- Immediate Execution: Queries are executed as soon as they are called, returning results immediately (e.g., using methods like .ToList(), .ToArray()).

4. Can you explain LINQ syntax?

Answer: LINQ can be written using two main syntaxes:

- Query Syntax: Similar to SQL. Example:

```
var results = from item in collection
where item.Condition
select item;
```

- Method Syntax: Uses extension methods. Example:

```
var results = collection.Where(item => item.Condition);
```

5. What is a LINQ provider?

Answer: A LINQ provider is a component that translates LINQ queries into a format that can be executed against a specific data source. Examples include LINQ to SQL and Entity Framework.

6. What is the role of the IQueryable interface?

Answer: IQueryable is an interface that allows LINQ to be used with external data sources. It enables querying capabilities against data stores while deferring execution until the query is actually executed.

7. What is the IEnumerable interface?

Answer: IEnumerable is an interface that defines a method for iterating over a collection. LINQ to Objects primarily uses this interface, allowing for query capabilities over in-memory collections.

8. Explain the Select method in LINQ.

Answer: The Select method is used to project each element of a sequence into a new form. It transforms the data by specifying a selector function. Example:

```
var names = people.Select(person => person.Name);
```

9. What is the difference between Select and SelectMany?

Answer:

- Select: Projects each element into a new form, typically one-to-one.
- SelectMany: Flattens the results from a collection of collections, allowing for one-to-many relationships. Example:

```
var flatList = people.SelectMany(person => person.Children);
```

10. What is the purpose of the GroupBy method in LINQ?

Answer: The GroupBy method is used to group elements that share a common attribute into collections. This is helpful for categorizing data. Example:

```
var grouped = people.GroupBy(person => person.Age);
```

11. What is the OrderBy method in LINQ?

Answer: The OrderBy method is used to sort elements of a sequence in ascending order based on a specified key. The OrderByDescending method sorts in descending order. Example:

```
var sorted = people.OrderBy(person => person.Name);
```

12. Explain the Join method in LINQ.

Answer: The Join method is used to combine elements from two sequences based on matching keys. Example:

```
var result = employees.Join(departments,

emp => emp.DepartmentId,

dept => dept.Id,

(emp, dept) => new { emp.Name, dept.Name });
```

13. What is the FirstOrDefault method in LINQ?

Answer: The FirstOrDefault method returns the first element of a sequence that satisfies a specified condition or a default value if no such element is found. Example:

```
var firstPerson = people.FirstOrDefault(p => p.Age > 30);
```

14. How do you perform pagination in LINQ?

Answer: Pagination can be performed using the Skip and Take methods. Example:

```
var pagedResults = people.Skip(10).Take(5);
```

15. What is the Any method in LINQ?

Answer: The Any method determines whether any elements in a sequence satisfy a specified condition or if the sequence is empty. Example:

```
bool hasAdults = people.Any(p => p.Age >= 18);
```

16. What is the All method in LINQ?

Answer: The All method checks if all elements in a sequence satisfy a specified condition. Example:

bool allAdults = people.All(p => p.Age >= 18);

17. How do you combine multiple LINQ queries?

Answer: Multiple LINQ queries can be combined using method chaining or through query syntax. Example:

var combinedResults = people.Where(p => p.Age > 18)

.OrderBy(p => p.Name)

.Select(p => new { p.Name, p.Age });

18. What is the difference between ToList and ToArray in LINQ?

Answer:

- ToList: Converts a sequence to a List<T>, which is a dynamic array that can change size.
- ToArray: Converts a sequence to an array of fixed size.

