**Q: 1) What is your current project Architecture?**

**1. Presentation Layer (MVC)**

* **Controllers:** Handle user input, interact with services, and determine which view to display.
* **Views:** Define the user interface (UI) elements and display the data returned by controllers.
* **Models (View Models):** Define data structures specific to views, often used to represent data being transferred between views and controllers.

**2. Business Logic Layer (Service Layer)**

* **Services:** Contain business logic and interact with the data layer to process information.
* This layer is usually implemented using classes that encapsulate business logic, making it reusable and easier to test.

**3. Data Access Layer (Repository Layer)**

* **Repositories:** Manage data access, connecting to the database to perform CRUD operations.
* Often uses Entity Framework or Dapper to handle database interaction.
* **Data Models (Entities):** Represent the database schema as objects, mapping tables to classes.

**4. Database Layer**

* The actual database, such as SQL Server, stores data.
* **Stored Procedures, SQL Queries, or ORM Mappings** are used here to perform data retrieval and storage.

**Q: 2) What are your current project Features?**

**1. Model Binding and Data Validation**

* **Example**: Suppose you have a form on a website for user registration. ASP.NET MVC’s model binding automatically maps form input to properties in a model class, such as UserModel. You can also use data annotations like [Required], [EmailAddress], or [Range] in the model to enforce validation, making sure that users provide valid data before submission.

**2. Routing**

* **Example**: In an e-commerce application, custom routing can be used to create SEO-friendly URLs. For instance, instead of a URL like /Product/Details/5, you might have a route configuration that produces /Product/Laptop-XYZ, making URLs more descriptive and user-friendly. ASP.NET MVC lets you configure routing rules in the RouteConfig.cs file.

**3. Controllers and Action Methods**

* **Example**: A ProductController in an e-commerce project can have multiple action methods like Index, Details, Create, Edit, and Delete to handle CRUD operations. These methods receive requests, process them, and return appropriate responses or views.

**4. View Engine (Razor)**

* **Example**: Razor syntax allows embedding C# code within HTML files. For instance, a product list page might use a Razor view to loop over a list of Product objects and display their names, prices, and descriptions in HTML. The syntax @foreach (var product in Model.Products) in a Razor view allows efficient rendering of dynamic data.

**5. Dependency Injection (DI)**

* **Example**: Suppose your project relies on a service to manage product data. You might define an IProductService interface and inject it into your controller. ASP.NET MVC supports DI natively, making it easy to decouple components and improve testability by configuring services in Startup.cs or through a dependency injection container.

**6. Filters (Authorization, Action, and Exception Filters)**

* **Example**: In an intranet application, you can apply the [Authorize] filter to specific controllers or actions to restrict access to authorized users only. Custom action filters can also be used to log action execution time or handle errors globally.

**7. Bundling and Minification**

* **Example**: To improve performance, you might use bundling and minification to combine and compress CSS and JavaScript files. This reduces the number of HTTP requests and optimizes asset delivery, especially useful in applications with many front-end dependencies.

**8. Areas for Modular Development**

* **Example**: In a large application with several functional modules (like Admin, User, and Reporting), you can create "Areas" to organize controllers, views, and models. This keeps code organized and helps with modular development.

**9. Ajax Helpers and Partial Views**

* **Example**: For a dashboard that updates data dynamically, you could use Ajax helpers and partial views. For instance, clicking "Load More" might trigger an Ajax call that fetches additional data without refreshing the entire page. Partial views allow you to render only a part of a page with updated data.

**10. Custom Model Binding**

* **Example**: If your project has complex data inputs, like multi-part forms, you can create custom model binders to handle specialized binding. For instance, if users upload files along with form data, a custom model binder can map this to a FileUploadModel.

**Q: 3) What are your role and responsibilities?**

**Backend Developer (ASP.NET MVC)**

* **Responsibilities**:
  + Develops and maintains server-side logic and business logic within the MVC framework.
  + Implements controllers, models, and services for data processing and application flow.
  + Works on database interactions, including querying, data retrieval, and optimization.
  + Implements authentication and authorization mechanisms (e.g., ASP.NET Identity, Role-based Authentication).
  + Writes unit tests and performs code reviews to ensure code quality.

