TESTYANTRA

SOFTWARE SOLUTIONS (INDIA) PVT. LTD.

Angular

EXPERIENTIAL
learning factory

Overview of Angular

- Angular is a javascript framework for developing single page applications

- Angular Applications can be built using Typescript

- Typescript uses
  - ✓ Class-based object oriented programming.
  - ✓ Static Typing.

History

- Angular JS 1.x
  - Uses Javascript.
- Angular 2
  - Totally rewritten in typescript.
  - Stable version released to market in 2016
- Angular 4
  - Http Client was introduced.
  - Router Guards were proposed.
  - Package size reduced by 40%.
- Angular 5
  - Support for progressive web apps.
- Angular 6
- Angular 7
- Angular 8
  - @ViewChild is added.
  - Faster re-build time

Note:

Every 6 months Google releases a new version of Angular. From Angular 4 onwards there were no major changes but came up with stability improvements

AngularJS vs. Angular

- Angular is a completely revived component-based framework in which an application is a tree of individual components.
- AngularJS
  - It is based on MVC architecture
  - This uses use JavaScript to build the application
  - Not a mobile friendly framework
  - Difficulty in SEO friendly application development
- Angular
  - This is based on Service/Controller
  - Introduced the typescript to write the application
  - This is a component based UI approach
  - Developed considering mobile platform
  - Ease to create SEO friendly applications

Angular CLI

- Angular CLI(Command Line Interface) is a command line interface to build angular apps using nodejs modules.
- You need to install using below npm command,

  “npm install -g @angular/cli”
- Command to create a new Project

  “ng new project-name”

Angular Project Structure

- e2e - This folder is for an end to end testing purposes. It contains the configuration files related to performing the unit test of the projects.
- node_modules - This folder contains the downloaded packages as per the configuration.
- src - This folder contains the actual source code. It contains 3 subfolders as –
- app - App folder contains the Angular project-related files like components, HTML files, etc.
- assets - Assets folder contains any static files like images, stylesheets, custom javascript library files (if any required), etc.
- environments - Environments folder contains the environment-related files which are required during deployment or build of the projects.

Config Files

When we create any Angular based project using Angular CLI, then every time it will create 3 different configuration files which help us to configure the projects along with its related dependencies.

- **tsconfig.json** – If tsconfig.json files exist within the project root folder, that means that the project is a basically TypeScript project. The tsconfig.json file specifies the root files and the compiler options required to compile the project.

- **package.json** – package.json is basically a JSON file that contains all information related to the required packages for the project. Also, with the help of this configuration files, we can maintain the Project Name and its related version by using the “name” and “version” property. Also, we can provide the build definition of the project using this file.

- **angular.json** – angular.json file is an Angular Application Environment based JSON file which contains all the information related to the project build and deployment. It tells the system which files need to change when we use ng build or ng serve command.

