**Angular Documentation**

**Angular** :Angular is a platform and framework for building single-page client applications using HTML and TypeScript.

It implements core and optional functionality as a set of TypeScript libraries that you import into your applications.

**Why Angular** : Angular helps build interactive and dynamic single page applications(SPA) with its compelling features including templating.

Two-way binding, modularization, Restful API handling, dependency injection and AJAX handling.

**Pre-requisites** :

* **Required following System configuration.**
* System with at least 8GB Ram and 100 GB HDD
* Node 12.x+
* Visual studio code
* **Required following technologies knowledge.**
* HTML
* CSS
* Javascript
* ES6
* TypeScript

**Project Setup :** Please follow the link for creating [Angular project](https://angular.io/guide/setup-local)

* Open Visual Studio code and open terminal
* Open your newly created project folder
* Then execute the following commands

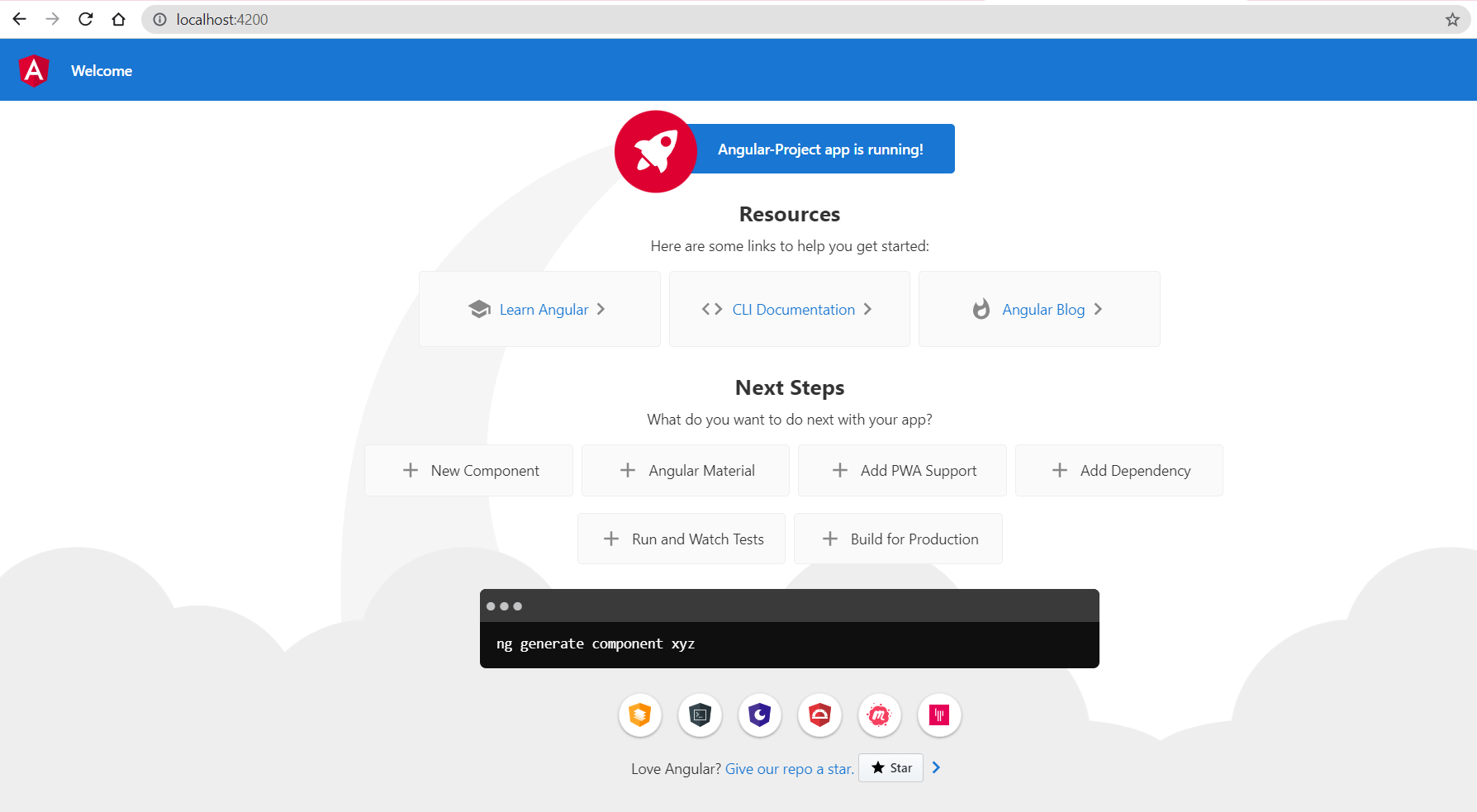
**Note** : my-app is a customized name and you can give the name whatever you want.

