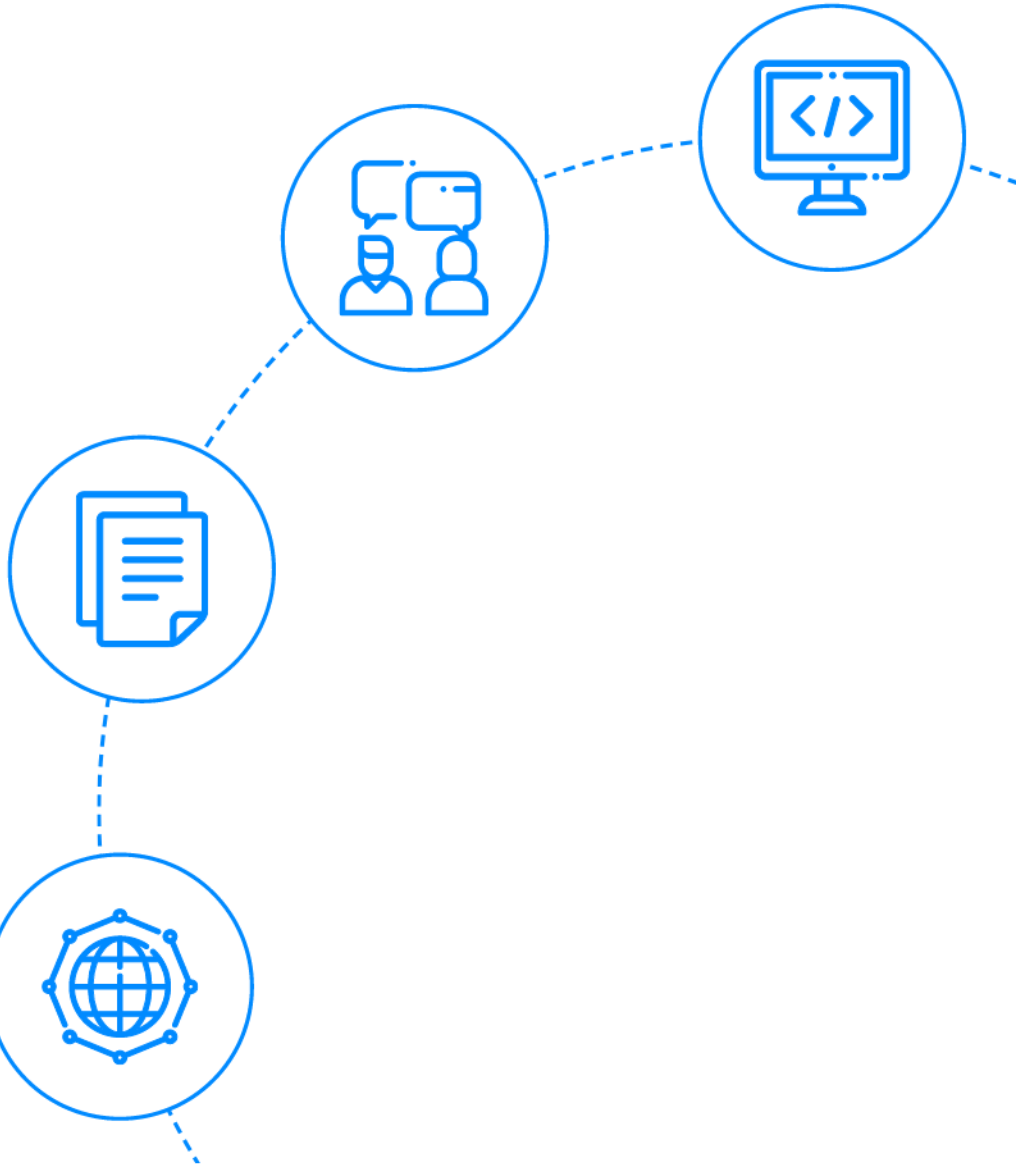
InterviewBit

ASP.NET Interview Questions



To view the live version of the page, click here.

Contents

Basic ASP.NET Interview Questions

Advanced ASP.NET Interview Questions

Advanced ASP.NET Interview Questions (.....Continued)

19. What is model binding in ASP.NET?
20. What is an Action Method?
21. What are the different types that implement the IActionResult interface?
22. What's the HTTPContext object? How can you access it within a Controller?
23. What is dependency injection?
24. Explain how dependency injection works in ASP.NET Core?
25. What is a cookie?
26. Explain the concept of middleware in ASP.NET Core?
27. What is routing, and how can you define routes in ASP.NET Core?
28. Explain how conventional routing works?
29. Explain how attribute-based routing works?
30. What is a RESTful Web Service or a Web API?
31. What is Entity Framework?

Conclusion

32. Conclusion

Let's get Started

ASP.NET is a web application framework developed by Microsoft which was released as part of the .NET framework. It makes it easy to build dynamic web applications. Some other examples of similar web application frameworks include Ruby on Rails (Ruby), Django (Python), and Express (JavaScript).

