**Angular**

**Creating Project**

1. npm init @angular myApp

2.After creating a project check in vscode editor.

3.Can you run the project by using the command npm start.

4.for stop your project by using a ctrl + c (press y).to stop the project.

5.After running the project npm start.

**Installing Angular CLI:**

1.npm install -g @angular/cli

2.After run the command by using an ng

3.set the ng values **set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy. RemoteSinged.**

4.By using ng -- help give the details of the ng command process.

**Using Ng Command:**

1.After install the ng cli then by using an ng serve to run the project.

**Project Structure:**

1.src is the main page in the

index.html file as an important file.to run angular.this index.html has base html.

Style.css to handle the all style page.

2.main.ts

This will run your application run typescript.

3.Favicon.ico

Logo of image.

4.app

Main files has app.component.ts.this one of main components

Each components has main html.css properties

app.routes.ts has an url declared and rotes.url all files.

**Create a Hello word:**

@**Component**({

selector: 'app-root',

imports: [**RouterOutlet**],

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

template:"<h1>Hello world</h1>",

styleUrl: './app.component.css'

})

export class **AppComponent** {

title = 'myApp';

}

By using String interpolation example :Hello world.

@**Component**({

selector: 'app-root',

imports: [**RouterOutlet**],

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

template:"<h1>{{title }}</h1>",

styleUrl: './app.component.css'

})

export class **AppComponent** {

title = 'Hello world !';

}

**Fixing Missing app.modules.ts file.**

This app.modules now show an current version of angular project.

Now,Create standaloneComponents.

1.Where you stand alone an app.components.ts when used as standalone values.no need to create an individual module.

2.it will import all related components.

3.App not using any standalone as this and creating an app will cause a performance issue and file.

4.If can create non standaloneComponents.

By using an command as : **ng new myApp --no-standalone**

If using a standalone project by using an **ng new myApp --no-standalone** this command. Orelse an an normal command **ng new myApp.**

**Data Binding.**

Data binding in Angular is a mechanism that allows communication between the component and the view (HTML). It helps synchronize data between the TypeScript logic and the UI without manually updating the DOM.

Type of data Binding

1. one way Binding.

2.Two way Binding.

One way Binding:

**One-way binding in Angular** is a data binding technique where data flows in only one direction—either from the component to the view or from the view to the component.

**OneWay Data Binding Example:**

By using an oneway data binding an only communication data into view page into component

Example

@**Component**({

selector: 'app-root',

imports: [**RouterOutlet**],

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

styleUrl: './app.component.css'

})

export class **AppComponent** {

title = 'myApp';

}

In an view template url value has an './app.component.html',

<h1>{{title}}</h1>

It will show a chrome page.

Error

<h1>{{title= “Sometext”}}</h1>

Parser Error: Bindings cannot contain assignments at column 8 in [{{title= "Sometext"}}]

Note: In an oneway binding if an html file is not reflected in the component.only show any data in view page.

**Two Way Data Binding**

Two-way data binding in Angular allows synchronization of data between the component and the view. When the user updates the value in the UI, it automatically updates the component, and vice versa.

By using an Ngmodel value.

**Two Way Data Binding Example:**

@**Component**({

selector: 'app-root',

standalone: true,

imports: [**RouterOutlet**,**FormsModule**],

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

styleUrl: './app.component.css'

})

export class **AppComponent** {

fullName = "Testing Input ";

}

In an view template url value has an './app.component.html',

<h1>{{fullName}}</h1>

<input *type*="text" *[(ngModel)]* = "fullName"> //here,*ngModel for using an databinding.*

It will show a chrome page.

**String InterPolation**

**String Interpolation in Angular** is a data binding technique used to display data from the component in the view (HTML). It is done using **double curly braces {{ }}**.

