Graphical user interface, application

Description automatically generated

Diagram

Description automatically generated

There are two ways in which an application developed using MVC can process a request.

1. Server side
2. Client side

You will understand the difference between server side and client side processing in the next page.

In server side MVC the request sent from a client is processed as shown below.

**1. The client sends a request to the server which contains the data logic and business logic**

Graphical user interface, diagram

Description automatically generatedGraphical user interface, application

Description automatically generated

In client side MVC the request sent from a client is processed as shown below.

**1. The client sends a request to the server. The server contains only the data logic.**

Graphical user interface

Description automatically generated Graphical user interface, diagram

Description automatically generated with medium confidence

Data logic handles data and business logic deal with user interaction/business rules. In case of SERVER side MVC, both are present at server end and in case of CLIENT side MVC : only data logic is in server side. Simply saying Data access layer which deals with data logic..

# Introduction to Angular and Typescript

Presentation layer can be developed using Server Side or Client Side MVC.

Server side MVC means the identification of appropriate model by the controller and construction of the view will happen on the server. The server will construct the view and deliver it to the browser.

Client side MVC means that the model,view and controller are built entirely on the client side. So the server only delivers data to the application. The binding of the data to the models happens in the client side. This helps to create views and retrieve data quickly.

One such framework which implements client side MVC is Angular.

Angular is a client-side JavaScript framework to build **S**ingle **P**age **A**pplications (SPA).

SPA are applications that are accessed through a web browser with reduced amount of page refreshes.

It communicates with back-end servers without doing a full page refresh to get data loaded into application. As a result, the process of rendering pages happen mostly on the client-side.

Example- Gmail

When you open mail from the inbox,it will only fetch the email and display it in place of the e-mail list. The rest of the components like sidebar, navigation bar etc. will not be reloaded. It only refreshes the content of required section and avoids loading of the entire website.

In angular every functionality has two main files associated with it.

* .html file which represents the view
* .ts file which represents the controller and the model

You already know to work with html files. ts stands for TypeScript. You have to learn TypeScript so that you will be able to create the controller and model.

TypeScript files contain classes, methods, constructors and interfaces which are the building blocks of Angular. They look similar to C# code with few syntactical changes.

So before you learn the concepts of Angular, you will learn how to work with TypeScript.

TypeScript is a strongly typed, object oriented, compiled language developed by Microsoft.

It is a superset of JavaScript. The TypeScript code is compiled to JavaScript before execution. In other words, it is JavaScript plus some additional features.

It is portable across browsers, devices, and operating systems. It can run on any environment that JavaScript runs on.

There are a lot of client side scripting languages created from JavaScript. All such languages should follow a set of specifications.

The ECMAScript(**E**uropean **C**omputer **M**anufacturers **A**ssociation) specification is a standardized specification of a scripting language. It was created to standardize JavaScript, so as to foster multiple independent implementations.

There are six editions of ECMA published. TypeScript adopts its basic language features from the ECMAScript5 specification, which is the official specification for JavaScript.

Language features like modules and class based orientation are in line with the ECMAScript6 specification.

# TypeScript Basics

Note: There are different IDE available to work with TypeScript. You can even use notepad to create a .ts file. Also note any valid .js file can be renamed to .ts and compiled with other TypeScript files.

If you do not find the option to create the HTML Application with TypeScript.

Go to Extensions plugin at the top of VS -->click on Manage Extensions--> search for **HtmlTypeScriptApp** extension file under online and install it.

TypeScript compileris a**transcompiler.**It produces an equivalent JavaScript source code ( \*.js)  from the Typescript file( \*.ts) given as an input to it. This process is termed as transpilation.

Graphical user interface, text, application

Description automatically generated

Note:When a TypeScript script gets compiled, there is an option to generate a declaration file(with the extension .d.ts) that functions as an interface to the components in the compiled JavaScript. The concept of declaration files is analogous to the concept of header files found in C/C++. The declaration files (files with .d.ts extension) provide intellisense for types, function calls, and variable support for JavaScript libraries like jQuery, MooTools,etc. You will not focus more on .d.ts file as it is out of scope of this course.

