* **Components**: A component can control numerous views wherein each of the views is a particular part on the screen. All Angular applications have a minimum of one component called the **root** component. This component is bootstrapped in the **root** module, the main module. All the components include the logic of the application that is defined in a class, while the main role of the class is to interact with the view using an API of functions and properties.
* **Template**: The view of each component is linked with a template, and an Angular template is a type of HTML tag that allows Angular to get an idea of how it needs to **render the component**.
* **Decorators** are a design pattern or functions that define how Angular features work. Decorator is a function. Decorator works similarly to attributes in c#. They are used to make prior modifications to a class, service, or filter. Angular supports four types of decorators, they are: Class Decorators, Property Decorators, Method Decorators, Parameter Decorators
* **Metadata**: Classes have metadata attached to them with the help of decorators so that Angular will have an idea of processing the class.
* [**Modules**](https://intellipaat.com/blog/tutorial/angularjs-tutorial/angularjs-modules/) (**NgModule**)**:** Groups components and other (components, services, directive, pipes) pieces of the applications. Module can be loaded in memory independent from other modules. Contains Imports: external/internal modules; Declarations: Internal components; Bootstrap: start-up components (like selector) Module or NgModule is a block of code organized using the necessary capabilities set, having one specific workflow. All Angular applications have at least one module, the root module, and most of the applications have numerous modules.
* **Bootstrap:** Is specified in app.Module. Module defines bootstrap array. This array should contain at least app.Component (root), with its selector custom HTML tag specification. Having app.cpmponent in the bootstrap array makes it loading with the module loading.
* **DOM**: HTML as a node tree, displayed in the page. Index.html + component selector tag
* **Services**: Service is a vast category that ranges from functions and values to features that play a significant role in Angular applications.
* **Dependency injection**: It uses DI so that it can offer the necessary dependencies, mainly services, to the new components. The constructor parameters of a component inform Angular regarding the `+numerous services needed by the component, and DI provides a solution that gives the necessary dependencies to the new class instances.
* **`Routing**: Routing functionality is provided by imported RouterModule. Router manipulates template HTML using router directives. Using router for Root method, router can be configured with available route definitions. Angular router helps interpret the URL of a browser to get a client-generated experience and view. This router is bound to page links so that Angular can go to the application view as soon as the user clicks on it.
* **Selector:** Not sure about details of what it is doing
* [**Directives**](https://intellipaat.com/blog/tutorial/angularjs-tutorial/angularjs-directives/): A custom HTML element, directing outer HTML what to display. Directive **extends** page html Component selector+template work as a Directive, when placed inside index.html . Visibility of directive is determined by using Module.ts to find declared components.ts There are also logical directives \*ngIf and \*ngFor Angular templates are of dynamic nature, and directives help Angular understand how it can transform the DOM while manifesting the template.
* **Data binding**: Data binding is the process in which the various sections of a **template** interact with the **component**. The binding markup needs to be added to the HTML template so that Angular can understand how it can connect with the component and template.
* **Interpolation**: One way **binding** from component class to template {{ class property }} aka **template expression.**
* **Piped**: Pipe in **binding** expression is activated by ‘pipe’ character ‘|’. It transforms bound properties before they are displayed.
* **Input and Output property and Reference variables** inter-component communication
* **Lifecycle hooks** every component has a lifecycle. Angular creates and renders these components and also destroys them before removing them from the DOM. This is achieved with the help of lifecycle hooks. ngOnInit, ngAfterViewInit, ngOnChanges, ngOnDestroy
* **Styles (CSS)** can be added to component as an array
* **Interface**: can specify Type, of Feature set
* **Promises and Observables** While both the concepts deal with Asynchronous events in Angular, Promises handle one such event at a time while observables handle a sequence of events over some time. Promises - They emit a single value at a time. They execute immediately after creation and are not cancellable. They are Push errors to the child promises. Observables - They are only executed when subscribed to them using the subscribe() method. They emit multiple values over a period of time.
* **CLI** assits in building application, generate components and module code, build and serve application, run tests, prepare deployment

The MVVM architecture plays a significant role in eliminating tight coupling between the components. This architecture includes the following three parts:

* **Model:** The model represents the business logic and data of a particular application. In other words, it consists of an entity structure. The model has the business logic, including model classes, remote and local data sources, and the repository.
* **View**: View is the application’s visual layer that comprises the UI code. The view sends the action of the user to the ViewModel. However, it does not receive the response directly. The view must subscribe to the observables that are exposed to it by the ViewModel to receive a response.
* **ViewModel**: ViewModel is the application’s abstract layer that connects the View and the Model and acts as a bridge between the two. It does not know which View needs to be made use of since it does not have any direct access to the View. The two are connected using data binding, and the ViewModel records all the changes that are made to the View and makes the necessary changes to the Model.

