Contents

[Introduction 3](#_Toc28257113)

[Installation Guides 3](#_Toc28257114)

[Required Tools 3](#_Toc28257115)

[Installing Node Js (npm) & Visual Studio Code 3](#_Toc28257116)

[Update Environment 4](#_Toc28257117)

[Hello World App 6](#_Toc28257118)

[Code scaffolding 6](#_Toc28257119)

[Build 6](#_Toc28257120)

[Running unit tests and end-to-end tests 7](#_Toc28257121)

[File-Structure 7](#_Toc28257122)

[Angular Architecture Summary 7](#_Toc28257123)

[Generic Workflow 7](#_Toc28257124)

[File-Structure Details 7](#_Toc28257125)

[App 7](#_Toc28257126)

[Assets: To save your images, js files in this folder. 8](#_Toc28257127)

[Environment: 8](#_Toc28257128)

[favicon.ico: 8](#_Toc28257129)

[index.html: 8](#_Toc28257130)

[main.ts 8](#_Toc28257131)

[polyfill.ts 9](#_Toc28257132)

[styles.css 9](#_Toc28257133)

[test.ts 9](#_Toc28257134)

[tsconfig.app.json 9](#_Toc28257135)

[tsconfig.spec.json 9](#_Toc28257136)

[Component 9](#_Toc28257137)

[Interpolation 10](#_Toc28257138)

[Property Binding 10](#_Toc28257139)

[Class Binding 10](#_Toc28257140)

[Style Binding 11](#_Toc28257141)

[Event Binding 11](#_Toc28257142)

[Template Reference Variables 11](#_Toc28257143)

[Two Way Binding 12](#_Toc28257144)

[Structural Directives 12](#_Toc28257145)

[ngIf 12](#_Toc28257146)

[ngSwitch 13](#_Toc28257147)

[ngFor 13](#_Toc28257148)

[Component Interaction 13](#_Toc28257149)

[Display the data test.component.ts 14](#_Toc28257150)

[Sending data from Child to Parent. 14](#_Toc28257151)

[Pipes 14](#_Toc28257152)

[Services 15](#_Toc28257153)

[Dependency Injection 15](#_Toc28257154)

[Dependency Injector as Framework 15](#_Toc28257155)

[HTTP, Observables & RxJS 17](#_Toc28257156)

[Routing and Navigation 18](#_Toc28257157)

[Wildcard Route and Redirecting Routes 20](#_Toc28257158)

[Route Parameters 20](#_Toc28257159)

[Passing parameter to URI 21](#_Toc28257160)

[Read parameter from URI 22](#_Toc28257161)

[paramMap Observable 22](#_Toc28257162)

[Optional Route Parameters 23](#_Toc28257163)

[Relative Navigation 24](#_Toc28257164)

[Child Routes 24](#_Toc28257165)

[Angular Material 26](#_Toc28257166)

Angular

# Introduction

Angular is a complete front end framework with a wide range of features

* Strong Module System
* Strong Component System
* Forms Handling
* Routing System
* Dependency Injection
* HTTP Requests

## Installation Guides

### Required Tools

* Node v8+ for npm
* Visual Studio Code - Latest Version
* Java 8+

### Installing Node Js (npm) & Visual Studio Code

* Step 01 - Installing NodeJs and NPM - Node Package Manager
* Step 02 - Installing Visual Studio Code - Front End Java Script Editor

|  |
| --- |
| Download the MSI of NodeJS and install. Run these below command as necessary to check.    $ node –v # Node version  $ npm –v # NPM version    $ npm install -g @angular/cli # Install Angular Cli  $ ng update [PackageName] //update individual package  $ ng update all //update all |
| $ ng version |

## Update Environment

In order to update the angular-cli package installed globally in your system, you need to run:

npm uninstall -g angular-cli

npm cache clean or npm cache verify #(if npm > 5)

npm install -g @angular/cli@latest

Depending on your system, you may need to prefix the above commands with sudo.

Also, most likely you want to also update your local project version, because inside your project directory it will be selected with higher priority than the global one:

rm -rf node\_modules

npm uninstall --save-dev angular-cli

npm install --save-dev @angular/cli@latest

npm install

Thanks grizzm0 for pointing this out on [GitHub](https://github.com/angular/angular-cli/issues/4391#issuecomment-277199786).

After updating your CLI, you probably want to [update your Angular version](https://update.angular.io/) too.

**Note**: if you are updating to Angular CLI 6+ from an older version, you might need to read [this](https://stackoverflow.com/questions/51926816/ng-build-prod-does-not-minify-uglify-remove-comments-since-angular-cli-6/51926817).

**Edit**: In addition, if you were still on a 1.x version of the cli, you need to convert your angular-cli.json to angular.json, which you can do with the following command:

ng update @angular/cli --from=1.7.4 --migrate-only

(check [this](https://stackoverflow.com/questions/50232874/angular-6-migration-angular-cli-json-to-angular-json) for more details).

Other Working Solution

After upgrading to Angular 8

problem may occur

ng update --all

Using package manager: 'npm'

Collecting installed dependencies...

Found 58 dependencies.

Package "@angular/compiler-cli" has an incompatible peer dependency to "typescript" (requires ">=3.4 <3.5", would install "3.5.2")

Package "@angular-devkit/build-angular" has an incompatible peer dependency to "typescript" (requires ">=3.1 < 3.5", would install "3.5.2")

Package "angular2-datatable" has an incompatible peer dependency to "@angular/common" (requires "^2.0.0" (extended), would install "8.0.1").