If property value show in html value as been into show and values is known an string Interpolation.it allow an functions also

By using an Syntax : {{}} an show an value in the view page. {{expression}}

**String InterPolation Example:**

@**Component**({

selector: 'app-root',

standalone: true,

imports: [**RouterOutlet**,**FormsModule**],

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

styleUrl: './app.component.css'

})

export class **AppComponent** {

title = "Hello world"

**getName**(){

return "Angular Developer";

}

}

In an view template url value has an './app.component.html',

<h1>{{title}}</h1>

<h2>{{getName()}}</h2>

It will show a chrome page.

**String Interpolation with Attribute Binding**

By using string interpolation can we add this to attribute binding values.

By using an html value has attribute values.

@**Component**({

selector: 'app-root',

standalone: true,

imports: [**RouterOutlet**,**FormsModule**,**CommonModule**],

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

styleUrl: './app.component.css'

})

export class **AppComponent** {

imageurl = "https://picsum.photos/200"

imageAlt = "Angular Logo"

}

<img *[src]*="imageurl" *[alt]*="imageAlt">

Example

One more : <img src="{{imageurl}}" alt="this is the {{imageAlt}}">.

**Property Binding**

Property Binding is a data binding technique in Angular where data flows from the component to the view (HTML element property) using square brackets [].

Check DOM properties. It will show an all html properties

Example: <h1 [innerText] = “tittle”></h1>

|

This has properties

<button [disabled] = “isDisabled”>Click me! </button>.

|

This has properties

**Property Binding Example:**

@**Component**({

selector: 'app-root',

standalone: true,

imports: [**RouterOutlet**,**FormsModule**],

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

styleUrl: './app.component.css'

})

export class **AppComponent** {

title = "Hello world"

}

In a view template url value has an './app.component.html',

<h1 *[innerText]* = "title"></h1>

**Property Binding with Safe Navigation Operator**

The **Safe Navigation Operator (?.)** in Angular is used to **prevent errors** when accessing properties of objects that might be null or undefined. It is especially useful in **property binding** to avoid runtime errors.

Example:

@**Component**({

selector: 'app-root',

standalone: true,

imports: [**RouterOutlet**,**FormsModule**,**CommonModule**],

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

styleUrl: './app.component.css'

})

export class **AppComponent** implements **OnInit**{

*// user = {*

*// name:'John Deo',*

*// age :30*

*// }*

users : any = null

**ngOnInit**(){

this.users={

name : "John Deo",

}

}

}

In a view template url value has an './app.component.html',

<p *[textContent]* = "users?.name"></p>

|

This is a safe navigation operator. This does not show in use error values.

**Event Binding**

**Event Binding** is a data binding technique that allows communication from the **view (HTML)** to the **component (TypeScript)**. It is used to handle user interactions like clicks, keypresses, or input changes.

example : Component onsave()

<button (click) = “onsave()” > Save</button>

**Event Binding Example:**

@**Component**({

selector: 'app-root',

standalone: true,

imports: [**RouterOutlet**,**FormsModule**],

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

styleUrl: './app.component.css'

})

export class **AppComponent** {

clickcount = 0;

**clickme**(){

this.clickcount ++;

}

}

<h1{{clickcount}}> times</h1>

<button (click)= “**clickme()”> click me</button>**

Create a new project for an counter.

Now create an new component **ng generate component counter** by using this command

After then show an counter components in html value

* remove an app.component html.
* Add app.component.ts import the **counterComponent.**

here ,attached main value component of Angular file.

@**Component**({

selector: 'app-root',

imports: [**RouterOutlet**,**CounterComponent**],----> here import new component

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

styleUrl: './app.component.css'

})

export class **AppComponent** {

title = 'simplecount-app';

}

counter.component.ts:

@**Component**({

selector: 'app-counter',

imports: [],

templateUrl: './counter.component.html',

styleUrl: './counter.component.css'

})

export class **CounterComponent** {

count = 0;

**increment**(){

this.count++;

}

**decrement**(){

if(this.count > 0){

this.count--;

}

}

}

Counter.component.html

<h1>Count : {{count}}</h1>

<button *(click)* = "increment()">+</button> &nbsp;

<button *(click)* = "decrement()">-</button>

**Directives**

A **directive** in Angular is a special instruction that **enhances HTML elements** by adding behavior, modifying structure, or styling elements dynamically.