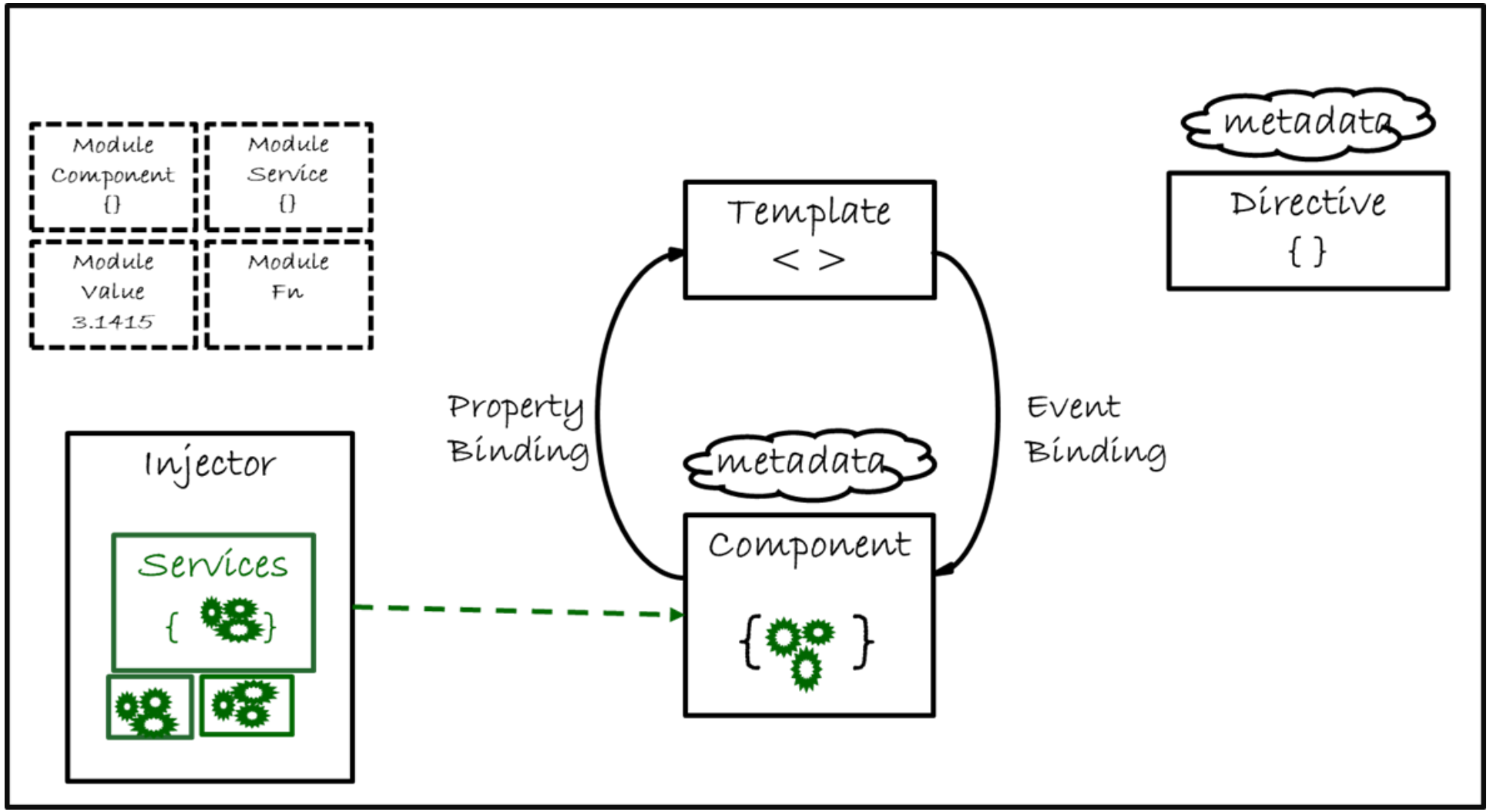
main.ts

- **main.ts** - The main.ts file acts as the main entry point of our Angular application. This file is responsible for the bootstrapper operation of our Angular modules. It contains some important statements related to the modules and some initial setup configurations like
- **enableProdMode** – This option is used to disable Angular's development mode and enable Productions mode. Disabling Development mode turns off assertions and other model-related checks within the framework.
- **platformBrowserDynamic** – This option is required to bootstrap the Angular app n the browser.
- **AppModule** – This option indicates which module acts as a root module in the applications.
- **environment** – This option stores the values of the different environment constants.

Key Components

- **Component:** These are the basic building blocks of angular application to control HTML views.
- **Modules:** An angular module is set of angular basic building blocks like component, directives, services etc. An application is divided into logical pieces and each piece of code is called as "module" which perform a single task.
- **Templates:** This represent the views of an Angular application.
- **Services:** It is used to create components which can be shared across the entire application.
- **Metadata:** This can be used to add more data to an Angular class.

Angular Architecture



Meta Data

- Metadata is used to decorate a class so that it can configure the expected behavior of the class.
- The metadata is represented by decorators.

1. Class decorators.

   e.g. @Component and @NgModule

2. Property decorators: Used for properties inside classes.

   e.g. @Input and @Output

3. Method decorators: Used for methods inside classes.

   e.g. @HostListener

4. Parameter decorators: Used for parameters inside class constructors.

   e.g. @Inject

Component

- Most basic building block of a UI in an Angular App.
- Angular components are a subset of directives.
- Only one component can be instantiated per element in a template.
- @Component marks a class as an Angular component and provides additional metadata to determine how the component should be processed, instantiated and used at runtime.
- Must belong to an NgModule in order for it to be usable by another component or application i.e., should be listed in the declarations field of that NgModule.
- Control their runtime behavior by implementing various Life-Cycle hooks.
- Component encapsulates the data, HTML markup, logic.