ASP.NET Core is a **cross-platform**, **high-performance**, and **open-source** web application framework. Microsoft released the first version of ASP.NET Core in 2016. It enables developers to create modern, cloud-enabled applications.

- **Cross-Platform:** The main advantage of ASP.NET Core is that it's not tied to a Windows operating system, like the legacy ASP.NET framework. You can develop and run production-ready ASP.NET Core apps on Linux or a Mac.
- **High Performance:** It's designed from scratch, keeping performance in mind. It's now one of the fastest web application frameworks.
- **Open Source:** Finally, it's open-source and actively contributed by thousands of developers all over the world.

Both the ASP.NET and ASP.NET Core run on C#, an object-oriented, general-purpose programming language. ASP.NET Core inherits many concepts and features from its ASP.NET heritage, but it's fundamentally a new framework.

Though Microsoft is going to support the legacy ASP.NET framework, it's not going to develop it actively. The new ASP.NET Core framework will include all the new features and enhancements. Going forward, Microsoft is recommending developers to build all the new web applications with ASP.NET Core instead of the legacy ASP.NET framework.

In this article, we will focus on both ASP.NET and ASP.NET Core interview questions. To limit the article's scope, we assume that you have programmed in the C# programming language. A basic understanding of common object-oriented concepts and front-end technologies such as HTML, CSS, and JavaScript is also expected.

We have divided the interview questions into two sections. The basic interview questions cover the fundamentals and focus on understanding the application structure of a basic ASP.NET project. Then, we cover the more advanced concepts such as dependency injection, routing, and model binding in the advanced interview questions.

Basic ASP.NET Interview Questions

1. What is a web application?

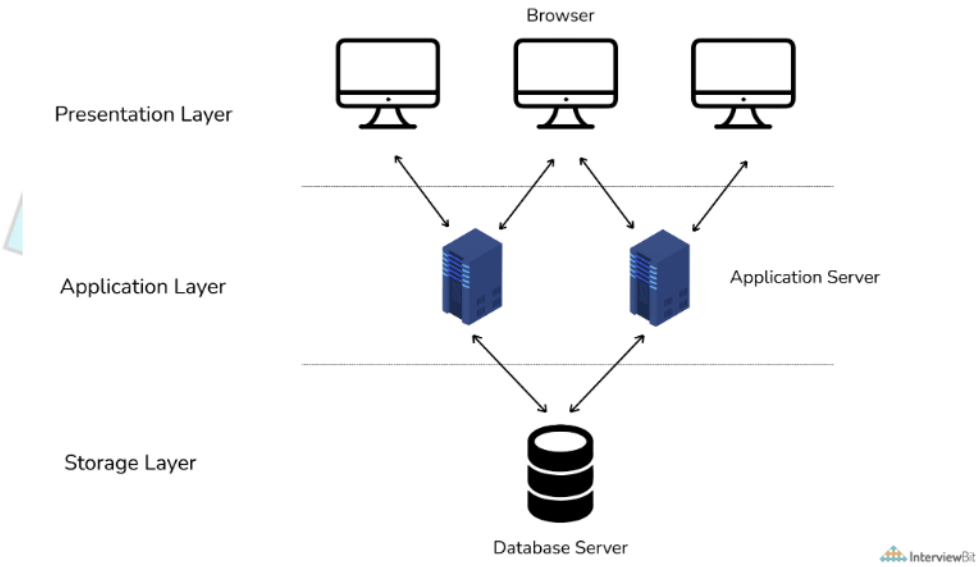
A Web application is a software that the users can access through a web browser such as Chrome or Firefox. The browser makes an HTTP request for a specific URL for the web application. The web application server intercepts and processes the request to build a dynamic HTML response sent to the user. Some examples of popular web applications include StackOverflow, Reddit, Google, etc.

A web application is different from a typical website. A website is static. When you go to the website, it returns an HTML page without doing any processing to build the contents of that HTML page. You will see the same page if you reload the browser. In contrast, a web application might return a different response each time you visit.

For example, let's say you ask a question on Stack Overflow. Initially, you will see only your question when you visit the URL. However, if another user answers your question, the browser will display that answer on your next visit to the same URL.

A web application consists of multiple separate layers. The typical example is a three-layered architecture made up of presentation, business, and data layers. For example, the browser (presentation) talks to the application server, which communicates to the database server to fetch the requested data.

The following figure illustrates a typical Web application architecture with standard components grouped by different areas of concern.



2. What is a web application framework, and what are its benefits?

Learning to build a modern web application can be daunting. Most of the web applications have a standard set of functionality such as:

- Build a dynamic response that corresponds to an HTTP request.
- Allow users to log into the application and manage their data.
- Store the data in the database.
- Handle database connections and transactions.
- Route URLs to appropriate methods.
- Supporting sessions, cookies, and user authorization.
- Format output (e.g. HTML, JSON, XML), and improve security.

Frameworks help developers to write, maintain and scale applications. They provide tools and libraries that simplify the above recurring tasks, eliminating a lot of unnecessary complexity.