### **Types of Directives:**

**Component Directives** – Custom elements with templates (@Component).

**Structural Directives** – Modify the DOM structure (\*ngIf, \*ngFor, \*ngSwitch).

**Attribute Directives** – Change element appearance/behavior ([ngClass], [ngStyle]).

**Structural Directives**

\*ngFor

@**Component**({

selector: 'app-root',

standalone: true,

imports: [**RouterOutlet**,**FormsModule**,**CommonModule**],

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

styleUrl: './app.component.css'

})

export class **AppComponent** {

movies = ['Zootopia','Batman Vs Superman','Harry Potter','X-men'];

heroes = ['Vijay','Ajith','Karthick',"Arunvijay"];

}

In a view template url value has an './app.component.html',

<h1>\*ngFor for example </h1>

<ul>

<li *\*ngFor* = "let movie of movies"> {{movie}}</li>

</ul>

<ul>

<li *\*ngFor* = "let hero of heroes">{{hero}}</li>

</ul>

\*ngIf

This a condition statement from based on working the ngif method.

@**Component**({

selector: 'app-root',

standalone: true,

imports: [**RouterOutlet**,**FormsModule**,**CommonModule**],

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

styleUrl: './app.component.css'

})

export class **AppComponent** {

showme = true;

}

In a view template url value has an './app.component.html',

<input *type*="checkbox" *[(ngModel)]*="showme">

<p *\*ngIf*="showme">can you see me </p>

**Example for an index,first and last,even and odd example by using \**ngFor, \*ngIf***

@**Component**({

selector: 'app-root',

standalone: true,

imports: [**RouterOutlet**,**FormsModule**,**CommonModule**],

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

styleUrl: './app.component.css'

})

export class **AppComponent**{

fruits = ['Apple','Banana','Cherry','Dates'];

}

In a view template url value has an './app.component.html',

*<!-- Index -->*

<h2>Index Example</h2>

<ul>

<li *\*ngFor* = "let Fruit of fruits; let i = index">

{{i+1}}{{Fruit}}

</li>

</ul>

*<!-- First and last-->*

<h3>First and last Example</h3>

<ul>

<li *\*ngFor* = "let Fruit of fruits; let isFirst = first; let islast = last">

<strong *\*ngIf*="isFirst">First : </strong>

<strong *\*ngIf*="islast">Last : </strong>

{{Fruit}}

</li>

</ul>

*<!-- even and odd-->*

<h3> even and odd</h3>

<ul>

<li *\*ngFor* = "let Fruit of fruits; let isEven = even ; let isOdd = odd"

*[ngClass]*="{'event-item' : isEven ,'odd-item ': isOdd}">

{{Fruit}}

</li>

</ul>

In a view template url value has an './app.component.css,

.event-item{

background-color: aquamarine;

}

.odd-item{

background-color: blanchedalmond;

}

**Attribute Directives**

[ngClass]

This a condition from based on working the ngClass method.values by using an ngClass.

@**Component**({

selector: 'app-root',

standalone: true,

imports: [**RouterOutlet**,**FormsModule**,**CommonModule**],

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

styleUrl: './app.component.css'

})

export class **AppComponent** {

applyred = false

}

In a view template url value has an './app.component.html',

Apply Red<input *type*="checkbox" *[(ngModel)]*="applyred">

<p *[ngClass]*="{'primary': applyred}"> Sampletext </p>

In a view template url value has an ‘./app.component.css’,

.primary{

color:red;

}

.big{

font-family: 'Courier New', Courier, monospace;

font-size: larger;

}

[ngStyle]

Ngstyle to apply a style in value by an object value.