**4. Frontend Developer**

* **Responsibilities**:
  + Creates and maintains the application’s user interface, primarily using Razor views, HTML, CSS, and JavaScript.
  + Works on AJAX calls, validation, and user experience (UX) improvements.
  + Integrates with backend services, ensuring a seamless flow of data between the client and server.
  + Uses JavaScript frameworks (like jQuery or React, if applicable) to enhance front-end functionality.
  + Ensures the application is responsive and accessible across different devices and screen sizes.

**Q : 4) Which models are used in your current Project?**

In an ASP.NET MVC project, various models are used to represent the application's data and business logic. Here are the primary types of models typically used:

1. **Domain Models**: These models represent the core data and business logic of the application. They are often simple classes that map to database tables and encapsulate the properties and behaviours of the data.
2. **View Models**: These models are specifically designed for the views in your application. They aggregate data from multiple domain models and are shaped to meet the needs of the user interface. View models help separate the view logic from the domain logic.
3. **Data Transfer Objects (DTOs)**: DTOs are used to transfer data between layers, especially in scenarios involving APIs. They are often lightweight objects that contain only the data needed for a specific operation, helping to minimize the amount of data transmitted over the network.
4. **Entity Framework Models**: If using Entity Framework for data access, these models represent the database entities. They are typically generated based on the database schema and can include navigation properties for relationships between entities.
5. **Validation Models**: Models that include data annotations or custom validation logic to ensure that the data meets certain criteria before being processed or saved. This is often used in conjunction with view models to provide feedback on user input.
6. **Custom Models**: Depending on the complexity of the application, you may create custom models tailored to specific requirements, such as combining several domain models for specific operations.

**Q : 5) Tell me about SDLC used in Asp Dot Net MVC current project?**

The Software Development Life Cycle (SDLC) is a framework that outlines the processes and stages involved in developing software applications, including those using ASP.NET MVC. In the context of an ASP.NET MVC project, the SDLC typically follows several key stages:

1. **Planning**: This initial stage involves defining the project scope, objectives, and requirements. Stakeholders identify what the application should achieve, and preliminary feasibility studies may be conducted.
2. **Requirements Analysis**: Detailed requirements gathering takes place in this phase. Developers and business analysts work together to collect and document functional and non-functional requirements, ensuring a clear understanding of what the end product should entail.
3. **Design**: In the design phase, the system architecture is planned. For ASP.NET MVC, this includes defining the models, views, and controllers, as well as any necessary database design. The design may include wireframes and user interface designs.
4. **Implementation (Coding)**: During this phase, developers write the code for the application. In an ASP.NET MVC project, this involves creating the necessary controllers, models, views, and integrating other components like databases and APIs.
5. **Testing**: After coding, the application undergoes various testing processes, such as unit testing, integration testing, system testing, and user acceptance testing (UAT). The goal is to identify and fix any bugs or issues before deployment.
6. **Deployment**: Once testing is complete and the application is deemed stable, it is deployed to a production environment. This could involve uploading the application to a web server or cloud platform.
7. **Maintenance**: After deployment, the application enters the maintenance phase. This includes regular updates, bug fixes, and enhancements based on user feedback and changing requirements.
8. **Evaluation**: In some models, this final phase involves evaluating the application's performance and success against the initial goals. Feedback is collected to inform future projects or updates.

**Q : 6) What kind of document used in your current project?**

In an ASP.NET MVC project, various types of documents may be used to manage development and maintain project organization. Here are some common types:

1. **Requirements Documents**: Outline the functionalities and features needed in the application. They may include user stories or use cases.
2. **Design Documents**: Describe the architecture of the application, including diagrams for the overall structure, data flow, and interaction between components.
3. **API Documentation**: If your application exposes or consumes APIs, this document will describe the endpoints, request/response formats, and authentication methods.
4. **Technical Specifications**: Provide detailed information about the technologies used, including frameworks, libraries, and tools, as well as versioning details.
5. **Database Schemas**: Define the structure of the database, including tables, fields, relationships, and constraints.
6. **User Interface (UI) Mockups**: Visual representations of the UI, often created using design tools, to illustrate the look and feel of the application.
7. **Deployment Guides**: Outline steps and requirements for deploying the application to different environments (development, staging, production).
8. **Testing Plans**: Define the testing strategies, including unit tests, integration tests, and user acceptance tests, along with test cases and expected outcomes.
9. **User Manuals**: Provide instructions for end users on how to use the application, including workflows and troubleshooting tips.
10. **Change Logs**: Document changes made in each version of the application, including bug fixes, new features, and other modifications.
11. **Meeting Notes**: Capture discussions and decisions made during team meetings, which can help in tracking progress and responsibilities.