3. What are some benefits of ASP.NET Core over the classic ASP.NET?

- **Cross-Platform:** The main advantage of ASP.NET Core is that it's not tied to a Windows operating system, like the legacy ASP.NET framework. You can develop and run production-ready ASP.NET Core apps on Linux or a Mac. Choosing an open-source operating system like Linux results in significant cost-savings as you don't have to pay for Windows licenses.
- **High Performance:** It's also designed from scratch, keeping performance in mind. It's now one of the fastest web application frameworks.
- **Open Source:** Finally, it's open-source and actively contributed by thousands of developers all over the world. All the source code is hosted on GitHub for anyone to see, change and contribute back. It has resulted in significant goodwill and trust for Microsoft, notwithstanding the patches and bug-fixes and improvements added to the framework by contributors worldwide.
- **New Technologies:** With ASP.NET Core, you can develop applications using new technologies such as Razor Pages and Blazor, in addition to the traditional Model-View-Controller approach.

4. When do you choose classic ASP.NET over ASP.NET Core?

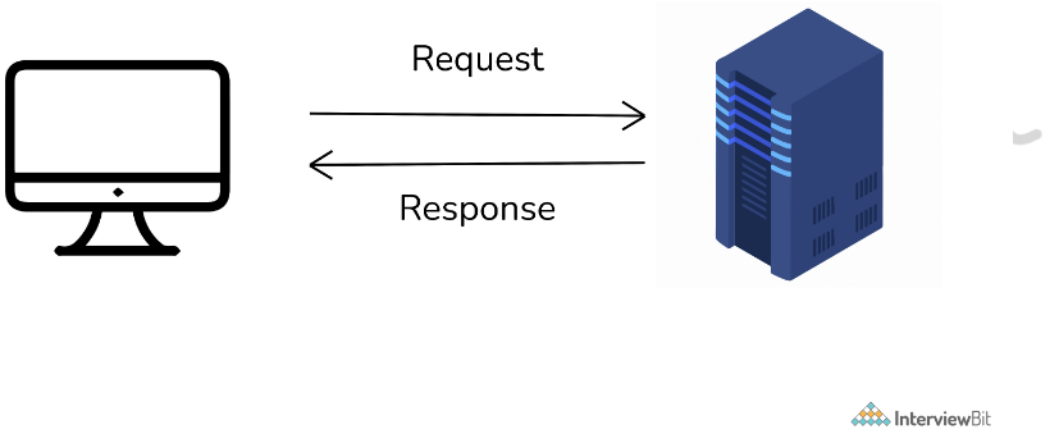
Though it's a better choice in almost all the aspects, you don't have to switch to ASP.NET Core if you are maintaining a legacy ASP.NET application that you are happy with and that is no longer actively developed.
ASP.NET MVC is a better choice if you:

- Don't need cross-platform support for your Web app.
- Need a stable environment to work in.
- Have nearer release schedules.
- Are already working on an existing app and extending its functionality.
- Already have an existing team with ASP.NET expertise.

5. Explain how HTTP protocol works?

Hypertext Transfer Protocol (HTTP) is an application-layer protocol for transmitting hypermedia documents, such as HTML. It handles communication between web browsers and web servers. HTTP follows a classical client-server model. A client, such as a web browser, opens a connection to make a request, then waits until it receives a response from the server.

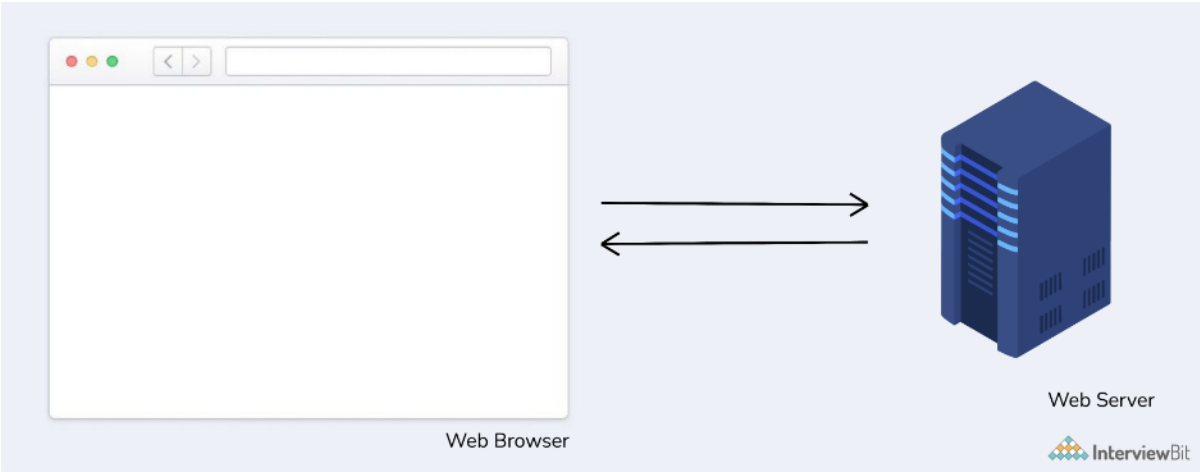
HTTP is a protocol that allows the fetching of resources, such as HTML documents. It is the foundation of any data exchange on the Web, and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser.