Package "@angular/http" has an incompatible peer dependency to "@angular/core" (requires "7.2.15", would install "8.0.1")

Package "angular2-datatable" has an incompatible peer dependency to "@angular/core" (requires "^2.0.0" (extended), would install "8.0.1").

Package "angular2-datatable" has an incompatible peer dependency to "@angular/platform-browser" (requires "^2.0.0" (extended), would install "8.0.1").

Package "@angular/compiler-cli" has an incompatible peer dependency to "typescript" (requires ">=3.4 <3.5", would install "3.5.2").

Package "angular2-datatable" has an incompatible peer dependency to "rxjs" (requires "^5.0.0-beta.12", would install "6.5.2").

Incompatible peer dependencies found. See above

Solution for these error

ng update --all --force

Then error

ERROR in The Angular Compiler requires TypeScript >=3.4.0 and <3.5.0 but 3.5.2 was found instead.

npm install typescript@">=3.4.0 <3.5.0" --save-dev --save-exact

Just to be sure run the following steps:

npm uninstall -g @angular/cli

npm cache verify

npm install -g @angular/cli@latest

Then in your Local project package:

rm -rf node\_modules dist

npm install --save-dev @angular/cli@latest

npm i

ng update @angular/cli

ng update @angular/core

will fix the issue

## Hello World App

|  |
| --- |
| Create a folder C:\Users\GTech\angular-project. This can be anywhere in the drive. Use VScode IDE internal terminal for easier operation, which can be found at Menu>View>Terminal  $ cd angular-project  $ ng new hello-world  $ cd hello-world  $ ng serve –-open # To compile the code  Browse <http://localhost:4200/> from browser. For changing the port,  $ ng serve --port 8080 # Temporary way  ?? Permanent way?? |

## Code scaffolding

|  |
| --- |
| Run ng generate component component-name to generate a new component. You can also use ng generate directive|pipe|service|class|guard|interface|enum|module |

## Build

|  |
| --- |
| Run ng build to build the project. The build artifacts will be stored in the dist/ directory. Use the --prod flag for a production build. In the index.html: change the base href information as per the location of the directory again root. Example:  <base href="/material-demo/">  From, CLI It is to set base tag href to /myUrl/ in your index.html  $ ng build --base-href /myUrl/ |

## Running unit tests and end-to-end tests

|  |
| --- |
| Run ng test to execute the unit tests via [Karma](https://karma-runner.github.io/).  Run ng e2e to execute the end-to-end tests via [Protractor](http://www.protractortest.org/). |

## File-Structure

The angular-app/ folder has the following folder structure:

* e2e/:end to end test folder. Mainly e2e is used for integration testing and helps ensure the application works fine.
* node\_modules/: The npm package installed is node\_modules. You can open the folder and see the packages available.
* src/: This folder is where we will work on the project using Angular. Inside src/ you will app/ folder created during the project setup and holds all the required files required for the project.

The angular7-app/ folder has the following file structure:

* angular.json: It basically holds the project name, version of cli, etc.
* .editorconfig: This is the config file for the editor.
* .gitignore: A .gitignore file should be committed into the repository, in order to share the ignore rules with any other users that clone the repository.
* package.json: The package.json file tells which libraries will be installed into node\_modules when you run npm install. In case you need to add more libraries, you can add those over here and run the npm install command.
* tsconfig.json: This basically contains the compiler options required during compilation.
* tslint.json: This is the config file with rules to be considered while compiling.

The src/ folder is the main folder, which internally has a different file structure.

## Angular Architecture Summary

* Angular App – One or more modules
* Module – One or more components and services
* Components – HTML + Class
* Services – Business logic
* Modules interest and ultimately render the view in the browser

## Generic Workflow

* In the root folder > src > main.ts in the index file to work start. It initiates the root app > app.module.ts which then initiates other root components like app.component.ts contains the logic for the templates of app.component.html, app.component.css for web view.

## File-Structure Details

### App

* This is the main root module. It has several different libraries, which are imported. The names itself explain the usage of the libraries. They are imported and saved into variables such as declarations, imports, providers, and bootstrap. We can see app-routing.module is also added. This is because we had selected routing at the start of the installation. The module is added by @angular/cli.
* @NgModule is imported from @angular/core and it has object with following properties:
  + declarations: In declarations, the reference to the components is stored. The Appcomponent is the default component that is created whenever a new project is initiated.
  + imports: This will have the modules imported as shown above. At present, BrowserModule is part of the imports which is imported from @angular/platform-browser. There is also routing module added AppRoutingModule.
  + providers: This will have reference to the services created.
  + bootstrap: This has reference to the default component created, i.e., AppComponent.
  + app.component.css: You can write your css over here.
  + app.component.html: This is the default html code currently available with the project creation.
  + app.component.spec.ts: These are automatically generated files which contain unit tests for source component.
  + app.component.ts: The class for the component is defined over here. You can do the processing of the html structure in the .ts file. The processing will include activities such as connecting to the database, interacting with other components, routing, services, etc.
  + app-routing.module.ts: This file will deal with the routing required for your project. It is connected with the main module, i.e., app.module.ts.

### Assets: To save your images, js files in this folder.

### Environment:

This folder has details for the production or the dev environment. The folder contains two files.

* environment.prod.ts
* environment.ts

Both the files have details of whether the final file should be compiled in the production environment or the dev environment.

favicon.ico: This is a file that is usually found in the root directory of a website.

