**Angular Framework**

Angular is a javascript framework which is used to create SPA(single page applications).

Developed by Google.

It is built using Typescript and uses component based architectiure.

The default port is 4200.

**ng s --port 4201 –open** (open application in 4201 port in default browser automatically).

**Install Angular CLI:**

The below command will globally(-g) install Angular.

***npm install -g @angular/cli***

for specific version use *🡪* ***npm install -g @angular/cli@version***

* ***npm cache verify***
* ***npm help***

Uninstalling 🡪 ***npm uninstall -g angular-cli @angular/cli***

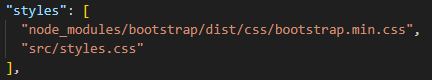
Angular Version 🡪 ***ng v***

Create Angular Application 🡪 ***ng new Application\_name***

Running Application 🡪 ***ng serve***  
Installing Node\_modules 🡪 **npm install** (or) **npm i**

Installing Bootstrap 🡪 ***npm i --save*** [***bootstrap@3***](mailto:bootstrap@3)

add below path in angular.json file at styles array.



Installing Angular Material 🡪 ***ng add @angular/material***

**Flow:**

index.html **→** main.ts **→** app.module.ts **→** app.component.ts

1. **Angular.JSON**

Contains all properties and configurations of Angular project.

Builder starts application from main file by finding it in this json file.

A black screen with white text

Description automatically generated

1. **Main.ts:**

Entry point of Application.

This file calls the function bootstrapModule(AppModule) which tells the builder to bootstrap the AppModule.



1. **App.Module.ts:**

AppModule is defined in App.module.ts (root module).

This module is created by using **@NgModule** decorator.

Module contains all the related components which will be used together.

A screen shot of a computer code

Description automatically generated

**Declarations:**

AppModule has declarations of all the components we are creating with in the app module so that angular is aware of them.

Lists all the components which should be known for the angular at the point of time when it analyzes index.html file

Providers:

**Imports:**

This imports arrays contains all the other modules which are required to use in this application.

From here AppComponent will be loaded.



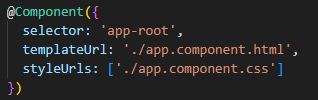
1. **App.Component.ts:**

It is the root component and created using **@component** decorator.  
 it is imported from **@angular/core**

It requires a javascript object as below.

Reads AppComponent and knows the selector

Then it renders AppComponent in index.html



**Selector:**

Used to add the Component in the other Component template.

<app-root></app-root>

1. **Attribute style:**

selector:’[app-root]’ 🡪 <div app-root></div>

1. **Class Style:**

selector:’.app-root’ 🡪 <div class=”app-root”></div>

1. **Element Style: (most preferred):**

selector:’app-root’ 🡪 <app-root></app-root>

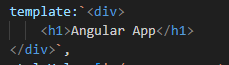
**Template or TemplateUrl:**

Template 🡪 for Inline Template

Use ‘ ‘ for single line code.



Use ` ` for multi-line code.



TemplateUrl 🡪 for External template(html)



**Styles array:**

Has StyleUrls array where component specific style sheets can be placed.

For Inline style 🡪 Styles

For External Style 🡪 StyleUrls

1. **Index.html**

Compiler dynamically adds all the javascript files at the end of this file.

This file calls the root component <app-root></app-root>

**Creation of Component:**

***ng g c component\_name***

to skip test file use:

***ng g c component\_name –skip-tests***

Once component is generated, 4 files will be created. The Component will be automatically added in AppModule in Declarations section.

.html file (template)

.css file (styling)

.spec.ts (for unit testing)

.ts (component)

Binding:

Communication between Component to Template and vice-versa.

1. One-way Binding
2. Two-way Binding

One-way Binding:’

🡪Component to Template

1. Interpolation: {{}}

Uses {{message}} to access message in template.

1. Property Binding: []

Mostly used when we want to change any property value.

serverDisabled=true;

<button class="btn btn-primary" [disabled]="serverDisabled">Save</button>

🡪Template to Component:

1. Event Binding: ()

When an event is triggered the data flows from template to component.

Success(){

alert('Hi saikiran!');

}

<button class="btn btn-primary" [disabled]="serverDisabled" (click)="Success()">Save</button>

Angular Directives:

Used to Manipulate the DOM (Document Object Model)

1. Component Directive
2. Structural Directive
3. Attribute Directive

**Component Directive:**

Special kind of Directive.

**Structural Directives:**

These can change the DOM layout by adding or removing DOM elements.

Looks like a normal HTML attribute but having a leading \*

Affects whole area in the DOM. (elements get added/removed from DOM)

**Attribute Directives:**

These can change the appearance or behavior of an element.

Looks like a normal HTML attribute (possibly with databinding or event binding).

Only affect/change the element they are added to.