In order to execute the compiled javascript file include the greeting.js file in line 7 by modifying the index.html file as shown below:

<!DOCTYPE html>

<html>

<head>

<title>TypeScript Greeter</title>

</head>

<body>

<script src="greeting.js"></script>

</body>

</html>

## Defining datatypes

The syntax for declaring a variable in TypeScript is to include a colon (:) after the variable name, followed by its type. var keyword is used to define the variables.

1. Declare its type and value.

**var [identifier] :[type-annotation]=value;**

2. Declare its type but no value.

**var[identifier]:[type-annotation];**

3. Declare its value but no type.

**var[identifier]=value;**

4. Declare neither value nor type.

**var[identifier];**

**Table

Description automatically generated**

**Table

Description automatically generated**

**var productId: number = <number><any>"1001";**

**var productName: string = "Chess board";**

**var price: number = 99.50;**

**var isAvailable: boolean = true;**

**console.log("productId:", productId);**

**console.log("productName:", productName);**

**console.log("price:", price);**

**console.log("Availability:", isAvailable);**

**var contact: any;**

**console.log(typeof (contact))**

**contact = 9808034556;**

**console.log(typeof (contact))**

**contact = "John@gmail.com";**

**console.log(typeof (contact))**

**var productDesc: string = "Biscuits";**

**var ProductDesc: string = "Biscuits";**

**var picture = null;**

**console.log(picture); //shows null since the variable is assigned to null**

**console.log(typeof picture);**

**var cart;**

**console.log(cart); //shows undefined since it is not initialized**

**console.log(typeof cart);**

## Arrays and control structures

Text

Description automatically generated with low confidence

Text, website

Description automatically generated

var employees: number[] = [1001, 1002, 1003, 1004]

var employeeName: string[] = ["John", "Jesica", "Jacob", "Jim"]

var salary: number[] = [10000, 20000, 30000, 30000]

Alternate array declaration and initialization is shown below

1. var employees: number[];
2. employees = [1001, 1002, 1003, 1004]
3. var employeeName: string[];
4. employeeName=["John", "Jesica", "Jacob", "Jim"]
5. var salary: number[];
6. salary = [10000, 20000, 30000, 30000]

//var employees: number[];

//employees = [1001, 1002, 1003, 1004]

//var employeeName: string[];

//employeeName = ["John", 1002, "Jacob", "Jim"]

//var salary: number[];

//salary = [10000, 20000, 30000, 30000]

//console.log(typeof employees);

var employees: number[] = [1001, 1002, 1003, 1004]

var employeeName: string[] = ["John", "Jesica", "Jacob", "Jim"]

var salary: number[] = [10000, 20000, 30000, 30000];

var sum = 0;

for (var i = 0; i < salary.length; i++) {

sum = sum + salary[i];

}

console.log(`Total salary ${sum}`)

console.log(`------------------------------------`)

console.log('List of employees');

for (var employee of employeeName)

{

console.log(employee)

}

console.log(`------------------------------------`)

console.log('Incremented salary of employees');

for (var index in employees) {

salary[index] = salary[index] + 1000;

console.log(`${employeeName[index]} incremented salary is Rs${salary[index]}`);

}

console.log(`------------------------------------`)

  var choice:number;

  var productId:number[] = [1001, 1002, 1003, 1004, 1005 ];

  var productName:string [] = [ "Chess board", "Alice in wonderland", "Nike running shoes", "Moto G5", "Landscape oil painting" ];

  var price:number[] = [ 349.50, 250.25, 1700, 13000, 2750.75 ];

        console.log("\n -----Welcome to the world of shopping-----");

        console.log(" 1. View all products");

        console.log(" 2. Bill products");

        console.log(" 3. Exit");

        choice = 1

        if (choice == 1) {

            //View all products

            console.log();

            console.log(" Product Id\tProduct Name\tPrice\n");

            for (var index = 0; index < productId.length; index++)