6. What is a web server?

The term web server can refer to both hardware or software, working separately or together.

On the hardware side, a web server is a computer with more processing power and memory that stores the application's back-end code and static assets such as images and JavaScript, CSS, HTML files. This computer is connected to the internet and allows data flow between connected devices.

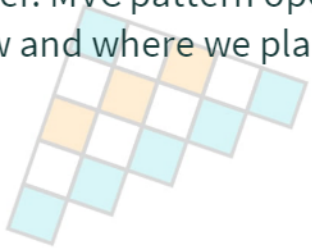


On the software side, a web server is a program that accepts HTTP requests from the clients, such as a web browser, processes the request, and returns a response. The response can be static, i.e. image/text, or dynamic, i.e. calculated total of the shopping cart.

Popular examples of web servers include Apache, Nginx, IIS.

7. What is the MVC pattern?

The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the **model**, the **view**, and the **controller**. It's different to note that this pattern is unrelated to the layered pattern we saw earlier. MVC pattern operates on the software side, while the layered pattern dictates how and where we place our database and application servers.



In an application that follows the MVC pattern, each component has its role well specified. For example, model classes only hold the data and the business logic. They don't deal with HTTP requests. Views only display information. The controllers handle and respond to user input and decide which model to pass to which view. This is known as the separation of responsibility. It makes an application easy to develop and maintain over time as it grows in complexity.

Though Model-View-Controller is one of the oldest and most prominent patterns, alternate patterns have emerged over the years. Some popular patterns include MVVM (Model-View-ViewModel), MVP (Model-View-Presenter), MVA (Model-View-Adapter).

8. Explain the role of the various components of the MVC pattern?

Model: Represents all the data and business logic that the user works within a web application. In ASP.NET, the model is represented by C# classes that hold the data and the related logic that operates on that data. The 'Models' directory stores the model classes.

For example, a model class representing a blog post might look like this:

```
// Models/Post.cs
namespace app.Models
{
  public class Post
  {
      public int ID { get; set; }

      public string Title { get; set; }

      public string Body { get; set; }
  }
}
```

View: Represents all the UI logic of the application. In a web application, it represents the HTML that's sent to the user and displayed in the browser.

One important thing to remember is that this HTML is not static or hard-coded. It's generated dynamically by the controller using a model's data. In ASP.NET, the 'Views' directory contains the views in files ending with the .cshtml file extension.

To continue our example of a blog post, a view to render a post might be:

```
// Views/Post.cshtml

<div class="post">
  <div class="title">
      <a href="/posts/@post.ID">@post.Title</a>
  </div>

  <div class=body>
      @Html.Raw(post.Body)
  </div>
</div>
```

Controller: Acts as an interface between Model and View. It processes the business logic and incoming requests, manipulates data using the Model, and interacts with the Views to render the final output.

In ASP.NET, these are C# classes that form the glue between a model and a view. They handle the HTTP request from the browser, then retrieve the model data and pass it to the view to dynamically render a response. The 'Controllers' directory stores the controller classes.

A PostController that builds the view for the post by fetching the Post model will be:

```
// Controllers/PostController
namespace app.Controllers
{
  public class PostsController : BaseController
  {
      public IActionResult Post(int id)
      {
          // Get the post from the database
          Post post = _service.Get(id);

          // Render the post.cshtml view, by providing the post model
          return View(post);
      }
  }
}
```

9. What is the purpose of the .csproj file?

The project file is one of the most important files in our application. It tells .NET how to build the project.

All .NET projects list their dependencies in the .csproj file. If you have worked with JavaScript before, think of it like a package.json file. The difference is, instead of a JSON, this is an XML file.

When you run dotnet restore, it uses the .csproj file to figure out which NuGet packages to download and copy to the project folder (check out the next question to learn more about Nuget).

The .csproj file also contains all the information that .NET tooling needs to build the project. It includes the type of project you are building (console, web, desktop, etc.), the platform this project targets, and any dependencies on other projects or 3rd party libraries.

Here is an example of a .csproj file that lists the Nuget packages and their specific versions.

```
<Project Sdk="Microsoft.NET.Sdk.Web">
<PropertyGroup>
  <TargetFramework>net5.0</TargetFramework>
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="AWSSDK.S3" Version="3.5.6.5" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="5.0.1" />
  <PackageReference Include="Microsoft.Extensions.Caching.Memory" Version="5.0.0" />
  <PackageReference Include="Npgsql" Version="5.0.1.1" />
  <PackageReference Include="Serilog" Version="2.10.0" />
</ItemGroup>
</Project>
```

In the above example,

- The SDK attribute specifies the type of .NET project.
- TargetFramework is the framework this application will run on, .NET 5 in this case.
- The PackageReference element includes the NuGet packages. The Version attribute specifies a version of the package we want.

10. What is NuGet package manager?

Software developers don't write all code from scratch. They rely on libraries of code written by other developers. Any modern development platform must provide a mechanism where developers can download and use existing libraries, often called packages. For example, the JavaScript ecosystem has NPM (Node Package Manager), where developers can find and use libraries written by other JavaScript developers.

