ANGULAR INTERVIEW QUESTIONS

PART - 2

What is a Services ?

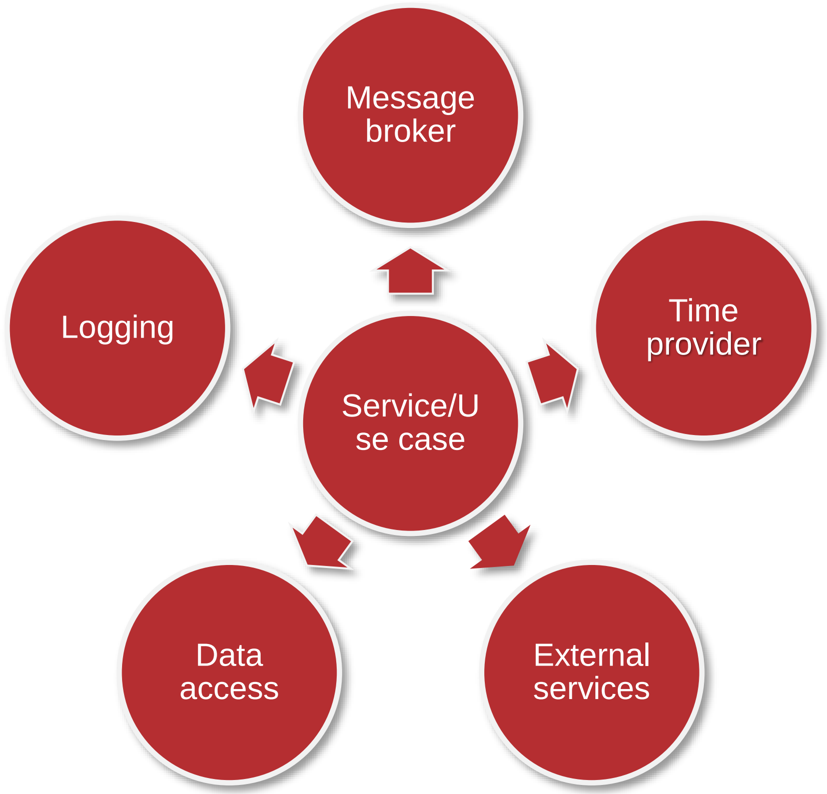
When numerous modules want access to a single piece of functionality, a service is utilized. By enabling you to isolate common functionality from components, services help your application have a better modular structure and greater separation of concerns.

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';

@Injectable({ // The Injectable decorator is required for dependency injection to work
  // providedIn option registers the service with a specific NgModule
  providedIn: 'root',  // This declares the service with the root app (AppModule)
})
export class RepoService{
  constructor(private http: Http){
  }