**Q : 7) How do you maintain version of your current project?**

Maintaining the version of an ASP.NET MVC project involves several best practices to ensure that your application remains up-to-date, secure, and efficient. Here are some strategies you can implement:

### 1. ****Use Version Control Systems****

* **Git**: Utilize Git or another version control system to manage your project code. This allows you to track changes, collaborate with others, and revert to previous versions if necessary.
* **Branching**: Use branches for new features, bug fixes, or experiments, merging them back to the main branch when ready.

### 2 ****Continuous Integration/Continuous Deployment (CI/CD)****

* Set up a CI/CD pipeline to automate the build, testing, and deployment processes. Tools like Azure DevOps, Jenkins, or GitHub Actions can help streamline these processes.

### 3. ****Documentation****

* Maintain up-to-date documentation for your codebase, including architecture, setup instructions, and API references. This can help new developers onboard quickly and assist with maintenance.

**Q : 8) What was most challenging work in your current project?**

When discussing the most challenging work in a project, it's helpful to consider various aspects such as technical difficulties, team dynamics, and project management. Here's an example response:

**One of the most challenging aspects of our project was integrating multiple software systems. We faced several hurdles during this process:**

1. **Technical Complexity**: The systems we needed to integrate were built on different platforms and used various programming languages, which made communication between them difficult. Ensuring that data flowed seamlessly without loss or corruption required extensive testing and debugging.
2. **Resource Limitations**: Our team was relatively small, and the complexity of the integration demanded a considerable amount of time and expertise. Balancing this workload while meeting deadlines was a significant challenge, requiring us to prioritize tasks and sometimes work overtime.

**Q : 9) How do you communicate to your client in your project?**

Effective communication with clients is crucial for the success of any project. Here are some key strategies:

1. **Establish Clear Channels**: Determine preferred communication methods (email, phone, video calls, project management tools) and make sure everyone is comfortable using them.
2. **Set Regular Check-ins**: Schedule regular meetings or updates to discuss progress, address concerns, and adjust plans as necessary.

**Q: 10) Explain complete process followed for development in your project?**

Developing an ASP.NET MVC project involves several stages, from planning and design to implementation and deployment. Here’s a complete overview of the typical process followed in ASP.NET MVC project development:

**1. Planning and Requirements Gathering**

* **Identify Stakeholders**: Engage with clients, users, and other stakeholders to understand their needs.
* **Gather Requirements**: Document functional and non-functional requirements (e.g., performance, security).
* **Define Scope**: Establish what will be included in the project and set realistic timelines.

**2. System Design**

* **Architecture Design**: Choose an architecture pattern (e.g., MVC) suitable for the project.
* **Database Design**: Design the database schema, including tables, relationships, and indexes.
* **UI/UX Design**: Create wireframes and prototypes to visualize the user interface and experience.

**3. Setting Up the Development Environment**

* **Select Tools**: Choose IDEs (like Visual Studio), source control (like Git), and project management tools.
* **Install Frameworks**: Set up ASP.NET MVC and any other required libraries or frameworks.

**4. Project Initialization**

* **Create Project**: Use Visual Studio to create a new ASP.NET MVC project.
* **Configure Project**: Set up necessary configurations in Web.config, including connection strings and app settings.

**5. Development**

* **Model Creation**: Develop the data models representing the application’s data structure.
* **Controller Development**: Create controllers to handle incoming requests and define actions.
* **View Creation**: Design views using Razor syntax to render HTML.
* **Routing Configuration**: Define routes in RouteConfig.cs to map URLs to controllers.
* **Implement Business Logic**: Add business logic to controllers or services as necessary.
* **Data Access Layer**: Create repositories or use Entity Framework to interact with the database.

**6. Testing**

* **Unit Testing**: Write unit tests for models and controllers to ensure functionality.
* **Integration Testing**: Test how different components work together.
* **User Acceptance Testing (UAT)**: Conduct testing with stakeholders to ensure requirements are met.

**7. Debugging**

* Use debugging tools in Visual Studio to identify and fix any issues in the code.
* Review logs and error messages to resolve unexpected behaviours.

**8. Deployment**

* **Select Hosting Environment**: Choose a hosting service (IIS, Azure, etc.).
* **Deploy the Application**: Publish the project and move it to the hosting environment.
* **Database Migration**: Apply any necessary migrations to set up the production database.

