CSM
breaking thought barriers

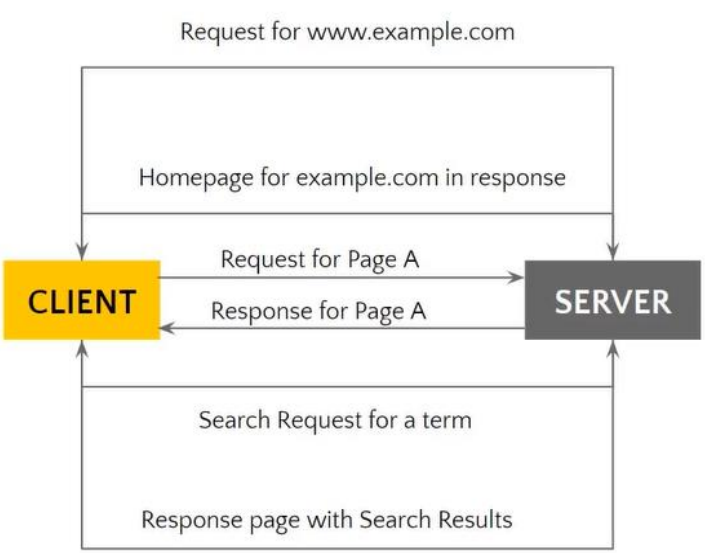
Angular Training

Session -12

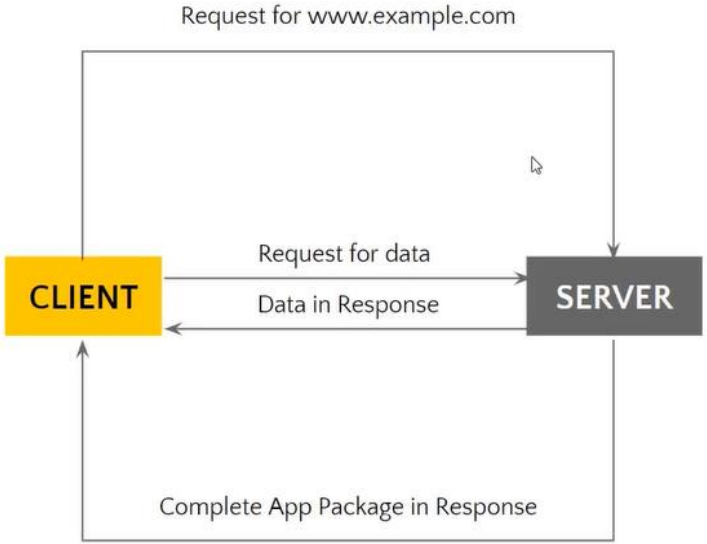
Outlines

- Angular Component
- Template VS TemplateUrl
- Nested Components
- Styling Components
- Databinding

SPA



- Multiple Page Requests going to Server.
- Client getting reloaded continuously.
- Eg: MVMT Watches



- Single Request to load the App, multiple data requests going to the server
- Client doesn't get reloaded continuously.
- Eg: AngularIO

Why Angular


Angular
Protractor
Forms
PWA
Augury
Language Services
Router
Elements
CDK
Universal
Karma
Labs
Compiler
I18N
HTTP
Material
Animations
CLI