npm install -g @angular/cli  
ng new my-app

cd my-app

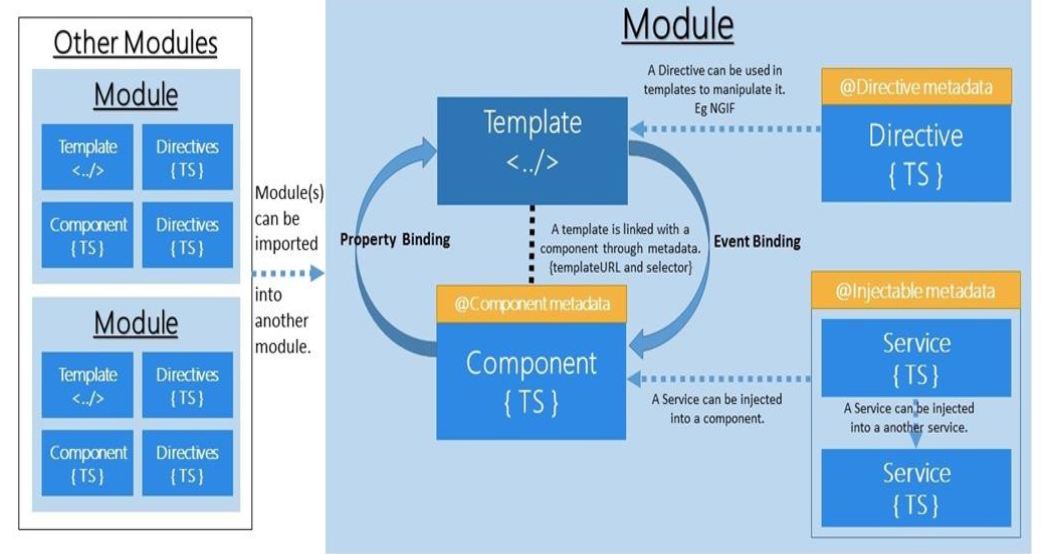
ng serve

**Note** : Some Operating Systems(OS) will not allow you to run the ng command for which you need to make some changes. Please follow the following [link](https://www.c-sharpcorner.com/article/how-to-fix-ps1-can-not-be-loaded-because-running-scripts-is-disabled-on-this-sys/).



**Structure or Architecture** : Angular is **a framework for building client applications in HTML and either JavaScript** or a language like Type Script that compiles to JavaScript.

The framework consists of several libraries, some of them core and some optional.



**Modules** : Modules are logical functionality. Modules can have components, services, directives, pipes.

* Regisert
* Forgot
* Signup
* Services

We need to import required modules from core packages.Every modules need to be defined by @NgModule.

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

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

@NgModule({

declarations: [],

imports: [

CommonModule

]

})

export class TodoModule { }

**Decorators** : Decorators are functions that will return functions and they are invoked at runtime. It will also allow us to invoke functions.