CORS - Cross Origin Resource Sharing

## Q: What is Stream? *"A stream refers to values of data overtime"*

The reason for it being called a stream is that you should think of the data as continuous and not really having an end, unless you explicitly define an end.

## Q: What is Observable? An Observable is an object that over time and asynchronously emits multiple data values (data stream). Click [here](https://www.techgeeknext.com/angular/angular-rxjs-observables) to know more.

## Q: What is the difference between an observable and a promise? Promise:

A Promise emits a single event at the completion or failure of an async operation.

* promise emits a single value
* A promise is Not Lazy A Promise cannot be cancelled

#### Observable:

An observer is like a stream and allows you to pass at least zero or more events where the callback is needed for each event.

* Observable is favored over Promise, it can emits multiple values over a time.
* The "Observable" is cold. It's not called until we're registered to it.
* You may cancel an Observable with the unsubscribe() method
* Observable provides a lot of efficient operators like map, foreach, filter, reduce, retry, retryWhen etc.

## Q: What is the difference between Cold and Hot Observables? *"Cold observables start to run in up and subscription, so observable sequence only starts pushing values to observers when subscribe is called.*

*"hot observables produce values even before subscriptions made.*

Hot observables such as mousemove events, stock pickers or WebSocket connections, are already produced in values even before subscription is active

if you want to share some resources or data between many subscribers, you have to use hot observables.

## Q: What is Observers and Subscriptions? Observers and Subscriptions are related to the concept of Observable.

## Observers:

*Observer is a set of callbacks that know how to listen to the values of the Observable.*

* Observers are also referred to as listeners (or consumers)
* Observers may listen or subscribe to the data being observed.

#### Subscription:

*Subscription is an observable execution*

* Subscriptions are objects returned when an Observable is subscribed.
* Subscription is useful mainly to cancel the execution

## Q: What is Subject? Subjects are special types of Observers, so you can also subscribe to other Observables and listen to published data *Special thing about subject is they are multicasted. It means - "The values are multicasted to many Observers" while default RxJS Observable is unicast*

## Q: What are different types of Subject? There are two types of Subjects : BehaviorSubject and ReplaySubject.

## Q: What are different between of Subject, BehaviorSubject and ReplaySubject?

## Subject :

In case of Subject, Observers who are subscribed at a later date will not obtain the data values emitted prior to their subscription.

#### ReplaySubject :

In ReplaySubject, Observers who are subscribed at a later point will receive all the old data values issued prior to their subscription. As it operates by using a buffer that holds the values emitted and re-emits (replaying) a sequence of old values once new Observers are subscribed.

#### BehaviorSubject :

BehaviorSubject functions similar to ReplaySubject but only re-issues the last emitted value (current value). So you're interested in the last / current value of the observer, if BehaviorSubject is useful.

## Q: What is Reactive programming and how does it relate to Angular? you can just build the different streams and operations that take place on those flows by specifying the whole program. Angular uses RxJS for some aspects of its internal service, such as Http, Router, etc. RxJS is a very powerful library that facilitates the design of applications.

## Q: What is RxJS Map and What is Higher-Order Observable Mapping? RxJS map operator lets us project the payload of the Observable into something else.The power of Observables is revealed when you start using Rx operators to transform, combine, manipulate, and work with sequences of items emitted by Observables.

To understand RxJS Map and Higher-Order Observable Mapping click [here](https://www.techgeeknext.com/angular/angular-rxjs-higher-order-mapping-operator)

## Q: When we use the switchMap, mergeMap and concatMap? concatMap(), mergeMap(), switchMap() and exhaustMap(). All of these operators are flattening operators used to flatten observables, but they are applicable in very different scenarios. switchMap and mergeMap are probably going the be the most powerful and frequently used operators. Its is thereby critical to understand the difference between the two in order to spend less time debugging code.