PWA

Dependency Injection

Lazy Loading

Libraries

Performance

Templates

Environment Setup

Web Browser

NodeJS

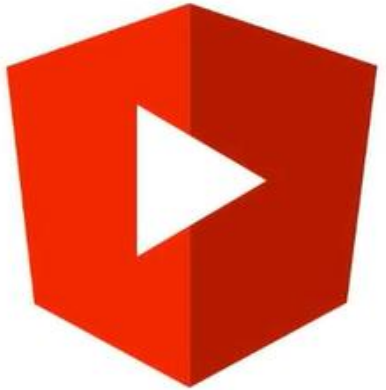
TypeScript

Angular CLI

Code Editor

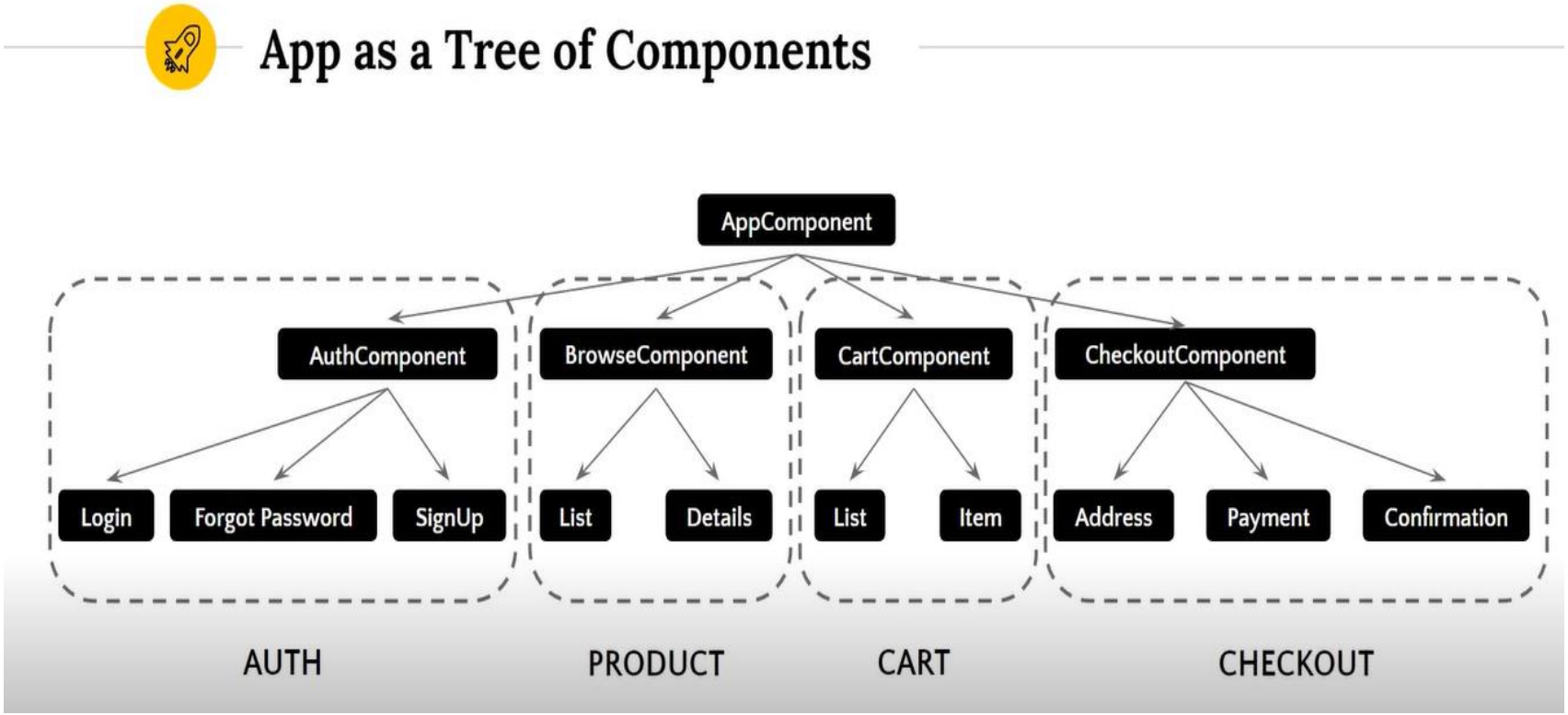
StackBlitz

CodeSandbox

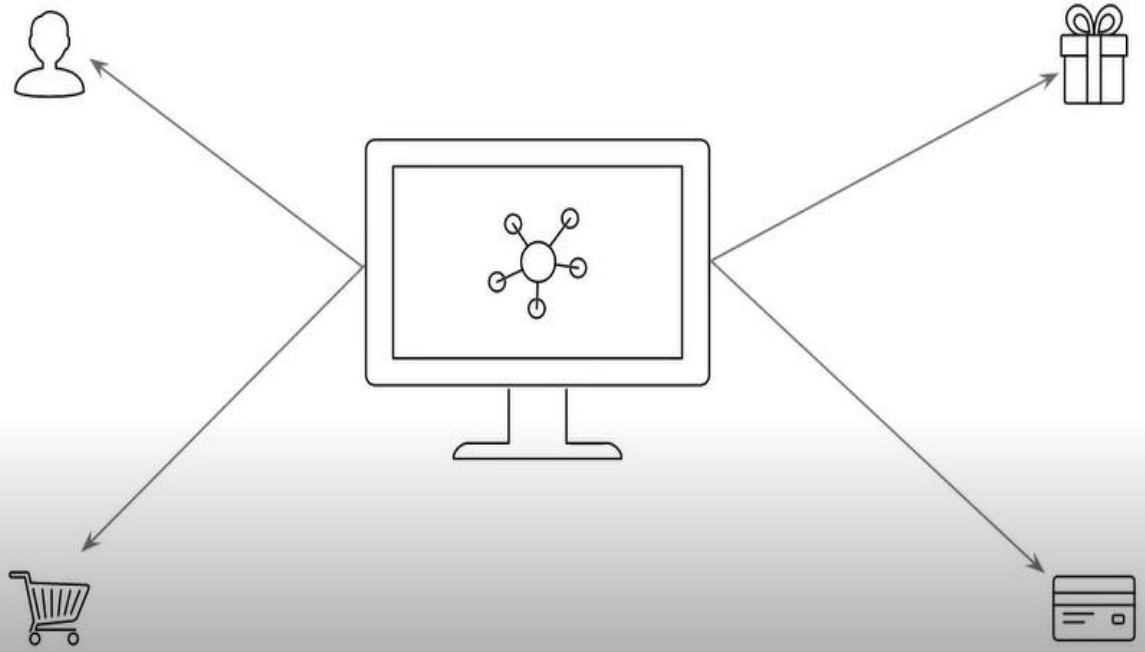


NG-Run

Angular Application Architecture



Overall App Architecture



Angular CLI

The Angular CLI (Command Line Interface) is a powerful tool that simplifies and streamlines the process of creating, developing, testing, and deploying Angular applications.