**Types of Decorators** :

* Class Decorators
* Property Decorators
* Method Decorators
* Parameters Deccorators

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

@Component({

selector: 'app-root',

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

template: '<h1>Welcome<h1>',

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

})

export class AppComponent {

title = 'Angular-Project';

@Input() data: string;

@Output() params : string;

}

**Component** : Component is a smaller functionality which can be reused multiple time in the application.

* Login
* Forgot-Password
* Authenticate
* Register

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

@Component({

selector: 'app-root',

template: '<h1>{{title}}<h1>',

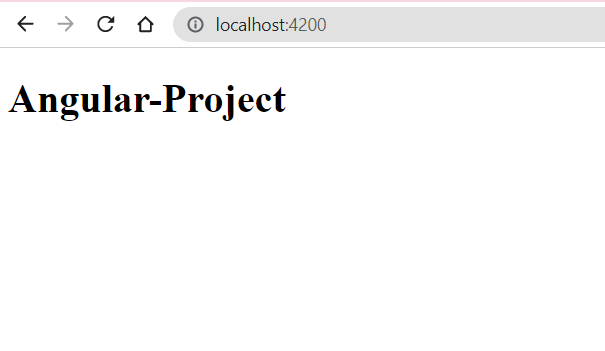
styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'Angular-Project';

}



**Directives** : Directives are the classes that add additional behavior to elements in your project.

* **Component Directives** : Every Angular application mush have atleast 1 component with its own templates and events attached.
* **Structural Directives** : It will update the structure of the view.
* ngFor
* ngIf
* ngSwitch
* **Attribute Directives** : It is used for Styling.
* ngStyle
* ngClass

**app.component.ts**

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

@Component({

selector: 'app-root',

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

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

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

})

export class AppComponent {

title = 'Angular-Project';

readMe() {

console.log("button Clicked");

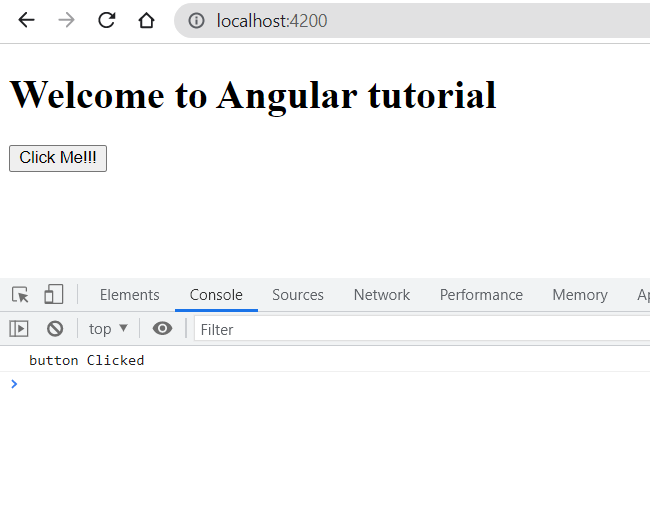
}

}

app.component.html

<h1>Welcome to Angular tutorial</h1>

<button (click)="readMe()">Click Me!!!</button>



**Data Binding** : Data binding means to bind the data(interacting) from view to controller and vice versa.

* **One way Data Binding** : Data flow from view to component or component to view.
* **Component to View**
* Interpolation
* Property Binding
* Style Binding
* Attribute Binding
* **View to Component**
* Event Binding
* **Two way Data Binding** : Data flows from view to component and back to component from the view.

**Interpolation** : It is a technique that allows the user to bind data from component to view. The data flows only in one-way(component to view).

Can be used for integer, String, objects arrays…..

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

@Component({

selector: 'app-root',

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

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

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

})

export class AppComponent {

title = 'Angular-Project';

subTitle = "It's fun to learn the interpolation";

}



**Property Binding** : It is a technique that allows the user to bind properties of elements from component to view.Data flows in only one-way(component to view).

Can be used for all the properties like innerHTML, src…..