**9. Post-Deployment**

* **Monitor the Application**: Set up monitoring to track performance and usage.
* **Maintenance**: Schedule regular updates and bug fixes.

**10. Documentation**

* **Code Documentation**: Comment on code and create documentation for future developers.
* **User Manuals**: Develop manuals or guides for end-users.

**Q: 11) How do you communicate with team member in your project?**

Communicating effectively with team members in a project is crucial for success. Here are some strategies to consider:

1. **Regular Check-Ins:** Schedule regular meetings or check-ins to discuss progress, address concerns, and align on goals.
2. **Clear Channels:** Use established communication channels (like Slack, Microsoft Teams, or email).
3. **Project Management Tools:** Utilize tools like Trello, Asana, or Jira to track tasks, share updates, and manage deadlines, making it easier for everyone to stay informed.

**Q : 12 ) If your client ask for early delivery , how will you manage in your project?**

If a client requests early delivery, it's important to evaluate the feasibility and impact of meeting their request without compromising quality. Here’s how I would approach managing an early delivery:

1. **Prioritize Tasks**
   * Identify required tasks for early delivery and prioritize them, and putting less Prioritizetasks on hold.
2. **Reallocate Resources**
   * Evaluate the availability of team members and reassign resources as necessary to accelerate work on prioritized tasks.
   * If possible, bring in additional resources or allocate overtime (with team consent) to expedite completion.
3. **Adjust the Timeline**
   * Compress or overlap tasks where feasible. This may involve parallelizing work streams that were initially scheduled to be sequential.

**Q: 13) How will gather requirements and where do record or do you have any tool in your project?**

**Tools for Requirements Management**

If your project has a dedicated tool for managing requirements, it may be one of the following:

* **Jira**: Widely used for managing user stories and requirements, especially in Agile environments.
* **Microsoft Azure DevOps**: Offers boards and backlog management for tracking requirements and user stories.

**Q : 14 ) How do handle change request in your project?**

**1. Analyze the Impact of the Change**

* **Assess impact**: Work with relevant team members to analyse how the change will impact the project’s scope, timeline, budget, and resources.

**3. Update Project Documentation and Plans**

* **Revise project plans**: If approved, adjust timelines, resources, budgets, and deliverables in project documentation.

**Q : 15 ) How do write unit test cases in your project?**

* **Identify the code unit**: Choose a code unit to test, like a function or method.
* **Create test cases**: Write test cases that cover normal, error, and edge scenarios.
* **Write test code**: Write code to run the unit with specific inputs, validate the expected outcomes, and confirm that the code unit works as intended.
* **Use descriptive test names**: Give your tests descriptive names so that you can understand what the test does and what its expected outcome is.
* **Write deterministic tests**: Deterministic tests either pass all the time or fail all the time until fixed.
* **Isolate the unit**: Make sure the unit is tested in isolation. You can use stubs or mocks to simulate the behaviour of external dependencies.
* **Use assertions**: Assertions can help developer
* s quickly identifies issues with their code.

**Q: 16) How do you maintain your code quality, any tool that you use?**

**3. Version Control Best Practices**

* **Git Best Practices**: Small, meaningful commits, well-written commit messages, and using branches for new features or bug fixes can help keep the project organized.

**Q : 16 ) How to create unit test by using MS test ?**

MS Test is Microsoft's testing framework, often used in Visual Studio, which allows you to test .NET code. Here’s a step-by-step guide.

1. **Create a Test Project**: Open Visual Studio and create a new MS Test project by selecting **Unit Test Project (.NET Framework)** or **Unit Test Project (.NET Core)** (based on your application).
2. **Install MS Test NuGet Package**: If you do not already have MS Test installed, add the MSTest.TestFramework and MSTest.TestAdapter NuGet packages to your test project.

**2. Write Your Code to Be Tested**

Before creating tests, you should have a piece of code that you want to test. For example, let’s say we have a simple class Calculator with an Add method:

public class Calculator

{

public int Add(int a, int b)

{

return a + b;

}

}

**3. Create a Test Class**

1. In your test project, create a new class (e.g., CalculatorTests.cs).
2. Use the TestClass attribute to mark this as a test class.