It provides a set of commands that help developers generate code, manage dependencies, run development servers, and perform various tasks related to building Angular applications.

The Angular CLI is an essential tool for any Angular developer as it significantly boosts productivity and enforces best practices.

Key Features

- Development Server
- Project Generation
- Build and Compilation
- Code Generation
- Testing
- Version Updates
- Deployment

1. **Project Creation:**

   ng new project-name

2. **Component Generation:**

   ng generate component component-name

3. **Service Generation:**

   ng generate service service-name

4. **Module Generation:**

   ng generate module module-name

5. **Directive Generation:**

   ng generate directive directive-name

6. Pipe Generation:

ng generate pipe pipe-name

7. Testing and Running:

ng serve

ng test

8. Building for Production:

ng build

9. Updating the CLI:

ng update @angular/cli

Workspace configuration files

WORKSPACE CONFIGURATION FILES	PURPOSE
.editorconfig	Configuration for code editors
.gitignore	Specifies intentionally untracked files that Git should ignore.
README.md	Introductory documentation for the root application.
angular.json	CLI configuration defaults for all projects in the workspace, including configuration options for build, serve, and test tools that the CLI uses, such as Karma, and Protractor.
package.json	Configures npm package dependencies that are available to all projects in the workspace. See npm documentation for the specific format and contents of this file.
package-lock.json	Provides version information for all packages installed into node_modules by the npm client.
src/	Source files for the root-level application project.
node_modules/	Provides npm packages to the entire workspace. Workspace-wide node_modules dependencies are visible to all projects.
tsconfig.json	The base TypeScript configuration for projects in the workspace. All other configuration files inherit from this base file. For more information, see the Configuration inheritance with extends section of the TypeScript documentation.

Application source files

APPLICATION SUPPORT FILES	PURPOSE
app/	Contains the component files in which your application logic and data are defined.
assets/	Contains image and other asset files to be copied as-is when you build your application.
favicon.ico	An icon to use for this application in the bookmark bar.
index.html	The main HTML page that is served when someone visits your site. The CLI automatically adds all JavaScript and CSS files when building your app, so you typically don't need to add any <script> or <link> tags here manually.
main.ts	The main entry point for your application. Compiles the application with the JIT compiler and bootstraps the application's root module (AppModule) to run in the browser. You can also use the AOT compiler without changing any code by appending the --aot flag to the CLI build and serve commands.
styles.css	Lists CSS files that supply styles for a project. The extension reflects the style preprocessor you have configured for the project.

Inside the src folder, the app folder contains your project's logic and data.

SRC/APP/ FILES	PURPOSE
app/app.config.ts	Defines the application config logic that tells Angular how to assemble the application. As you add more providers to the app, they must be declared here. *Only generated when using the --standalone option.*
app/app.component.ts	Defines the logic for the application's root component, named AppComponent. The view associated with this root component becomes the root of the view hierarchy as you add components and services to your application.
app/app.component.html	Defines the HTML template associated with the root AppComponent.
app/app.component.css	Defines the base CSS stylesheet for the root AppComponent.
app/app.component.spec.ts	Defines a unit test for the root AppComponent.
app/app.module.ts	Defines the root module, named AppModule, that tells Angular how to assemble the application. Initially declares only the AppComponent. As you add more components to the app, they must be declared here. *This file is not generated when using the --standalone option.*

Application configuration files

APPLICATION-SPECIFIC CONFIGURATION FILES	PURPOSE
tsconfig.app.json	Application-specific TypeScript configuration, including TypeScript and Angular template compiler options.
tsconfig.spec.json	TypeScript configuration for the application tests.

Angular Decorators

- Decorators are the features of Typescript and are implemented as functions.
- The name of the decorator starts with @ symbol following by brackets and arguments.

-The **@NgModule** decorator provides the necessary metadata to make the AppModule class as a module.

Note: If you want to create a module in angular, then you must decorate your class with @NgModule decorator. Once a class is decorated with @NgModule decorator, then only the class works as a module.

Commonly used Decorators:

@NgModule to define a module.
@Component to define components.
@Injectable to define services.
@Input and @Output to define properties

Note: All the above built-in decorators are imported from @angular/core library and so before using the above decorator, you first need to import the decorators from @angular/core library.

