Asp.Net Core Important Interview Questions

<https://www.codingninjas.com/studio/library/interview-questions-for-net-core>

<https://code-maze.com/net-core-web-api-ef-core-code-first/>

# **JWT Authentication in ASP.NET Core Web API**

<https://code-maze.com/authentication-aspnetcore-jwt-1/>

### 1. What is .NET Core Framework?

.NET Core is an open-source web development framework that is the successor of .NET and both are maintained by Microsoft.

.NET Core is also cross-platform meaning that you can use it on multiple operating systems/ platforms- Linus, macOS, and Windows. Its important features include sharable core libraries and others that come as NuGet packages. You can add the packages as needed to optimize the performance of the application, make it easier to maintain, and decrease its memory footprint. It follows a more modular approach in comparison to its predecessor, i.e. instead of installing the whole of the .NET framework, you can just install or use what is required. Also, .NET Core comes with base class libraries whose code is largely similar to that of .NET Framework class libraries.

.NET is a software framework created by Microsoft that helps developers build applications for computers, mobile devices, and the web. It provides a set of tools and libraries that make it easier for developers to create software. .NET Core is a free, open-source, cross-platform framework for building modern applications.

### **2. What is ASP.NET Core?**

[ASP.NET](https://www.codingninjas.com/studio/library/asp-net) Core is a framework for building web applications using .NET Core. It includes a rich set of tools and libraries for building scalable and secure web applications.

Some of the key features of ASP.NET Core include **middleware components for handling HTTP requests and responses**, support for web sockets and SignalR for real-time communication, and support for multiple authentication and authorization schemes.

Overall, ASP.NET Core is a powerful and versatile web framework that enables developers to build fast, scalable, and cross-platform web applications and services.

### **3. What are the advantages of ASP.NET Core over ASP.NET?**

ASP.NET Core has several advantages over ASP.NET. Some of these include:

* **Cross Platform Compatibility**
* **Lightweight and less resource consuming**
* **Better efficiency and performance**
* **Open-source**
* **Built-in API Support**

### **4. What are the features provided by ASP.NET Core?**

Some of the key features of ASP.NET Core include middleware components for handling HTTP requests and responses, support for web sockets and SignalR for real-time communication, and support for multiple authentication and authorization schemes.

### **5. How do you handle errors in .NET Core?**

.NET Core provides a built-in middleware for handling errors called the Exception Handling Middleware. Developers can also write custom middleware to handle specific types of errors.

### **6. What are the key features of .Net Framework?**

Some key features of the .NET Framework include:

* **Common Language Runtime (CLR)**, which manages the execution of code and provides features such as memory management and security.
* **Class Library**, a collection of pre-built code that developers can use to build applications.
* **Language-Integrated Query (LINQ)**, a feature that allows developers to query data from various data sources using a common syntax.
* **Windows Presentation Foundation (WPF)**, a framework for building graphical user interfaces for desktop applications.
* **ASP.NET**, a framework for building web applications and services.

### **7. What is the difference between .NET Framework and .NET Core?**

Below is the major difference between .Net Framework and .Net Core:

* .NET Framework is a Windows-only framework, while .NET Core is cross-platform and can run on Windows, Linux, and macOS.
* .NET Framework applications require the framework to be installed on the target machine, whereas .NET Core applications can be deployed as self-contained executables that include all of the necessary dependencies.

### **8. What is a namespace in .NET?**

In .NET, a namespace is a way to organize and group related classes, structures, interfaces, enumerations, and other types within a common naming context. It provides a way to avoid naming conflicts between types and helps in better organization of the code.

### **9. What is an assembly in .NET?**

In .NET, an assembly is a unit of deployment that contains one or more files, such as executable files, DLLs, or resource files, that form a logical unit of functionality. It is the smallest unit of deployment and versioning in .NET, and can be shared among multiple applications.