            {

                console.log(productId[index] + "\t" + productName[index] + "\t" + price[index]);

            }

        }

        else if (choice == 2) {

            //Bill the products

            console.log("You have chosen to bill the products.");

        }

        else if (choice == 3) {

            console.log("You have chosen to exit.");

            }

        else {

            console.log("Please enter a valid menu option!");

        }

<!DOCTYPE html>

<html>

<head>

    <title>TypeScript Greeter</title>

    <script src="shop.js"></script>

</head>

<body>

</body>

</html>

debugger;

var input = 'stringofcharacters';

var count: number = 0;

for (var i = 0; i < input.length; i++) {

var substring = input.substring(i, i+1);

if (substring== 's') {

count++;

}

}

console.log('Number of characters',count);

## Working with Classes

Text

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

let is another keyword which can be used for variable declarations in TypeScript.

The syntax for variable declaration using let keyword is as shown below:

1. let variableName[:dataType];

The difference is scoping. var is scoped to the nearest function block and let is scoped to the nearest enclosing block, which can be smaller than a function block. Both are global if outside any block.

The difference between let and var keyword in typescript..

Class in typescript:

Text

Description automatically generated

class Customer {

//fields

id: number;

name: string;

contact: number;

hasCard: boolean;

//constructor

constructor(id: number, name: string, contact: number);

constructor(id: number, name: string, contact: number, hasCard: boolean);

constructor(id: number, name: string, contact: number, hasCard?: boolean) {

this.id = id;

this.name = name;

this.contact = contact;

if (hasCard != undefined) {

this.hasCard = hasCard;

}

else {

this.hasCard = false;

}

}

// function

RequestCard(): Boolean {

if (this.hasCard == false) {

this.hasCard = true;

return true;

}

else {

return false;

}

}

}

//defining a class and setting class parameters using a constructor

var customerOne: Customer = new Customer(1001, "John", 7204011993);

console.log('customer Id:', customerOne.id);

console.log('customer name:', customerOne.name);

console.log('customer contact:', customerOne.contact);

console.log('customer has card:', customerOne.hasCard);

// calling a method

console.log('\n Calling the request method')

customerOne.RequestCard();

console.log('customer has card:', customerOne.hasCard);

var customerTwo: Customer = new Customer(1002, "joe", 7204011994, true);

console.log('customer Id:', customerTwo.id);

console.log('customer name:', customerTwo.name);

console.log('customer contact:', customerTwo.contact);

console.log('customer has card:', customerTwo.hasCard);

Inheritance in TS

Text

Description automatically generated

Export and Import keywords in TS

Enum in TS

Text

Description automatically generated

Interfaces in TS

There are two more concepts namely interfaces and arrow functions(lambda function) which you should know before you start with Angular.

You will learn interfaces and arrow functions by trying out the following code.

Text

Description automatically generated

# Building a Component in Angular

Text

Description automatically generated

## Setting up the environment

Text

Description automatically generated

**npm install –g @angular/cli@9.0.0**

**ng version**

**Text

Description automatically generated with medium confidence**

**ng** – to call Angular CLI.

**new** - one of the many commands issued to the CLI.

**QuickKartApp** is the name of the project.

Select No when be prompted to add Angular routing. Select CSS for the stylesheet format.

Angular CLI installs the necessary packages, creates the project files and populates the project with a default app.

In order to build and launch the application in the browser type the following command.

1. ng serve -o

**ng serve** command launches the server

**--o or --open** option will automatically open the browser on <http://localhost:4200/>

You can view your app in the browser

## Understanding angular flow

Highlights:

* To understand the execution flow of Angular application
* To observe the default settings and files used for execution

Graphical user interface, text

Description automatically generated

**angular.json** - configuration for Angular CLI. In this file you can set several defaults and also configure what files are included when your project is built.