## Q: What is RxJS concatMap?

concatMap : Projects each source value to an Observable which is merged in the output Observable, in a serialized fashion waiting for each one to complete before merging the next - Official RxJS Docs

## Q: What is RxJS mergeMap?

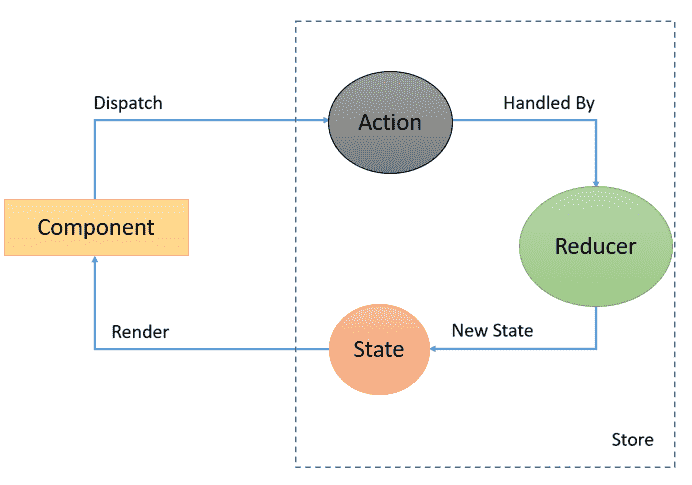
mergeMap : mergeMap allows for multiple inner subscriptions to be active at a time. Because of this, one of the most common use-case for mergeMap is requests that should not be canceled.

## Q: What is RxJS switchMap?

switchMap : Simply put, it means switching to a new observable. The previous inner observable (result of the function you provided) is canceled for each emission and the new observable is subscribed.

## Q: What is NgRx?

*NgRx stands for Angular Reactive Extensions, NgRx Store provides reactive state management for Angular apps inspired by Redux. NgRx has libraries for managing both global and local state. Isolation of side effects in order to achieve a more streamlined component architecture. Developer tooling that makes it easier for developers to construct a variety of applications.*

[](https://www.techgeeknext.com/img/angular/ngrx-components.PNG)

Refer for more questions on [NgRX Interview Questions](https://www.techgeeknext.com/angular/ngrx-interview-questions" \t "_blank)

## Q: When we use zip and combineLatest and withLatestFrom? zip and combineLatest are functions and withLatestFrom is operator that allows to combine a few observable sequences in a different ways, that are really helpful in real world application.

Composing functions and operators usually accept observables as their params and also they return observable that emits array with values produced by argument observables. This result observable emission logic is different depending on which operator or function we use

To understand RxJS more about Zip and combineLatest and withLatestFrom, click [here](https://www.techgeeknext.com/angular/angular-rxjs-combine-latest-zip)

## Q: What is Angular? It's a frontend framework, which was developed to build a single page application (SPA). Click [here](https://www.techgeeknext.com/angular/learning-angular) to understand more about Angular.

## Q: How to build full stack web application?

Angular is extremely famous for modern web application development, Spring Boot and Angular are a strong and developer-friendly combination if you want to create the full stack web application.

To know more about Angular Spring Boot Example click [here](https://www.techgeeknext.com/angular/angular-spring-boot-example)

## Q: How to implement authentication in web application?

We can use Spring Boot JWT with Angular for token authentication in web application.

To know more about Angular Spring Boot JWT Example click [here](https://www.techgeeknext.com/spring/angular-spring-boot-jwt-authentication)

## Q: What is latest version of Angular?

The lastest version of Angular framework is Angular 9 which is in pipeline to release in this month.

To know more about Angular 9 features, click [here](https://www.techgeeknext.com/angular/angular-9-features)

## Q: What is difference between Angular and AngularJS?

Here are few differences between Angular and AngularJS are stated as follows

* AngularJS supports the MVC design model. Angular relies on components and directives instead.
* Angular supports a hierarchical Dependency Injection with unidirectional tree-based change detection. AngularJS doesn’t support DI
* In AngularJS, a specific ng directive is required for the image or property and an event. Angular, on the other hand, use () and [] for blinding an event and accomplishing property binding, respectively
* AngularJS doesn’t have mobile support while Angular does have
* While JavaScript is the recommended language for AngularJS, TypeScript is the recommended language for Angular

## Q: What is Angular Material?

It is a UI component library. Angular Material helps in creating attractive, consistent, and fully functional web pages as well as web applications. It does so while following modern web design principles, including browser portability and graceful degradation.