### **10. What is the difference between value types and reference types in .NET?**

Below is the difference between value types and reference types in .Net:

| **Value Types** | **Reference Types** |
| --- | --- |
| Value types are types that store their data directly in memory rather than storing a reference to an object. | Reference types, on the other hand, are types that store a reference to an object in memory rather than storing the object's data directly. |
| Value types are managed by the compiler. | Reference types are managed by the garbage collector, which periodically frees up memory that is no longer being used |
| They are cleaned up automatically when they go out of scope. | They are cleaned up by the Garbage collector. |
| Store data directly in memory. | Store a reference to an object in memory |
| Small in size | Large in size |

### **11. Explain about major components of the .NET framework.**

The major components of the .NET framework include the **Common Language Runtime (CLR), .NET Languages, Base Class Library (BCL),**and the **Just-In-Time Compiler.**

* **Common Language Runtime (CLR)** - It provides the environment for the user to run .NET applications.
* **Base Class Library (BCL)** - This is a collection of classes, methods, etc., that provide the functionality to the applications.
* **.NET Languages** - It is the collection of languages in the .NET Framework. It includes languages like C#, VB.NET, etc.
* **Just-In-Time Compiler** - This compiler converts the user code into the Machine code during runtime.

### **12.  What is the role of the Startup class in .NET Core?**

The Startup class is a fundamental component of the .NET Core web application architecture. It is responsible for configuring the application and its dependencies.

When the application starts, the Startup class is executed, and it performs the following tasks:

* **Sets up the environment**: The Startup class sets up the hosting environment for the application, such as development, staging, or production. It also loads the configuration settings for the application. Use this method to configure the HTTP request pipeline.
* **Configures services**: The Startup class is responsible for configuring the services that the application needs to function correctly.

You can create your own Startup.cs.

### **13. What is middleware in .NET Core?**

In .NET Core, middleware is a software component that sits between a web application's client-side and server-side components. It is responsible for processing incoming HTTP requests and generating appropriate HTTP responses. Middleware is a key feature of the ASP.NET Core framework, which is used to develop web applications.

**Middleware Configuration**: In the **Configure** method, you set up the request processing pipeline by adding middleware components. Middleware components handle tasks such as routing, authentication, authorization, and serving static files.

### **14. What is the difference between a managed and an unmanaged code?**

Managed code is code that is executed within a managed execution environment, such as the .NET Common Language Runtime (CLR).Whereas Unmanaged code, on the other hand, is code that is executed directly by the operating system without the use of a managed execution environment.

### **15. Explain the architecture of .NET Core.**

.NET Core is an open-source, cross-platform development framework that can be used to create applications for Windows, Linux, and macOS operating systems.

The architecture of .NET Core can be broken down into several key components:

* **CoreCLR:** The CoreCLR is the runtime environment for .NET Core. It is responsible for loading, compiling, and executing managed code.
* **Class Library:** The Class Library is a collection of reusable code that provides a set of APIs for common programming tasks, such as working with strings, files, and networking.
* **Host:** The Host is responsible for initializing and running the application. It includes a command-line interface (CLI) that allows developers to build, package, and deploy applications.

### **16. What are the different types of hosting models supported in .NET Core?**

.NET Core supports several different hosting models, which allow developers to deploy and run their applications in a variety of environments.

The following are the different types of hosting models supported in .NET Core:

1. **In-process hosting**: In-process hosting allows an application to be hosted within the same process as the hosting application. This is the default hosting model used by ASP.NET Core applications.
2. **Out-of-process hosting**: Out-of-process hosting allows an application to be hosted in a separate process from the hosting application.

### **17. How do you implement middleware in .NET Core?**

Steps to implement middleware in .NET Core are:

1. **Create a middleware class:** Create a new class that implements the IMiddleware interface or the Microsoft.AspNetCore.Http.IMiddleware interface.
2. **Implement the middleware logic:** In the middleware class, implement the logic for handling requests and responses. This could involve adding headers, modifying the response, or performing some other task.
3. **Register the middleware:** In the Configure method of the Startup class, use the UseMiddleware extension method to register the middleware.

### **18. What is the role of the Program.cs file in .NET Core?**

The Program.cs file is the entry point of a .NET Core application. It contains the Main method, which is the starting point of the application. The Main method is responsible for configuring the application and starting the application's host.

The host is responsible for managing the application's lifetime, including starting and stopping the application. In addition to configuring the host, the Main method can also configure services that are used by the application. These services can include middleware, logging, and other components that the application needs to run.

### **19. What is the Common Language Runtime (CLR) in .NET?**

The Common Language Runtime (CLR) is a component of the .NET Framework that provides a runtime environment for managed code. The CLR is responsible for managing the execution of .NET programs, including memory management, security, and exception handling.