### index.html:

The body has <app-root></app-root>. This is the selector which is used in app.component.ts file and will display the details from app.component.html file.

### main.ts

main.ts is the file from where we start our project development. It starts with importing the basic module which we need. The platformBrowserDynamic().bootstrapModule(AppModule) has the parent module reference AppModule. Hence, when it executes in the browser, the file is called index.html. Index.html internally refers to main.ts which calls the parent module. When AppModule is called, it calls app.module.ts which further calls the AppComponent based on the bootstrap. In app.component.ts, there is a selector: app-root which is used in the index.html file. This will display the contents present in app.component.html

polyfill.ts This is mainly used for backward compatibility.

styles.css This is the style file required for the project.

test.ts Here, the unit test cases for testing the project will be handled.

tsconfig.app.json This is used during compilation, it has the config details that need to be used to run the application.

tsconfig.spec.json This helps maintain the details for testing.

# Component

|  |  |
| --- | --- |
|  | Component consists of three parts. Class contains the code with controls the Templates View and HTMLs. On the contrary, Metadata contains information to inform Angular whether it’s a class or regular component. Metadata is defined using Decorator which is a feature in Typescript, which provides information which class is attached to it. |

$ ng generate component test #Create new component

Three ways to specify a selector.

1. After the new component generation, mention the newly generated selector tag (here, app-test) in test.component.ts and add it to the root component html (here, app.component.html) as a custom HTML tag, which will be later initiated by the root component selector (here, app-root) which is mentioned in index.html
2. In the new component (here, test.component.ts) add a dot(.) in front of the selector value to represent it as CSS class attribute, which can be added to any standard HTML tag to app.component.html
3. In the new component (here, test.component.ts) wrap the value of the component selector with third bracket ie [] and add the value as attribute directly to any standard HTML tag to app.component.html

Component Metadata also has templateUrl which shows the View part. Here, we also can specify the template in line. By the changing the decorator from templateUrl to template, we can write any standard HTML tags within the two single quotation mark. In case of multiple lines of tags, remove the single quote sign and wrap the content with back ticks (ie `)

Same way, we can specify use styles instead of styleUrls in the array and with back ticks to add CSS codes.

# Interpolation

This is tactic to bind a data to the template from component class section. Define any variable in this section, add the variable to the templates between two curly braces. Example,

|  |  |
| --- | --- |
| **Class Section**  public name = “Sujan”; | **Templates Section**  <h1>Hello {{name}}</h1> |

Interpolation can perform arithmetic operation, string concatenation, can call predefined and custom method, but it has some limitation. It can’t assign value to binding (ie: {{a=2+2}}). It’s also can’t access global JS variable such as, {{window.location.href}} But if we assign new property to the class, then we can bind the value to the template. **Important Note**: Interpolation works only with String.

# Property Binding

Attributes & Properties are not the same. Attributes is defined by the HTML, whereas Properties are defined by the DOM (Document Object Model). Attributes initialize DOM properties and then they are done. Attribute values cannot change once they are initialized. Property values however can change.

Example:

|  |  |
| --- | --- |
| **Class Section**  public myID = “testID”; | **Templates Section**  <input [id]=”myId” type=”text” value=”Sujan”> |

# Class Binding

Useful for dynamically add or remove classes from HTML elements.

There are three ways to do Class Binding.

|  |  |  |
| --- | --- | --- |
| **Class Section**  Public successClass = "text-success"; | **Templates Section**  <h2 [class]="successClass">CodeEvolution</h2> | **Styles Section**  .text-success{        color: green;  } |
| public hasError = true; | <h2 [class.text-danger]="hasError">CodeEvolution</h2> | .text-danger{        color: red;      } |
| public hasError = false;    public isSpecial = true;    public messageClasses = {      "text-success": !this.hasError,      "text-danger" : this.hasError,      "text-special" : this.isSpecial    } | <h2 [ngClass]="messageClasses">CodeEvolution</h2> | .text-danger{        color: red;      }      .text-special{        font-style: italic;      } |

# Style Binding

|  |  |  |
| --- | --- | --- |
| **Class Section** | **Templates Section** | **Styles Section** |
|  | <h2 [style.color]="'orange'">Style Binding 1</h2> |  |
| public hasError = false; | <h2 [style.color]="hasError ? 'red' : 'green'">Style Binding 2</h2> |  |
| public highlighColor = "orange"; | <h2 [style.color]="highlightColor">Style Binding 3</h2> |  |
| public titleStyles = {      "color": "blue",      "fontStyle" : "italic"    } | <h2 [ngStyle]="titleStyles">Style Binding 4</h2> |  |

# Event Binding

|  |  |  |
| --- | --- | --- |
| **Class Section** | **Templates Section** | **Styles Section** |
| onClick(){      console.log("Welcome to CodeEvolution");      this.greeting = "Welcome to CodeEvolution";    } | <button (click)="onClick()">Greet1</button> {{greeting}} |  |
| onClick1(event){      console.log(event.type);  }  //We can explore the event method from the console as well for rest | <button (click)="onClick1($event)">Greet2</button> |  |
| public greeting = ""; | <button (click)="greeting='Welcome Sujan'">Greet3</button>{{greeting}} |  |

# Template Reference Variables

|  |  |  |
| --- | --- | --- |
| **Class Section** | **Templates Section** | **Styles Section** |
| onClick(value){      console.log(value);    } | <input #myInput type="text" />    <button (click)="onClick(myInput.value)">Log</button> |  |

# Two Way Binding

It requires forms module to be included in App.module.ts,

import { FormsModule } from '@angular/forms';

then add property FormsModule to imports section. Then in the test.component.ts, add these below codes.

|  |  |  |
| --- | --- | --- |
| **Class Section** | **Templates Section** | **Styles Section** |
| public name = ""; | <input [(ngModel)]="name" type="text" /> {{name}} |  |

# Structural Directives

Add or remove HTML elements form the DOM.