Why Components?

As per the current trend in web application development, the component-based architecture will be act as the most usable architecture in future web development. Because, with the help of this technique, we can reduce both development time and cost in a large volume in any large-scale web development projects. That's why technical experts currently recommend implementing this architecture in web-based application development.

- **Reusability**
- **Increase Development Speed**
- **Easy Integration**
- **Optimize Requirement and Design**
- **Lower Maintenance Costs**

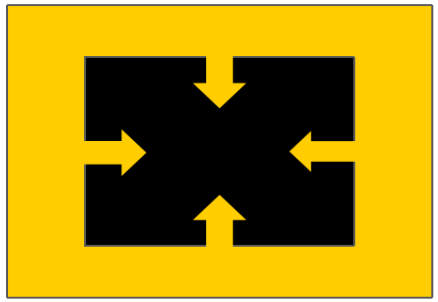
Component: Metadata

- **selector:** css selector that identifies this component in a template
- **template:** inline-defined template for the view
- **templateUrl:** url to an external file containing a template for the view
- **styles:** inline-defined styles to be applied to this component's view
- **styleUrls:** list of urls to stylesheets to be applied to this component's view
- **providers:** list of providers available to this component and its children
- **animations:** list of animations of this component
- **encapsulation:** style encapsulation to be used by this component
- **changeDetection:** change detection strategy used by this component

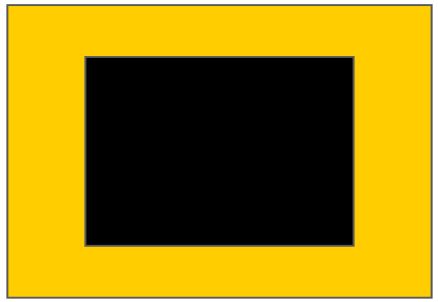
View Encapsulation

View encapsulation defines whether the template and styles defined within the component can affect the whole application or vice versa. Angular provides three encapsulation strategies:

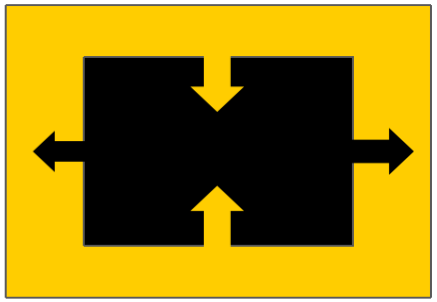
- Emulated (default) – Default. Inherit global styles. Local styles are private.
- Native – Deprecated in favor of ShadowDom.
- ShadowDom – Shadow DOM used to encapsulate styles. Global styles not inherited in Component. Local styles are private.
- None – Global styles are inherited. Local styles bleed out and are not private.