## Q: What is AOT (Ahead-Of-Time) Compilation?

The Angular Ahead-of-Time compiler pre-compiles application components and their templates during the build process. Apps compiled with AOT launch faster for several reasons.

* Templates are embedded as code within their components so there is no client-side request for template files.
* At the end of the restore view phase of the JSF request lifecycle, Seam attempts to restore any previous long-running conversation context. If none exists, Seam creates a new temporary conversation context.
* Application components execute immediately, without client-side compilation
* The compiler discards unused Angular directives that a tree-shaking tool can then exclude

## Q: What's New with Angular 8 Features? Angular 8 interview Questions

Angular 8 has been official release in May 2019, new features in Angular 8 are limited, but there are still there are a lot of interesting things happening, we have much awaited Ivy compiler as an opt-in feature, ng deploy is officially support by CLI and many more.

Below are the most prominent features of Angular 8:

* Differential Loading of Modern JavaScript:

It is the most important feature in terms of performance, so new apps generated by Angular CLI to have two different bundles for legacy JavaScript(ES5) and modern JavaScript(ES2015+). This translation has provided better the speed for modern browsers which has ES2015 support to download small app bundles.

* Angular 8 supports TypeScript 3.4

In the newest version of Angular 8, it has updated the core dependencies which include tools like RxJS and TypeScript to v3

* The new compiler for Angular 8 is Ivy Rendering Engine

Another much awaited feature, the new rendering engine, and Bazel, the new build system, which make possibility to build your CLI application more quickly. An opt-in preview of the two should be available shortly

* Angular 8 enhance supports for building Web Workers

The improvements that have been made to Angular CLI bundling to eliminates the need to put the web workers in a separate file

* Angular 8 provides dynamic imports for lazy-loaded modules

Lazy-loaded routes now use the standard dynamic import syntax instead of a custom string. This means that TypeScript and linters will be strict checking when modules are missing or misspelled. lazy-loaded import that looked like this:

{ path: '/admin, loadChildren: './ admin / admin.module#AdminModule' }

Will be looked like this:

{ path: `/ admin`, loadChildren: () => import(`./admin/ admin.module`).then(a => a.AdminModule) }

* Improvement of CLI workflow

The CLI continues to improve, and the ng build, ng test and ng run are now designed to be expanded by third-party libraries and software. For example, with a deploy order, AngularFire is already using these new capabilities.

* Angular 8 supports Unit Test Helpers

2 New Unit Test Helpers were introduced so that we can upgrade Angular Project to Unit Test Angular Service

The new feature of Angular 8 has benefits, such as:

* Faster loading time
* Pre-compilation of code and rebuilding is done more faster
* Reduction in app bundle size- reduce up to 90%
* It does not affect SEO negatively

Shortly Angular 9 will be released around November 2019, and Ivy is expected to be an Angular version 9 engine by design. Watch out for an additional feature with Angular 9.

## Q: What's new in Angular 6?

Below major new aspects introduced in Angular 6:

#### **RxJS 6**

- Angular 6 makes use of RxJS 6 internally, RxJS released a library called rxjs-compat, that allows you to still using one of the “old” syntaxes.

#### **Elements:**

lets you wrap your Angular components as Web Components and embed them in a non-Angular application

#### **i18n (internationalization):**

Without having to build the application once per locale, any Angular application can have “runtime i18n”

#### **Tree-shakeable providers:**

recommended, way to register a provider, directly inside the @Injectable() decorator, using the new providedIn attribute

#### **New Rendering Engine:**

Ivy - increases in speed and decreases in application size.

## Q: What is ViewEncapsulation and how many ways are there do to do it in Angular?

To put simply, ViewEncapsulation determines whether the styles defined in a particular component will affect the entire application or not. Angular supports 3 types of ViewEncapsulation:

* Emulated : Styles used in other HTML spread to the component
* Native : Styles used in other HTML doesn’t spread to the component
* None : Styles defined in a component are visible to all components of the application

|  |
| --- |
| Features found both in Angular and Blazor:  DOM  HTML/CSS  Components  Binding  Dependency Injection  Async observables  Open issues:  Library of controls  Community support  If open issues are resolved even, Blazor wins. Reasons: C#, same code as in server, testability, SLDC |