**package.json -**node configuration file which contains all dependencies required for your project. Node.js creates a folder **node-modules**and puts all the modules listed here in that folder. You can browse to node-modules folder present inside Infosys.QuickKart--> QuickKartApp in File Explorer and observe it.

**tsconfig.json -**typeScript compiler configuration for your IDE to pick up and give you helpful tooling.

**tslint.json -**used when running **ng lint**. Linting helps keep your code style consistent.

**README.md -**basic documentation for your app.

Text

Description automatically generated

Text

Description automatically generated

The main.ts file bootstraps(starts) the application’s root module i.e. the AppModule

AppModule is the root module of the application

If you open the app.module.ts file in the QuickKartApp->src->app folder you'll notice that, AppComponent is added to the **bootstrap** array of the module definition

Hence the AppModule in turn makes a call to the **AppComponent**, which is the root component of the application and makes it ready for use.

Graphical user interface, text, application

Description automatically generated

The **@NgModule**decorator identifies AppModule as an Angular Module class

The **declarations**array tells Angular which components belong to AppModule. As you create more components they need to be added to this array else an error message will be displayed while loading the component in the browser.

The **imports**array consists of various modules whose features are used in the application. For example, every application that runs in a browser requires the BrowserModule therefore, it is included in the imports array.

The **bootstrap**array tells Angular which component to execute when the application is launched. Here the AppComponent is bootstrapped when the application launches.

Graphical user interface, text

Description automatically generated

**@Component** is a decorator which helps to identify a class as a component.

Observe the selector i.e. **app-root** and templateUrl with a path in AppComponent class.

**selector -**custom tag which is used to render the html of the associated component.

**templateUrl -**contains the path of the html file that is associated with the current component.

**styleUrls -** Specifies the stylesheet file which contains CSS styles to be applied to the template.

Text

Description automatically generated

## Components

@Component is a decorator which takes some parameters of the format property:value to provide additional information to the class. One such property is templateUrl which defines the html view that should be associated with this class.

A class decorated with @Component with some additional information is called as an Angular component.

Component is a logical piece of code which is the basic building block for Angular application.

It consists of the following

* Template -  view for the application created in HTML
* Class - code with properties and methods which is used to support the view. It is defined in TypeScript.
* Metadata - extra data defined for the Angular class​​​​​​​​​​​​​​​​​​​

**Templates:**

Template is a part of component which is used as user interface by which the end-user can interact easily. We can create a template in two ways as follows:

1. Inline Template
2. External Template

Inline Template:

      When a template must be defined for a component in a .ts file inline templates are used. Inline templates are defined in the component decorator using the template property.

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

Highlights:

* Create a component using Angular CLI
* Observe the files created

Text

Description automatically generated

ng generate component viewProducts

ng g c viewProducts

A component is a class with some metadata attached to it.

Graphical user interface, text, application

Description automatically generated

**@Component** : Although the name of the class is Component, it is a normal class. It will be considered as a component only when it is attached with a decorator @Component.

**Decorators** are functions that are identified with a prefixed '@' symbol, and are immediately followed by a class, parameter, method or property. The decorator is supplied information about the class, method or property. Whenever the class is used, it will get this metadata to configure the expected behavior.

**import { Component }**: To import **Component**class from **core**module present in **node\_modules** folder.

Observe the **templateUrl**property in line no 5 in the above image. In place of the path to the view-products.component.html file, you can directly write the html code using the property **template**and the html content should be given as a string enclosed in single quotation marks('). To specify html in multiple lines you have to use back ticks (`).

**export class  :** The view-products.component.ts should be exported so that it can be used in another TypeScript files.

**Note**: The naming convention followed for component class is <functionalityname> suffixed with Component.

The selector of the component would be app-<componentname>. You can customise it based on the need.

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Any component class that you create will automatically be added in the declarations array of @NgModule decorator of the AppModule in order to make it visible to all the components.

**Note**:  You can now refer the selector of ViewProductsComponent in the AppComponent and directly in other components of the application. As you keep building more components in the application, the import statements will automatically be loaded into the AppModule.