19. Explain the Distinct method in LINQ.

Answer: The Distinct method returns a new sequence that contains only unique elements from the original sequence, eliminating duplicates. Example:

var uniqueAges = people.Select(p => p.Age).Distinct();

20. How can you handle exceptions in LINQ queries?

Answer: Exceptions in LINQ queries can be handled using try-catch blocks. Additionally, using methods like DefaultIfEmpty() can help prevent exceptions when working with empty sequences. Example:

try {

var firstPerson = people.FirstOrDefault();

} catch (Exception ex) {

// Handle exception

}

--==========

WEB API:

1. What is ASP.NET Web API?
    - ASP.NET Web API is a framework for building HTTP services that can be accessed from various clients, including browsers and mobile devices. It supports RESTful principles and allows for the creation of RESTful services.
2. What are the key features of ASP.NET Web API?
    - **RESTful Services:** It is designed to support REST architecture.
    - **Content Negotiation:** It can return data in various formats like JSON, XML, etc.
    - **Routing:** It uses attribute routing and convention-based routing.
    - **Dependency Injection:** It supports dependency injection out of the box.
    - **Testability:** Promotes easy unit testing of APIs.
3. What is the difference between ASP.NET MVC and ASP.NET Web API?
    - ASP.NET MVC is used for creating web applications that return HTML, while ASP.NET Web API is used for building services that return data, typically in JSON or XML format. Web API is optimized for HTTP and is better suited for RESTful services.
4. Explain the concept of routing in ASP.NET Web API.
    - Routing in Web API is the mechanism that maps HTTP requests to action methods. It can be configured using attribute routing or convention-based routing defined in the WebApiConfig class.
5. What is content negotiation in Web API?
    - Content negotiation is the process through which a client can request a response in a specific format (like JSON or XML) and the server can return the appropriate format based on the client's request headers.

HTTP Methods and Status Codes

6. What are the common HTTP methods used in Web API?
    - **GET:** Retrieve data.
    - **POST:** Create new data.
    - **PUT:** Update existing data.
    - **DELETE:** Remove data.

7. What is the purpose of the HttpResponseMessage class?
    - The HttpResponseMessage class represents an HTTP response message, which includes status codes, headers, and the content of the response.
8. What are some common HTTP status codes and their meanings?
    - 200 OK: The request was successful.
    - 201 Created: A resource was created successfully.
    - 204 No Content: The request was successful but there's no content to return.
    - 400 Bad Request: The request is invalid.
    - 401 Unauthorized: Authentication is required.
    - 404 Not Found: The requested resource was not found.
    - 500 Internal Server Error: An error occurred on the server.

Model Binding and Validation

9. What is model binding in ASP.NET Web API?
    - Model binding is the process of mapping data from the request (like query parameters or JSON payloads) to action method parameters or model classes.
10. How do you implement validation in Web API?
    - You can implement validation using data annotations in model classes (e.g., [Required], [StringLength]) and custom validation attributes. You can also use IValidatableObject for more complex validation scenarios.

Dependency Injection and Services

11. What is Dependency Injection (DI)?
    - Dependency Injection is a design pattern that allows a class to receive its dependencies from an external source rather than creating them itself. This promotes loose coupling and easier testing.
12. How do you implement DI in Web API?
    - You can implement DI in Web API by using a dependency injection container, such as Unity, Autofac, or built-in DI in .NET Core. Register services in the Startup.cs file.

Authentication and Authorization

13. **What are the different types of authentication supported in Web API?**
    - **Basic Authentication:** Transmits credentials in base64 encoding.
    - **Token-Based Authentication:** Issues a token after successful login, which must be sent with each request.
    - **OAuth:** A standard for access delegation, commonly used for API access.
    - **JWT (JSON Web Tokens):** A compact token format for secure information exchange.
14. **What is CORS and how do you enable it in Web API?**
    - Cross-Origin Resource Sharing (CORS) is a security feature that allows or restricts resources requested from another domain. You can enable CORS in Web API by installing the CORS NuGet package and configuring it in WebApiConfig.cs.

Exception Handling

15. **How do you handle exceptions in ASP.NET Web API?**
    - You can handle exceptions globally by implementing a custom ExceptionFilterAttribute or using try-catch blocks in your controllers. Additionally, you can configure global error handling in the Application_Error method in Global.asax.
16. **What is the difference between IHttpActionResult and HttpResponseMessage?**
    - IHttpActionResult is an interface that provides a standardized way to create HTTP responses in controllers. HttpResponseMessage is a class representing an HTTP response message. Using IHttpActionResult promotes better abstraction and testability.