## Question 1. What is Blazor?

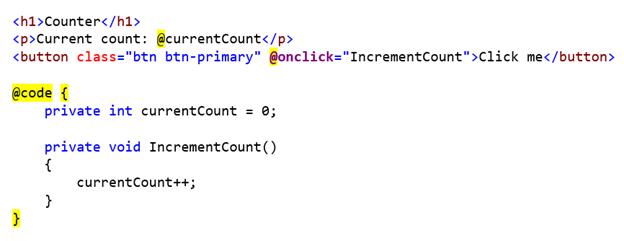
Blazor is a free, open-source, cross-platform Web framework that allows developers to build modern, scalable, and cross-platform Web applications using C# and .NET. Blazor developed by Microsoft and the open-source community is initially designed to keep in mind C# and .NET developers who want to build web client applications using C# language. Blazor is modern, fast, and rapidly evolving.

Most web client applications are written using JavaScript where code mostly runs in the browser. Blazor framework allows developers to create rich interactive UIs using C# instead of JavaScript.

Blazor supports both client-side and server-side coding. Both server-side and client-side app logic is written in .NET.

Even though the code is written in .NET and C#, Blazor renders the UI as HTML and CSS for wide browser support, including mobile browsers.

Here is a simple example of Blazor code that shows how HTML and C# are in the same file and how a function can be called from HTML code.



Blazor integrates with modern hosting platforms, such as **Docker**.

Want to learn more about Blazor, start here: [Introduction to Blazor with .NET Core](https://www.c-sharpcorner.com/article/introduction-to-blazor-with-net-core/)

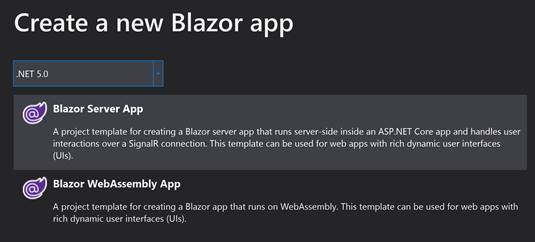
## Question 2. Why use Blazor?

Blazor is developed for developers who are not comfortable with JavaScript and mostly have C# and .NET backgrounds. Blazor offers the following advantages,

* Write code in C# instead of JavaScript.
* Leverage the existing .NET ecosystem of .NET libraries.
* Share app logic across server and client.
* Benefit from .NET's performance, reliability, and security.
* Stay productive with Visual Studio on Windows, Linux, and macOS.
* Build on a common set of languages, frameworks, and tools that are stable, feature-rich, and easy to use.

## Question 3. What types of apps we can build using Blazor?

Blazor supports two types of apps, Blazor Sever app, and Blazor WebAssembly App.



A Blazor Server app runs server-side inside an ASP.NET Core app and handles user interactions over a SignalR connection.

A Blazor WebAssembly app runs in a browser on the client-side via WebAssembly.

## Question 4. What are the components in Blazor?

Component is a fundamental concept on Blazor and all Blazor apps are based on components. All UI elements in Blazor are components such as a page, dialog, or data entry form. From a code perspective, components in Blazor are C# classes built into .NET assemblies that define UI rendering logic and handle control and page events.

 A Blazor component uses the combination of Razor, HTML, and C# code. A component is the base element of the Blazor application, i.e., every page is considered as a component in Blazor. Here is a simple client-side Blazor app that shows the default page component code.

1. @page "/"
2. <h2>@Title</h2>
3. @functions {
4. const string Title = "Blazor";
5. }

In the above code, HTML and code behind are in the same HTML file. The HTML part of the component contains Razor syntax as well as HTML tags and the code behind the section contains the actual logic. In short, in this method, we can add view markup and logic on the same page. The logic is separated by using a function block.

In function block, we can define all the properties that are used in view markup and the methods are bound with control as an event.

In Blazor, we can also write components with code-behind, where C# code is written in a separate file. Here is a detailed article on [components in Blazor](https://www.c-sharpcorner.com/article/creating-component-in-blazor/)

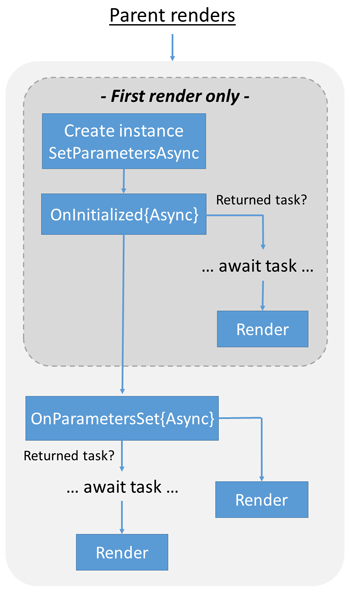
## Question 5. What are Blazor lifecycle and Blazor lifecycle methods?