NuGet is a package manager for the .NET ecosystem. Microsoft developed it to provide access to thousands of packages written by .NET developers. You can also use it to share the code you wrote with others.

A typical web application developed using ASP.NET relies on many open source NuGet packages to function. For example, Newtonsoft.Json is a very popular package (with 91,528,205 downloads at the time of writing) used to work with JSON data in .NET.

11. What is the purpose of the Program class?

Program.cs class is the entry point of our application. An ASP.NET application starts in the same way as a console application, from a **static void Main()** function.

This class configures the web host that will serve the requests. The host is responsible for application startup and lifetime management, including graceful shutdown.

At a minimum, the host configures a server and a request processing pipeline. The host can also set up logging, configuration, and dependency injection.

12. What is the purpose of the Startup class?

This class handles two important aspects of your application, namely service registration, and middleware pipeline.

Services are C# classes that your application depends on for providing additional functionality, both used by the framework and your application. Examples include logging, database, etc. These services must be registered to be instantiated when your app is running and when it needs them.

The middleware pipeline is the sequence in which your application processes an HTTP request (the next question explains the concept of Middleware in detail).

Startup class contains two methods: ***ConfigureServices()*** and ***Configure()***. As the name suggests, the first method registers all the services that the application needs. The second method configures the middleware pipeline.

13. What is the purpose of the wwwroot folder?

The wwwroot folder contains static files and compiled assets, such as JavaScript, CSS, and images that your web application needs. Wwwroot is the only folder in the entire project that's exposed as-is to the browser.

14. What is the purpose of the appsettings.json file?

Appsettings.json contains all of the application's settings, which allow you to configure your application behavior.

Here is an example of an appsettings.json file.

```
{
 "Logging": {
     "LogLevel": {
          "Default": "Information",
    "Microsoft": "Warning",
    "Microsoft.Hosting.Lifetime": "Information"
      }
  },
"ConnectionStrings": {
    "AppConnection": ""
},
"AWS": {
  "Profile": "local-test-profile",
  "Region": "us-west-2"
},
"AllowedHosts": "*"
}
```

15. What is IIS?

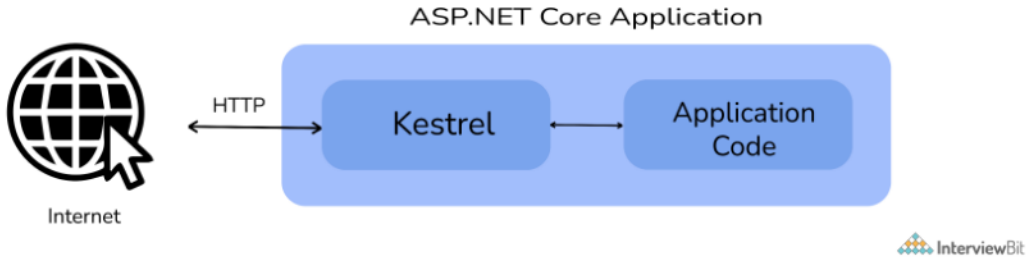
IIS stands for Internet Information Services. It is a powerful web server developed by Microsoft. IIS can also act as a load balancer to distribute incoming HTTP requests to different application servers to allow high reliability and scalability.

It can also act as a reverse proxy, i.e. accept a client's request, forward it to an application server, and return the client's response. A reverse proxy improves the security, reliability, and performance of your application.

A limitation of IIS is that it only runs on Windows. However, it is very configurable. You can configure it to suit your application's specific needs.

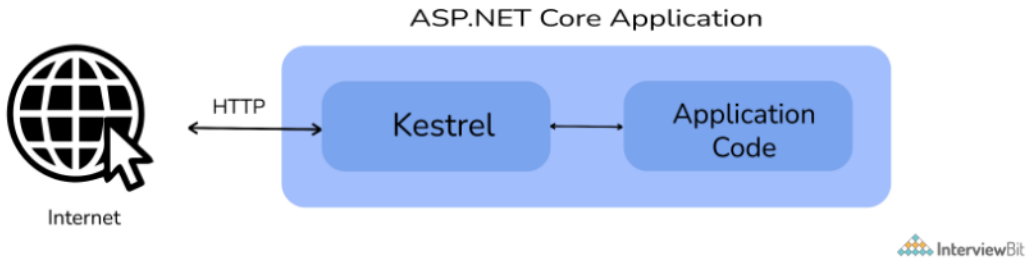
16. What is Kestrel?

Kestrel is an open-source, cross-platform web server designed for ASP.NET Core. Kestrel is included and enabled by default in ASP.NET Core. It is very light-weight when compared with IIS.



Kestrel can be used as a web server processing requests directly from a network, including the Internet.

Though Kestrel can serve an ASP.NET Core application on its own, Microsoft recommends using it along with a reverse proxy such as IIS, Nginx, or Apache, for better performance, security, and reliability.



17. What is the difference between IIS and Kestrel? Why do we need two web servers?

The main difference between IIS and Kestrel is that Kestrel is a cross-platform server. It runs on Windows, Linux, and Mac, whereas IIS only runs on Windows.