## Directives in Angular

A directive is a custom HTML element that extends the power of HTML. Whenever the Angular compiler finds a directive in the DOM(Document Object Model) internally it is a function that gets executed.

Angular directives are divided into three types.

* Component
* Structural
* Attribute

The component in Angular which you learnt earlier is internally a directive. You will now understand about Structural and Attribute directives.

Structural directives are responsible shaping or reshaping the structure of DOM by adding, removing, or manipulating elements.

The most commonly used structural directives are

* ngIf
* ngFor
* ngSwitch

ngIf

ngIf takes a boolean expression as input. It adds or removes elements from DOM based on the value of the expression.

Syntax:

1. \*ngIf='expression'

If the expression evaluates to true then the element will be added to DOM, else it will be removed.

ngFor

ngFor is a way to present a list of items. It can be attached to a block of HTML that defines how a single item should be displayed. Angular will use that block as a template for rendering each item in the list.

Syntax:

1. \*ngFor = 'let item of itemlist'

Here item is a temporary variable to iterate the itemlist.

You will check if there are any products to be displayed to the user in QuickKart and create a template to iterate and display the products in the next demo.

ngSwitch

ngSwitch is another structural directive in Angular which allows us to add or remove DOM elements. It is similar to switch statement of Javascript.

Syntax:

1. <container\_element [ngSwitch]="switch\_expression">
2. <inner\_element \*ngSwitchCase="match\_expression\_1">...</inner\_element>
3. <inner\_element \*ngSwitchCase="match\_expression\_2">...</inner\_element>
4. <inner\_element \*ngSwitchCase="match\_expression\_3">...</inner\_element>
5. <inner\_element \*ngSwitchDefault>...</element>
6. </container\_element>

Observe the below code where choice is a variable declared in the component.

1. <div [ngSwitch]="choice">
2. <div \*ngSwitchCase="'1'">One</div>
3. <div \*ngSwitchCase="'2'">Two</div>
4. <div \*ngSwitchCase="'3'">Three</div>
5. <div \*ngSwitchCase="'4'">Four</div>
6. <div \*ngSwitchCase="'5'">Five</div>
7. <div \*ngSwitchDefault>This is Default</div>
8. </div>

Here, you are binding the ngSwitch directive to the div. Next, we have ngSwitchCase with matching expressions. Whenever the value of choice matches the expression, the element attached to it will be displayed. The ngSwitchDefault is used to display when all the other cases fail.

Here, if you observe [ngSwitch] the [] represent property binding which you will be learning in detail in the upcoming demos.

Attribute directive changes the appearance/behavior of a component/element

Following are built-in attribute directives

* ngStyle
* ngClass

ngStyle directive is used to modify a component/element’s style. If there are more than one css styles to be applied, you can use ngStyle attribute.

Syntax:

1. [ngStyle] = "expression"

The expression can accept an array, string or an object.

ngClass helps to dynamically set and change the CSS classes for a given DOM element.

If you have more than one CSS class to apply, then you can use ngClass.

Syntax:

1. [ngClass] = "expression"

The expression can accept an array, string or an object.

1. [ngClass] = "{css\_class\_name1 : Boolean expression, css\_class\_name2: Boolean expression, ……}"

Highlights:

* To change the visibility of controls using ngIf directive
* To create template for repetitive data using ngFor directive

Declare a **products**array of **any**type inside ViewProductsComponent class as shown below in line 10:

1. import { Component, OnInit } from '@angular/core';
2. @Component({
3. selector: 'app-view-products',
4. templateUrl: './view-products.component.html',
5. styleUrls: ['./view-products.component.css']
6. })
7. export class ViewProductsComponent implements OnInit {
8. products: any[];
9. constructor() { }
10. ngOnInit() {
11. }
12. }

Open view-products.component.html and replace the existing code with the following lines of code to create a table, for displaying products details.