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

Types of Decorators in Angular:

1.Class Decorators: @Component and @NgModule

2.Property Decorators: @Input and @Output (These two decorators are used inside a class)

3.Method Decorators: @HostListener (This decorator is used for methods inside a class like a click, mouse hover, etc.)

4.Parameter Decorators: @Inject (This decorator is used inside class constructor).

Note: In Angular, each decorator has a unique role.

Angular Components

According to Team Angular, A component controls a patch of screen real estate that we could call a view and declares reusable UI building blocks for an application.

The core concept or the basic building block of Angular Application is nothing but the components. That means an angular application can be viewed as a collection of components and one component is responsible for handling one view or part of the view.

An Angular Component encapsulates the data, the HTML Mark-up, and the logic required for a view.

You can create as many components as required for your application.

Every Angular application has at least one component that is used to display the data on the view.

Technically, a component is nothing but a simple typescript class and composed of three things as follows:

- Class (Typescript class)
- Template (HTML Template or Template URL)
- Decorator (@Component Decorator)

Template:

- ✓ The template is used to define an interface with which the user can interact.
- ✓ As part of that template, you can define HTML Mark-up; you can also define the directives, and bindings, etc.
- ✓ The template renders the view of the application with which the end-user can interact i.e. user interface.

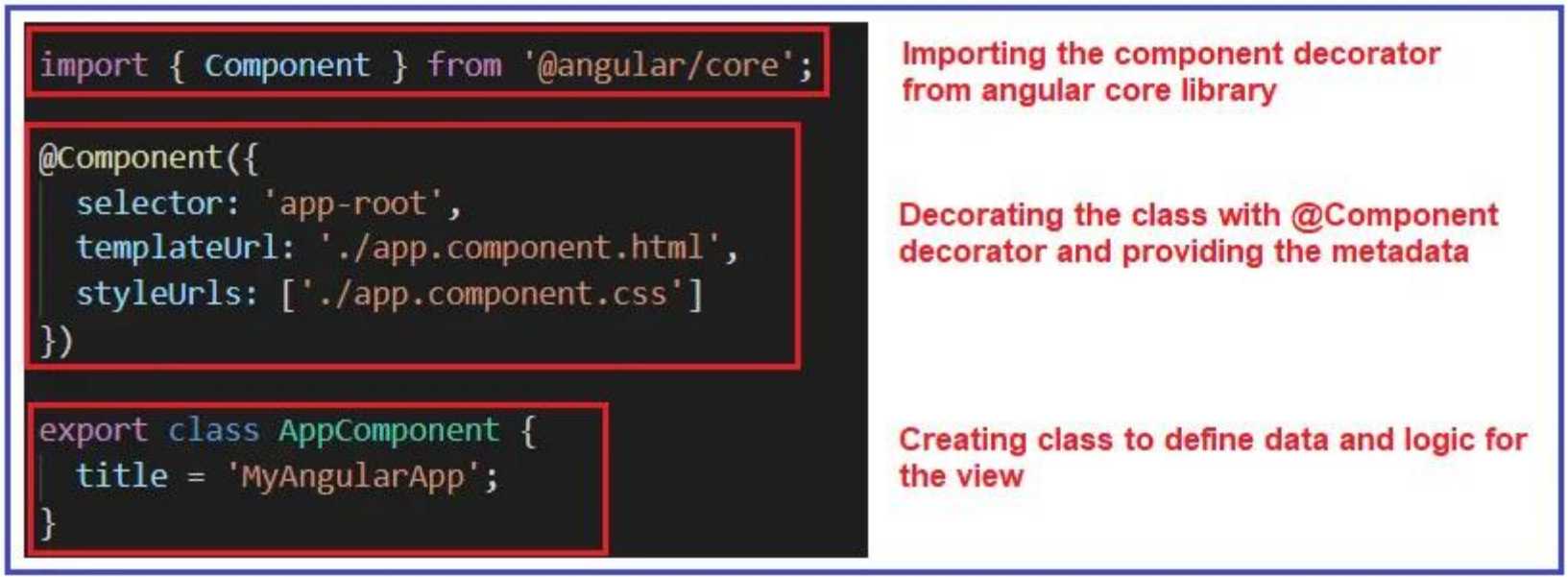
Class:

- ✓ The Class is the most important part of a component in which we can write the code which is required for a template to render in the browser.
- ✓ You can compare this class with any object-oriented programming language classes such as C++, C# or Java.
- ✓ The angular component class can also contain methods, variables, and properties like other programming languages.
- ✓ The angular class properties and variables contain the data which will be used by a template to render on the view.
- ✓ Similarly, the method in an angular class is used to implement the business logic like the method does in other programming languages.

Decorator:

- ✓ In order to make an angular class as a component, we need to decorate the class with the @Component decorator.
- ✓ Decorators are basically used to add metadata.