@**Component**({

selector: 'app-root',

standalone: true,

imports: [**RouterOutlet**,**FormsModule**,**CommonModule**],

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

styleUrl: './app.component.css'

})

export class **AppComponent** {

}

In a view template url value has an './app.component.html',

<p *[ngStyle]*= "{'color':'red','font-size':'40px'}">Sample text</p>

**Creating Custom Directive:**

**Pipes**

**Pipes** in Angular are used to transform the data before displaying it in the view. They help in formatting and manipulating values within templates.

Built-in Pipes in Angular

* Date Pipe
* Slice Pipe
* Uppercase and Lowercase Pipe
* Currency Pipe

**Date Pipe:**

Formats date values.

Example:

@**Component**({

selector: 'app-root',

standalone: true,

imports: [**RouterOutlet**,**FormsModule**,**CommonModule**],

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

styleUrl: './app.component.css'

})

export class **AppComponent**{

toDate = new **Date**();

}

In a view template url value has an './app.component.html',

<p>unfromated date:{{toDate}}</p>

<p>fromated date: {{toDate|date}}</p>

<p>mediam date: {{toDate|date:'medium'}}</p>

<p>short date: {{toDate|date:'short'}}</p>

<p>full date: {{toDate|date:'fullDate'}}</p>

<p>long date: {{toDate|date:'longDate'}}</p>

**Slice Pipe**

Extracts a section of an array or string

Example:

.@**Component**({

selector: 'app-root',

standalone: true,

imports: [**RouterOutlet**,**FormsModule**,**CommonModule**],

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

styleUrl: './app.component.css'

})

export class **AppComponent**{

title ='Hello World'

movies = ['Zootopia','Batman Vs Superman','Harry Potter','X-men']

}

In a view template url value has an './app.component.html',

<p>defult value:{{title}}</p>

<p> getting hello: {{title|slice:0:5}}</p>

<h2> movies example</h2>

<ul>

<li *\*ngFor*="let movie of movies|slice:0:3"> {{movie}} </li>

</ul>

**UpperCase and LowerCase:**

Converts text to uppercase.and Converts text to lowercase.

@**Component**({

selector: 'app-root',

standalone: true,

imports: [**RouterOutlet**,**FormsModule**,**CommonModule**],

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

styleUrl: './app.component.css'

})

export class **AppComponent**{

title ='Welcome to Angular document'

}

In a view template url value has an './app.component.html',

<p>defult value:{{title}}</p>

<p> UpperCase: {{title|uppercase}}</p>

<p> lowerCase: {{title|lowercase}}</p>

**Currency Pipe**

Formats numbers as currency.

@**Component**({

selector: 'app-root',

standalone: true,

imports: [**RouterOutlet**,**FormsModule**,**CommonModule**],

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

styleUrl: './app.component.css'

})

export class **AppComponent**{

title ='Welcome to Angular document'

price = 200;

}

In a view template url value has an './app.component.html',

<p>defult value:{{title}}</p>

<p>Price without Formatting : {{price}}</p>

<p>Price currency Formatting : {{price|currency}}</p>

<p>Price currency India Formatting : {{price|currency:"INR"}}</p>

**Sharing Data between Components:**

Sharing one components data into multiple components by an data into view page

Example: From components data sharing into a List components

**Using Input and Output Decorator:**

**Example:**

Create an new components by using the command: ng g c from.

Create an new components by using the command: ng g c list.

In appcomponents

@**Component** ({

  selector: 'app-root',

  standalone: true,

  imports: [**RouterOutlet**,**FormsModule**,**CommonModule**,**FromComponent**,**ListComponent**],

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

  styleUrl: './app.component.css'

})

export class **AppComponent**{

  title = 'Hello world';

  carName:string[]=[];

**onCarAdded**(carName:string) {

      console.**log**(carName,'From APP')

      this.carName.**push**(carName);

  }

}

In a view template url value has an './app.component.html',

<div>

    <app-from *(carAdded)* = "onCarAdded($event)"></app-from> 🡪 this from components.

    <app-list *[carName]*="carName"></app-list> -->this listcomponents.