EMULATED



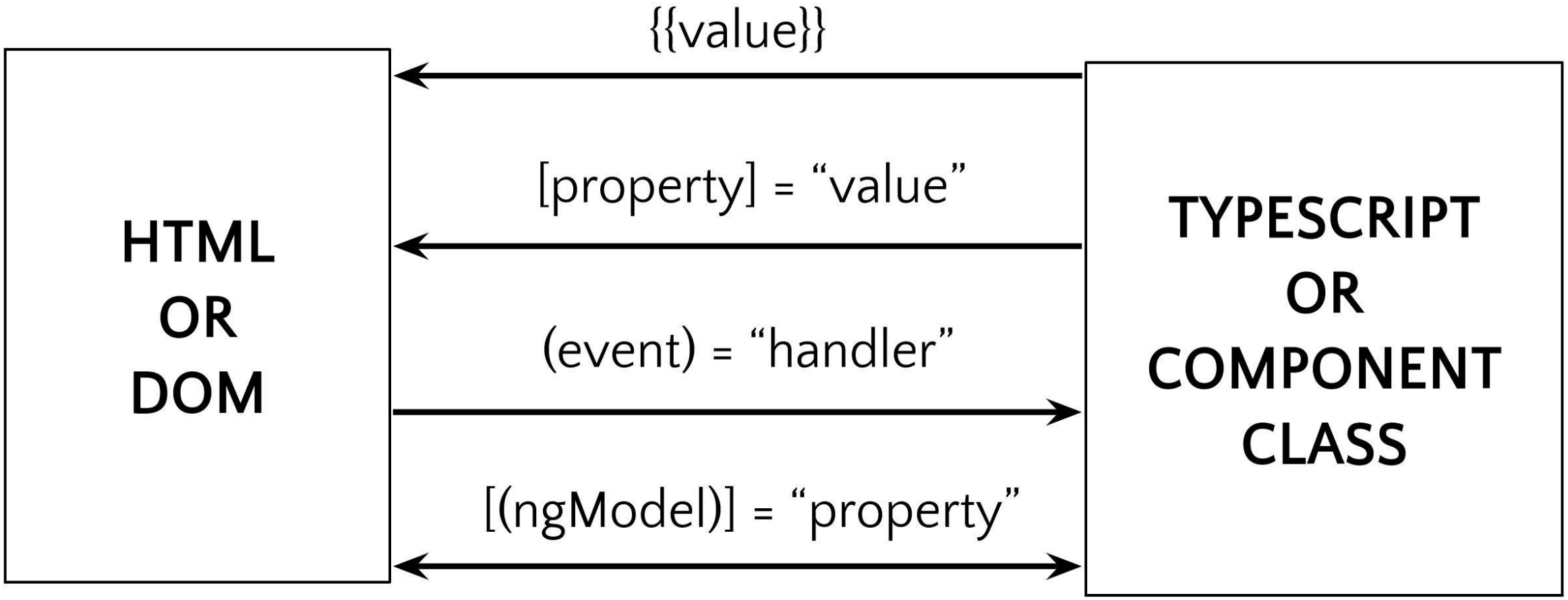
SHADOW DOM



NONE

Yellow – Global Styles
Black – Component Styles

Component Bindings



Interpolation

- When ever we need to communicate properties (variables, objects, arrays, etc.) from the component class to the template, we can make use of Interpolation.
- String Interpolation uses template expressions in double curly {{ }} braces to display data from the component, the special syntax {{ }}, also known as moustache syntax.
- The {{ }} contains JavaScript expression which can be run by Angular and the output will be inserted into the HTML.
- Say if we put {{ 5 + 5 }} in the template 10 will be inserted into the HTML.

Data Binding

Data binding is a core concept in Angular and allows to define communication between a component and the DOM, making it very easy to define interactive applications without worrying about pushing and pulling data.

- **One Way Data Binding**
  - Property Binding
  - Style Binding
  - Class Binding
  - Attribute Binding
  - Event Binding
- **Two Way Data Binding**

Property Binding - [...] = "..."

- Anything between the square braces[ ] represent the name of a property on an Angular Element or HTML Element.
- Anything between " " represents the value that we intend to assign to that property.
- This can be a component property name, an expression that resolves to a value, a method that returns a value or a value that was generated on the fly.
- Flows from Component TypeScript Class to Component HTML View.

Event Binding - (...) = “fn()”

- Anything between the parenthesis() represent the name of the event on an Angular Element or HTML Element.
- Anything between “ “ represents the Class method or an expression that we intend to execute.
- Usage: (eventName)=”someExpression” or (eventName)=”someMethod(someArg.)” .
- Flows from Component HTML View to Component TypeScript Class.

Attribute Binding - [attr.]