Note: Whenever we create any component, we need to define that component in @NgModule.


import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
title = 'MyAngularApp';
}
Importing the component decorator from angular core library
Decorating the class with @Component decorator and providing the metadata
Creating class to define data and logic for the view

How to create a Component in Angular?

```
ng g c componentname
```

Template VS TemplateUrl in Angular

Different ways to create Templates in Angular

- Inline template
- External Template

template

templateUrl

Angular Nested Components

The Angular framework allows us to use a component within another component and when we do so then it is called Angular Nested Components.

The outside component is called the parent component and the inner component is called the child component.

Styling Angular Components

Option1: Styling Angular Components using External Stylesheets

Option2: Styling Angular Components using Inline Styles

Option3: Styling in the component HTML file

Option4: Styling in the Component file

Option5: Styling using the @component decorator styleUrls property

Data Binding in Angular Application

Data binding is one of the most important features provided by Angular Framework which allows communicating between the component and its view.

Why do we need Data Binding?

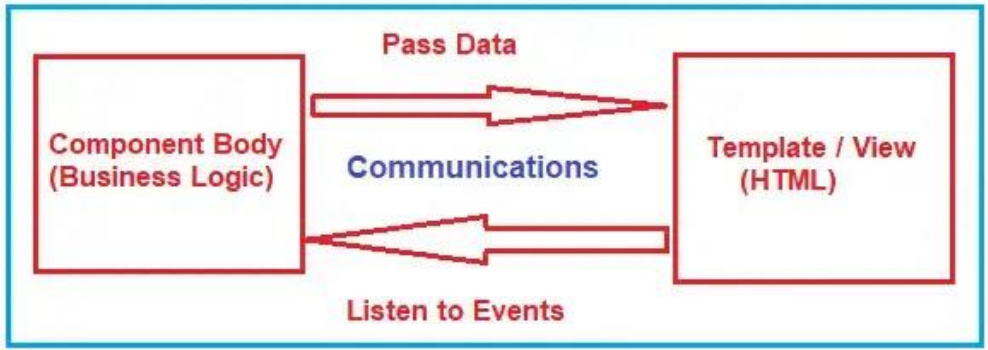
Whenever you want to develop any data-driven web application, then as a developer you need to keep the focus on two important things i.e. Data and the UI (User Interface) and it is more important for you to find an efficient way to bind them (Data and UI) together.

The angular framework provides one concept called Data Binding which is used for synchronizing the data and the user interface (called a view).

What is Data Binding in Angular Application?

In Angular, Data Binding means to bind the data (Component's filed) with the View (HTML Content). That is whenever you want to display dynamic data on a view (HTML) from the component then you need to use the concept Data binding.

Data Binding is a process that creates a connection to communicate and synchronize between the user interface and the data. In order words, we can say that Data Binding means to interact with the data and view. So, the interaction between the templates (View) and the business logic is called data binding.



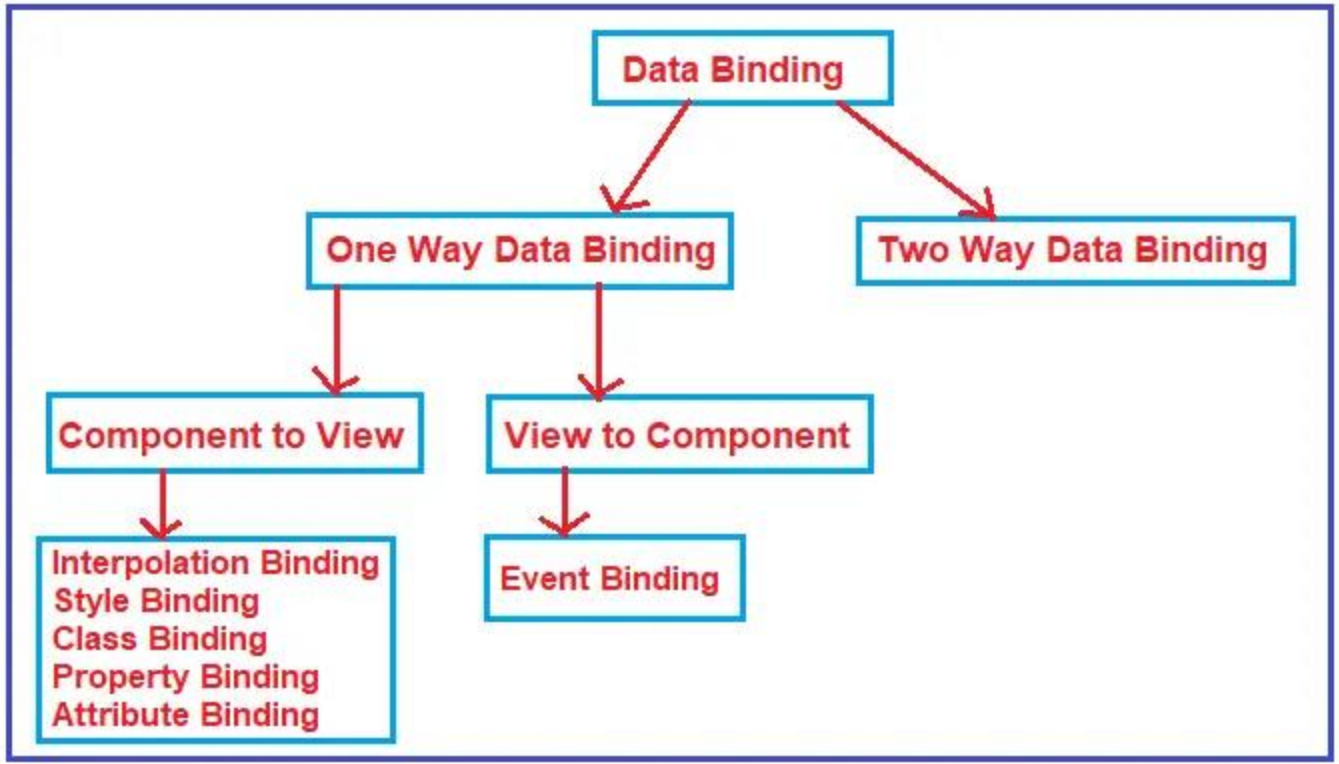
Types of Data Binding in Angular:

One-way Data Binding

where a change in the state affects the view (i.e. From Component to View Template) or change in the view affects the state (From View Template to Component).

Two-way Data Binding

where a change from the view can also change the model and similarly change in the model can also change in the view (From Component to View Template and also From View template to Component).


Data Binding
One Way Data Binding
Two Way Data Binding
Component to View
View to Component
Interpolation Binding
Style Binding
Class Binding
Property Binding
Attribute Binding
Event Binding

Examples of Angular Data Bindings:

- ❑ Interpolation
- ❑ Property Binding
- ❑ Attribute Binding
- ❑ Class Binding
- ❑ Style Binding
- ❑ Event Binding
- ❑ Two-way binding

Interpolation

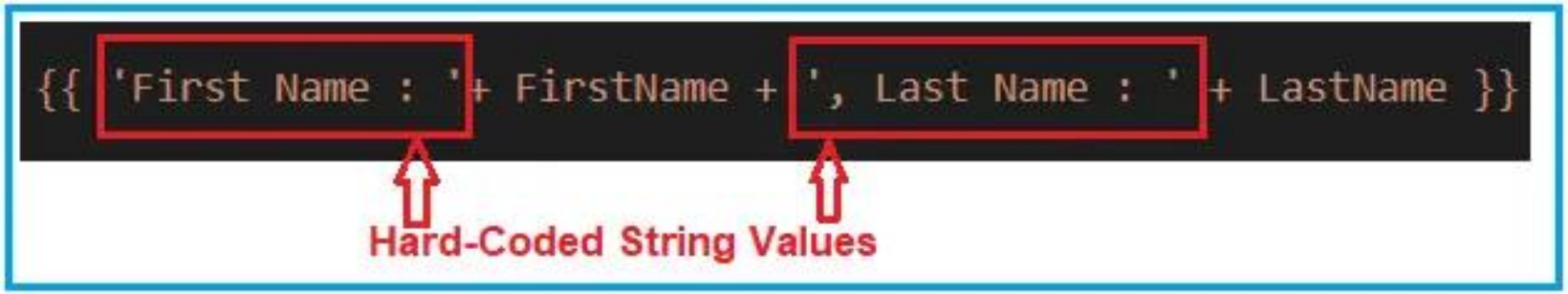
Angular Interpolation

If you want to display the read-only data on a view template (i.e. From Component to the View Template), then you can use the one-way data binding technique i.e. the Angular interpolation.

The Interpolation in Angular allows you to place the component property name in the view template, enclosed in double curly braces i.e. **{{propertyName}}.**

The Angular Interpolation is a technique that allows the user to bind a value to a UI element.