</div>

In an **FromComponent**

@**Component**({

  selector: 'app-from',

  imports: [**CommonModule**,**FormsModule**],

  templateUrl: './from.component.html',

  styleUrl: './from.component.css'

})

export class **FromComponent** {

  carName:string ='';

  @**Output**() **carAdded** = new **EventEmitter**<string>();

**onSubmit**(){

    console.**log**(this.carName);

*// sending the value in outside*

    this.**carAdded**.**emit**(this.carName);

    this.carName = '';

  }

}

In a view template url value has an './app.fromcomponent.html',

<form *(submit)*="onSubmit()">

    <label *for*="">Car </label>

    <input *type*="text" *[(ngModel)]* ="carName" *name* ="carName">

    <button *type* ="submit">Submit</button>

</form>

In **ListComponent**

import { CommonModule } from '@angular/common';

import { Component, Input } from '@angular/core';

@**Component**({

  selector: 'app-list',

  imports: [**CommonModule**],

  templateUrl: './list.component.html',

  styleUrl: './list.component.css'

})

export class **ListComponent** {

*// @Input() carName:string[] =['Ford','Ferrai'];*

   @**Input**() carName:string[] =[];

}

In a view template url value has an './app.listcomponent.html',

<div>

    <h2> Cars</h2>

    @if(carName.length>0){

    <ul>

        <li *\*ngFor* = "let car of carName">{{car}}</li>

    </ul>

    }@else {

         <p>no cars found</p>

    }

</div>

**Service:**

In **Angular**, a **Service** is a **reusable class** that provides functionality such as data fetching, business logic, or shared data between components.A **Service** in Angular is a class that **encapsulates reusable logic** and can be **injected into components** using **Dependency Injection (DI)**.

Create an new Services class by using command ng g s car

In services class:

@**Injectable**({

  providedIn: 'root'

})

export class **CarService** {

  constructor() { }

private carNamesubject = new **BehaviorSubject**<string[]>([]);

carNames$ = this.carNamesubject.**asObservable**();

**addCarName**(name :string){

*//will added an list components.*

  const currentNames =  this.carNamesubject.**getValue**();

  const updateNames = [...currentNames,name];

  this.carNamesubject.**next**(updateNames);

  }

}

In a view template url value has an './app.fromcomponent.html',

<form *(submit)*="onSubmit()">

    <label *for*="">Car </label>

    <input *type*="text" *[(ngModel)]* ="carName" *name* ="carName">

    <button *type* ="submit">Submit</button>

</form>

In an **FromComponent**

@**Component**({

  selector: 'app-from',

  imports: [**CommonModule**,**FormsModule**],

  templateUrl: './from.component.html',

  styleUrl: './from.component.css'

})

export class **FromComponent** {

  carName:string ='';

constructor(private carservice :**CarService**){

}

In an **ListComponent**

@**Component**({

  selector: 'app-list',

  imports: [**CommonModule**],

  templateUrl: './list.component.html',

  styleUrl: './list.component.css'

})

export class **ListComponent** implements **OnInit** {

  carNames:string[] = [];

  constructor(private carservice :**CarService**){

  }

**ngOnInit**(): void {

      this.carservice.carNames$.**subscribe**((names)=>{

        this.carNames = names;

      })

  }

}

In a view template url value has an './app.listcomponent.html',

<div>

    <h2> Cars</h2>

    @if(carName.length>0){

    <ul>

        <li *\*ngFor* = "let car of carNames">{{car}}</li>

    </ul>

    }@else {

         <p>no cars found</p>

    }

</div>

**Component Lifecycle Hooks**

Property Changes

yy

Created

Content Loaded

Destroyed

Content changed

View changed

ngAfterViewChecked

ngAfterContentInit

ngAfterViewInit

ngDoCheck

ngOnInit

ngOnChanges

ngOnDestroy

Component

View loaded

**NgOnChanges**:

ngOnChanges is called **before ngOnInit and whenever an @Input() property value changes**. It helps in detecting and responding to changes in parent-bound properties.