It is the component that actually executes compiled .NET code.

### **20.  Explain the concept of Razor Pages in .NET Core.**

Razor Pages is a way of building web applications in .NET Core that makes it easier for developers to create web pages.

With Razor Pages, developers can create a page that combines the HTML markup and the C# code needed to generate the page in a single file.

This approach simplifies the programming model by removing the need for a separate controller and view. Instead, each Razor Page has its own page model that handles the business logic for that page.

Razor Pages are built on top of the existing Razor view engine that is used for rendering HTML, which makes it easier for developers who are already familiar with Razor syntax to work with them.

### **21. How does .NET Core support cross-platform development?**

.NET Core is designed from the ground up to support cross-platform development, which means that developers can write code that can run on multiple platforms, including Windows, macOS, and Linux.

This is achieved through several key features of the .NET Core platform:

* **Cross-platform runtime**: .NET Core includes a runtime that can be installed on multiple platforms, allowing .NET Core applications to run on any platform that supports the runtime.
* **Cross-platform tooling**: .NET Core includes a set of command-line tools that can be used on multiple platforms, including the .NET Core CLI and Visual Studio Code.

### **23.  What are delegates in .NET?**

Delegates in .NET are type-safe function pointers that allow you to reference methods or functions in a way that is similar to C/C++ function pointers. Delegates can be used to pass methods as parameters to other methods or to store references to methods in a data structure.

A delegate is a reference type that encapsulates a method with a specific signature, consisting of a return type and zero or more input parameters. When a delegate is invoked, the method it references is executed with the specified input parameters, and the result is returned.

### **24. What is the difference between Debug and Release mode in .NET?**

Below is the difference between Debug Mode and Release Mode in .Net:

| **Debug Mode** | **Release Mode** |
| --- | --- |
| Debug mode is used during development, and testing stages | Release mode is used when an application is ready for distribution. |
| Debug mode is used to identify and fix bugs. | Release mode is used to optimize the performance of an application and prepare it for deployment. |
| When an application is built in Debug mode, the compiler includes additional debugging information, such as symbols and metadata, that can be used to help identify and diagnose problems in the code. | When an application is built in Release mode, the compiler applies optimization techniques to generate code that runs faster and more efficiently. |

### **25. How is a Managed code executed?**

In .NET Core, managed code is executed using the .NET Core Runtime, which is a cross-platform runtime that includes the Common Language Runtime (CLR) and the Base Class Library (BCL).

When you compile a .NET Core application, the source code is compiled into an intermediate language called Common Intermediate Language (CIL), which is also known as Microsoft Intermediate Language (MSIL). This is a high-level, CPU-independent representation of the code that can be executed on any platform that has the .NET Core runtime installed.

### **26. What is the difference between synchronous and asynchronous programming in .NET Core? When should you use each approach?**

In .NET Core, synchronous programming is the traditional programming approach, where the program execution is blocked until the operation completes, and the calling thread is suspended until the result is returned.

In contrast, asynchronous programming allows the program to continue executing while the operation is in progress, freeing up the calling thread to perform other tasks.

Synchronous programming is suitable when the program does not have a large number of concurrent tasks, and the tasks are relatively short-lived, and the program does not need to wait for the results of other tasks. It is easier to understand and debug compared to asynchronous programming.

### **27. What is Dependency Injection, and how is it used in .NET Core?**

Dependency Injection (DI) is a design pattern and technique used to achieve Inversion of Control (IoC) in software applications. The main idea behind Dependency Injection is to decouple components in a system by providing the required dependencies from the outside rather than creating them within the component itself. In short, it's a way to provide objects (dependencies) that a class needs to perform its work, rather than the class creating those objects itself.

In .NET Core (and ASP.NET Core), Dependency Injection is an integral part of the framework, and it's used to manage the dependencies of your application. Here's a brief explanation of how DI is used in .NET Core:

Here's how Dependency Injection is used in .NET Core:

**Service Registration**: In .NET Core, you typically register your services and dependencies in the **Startup** class within the **ConfigureServices** method. This method is part of the application's service configuration. You can register services as transient, scoped, or singleton, depending on their lifecycle.