Versioning

17. **How do you implement versioning in Web API?**
    - Versioning can be implemented through URL segments (e.g., /api/v1/resource), query strings (e.g., /api/resource?version=1), or HTTP headers. The choice of method depends on your API design preferences.

Performance and Security

18. **What are some best practices for improving the performance of a Web API?**
    - **Use caching:** Implement caching mechanisms to reduce load.
    - **Use asynchronous programming:** Improve responsiveness by using async/await.
    - **Minimize data:** Return only the required data fields.

- **Optimize database queries:** Use efficient queries and indexing.

19. **What are some security best practices for Web API?**
    - **Use HTTPS:** Encrypt data in transit.
    - **Validate user input:** Protect against injection attacks.
    - **Implement authentication and authorization:** Ensure users have the right access.
    - **Limit request size:** Protect against DoS attacks.

Tools and Technologies

20. **What tools are commonly used for testing Web APIs?**
    - Tools like Postman, Swagger, and Fiddler are commonly used for testing and documenting Web APIs.
21. **What is Swagger and how does it relate to Web API?**
    - Swagger is a tool for documenting APIs. It provides a user-friendly interface to explore and test API endpoints, making it easier for developers to understand and use your API.

Serialization and Deserialization

22. **What is JSON serialization and deserialization?**
    - JSON serialization is the process of converting an object into a JSON string. Deserialization is the reverse process of converting a JSON string back into an object. In Web API, Newtonsoft.Json or System.Text.Json is commonly used for these operations.
23. **How do you customize JSON serialization settings in Web API?**
    - You can customize JSON serialization settings by modifying the JsonFormatter in the WebApiConfig.cs file, such as setting date formats, ignoring null values, or formatting indents.

Middleware and Hosting

24. **What is middleware in ASP.NET Core?**
    - Middleware is a component that is executed on every HTTP request in an ASP.NET Core application. It can handle requests and responses and can be used for logging, authentication, error handling, etc.

25. How do you host a Web API?

    - Web APIs can be hosted on various platforms, such as IIS, Kestrel (built-in web server in ASP.NET Core), Azure, or Docker containers.

Advanced Concepts

26. What is OData and how is it used in Web API?

    - OData (Open Data Protocol) is a standard protocol for building and consuming RESTful APIs. It allows clients to query and manipulate data using a set of conventions. You can enable OData in Web API through the OData NuGet package.

27. What is the difference between REST and SOAP?

    - REST (Representational State Transfer) is an architectural style using HTTP and is stateless. SOAP (Simple Object Access Protocol) is a protocol that relies on XML and has more rigid standards. REST is generally lighter and easier to work with compared to SOAP.

28. How do you implement pagination in a Web API?

    - Pagination can be implemented by accepting query parameters for page number and page size in your API endpoint and returning a subset of data accordingly.

29. What are background tasks in ASP.NET Web API?

    - Background tasks are operations that run independently of user requests, allowing you to perform tasks like sending emails or processing files without blocking the main application thread. In .NET Core, you can use IHostedService for implementing background tasks.

Testing and Debugging

30. How do you unit test a Web API?

    - You can unit test a Web API by creating mock objects for dependencies and using testing frameworks like xUnit, NUnit, or MSTest. You can also use libraries like Moq for mocking.

31. What is the purpose of integration testing in Web API?

    - Integration testing validates the interaction between different components of the application, ensuring that the API works as expected when integrated with other parts of the system, such as databases or external services.

Configuration and Deployment

32. How do you manage configuration settings in a Web API?
    - Configuration settings can be managed using appsettings.json, environment variables, or Azure App Configuration in .NET Core applications. You can access these settings via IConfiguration interface.
33. What are environment variables and how are they used in .NET?
    - Environment variables are key-value pairs stored in the system environment. They are used to configure applications without hardcoding settings in the code, allowing for different settings in development, staging, and production environments.