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

@Component({

selector: 'app-root',

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

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

})

export class AppComponent {

title = 'Angular-Project';

colorVal = 'red';

}



**Attribute Binding** : It is a technique that allows the user to bind attributes of elements from component to view.Data flows only one-way(component to view).

Can be used for custom attributes and existing properties.

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

@Component({

selector: 'app-root',

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

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

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

})

export class AppComponent {

title = 'Angular-Project';

placeholderVal = "Enter Value";

hrefVal = "http://google.com";

}



**Event Binding** : It is a technique that allows the user to bind events of elements from view to component.Data flows only one-way(view to component).

Can be used for all available events.

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

@Component({

selector: 'app-root',

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

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

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

})

export class AppComponent {

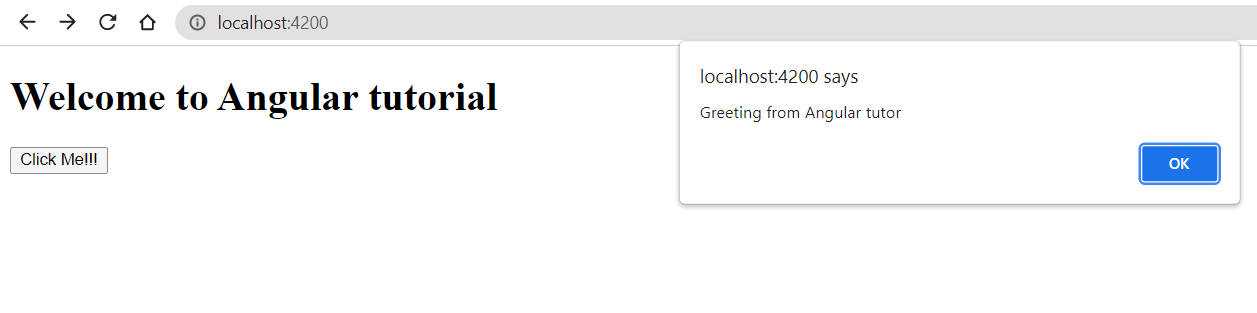
title = 'Angular-Project';

showAlert() {

alert("Greeting from Angular tutor");

}

}



**Two way Data Binding** : It is a technique that allows the user to bind events of elements from view to component and vice versa.Data flows in both way(view to component or vice versa).

Its a combination of property binding and event binding. It make use of ngModel to bind the data.

**app.component.html**

<h1>Welcome to Angular tutorial</h1>

<!-- <button (click)="readMe()">Click Me!!!</button> -->

<!-- <div>{{subTitle}}</div> -->

<!-- <div [ngStyle]="{color:colorVal}">This is example of Property Bindings</div> -->

<!-- <input [placeholder]="placeholderVal">

<a [href]="hrefVal">Google</a> -->

<!-- <button (click)="showAlert();">Click Me!!!</button> -->

<input (ngModel)="firstname" />

<div>{{firstname}}</div>

<router-outlet></router-outlet>



**Pipes** : Pipes are used to transform the data, it will take data input and convert/transform into a desired format. Can be applied to any view and to any data inputs.

* Built in Pipes
* Lowercase
* Uppercase
* Currency
* Date
* Percent
* JSON

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

@Component({

selector: 'app-root',

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

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

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

})

export class AppComponent {

title = 'Angular-Project';