**4. Write Test Methods**

Each test method should be decorated with the [TestMethod] attribute. Here’s how you would write a unit test for the Add method in the Calculator class:

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace MyProject.Tests

{

[TestClass]

public class CalculatorTests

{

[TestMethod]

Public void AddSum()

{

// Arrange

var calculator = new Calculator();

int a = 5;

int b = 3;

int expected = 8;

// Act

int result = calculator.Add(a, b);

// Assert

Assert.AreEqual(expected, result);

}

}

}

**Explanation of Test Structure**

1. **Arrange**: Set up any data or objects you need.
2. **Act**: Call the method you are testing.
3. **Assert**: Use assertions to verify that the result matches expectations.

**5. Run the Test**

1. Open the **Test Explorer** in Visual Studio (go to **Test > Test Explorer**).
2. Build your solution, and you should see your test appear in the Test Explorer.
3. Run the test by right-clicking on it in the Test Explorer and selecting **Run** or by using the **Run All** button.

**6. Verifying the Result**

If the test passes, it means the Add method works as expected. If it fails, the Test Explorer will indicate the failure, and you can investigate why the output didn’t match your expectation.

**Example of Other Assertions**

MS Test provides several assertions:

* Assert.IsTrue(condition): Checks if a condition is true.
* Assert.IsFalse(condition): Checks if a condition is false.
* Assert.AreEqual(expected,actual): Checks if two values are equal.
* Assert.AreNotEqual(expected,actual): Checks if two values are not equal.
* Assert.ThrowsException<ExceptionType>(action): Checks if a specific exception is thrown.

Each of these can be used in your unit tests to validate various behaviours and edge cases.

**Q : 17 ) How to Create a MS Test Project by Open Visual Studio ?**

To create a **MS Test Project** in Visual Studio, follow these steps:

1. **Open Visual Studio**:
   * Start Visual Studio from your Start menu or application list.
2. **Create a New Project**:
   * Go to **File > New > Project...** or select **"Create a new project"** from the Visual Studio start window.
3. **Select MS Test Project Template**:
   * In the "Create a new project" window, use the **Search** box and type "MS Test" to find the **MS Test Project** template.
   * Select **MS Test Project** from the list.
   * Click **Next**.
4. **Configure the Project**:
   * In the **Configure your new project** window, name your project (e.g., "MyMSTestProject").
   * Choose the location where you want to save it, and provide a solution name if necessary.
   * Click **Create**.
5. **Add Test Methods**:
   * Visual Studio will create a basic project structure with a default test class (UnitTest1.cs) containing a sample test method.
   * Open UnitTest1.cs, and you can add more test methods by creating new methods with the [Test Method] attribute.
6. **Run Your Tests**:
   * To run your tests, go to the **Test** menu and select **Run All Tests**. You can also open the **Test Explorer** from **Test > Test Explorer** to see individual test results.

This setup will allow you to start writing and executing unit tests using MS Test in Visual Studio.

**Scenario Based Interview Questions and Answer**

**1)** **How to optimize more record through UI in Dot Net?**

**Answer 🡪** To optimize the handling of a large number of records in a .NET application's user interface (UI), you can use various strategies to enhance performance and user experience. Here are several techniques:

**1. Pagination**

* **Implementation:**
  + Use server-side pagination to fetch only the required data for each page.
  + Use controls like DataGrid, GridView, or ListView with built-in pagination features.
* **Benefits:** Reduces the memory footprint and rendering time.
* **Example:**

public IActionResult GetRecords(int pageNumber,int pageSize)

{

var records = db.Records.Skip((pageNumber - 1) \* pageSize)

.Take(pageSize).ToList();

return Json(records);

}

**2. Asynchronous Data Loading**

* **What:** Fetch and load records asynchronously to avoid blocking the UI thread.
* **Implementation:** Use async and await for data operations.
* **Benefits:** Keeps the UI responsive.
* **Example:**

private async Task LoadDataAsync()

{

var data = await FetchRecordsAsync();

// Bind data to UI

}

**3. Client-Side Filtering and Searching**

* **What:** Allow users to filter and search records within the loaded dataset.
* **Implementation:** Use JavaScript or frameworks like Angular, React, or Blazer for a faster client-side experience.
* **Benefits:** Enhances usability without additional server calls.

**4. Use Caching**

* **What:** Cache frequently accessed data on the client or server.
* **Implementation:** Use in-memory caching with MemoryCache or distributed caching with Redis.
* **Benefits:** Reduces database queries and improves load times.

**Q 2) Scenario : Exception Handling in .NET**

**Question:**  
You are working on a financial application where unhandled exceptions must be logged, and the application should not crash. How would you design exception handling for this scenario?