* + Transient: A new instance is created every time it's requested.
  + Scoped: A single instance is created per HTTP request (in the context of a web application).
  + Singleton: A single instance is created and shared across the application.

Here's an example of registering a service in the **Startup** class:

public void ConfigureServices(IServiceCollection services)

{

services.AddScoped<ISomeService, SomeService>();

}

**Service Consumption**: Once a service is registered, you can inject it into your application's components, such as controllers, services, or other classes that require those dependencies. The framework automatically provides the necessary dependencies based on the registrations.

Here's an example of injecting a service into an ASP.NET Core controller:

**IoC Container**: .NET Core's IoC container, such as the built-in **IServiceProvider**, **manages the registered services and their lifetimes**. It ensures that the dependencies are created and disposed of correctly, following the specified lifetime.

By using Dependency Injection, .NET Core promotes separation of concerns, testability, and maintainability in your applications. It also simplifies the management of dependencies, as the framework takes care of creating and providing them. This pattern is not limited to ASP.NET Core but can be used in various .NET Core applications, including console applications, Windows services, and more.

**What is ASP.NET Core?**

**Answer:** ASP.NET Core is an open-source, cross-platform web framework developed by Microsoft. It is a modern, high-performance framework for building web applications and services. It is a complete rewrite of the original ASP.NET framework and is designed to be modular and lightweight.

**What are the key features of ASP.NET Core?**

**Answer:** Key features of ASP.NET Core include:

* Cross-platform support (Windows, macOS, Linux)
* High performance
* Modular and customizable architecture
* Integrated dependency injection
* Built-in support for containerization (Docker)
* Support for cloud-based hosting
* Improved testability and test-driven development
* Improved support for modern web development practices, such as REST APIs and client-side frameworks.

**What is the difference between ASP.NET Core and ASP.NET Framework?**

**Answer:** ASP.NET Core is a complete rewrite of ASP.NET Framework and has several key differences, including cross-platform support, improved performance, a modular and lightweight architecture, and built-in support for dependency injection. ASP.NET Framework is Windows-only, monolithic, and lacks many of the modern development features found in ASP.NET Core.

**What is Middleware in ASP.NET Core?**

**Answer:** Middleware in ASP.NET Core is software components that are used to process HTTP requests and responses in a pipeline. Each middleware component in the pipeline can perform specific tasks, such as authentication, routing, logging, or compression. Middleware is executed in a specific order, and it allows developers to customize and extend the request/response processing flow.

**Explain Dependency Injection in ASP.NET Core.**

**Answer:** Dependency Injection (DI) is a design pattern in ASP.NET Core where components, such as services and repositories, are injected into the application's components instead of creating them within the component. It promotes modularity, testability, and maintainability. ASP.NET Core provides a built-in DI container that can be used to manage the dependencies in your application.

**What is the Startup.cs file in ASP.NET Core?**

**Answer:** The Startup.cs file is a central part of an ASP.NET Core application. It configures various services, middleware, and the request processing pipeline. It defines the **Configure** method for configuring the HTTP request pipeline and the **ConfigureServices** method for configuring services, including dependency injection.

**What is the difference between Razor Pages and MVC in ASP.NET Core?**

**Answer:** Razor Pages and MVC are two approaches to building web applications in ASP.NET Core. MVC (Model-View-Controller) provides more control and separation of concerns, while Razor Pages offer a simpler and more convention-based approach, with the view and controller logic combined in a single Razor Page. The choice between them depends on the complexity of your application and your preference for structure and separation of concerns.

**What is Entity Framework Core, and how does it relate to ASP.NET Core?**

**Answer:** Entity Framework Core (EF Core) is the ORM (Object-Relational Mapping) framework for ASP.NET Core. It allows developers to work with databases using .NET objects. EF Core is used to interact with databases, perform CRUD operations, and work with data models in ASP.NET Core applications.

**What is the difference between authentication and authorization in ASP.NET Core?**

**Answer:** Authentication is the process of verifying the identity of a user, while authorization is the process of determining what actions or resources a user is allowed to access. Authentication is about verifying credentials, such as a username and password, while authorization is about defining access control rules for authenticated users.

**How do you handle configuration in ASP.NET Core?**

**Answer:** ASP.NET Core provides a built-in configuration system that can read settings from various sources, such as JSON files, environment variables, and command-line arguments. You can use the **IConfiguration** interface to access configuration settings in your application.