Angular Interpolation with hardcoded string



Angular Interpolation with Expression:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
    template: `<div>
                <h1> Bonus = {{ Salary * .10 }} </h1>
              </div>`
})

export class AppComponent {
  Salary : number = 100000;
}
```

Interpolation in Angular with Ternary Operator:

```
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
template: `<div>
<h1> Last Name : {{ LastName ? LastName : 'Not Available' }} </h1>
</div>`
})
export class AppComponent {
LastName : string = null;
}
```

Method Interpolation in Angular Application:

```
{{ GetFullName() }}
```

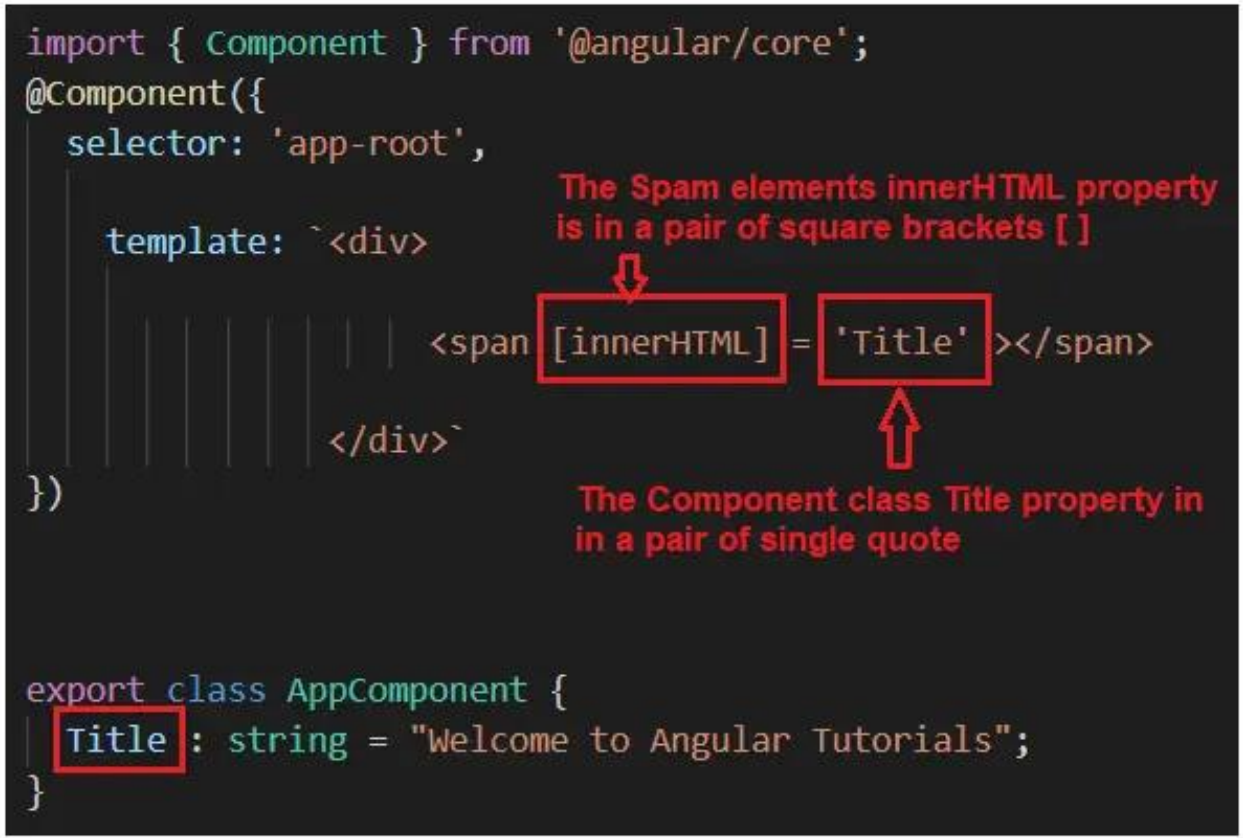
Displaying Images using Angular Interpolation:

Angular Property Binding

The Property Binding in Angular Application is used to bind the values of component or model properties to the HTML element. Depending on the values, it will change the existing behavior of the HTML element

The syntax : **[property] = ‘expression’**

Example : **span[innerHTML] = ‘ FirstName’.**


import { Component } from '@angular/core';
@Component({
selector: 'app-root',
template: `
<span [innerHTML] = 'Title' >
`
})
The Spam elements innerHTML property is in a pair of square brackets [ ]
The Component class Title property in in a pair of single quote
export class AppComponent {
Title : string = "Welcome to Angular Tutorials";
}

Angular Interpolation and Property Binding

Interpolation in Angular is just an alternative approach for property binding. It is a special type of syntax that converts into a property binding.

Scenarios where we need to use interpolation instead of property binding :

1. **If you want to concatenate strings then you need to use angular interpolation instead of property binding**

Working with non-string (Boolean) data:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
    template: `<div>
                <button [disabled] = ' IsDisabledClick' > Click Here </button>
            </div>`
})
export class AppComponent {
  IsDisabledClick : boolean = true;
}
```

<button disabled = {{IsDisabledClick} **> Click Here </button>**

With the above changes in place, irrespective of the IsDisabledClick property value of the component class, the button is always disabled. Here we set the IsDisabled property value as false but when you run the application, it will not allow the button to be clickable.

Providing Security to Malicious Content:

From the security point of view, both Angular data binding and Angular Interpolation protect us from malicious HTML content before rendering it on the web browser.