**Answer:**

* Use a try-catch block to handle known exceptions gracefully.
* Implement a global exception handler using middleware in ASP.NET Core or Application\_Error in older ASP.NET.
* Log exceptions using a logging framework like Serilog, NLog, or Microsoft.Extensions.Logging.
* Return a generic error message to the user, ensuring sensitive information is not exposed.
* Example middleware in ASP.NET Core:

app.UseExceptionHandler(errorApp =>

{

errorApp.Run(async context =>

{

var exception = context.Features.Get<IExceptionHandlerFeature>()?.Error;

// Log the exception

logger.LogError(exception, "Unhandled Exception");

context.Response.StatusCode = 500;

await context.Response.WriteAsync("An unexpected error occurred.");

});

});

**3) Scenario: Dependency Injection**

**Question:**  
You are tasked with developing a web API. Explain how you would implement dependency injection in ASP.NET Core.

**Answer:**

* ASP.NET Core has built-in support for dependency injection.
* Register dependencies in the Startup.cs file using the ConfigureServices method:

services.AddScoped<IMyService, MyService();

services.AddTransient<IAnotherService, AnotherService>();

services.AddSingleton<ILogger, Logger>();

* Use constructor injection to inject the services into the controller:

public class MyController : ControllerBase

{

private readonly IMyService \_myService;

public MyController(IMyService myService)

{

\_myService = myService;

}

}

**4) Scenario: Handling Large Data in Web Applications**

**Question:** How would you optimize an ASP.NET Core web API to handle large datasets efficiently?

**Answer:**

1. **Pagination:**
   * Return data in chunks using paging techniques:

SELECT \* FROM Orders ORDER BY OrderID OFFSET @PageSize \* (@PageNumber - 1) ROWS FETCH NEXT @PageSize ROWS ONLY;

1. **Streaming Data:**
   * Use IAsyncEnumerable to stream data from the API instead of loading it all into memory.
2. **Compression:**
   * Enable GZIP compression using middleware:

app.UseResponseCompression();

1. **Caching:**
   * Use caching for frequently accessed data with in-memory or distributed cache.
2. **Database Optimization:**
   * Optimize database queries by adding indexes, avoiding unnecessary joins, and selecting only required columns.

**5) Scenario: Implementing Authentication and Authorization**

**Question:** You are building a multi-tenant web application. How would you implement authentication and authorization in ASP.NET Core?

**Answer:**

**Types of Authentications in Asp.Net MVC Project:**

1. **Forms Authentication :**

 Used for handling authentication in web applications.

 Stores authentication credentials (username/password) in a database.

1. **Windows Authentication:**

 Uses Windows user accounts for authentication.

 Best for intranet applications where users are within the same Active Directory.

**3. Token-Based Authentication (JWT):**

Uses JSON Web Tokens (JWT) for authentication.

**4. Claim-Based Authentication:**

Users are authenticated based on a set of claims (e.g., role, email, permission).

* **Authentication:** Use JWT tokens for stateless authentication. Implement Microsoft.AspNetCore.Authentication.JwtBearer.

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("YourSecretKey"))

};

});

* **Authorization:** Use policies for role-based or claim-based access:

services.AddAuthorization(options =>

{

options.AddPolicy("AdminOnly", policy => policy.RequireRole("Admin"));

});

Apply policies in controllers:

[Authorize(Policy = "AdminOnly")]

public IActionResult AdminAction()

{

return Ok("Welcome, Admin!");

}

**6) Scenario: Performance Optimization**

**Question:** Your ASP.NET Core web application is experiencing performance issues under heavy load. How would you troubleshoot and resolve them?

**Answer:**

1. **Identify Bottlenecks:**
   * Use tools like Application Insights, Performance Profiler, or ELMAH to identify slow code segments.
2. **Optimize Database Queries:**
   * Ensure queries are optimized and use stored procedures where appropriate.
   * Use **SQL Profiler** to check slow queries.
3. **Asynchronous Programming:**
   * Use async/await to perform non-blocking I/O operations.
4. **Caching:**
   * Cache frequently accessed data using in-memory or distributed cache (e.g., Redis).
5. **Load Balancing:**
   * Distribute traffic using a load balancer.