Blazor framework lifecycle methods represent how various methods are executed when a Balzor application starts, ends, and in between. Blazor uses both synchronous and asynchronous lifecycle methods. Override lifecycle methods to perform additional operations on components during component initialization and rendering.

The following diagrams and the lifecycle events represent Blazor component life cycle events, DOM event processing, and render lifecycle methods and how they are executed in the lifecycle of an application.

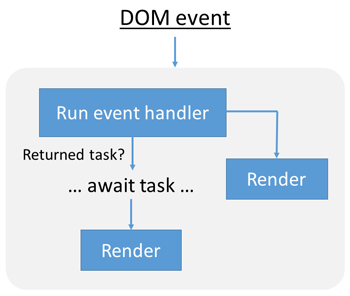
Component lifecycle events,

1. If the component is rendering for the first time on a request:
   * Create the component's instance.
   * Perform property injection. Run SetParametersAsync.
   * Call OnInitialized{Async}. If a Task is returned, the Task is awaited and then the component is rendered. If a Task isn't returned, render the component.
2. Call OnParametersSet{Async}. If a Task is returned, the Task is awaited and then the component is rendered. If a Task isn't returned, render the component.



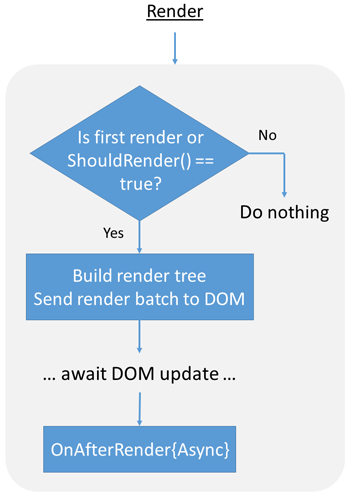
Document Object Model (DOM) event processing,

1. The event handler is run.
2. If a Task is returned, the Task is awaited and then the component is rendered. If a Task isn't returned, the component is rendered.



The Render lifecycle,

1. If this isn't the component's first render or ShouldRender is evaluated as false, don't perform further operations on the component.
2. Build the render tree diff (difference) and render the component.
3. Await the DOM to update.
4. Call OnAfterRender{Async}.



## Question 6. What is Blazor WebAssembly?

Blazor WebAssembly is a single-page app (SPA) framework for building browser-based web apps with .NET. The magic that runs a C#/.NET app inside a web browser is called WebAssembly (wasm). Blazor WebAssembly is based on open web standards and works in all modern browsers without the need for any plugin or complication.

Blazor WebAssembly runs .NET code in the browser with WebAssembly. WebAssembly uses **JS Interop** to generate and interact with JavaScript in the browser.

Technically, WebAssembly is a low-level assembly language runtime with a compact binary format that provides a way to run the code written in different high-level languages in the browser at the same speed as native.

**Official definition of WebAssembly**

WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.

 WebAssembly is efficient and fast, safe, open and debuggable, and part of an open web platform.

**Efficient and fast**

 The Wasm stack machine is designed to be encoded in a size- and load-time-efficient binary format. WebAssembly aims to execute at native speed by taking advantage of common hardware capabilities available on a wide range of platforms..

**Safe**

 WebAssembly describes a memory-safe, **sandboxed execution environment** that may even be implemented inside existing JavaScript virtual machines. When embedded in the web, WebAssembly will enforce the same-origin and permissions security policies of the browser.

**Open and debuggable**

 WebAssembly is designed to be pretty-printed in a textual format for debugging, testing, experimenting, optimizing, learning, teaching, and writing programs by hand. The textual format will be used when viewing the source of Wasm modules on the web.

**Part of the open web platform**

 WebAssembly is designed to maintain the version less, feature-tested, and backward-compatible nature of the web. WebAssembly modules will be able to call into and out of the JavaScript context and access browser functionality through the same Web APIs accessible from JavaScript. WebAssembly also supports non-web embeddings.

## Question 7. What is Blazor Server?

Blazor server or server-side Blazor runs on a web server inside ASP.NET Core and separates business logic from the UI. The UI is developed as an ASP.NET Core app and the backend code is hosted as Razor components on the server. UI updates from the server to the client-side are handled over a SignalR connection.