1. <div style="text-align:center;">
2. <h1>View Products</h1>
3. <div class="table-responsive">
4. <table class="table" style="border:5px solid rgba(220,230,242,1);">
5. <tr style="background-color:rgba(220,230,242,1); font-size:12pt">
6. <th style="text-align:center">Product Id</th>
7. <th style="text-align:center">Product Name</th>
8. <th style="text-align:center">Category Id</th>
9. <th style="text-align:center">Price</th>
10. <th style="text-align:center">Quantity Available</th>
11. <th style="text-align:center">Action</th>
12. </tr>
13. <tr \*ngFor="let product of products" style="background-color:white">
14. </tr>
15. </table>
16. </div>

</div>

Note that here the structural directive **\*ngFor**is used to iterate over the products array declared in the ViewProductsComponent class in the previous step.

Currently, the products array is empty. The code for displaying the details of the products will be written in the upcoming demos.

Execute your application and check if the page is loaded as shown below.

Text

Description automatically generated

**npm install bootstrap font-awesome**

@import "~bootstrap/dist/css/bootstrap.css";

@import "~font-awesome/css/font-awesome.css";

Since the products array is empty, the table added in the view-products.component.html should not be visible to the end users.

It should made visible to users only when product details are present in products array.

You can use **\*ngIf**to achieve this.

Modify the existing table present in view-products.component.html as shown below by adding line 4:

1. <div style="text-align:center;">
2. <h1>View Products</h1>
3. <div class="table-responsive">
4. <table class="table" style="border:5px solid rgba(220,230,242,1);" \*ngIf="products">
5. <tr style="background-color:rgba(220,230,242,1); font-size:12pt">
6. <th style="text-align:center">Product Id</th>
7. <th style="text-align:center">Product Name</th>
8. <th style="text-align:center">Category Id</th>
9. <th style="text-align:center">Price</th>
10. <th style="text-align:center">Quantity Available</th>
11. <th style="text-align:center">Action</th>
12. </tr>
13. <tr \*ngFor="let product of products" style="background-color:white">
14. </tr>
15. </table>
16. </div>
17. </div>

Here \*ngIf="products" checks if the products array is empty. If it is empty then the table will not be displayed.

Execute the application and observe that the table is not displayed.

When the table is not displayed, appropriate message should be displayed to the end users.

Here you can make use of **else**statement.

Add line 4 to view-products.component.html file.

1. <div style="text-align:center;">
2. <h1>View Products</h1>
3. <div class="table-responsive">
4. <table class="table" style="border:5px solid rgba(220,230,242,1);" \*ngIf="products; else elseBlock">
5. <tr style="background-color:rgba(220,230,242,1); font-size:12pt">
6. <th>Product Id</th>
7. <th>Product Name</th>
8. <th>Category Id</th>
9. <th>Price</th>
10. <th>Quantity Available</th>
11. <th>Action</th>
12. </tr>
13. <tr \*ngFor="let product of products" style="background-color:white"></tr>
14. </table>
15. </div>
16. <ng-template #elseBlock>
17. <div>
18. <h4 class="jumbotron" style="text-align:center;">No products available</h4>
19. </div>
20. </ng-template>
21. </div>

Here a reference variable **#elseBlock** is declared.

 If the first \*ngIf statement is false then the else part is executed i.e. the template with the reference as #elseBlock will be displayed.

Execute your application and check if the page is loaded as shown below.

Graphical user interface, text, application

Description automatically generated

Another way to display the appropriate message is to add another **\*ngIf**in place of the else part.

In order to do this, add a variable showMsgDiv which is a boolean value to indicate if the products array has elements or not.

Modify the view-products.component.ts by adding line 12 as shown below:

1. import { Component, OnInit } from '@angular/core';
2. @Component({
3. selector: 'app-view-products',
4. templateUrl: './view-products.component.html',
5. styleUrls: ['./view-products.component.css']
6. })
7. export class ViewProductsComponent implements OnInit {
8. products: any[];
9. showMsgDiv: boolean = false;
10. constructor() { }
11. ngOnInit() {
12. }
13. }

Here showMsgDiv is initialized with false. Its value should be changed based on whether products array contains any data. This check should be performed every time the component is loaded.