  fetchAll(){
    return this.http.get('https://api.github.com/repositories');
  }
}
```

2

What is a dependency injection in Angular ?

A important application design pattern known as dependency injection (DI) involves a class asking for dependencies from outside sources as opposed to building them on its own. For resolving dependencies (the services or objects that a class needs to perform its job), Angular includes its own dependency injection mechanisms. As a result, you can have your services depend on other services throughout your application.



What are Life Cycle Hooks available ?

A component instance has a lifecycle that starts when Angular instantiates the component class and renders the component view along with its child views. The lifecycle continues with change detection, as Angular checks to see when data-bound properties change, and updates both the view and the component instance as needed. The lifecycle ends when Angular destroys the component instance and removes its rendered template from the DOM.

ngOnChanges	This method is invoked whenever the value of a data-bound property changes.
ngOnInit	This is invoked whenever the directive or component is initialized after Angular shows the data-bound properties.
ngDoCheck	This is to detect changes that Angular can't or won't notice on its own and to take appropriate action.
ngAfterContentInit	After Angular renders external content into the component's view, this is invoked in response.
ngAfterContentChecked	This is triggered after Angular has examined the material that was projected into the component.
ngAfterViewInit	After Angular initializes the component's views and child views, this is invoked in response.
ngAfterViewChcked	Angular examines the component's views and child views before calling this.
ngOnDestroy	This is the finishing stage before Angular deletes the directive or component.

What is Data binding ?

A component instance has a lifecycle that starts when Angular instantiates the component class and renders the component view along with its child views. The lifecycle continues with change detection, as Angular checks to see when data-bound properties change, and updates both the view and the component instance as needed. The lifecycle ends when Angular destroys the component instance and removes its rendered template from the DOM.

Component to the DOM

- **Interpolation:** {{ value }}: Adds the value of a property from the component.
  Ex: <li>Name: {{user.name}}</li>
- **Property binding:** [property]="value": The value is passed from the component to the specified property or simple HTML attribute.
  Ex: <input type="email" [value]="user.email">

DOM to the Component

- **Event binding: Event binding: (event)="function":** When a specific DOM event happens (e.g.: click, change, keyup), call the specified method in the component.
  Ex: <button (click)="insertData()"></button>

Two-way binding

- **Two-way data binding:** [(ngModel)]="value": Two-way data binding allows to have the data flow both ways. For example, in the below code snippet, both the email DOM input and component email property are in sync.
  Ex: <input type="email" [(ngModel)]="user.email">

What is Interpolation ?

Angular converts a particular syntax known as interpolation into property binding. It's an easy replacement for property binding. Double curly brackets are used to symbolize it (). The name of a component property is frequently seen in the text enclosed by braces. That name is changed by Angular to the string value of the relevant component property.

```
<h3>
  {{title}}
  <img src="{{url}}" style="height:30px">
</h3>
```

6

What is a bootstrapping module ?

Every application has at least one Angular module, known as the bootstrapping module, which is the root module used to launch the application. It is frequently referred to as AppModule. The AppModule produced by AngularCLI would have the following default structure:

```
import { BrowserModule } from '@angular/platform-browser';
  import { NgModule } from '@angular/core';
  import { FormsModule } from '@angular/forms';
  import { HttpClientModule } from '@angular/common/http';

  import { AppComponent } from './app.component';

  /* the AppModule class with the @NgModule decorator */
  @NgModule({
    declarations: [
      AppComponent
    ],
    imports: [
      BrowserModule,
      FormsModule,
      HttpClientModule
    ],
    providers: [],
    bootstrap: [AppComponent]
  })
  export class AppModule { }
```

What is an Observable ?

An Observable is a special Object that is comparable to a Promise and can be used to manage async function. Since observables are not a feature of the JavaScript programming language, we must rely on a well-known observable library called RxJS. Using the new term, the observables are created.

```
 import { Observable } from 'rxjs';

const observable = new Observable(observer => {
  setTimeout(() => {
    observer.next('Hello!');
  }, 2000);
});
```

What is an Observer ?

Observer is an interface for a consumer of push-based notifications delivered by an Observable. It has below structure:

```
interface Observer<T> {
  closed?: boolean;
  next: (value: T) => void;
  error: (err: any) => void;
  complete: () => void;
}
```

A handler that implements the Observer interface for receiving observable notifications will be passed as a parameter for observable as below:

```
myObservable.subscribe(myObserver);
```

What is Multicasting ?

Multi-casting is the practice of broadcasting to a list of multiple subscribers in a single execution.

```
var source = Rx.Observable.from([1, 2, 3]);
var subject = new Rx.Subject();
var multicasted = source.multicast(subject);

// These are, under the hood, `subject.subscribe({...})`:
multicasted.subscribe({
  next: (v) => console.log('observerA: ' + v)
});
multicasted.subscribe({
  next: (v) => console.log('observerB: ' + v)
});

// This is, under the hood, `s
```

10

What is the purpose of base href tag ?

In order to specify how to construct navigation URLs, the routing application should add an element to the index.html as the first child in the tag. You can set the href value as follows if app folder is the application root.

```
<base href="/">
```