1. How can you communicate with components in angular?

In Angular, components can communicate with each other in several ways:

1. **Input and Output Properties**:
   * Components can communicate by passing data through input properties (bindings) and emitting events through output properties.
   * Parent components can pass data to child components using input properties.
   * Child components can emit events to communicate with their parent components using output properties.
   * This is a unidirectional data flow.
2. **@ViewChild and @ContentChild**:
   * Parent components can access child components using **@ViewChild** or **@ContentChild** decorators.
   * **@ViewChild** is used to access a single child component or element, while **@ContentChild** is used to access a projected content within the component's template.
   * This allows parent components to directly interact with their child components or elements.
3. **Services**:
   * Components can communicate indirectly through services.
   * Services are singleton objects that can hold and manipulate data, and they can be injected into multiple components.
   * Components can share data and functionality by injecting the same service instance.
4. **RxJS Observables and Subjects**:
   * Components can communicate using RxJS Observables and Subjects.
   * Components can subscribe to Observables to receive data, and they can emit data using Subjects.
   * This provides a flexible and powerful way to handle asynchronous data streams.
5. **EventEmitter**:
   * Components can communicate using Angular's **EventEmitter**.
   * **EventEmitter** allows components to emit custom events that other components can subscribe to.
   * This is useful for custom communication between components.
6. **Angular Router**:
   * Components can communicate indirectly through the Angular Router.
   * Components can navigate to different routes, passing data through route parameters or query parameters.
   * Components can also share data using route resolvers or route guards.
7. **Local Storage or Session Storage**:
   * Components can communicate by storing data in local storage or session storage.
   * Data stored in local storage or session storage can be accessed by multiple components within the same browser session.

These are some of the common ways components can communicate with each other in Angular. The choice of communication method depends on the specific requirements and architecture of the application.

2**. What is the Difference between view child and view children in Angular?**

**@ViewChild** is used to query for a single element or component from the view DOM.

**@ViewChildren** allows you to react to changes in child components or elements and update the parent component's view accordingly.

* Use **@ViewChild** when you need to access a single child element or component.
* Use **@ViewChildren** when you need to access multiple child elements or components.

3. What is Lazy loading in Angular?

It allows you to loads the modules on demand.

Syntax:

**const routes: Routes = [**

**{ path: 'lazy', loadChildren: () => import('./lazy/lazy.module').then(m => m.LazyModule) }**

**];**

**Benefits:**

* **Improved initial loading time:** Only essential modules are loaded initially, reducing the size of the initial bundle.
* **Faster startup**: Smaller initial bundles mean faster startup times for the application.
* **Better resource utilization:** Modules are loaded on-demand, reducing unnecessary resource consumption.

**4. Encapsulation in Angular?**

Encapsulation in Angular refers to the practice of **bundling the data and methods that operate on the data into a single unit known as a component.**

*We can achieve the encapsulation through the feature modules in angular.*

5. **What are the decorators in Angular?**

In Angular, decorators are ***functions that can be attached to classes, methods, properties***, or parameters to modify or add metadata to them.

* **@Component**: Defines a component and its metadata.
* **@Directive**: Defines a directive and its metadata.
* **@NgModule**: Defines a module and its metadata.
* **@Pipe**: Defines a pipe and its metadata.
* **@Injectable**: Marks a class as a candidate for dependency injection.
* **@Input**: Declares an input property that can be passed to a component.
* **@Output**: Declares an output property that emits events from a component.

6.  **What are the directives in Angular?**

In Angular, directives are a powerful feature that allows you to extend the HTML syntax with custom behaviors and functionalities. ***Directives are used to manipulate the DOM***.

1. **Component Directives**:
   * Components are declared using the **@Component** decorator and typically consist of a TypeScript class and an HTML template.
   * They encapsulate the presentation logic, behavior, and styling of a part of the UI.
   * Components are reusable and composable, allowing you to create complex UIs by combining smaller, reusable components.
2. **Attribute Directives**:
   * Attribute directives are declared using the **@Directive** decorator.
   * They are used to change the appearance or behavior of an existing element by applying special behaviors to them.
   * Attribute directives can be applied to elements as attributes in HTML tags, and they can accept input properties to customize their behavior.
3. **Structural Directives**:
   * Structural directives are declared using the **@Directive** decorator with a special selector that includes an asterisk (**\***).
   * They are used to add or remove elements from the DOM based on certain conditions.
   * Structural directives are typically used in conjunction with **ng-template** elements to conditionally render content in the DOM.
4. Pipes in Angular.
   * In Angular, pipes are a feature **that allows you to transform data** in your templates before displaying it.

Syntax:

Pipes are applied to expressions in Angular templates using the pipe operator (|).

The pipe operator is followed by the name of the pipe, optionally followed by one or more parameters separated by colons (:).

Built-in Pipes:

Angular comes with several built-in pipes that provide common transformations, such as formatting dates, numbers, currencies, and text, as well as filtering and sorting arrays.

Some of the built-in pipes in Angular include DatePipe, DecimalPipe, CurrencyPipe, UpperCasePipe, LowerCasePipe, JsonPipe, AsyncPipe, SlicePipe, FilterPipe, and OrderByPipe.

Custom Pipes:

You can also create custom pipes to perform custom transformations or apply specific business logic to your data.

Custom pipes are TypeScript classes decorated with @Pipe, and they implement the PipeTransform interface, which requires them to implement a transform method.

Custom pipes can have parameters, which allow you to customize their behavior based on input values.

Chaining Pipes:

You can chain multiple pipes together in a single expression, with each pipe transforming the output of the previous pipe.

Pipe expressions are evaluated from left to right, with the output of each pipe becoming the input for the next pipe in the chain.

Async Pipes:

The AsyncPipe is a special built-in pipe in Angular that is used to handle asynchronous data streams.

It subscribes to an Observable or Promise and automatically unsubscribes when the component is destroyed, preventing memory leaks.

Parameterized Pipes:

Some built-in and custom pipes accept parameters to customize their behavior.

Parameters are passed to pipes by appending them to the pipe name separated by colons (:) in the template.

1. What is the use of the async pipe in Angular?

In Angular, the **AsyncPipe** is a built-in pipe that is used to handle asynchronous data streams in templates. It simplifies the process of subscribing to Observables or Promises and automatically unsubscribes when the component is destroyed, preventing memory leaks.