user ={

userId : 20,

firstName : 'Rahul',

lastName : 'Chavan',

DOB : '19/11/2021',

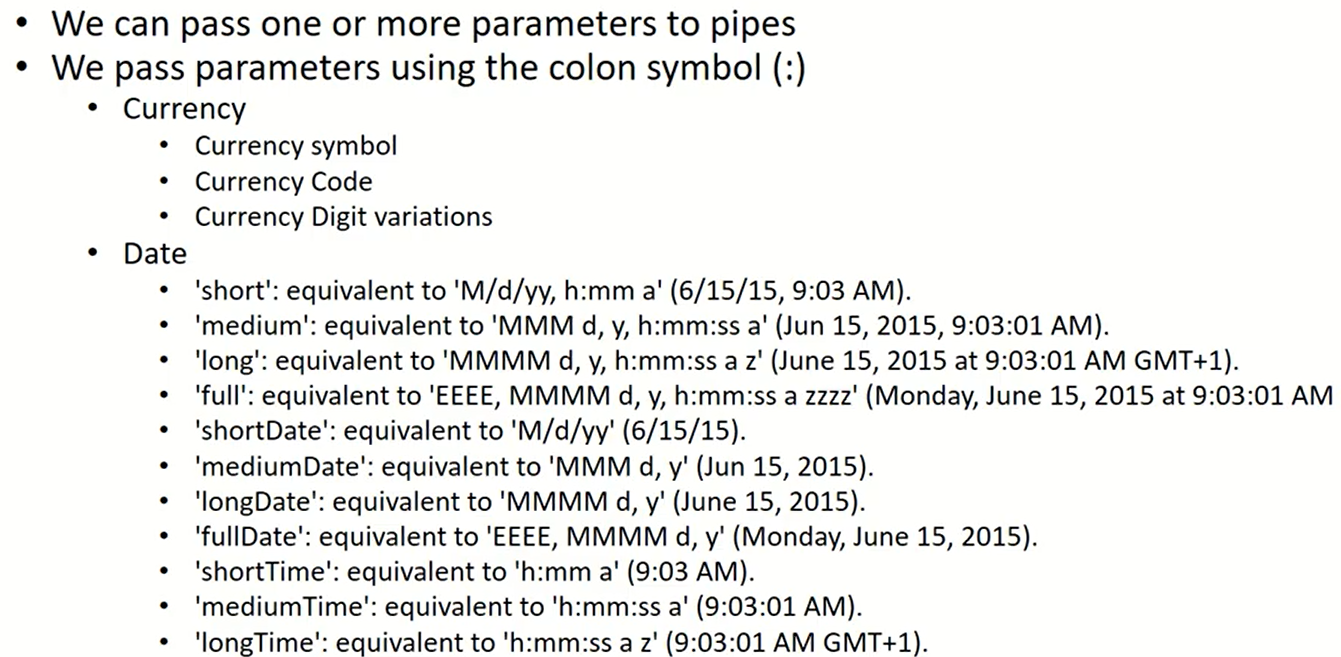
salary : 5000

};

}



* Parametrized Pipes
* This is used to pass one or more parameters to pipes.



<h1>Welcome to Angular tutorial</h1>

<div>

{{user.firstName | uppercase}}, {{user.lastName | lowercase}}

</div>

<div>

{{user.DOB | date:'long'}}

</div>

<div>

{{user.salary | currency: 'INR'}}

</div>

<router-outlet></router-outlet>



* Chaining Pipes
* By using this we can connect multiple pipes to a data input.