Another essential difference between the two is that Kestrel is fully open-source, whereas IIS is closed-source and developed and maintained only by Microsoft.

IIS is very old software and comes with a considerable legacy and bloat. With Kestrel, Microsoft started with high-performance in mind. They developed it from scratch, which allowed them to ignore the legacy/compatibility issues and focus on speed and efficiency.

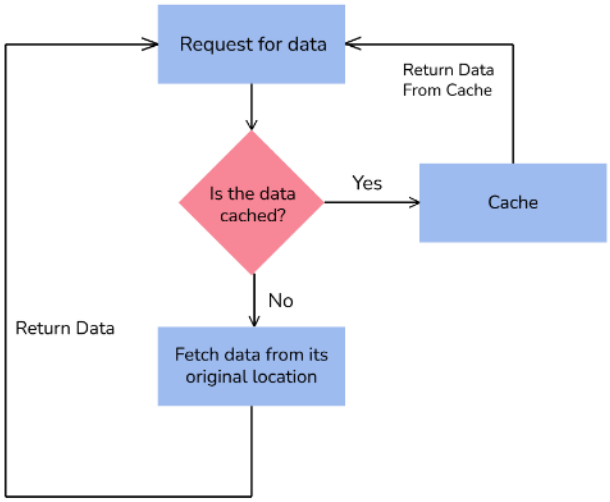
However, Kestrel doesn't provide all the rich functionality of a full-fledged web server such as IIS, Nginx, or Apache. Hence, we typically use it as an application server, with one of the above servers acting as a reverse proxy.

18. What is caching?

Caching is the process of storing data in a temporary storage location that is quicker to access than the original location of the data so that it can be accessed more quickly when the same data is needed next time.

Caching improves the scalability and performance of your application. It does this by reducing the work required to fetch the data. Caching is useful for data that doesn't change frequently and is expensive to create and retrieve.

ASP.NET provides a set of caching features out of the box. You can use the IMemoryCache interface for simple use cases. It represents a cache stored in the web server's memory. ASP.NET also supports distributed caching, which is a cache shared by multiple app servers, with Redis.



Advanced ASP.NET Interview Questions

19. What is model binding in ASP.NET?

Controllers and views need to work with data that comes from HTTP requests. For example, routes may provide a key that identifies a record, and posted form fields may provide model properties. The process of converting these string values to .NET objects could be complicated and something that you have to do with each request. Model binding automates and simplifies this process.

The model binding system fetches the data from multiple sources such as form fields, route data, and query strings. It also provides the data to controllers and views in method parameters and properties, converting plain string data to .NET objects and types in the process.

Example:
Let's say you have the following action method on the PostsController class:

```
[HttpGet("posts/{id}")]
public ActionResult<Post> GetById(int id, bool archivedOnly)
```

And the app receives a request with this URL:

```
http://yourapp.com/api/Posts/5?ArchivedOnly=true
```

After the routing selects the action method, model binding executes the following steps.

- Locate the first parameter of GetByID, an integer named id, look through the available sources in the HTTP request and find id = "5" in route data.
- Convert the string "5" into an integer 5.
- Find the next parameter of GetByID, a boolean named archivedOnly.
- Look through the sources and find "ArchivedOnly=true" in the query string. It ignores the case when matching the parameters to the strings.
- Convert the string "true" into boolean true.

Some other examples of attributes include:
1. [FromQuery] - Gets values from the query string.
2. [FromRoute] - Gets values from route data.
3. [FromForm] - Gets values from posted form fields.
4. [FromBody] - Gets values from the request body.
5. [FromHeader] - Gets values from HTTP headers.

20. What is an Action Method?

An action method is a method in a controller class with the following restrictions:

1. It must be public. Private or protected methods are not allowed.
2. It cannot be overloaded.
3. It cannot be a static method.

An action method executes an action in response to an HTTP request.

For example, here is an example of an Index() action method on the PostController. It takes an id as an input and returns an IActionResult, which can be implemented by any result classes (see the following question).

```
public class PostController : Controller
{
  public IActionResult Index(int id)
  {

  }
}
```

21. What are the different types that implement the IActionResult interface?

ASP.NET Core has many different types of IActionResult:

- ViewResult—Generates an HTML view.
- RedirectResult—Sends a 302 HTTP redirect response to send a user to a specified URL automatically.
- RedirectToRouteResult—Sends a 302 HTTP redirect response to automatically send a user to another page, where the URL is defined using routing.
- FileResult—Returns a file as the response.
- ContentResult—Returns a provided string as the response.
- StatusCodeResult—Sends a raw HTTP status code as the response, optionally with associated response body content.
- NotFoundResult—Sends a raw 404 HTTP status code as the response.

22. What's the HTTPContext object? How can you access it within a Controller?

HTTPContext encapsulates all HTTP-specific information about an individual HTTP request. You can access this object in controllers by using the **ControllerBase.HttpContext** property:

```
public class HomeController : Controller
{
  public IActionResult About()
  {
      var pathBase = HttpContext.Request.PathBase;

      ...

      return View();
  }
}
```

23. What is dependency injection?

Dependency injection is a design pattern that helps to develop loosely coupled code. This pattern is used extensively in ASP.NET.