In from.component.ts

export class **FromComponent** implements **OnChanges** {

  @**Input**() items:any[] = [];

**ngOnChanges**(changes: **SimpleChanges**): void {

    if(changes['items']){

console.**log**('Items Property has been changed',changes['items']);

    }

  }

}

In add in appcomponents values.

export class **AppComponent** {

  itemList = [

    {id:1,name:'Apple'},

    {id:2,name:'Banana'},

    {id:3,name:'Grapes'},

    {id:4,name:'JackFruit'},

    {id:5,name:'Gauva'},

    {id:6,name:'CustedApple'},

  ]

}

In app.components.ts

<h2>Life of CycleHooks</h2>

<app-from *[items]* = "itemList"></app-from>

In View page of chromes folders

This value an console.log (value) -🡪 inspects values

**ngOnchangesFirstChange**

Whenever, Change the value a given Input format with an function process.

In app.components. add one more property’s

export class **AppComponent** {

  itemList = [

    {id:1,name:'Apple'},

    {id:2,name:'Banana'},

    {id:3,name:'Grapes'},

    {id:4,name:'JackFruit'},

    {id:5,name:'Gauva'},

    {id:6,name:'CustedApple'},

  ]

  title = "hello";

}

In a app.components.html values with two-way binding process

<h2>Life of CycleHooks</h2>

<input *type*="text" *name*="title" *[(ngModel)]* = "title">

<app-from *[items]* = "itemList" *[title]* = title></app-from> set property files

@**Component**({

  selector: 'app-from',

  imports: [**CommonModule**,**FormsModule**],

  templateUrl: './from.component.html',

  styleUrl: './from.component.css'

})

export class **FromComponent** implements **OnChanges** {

  @**Input**() items:any[] = [];

  @**Input**() title = '';

**ngOnChanges**(changes: **SimpleChanges**): void {

    if(changes['items']){

console.**log**('Items Property has been changed',changes['items']);

    }

    if(changes['title']){

      console.**log**('Items Property has been changed',changes['title']);

          }

  }

}

In view an html Console log value in process with an inspect seen the value

**ngOnInit().**

ngOnInit is a lifecycle hook in Angular that is called **once** after the component has been initialized. It is mainly used for initializing data, making API calls, or setting up component logic.

🔹 ngOnInit is called **after the constructor** and before the component is rendered.  
🔹 It runs **only once**, unlike ngOnChanges, which runs multiple times.

It will add an app.component.

@**Component**({

  selector: 'app-root',

  standalone: true,

  imports: [**RouterOutlet**,**FormsModule**,**CommonModule**,**FromComponent**,**ListComponent**],

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

  styleUrl: './app.component.css'

})

export class **AppComponent** implements **OnInit**{

  message = " ";

**ngOnInit**(): void {

    this.message = "Welcome an life cycle hooks class!"

  }

}

In app.componenthtml.

<h2>OnInit hooks</h2>

<p> Message : {{message}} </p>

This result in after page loaded

**ngDoCheck()**

ngDoCheck() is a lifecycle hook in Angular that is called during every change detection cycle. It allows developers to implement custom change detection logic beyond what Angular's default mechanisms provide.

In app.componenthtml

<h2>ngDoCheck hooks</h2>

<input *type* ="text" *name* = "title" *[(ngModel)]* = "title">

@**Component**({

  selector: 'app-root',

  standalone: true,

  imports: [**RouterOutlet**,**FormsModule**,**CommonModule**,**FromComponent**,**ListComponent**],

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

  styleUrl: './app.component.css'

})

export class **AppComponent** implements **DoCheck**{

  title = "Hello";

  pretitle = "Hello"

**ngDoCheck**(): void {

    if(this.title !== this.pretitle){ // by condition details check value by using an ngDocheck concept will be used.

      console.**log**("This title property changed value " + this.title);

      this.pretitle = this.title;

    }

  }

}

**ngAfterContentInit()**

ngAfterContentInit() is an Angular lifecycle hook that is called once after the component's content (such as <ng-content>) has been projected into the component. It runs only once after the first ngDoCheck().

