**Angular Documentation**

**What are Modules?**  
Modules can be described as self-contained chunks of the functionality in your application which can run independently. In Angular, Modules are the collection of the Components, Service directives, and Pipes which are related such that they can be combined to form a module. In a more simple way, we can imagine the Modules as the packages which are self-contained set of the classes and methods  
  
 **import {**

**BrowserModule**

**} from '@angular/platform-browser';**

**import {**

**NgModule**

**} from '@angular/core';**

**import {**

**AppComponent**

**} from './app.component';**

**Components**

The most important feature of any Angular application is the component which controls the View or the template that we use. Generally, we write all the application logic for the View that is mapped to this component  
  
**import {**

**Component**

**} from '@angular/core';**

**@Component({**

**selector: 'app-root',**

**templateUrl: './app.component.html',**

**styleUrls: ['./app.component.css']**

**}) export class AppComponent { title = 'app'; }**

**Bindings**

String Interpolation:-   
Interpolation refers to embedding expressions into marked up text. By default, interpolation uses the double curly braces {{ text }} as delimiters.  
  
ngModel:-   
The ng-model directive binds the value of HTML controls (input, select, text-area) to application data. It is a part of the FormsModule. This directive is used by itself or as part of a larger form

EventBinding:-  
To bind to an event you use the Angular event binding syntax. This syntax consists of a target event name within parentheses to the left of an equal sign, and a quoted template statement to the right.

Html:-  <input type="text" (keyup)="onKeyUserName($event)"

js :- onKeyUserName(value :any){

    this.name = value.target.value

  }

PropertyBinding:-It is used to change the properties or attribute of the html element.Property Binding provide Boolean value but interpolation doesn’t provide Boolean

Template reference:- A template reference variable is a reference to a DOM element or directive within a template. Using template reference variable, we can access the values of DOM element properties

We can use template reference variable by two ways.  
**1.** Using **#**  
<input type="text" #myVar>

Here myVar is a template reference variable.

**2**. Using ref-

<input type="text" ref-myVar>

@ChildView:-

This decorator is used to combine plain html with other Angular Component.  
Many times we can coordinate these multiple components and HTML elements directly in the template by using template references

HTML File :- <input #inputLogin type="text" name="username" id="username" required="required" [(ngModel)]="credentials.username"/>

Ts File:-

export class BindingsComponent implements AfterViewInit{

@ViewChild('inputLogin') elementRef! : ElementRef

ngAfterViewInit(){

this.elementRef.nativeElement.focus()

}

}

Component Lifecycle:-  
Component Contain 8 phase to execute the program, Angular calls the hook methods

1. **ngOnChanges()**   
 Respond when Angular sets or resets data-bound input properties

2. **ngOnInit()**  
 Initialize the directive or component after Angular first displays the data-bound properties and sets the directive or component's input properties.

**3. ngDoCheck()** Detect and act upon changes that Angular can't or won't detect on its own

**4. ngAfterContentInit()** Respond after Angular projects external content into the component's view, or into the view that a directive is in

**5. ngAfterContentChecked()** Respond after Angular checks the content projected into the directive or component

**6. ngAfterViewInit()** Respond after Angular initializes the component's views and child views, or the view that contains the directive

**7. ngAfterViewChecked()** Respond after Angular checks the component's views and child views, or the view that contains the directive

**8. ngOnDestroy()** Cleanup just before Angular destroys the directive or component

**Directives**

1. **ngIf-else:-** Used To check the conditions
2. **ngFor:-** Used To run the loop
3. **ngClass:-** NgClass is used to Add or remove CSS classes on an HTML element
4. **ngStyle:- NgStyle** is used to add some style to an HTML element
5. **Custome Directive:-** Angular Directives enhance the capability of HTML elements by attaching custom behaviors to the DOM. Provide Additional Feature to HTML Element

**Pipes**

Pipes is used to the data format, It transform one format to another format

**Custome Pipes**

Command to generate Custome pipe:- ng g p name  
  
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

name: 'usdToInr'

})

export class UsdToInrPipe implements PipeTransform {

transform(value: number, ...args: number[]): unknown {

const [x] = args

return value\*x;

}

}  
//html file  
<h2>{{20 | usdToInr : 100}}</h2>

**Communication between Components**

@Input is used to pass the data from parent Component to child  
Syntax:-

Parent  
<app-child-component [item] = "passDataToChild"></app-child-component>  
Child

@Input() item = 0

@Output is used to get the data from Child Component to parent  
EventEmitter is used to receive or emit the data.