**7) Scenario : Migrating to .NET Core**

**Question:** How to migrate an existing .NET Framework application to .NET Core. What steps would you take?

**Answer:**

1. **Assess the Application:**
   * Check for compatibility using the .Net Portability Analyzer.
2. **Update Dependencies:**
   * Ensure third-party libraries have .NET Core versions or replacements.
3. **Refactor Code:**
   * Update legacy APIs and remove unsupported features.
4. **Create a .NET Core Project:**
   * Start with a new project and migrate code in modules.
5. **Test Thoroughly:**
   * Use unit and integration tests to ensure functionality.
6. **Deploy Incrementally:**
   * Deploy the application in phases, starting with non-critical components.

**8) Scenario : How to get Real-Time Updates or Notification.**

**Question:** How would you implement real-time data updates in an ASP.NET Core application?

**Answer:**

* Use SignalR for real-time communication.
* Add SignalR to the project in Startup.cs:

services.AddSignalR();

app.UseEndpoints(endpoints =>

{

endpoints.MapHub<ChatHub>("/chatHub");

});

* Create a Hub for communication:

public class ChatHub : Hub

{

public async Task SendMessage(string user, string message)

{

await Clients.All.SendAsync("ReceiveMessage", user, message);

}

}

* Use JavaScript on the client to connect and receive messages:

javascript

const connection = new signalR.HubConnectionBuilder()

.withUrl("/chatHub")

.build();

connection.on("ReceiveMessage", (user, message) => {

console.log(`${user}: ${message}`);

});

connection.start().catch(err => console.error(err));

**9) Scenario : How to Troubleshoot Asp.net MVC Project or Application.**

Troubleshooting in an **ASP.NET MVC** project involves identifying, diagnosing, and fixing issues that occur during development, deployment, or runtime. Following are some common troubleshooting steps:

**1. Check Error Messages and Logs**

* **Enable Developer Exception Page**  
  In Startup.cs (for .NET Core)
* Global.asax.cs (for .NET Framework), ensure errors are displayed in development:

app.UseDeveloperExceptionPage();

* **Custom Error Pages**  
  If your app is showing a generic error page, modify web.config:

<system.web>

<customErrors mode="Off"/>

</system.web>

* **Logging with Serilog/NLog**  
  Use a logging library like Serilog or NLog to capture detailed logs.

**2. Debugging with Visual Studio**

**3. Database Connection Issues**

* **Check Connection String** in appsettings.json or web.config
* **Use SQL Profiler** to track queries

**6. Authentication & Authorization Issues**

services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme).AddCookie();

**8. Deployment Issues**

* **Check IIS Logs** (C:\inetpub\logs\LogFiles)
* **Verify Permissions** for IIS user
* **Set Environment Variables** for production settings in appsettings.Production.json

Would you like help troubleshooting a specific issue in your ASP.NET MVC project?

**Q 10 ) What is your the daily routing in your current project.**

A Dot Net developer’s daily routine in a project typically involves.

**Morning Routine**

1. **Checking Emails & Updates**
   * Review emails, messages, and any updates from team members.
   * Check for pull request reviews or feedback.
2. **Daily Standup Meeting (Scrum)**
   * Discuss what was completed yesterday.
   * Share today’s tasks and any blockers.
   * Get updates from other team members.

**Development Work**

1. **Coding & Feature Development**
   * Work on assigned user stories or tasks.
   * Write new features or modify existing functionality using C#, ASP.NET, Blazor, etc.
   * Implement front-end (if using .NET with React/Angular) and back-end logic.
2. **Debugging & Fixing Issues**
   * Investigate and fix bugs reported by QA or end-users.
   * Use debugging tools like Visual Studio, logs, and breakpoints.
3. **Database Work**
   * Write or modify SQL queries, stored procedures, or Entity Framework models.
   * Optimize queries for better performance.
4. **Unit Testing & Code Reviews**
   * Write unit and integration tests (MSTest, xUnit, NUnit).
   * Review code submitted by other developers.
   * Refactor code based on feedback and best practices.

**Afternoon Tasks**

1. **Collaborating with Team**
   * Discuss technical challenges with colleagues.
   * Attend sprint planning, backlog grooming, or design meetings.
   * Sync with DevOps teams for deployments or CI/CD pipeline updates.
2. **Documentation & Learning**
   * Update technical documentation (API contracts, architecture diagrams, etc.).
   * Explore new frameworks or .NET updates for better performance.