**Explain the concept of Tag Helpers in ASP.NET Core.**

**Answer:** Tag Helpers are a feature in ASP.NET Core that allow developers to create custom HTML-like tags in Razor views. These tags provide a way to generate HTML elements with server-side logic, making it easier to work with HTML in a more intuitive and structured manner.

**What are the different types of caching available in ASP.NET Core, and how do they work?**

**Answer:** ASP.NET Core supports various caching mechanisms, including in-memory caching, distributed caching, and response caching. In-memory caching stores data in memory on the web server, distributed caching stores data in a shared cache that multiple servers can access, and response caching stores HTTP responses to reduce server load and improve performance.

**How does ASP.NET Core handle security and protect against common web application vulnerabilities like XSS and CSRF?**

**Answer:** ASP.NET Core includes built-in security features to help protect against vulnerabilities. It provides features like Cross-Site Scripting (XSS) protection through content encoding, Cross-Site Request Forgery (CSRF) protection with anti-forgery tokens, and built-in authentication and authorization mechanisms.

**Explain the concept of Entity Framework Core migrations.**

**Answer:** Entity Framework Core migrations are a way to manage database schema changes in your application. Migrations are used to create, update, and roll back database schemas as your application evolves. They allow you to version and apply changes to the database in a structured and consistent manner.

**What is the role of the appsettings.json file in ASP.NET Core and how can you access configuration values from it?**

**Answer:** The **appsettings.json** file is used to store application configuration settings. It is a JSON file that can hold various configuration options for your ASP.NET Core application. You can access configuration values from it using the **IConfiguration** interface, and ASP.NET Core provides built-in tools to read and manage these settings.

**Explain the concept of model binding in ASP.NET Core.**

**Answer:** Model binding is the process of mapping HTTP request data to the parameters of a controller action method. In ASP.NET Core, the framework automatically binds data from the request to action method parameters based on parameter names and types. This simplifies the process of handling incoming data, such as form submissions or query string values.

**What is the purpose of the HttpContext in ASP.NET Core, and how can it be used in an application?**

**Answer:** The **HttpContext** represents the current HTTP request and response in ASP.NET Core. It provides access to various aspects of the request, including the request and response streams, request headers, user information, and more. Developers can use it to interact with and manipulate the current request and response.

**How do you handle errors and exceptions in ASP.NET Core?**

**Answer:** ASP.NET Core provides various ways to handle errors and exceptions, including global error handling via middleware, using the **try...catch** block in code, and custom error pages. The framework also provides detailed error information in development mode and more user-friendly error pages in production.

**What is the purpose of the Startup.cs ConfigureServices method?**

**Answer:** The **ConfigureServices** method in **Startup.cs** is used to configure the services and dependencies that the application will use. It is where you register services, such as Entity Framework DbContexts, dependency injection services, and third-party libraries. This method is called during application startup and sets up the service container.

**What is Razor and how is it used in ASP.NET Core?**

**Answer:** Razor is a markup syntax used in ASP.NET Core views to combine C# code with HTML. It allows developers to embed C# code directly within HTML views, making it easier to generate dynamic content. Razor is the default view engine for ASP.NET Core, and it is known for its simplicity and readability.

**Explain the role of the app.UseAuthentication() and app.UseAuthorization() methods in the Startup.cs file.**

**Answer:** The **app.UseAuthentication()** method configures authentication middleware, enabling authentication services and allowing users to log in. The **app.UseAuthorization()** method configures authorization middleware, which enforces access control policies on resources and actions based on user roles and permissions.

**What is the purpose of the Model-View-Controller (MVC) pattern in ASP.NET Core, and how does it work?**

**Answer:** The Model-View-Controller (MVC) pattern is a design pattern used in ASP.NET Core to separate an application into three main components: Model (data and business logic), View (presentation and user interface), and Controller (handling user input and coordinating actions). MVC promotes the separation of concerns, making applications more maintainable and testable.

**How can you enable CORS (Cross-Origin Resource Sharing) in an ASP.NET Core application?**

**Answer:** To enable CORS in an ASP.NET Core application, you need to configure the **AddCors** and **UseCors** methods in the **Startup.cs** file. This allows you to specify which origins are permitted to access resources on your server and configure various CORS policies.