**Routing main part of an angular**

Routing in Angular allows you to navigate between different views or pages within a single-page application (SPA). It is an essential part of building dynamic and interactive web applications. Angular provides a Router service that makes it easy to manage the navigation of different views based on user interactions or URLs.

Sample example:

Can you create a new project

Here, we create an three Component, ng generate component home, ng generate component about, ng generate component contact.

In this component will routing with each other component by using an

<router-outlet /> It will router-outlet mention in app.html

In app.routes.ts

App.routes values mention path values define in path specification in value

syntax:

{Path: ‘’, component: component name }

import { Routes } from '@angular/router';

import { HomeComponent } from './home/home.component';

import { AboutComponent } from './about/about.component';

import { ContactComponent } from './contact/contact.component';

export const routes: **Routes** = [

    {path:'home',**component**: **HomeComponent**},

    {path:'',**component**: **HomeComponent**},

    {path:'about',**component**: **AboutComponent**},

    {path:'contact',**component**: **ContactComponent**}

];

In view page screen in values has <http://localhost:4200/> with a path name.

Example : <http://localhost:4200/home>, http://localhost:4200/about, <http://localhost:4200/contact>,

**Redirect Route**

In an redirect route by using process same like an redirect route method process

Syntax : {path:'',redirectTo:"/about",pathMatch:'full'},

Redirect to represent in path call function, path match represent into an full value an path values.

import { Routes } from '@angular/router';

import { HomeComponent } from './home/home.component';

import { AboutComponent } from './about/about.component';

import { ContactComponent } from './contact/contact.component';

export const routes: **Routes** = [

    {path:'home',**component**: **HomeComponent**},

    {path:'',redirectTo:"/about",pathMatch:'full'},

    {path:'about',**component**: **AboutComponent**},

    {path:'contact',**component**: **ContactComponent**}

];

If the url path empty, the redirect into an mention in redirect To value path process.

**Navigation Links**

Navigation links is used a redirect a path values mention in an router link.

In app component add an nav syntax:

<router-outlet />

<nav>

    <a *routerLink* ="/home">Home</a> |

    <a *routerLink* ="/about">About</a> |

</nav>

In an appcomponent mention into an value a

import { Component } from '@angular/core';

import { RouterLink, RouterOutlet } from '@angular/router';

@**Component**({

  selector: 'app-root',

  standalone: true,

  imports: [**RouterOutlet**, **RouterLink**],

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

  styleUrl: './app.component.css'

})

export class **AppComponent** {

  title = 'angular-routing';

}

Add into a Router Link then only redirect a path values.

**Active Router Link**

Router active link used a highlight when doing the active of with an when click the values of link.

Activerouterlink.

app.component.css:

.active-link{

    color: coral;

}

In app.component.html details value a routeractivelink process.an app.routes.ts page

<router-outlet />

<nav>

    <a *routerLink* ="/home" *routerLinkActive*="active-link">Home</a> |

    <a *routerLink* ="/about" *routerLinkActive*="active-link">About</a> |

    <a *routerLink* ="/contact" *routerLinkActive*="active-link">Contact</a> |

</nav>

In app.component.ts will import a routeractivelink.

import { Component } from '@angular/core';

import { RouterLink, RouterLinkActive, RouterOutlet } from '@angular/router';

@**Component**({

  selector: 'app-root',

  standalone: true,

  imports: [**RouterOutlet**, **RouterLink**,**RouterLinkActive**],

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

  styleUrl: './app.component.css'

})

export class **AppComponent** {

  title = 'angular-routing';

}

When click an each component a link value based on change into without a page load get a values.

**Programmatic Navigation with Router Service**

Can call value of navigation by using an router service.

home.component in an html component.