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
    template: `<div>
                  {{MaliciousData}}
              </div>`
})
export class AppComponent {
  MaliciousData : string = "Hello <script>alert('your application is hacked')</script>";
}
```

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
    template: `<div [innerHTML] = 'MaliciousData'>
              </div>`
})
export class AppComponent {
  MaliciousData : string = "Hello <script>alert('your application is hacked')</script>";
}
```

HTML Attribute VS DOM Property

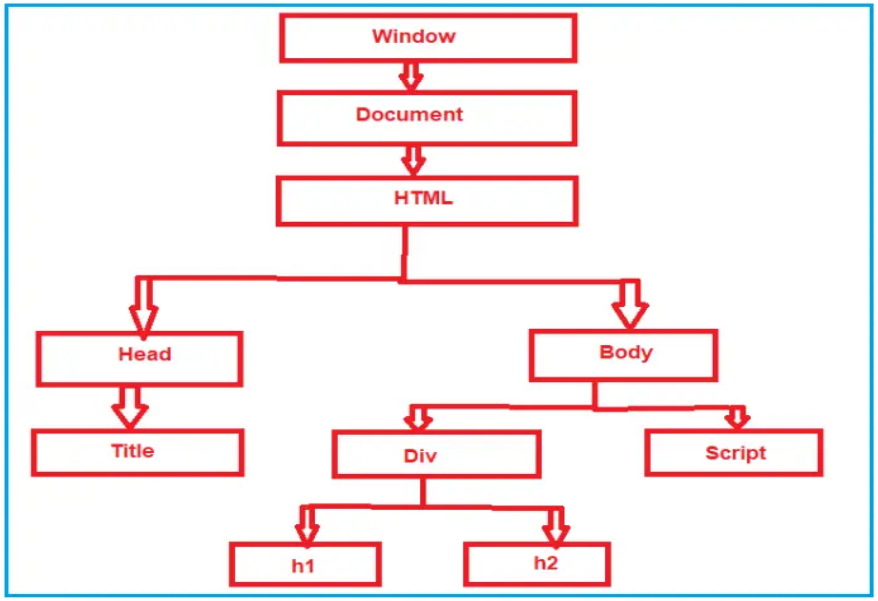
What is DOM?

The DOM stands for Document Object Model. When a browser loads a web page, then the browser creates the Document Object Model (DOM) for that page.

```
<html>
<head>
    <title>This is Title</title>
</head>
<body>
    <script src="Scripts/jquery-1.10.2.js"></script>
    <div>
        <h1>This is Browser DOM</h1>
        <h2>This is Inside H2</h2>
    </div>
</body>
</html>
```

The DOM is an application programming interface (API) for the HTML, and we can use the programming languages like JavaScript or JavaScript frameworks like Angular to access and manipulate the HTML using their corresponding DOM objects.

HTML Attribute VS DOM Property



We can say that the DOM contains the HTML elements as objects, their properties, methods, and events and it is a standard for accessing, modifying, adding or deleting HTML elements.

Interpolation example: <button disabled='{{IsDisabled}}'>Click Me</button>
Property binding example: <button [disabled]='IsDisabled'>Click Me</button>

The Angular data-binding is all about binding to the DOM object properties and not the HTML element attributes.

What is the difference between the HTML element attribute and DOM property?

- The Attributes are defined by HTML whereas the properties are defined by the DOM.
- The attribute's main role is to initializes the DOM properties. So, once the DOM initialization complete, the attributes job is done.
- Property values can change, whereas the attribute values can never be changed.
- Angular binding works with the properties and events, and not with the attributes.

Example :

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
    template: `<div>
                <input id=‘ inputId ' type='text' value=‘ Alok '>
              </div>`
})
export class AppComponent {
}
```

Angular Attribute Binding

- In Angular Interpolation and Property Binding, we have seen that they both (Interpolation and Property Binding) are dealing with the DOM Properties but not with the HTML attributes.

- But there are some HTML elements (such as colspan, area, etc) that do not have the DOM Properties.

- With Attribute Binding in Angular, you can set the value of an HTML Element Attribute directly. So, the Attribute Binding is used to bind the attribute of an element with the properties of a component dynamically.

```
<thead>
    <tr>
        <th [attr.colspan]="ColumnSpan">
            {{pageHeader}}
        </th>
    </tr>
</thead>
```

```
<thead>
    <tr>
        <th attr.colspan={{ColumnSpan}}>
            {{pageHeader}}
        </th>
    </tr>
</thead>
```

Note: **The Angular team recommends using the property binding or Interpolation whenever possible and use the attribute binding only when there is no corresponding element property to bind.**

Angular Class Binding

The Angular Class Binding is basically used to add or remove classes to and from the HTML elements.

It is also possible in Angular to add CSS Classes conditionally to an element, which will create the dynamically styled elements and this is possible because of Angular Class Binding.