## ngIf

|  |  |  |
| --- | --- | --- |
| **Class Section** | **Templates Section** | **Styles Section** |
|  | <h2 \*ngIf="true">This is the first block</h2> |  |
| public displayName = true; | <h2 \*ngIf="displayName">This is the Second block</h2> |  |
| public displayName = true; | <h2 \*ngIf="displayName; else elseBlock">This is the Third block</h2>    <ng-template #elseBlock>      <h2>This is the else block</h2>    </ng-template> |  |
| public displayName = true; | <h2 \*ngIf="displayName; then thenBlock; else elseBlock">This is the Fourth block</h2>    <ng-template #thenBlock>      <h2>This is the then block</h2>    </ng-template>    <ng-template #elseBlock>      <h2>This is the else block</h2>    </ng-template> |  |

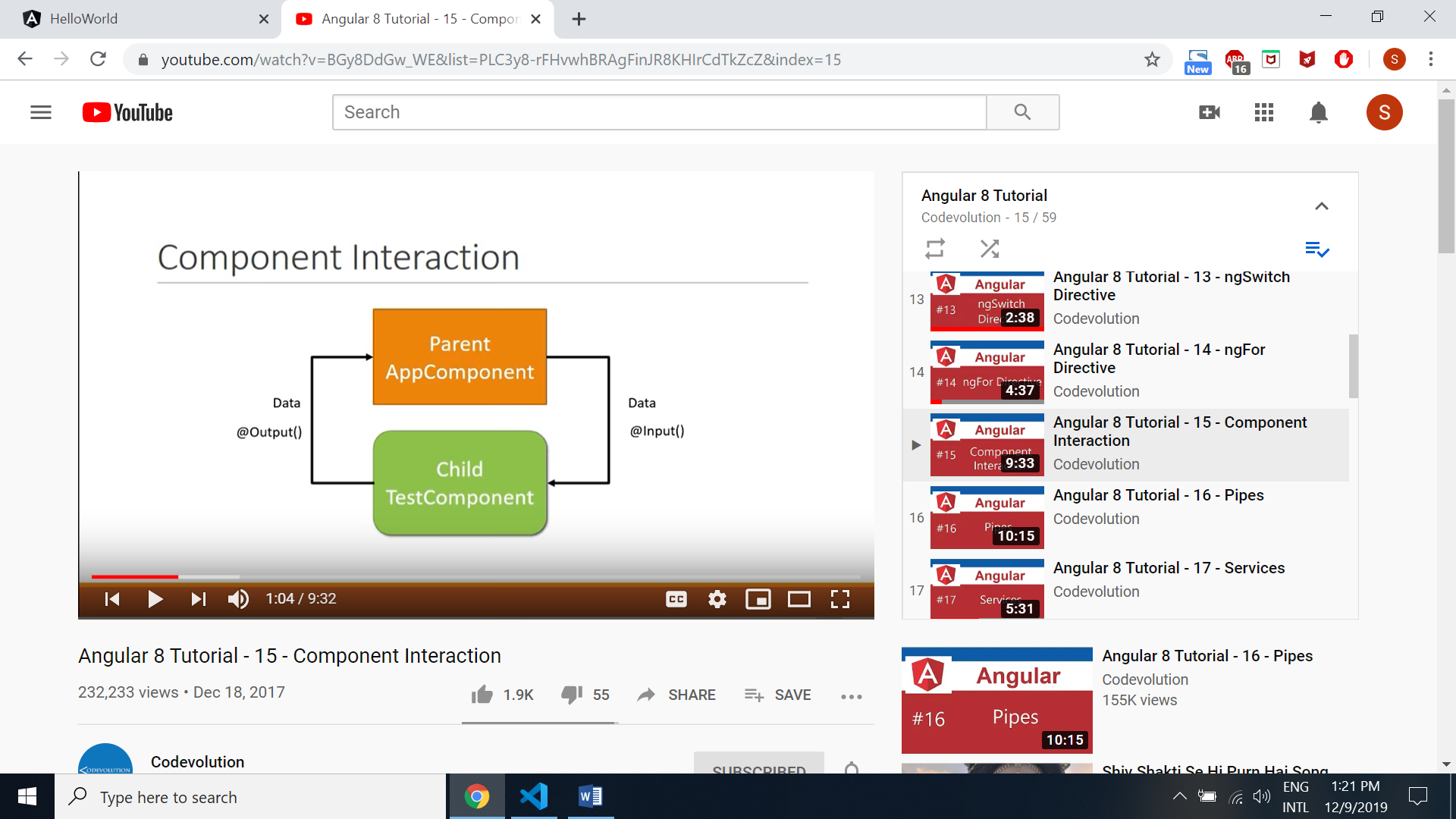
## ngSwitch

|  |  |  |
| --- | --- | --- |
| **Class Section** | **Templates Section** | **Styles Section** |
| public color = "red1"; | <div [ngSwitch]="color">      <div \*ngSwitchCase="'red'">Red Color</div>      <div \*ngSwitchCase="'blue'">Blue Color</div>      <div \*ngSwitchCase="'green'">Green Color</div>      <div \*ngSwitchDefault>Default Color</div>    </div> |  |