Logging

34. How do you implement logging in ASP.NET Web API?
    - Logging can be implemented using built-in logging providers in .NET Core or third-party libraries like Serilog or NLog. You can log information, warnings, errors, and critical messages to various outputs (files, databases, etc.).

Attributes and Filters

35. What are action filters in ASP.NET Web API?
    - Action filters are attributes that allow you to run code before or after an action method executes. They can be used for logging, authorization, or modifying the response.
36. What is the difference between AuthorizeAttribute and AllowAnonymousAttribute?
    - AuthorizeAttribute restricts access to authorized users only, while AllowAnonymousAttribute allows access to all users, regardless of authentication status. You can use these attributes to control access at the action or controller level.

Additional Questions

37. What is the role of the Global.asax file?
    - The Global.asax file is used for handling application-level events, such as application start, application end, session start, and session end. In Web API, it's less commonly used than in traditional ASP.NET MVC applications.
38. Explain the concept of a Resource in RESTful API design.
    - In RESTful API design, a resource is an entity or a piece of data that can be accessed or manipulated. Resources are identified using URIs, and operations on resources are performed using standard HTTP methods.

39. What is the purpose of a DTO (Data Transfer Object)?

    - A DTO is an object used to encapsulate data for transfer between layers or systems. It helps in reducing the amount of data sent over the network and can provide a simplified view of the data.

40. What is the role of the Startup class in ASP.NET Core?

    - The Startup class is the entry point for configuring the application. It contains methods for configuring services (ConfigureServices) and the HTTP request pipeline (Configure).

41. How do you enable Swagger in a .NET Core Web API?

    - You can enable Swagger in a .NET Core Web API by installing the Swashbuckle.AspNetCore NuGet package and configuring it in the Startup class.

42. What is a health check in ASP.NET Core?

    - Health checks are a way to monitor the status of an application. ASP.NET Core provides a health check middleware that allows you to define endpoints to check the health of the application and its dependencies.

43. What is the use of IHttpClientFactory?

    - IHttpClientFactory is a factory for creating HttpClient instances. It helps in managing the lifetime of HttpClient, improving performance, and avoiding socket exhaustion issues.

Final Questions

44. What is Middleware in ASP.NET Core?

    - Middleware is software that is assembled into an application pipeline to handle requests and responses. Each middleware component can perform operations before and after the next component in the pipeline is invoked.

45. What are the main differences between ASP.NET Core and ASP.NET Framework?

    - ASP.NET Core is a cross-platform, modular framework designed to be cloud-ready, whereas ASP.NET Framework is Windows-only. ASP.NET Core is designed for better performance and flexibility.

46. Explain the concept of a Token-based Authentication.

    - Token-based authentication is a stateless authentication mechanism where a user receives a token after successful login. This token is then sent with subsequent requests to validate the user's identity.

47. How do you secure a Web API?

    - You can secure a Web API using HTTPS, authentication (JWT, OAuth), authorization mechanisms, input validation, and setting CORS policies.

48. What are attribute routing and convention-based routing?

    - Attribute routing allows you to define routes directly on the action methods or controllers using attributes. Convention-based routing uses predefined patterns to define routes in a centralized location (typically in WebApiConfig).

49. What is the difference between a service and a repository pattern?

    - The repository pattern is used to abstract data access logic, allowing for a separation of concerns between data access and business logic. The service pattern contains business logic and communicates with repositories to retrieve or save data.

50. What is ASP.NET Core Identity?

    - ASP.NET Core Identity is a membership system that adds login functionality to your application, allowing you to manage users, passwords, roles, and claims. It can be configured with various authentication schemes.

---==========

SQL:

1. What is SQL? Why is it used?
   SQL (Structured Query Language) is used to manage and manipulate relational databases. It is used to query, update, insert, and delete data, as well as manage schema and control access to the database.