Property & Event Binding:

**Components**

**Directives**

**HTML Element**

**Custom properties and Events**

**Custom properties and Events**

**Native properties and Events**

By Default, all properties are accessible only inside that component.

To Explicitly bind data from parent to child component use **@Input** decorator.

* Property names need to have same while accessing. (element)

Child component (contains **@Input**)

A black screen with white text

Description automatically generated

🡪Send data to child component

Parent Component

A screen shot of a computer program

Description automatically generated

🡪 Alias: we can use custom name to access the data from parent component.



A black screen with white text

Description automatically generated

**Binding to custom Events:**

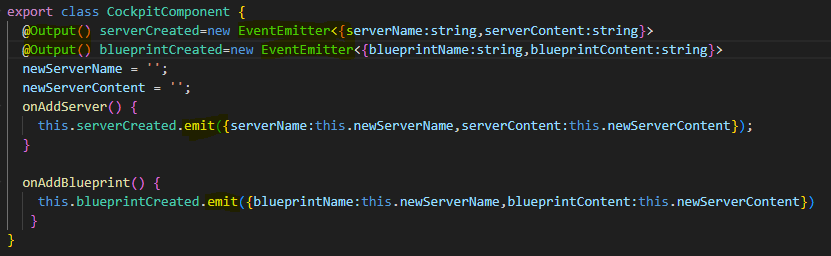
Use **@Output** Decorator to pass data from child to parent when any action is done.

We initialize the property with **EventEmitter**.

The child component raises the event and passes the data as the argument to the event.

The parent component listens to events using [event binding](https://www.tektutorialshub.com/angular/event-binding-in-angular/) and reads the data.

Child component :



Parent Component:

A black background with white text

Description automatically generated

**Assigning Alias to Custom events:**





|  |
| --- |
| Note:  To communicate between parent child we use @Input and @Ouput.  If it is a chain of communications then the above is not recommendable. |

View Encapsulation:

Emulated,(default stylings are applies to only that component)

None,(applies styling globally)

Native

Local References in Template:

Use #

Stores whole html element with all its properties.

We can use this anywhere in the same template to use that element.

Access the value of element:

serverName.value

1. To handle logic primarily in component.ts file use two-way binding.

A black screen with white text

Description automatically generated

1. If you need to access the input element in the template or perform some template-related logic use Local Reference.

A black screen with white text

Description automatically generated

|  |
| --- |
| Note:  Here serverName stores full Html Element.  <input type="text" class="form-control"> |

**Getting Access to the Template & DOM with @ViewChild**

Getting access any html element in typescript file from template.

While calling a method if you want to access a method, we can use local reference.

Sometimes we want access a element before a method call, then use @ViewChild Decorator.

Accessing element in ngOnInit:

@ViewChild('serverContentInput', {static: true}) serverContentInput: ElementRef;

**Projecting content into components with ng-content:**

To use code inside any component selector then need to use ng-content in that child component template.

Life Cycle Hooks:

While creating components, angular goes into different phases. it’ll hook into these phases and execute some code.

ngOnChanges 🡪 Called after a bound input property changes.

1. When new component is created.
2. Properties decorated with @Input changes whenever these properties receive new values.

ngOnInit 🡪Called once the component is initialized.

**Note***: It’ll run after the constructor.*

ngDoCheck 🡪Called during every change detection run.

Event triggered, timer fired, observable resolved

ngAfterContentInit 🡪Called after content (ng-content) has been projected into view.

ngAfterContentChecked ->Called every time the projected content has been checked.

ngAfterViewInit 🡪Called after the components view (and child view’s ) has been initialized.

ngAfterViewChecked 🡪Called everytime the view(and child views) have been checked.

ngOnDestroy 🡪Called right before a component is about to destroyed

@ContentChild:

Access local reference from a parent in child

ElementRef :

We can use the nativeElement property of the [ElelemtRef](https://www.tektutorialshub.com/angular/elementref-in-angular/) to manipulate the DOM. We learned this in our last tutorial on [ElementRef](https://www.tektutorialshub.com/angular/elementref-in-angular/). The [nativeElement](https://www.tektutorialshub.com/angular/elementref-in-angular/) Property contains the reference to the underlying DOM object.

Renderer2:

HostListener:

HostBinding:

We can use instead of Renderer to manipulate DOM element.

TemplateRef:

Services:

Code Reusability.

Dependency Injection:

Approach 1:

1. @Component(...)
2. export class AccountComponent {
3. // @Input() & @Output() code as shown in the previous lecture
5. constructor(private loggingService: LoggingService) {}
6. }

Approach 2: using @Inject

1. @Component(...)
2. export class AccountComponent {
3. // @Input() & @Output() code as shown in the previous lecture
4. private loggingService?: LoggingService; // <- must be added
6. constructor() {
7. this.loggingService = inject(LoggingService);
8. }
9. }

Hierarchical Injector:

AppModule:

The same instance will be given all the child components if the service is mentioned in AppModule’s Providers section.(available application-wide)

AppComponents:

Same instance of service will be available for all Components (but not for other services)

Any Other Component:

Same instance of service is available for the component and all its child components.