- Usually, setting an element property with a property binding is preferable to setting the attribute with a string. However, sometimes there is no element property to bind, so attribute binding is the solution.
- Consider the ARIA and SVG. They are purely attributes, don't correspond to element properties, and don't set element properties. In these cases, there are no property targets to bind to.
- Attribute binding is useful where we don’t have any property in DOM respected to an HTML element attribute.

  syntax: [attr.propertyname]=[propertyvalue]

Class Binding - [class.name]

- You can add and remove CSS class names from an element's class attribute with a class binding.
- Can add CSS Classes conditionally to an element, hence creating a dynamically styled element.
- Use [class]=”overridingCSSClassPropertyName” to override all the classes applied on an HTML Element.
- Toggle a specific CSS class on an HTML Element using

  [class.cssClassToToggle] = ”booleanPropertyName”

Style Binding - [style.name]

- Similar to Property Binding Syntax. Instead of property name in [ ], prefix a ‘style’, followed by . and then style name.
- Styles can be changed dynamically using Style Binding.
- Eg: [style.color]="isSpecial ? 'red': 'green'"

Two Way Data Binding - [()]

- Two-way binding gives your app a way to share data between a component class and its template.
- Two-way binding does two things:

  Sets a specific element property

  Listens for an element change event
- Angular offers a special two-way data binding syntax for this purpose, [()]. The [()] syntax combines the brackets of property binding, [], with the parentheses of event binding, ().
- Data Flows from Component HTML View to Component TypeScript Class and vice-versa.
- NgModel – adds two-way data binding to an HTML form element.

Component Interaction

@Input

- Only applicable for a Parent-Child Relationship
- Data to be sent from Parent to Child
- Child decorates a public property on it with the @Input decorator
- Parent passes some property to the Child in template using property name

@Output

- Only applicable for a Parent-Child Relationship
- Data to be sent from Child to Parent
- Child decorates a public property on it with the @Output decorator
- This property is generally of type EventEmitter<T>
- Children call emit on these properties which eventually calls a function binded to these properties via event binding in the Parent Component Template.

Directives

- Directives are a way of attaching behaviour to DOM elements.
- Directives are decorated with the @Directive decorator.
- Can be used in other components and directives.
- Directives are registered in the Declarations array of an NgModule.
- Directives are configured through the metadata passed to the @Directive decorator.
- Directives can implement Lifecycle hooks to control their runtime behavior.

Types of Directives

- **Components**

  Directives with a template

  Most common directive used throughout the app

- **Structural Directives**

  Directives that change the DOM layout by adding/removing elements

  Prefixed with an asterisk. Eg. *ngFor, *ngIf, *ngSwitch

- **Attribute Directives**

  Directives that change the appearance or behaviour of an element

  Used as attributes of elements. Eg. ngStyle, ngClass

Lifecycle Hooks

- **ngOnChanges:** Every time there is a change in one of the input properties of the component.
- **ngOnInit:** When given component has been initialized. Only called once after the first ngOnChanges.
- **ngDoCheck:** When the change detector of the given component is invoked. Allows us to implement our own change detection algorithm for the given component.
- **ngOnDestroy:** Just before Angular destroys the component. Use this hook to unsubscribe observables and detach event handlers to avoid memory leaks.
- **ngAfterContentInit:** After Angular performs any content projection into the components view
- **ngAfterContentChecked:** Each time the content of the given component has been checked by the change detection mechanism of Angular.
- **ngAfterViewInit:** When the component's view has been fully initialized.
- **ngAfterViewChecked:** Each time the view of the given component has been checked by the change detection mechanism of Angular.

Constructor vs. ngOnInit

- TypeScript classes has a default method called constructor which is normally used for the initialization purpose.
- Whereas ngOnInit method is specific to Angular, especially used to define Angular bindings.
- Even though constructor getting called first, it is preferred to move all of your Angular bindings to ngOnInit method.
- In order to use ngOnInit, you need to implement OnInit interface.

Content Projection

- Content Projection is the rendering of html specified within the component tags, inside the component html.
- This is achieved by adding the tags “<ng-content> </ng-content>” within the component html.
- The <ng-content> tags get replaced by the html enclosed within the component tags.
- The projected content can be accessed through the component using @ContentChild().

View Queries

- Angular provides the decorators @ViewChild, @ViewChildren, @ContentChild, @ContentChildren to get element references.
- ViewChild can be used to capture elements in the component template.
- ContentChild can be used to capture elements present in the opening and closing tags of a component.
- Angular allows us to create template references by adding a local variable #name to the HTML element.
- Template references can be used with ViewChild and ContentChild in order to get the element reference (ElementRef) in component.
- We can access and modify the native element properties through the element reference provided by ViewChild or ContentChild.