2. Differences between Stored Procedure and User Defined Function in SQL Server

Funtion	Stored proced
Function must return a value.	Stored Procedure may or not return values.
Will allow only Select statements, it will not allow us to use DML statements.	Can have select statements as well as DML statements such as insert, update, delete and so on
It will allow only input parameters, doesn't support output parameters.	It can have both input and output parameters.
It will not allow us to use try-catch blocks.	For exception handling we can use try catch blocks.
Transactions are not allowed within functions.	Can use transactions within Stored Procedures.
We can use only table variables, it will not allow using temporary tables.	Can use both table variables as well as temporary table in it.
Stored Procedures can't be called from a function.	Stored Procedures can call functions.
Functions can be called from a select statement.	Procedures can't be called from Select/Where/Having and so on statements. Execute/Exec statement can be used to call/execute Stored Procedure.
A UDF can be used in join clause as a result set.	Procedures can't be used in Join clause

3. What are the different types of SQL commands?
   SQL commands are categorized as:
   - DDL (Data Definition Language): CREATE, ALTER, DROP
   - DML (Data Manipulation Language): SELECT, INSERT, UPDATE, DELETE
   - DCL (Data Control Language): GRANT, REVOKE
   - TCL (Transaction Control Language): COMMIT, ROLLBACK, SAVEPOINT
   - DQL (Data Query Language): SELECT

4. Explain the difference between DELETE, TRUNCATE, and DROP.
   - DELETE: Removes rows from a table based on a condition. Can be rolled back.
   - TRUNCATE: Removes all rows from a table but keeps the structure. Cannot be rolled back.
   - DROP: Deletes the entire table, including its structure.

5. What is a primary key?
   A primary key uniquely identifies each record in a table. It cannot have NULL values and must be unique.

6. What is a foreign key?
   A foreign key is a field (or a collection of fields) in one table that refers to the primary key in another table. It creates a relationship between two tables.
7. What is normalization? Explain its types (1NF, 2NF, 3NF, BCNF).
   Normalization is the process of organizing data to reduce redundancy:
   - 1NF: Eliminate duplicate columns, ensure atomicity.
   - 2NF: Meet 1NF and remove partial dependencies.
   - 3NF: Meet 2NF and remove transitive dependencies.
   - BCNF: A stricter form of 3NF where every determinant is a candidate key.

8. What are joins in SQL? Describe different types of joins.
   Joins are used to retrieve data from multiple tables:
   - INNER JOIN: Returns records with matching values in both tables.
   - LEFT JOIN: Returns all records from the left table, and matched records from the right.
   - RIGHT JOIN: Returns all records from the right table, and matched records from the left.
   - FULL OUTER JOIN: Returns all records when there is a match in either table.

9. What is the difference between INNER JOIN and OUTER JOIN?
   - INNER JOIN: Retrieves only records that have matching values in both tables.
   - OUTER JOIN: Retrieves all records, whether or not there's a match. Variants include LEFT, RIGHT, and FULL OUTER JOIN.
10. What is a self-join?
    A self-join is a join where a table is joined with itself. It can be used to compare rows within the same table.

11. What is a view? How can you update a view?
    A view is a virtual table based on the result of a SELECT query. You can update a view if it contains data from a single table, has no aggregated columns, and does not involve complex joins or subqueries.

12. What are indexes in SQL? Why are they used?
    Indexes are database objects that improve the speed of data retrieval operations. They act as pointers to the data in a table, making queries faster.

13. What is the difference between clustered and non-clustered indexes?
    - Clustered index: Determines the physical order of data in the table. A table can have only one clustered index.
    - Non-clustered index: Does not affect the physical order of data. A table can have many non-clustered indexes.

13. How can you retrieve the second-highest salary in a table?
    SELECT MAX(Salary) FROM Employees WHERE Salary < (SELECT MAX(Salary) FROM Employees);

14. How do you use the GROUP BY and HAVING clauses?
    GROUP BY groups rows that share a property, and HAVING filters groups based on a condition. Example:

```
SELECT Department, COUNT(*)
FROM Employees
GROUP BY Department
HAVING COUNT(*) > 5;
```

15. What is a subquery? Explain with an example.
    A subquery is a query within another query. Example:

```
SELECT * FROM Employees WHERE DepartmentID = (SELECT DepartmentID FROM
Departments WHERE Name = 'HR');
```

16. What is a correlated subquery?
    A correlated subquery is one where the subquery references columns from the outer query. It is evaluated once for each row processed by the outer query.