## Question 8. What platforms does Blazor support?

Blazor apps can be installed and run on Windows, Linux, macOS, Cloud, and Web platforms. Blazor supports both client-side and server-side applications. Blazor server-side applications are developed using ASP.NET Core. The client-side Blazor apps run in web browsers on WebAssembly (wasm).

Blazor is used to develop Windows client applications, web applications, cloud-native applications, and native mobile apps.

## Question 9. What are templated components in Blazor?

Blazor uses the Razor template engine that generates HTML and serves to the web browsers. We can mix HTML and C# syntax in the Razor templates and Razor engine then compiles the Razor templates to generate the HTML.

Blazor (version 0.6.0) supports the templated component. It is a component that accepts one or more UI templates as parameters and these parameters can be used for component rendering. It allows us to write high-level reusable components. Using templated components, we can write a generic and more reusable component.

 We can create the templated component by using one or more component parameters of type RenderFragment or RenderFragment<T>. The RenderFragment is a part of UI which is rendered by the component. It may have parameters that are used during the rendering of the component or while the RenderFragment is invoked.

* [Introduction To Templated Components In Blazor](https://www.c-sharpcorner.com/article/introduction-to-templated-components-in-blazor/)

## Question 10. What is routing in Blazor?

A Route is a URL pattern and Routing is a pattern matching process that monitors the requests and determines what to do with each request.

Blazor server app uses ASP.net Core Endpoint Routing. Using MapBlazorHub extension method of endpoint routing, ASP.net Core is starting to accept the incoming connection for the Blazor component. The Blazor client app provides the client-side Routing. The router is configured in the Blazor client app in App.cshtml file.

The Blazor Server app allows to set fallback route. It operates with low priority in routing matching. The fallback route is only considered when other routes are not matched. The fallback route is usually defined in \_Host.cshtml component.

## Question 11. What is data binding is Blazor?

Blazor supports the following ways of data banding.

* One-way Data Binding
* Two-way Data Binding
* Event Binding

**One-way data binding is also known as an interpolation in other frameworks, such as Angular**. It is very similar to Razor and also it will be quite straightforward. In one-way binding, we need to pass property or variable name along with @ i.e. @Name (here Name is either the property or variable).

1. @page "/databindingExample"
2. <h3>One-way Data Binding</h3>
4. <p>
5. @currentTask
6. </p>
8. @functions {
9. string currentTask = "Test One-way Data Binding!";
10. }

Two-way data binding is achieved by using the "bind" attribute. For example,

Enter your name: <input type="text" bind=@Name /><br />

Event binding is in Blazor is supported by using function name along with @ i.e. @ButtonClicked (here ButtonClicked is function name).

Here is a detailed article on [data binding in Blazor](https://www.c-sharpcorner.com/article/data-binding-in-blazor/).

## Question 12. How to deploy the Blazor app in Azure?

We can use Visual Studio 2019 to deploy Blazor applications to the Azure cloud. To publish the Blazor app on Azure, Right-click on the Server project of your solution and click publish.

It will open the “Pick a publish target” window. Select “App Service” from the left menu. Select the “Create New” radio button and click on “Create profile”.

The next window will ask you to log in to your Azure account if you are not logged in. Once the login is successful, a “Create App Service” window will open.

The fields of this window are prepopulated as per the configuration of your Azure account. However, you can change these values as per your requirement.

## Question 14. What is authorization in Blazor Server

Authorization is a process to validate that the user has rights to access the application resource. In other words, it helps you to control user access to a resource based on roles, claims, and policies. Blazor uses ASP.NET Core authorization mechanism and it can be achieve using attributes, built-in components and by defining authorization rules.

The AuthorizeView is Blazor's built-in component that able to show page content based on the user's authentication state. This component also supports policy-based authorization and role-based authorization. This component is very useful when you want to show page content based on the role, policy, or authentication status of the user. It uses AuthenticationStateProvider to know the user authentication state.

This component provides Authorized, and NotAuthorized render fragments. The Authorized fragment renders when the user is authenticated, and NotAuthorized fragment renders when the user is unauthenticated. Both fragments accept other interactive components.