## ngFor

|  |  |  |
| --- | --- | --- |
| **Class Section** | **Templates Section** | **Styles Section** |
| public colors = ["red", "blue", "green", "yellow"]; | <div \*ngFor="let color of colors">      <h2>{{color}}</h2>    </div> |  |
| public colors = ["red", "blue", "green", "yellow"]; | <div \*ngFor="let color of colors; index as i">      <h2>{{i}}{{color}}</h2>    </div> | //Note  Here index is a keyword for indexing. There are some other keywords as well like even, odd, first, last |

# Component Interaction



We need to modify add app.component.ts by declaring a name property in the metadata section.

public name = "Sujan"; //Data to be transferred

Then bind this property to app.component.html by assigning to a variable.

<app-test [parentData] = "name"></app-test>

## Display the data test.component.ts

Import Input class to test.component.ts

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

There are two ways sending data from Parent to Child.

|  |  |
| --- | --- |
| **Class Section** | **Templates Section** |
| @Input() public parentData; | <h2>{{"Hello " + parentData}}</h2> |
| @Input('parentData') public name; | <h1>{{"Hello " + name}}</h1> |

### Sending data from Child to Parent.

Child component doesn’t have information on parent’s selector. Data has to be transferred via event emitter.

Import necessary classes to test.component.ts

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

|  |  |
| --- | --- |
| **Class Section** | **Templates Section** |
| @Output() public childEvent = new EventEmitter();  fireEvent(){      this.childEvent.emit ("Feedback from test component");    } | <button (click)="fireEvent()" >Send Event</button> |

In the app.component.ts, declare new property for outbound message, public message = "";

Capture and show the data to app.component.html

<app-test (childEvent) = "message=$event" [parentData] = "name"></app-test>

<h2>{{message}}</h2>

# Pipes

|  |  |
| --- | --- |
| **Class Section** | **Templates Section** |
| public name = "Sujan";    public message = "Welcome to Angular";    public person = {      "firstName" : "John",      "lastName" : "Wick"    }; | <h1>{{name}}</h1>      <h1>{{name | lowercase}}</h1>      <h1>{{name | uppercase}}</h1>      <h1>{{message | titlecase}}</h1>      <h1>{{name | slice:3}}</h1>      <h1>{{name | slice:1:3}}</h1>      <h1>{{person | json}}</h1> |

For more pipes, <https://angular.io/api?type=pipe>

# Services

Service is a class with a specific purpose. It used for:

1. Share data
2. Implement application logic (such as calculation)
3. External Interaction (such as database)

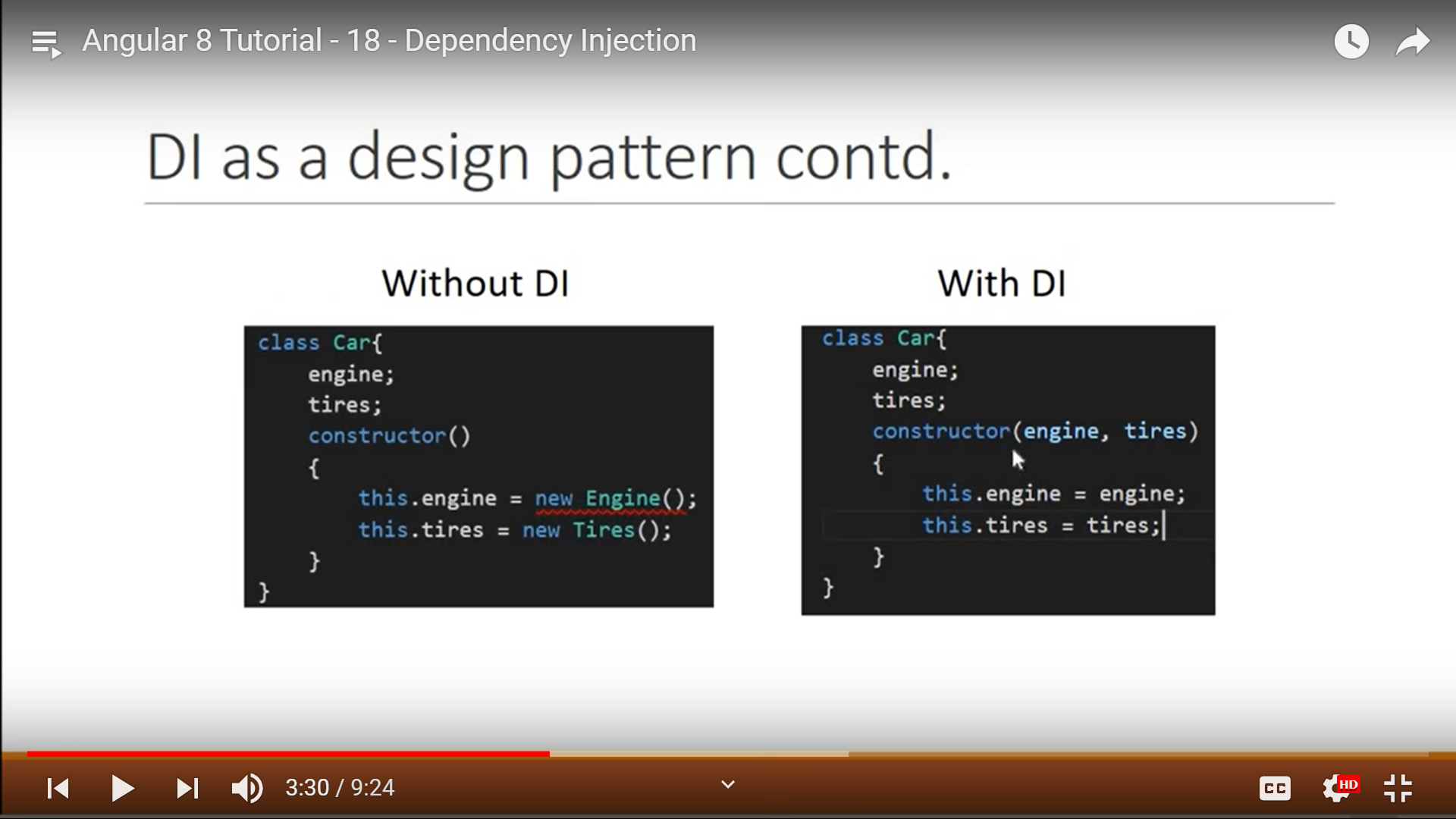
Naming convention: .service.ts

CLI command for generating new service.

$ ng generate service employee

## Dependency Injection

Dependency Injection is a coding pattern in which a class receives its dependency from the external sources, rather than creating them itself.



## Dependency Injector as Framework

1. Define the Service class which will be the injector
2. Register the injector. Better to register the service at modular level
3. Declare the injector as dependency to other required class(es)

*Step 1: Define the Service class which will be the injector, here it is data source.*

Create the Service, $ ng generate service employee

Add array as data source.

getEmployees(){

    return [

      {"id": 1, "name" : "First Employee", "age": 21},

      {"id": 2, "name" : "Second Employee", "age": 31},

      {"id": 3, "name" : "Third Employee", "age": 41}

    ];

  }

*Step 2: Register the injector to app.module.ts*

Usually Angular CLI will import all necessary components automatically while creating components. Below is for reference. But service path needs to be imported.

import { EmployeeListComponent } from './employee-list/employee-list.component';

import { EmployeeDetailComponent } from './employee-detail/employee-detail.component';

import { EmployeeService } from './employee.service';

in the metadata section:

@NgModule({

  declarations: [

    AppComponent,

    EmployeeListComponent,

    EmployeeDetailComponent

  ],

……

Add Service to the providers array

providers: [EmployeeService],

……..

*Step 3: Declare the injector as dependency to other required class(es)*

In the employee-list.component.js,

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

import { EmployeeService } from './../employee.service';

…..

export class EmployeeListComponent implements OnInit {

  public employees = []; //initialize the array

  constructor(private \_employeeservice: EmployeeService) { }

//Instantiate EmployeeService and assign to local var

  ngOnInit() { //Upon calling this by appModule, method getEmployees fetch data

    this.employees = this.\_employeeservice.getEmployees();

  }

}

Same goes for employee-detail.component.ts

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

import { EmployeeService } from './../employee.service';

….

export class EmployeeDetailComponent implements OnInit {

  public employees = [];

  constructor(private \_employeeservice: EmployeeService) { }

  ngOnInit() {

    this.employees = this.\_employeeservice.getEmployees();

  }

}

Add these selector to app.component.html

<employee-list></employee-list>

<employee-detail></employee-detail>

# HTTP, Observables & RxJS

|  |  |
| --- | --- |
|  | 1. HTTP Get request from EmpService  2. Receive the observable and cast it into an employee array  3. Subscribe to the observable from EmpList and EmpDetail  4. Assign the employee array to a local variable  RxJS  - Reactive Extensions for Javascript  - External library to work with Observables |

*Step 1: HTTP Get request from EmpService*

In app.module.ts,

import { HttpClientModule } from '@angular/common/http';

…

imports: [

    BrowserModule,

    AppRoutingModule,

    FormsModule,

    HttpClientModule //Registering the module, no need to add to provider. Angular does it automatically

  ], ….

In the employee.service.ts,

import { HttpClient } from '@angular/common/http';

….

export class EmployeeService {

  private \_url = "/assets/data/employees.json"; //create json file

  constructor(private http: HttpClient) { }

  getEmployees(){

    return this.http.get(this.\_url);

  }

}

employees.json

[

    {"id": 1, "name": "first", "age": 11},

    {"id": 2, "name": "second", "age": 21},

    {"id": 3, "name": "third", "age": 31},

    {"id": 4, "name": "fourth", "age": 41},

    {"id": 5, "name": "fifth", "age": 51}

]

*Step 2. Receive the observable and cast it into an employee array*

<https://www.youtube.com/watch?v=LmIsbzt-S_E&list=PLC3y8-rFHvwhBRAgFinJR8KHIrCdTkZcZ&index=21>

need to explore .+++.

# Routing and Navigation

1. Generate a project with routing option
2. Generate departmentList and employeeList components
3. Configure the routes
4. Add buttons and use directives to navigate

*Step 1: Generate a project with routing option*

Command for routing enabled project.

$ ng new routing-demo –-routing

Include, <base href="/"> to index.html in <head> section, if not generated.

Include below to app.module.ts, if not generated.

