Angular

1. **Dependency Injections and How angular handles it.**

=> DI is a service or object that a class needs to perform its methods/functions. It is a design pattern in which a class request dependencies from external resources rather than creating themselves. To inject a dependency in a component, supply the dependency in a constructor as an argument with the type.

1. **Lazy Loading and how to achieve it.**

It is a concept of angular routing. It is used to download web pages only when required using property loadChildern.

const routes:Routes = [

{ path='pathname', loadChildern:()=> import(./module\_path).then(m=> m.ModuleName)}

]

1. **Life Cycle Hooks**

It is a set of processes that an Angular Application goes through starting from initiating to the end of the application.

8 Life Cycle Hooks

* ngOnChanges – whenever the value of a data bound property changes.
* ngOnInit – whenever initialization of directive/component happened after Angular displays the data bound property then this is called.
* ngDoCheck- this is for detection when there is a changes that angular cannot detect.
* ngAfterContentInit – when an external content is projected in the component view
* ngAfterContentChecked –when the external content is projected the component view
* ngAfterViewInit – when Angular initializes component and child view
* ngAfterViewChecked – when Angular has initialized component and child view
* ngOnDestory – when there is no need of a directives

1. **Content Projection (<ng-content>)**

When a user wants to project some information into child component at a particular location, it is called content project. We use <ng-content> element at the child component as a placholder.

Syntax:

App.comp.hmtl

<app-child>

<p childshow>Project this field using select in ng-content element</p>

</app-child>

child.comp.html

<ng-conent select=”[childshow]”/>

Note: If <p show>, then select=”[show’]”.

If <p class=”show” then select=”.show”.

If <p #show> then select=”show”.

Example: A feedback component shows feedback of a product. So a user will enter a feedback from a user-component should show at the feedback-component.to do so the feedback-component would have a ngFor loop to show from content from the array. Where the name of the product being the first placeholder and the comments being the second placeholder.

1. **@viewChild/@viewChildren-**

When a DOM element needs to be manipulated

Syntax:

template: ` <h2 #red>Change this text to red</h2>`

export class Component implements ngOnInit , ngAfterViewInit{

@ViewChild(‘red’) H2R:ElementRef;

ngAfterViewInit( ){

console.log(this.H2R);

this.H2R.nativeElement.setAttribute(‘style’,’color:”red”’);

}

}

1. **@ContentChild/@ContentChildren**

When a child component class wants to access the template content coming from parent component

Syntax:

Child.comp.ts

export class ChildComponent implements ngAfterContentInit{

@ContentChild(“ngContentFromParent”) fromParent: ElementRef;

}

1. REST API Calls
2. What are services? Uses of services, how to create a service
3. getter and setter
4. RxJs - Reactive Extension for Javascript
5. Change Action
6. API Call, 1 URL call, Dependent Call - proper method to do it - forkjoin - Dependent API Calls
7. Observables, Promises - Stream of data

Observables are like promises, both are used to handle async data.

Diff-

1. Observables emits streams of data, but unless it is subscribed the calling function won’t get the data. And Promises return a single values and it will respond to the function as soon as it is called.
2. Observables are external libraries so it needs to be imported from rxjs. Promises are native to JS, so it can be used whenever needed.
3. Observable behaves like an array so operators can be used by importing rxjs operators class.
4. Observables can be unsubscribed.
5. Fetch 1st stream of data
6. How to connect to backend services
7. HTTP Service - get/put/push
8. Optional Variables
9. 2-way Data Binding
10. Data Transfer between components
11. onOnChanges
12. ngOnInit vs constructor
13. Attribute vs structural directives
14. Intercepters
15. handle Exception in angular
16. SwitchMap - dependent API Call
17. How to Test
18. ExpressJS