Dependency injection means **providing** the objects that an object needs (its dependencies) in that object's constructor instead of requiring the object to construct them.

Dependency injection reduces and often eliminates unnecessary dependencies between objects that don't need to know each other. It also helps in testing by mocking or stubbing out the dependencies at runtime.

24. Explain how dependency injection works in ASP.NET Core?

ASP.NET Core injects instances of dependency classes by using the built-in IoC (Inversion-of-Control) container. This container is represented by the IServiceProvider interface that supports constructor injection.

The types (classes) managed by the container are called services. To let the IoC container automatically inject our services, we first need to register them with the IoC container in the Startup class.

ASP.NET Core supports two types of services, namely framework and application services. **Framework** services are a part of ASP.NET Core framework such as ILoggerFactory, IApplicationBuilder, IHostingEnvironment, etc. In contrast, a developer creates the **application services** (custom types or classes) specifically for the application.

25. What is a cookie?

A cookie is a small amount of data that is persisted across requests and even sessions. Cookies store information about the user. The browser stores the cookies on the user's computer. Most browsers store the cookies as key-value pairs.

Write a cookie in ASP.NET Core:

```
Response.Cookies.Append(key, value);
```

Delete a cookie in ASP.NET Core

```
Response.Cookies.Delete(somekey);
```

26. Explain the concept of middleware in ASP.NET Core?

In general, middleware is plumbing software that facilitates communication flow between two components. In a web application framework, middleware provides common services and capabilities to the application outside of the default framework.

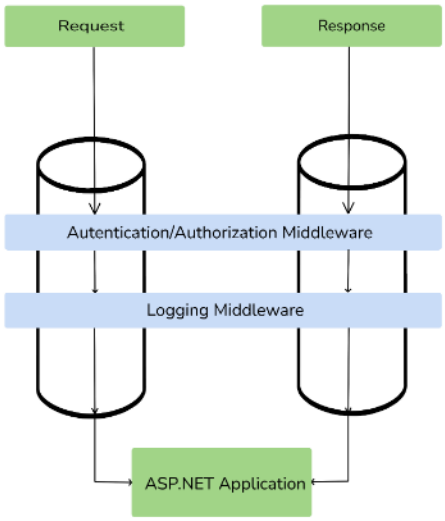
In ASP.NET Core, middleware refers to the C# classes that manipulate an HTTP request when it comes in or an HTTP response when it's on its way out. For example,

Generate an HTTP response for an incoming HTTP request
Intercept and make changes to an incoming HTTP request and pass it on to the next piece of middleware.
Intercept and make changes to an outgoing HTTP response, and pass it on to the next piece of middleware.

One of the most common use cases for middleware is to deal with concerns that affect your entire application. These aspects of your application need to occur with every request, regardless of the specific path in the request or the resource requested. These include things like logging, security, authentication, authorization, etc.

For example, logging middleware logs when a request comes in and passes it to another piece of middleware. Some other common middleware uses include database middleware, error handling middleware, and authentication/authorization middleware.

In each of these examples, the middleware receives a request, modifies it, and then passes it to the next middleware piece in the pipeline. Subsequent middleware uses the details added by the earlier middleware to handle the request in some way. The following diagram illustrates this.



27. What is routing, and how can you define routes in ASP.NET Core?

Routing is the process of mapping an incoming HTTP request to a specific method in the application code. A router maps the incoming requests to the route handler. It takes in a URL as an input and deconstructs it to determine the controller and action method to route the request.

A simple routing pattern, for example, might determine that the **/posts/show** URL maps to the **Show** action method on the **PostsController**.

There are two ways to define routes in an ASP.NET Core MVC application.

- Conventional Routing
- Attribute-Based Routing.

We can use both Conventional Routing and Attribute Routing in an application.

28. Explain how conventional routing works?

As the name suggests, conventional routing uses predefined conventions to match the incoming HTTP request to a controller's action method. It handles the most general cases that a typical web application needs, but you can modify it to suit your specific needs.

For example, the Configure() method in the Startup.cs class contains the following code that sets up the conventional routing.

```
app.UseEndpoints(endpoints =>
{
  endpoints.MapControllerRoute(
      name: "default",
      pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

This code creates a single route named 'default'. The route template pattern '{controller=Home}/{action=Index}/{id?}' is used to match an incoming URL such as **/Posts/Archived/5** to the **Archived(int id)** action method on the PostsController, passing 5 for the id parameter. By default, the router uses the Index method on the HomeController.

29. Explain how attribute-based routing works?

Attribute routing is an alternative routing strategy for conventional routing. It is often used to create REST API endpoints for web services. It uses a set of attributes to map action methods directly to route templates.

Attribute routing directly defines the routes on action methods. We can also use these attributes on the controllers. It enables us to get fine-grained control over what routes map to which actions. With attribute routing, the controller and action names play no part in determining the action method.

For example, we use attributes Blog and Home to map an incoming URL such as **myapp.com/blog/post/3** to the **Show** method on the **PostsController**.