Services and Dependency Injection

TESTYANTRA SOFTWARE SOLUTIONS (INDIA) PVT. LTD.

EXPERIENTIAL learning factory

- An angular service is simply a Class that allows you to access it's defined properties and methods
- Services are decorated with @Injectable to allow injection of other services as dependencies
- Services are used to:

  share the same piece of code across multiple files

  Hold the business logic

  Interact with the backend

  Share data among components
- Services in Angular can be Singletons
- Services are registered on Modules or Components through providers

Routing - The Basics

- In Single Page Applications(SPAs), when there's a need for some new content, the whole page never changes.
- Only the content on that particular page changes.
- This gives the App a more Desktop Application like feeling.
- SPAs are faster as compared to normal Web Apps for the same reason.
- Routing is an Important Part of this behavior that SPAs exhibit.

Routing - How To?

- Create a separate module for routing.
- Import RouterModule, Routes in your AppRoutingModule.

```
import { Routes, RouterModule } from '@angular/router';
```

- Create a Routes Config.

```
let routes: Routes= [
    { path: 'home', component: HomeComponent },
    { path: 'about', component: AboutComponent },
    { path: 'register', component: RegisterComponent },
    { path: 'login', component: LoginComponent },
    { path: '', redirectTo: '/home', pathMatch: 'full' },
    { path: '**', component: PageNotFoundComponent }
]
```

- Call RouterModule.forRoot() and give it the Routes config that you just created.
- Export this Module into your RootModule.

```
@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule]
})
export class AppRoutingModule {
}
```

- Place a <router-outlet></router-outlet> tag in your template where you want to perform it.
- Place links that will take your user to those routes and use routerLink attribute to give them links.

Child Routes & Params

- Add a children property to a route, the value of it would be an array of routes.
- Each child route in the children will again contain a path property and a component property.
- If you want to show the component content in some content that's already present in <router-outlet>, you'll have to add another router outlet in its parent's template.
- You can configure a route to take params as well. Do that by supplying a colon(:) in front of the param name.
- You can get the value of the current route params or route query params using ActivatedRoute as a dependency.
- ActivatedRoute exposes a params Observable you can subscribe to, to get the params on the current route.
- ActivatedRoute also exposes a queryParams Observable you can subscribe to, to get the query params on the current route.

Guards

- Guards are a way of performing checks before we start navigating to or from different routes in our application.
- They allow us to restrict access to certain routes in our application to certain users.
- They allow us to validate/confirm before navigating out of routes.
- Guards themselves are simple classes, which can have dependencies injected into them.
- Guard functions return Booleans, or Observables and Promises which resolve Booleans.
- Navigation is carried out if Boolean returned is true, else it is prevented.
- A single route can have multiple guards, and they are checked in the order of injection .

Types of Route Paths

- **Absolute Path**

  Has '/' in the front. Takes you to hostname:port/name-of-the-supplied-path.

- **Relative Path**

  Has './' or nothing in front. Takes you to the current route followed by the route name provided.

  Eg: hostname:port/path-on/path-provided.

- **Parent Path**

  Has '../' in front. Takes you one level up in the route structure.

  Eg, if you're on hostname:port/level1/level2, it will take you to hostname:port/level1.

CanActivate/CanActivateChild

- CanActivate checks to see if a user can visit a route.
- CanActivateChild checks to see if a user can visit a routes children.
- Class which implements CanActivate/CanActivateChild interface from @angular/router.
- Accepts the arguments:
  - **route**: ActivatedRouteSnapshot - Future route. Contains params
  - **state**: RouterStateSnapshot - Future RouterState. Contains URL
- Needs to be registered on the providers array of module.
- Added to the canActivate/canActivateChild Array of route.
- Most commonly used to check if user is logged in or has sufficient privileges‘.

CanDeactivate

- CanDeactivate checks to see if a user can exit a route.
- Class which implements CanDeactivate interface from @angular/router.
- Accepts the arguments:
  - **component**: Component - The current component
  - **route**: ActivatedRouteSnapshot - Future route. Contains params
  - **state**: RouterStateSnapshot - Future RouterState. Contains URL
- Needs to be registered on the providers array of module.
- Added to the canDeactivate Array of route.
- Most commonly used to check if user is navigating out of a route without saving some change.