**What is SignalR in ASP.NET Core, and how is it used for real-time communication?**

**Answer:** SignalR is a library in ASP.NET Core that allows for real-time, bidirectional communication between the server and clients over websockets, server-sent events, or other transport methods. It's commonly used for building applications that require real-time features, such as chat applications or live updates.

**Explain the difference between Razor Pages and Razor Views in ASP.NET Core.**

**Answer:** Razor Pages are a higher-level concept in ASP.NET Core that combines the view and controller logic in a single page (**.cshtml** file). Razor Views, on the other hand, are typically used in the context of MVC and provide templates for rendering data without logic. Razor Pages simplify development for smaller applications, while Razor Views are used in more complex applications with separate controllers.

**What is the role of the .csproj file in an ASP.NET Core project, and how does it relate to project dependencies?**

**Answer:** The **.csproj** file is the project file for an ASP.NET Core project. It defines project structure, dependencies, compilation settings, and more. It lists the project's dependencies on external libraries and other projects, making it easier to manage and build the application.

**Explain how to perform unit testing in ASP.NET Core.**

**Answer:** To perform unit testing in ASP.NET Core, you can use testing frameworks like xUnit, MSTest, or NUnit. You write test methods that verify the behaviour of your application's components, such as controllers and services, and use dependency injection to isolate dependencies during testing.

**What are the benefits of using the ASP.NET Core Razor Pages framework?**

**Answer:** The benefits of using Razor Pages in ASP.NET Core include a simplified development model, where the view and controller logic are combined in a single Razor Page, making it easier for developers to work on small to medium-sized applications. Razor Pages promote a convention-based approach and are well-suited for scenarios where CRUD operations are the primary focus.

**Explain the concept of middleware chaining in ASP.NET Core.**

**Answer:** Middleware chaining is the process of adding multiple middleware components to the request processing pipeline in a specific order. Each middleware component processes the request or response and passes control to the next middleware in the chain. Middleware chaining allows you to create a sequence of processing steps in your application.

**What is query string?**

A query string is a part of a URL (Uniform Resource Locator) that contains data in the form of key-value pairs. It is typically located at the end of a URL, following a question mark (?) and is used to pass data from a client (usually a web browser) to a web server. Query strings are commonly used to customize and filter the content of a web page, making it more dynamic and interactive.

Here's an example of a URL with a query string:

[**https://example.com/products?category=electronics&page=2**](https://example.com/products?category=electronics&page=2)

In this example, the URL is "<https://example.com/products>," and the query string starts with the question mark (?). The query string contains two key-value pairs separated by an ampersand **&**:

* **category=electronics**: This pair specifies that the category parameter is set to "electronics."
* **page=2**: This pair specifies that the page parameter is set to "2."

**Explain Life cycle of Asp.Net Core.**

The lifecycle of an ASP.NET Core application can be broken down into a few key stages:

1. **Startup:**
   * The application starts when the server is launched or the application is deployed.
   * The **Program.cs** file is the entry point, and it configures the web host.
   * In the **Startup.cs** file, the **ConfigureServices** method sets up services (dependencies), while the **Configure** method configures the request pipeline.
2. **Request Handling:**
   * When a client (usually a web browser) sends an HTTP request to the server, the request enters the ASP.NET Core application.
   * The request travels through the middleware pipeline, where each middleware component can perform specific tasks, such as authentication, routing, logging, or handling the request.
3. **Routing:**
   * The routing middleware determines which controller and action method should handle the request based on the URL route.
   * It maps the URL to a specific controller and action method.
4. **Controller and Action Execution:**
   * The controller and action method responsible for handling the request are invoked.
   * The controller processes the request, interacts with the model and data, and generates a response.
5. **View Rendering:**
   * If the response includes a view, the view engine processes the view and generates HTML to be sent back to the client.
6. **Response Generation:**
   * The response, which may include HTML content, is generated.
   * The response passes through the middleware pipeline in the reverse order, potentially being modified by response-specific middleware.
7. **Sending the Response:**
   * The server sends the response back to the client (e.g., a web browser).
8. **Termination:**
   * The application's work for that request is completed, and the resources associated with the request are released.
9. **Application Shutdown:**
   * If the application is terminated, for example, when the server is stopped, any cleanup or finalization tasks are performed.