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

….

  imports: [

    BrowserModule,

    AppRoutingModule

  ],

*Step 2: Generate departmentList and employeeList components*

Create component with inline templates and inline styles

$ ng g c department-list -it –is

$ ng g c employee-list -it –is

*Step 3: Configure the routes*

Config routes can be done directly writing code on app-routing.module.ts

…

import { DepartmentListComponent } from "./department-list";

import { EmployeeListComponent } from "./employee-list";

…

const routes: Routes = [

  { path: "departments", component: DepartmentListComponent},

  { path: "employees", component: EmployeeListComponent},

];

But, for better code management, we can write code on app-routing.module.ts, define a new constant and add it to.

… export const routingComponents = [DepartmentListComponent, EmployeeListComponent];

In the app.module.ts,

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

Discard other routing components imports and register the previous constant to decorator.

…

@NgModule({

  declarations: [

    AppComponent,

    //DepartmentListComponent,

    //EmployeeListComponent

    routingComponents

  ],

…

*Step4: Add buttons and use directives to navigate*

Add these to app.component.html

<h1>Routing and Navigation</h1>

  <nav>

      <a routerLink="/departments" routerLinkActive="active">Departments</a>

      <a routerLink="/employees" routerLinkActive="active">Employees</a>

  </nav>

  <router-outlet></router-outlet>

Note: active class is been defined to styles.css.

# Wildcard Route and Redirecting Routes

Create a 404 page and add.

$ ng g c page-not-found -it –is

…

template: `

    <h3>Page Not Found</h3>

  `,

In the app-routing.module.ts:

…

import { PageNotFoundComponent } from './page-not-found/page-not-found.component'; …..

const routes: Routes = [

  //{ path: "", component: DepartmentListComponent}, default page calling via component

  //{ path: '', redirectTo: '/departments', pathMatch: 'prefix' }, prefixwise match, not good

  { path: '', redirectTo: '/departments', pathMatch: 'full' }, //reliable default

  { path: "departments", component: DepartmentListComponent},

  { path: "employees", component: EmployeeListComponent},

  { path: "\*\*", component: PageNotFoundComponent} //wildcard route must comes last

]; ……

export const routingComponents = [DepartmentListComponent,

  EmployeeListComponent,

  PageNotFoundComponent];

# Route Parameters

In the department-list.component.ts, add data to array and iterate them over templates.

template: `

  <h3>

  Department List

</h3>

<ul class="items">

  <li \*ngFor="let department of departments">

    <span class="badge">{{department.id}}</span> {{department.name}}

  </li>

</ul>

…

departments = [

    {"id": 1, "name": "Angular"},

    {"id": 2, "name": "Node"},

    {"id": 3, "name": "MongoDB"},

    {"id": 4, "name": "Ruby"},

    {"id": 5, "name": "Bootstrap"}

  ]

….

Add a new component, $ ng g c department-detail -it –is

In the app-routing.module.ts,

…

import { DepartmentDetailComponent } from './department-detail/department-detail.component';

…

const routes: Routes = [

  //{ path: "", component: DepartmentListComponent},

  //{ path: '', redirectTo: '/departments', pathMatch: 'prefix' },

  { path: '', redirectTo: '/departments', pathMatch: 'full' },

  { path: "departments", component: DepartmentListComponent},

  { path: "departments/:id", component: DepartmentDetailComponent}, //here we will fetch the id

  { path: "employees", component: EmployeeListComponent},

  { path: "\*\*", component: PageNotFoundComponent}

];

….

export const routingComponents = [DepartmentListComponent,

  EmployeeListComponent,

  PageNotFoundComponent,

  DepartmentDetailComponent];

## Passing parameter to URI

In department-list.component.ts,

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

….

template: `

  <h3>

  Department List

</h3>

<ul class="items">

  <li \*ngFor="let department of departments" (click)="onSelect(department)">

    <span class="badge">{{department.id}}</span> {{department.name}}

  </li>

</ul>

  `,

….

constructor(private router: Router) { }

…

onSelect(department) {

    this.router.navigate(['/departments', department.id]);

  }

## Read parameter from URI

For reading parameter from URI, we need Activated Route Service. In department-detail.component.ts,

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

…

template: `

  <h3>You selected department with id = {{departmentId}}</h3>

  `,

…

export class DepartmentDetailComponent implements OnInit {

  public departmentId;

  constructor(private route: ActivatedRoute) { }

  ngOnInit() {

    let id = parseInt(this.route.snapshot.paramMap.get('id'));

    this.departmentId = id;

  }

}

# paramMap Observable

in the department-detail.component.ts, if we add the previous next button, snapshot doesn’t get the updated id from the node. For overcoming this issue, we need to use subscribe to the paramMap to fetch the current id.

…

import { ActivatedRoute, Router, ParamMap} from '@angular/router';

…

template: `

  <h3>You selected department with id = {{departmentId}}</h3>

  <p>

    <button (click)="goPrevious()">Previous</button>

    <button (click)="goNext()">Next</button>

 </p>

…

constructor(private route: ActivatedRoute, private router: Router) { }

…

  ngOnInit() {

    //let id = parseInt(this.route.snapshot.paramMap.get('id'));

    //this.departmentId = id;

    this.route.paramMap.subscribe((params: ParamMap) => {

      let id = parseInt(params.get('id'));

      this.departmentId = id;

    });

  }

  goPrevious() {

    let previousId = this.departmentId - 1;

    this.router.navigate(['/departments', previousId]);

  }

  goNext() {

    let nextId = this.departmentId + 1;

    this.router.navigate(['/departments', nextId]);

  }

# Optional Route Parameters

In the department-detail.component’s template section:

<button (click)="gotoDepartments()">Back</button>

Define the handler.

gotoDepartments(){

    let selectedId = this.departmentId ? this.departmentId : null;

    this.router.navigate(['/departments', {id: selectedId}]);

}

This shall send an optional paramerter (id) to browser url like <http://localhost:4200/departments;id=3>

Capture this value from department-list.component.

import { ActivatedRoute, Router, ParamMap} from '@angular/router';

Inject to constructor(private router: Router, private route: ActivatedRoute) { }

public selectedId; //Declare property

ngOnInit() {

    this.route.paramMap.subscribe((params: ParamMap) => {

      let id = parseInt(params.get('id')); //capture using ParamMap Observable

      this.selectedId = id;

    });

  }

isSelected(department) { return department.id === this.selectedId; } //Method for checking department ID with the selected ID

Now, we need to bind with the isSelected method with the li tag.

<li \*ngFor="let department of departments" [class.selected]="isSelected(department)" (click)="onSelect(department)">

…..

Please note: optional route parameter, no need to specify the id parameter to app-routing.module for routing.

We can also pass more parameters if required.

gotoDepartments(){

    let selectedId = this.departmentId ? this.departmentId : null;

    this.router.navigate(['/departments', {id: selectedId, test: 'testValue'}]);

  }

# Relative Navigation

Till now, we have worked on the status URL. But, in case, we want to do that dynamic URL, we need Relative Navigation. If we change in the app-routing.module, we need to change all the consecutive usage of the respective URL as well. That is not flexible.

In the department-list.component,

onSelect(department) {

    //this.router.navigate(['/departments', department.id]);

    this.router.navigate([department.id], { relativeTo: this.route });

}

Now, onSelect will pass the value of department.id which is relative to this.router instead of static '/departments', hence it will go to the department detail page which was already mapped in const routes: Routes section of app-routing.module.

In the department-list.component, if we want to come to the previous page:

gotoDepartments(){

    let selectedId = this.departmentId ? this.departmentId : null;

    //this.router.navigate(['/departments', {id: selectedId, test: 'testValue'}]);

    this.router.navigate(['../', { id: selectedId }], { relativeTo: this.route }); //with optional parameter. Discard the yellow marked for discarding optional parameter.

}

# Child Routes

Create two new components

$ ng g c department-overview -it –is

$ ng g c department-contact -it –is

Import the new component to app-routing.module,

import { DepartmentContactComponent } from "./department-contact/department-contact.component";

import { DepartmentOverviewComponent } from "./department-overview/department-overview.component";

….

Here we hooked-up the routing table with DepartmentDetailComponent.

const routes: Routes = [

  //{ path: "", component: DepartmentListComponent},

  //{ path: '', redirectTo: '/departments', pathMatch: 'prefix' },

  { path: '', redirectTo: '/departments', pathMatch: 'full' },

  { path: "departments", component: DepartmentListComponent},

  { path: "departments/:id",

    component: DepartmentDetailComponent,

    children: [ //child is responsible for the child param under departments ID

      {path: "overview", component: DepartmentOverviewComponent},

      {path: "contact", component: DepartmentContactComponent},

    ]

  },

  { path: "employees", component: EmployeeListComponent},

  { path: "\*\*", component: PageNotFoundComponent}

];

….

In the department-detail.component, template section:

  <p>

    <button (click)="showOverview()">Overview</button>

    <button (click)="showContact()">Contact</button>

  </p>

  <router-outlet></router-outlet>

Define this function in code section,

  showOverview(){

    this.router.navigate(['overview'], { relativeTo: this.route });

  }

  showContact(){

    this.router.navigate(['contact'], { relativeTo: this.route });

  }

# Angular Material

Setting up new material project.

$ ng new material-demo

$ ng add @angular/material //Using the command below will install Angular Material, the Component Dev Kit (CDK), and Angular Animations in your project with theme. For more detail schematic, <https://material.angular.io/guide/schematics>

In the app.module.ts, **First Step: Import the library**

import { MatButtonModule } from '@angular/material';

**Second Step: Add it import array**

imports: [

    BrowserModule,

    AppRoutingModule,

    BrowserAnimationsModule,

    MatButtonModule //add it to import array

  ],

**Third Step: Use in HTML**

In the app.component.html,

<button mat-button>Hellow World!</button>

<button mat-raised-button>Hellow World!</button>

But for better code mgmt., we should create a separate module for centralizing for hosting necessary imports for material components. It also helps the app.module.ts nice and clean.

$ ng g m material

From app.module.ts, we deleted, import { MatButtonModule } from '@angular/material';

Imported, import { MaterialModule } from "./material/material.module"

…

  imports: [

    BrowserModule,

    AppRoutingModule,

    BrowserAnimationsModule,

    MaterialModule //imported custom module

    //MatButtonModule //Deleted

  ],

In the material.module.ts,

import { MatButtonModule } from '@angular/material';

…

const MaterialComponents = [

  MatButtonModule //created a const and used in the imports and exports section for rendering

]

@NgModule({

  declarations: [],

  imports: [

    CommonModule,

    MaterialComponents

  ],

  exports: [MaterialComponents]

})

Easy way to add Angular Material Responsive Navigation:

$ ng g @angular/material:material-nav --name=main-nav

### Learning:

To change angular CLI favicon icon, we have to put icon file in "**assets**" folder and give that path in index.html. Angular is not detecting any custom folder for that.

<link rel="icon" type="image/x-icon" href="./assets/images/favicon.png">