Resolve

- Resolve performs route data retrieval before route activation.
- Class which implements Resolve interface from @angular/router.
- Accepts the arguments:
  - **route**: ActivatedRouteSnapshot - Future route. Contains params
  - **state**: RouterStateSnapshot - Future RouterState. Contains URL
- Needs to be registered on the providers array of module.
- Added to the resolve Object of route with a data key.
- Accessed in component as route.snapshot.data['key'].
- Used to load necessary data before loading a route, often to set flags or prevent undefined/nulls.

Template Driven Forms

- Fully programmed in component's template.
- Angular is responsible for generating the JS Object Representation of the form.
- Template defines structure of the form.
- Validation rules are also defined in the template.

Reactive forms

- Form model created programmatically in the code (typescript code).
- Template can be dynamically generated based on the model.
- FormControls and FormGroups can be added dynamically in FormArrays.

Steps - Template Driven

- import { FormsModule } from '@angular/forms' and add it to imports array.
- <form #formName="ngForm" (ngSubmit)="submit(formName)">
- <input required minlength="8" maxlength="20" pattern="John Doe" name="name" ngModel #name="ngModel">
- Use the template variables to check whether there's an error in that FormControl.

Reactive Form

- When building Reactive Forms, the FormGroup, FormControl & FormArray classes are the fundamental building blocks of a form.
- To simplify the creation of forms, Angular provides a FormBuilder service.
- A Form Group is a collection of Form Controls.
- A Form Control is the programmatic connection between the form control in the template and the TypeScript code for the component.
- A Form Array supports a dynamic number of form controls.
- The Form Builder service uses an object literal to configure an entire form

Reactive vs Template Driven

Reactive Forms	Template Driven Forms
❖ The form control tree is created synchronously with code	❖ The form control tree is created asynchronously as part of the compilation process as directives are processed
❖ The form control tree is available immediately even before the child form elements have been created	❖ The form control tree is available after the child form elements have been created
❖ Good for complex dynamic forms	❖ Old approach for 2 way data binding

Form Validation

- Validation occurs at the control level, and the validation status is tracked on each control and is aggregated to the group and form level.

  Example: If a single control is invalid, its group and form are invalid

- Three states are tracked for each control: pristine, valid and untouched

  Pristine/Dirty – has any of the controls been modified

  Valid/Invalid – is the data in controls valid according to the validation rules

  Untouched/Touched – has the control fired its blur event

- Through these statuses form validation is performed.
- For Template Forms, the Form Control Object tree can be accessed either through template reference variables or by accessing the NgForm with ViewChild.
- For Reactive Forms, the Form Control Object tree is created as part of building the form, and if made available as a component property, it can be referenced from there in the template.

Showing Error messages

- **ng-invalid** class will be paired with the **ng-touched** class to show error messages for a control.
- CSS classes work for Template Forms and Reactive Forms.
- Each Form, Group and Control object in the Form Control object tree contains the following pairs of boolean properties

  valid / invalid

  pristine / dirty

  touched / untouched
- These properties can be used with template variables and directives such as NgIf to display validation messages

HttpClient

- Angular Apps can make AJAX Calls to fetch data.
- We use HttpClient for making AJAX Calls. It exposes APIs for all types of AJAX calls like GET, POST, PUT, DELETE, OPTIONS, PATCH, JSONP etc.
- Angular uses RXJS Observables to handle response.
- An Observable emit responses overtime, instead of response being a one time event.
- HttpClient can be used with HttpHeaders and HttpParams as well.

Observables

- Observable/Observer is a design pattern used for asynchronous programming.
- Observable is an object that streams data from some data source.
- It streams data to subscribers using push model.
- Observers subscribe to these Observables.
- Data pushed to subscribers can be transformed on the way from source to subscriber.

Observables vs Promises

Observables	Promises
Multiple values	Single values
Cancellable	Not Cancellable
Operators: Map, filter, reduce, etc	No Operators
Observables can be retried using retry and retryWhen operators	Access the original function to retry

Subject

- Subject is a special observable that act both as observer and observable.
- A subject can be subscribed to, just like an observable.
- A subject can subscribe to other observables.
- Subject can act as a bridge/proxy between the source Observable and many observers, making it possible for multiple observers to share the same Observable execution.
- Subjects can be used for sharing data between components.