<h1>Welcome to Angular tutorial</h1>

<div>

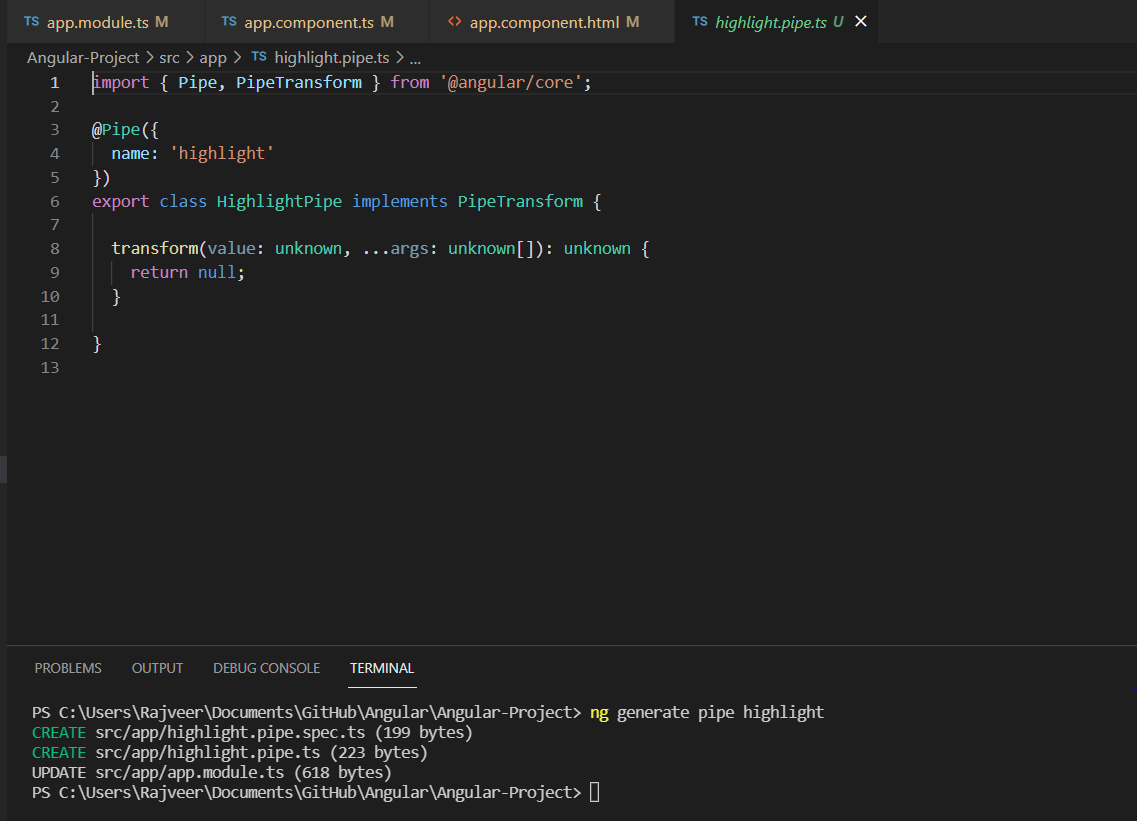
{{user.DOB | date:'long' | uppercase}}

</div>

<router-outlet></router-outlet>



* Custom Pipes
* By making use of this we can create our own custom pipes for various data formatting.

****

**Routing :** Routing is a mechanism used by Angular framework to manage the “paths” and “routes” of the application. It helps in navigation between various views in application.

Any Angular application comes with “Router” module which has everything we need to design, develop and implement routes and navigation link.

Various types of routes in Angular.

* Routing Strategies
* Base Href
* Routing Module
* Routing Outlet
* Configuring Routes
* Parametrized Routes
* Query params in Routes
* Redirecting Routes
* Wildcard Routes
* Child Routes
* Lazy loading
* Route Guards

**Routing Strategies** : This will help in understanding and planning what will be our routing strategy.

There are 2 types of routing strategy.

* PathLocationStrategy
* HashLocationStrategy

**Note** : By default Angular uses PathLocationStrategy.

**app.module.ts**

import { BrowserModule } from '@angular/platform-browser';

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

import { AppRoutingModule } from './app-routing.module';

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

import { TodoModule } from './todo/todo.module';

import { MouseHoverEffectPipe } from './mouse-hover-effect.pipe';

import { HighlightPipe } from './highlight.pipe';

import { HashLocationStrategy, LocationStrategy } from '@angular/common';

@NgModule({

declarations: [

AppComponent,

MouseHoverEffectPipe,

HighlightPipe

],

imports: [

BrowserModule,

AppRoutingModule,

TodoModule

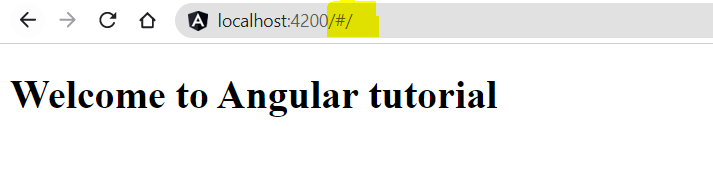
],

providers: [{provide: LocationStrategy, useClass : HashLocationStrategy}],

bootstrap: [AppComponent]