In order to check the presence of data in products array whenever the component is loaded you can make use of ngOnInit() method of OnInit.

Implement the ngOnInit() method from line 17 - 20 as shown below.

1. import { Component, OnInit } from '@angular/core';
2. @Component({
3. selector: 'app-view-products',
4. templateUrl: './view-products.component.html',
5. styleUrls: ['./view-products.component.css']
6. })
7. export class ViewProductsComponent implements OnInit {
8. products: any[];
9. showMsgDiv: boolean = false;
10. constructor() { }
11. ngOnInit() {
12. if (this.products == null)
13. {
14. this.showMsgDiv = true;
15. }
16. }
17. }

Here **OnInit**is a life cycle hook imported from the core module, which is implemented by the ViewProductsComponent class in order to recognize ngOnInit() method. This method checks if the the products array is null. If it is null, showMsgDiv property is set to true.

**Lifecycle hooks:**

Component instances have a lifecycle as angular creates, updates, and destroys them. Developers can tap into key moments in that lifecycle by implementing one or more of the lifecycle hook interfaces in the angular core library.

Each interface has a single hook method whose name is the interface name prefixed with ng. For example, the OnInit interface has a hook method named **ngOnInit()**

**ngOnInit()**will initialize the component after angular displays the data-bound properties and sets the component's input properties. It is invoked only once when the component is instantiated.

Similarly there are many Lifecycle hooks in angular but in this demo only **OnInit**is being used.

You can learn about [life cycle hooks](https://angular.io/guide/lifecycle-hooks) in detail.

Add the following code from line 17 - 19 in view-products.component.html to display message based on the value of showDivMsg property.

1. <div style="text-align:center;">
2. <h1>View Products</h1>
3. <div class="table-responsive">
4. <table class="table" style="border:5px solid rgba(220,230,242,1);" \*ngIf="products">
5. <tr style="background-color:rgba(220,230,242,1); font-size:12pt">
6. <th>Product Id</th>
7. <th>Product Name</th>
8. <th>Category Id</th>
9. <th>Price</th>
10. <th>Quantity Available</th>
11. <th>Action</th>
12. </tr>
13. <tr \*ngFor="let product of products" style="background-color:white">
14. </tr>
15. </table>
16. </div>
17. <div \*ngIf="showMsgDiv" style="">
18. <h4 class="jumbotron" style="text-align:center;">No products available</h4>
19. </div>
20. </div>

You can alternatively do this using \*ngIf="!products". The message will be displayed if there are no products.

**Highlights:**

* To change the style of elements using ngClass and ngStyle directives

**Demosteps:**

In order to learn how to use attribute directives, try out the following steps:

Make changes to the ViewProductsComponent that you created in the previous demo.

**ngClass**

ngClass directive allows you to add or remove CSS classes to an HTML element.

Add the following classes to view-products.component.css

1. .red{color:red;}
2. .size30{font-size:30px;}

Add the below code shown in Line 2 to view-products.component.html

1. <div style="text-align:center;">
2. <h1 ngClass='red size30'>View Products</h1>
3. <div class="table-responsive">
4. <table class="table" style="border:5px solid rgba(220,230,242,1);" \*ngIf="products">
5. <tr style="background-color:rgba(220,230,242,1); font-size:12pt">
6. <th>Product Id</th>
7. <th>Product Name</th>
8. <th>Category Id</th>
9. <th>Price</th>
10. <th>Quantity Available</th>
11. <th>Action</th>
12. </tr>
13. <tr \*ngFor="let product of products" style="background-color:white">
14. </tr>
15. </table>
16. </div>
17. <div \*ngIf="showMsgDiv" style="">
18. <h4 class="jumbotron" style="text-align:center;">No products available</h4>
19. </div>
20. </div>

You can also use ngClass as shown below.