Pipes

- Transforms some output into template.
- Can handle both synchronous and asynchronous data.
- Don’t change the value of the actual property itself. Just transforms the way it is presented on the UI.
- Since only responsible to transform the output, the logical place to use it, is the template.
- Several built in pipes provided as a part of Angular. Custom Pipes can also be built.
- Pipes are used by using the Pipe(|) symbol after the data to be transformed.
- Pipes can be chained with other pipes.
- Pipes can also be provided with arguments by using the colon (:) sign.

Custom Pipes

- Create a TypeScript Class with export keyword.
- Decorate it with the @Pipe decorator. Pass in the name property to its metadata.
- Implement the PipeTransform Interface on this class.
- Implement the transform method imposed due to the interface.
- Return the transformed data from the pipe.
- Add this pipe class to the declarations array of the module where you want to use it.
- OR simply use ng g p pipe-name. It will add the bare-bones of a pipe to your project and will also update your root module.
- You can also add arguments to your pipe by adding them to the transform function as parameters.
- By default the pipes are pure in nature. To change it and update view as the data changes, make them impure by adding the pure property to the metadata and setting it to false.

Pure & Impure Pipes

- The pipe is pure means that the transform() method is invoked only when its input arguments change. Pipes are pure by default
- For the impure pipes the transform() method is invoked on each change-detection cycle, even if the arguments have not changed

```
import { Pipe, PipeTransform } from '@angular/core';
import { Student } from './students/students.component';

@Pipe({
  name: 'filter',
  pure: false //set true for pure pipes and false for impure pipes
})
export class FilterPipe implements PipeTransform {

  transform(students: Student[], name: string): Student[] {
    if (name === undefined) {
      return students;
    } else {
      return students.filter(student => {
        return student.name.toLowerCase().includes(name.toLowerCase());
      })
    }
  }

}
```

Modules

- Module is a class annotated with @NgModule( ) Decorator.
- Angular Module is a mechanism to group components, directives, pipes and services that are related to a feature area of your angular application.

Types

- Feature Module
- Root Module
- Core Module
- Shared Module
- Routing Module

- ✓ Every Angular application must have a Root Module.
- ✓ By default AppModule is the Root Module.

Shared Module

- Shared Module contains all the commonly used directives, pipes and components that we want to share with other modules that import this shared module.
- Common Module consists of all the required basics to be used in our project. It is better to be imported in Shared Module.

Considerations for creating Shared Module

- May re-export other common angular modules (CommonModule, FormsModule, etc).
- Should not have providers.
- Shared Module should not import or re-export Modules that have providers.
- The Shared Module is imported by all the feature modules where we need the shared functionality.
- We can export an Angular Module without having it in imports array.

JIT (Just in Time Compilation)

- Compilation that happens at runtime
- It is perfectly fine for developing an application in the local machine
- The Angular Compiler will be shipped with part of vendor.bundle.js and it occupies more space

Drawbacks of JIT

1. In efficient for production
2. Happens for every user
3. More components, slower
4. We have to ship Angular compiler

AOT Compilation

User can download the precompiled application

Benefits

1. Faster Startup
2. Smaller bundle size
3. Catch template errors earlier
4. Better security

Optimization Techniques

Minification

- Removing comments & white spaces

Uglification

- Renaming long descriptive function and variable names to short ones

Bundling

- Multiple JavaScript files are bundled into a single file

Dead Code Elimination

- Removing the code that is not part of the application

AOT Compilation

- Precompiling Angular components and templates

All this optimization techniques can be applied with a single command

"ng build --prod"

TESTYANTRA

SOFTWARE SOLUTIONS (INDIA) PVT. LTD.

Thank You !!!

No.01, 3rd Cross Basappa Layout, Gavipuram Extension,
Kempegowda Nagar, Bengaluru, Karnataka 560019

praveen.d@testyantra.com

www.testyantra.com

EXPERIENTIAL
learning factory