```
[Route("blog")]
public class PostsController : Controller
{
  [HttpGet("post/{id:int}")]
  public IActionResult Show(int id = 0)
  {
      Post post = new Post()
      {
          ID = id
      };

      return View("Show", post);
  }

  [HttpGet("edit/{id:int}")]
  public IActionResult Edit(int id)
  {
      Post postToEdit = _service.Get(id);

      return View("Edit", postToEdit);
  }
}
```

In the above example, the attribute **[Route("blog")]** is placed on the controller, whereas the route **[HttpGet("post/{id:int}")]** is placed on the action method. A controller route applies to all actions in that controller. For example, the second **["edit/{id:int}"]** route matches the url **myapp.com/blog/edit/3**.

In addition to the above route templates, ASP.NET Core provides the following HTTP verb templates.

- [HttpGet]
- [HttpPost]
- [HttpPut]
- [HttpDelete]
- [HttpHead]
- [HttpPatch]

30. What is a RESTful Web Service or a Web API?

Not all web applications return an HTML view as a response to an HTTP request. Sometimes, a client only wants some data from your application, and it wants to handle how that data will be formatted.

For example, let's say your application supports both web and mobile interfaces. Instead of writing two separate projects which return HTML and mobile views, you can write a single application that only returns the specific data that the clients need. Once the clients receive this data, they format it accordingly. The web client renders the HTML using view templates and JavaScript, and the mobile clients generate the appropriate mobile view for its specific platform.

An application might also need to communicate with another application to fetch the data that it needs. For example, when you go to Amazon.com, it communicates with hundreds of other services and applications to retrieve data and renders the final HTML page you see.

Such back-end applications, which provide data, are commonly known as RESTful web services. REST protocol uses verbs like **GET, POST, PUT, DELETE** to communicate between multiple applications. The client and server can be written in different languages and technologies and still work together without knowing about each other, as long as each side knows the format of the data that is getting sent.

ASP.NET Core supports creating RESTful services, also known as web APIs, using C#. A Web API consists of one or more controllers that derive from ControllerBase class.

```
[PostController]
[Route("[controller]")]
public class PostController : ControllerBase
```

An MVC controller derives from the **Controller** class. However, a Web API controller should derive from the ControllerBase class. The reason is that Controller derives from ControllerBase and provides additional support for views, which you don't need for web API requests.

That said, you can use a controller for both rendering views and data. That is, it can act as both an MVC controller and a Web API controller. In this case, you can derive the controller from the Controller class.

31. What is Entity Framework?

Most applications require storing and retrieving data. Usually, we store this data in a database. Working with databases can often be rather complicated. You have to manage database connections, convert data from your application to a format the database can understand, and handle many other subtle issues.

The .NET ecosystem has libraries you can use for this, such as ADO.NET. However, it can still be complicated to manually build SQL queries and convert the data from the database into C# classes back and forth.

EF, which stands for Entity Framework, is a library that provides an object-oriented way to access a database. It acts as an object-relational mapper, communicates with the database, and maps database responses to .NET classes and objects.

Entity Framework (EF) Core is a lightweight, open-source, and cross-platform version of the Entity Framework.

Here are the essential differences between the two:

Cross-platform:

- We can use EF Core in cross-platform apps that target .NET Core.
- EF 6.x targets .NET Framework, so you're limited to Windows.

Performance:

- EF Core is fast and lightweight. It significantly outperforms EF 6.x.

Features:

- EF Core has some features that EF 6.x doesn't have (batching statements, client-side key generation, in-memory database for testing)
- EF 6.x is much more feature-rich than EF Core. EF Core is missing some headline features at the time of writing, such as lazy-loading and full server-side Group By. However, it is under active development, so those features will no doubt appear soon.

Conclusion

32. Conclusion

In this article on ASP.NET interview questions, we learned about the legacy ASP.NET framework and its modern alternative, that is ASP.NET Core. The article explored a broad range of basic and advanced questions that an interviewer would ask in a job interview for a junior/intermediate developer role. We hope it helps for your next job interview!

Useful Resources and References:

1. Official Website: https://dotnet.microsoft.com/apps/aspnet
2. ASP.NET Documentation: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-5.0

Links to More Interview Questions

C Interview Questions	Php Interview Questions	C Sharp Interview Questions
Web Api Interview Questions	Hibernate Interview Questions	Node Js Interview Questions
Cpp Interview Questions	Oops Interview Questions	Devops Interview Questions
Machine Learning Interview Questions	Docker Interview Questions	Mysql Interview Questions
Css Interview Questions	Laravel Interview Questions	Asp Net Interview Questions
Django Interview Questions	Dot Net Interview Questions	Kubernetes Interview Questions
Operating System Interview Questions	React Native Interview Questions	Aws Interview Questions
Git Interview Questions	Java 8 Interview Questions	Mongodb Interview Questions
Dbms Interview Questions	Spring Boot Interview Questions	Power Bi Interview Questions
Pl Sql Interview Questions	Tableau Interview Questions	Linux Interview Questions
Ansible Interview Questions	Java Interview Questions	Jenkins Interview Questions