17. Explain UNION and UNION ALL.
    - UNION: Combines the result sets of two queries and removes duplicates.
    - UNION ALL: Combines result sets, but does not remove duplicates.

18. How would you optimize a slow query?
    - Use indexes on frequently queried columns.
    - Avoid SELECT *, specify only the required columns.
    - Use EXISTS instead of IN for subqueries.
    - Minimize the use of JOIN where possible.
    - Analyze and rewrite slow subqueries.

19. What is the difference between WHERE and HAVING?
    - WHERE: Filters rows before grouping.
    - HAVING: Filters groups after aggregation.

20. What is the purpose of the EXISTS keyword?
    EXISTS is used to check whether a subquery returns any rows. It returns TRUE if the subquery finds at least one matching row.

21. How can you find duplicates in a table?

```
SELECT ColumnName, COUNT(*)
FROM TableName
GROUP BY ColumnName
HAVING COUNT(*) > 1;
```

22. What is a CASE statement in SQL? Give an example.
    CASE is used for conditional logic. Example:

```
SELECT
EmployeeID,
Salary,
CASE
WHEN Salary > 50000 THEN 'High'
WHEN Salary > 30000 THEN 'Medium'
ELSE 'Low'
END AS SalaryGrade
FROM Employees;
```

23. Explain the RANK(), DENSE_RANK(), and ROW_NUMBER() window functions.
    - RANK(): Assigns a unique rank to each row, skipping ranks if there are ties.
    - DENSE_RANK(): Similar to RANK(), but without gaps between ranks.
    - ROW_NUMBER(): Assigns a unique row number to each record.

24. What is the difference between a schema and a database?
    - A database is a collection of related tables and data.
    - A schema is the structure that defines tables, columns, and relationships within the database.

25. How would you design a one-to-many or many-to-many relationship?
    - One-to-many: Use a foreign key in the child table to reference the parent table.
    - Many-to-many: Use a junction table with foreign keys referencing both related tables.

26. What are constraints in SQL? Describe different types.
    Constraints are rules applied to table columns:
    - NOT NULL: Ensures that a column cannot have a NULL value.
    - UNIQUE: Ensures all values in a column are unique.
    - CHECK: Ensures that all values meet a condition.
    - DEFAULT: Assigns a default value if no value is specified.
    - PRIMARY KEY: Uniquely identifies each row.
    - FOREIGN KEY: Enforces referential integrity.

27. Explain the AUTO_INCREMENT property in SQL.
    The AUTO_INCREMENT attribute allows a column to automatically generate unique values when a new row is inserted.

28. What are transactions? Describe ACID properties.
    A transaction is a sequence of operations performed as a single logical unit. ACID properties:
    - Atomicity: All or nothing.
    - Consistency: Ensures the database is in a valid state after the transaction.
    - Isolation: Transactions do not interfere with each other.
    - Durability: Once a transaction is committed, changes are permanent.

29. What is the difference between COMMIT and ROLLBACK?
    - COMMIT: Makes changes made by a transaction permanent.
    - ROLLBACK: Reverts changes made by a transaction.

30. What is a trigger in SQL?
    A trigger is a set of SQL statements that automatically execute when a specified event occurs on a table (e.g., INSERT, UPDATE, DELETE).

31. What are stored procedures? How do they differ from functions?
    - A **stored procedure** is a set of SQL statements that can be executed as a unit.
    - A **function** returns a single value and can be used in a SQL statement. Stored procedures do not need to return a value.

32. What is a cursor? When should you use it?
    A cursor is a database object used to retrieve, manipulate, and iterate over result sets one row at a time. Cursors should be used when complex row-by-row operations are required but should be avoided when set-based operations are possible.