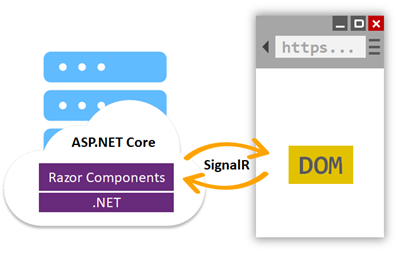
Blazor supports role-based, policy-based, and claim-based authorization.

Here is a detailed article on Blazor Authorization

* [Authorization In Blazor Server App](https://www.c-sharpcorner.com/article/understand-basic-of-authorization-in-blazor-server-app/)

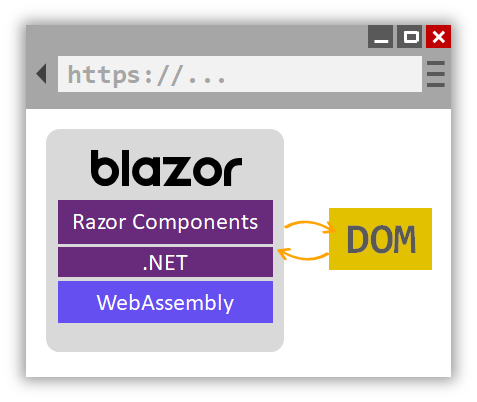
## Question 15. What is the difference between Blazor Server and Blazor WebAssembly?

Blazor offers two models to build applications, Blazor server, and Blazor WebAssembly. Blazor Server apps host Blazor components on the server and handle UI interactions over a real-time SignalR connection.



As you can see from the above image, ASP.NET Core is responsible for rendering Razor components and compile C# code on the server and the rendered UI elements are sent back to the browser using SignalR.

Unlike Blazor Server, the Blazor WebAssembly apps host components in the browser on the client-side using a WebAssembly runtime that is designed for .NET only. The **WebAssembly runtime is responsible for converting .NET into browser elements**. All components and their rendering process all are handled in the client-side browser.



 As you can see from the above diagram, the DOM is generated by Blazor. All Razor components and .NET code is hosted and executed in the client browser using .NET-based WebAssembly runtime.

## Question 16. When should I use Blazor Server vs Blazor WebAssembly?

Blazor server runs on the server-side and requires a web server such as IIS. You can compare Blazor server with an ASP.NET MVC application. Here are some of the use cases where you may want to choose Blazor Server over Blazor WebAssembly.

* If you’ve an existing MVC or Razor pages app and want to continue expanding that with Blazor Server is probably a better choice for you. You don’t need to rewrite your business logic or database libraries. Your application can use MVC and Razor Pages for server-rendering and Blazor for client-side interaction.
* Blazor Server apps are good where you want your client browser to do less work. In Blazor Server, most of the work is done on the server. It suits best for where you have a reliable low-latency network connection.
* Blazor Server apps are not recommended for real-time frequent updates since server sider rending could be slower.
* Blazor Server apps require an active network connection, offline scenarios are not supported.

## Question 16. When should I use Blazor WebAssembly vs Blazer Server?

Blazor WebAssembly is an option for C# and .NET developers who want to build client-side browser apps that run in the browser, without a need for a server. Unlike Blazor Server, a Blazor WebAssembly app does not require a server, and all code is executed in the browser. Here are some of the use cases where you may want to choose Blazor WebAssembly over Blazor Server.

* You do not have experience working with ASP.NET and server-side.
* You do not want to go through managing and deploying code on the server.
* You want to take advantage of client browsers, compute, memory, and storage.
* You want to build real-time responses and build a UI that updates frequently.
* You want to build fast static websites.
* You want to take advantage of modern technology such as progressive web apps (PWA) to implement live notifications, local browser storage, messages, and server workers.
* You want to build web apps that can completely work offline.

## Question 17. Can I use both Blazor WebAssembly to build full-stack apps?

Yes. You can use .NET and C# to write your backend or server-side code and use Blazor WebAssembly for the front-end UI.

## Question 18. Can I debug Blazor WebAssembly?

Yes. Blazor WebAssembly apps can be debugged using the browser dev tools in Chromium-based browsers (Edge/Chrome). You can also debug your app using Visual Studio and Visual Studio code by setting debug breaks and step through the code.

**References**

* <https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor>
* <https://www.c-sharpcorner.com/technologies/blazor>
* <https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-3.1>
* <https://devblogs.microsoft.com/aspnet/blazor-server-in-net-core-3-0-scenarios-and-performance/>