})

export class AppModule { }



**Base HREF** : Every Angular application has mandatory base href and by default it is set to “/” to the root folder.

**Routing Module** : Routing module is a placeholder for configuring all routes and navigation in one module.

**Routing Outlet** : It help us to define where the output should be displayed and it can be specified at App Module or individual modules.

Every Angular application must have router outlet. We can more then 1 router outlet in application.

**Configure Routes** : it is used to configure routes to redirect route for various paths.

* Path
* Component
* redirectTo
* Children

**Parametrized Routes** : By using this we can configure and send parameters to our routes.It is also used to read the values in the component class and process the parameters.

**Query parameters** : By using this we can configure query parameters in the URL routes.

**Redirecting Routes** : By making use of this we can redirect from one route to another route by using redirectTo in our routes array.

**Wildcard Routes** : Wild card intercepts any invalid URLs in our application. When no matching routes are found in the routes array then the router does not know where to go and hence results in console error.

Usually a component name PageNotFound is mapped as best practice.

**Child Routes** : we can configure child routes to create more meaningful URL Segments.

The child routes array will follow syntax and array concept as similar to defining the routes array.

**Lazy Loading** : This one of the main important concepts of Routing. Where it will make sure to load all the immediately required pages.

Lazy loading helps keep initial bundle sizes smaller, which in turn helps decrease load time.

**Route Guards** : By using this to prevent users from navigating to parts of an app without authorization.Its a secure route path.

In most of the cases the routes and screens are protected behind a good authentication system.It turns to true or false based on custom logic and functionality.

**Observer** : When we create an Observable, we keep track of the observable using an Observer.Observer will continuously listen to Observable.

Observer has 3 methods that we can use

* next()
* error()
* complete()

**Subscribe** : If we don't have a subscriber then it’s a waste of having Observable.We always subscribe to an observable in order to process the data.

We can also have multiple subscribers for any given observable.

**Dependency Injection** : It's an important application design pattern and has the ability to add the functionality of components at runtime.

It supplies data to a component from an injectable service class, defined in its own file.

Angular has its own Dependency Injection framework which is typically used in the design of Angular applications to increase their efficiency and modularity.

**Services** : Services allow us to create reusable common shared functionality between various modules and components. It’s a singleton.Services are injected into application using Dependency Injection mechanism.

Services are commonly used for making HTTP requests to our endpoints APIs to requests and receive the response.

A service can have a value, method or a combination of both.

**HttpClient-Get** : Making API calls to retrieve data is a **GET** method call, we will make use of following options to pass parameters like **header**, **params**, **responseType**, **withCredentials**.

**HttpClient-Post** : Making API calls to submit the data is referred to as a **POST** call,we can also pass various parameters as options to the **POST** call. We will make use of parameters like **header**, **params**, **responseType**, **withCredentials**.

**HttpClient-Put** : Making API calls to submit to update the existing data by using **PUT**.we can also pass various parameters as options to **PUT** call. We will make use of parameters like **header**, **params**, **responseType**, **withCredentials**.

**HttpClient-Delete** : Making API calls to submit to delta the existing data by using **DELETE**. We can also pass various parameters as options to **DELETE** call. We will make use of parameters like **header**, **params**, **responseType**, **withCredentials**.

**HttpClient-Headers** : We can pass data authorization and content type using this header. HttpHeader consists of below 7 methods.

* Append
* Has
* Get
* Keys
* getAll
* Set
* Delete

HttpClient-Params : It’s similar to the Header but we can pass data in forms of parameters and it has 8 different methods to pass the data.

* Append
* Has
* Get
* Keys
* getAll
* Set
* Delete
* toString