33. How does indexing impact query performance?
    Indexes improve the performance of SELECT queries by allowing the database engine to quickly locate rows. However, indexes can slow down INSERT, UPDATE, and DELETE operations because the index also needs to be updated.

34. What is the difference between IN and EXISTS?
    - IN: Used to filter rows based on a set of values.
    - EXISTS: Checks for the existence of rows returned by a subquery. EXISTS is generally faster for large result sets.

35. How do you analyze and troubleshoot query performance issues?
    - Use the EXPLAIN plan to understand the query execution.
    - Check for missing indexes or inefficient JOIN operations.
    - Avoid complex subqueries and use proper indexing.

36. What is an execution plan? How can you view it?
    An execution plan shows how the database engine executes a query, including table scans, index usage, and join types. You can view it using the EXPLAIN command.

37. How would you handle database locking and deadlock situations?
    - Use transactions appropriately to avoid holding locks for long.
    - Break large transactions into smaller ones.
    - Avoid SELECT statements that lock tables unnecessarily by using WITH(NOLOCK) or READ UNCOMMITTED isolation level.

38. What are the advantages of using JOIN over subqueries?
    - Joins are usually more efficient than subqueries because they directly relate tables rather than using intermediate results.
    - They are more readable and easier to maintain.

39. Explain query optimization techniques in SQL.
    - Use indexes wisely on columns that are frequently searched or joined.
    - Avoid SELECT * and retrieve only necessary columns.
    - Rewrite queries to minimize subqueries and use JOIN instead.
    - Analyze execution plans to identify bottlenecks.
40. What is partitioning in SQL? Why is it useful?
    Partitioning splits a table into smaller pieces to improve query performance and manageability. It is especially useful for large datasets as it allows queries to scan only relevant partitions.
41. What is the difference between MERGE and UPSERT?
    - MERGE: Combines INSERT, UPDATE, and DELETE into a single statement.
    - UPSERT: Inserts a new row or updates an existing row if a conflict occurs.
41. What is referential integrity?
    Referential integrity ensures that relationships between tables remain consistent (e.g., if a foreign key exists, it must correspond to a valid primary key in the related table).
42. How can you enforce data integrity in a database?
    Data integrity is enforced through constraints such as PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE, CHECK, and DEFAULT.
43. How can you retrieve data from multiple tables using joins?
    Joins combine rows from two or more tables based on a related column. Example:
    ```
    SELECT Employees.Name, Departments.Name
    FROM Employees
    INNER JOIN Departments
    ON Employees.DepartmentID = Departments.DepartmentID;
    ```
44. How do you handle NULL values in SQL?
    You can use IS NULL, IS NOT NULL, or functions like COALESCE() to handle NULL values.
45. Explain how to use COALESCE() and NULLIF() functions.
    - COALESCE(): Returns the first non-NULL value from the list. Example:
      SELECT COALESCE(NULL, 'Default Value');
    - NULLIF(): Returns NULL if two expressions are equal; otherwise, returns the first expression.
47. What is a temporary table? How is it different from a regular table?
    A temporary table is a table that exists only for the duration of a session or transaction. They are used for intermediate storage of data during complex operations. Unlike regular tables, they are automatically dropped when the session ends.
48. What is the difference between VARCHAR and TEXT data types?
    - VARCHAR: Stores variable-length strings with a defined maximum size. More efficient for shorter text.
    - TEXT: Stores long strings of text. It is less efficient for small strings but can hold much larger data.
49. How do you backup and restore a database?
    - **Backup**: Use BACKUP DATABASE to create a copy.
    - **Restore**: Use RESTORE DATABASE to restore from a backup.

50. What are common security best practices for SQL databases?
    - Use strong, encrypted passwords.
    - Implement least privilege access.
    - Regularly update software and apply security patches.
    - Encrypt sensitive data in transit and at rest.
    - Use parameterized queries to prevent SQL injection.