**Template Driven Form**Form Handling is managed by the template side (HTML file ) that is called Template driven Forms

**Reactive Form**Form Handling is managed by the Component side (Js file ) that is called Reactive Forms

**Routing**Routing in AngularJS is**a method that allows you to create Single Page Applications**. It enables you to create different URLs for different content in your web applications

**1.Route Params:-** If we want to pass values between views, then we can use route params. For example, if we’re going to pass an ID from one route to another and fetch the id on a component onInit(), then we can use route params.

**1.Query Params:-** To pass query parameters in Angular, you don’t need to define anything while defining the routes in the **app-routing.module.ts**

**Gaurds**

We use the Angular Guards to control, whether the user can navigate to or away from the current route.

The Angular Router supports Five different guards, which you can use to protect the route

1. **CanActivate:-** Guard navigation to the route
2. **CanDeactivate:-** Guard navigation away from the current route
3. **Resolve:-** perform route data retrieval before route activation
4. **CanLoad:-** Guard navigation to the feature module loaded asynchronously
5. **CanActivateChild:-** Guard navigation to the child route

**Service**

Service is a piece of reusable code with a focused purpose. A code that you will use across multiple components in your application

**LazyLoading**

Lazy loading is the technique where angular loads the Modules only on a need basis rather than all at once. It is also called**on-demand loading**. By default, Angular Loads the modules eagerly. Lazy Loading of Angular Modules reduces the initial load time of the app.

**Http,Observables**HttpModule is used to communicate with the server,Whenever we want to fetch the data from db or server in that case http come into the picture,

**Observable:-** when http receive the data from server then observable hold that data and provide our service class, Observable is a part of ReactiveX which external library, It is used for the asynchronous programming  
**signUpData(data : any) : Observable<any>{**

**return this.httpClient.post<any>("http://localhost:3000/signup",data)**

**}**

**Interceptors**The angular interceptor is a medium connecting the backend and front-end applications. Whenever a request is made, the interceptors handle it in between. They can also identify the response by performing Rxjs operators.

**Ways to Use Interceptors in Angular  
1. Loaders**   
Interceptors can be used as loaders whenever there exist different active requests. A loader function with both, hide and show features is used to handle the requests.

**2. URL**   
Changing the URLs is what the interceptor angular is capable of. You can change the URL, and the interceptor will behave like an API interceptor used to add prefixes.

**3. Headers**   
Angular interceptors manipulate the headers as they provide features such as authentication and authorization. The angular HTTP interceptors are used to protect the application against XSRF.

**4. Converting**   
Interceptor can even convert the format of an API that we receive. A more likely example is to convert the XML file to a JSON file.

**5. Errors**   
Interceptors are used in two ways to detect the errors- retry the HTTP call, and the second one is to keep a check on the status of the exception that occurred.

**6. Authentication**   
Authentication is the basic functionality added to any application to check the user’s authentication. This is a common and basic use of the interceptors to make a check-in at the application’s usage. It connects various parameters such as the refresh tokens, adds bearer tokens, and redirects to the login URL.

**<Ng-Template>**We can define <Ng-Template> content that is only being rendered by Angular when you, whether directly or indirectly  
 if we wrap content inside an [<ng-template>](https://angular.io/api/core/ng-template) without instructing Angular to render it, such content will not appear on a page  
<p>Hip!</p> <ng-template> <p>Hip!</p> </ng-template> <p>Hooray!</p>  
Output:-   
Hip  
  
Hooray!

**<Ng-Content>**

The ng-content tag acts as a placeholder for inserting external or dynamic content. The Parent component passes the external content to the child component. When Angular parses the template, it inserts the external content where ng-content appears in the child component’s template  
import { Component, Output, EventEmitter } from '@angular/core';  
@Component({  
selector: 'app-fancybtn',  
template: `<button>  
<ng-content></ng-content>  
</button> `})  
export class FancyBtnComponent {}