Frequently asked and likely questions for a candidate with 8+ years of experience in Web API Core in .NET, along with detailed answers to help you practice and appear for an interview with confidence:

**1. \*\*Question:\*\* What is ASP.NET Core Web API, and how does it differ from ASP.NET Web API?**

\*\*Answer:\*\* ASP.NET Core Web API is a framework for building RESTful services on the .NET Core platform. It is a cross-platform, high-performance, open-source framework for building modern, cloud-based, internet-connected applications. It differs from ASP.NET Web API in several key ways:

- \*\*Cross-Platform:\*\* ASP.NET Core runs on Windows, macOS, and Linux.

- \*\*Unified Framework:\*\* ASP.NET Core combines the MVC and Web API frameworks into a single framework.

- \*\*Modular and Lightweight:\*\* ASP.NET Core is modular, allowing developers to include only the necessary packages.

- \*\*Built-in Dependency Injection:\*\* ASP.NET Core has a built-in dependency injection mechanism.

- \*\*Improved Performance:\*\* ASP.NET Core is designed to be faster and more efficient than its predecessor.

**2. \*\*Question:\*\* Can you explain the middleware pipeline in ASP.NET Core?**

\*\*Answer:\*\* The middleware pipeline in ASP.NET Core is a sequence of request delegates, called one after the other, where each can perform operations before and after the next delegate in the pipeline. Middleware components are configured in the `Startup` class’s `Configure` method. Each middleware component can:

- \*\*Handle Requests:\*\* Terminate the request without passing it to the next component.

- \*\*Pass Requests:\*\* Call the next middleware component in the pipeline.

- \*\*Modify Responses:\*\* Perform actions on the outgoing response.

Example configuration in `Startup.cs`:

```csharp

public void Configure(IApplicationBuilder app)

{

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

```

**3. \*\*Question:\*\* How is dependency injection implemented in ASP.NET Core?**

\*\*Answer:\*\* ASP.NET Core has built-in support for dependency injection (DI). Services are registered in the `Startup` class within the `ConfigureServices` method, and they are then injected into constructors of classes where they are needed. ASP.NET Core DI supports three types of service lifetimes:

- \*\*Transient:\*\* Created each time they are requested.

- \*\*Scoped:\*\* Created once per request.

- \*\*Singleton:\*\* Created once and shared throughout the application's lifetime.

Example of registering services:

```csharp

public void ConfigureServices(IServiceCollection services)

{

services.AddTransient<IMyTransientService, MyTransientService>();

services.AddScoped<IMyScopedService, MyScopedService>();

services.AddSingleton<IMySingletonService, MySingletonService>();

services.AddControllers();

}

```

**4. \*\*Question:\*\* What is the role of the `Startup` class in ASP.NET Core?**

\*\*Answer:\*\* The `Startup` class in ASP.NET Core configures the application's services and defines the middleware pipeline. It contains two main methods:

- \*\*ConfigureServices:\*\* Used to register services with the dependency injection container.

- \*\*Configure:\*\* Used to define how the application responds to HTTP requests by configuring the middleware components.

Example:

```csharp

public class Startup

{

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

```

**5. \*\*Question:\*\* Explain how routing works in ASP.NET Core Web API.**

\*\*Answer:\*\* Routing in ASP.NET Core Web API is the process of mapping incoming HTTP requests to the appropriate action methods in controllers. There are two main types of routing:

- \*\*Conventional Routing:\*\* Defined in the `Startup` class using the `UseEndpoints` method.

- \*\*Attribute Routing:\*\* Defined by placing attributes directly on controller actions.

Example of attribute routing:

```csharp

[Route("api/[controller]")]

[ApiController]

public class ProductsController : ControllerBase

{

[HttpGet]

public IEnumerable<Product> Get() { ... }

[HttpGet("{id}")]

public Product Get(int id) { ... }

[HttpPost]

public void Post([FromBody] Product product) { ... }

}

```

**6. \*\*Question:\*\* What are Filters in ASP.NET Core, and how are they used?**

\*\*Answer:\*\* Filters in ASP.NET Core are used to run code before or after certain stages in the request processing pipeline. There are several types of filters:

- \*\*Authorization Filters:\*\* Run before the execution of an action method to determine if the user is authorized.

- \*\*Resource Filters:\*\* Run after authorization and before model binding.

- \*\*Action Filters:\*\* Run before and after the execution of an action method.

- \*\*Exception Filters:\*\* Run when an unhandled exception occurs during the execution of an action.

- \*\*Result Filters:\*\* Run before and after the execution of the action result.

Example of an action filter:

```csharp

public class MyActionFilter : IActionFilter

{

public void OnActionExecuting(ActionExecutingContext context) { ... }

public void OnActionExecuted(ActionExecutedContext context) { ... }

}

```

Registering a filter globally:

```csharp

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers(options =>

{

options.Filters.Add(new MyActionFilter());

});

}

```

**7. \*\*Question:\*\* How do you handle exceptions in ASP.NET Core Web API?**

\*\*Answer:\*\* ASP.NET Core provides several ways to handle exceptions:

- \*\*Exception Handling Middleware:\*\* Catch and handle exceptions globally.

- \*\*Exception Filters:\*\* Handle exceptions at the action or controller level.

- \*\*Try-Catch Blocks:\*\* Handle exceptions within specific actions.

Example of using exception handling middleware:

```csharp

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

app.UseHsts();

}

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

```

Example of a custom exception filter:

```csharp

public class CustomExceptionFilter : IExceptionFilter

{

public void OnException(ExceptionContext context)

{

// Handle exception and set the response

}

}

```

**8. \*\*Question:\*\* What is the purpose of the `IApplicationBuilder` interface in ASP.NET Core?**

\*\*Answer:\*\* The `IApplicationBuilder` interface is used to configure the application's request processing pipeline. It provides methods to define the middleware components that handle HTTP requests. It is typically used in the `Configure` method of the `Startup` class.

Example usage:

```csharp

public void Configure(IApplicationBuilder app)

{

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

```

**9. \*\*Question:\*\* Explain Model Binding and Validation in ASP.NET Core.**

\*\*Answer:\*\* Model binding in ASP.NET Core is the process of mapping data from HTTP requests to action method parameters. Model validation is the process of ensuring the data meets certain criteria before processing.

Example of model binding and validation:

```csharp

public class Product

{

[Required]

public string Name { get; set; }

[Range(1, 100)]

public int Quantity { get; set; }

}

[HttpPost]

public IActionResult CreateProduct([FromBody] Product product)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

// Process the valid product

return Ok();

}

```

**10. \*\*Question:\*\* How do you implement authentication and authorization in ASP.NET Core Web** **API?**

\*\*Answer:\*\* Authentication and authorization are implemented using middleware and attributes in ASP.NET Core. Authentication verifies the identity of the user, and authorization determines if the authenticated user has permission to access a resource.

Example of configuring authentication:

```csharp

public void ConfigureServices(IServiceCollection services)

{

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

// Set issuer, audience, and signing key

};

});

services.AddAuthorization();

services.AddControllers();

}

public void Configure(IApplicationBuilder app)

{

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

```

Using authorization attributes:

```csharp

[Authorize]

[ApiController]

[Route("api/[controller]")]

public class SecureController : ControllerBase

{

[HttpGet]

public IActionResult GetSecureData()

{

return Ok("This is secure data");

}

}

```

These questions and answers should provide a solid foundation for preparing for an interview focused on ASP.NET Core Web API. Good luck!