<h1>About pages</h1>

<button *(click)* = navigateToContact()> Go to contact page </button>

import { Component } from '@angular/core';

import { Router } from '@angular/router'; -- import an router service for an implementation by using an service class

@**Component**({

  selector: 'app-home',

  imports: [],

  templateUrl: './home.component.html',

  styleUrl: './home.component.css'

})

export class **HomeComponent** {

  constructor(private router : **Router**){

  }

**navigateToAbout**(){

    this.router.**navigate**(['/about']);

  }

}

When click the button to redirect and navigate process with call main html page.

**Route Parameters**

This Route parameter will has process get value by using url values based details an Route details process with an process.

Routes.ts

import { Routes } from '@angular/router';

import { HomeComponent } from './home/home.component';

import { AboutComponent } from './about/about.component';

import { ContactComponent } from './contact/contact.component';

export const routes: **Routes** = [

    {path:'home',**component**: **HomeComponent**},

    {path:'',redirectTo:"/home",pathMatch:'full'},

    {path:'about/:id',**component**: **AboutComponent**},Here,Id will represent an passing parameter by using an :id

    {path:'contact',**component**: **ContactComponent**}

];

In an aboutcomponent.html

import { ActivatedRoute } from '@angular/router';

@**Component**({

  selector: 'app-about',

  imports: [],

  templateUrl: './about.component.html',

  styleUrl: './about.component.css'

})

export class **AboutComponent** implements **OnInit** {

  id!:string |null;

  constructor(private route : **ActivatedRoute**){

  }

// activatedRoute -> An service an class

**ngOnInit**() {

    this.id= this.route.snapshot.paramMap.**get**("id");

  }

}

In about.html.

<h1>About pages</h1>

<p>Id is {{id}}</p>

In an view page process

When an add in search details <http://localhost:4200/about/3> --> 3 Is Id.

**Lazy Loading Components**

Lazy loading is a technique where components are loaded only when they are needed, instead of loading them all at once. This helps improve the application's performance by reducing the initial load time.

Here, we can create a new components file values.

An routes.ts values.

import { Routes } from '@angular/router';

import { HomeComponent } from './home/home.component';

import { AboutComponent } from './about/about.component';

import { ContactComponent } from './contact/contact.component';

export const routes: **Routes** = [

    {path:'home',**component**: **HomeComponent**},

    {path:'',redirectTo:"/home",pathMatch:'full'},

    {path:'about/:id',**component**: **AboutComponent**},

    {path:'contact',**component**: **ContactComponent**},

    {path:'featureone',

**loadComponent** : ()=> import('./featureone/featureone.component').**then**(m => m.**FeatureoneComponent**)

    }

// m will represent into an module name

];

In seated of a component her using a loadComponet use a function value.

In app.component will using add one more link values

<router-outlet />

<nav>

    <a *routerLink* ="/home" *routerLinkActive*="active-link">Home</a> |

    <a *routerLink* ="/about" *routerLinkActive*="active-link">About</a> |

    <a *routerLink* ="/contact" *routerLinkActive*="active-link">Contact</a> |

    <a *routerLink* ="/featureone" *routerLinkActive*="active-link">featureone</a>

</nav>

**How Lazy Loading works?**

When a Lazy loading process with a function in build value process. Will working a production environment.

By using a command:

First stop an development server.

Then,

**ng build**

* It will create a new folder, build values and dist**.**

**ng install –g http-server (node command).**

* It a get install process with a run in the server this package will be helpful to us.

Find a dist. folder value in your projects folder.

Then move a Brower folder.

Then will http –server accessible a project.

**By using command has http-server –p 8080**

F:\AngularProject\angular-routing> ng build

F:\AngularProject\angular-routing\dist\angular-routing

F:\AngularProject\angular-routing> npm install -g http-server

F:\AngularProject\angular-routing\dist> cd .\angular-routing\

F:\AngularProject\angular-routing\dist\angular-routing> cd browser

F:\AngularProject\angular-routing\dist\angular-routing\browser> http-server -p 8080

By an following step use an your used project