In simple terms, when an HTTP request is received, ASP.NET Core goes through a series of stages to process the request, determine the appropriate controller and action, handle the request, generate a response, and then send the response back to the client. The **Startup** class and middleware configuration in **Startup.cs** play a crucial role in defining how the request is handled and which components are involved in the process.

**1. Dependency Injection through Constructor Injection:**

public class OrderService

{

private readOnly IOrderRepository \_repository;

public OrderService(IOrderRepository repository)

{

\_repository = repository;

}

}

**2. DI through Property Injection:**

public class OrderService

{

public IOrderRepository Repository {get; set;};

}

**3. DI through Method Injection:**

public class OrderService

{

public void ProcessOrder(Order order, IOrderProcessor processor)

{

// Method-specific Injection

}

}

Create a basic endpoints using .net. use the following schema for same.

1) Create an API to filter by department.

2) Create a API to sort by salary.

3) Create a API to search by employee\_id.

{

employee\_id:'123'

first\_name:"xyz",

last\_name:"xyz",

department:"IT",

Address:"xyz",

hire\_date:"01-02-2023",

dob:"01-02-2012",

joiningDate:"20-02-2023",

salary:'123'

}

[Route("api/employees")]

[ApiController]

public class EmployeeController : ControllerBase

{

private readonly List<Employee> \_employees; // Assuming you have a list of employees

public EmployeeController()

{

// Initialize your list of employees (you may load this from a database)

\_employees = new List<Employee>

{

new Employee { /\* employee details \*/ },

// Add more employees

};

}

[HttpGet("filterByDepartment/{department}")]

public IActionResult FilterByDepartment(string department)

{

var filteredEmployees = \_employees.Where(e => e.Department.Equals(department, StringComparison.OrdinalIgnoreCase)).ToList();

return Ok(filteredEmployees);

}

// Other actions for sorting and searching

[HttpGet("sortBySalary")]

public IActionResult SortBySalary()

{

var sortedEmployees = \_employees.OrderBy(e => e.Salary).ToList();

return Ok(sortedEmployees);

}

[HttpGet("searchById/{employeeId}")]

public IActionResult SearchById(string employeeId)

{

var employee = \_employees.FirstOrDefault(e => e.EmployeeId.Equals(employeeId, StringComparison.OrdinalIgnoreCase));

if (employee != null)

{

return Ok(employee);

}

else

{

return NotFound($"Employee with ID {employeeId} not found");

}

}

}

**What are Design Patterns**   
Design patterns are reusable solutions to the problems that we encounter in the day to day programming. They are generally targeted at solving the problems of object generation and integration. In other words, Design patterns acts as templates which can be applied to the real-world programming problems.  
  
**History and evolution of design Patterns**  
The four authors of the book famously know as Gang of four are the ones who brought the concepts of design patterns in their book “Elements of reusable Object-Oriented software”.   
   
Gang of Four has divided the book into two parts with first part explaining about the pros and cons of object oriented programming and the second part describes the evolution of 23 classic software design patterns.  
   
For more details, please refer to the following Wikipedia article  
<https://en.wikipedia.org/wiki/Design_Patterns>  
  
**Types of Design Patterns**  
Gang of Four have categorised the design patterns in to 3 types based on different problems encountered in the real world applications. They are Creational, Structural and Behavioural.  
   
**Creational design patterns :**These patterns deal with object creation and initialization. Creational pattern gives the program more flexibility in deciding which objects need to be created for a given case.  
   
**Examples of Creational design patterns category:**Singleton, Factory and Abstract Factory etc.  
  
**Structural design patterns:**This pattern deals with class and object composition. In simple words, this pattern focuses on decoupling interface, implementation of classes and its objects.   
  
**Examples of Structural design patterns category:**Adapter, Facade and Bridge etc.  
  
**Behavioural design patterns:**These patterns deal with communication between Classes and objects.  
  
**Examples of Behavioural design patterns:** Chain of Responsibility, Command and Interpreter etc.  
   
For understanding design patterns better it is very important to have basic knowledge about the following object oriented concepts 

* Abstraction
* Inheritance
* Polymorphism
* Encapsulation
* Interfaces
* Classes
* Abstract classes