ASP.Net Core Web Api(Based on resume profile)

1. What is the Repository Pattern? How have you used it?

Ans. The Repository Pattern abstracts the data layer, allowing business logic to interact with the data source through a clean interface.

In my projects, I created generic repositories with CRUD operations to reduce code duplication and improve testability.

1. Explain JWT Authentication and how you implemented it.

Ans. JWT (JSON Web Token) is used for stateless authentication between client and server. I implemented JWT in my Web API projects for secure login. On successful login, a token is generated using user claims and secret key, which is then used In Authorization headers for subsequent API requests. I also implemented role-based access using claims inside JWT.

1. How did you handle role-based access control (RBAC)?

Ans. In my projects, I assigned role like Admin, HR, User and used [Authorize (Roles = “Admin”)] in controllers to restrict access.

Roles were stored in the database and assigned to users at login. JWT tokens were generated with roles as claims.

1. How do you ensure performance in SQL queries and stored procedures?

Ans. I optimized queries using

* Indexes on frequently filtered columns
* Avoiding select \* in stored procedures
* Proper use of joins and avoiding subqueries
* Analyzing execution plans for optimization

1. What is the difference between Code-First and Database-First in EF Core?

Ans.

* Code- First: Define models in code, and EF Core generates the database.
* Database-First : Generate models from an existing database using scaffolding.

1. How do you manage migrations in EF Core?

Ans.

* I use Add-Migration and Update-Database to apply schema changes.
* For production, I generate SQL scripts for controlled DB updates.

1. Explain Dependency Injection in Asp.net Core.

Ans. Dependency Injection is built-in in ASP.NET Core. It allows injecting dependencies like services repositories via constructors. I register services in Startup.cs using services.AddScoped<IRepositary>() and inject them into controllers.

1. How do you schedule background tasks in .Net Core?

Ans. I’ve implemented scheduled email functionality using Hangfire and also basic IHostedService for simpler jobs. Hangfire helped me queue and retry email jobs asynchronously.

1. How do you handle versioning in Web API?

Ans. I use URI versioning (api/v1/controller) and in some cases attribute-based versioning using

[ApiVersion (“1.0”)] to maintain backward compability.

1. What tools to do you use for testing APIs?

Ans. I use Postman to test and documents APIs.

1. Why do you want to switch your current company?

Ans. I have grown technically in my current company, handling API development, security, and database optimization independently. However, I’m now seeking better growth opportunities, both financially and in terms of exposure to new challenges and technologies.

1. What was your biggest challenge in the last project and how did you solve it?

Ans. In PMT Tank, migrating legacy systems to .Net Core with clean architecture was challenging due to tight deadlines and complex data. I used Generic Repository Pattern and optimized stored procedures to streamline performance and maintabilty.

Advanced .Net Core & C# Questions

1. Explain the SOLID principles and give examples of how you applied then in your projects.

Ans. Solid stands for Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion.

* Single Responsibility: Each class in my project was built to handle one responsibility. For example, in the PMT Tank project, the repository class was solely responsible for data access logic.
* Open/Close: By using the Repositary Pattern and dependency injection, I ensured that classes were open for extensions( new functionality added via new implementation) but closed for modification.
* Liskov Substitution: I made sure that derived classes ( like specific repository type) could be substituted without affecting the applications correctness
* Interface Segregation: I divided interfaces into smaller, more specific ones to avoid forcing classes to implement unused methods.
* Dependency Inversion: Through constructor injection in ASP.Net Core, high-level modules depended on abstractions ( interfaces) rather than concrete implementations, making the code more testable and maintainable.

1. What are Middleware in ASP/NET Core and how do you implement a custom middleware?

And. Middleware components are pieces of code that can handle requests and responses in the pipeline.

To create custom middleware,

* Define a class with an Invike or InvokeAsync method that takes an HttpContext.
* Finally, register the middleware in the Startup.cs using app.Use Middleware<CustomMiddleware>().

1. How do you secure your Web APIs against common security threats?

Ans. I secure APIs using a layered approach:

* JWT Authentication: To ensure each request is authenticated.
* CORS Policy: Configured correctly to limit which domains can interact with the API.

1. Can you explain asynchronous programming in C# and why it’s important?.

Ans. Asynchronous programming in C# is accomplished using the async and await keywords. It allows the application to handle long-running tasks (like I/O operations or HTTP requests) without blocking the main thread, improving responsiveness and scalability. For example, when querying the database with EFCore, using asynchronous methods such as ToListAsync () prevents thread blockage during heavy data retrieval.

1. What is Entity Framework Core and why do you prefer it over ADO.Net?

Ans. Entity Framework Core (EF Core) is and Object-Relational Mapping (ORM) framework that simplifies data access in .Net application by allowing developers to interact with databases using strongly typed C# objects rather than SQL queries. I prefer EF core because:

* It accelerates development with built-in migrations support for versioning the database schema.
* It reduces boilerplate code through LINQ queries.
* It automatically handles mapping between objects and database tables, which improves maintainability and testability.

1. How does LINQ improve the development process in .NET?

Ans. LINQ ( Language Integrated Query) offers a consistent syntax for querying different data sources, such as databases, XML, and, and in-memory collections. It simplifies data retrieval by allowing developers to write expressive queries in C# rather than constructing raw SQL queries. This results in improved readability, better compile-time checking, and easier debugging.

1. Can you provide an example of how you’ve collaborated with frontend teams?

Ans. While working on projects, I collaborated with the frontend team by coordinating API specifications early in the development process. We held regular stand-ups and code review sessions to ensure that API endpoints were consistent and met the UI requirements. This close collaboration allowed us to integrate backend services seamlessly with the Angular-based frontend, resulting in a smoother user experience.

C#

1. What is the difference between ref and out in C#?

Ans. Both ref and out are used to pass arguments by reference, allowing a method to modify the value of a variable defined outside the method.

* Ref requires variable to be initialize before passing.
* Out doesn’t require initialization; It must be initialized inside the called method.

1. What is a delegate?

Ans. A delegate is a type-safe function pointer that holds references to methods with a specific signature. Its’ used for implementing callbacks and event handling.

1. Difference between IEnumerable, IQueryable, and List?

Ans.

* Use IEnumerable for in-memory iteration and supports basic LINQ operations.
* IQueryable extends IEnumerable and is designed for querying remote data sources like databases. It allows LINQ queries to be translated into SQL, making it efficient for large datasets.
* List<T> is a concrete collection class used for storing and manipulating data in memory. It supports immediate execution of queries and fast data access.