 <h1 [ngClass]="'red size30'">View Products</h1>

Graphical user interface, text, application

Description automatically generated

**ngStyle**

ngStyle directive allows you to set the inline style of an HTML element using an expression.

Add the below code in Line 1 to view-products.component.html.

1. <div [ngStyle]="{'background-color':'yellow'}" style="text-align:center;">
2. <h1 ngClass='red size30'>View Products</h1>
3. <div class="table-responsive">
4. <table class="table" style="border:5px solid rgba(220,230,242,1);" \*ngIf="products">
5. <tr style="background-color:rgba(220,230,242,1); font-size:12pt">
6. <th>Product Id</th>
7. <th>Product Name</th>
8. <th>Category Id</th>
9. <th>Price</th>
10. <th>Quantity Available</th>
11. <th>Action</th>
12. </tr>
13. <tr \*ngFor="let product of products" style="background-color:white">
14. </tr>
15. </table>
16. </div>
17. <div \*ngIf="showMsgDiv" style="">
18. <h4 class="jumbotron" style="text-align:center;">No products available</h4>
19. </div>
20. </div>

Here, if you observe [ngStyle] the [] represent property binding which you will be learning in detail in the upcoming demos.

Graphical user interface, text, application, email, website

Description automatically generated

Discard all the changes you made in this demo and revert to the previous state of the demo before you proceed with the next demo.

# Data Binding in aglr

Binding coordinates the communication between the component class and template and often involves in passing data.

You can provide values from the class to the template for display and the template raises events to pass the user actions or user entered values back to the class.

Angular provides the below types of binding

* Interpolation
* Property binding
* Event binding
* Two way binding

Interpolation is a one way binding technique. It is recognized using {{}} (double curly braces).

The information that is present in between the curly braces is called template expression. Angular evaluates the expression converts the result into a string and assigns it to an HTML element or attribute property.

Diagram

Description automatically generated

Interpolation can be used to

* bind a value from the component's class to the template
* perform operations such as concatenation, calculations or method invocations etc.

Example

1. <h1>{{'Welcome'+ userName}}</h1> *<!--concatenation-->*
2. <p>{{20\*10+1}}</p> *<!--calculation-->*
3. <div>{{'Balance is' + getBalance()}}</div> *<!--method invocation-->*

Angular prevents Cross-site scripting (XSS) that enables attackers to inject malicious code into web pages. It treats all values as untrusted by default. When a value is inserted into the DOM from a template, via property, attribute, style, class binding, or interpolation, Angular sanitizes and escapes untrusted values.

Interpolated content is always escaped. This means the HTML will not be interpreted and the browser will display the data in the element's text content.

<p>{{<b>Hello</b>}}</p> *<!--This will display the output as <b>Hello</b> instead of showing Hello in bold -->*

Property binding is also a one way binding technique. It helps you to set the property of an HTML element to the value of a template expression.

Syntax:

1. <element [binding target]='template expression'/>

Here binding target is the property of the html element to be set. It should be enclosed with in []. The template expression is the binding source (property of component class) which should be enclosed in ' '.

Example:

1. <img [src]='card.imageUrl'/>

You can bind values to the elements either using interpolation or property binding. In general property binding is preferred over interpolation. However if you need to include a template expression as a part of a larger expression you may need to use interpolation.

Example:

1. <img src='http://amigowallet.org/{{card.imageUrl}}'/>

Event binding is used to send information from a template to the component's class based on some user actions like click of a button.

Syntax:

1. <element (target event)='Template statement'/>

Here target event is the name of the event on which information should be sent to the component's class. It should be enclosed with in (). Template statement is often the name of the component class method to be invoked on the occurrence of the event.

Example:

1. <button (click) ='getTransactions()'/>

In the above code getTransactions() is a method of the associated component class which will be invoked on click of the button.

You will use interpolation and event binding to display the products belonging to a selected category in QuickKart.