**End of the Day**

1. **Wrapping Up & Status Update**
   * Push final changes to GitHub/Azure DevOps/GitLab.
   * Update task status in Jira, Azure DevOps, or Trello.
   * Prepare notes for the next day’s work.

**Notes on Git Version Control Tool : -**

**There are two ways to work with Git: -**

* 1. **By using command Line Prompt.**
  2. **By using GUI through Visual Studio or any other Editor.**

**To Work with command Line Prompt, you need to Download Git from Git site into your Local System.**

**There are some Server to store our code on it, such as GitHub, Azure DevOps, Bitbucket etc**

**List of all Git commands**

Here’s a comprehensive list of commonly used Git commands that are particularly useful for .NET developers (or any developers working with version control in Git):

## **1. Configuration Commands**

* git config --global user.name "Your Name"  
  Set your name for Git commits.
* git config --global user.email "youremail@example.com"  
  Set your email for Git commits.
* git config --global core.autocrlf true  
  Ensure consistent line endings (useful for Windows environments).
* git config --list  
  View the current configuration.

## **2. Repository Setup**

* git init  
  Initialize a new Git repository.
* git clone <repository-url>  
  Clone an existing repository to your local machine.

## **3. Basic Workflow**

* git status  
  View the status of your working directory and staging area.
* git add <file>  
  Stage a file for commit.
* git add .  
  Stage all changed files. (means it is in stage area )
* git commit -m "Your commit message"  
  Commit staged changes with a message.
* git commit --amend  
  Modify the most recent commit (e.g., fix the commit message).

## **4. Branching and Merging**

* git branch  
  List all branches in the repository.
* git branch <branch-name>  
  Create a new branch.
* git checkout <branch-name>  
  Switch to a different branch.
* git checkout -b <branch-name>  
  Create and switch to a new branch.
* git merge <branch-name>  
  Merge a branch into the current branch.
* git branch -d <branch-name>  
  Delete a branch locally.

## **5. Remote Repository Management**

* git remote -v  
  View remote repositories.
* git remote add <name> <url>  
  Add a new remote repository.
* git pull  
  Fetch and merge changes from the remote repository.
* git fetch  
  Fetch changes from the remote repository (without merging).
* git push  
  Push commits to the remote repository.
* git push --set-upstream origin <branch-name>  
  Set the upstream branch for future pushes.

## **6. Stashing Changes**

* git stash  
  Save uncommitted changes and clean the working directory.
* git stash pop  
  Apply the most recent stash and remove it from the stash list.
* git stash list  
  View all stashed changes.
* git stash drop  
  Delete a specific stash.

## **7. Viewing History**

* git log  
  View the commit history.
* git log --oneline  
  View a simplified commit history.
* git log --graph --oneline  
  View a visual representation of the commit history.
* git show <commit-hash>  
  View details of a specific commit.

## **8. Undoing Changes**

* git checkout -- <file>  
  Discard changes in the working directory.
* git reset <file>  
  Unstage a file without discarding changes.
* git reset --soft HEAD~1  
  Undo the last commit but keep changes staged.
* git reset --hard HEAD~1  
  Undo the last commit and discard changes.

## **9. Tagging**

* git tag <tag-name>  
  Create a lightweight tag.
* git tag -a <tag-name> -m "Tag message"  
  Create an annotated tag.
* git push origin <tag-name>  
  Push a tag to the remote repository.
* git tag -d <tag-name>  
  Delete a tag locally.

## **10. Advanced Commands**

* git rebase <branch-name>  
  Reapply commits on top of another branch.
* git cherry-pick <commit-hash>  
  Apply changes from a specific commit to the current branch.
* git blame <file>  
  Show who made changes to each line in a file.
* git diff  
  Show changes between commits, branches, or your working directory.

## **11. Cleaning Up**

* git clean -f  
  Remove untracked files.
* git clean -fd  
  Remove untracked files and directories.

## **12. Aliases (Optional)**

* git config --global alias.st status  
  Create a shorthand alias for git status.
* git config --global alias.co checkout  
  Create a shorthand alias for git checkout.

### Useful Tips for .NET Developers

1. **.gitignore**: Ensure you include files like bin/, obj/, and \*.user in your .gitignore file to avoid committing unnecessary or machine-specific files.
2. **Commit Hooks**: Use Git hooks to automate tasks like running tests or formatting code before commits.
3. **Git GUIs**: Tools like Visual Studio, Git Extensions, or Sourcetree can provide a visual interface to Git commands.

**End of File**