Injecting Services into Services:

To get something injected in a service then use @Injectable() to that service.

Cross-Component Communication:

Instead of going with chain of events to communicate between components use services.

Use EventEmitter in service and emit() the data.

Subscribe the Observable in component to get data.

1. **providedIn: 'root'**, it means that the service is a singleton and will be provided at the root level, making it available application-wide.

If you mention as below then no need to add the service in AppModule’s Providers array.

A computer screen with text on it

Description automatically generated

1. if you want to use a different injection scope or if you are not using the **providedIn** option then add the service in AppModule.

A screen shot of a computer program

Description automatically generated

**Routing**

We can navigate either from typescript file or template file.

.ts file 🡪 inject Router from @angular/core

Use navigate() method

Template file 🡪 routerLink=”/home”

<router-outlet></router-outlet>

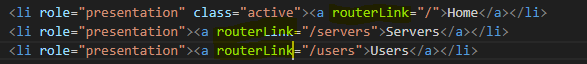
1. If we use below approach to route between components, it’ll refresh everytime(bad practice) when we click on any link.

Restarts on app on every navigation. The application whole state will be lost

A black screen with white text

Description automatically generated

1. Use a special directive called routerLink=”/servers”



routerLink=”../servers” 🡪 go one step back one path

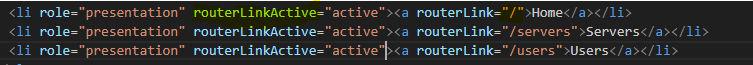
if the component is not a last one then,

you can use routerLink=”servers” instead of “/servers”

if it is last component then,

you need to mention routerLink=”/servers”

**routerLinkActive**:

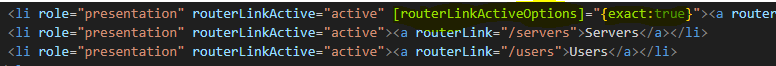


Output:

A screenshot of a computer

Description automatically generated

By defaultly / is selected whicl going to servers, users to avoid the above issue use below approach.



A screenshot of a computer

Description automatically generated

ActivatedRoute

Passing Parameters to Routes:

The ActivatedRoute Object we injected will give us access to the id passed in the URL 🡺Selected User

A screen shot of a computer

Description automatically generated

http://localhost:4201/**users/1/sai**

Use ActivatedRoute to access dynamic data from url

A screen shot of a computer program

Description automatically generated

If you are already on the same page and need to load the data on the same page then the above snapshot method won’t work.

Angular won’t render a component if you’re already on that same component.

Fine to use snapshot method for the first initialization. But To able to react on subsequent changes this won’t work for that we need to use Params.

Params:

Its an Observable( to work with Asynchronous task we use Observables).

A screen shot of a computer program

Description automatically generated

Fetching Route parameters reactively using **route.params** by **subscribing** it

|  |
| --- |
| **Note:**  Angular Cleans up subscriptions once the component is about to destroy.  Subscriptions and components are not tightly coupled even the component is destroyed the subscription might be exist in the memory but angular will clean up. |

Clean Up Subscription Manually:

A screen shot of a computer program

Description automatically generated

Line no:28 will be automatically done by angular once the component is about to destroy.

**Passing Query Parameters and Fragments:**

**In template:**

A computer screen with text

Description automatically generated

In Component:



Setting Up Child (nested) routes:

A screen shot of a computer code

Description automatically generated

Change As below:

A screen shot of a computer code

Description automatically generated

For all Components AppComponent Acts as root component so we are adding <router-outlet> there but for children of other components we need to add it in the parent component.

Here users and servers are acts as parent component for other component so to render those components in parent component use **<router-outlet></router-outlet>** in both users & servers components.

Redirecting and Wildcard (\*\*) Routes:

Catch all the paths you don’t know and redirect to page-not-found page.

|  |
| --- |
| **Note:**  Make sure that wildcard (\*\*) will be the last in the routes section |

A screen shot of a computer code

Description automatically generated

**{ path: '', redirectTo: '/somewhere-else', pathMatch: 'full' }**

Default behaviour of Angular is: {pathMatch: 'prefix'} for all routes.

Now, let's see what is the difference between the two:

If pathMatch: 'prefix' => Angular will search for a prefix of the path(in the URL) in the routes array.

If pathMatch: 'full' => Angular will search for the exact path(in the URL) in the routes array.

Guard:

Running some code before component is loaded.

Protecting Rotes with canActivate Guard:

Auth-guard.service.ts

CanActiveChild:

To guard child components

canDeactivate:

Observables: