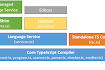
TypeScript Architecture  
  
1. TypeScript Core Compiler  
    - It translates the typescript code into OS native code.  
    - It identifies the issues in code and report them as errors.  
    - It comprises of various components like  
  
        core.ts                It verifies the keywords      
        program.ts            It verifies the structure  
        scanner.ts            It  control input  
        emitter.ts                It control output  
        parser.ts                It controls translating one type to another  
        checker.ts            It check the data type  
  
2. Standalone TS Compiler [tsc.ts]  
    - It is a transcompiler.  
    - It compiles the TypeScript code into JavaScript.  
    - So that you can use JavaScript in a webpage.  
  
3. Language Service [service.ts]  
    - Service is a pre-defined business logic.  
    - It provides data and funcitonality which you can inject into any  
      application.  
    - It provides library for TypeScript.  
  
4. TsServer [server.ts]  
    - Hosting  
    - Listening  
    - Handles Request and Response  
    - Ts Server  
        a) host typescript services  
        b) compiles typescript programs  
        c) handles all interactions  
  
5. VS Shim [shims.ts]  
    - Shimmer makes the code platform neutral  
    - They generate Managed Code.  
    - Managed Code is the code understandable to every OS.  
  
    TypeScript Code => TS Core Compiler => Unmanaged Code => Shims.ts => Managed Code [Every OS]  
  
6. Managed Language Service  
    - It is platform neutral service.  
    - It provides typescript functions across all platforms.  
  
7. Editors  
    - TypeScript editor configuration.  
    - It allows to configure rules for editor  
    - tsconfig.json  
    - tsLint.json [obsolete]  
  
  
Setup Environment for TypeScript  
----------------------------------------------  
1. Download and Install "NodeJS" on your PC  
    - Installing Node JS will provide a package manager called NPM.  
            Yarn  
            Bower  
            Grunt  
            RubyGems  
            NuGet  
            NPM etc..  
  
      [www.nodejs.org](http://www.nodejs.org/" \t "https://classroom.google.com/c/NTU2MTgyMTg5NzM0/m/NTU2MjE2MzEzODI4/_blank)  
  
  
2. Test Node JS and NPM Version  
  
    C:\> node  -v        16+  
    C:\> npm -v            8+  
  
   Node JS version 14+  
   NPM Version  6+  
  
3. Install TypeScript on your PC using NPM  
  
     C:\> npm install -g typescript  
  
      C:\> tsc   -v  
  
4. Download and Install :  Visual Studio Code

[](https://drive.google.com/file/d/1gWpfiaILaMuajlflJcXPUT1rCZ3GWQLG/view?usp=drive_web&authuser=0" \o "tsArchitecture.png" \t "https://classroom.google.com/c/NTU2MTgyMTg5NzM0/m/NTU2MjE2MzEzODI4/_blank)

[tsArchitecture.png](https://drive.google.com/file/d/1gWpfiaILaMuajlflJcXPUT1rCZ3GWQLG/view?usp=drive_web&authuser=0" \o "tsArchitecture.png" \t "https://classroom.google.com/c/NTU2MTgyMTg5NzM0/m/NTU2MjE2MzEzODI4/_blank)

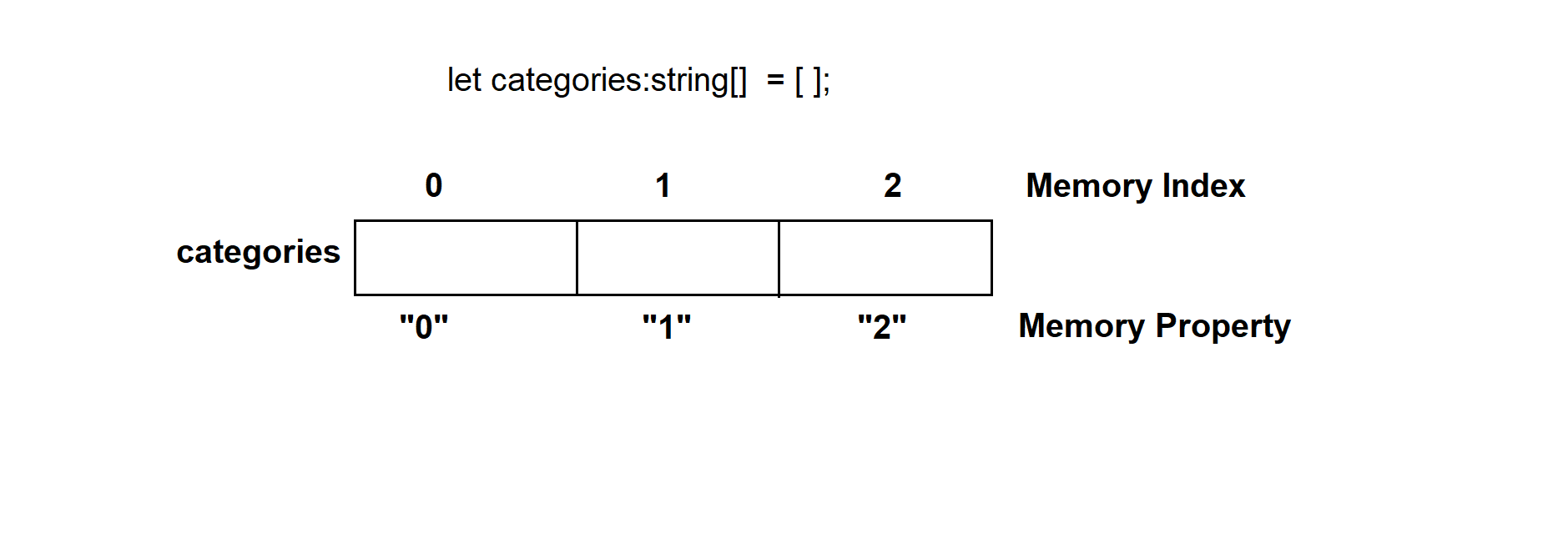
[Image](https://drive.google.com/file/d/1gWpfiaILaMuajlflJcXPUT1rCZ3GWQLG/view?usp=drive_web&authuser=0" \o "tsArchitecture.png" \t "https://classroom.google.com/c/NTU2MTgyMTg5NzM0/m/NTU2MjE2MzEzODI4/_blank)

Setup Environment for TypeScript  
1. Install Node JS  
     NPM  
  
2. Install TypeScript  
  
    C:\>npm install -g typescript  
  
3. Download and Install Editor : Visual Studio Code  
  
  
4. Install following extentions into VS Code  
  
        a) Live Server  
        b) vscode-icons  
  
  
            Create a Web Application with TypeScript  
  
1. Create a new folder for project on your PC  
  
            E:\TypeScript-Project  
  
2. Open Folder in VS Code  
  
3.  Open Project Terminal  [ Terminal Menu => New Terminal ]  
            (or)  
        Ctrl + `     (backtick)  
  
4. Change your terminal from Power Shell to Command Prompt.  
  
5.  Run the command  
  
                > npm  init -y  
  
6. It generates "package.json"  
  
FAQ: What is Package.json?  
Ans : It comprises of project meta data.  
        Information about project like  
            a) Project Name  
            b) Version  
            c) Dependencies etc..  
  
7.  Generate "tsconfig.json" by using following command  
  
        > tsc  -init  
  
FAQ: What is "tsconfig.json"?  
Ans : If configures the rules for typescript used in project.  
        It includes details like  
        a) Target JavaScript Version  
        b) Which module system to use: [CommonJS, UMD, AMD]  
        etc..  
  
8. Add following folders into project  
  
    a) public        : It comprises of static resources like : .html, .txt, .jpg..  
    b) src            : It comprises of dynamic resources like  
                        .css, .scss, .js, .ts etc..  
  
9. Every project starts with "index.html".  
  
10. Design Index.html page  
  
11. Click "Go Live"  
  
12. You can request from browser  
  
        [http://127.0.0.1:5500](http://127.0.0.1:5500/" \t "https://classroom.google.com/c/NTU2MTgyMTg5NzM0/m/NTU2NTA0Nzk4NDA0/_blank)  
                (or)  
        [http://localhost:5500](http://localhost:5500/" \t "https://classroom.google.com/c/NTU2MTgyMTg5NzM0/m/NTU2NTA0Nzk4NDA0/_blank)  
  
13.  Add a new file into "src" folder  
              Index.ts  
  
function bodyload(){  
    var username:string = "David";  
    document.querySelector("p")?.innerHTML = `Hello ! ${username} Welcome to TypeScript`;  
}  
  
14. Open Terminal and trans compile typescript into JavaScript  
  
            > tsc  index.ts  
  
15.  IT generates "index.js"  
  
16. Link index.js to HTML page "Index.html"  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>Index</title>  
    <script src="src/index.js"></script>  
</head>  
<body onload="bodyload()">  
    <h2>Index Page</h2>  
    <p></p>  
</body>  
</html>  
  
  
                         TypeScript Language Basics  
1. Variables  
2. Data Types  
3. Operators  
4. Statements  
  
  
                            Variables in TypeScript  
- Variables are storage locations in memory where you can store a value and use it as a part of any expression.  
  
- TypeScript variable configuration comprises of 3 phases  
  
        a) Declaration  
        b) Assignment  
        c) Initialization  
  
  
    var x;            => Declaring  
    x = 10;        => Assignment  
  
    var y = 20;    => Initialization  
  
- TypeScript variables are declared by using 3 keywords  
  
        a) var  
        b) let  
        c) const  
  
            var variableName;  
            let variableName;  
            const variableName;  
  
var  
- It configures function scope variable.  
- You can configure in a function and can access from any block in the function.  
- var allows  
        a) declaring  
        b) assigning  
        c) initialization  
- var allows shadowing  
- var allows hoisting  
  
let  
- It defines block scope variable.  
- It allows declaring, assigning, initialization  
- It will not allow shadowing and hoisting.  
  
const  
- It defines block scope variable  
- It allows only initialization.  
- It will not allow shadowing and hoisting.

TypeScript Language Basics  
1. Variables  
        a) var  
        b) let  
        c) const  
  
2. Data Types  
  
- Data Type defines data structure.  
- It specifies the type of data and its range.  
- JavaScript is not strongly typed.  
  
            var x = 10;        // x is number  
            x = "John";        // x is string  
  
- TypeScript is strongly typed.  
  
            var   variableName:dataType;  
  
- TypeScript follows "Duck Typing".  
  
- TypeScript Data Types are classified into 2 groups  
  
        a) Primitive Types  
        b) Non-Primitive Types  
  
  
                                    Primitive Types  
- Primitive types are immutable types.  
- They have a fixed range for values.  
- Value range can't change.  
- They are stored in a memory stack.  
- Stack uses "LIFO".  
- TypeScript Primitive Types are  
  
        a) number  
        b) string  
        c) boolean  
        d) null  
        e) undefined  
  
Number:  
- It refers to numeric value  
        signed integer                -10  
        unsigned integer            10  
        floating point                34.00  
        double                        340.402  
        decimal                        2400.00  
        binary                            01010  
        hexa                            09a-f  
        octa                            0o493  
        exponent                        2e3  
  
Syntax:  
            let  Price:number;  
            Price = 4500.44;                            // Assignment  
  
            const  Price:number = 45000.44;        // Initialization  
  
Ex:  
let Price:number;  
Price = 2e3;  
console.log("Price=" + Price);  
  
String  
- It is literal which can be enclose in  
        a) double quotes  
        b) single quotes  
        c) back tick  
  
Syntax:  
            let  link:string = "<a href='home.html'> Home </a>";  
            let  link:string = '<a href="home.html"> Home </a>';  
  
- Back tick represents a string with embedded expression.  
- Expression is defined by using "${  }"  
  
Syntax:  
          let UserName:string = "John";  
          let Age:number = 22;  
  
          let msg = `Hello ! ${UserName} you will be ${Age+1} next year.`;  
  
Ex:  
let UserName:string = "John";  
let Age:number = 22;  
  
let msg1:string = "Hello !" + " " + UserName + " " + "you will be" + " " + (Age+1) + " " + "next year.";  
  
let msg2:string = "Hello ! ${UserName} you will be ${Age+1} next year.";  
  
console.log(msg1);  
console.log(msg2);  
  
- Several chars in a string can escape printing.  
- To print non-printable chars you need escape sequence character "\".  
  
Syntax:  
            let  photo:string = "D:\images\pic.jpg";  
  
            D:imagespic.jpg  
  
            let  photo:string = "D:\\images\\pic.jpg";  
  
            D:\images\pic.jpg  
Ex:  
  
let photo:string = "\"D:\\Pictures\\flower.jpg\"";  
console.log(photo);  
  
  
  
Note: TypeScript supports "Union of Types"  
  
let value:string|number;  
value = "John";  
value = 10;  
  
Note:  TypeScript support "Type Inference"  
         It is a technique where data type is determined according to value initialized.  
  
                let x  = 10;            // x is number  
                x = "A";                // invalid  
  
        If data type is not defined and value is not initialized, then the data type  
        is defined as "any".  
  
Boolean:  
- true or false

TypeScript Data Types  
1. Primitive Types  
    a) Number  
    b) String  
    c) Boolean  
  
    Union of Types                  
    Type Inference  
  
    d) null  
    e) undefined  
  
                                        Null Type  
  
- It is used to verify value during run time of application.  
- It value is not supplied into variable then it returns null.  
  
Syntax:  
        let  username:string = "john";                                // valid  
        let  username:string = "";                                        // valid  
        let  username:string = prompt("Enter Name");            // invalid  
        let  username:string|null = prompt("Enter Name");        // valid  
  
Ex:  
Index.ts  
  
  
var username:string|null = prompt("Enter Name");  
if(username==null) {  
    document.write("You canceled..");  
} else {  
    document.write("Hello ! " + username);  
}  
  
> tsc  index.ts  
  
Index.html  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>Index</title>  
    <script src="src/index.js"></script>  
</head>  
<body>  
</body>  
</html>  
  
                                     Undefined Type  
- It specifies that the value is not default into variable and will be defined later when ever required.  
- It is marked for further assignment.  
  
Syntax:  
        let Price:number;  
        console.log("Price=" + Price);            // invalid - not assigned  
  
        let Price:number|undefined;  
        console.log("Price=" + Price);            // valid - store "undefined"  
  
Ex: Index.ts  
  
let Name:string = "Samsung TV";  
let Price:number|undefined;  
Price=46000.44;  
if(Price==undefined) {  
    console.log(`Name=${Name}`);  
}else {  
console.log(`Name=${Name}\nPrice=${Price}`);  
}  
  
> tsc index.ts  
> node index.js  
  
Summary  
- Primitive Types  
- Number  
- String  
- Boolean  
- Null  
- Undefined  
  
                                Non-Primtive Types  
- They are mutable types.  
- They can change the structure according to state and situation.  
- There is no fixed range for values.  
- Value range changes according to state and situation.  
- TypeScript Non-Primitives types  
        a) Array  
        b) Object  
        c) Map  
  
  
                                      Array Type  
- Arrays are used in computer programming to reduce overhead and complexity.  
- Array are introduced into computer programming to reduce overhead.  
- Array can reduce overhead by storing values in sequential order.  
- Array can store various types of values.  
- But few technologies restrict array for similar type of values.  
- TypeScript allows various types.  
- TypeScript can also restrict similar types.  
- Array can change size dynamically.  
  
Configuring Array:  
  
    let  variableName:string[];                [ ] => meta character, range of values  
    let  variableName:any[];  
  
  
Note: Data can't be stored into array. To store data into array you have to  
        assign or initialize memory.  
        Memory can be initialized or assigned for array by using 2 techniques  
                a) [ ]  
                b) Array()  
  
    let   categories:string[];                //configuring  
    caregories = [ ];                        //assigning memory  
  
    let categories:string[] = [ ]            // initialization of memory  
  
                                    (or)  
    let categories:string[];  
    categories = new Array();  
  
    let categories:string[] = new Array();  
  
FAQ: What is difference between  [ ] and Array() ?  
Ans:  [ ] allows to initialize different types of values when data type is "any".  
  
        Array() will not allows different types of values to initialize even when the type is "any".  
  
    Syntax:  
        let categories:any[] = [10, "A", true];                // valid  
  
        let categories:any[] = new Array(10, "A");        // invalid - only numeric

TypeScript Data Types  
- Primitive Types  
- Non Primitive Types  
     
                                        Array  
- Reducing overhead and complexity.  
- Store values in sequential order  
- Store multiple values under specified name.  
- Declaring Array  
- Initialization and assignemnt of memory  
- [ ]  & Array()  
  
Storing Values into Array:  
  
- You can initialize values into Array.  
- You can assign values into Array.  
- Values are assigned by using Array Property Name  
- Every property is a string type mapping to index in memory which is number.  
  
Syntax:  
            let categories:any[] =  [ ];  
            categories[0] = 10;                // valid  
            categories["1"] = 20;                // valid  
  
            console.log(categories[0]);        // valid  
            console.log(categories["1"]);    // valid  
Ex:  
  
let categories:any[] = [];  
categories[0] = "Electronics";  
categories[1] =  10;  
categories[2] =  true;  
categories["3"] = "Footwear";  
  
for(var property in categories){  
    console.log(`${property} [${typeof property}] - ${categories[property]} (${typeof categories[property]})`);  
}  
  
console.log(categories["1"]);  
  
- You can store any type of value into array memory, it can be  
        primitive type  
        non-primitive type  
        function  
  
Ex:  
  
let categories:any[] = [1,"TV",true,["Delhi","Hyd"], function(){console.log("Hello ! from Array")}];  
console.log(categories[3][1]);  
categories[4]();  
  
- TypeScript allows to access array elements using "Destructuring Technique".  
  
Ex:  
  
let categories:any[] = [1,"TV",true,["Delhi","Hyd"], function(){console.log("Hello ! from Array")}];  
  
let [id, Name, stock, citites, hello] = categories;  
  
console.log(Name);  
hello();  
  
Reading Elements from Array  
----------------------------------------  
toString()                    returns all array elements separated with ","  
join()                        returns all elements separated with custom symbol.  
slice()                        returns elements between specified index reference  
  
Ex:  
let categories:string[] = ["Electronics", "Footwear", "Fashion"];  
console.log(categories.toString());  
console.log(categories.join("-->"));  
console.log(categories.slice(1,2));  
  
find()                        It returns the first occurance element in array that  
                            matches the given condition.  
  
filter()                        It is same like find but can return all elements that match  
                            given condition.  
  
Ex:  
  
let sales:number[] = [34000,56000,36000,52000,67000];  
let result:number|undefined = sales.find(function(value){  
   return value > 50000;  
});  
console.log(result);  
  
Ex:  
  
let sales:number[] = [34000,56000,36000,52000,67000];  
let result:number[]|undefined = sales.filter(function(value){  
   return value > 50000;  
});  
console.log(result);  
  
- for..of iterator            : It is used to read and return all values from array.  
- for..in iterator            : It is used to read and return all properties from array.  
- for loop                    : It is used to read using initializer, condition and counter.  
  
Ex:  
let categories:string[]= ["Electronics", "Footwear", "Fashion"];  
  
console.log(`----Using Loops------`);  
for(var i=0; i<categories.length;i++) {  
    console.log(categories[i]);  
}  
  
console.log(`----Using Iterator-----`);  
for(var item of categories)  
{  
    console.log(item);  
}  
  
console.log(`---Iterator for Properties----`);  
for(var property in categories){  
    console.log(`${property} : ${categories[property]}`);  
}  
  
  
Adding new Elements into Array:  
push()                It adds a new element(s) as last item(s)  
unshift()                It adds a new element(s) as first item(s)  
splice()                It adds a new element(s) at specified location.  
                         
Syntax:  
    array.splice(indexToAdd, countOfItemsToDelete, "new Items...");  
  
Ex:  
let categories:string[]= ["Electronics", "Footwear", "Fashion"];  
categories.splice(1,0,"Bike Accessories","Car Accessories");  
for(var property in categories){  
    console.log(`[${property}] ${categories[property]}`);  
}  
  
Removing Elements from Array:  
pop()                : It removes last item.  
shift()                : It removes first item.  
splice()            : It removes specified item.  
  
Ex:  
let categories:string[]= ["Electronics", "Footwear", "Fashion"];  
categories.splice(1,1);  
for(var property in categories){  
    console.log(`[${property}] ${categories[property]}`);  
}  
  
Sorting Elements  
sort()                    : Ascending order  
reverse()                : It is reverse order of how they are arranged.  
  
let categories:string[]= ["D","A","C","B"];  
categories.sort();  
categories.reverse();  
for(var property in categories){  
    console.log(`[${property}] ${categories[property]}`);  
}

[](https://drive.google.com/file/d/1s01IhPmeHVec1LIRGQ4SNyd0y46W-JMk/view?usp=drive_web&authuser=0" \o "array1.png" \t "https://classroom.google.com/c/NTU2MTgyMTg5NzM0/m/NTYxMTM1NTY1NTY3/_blank)

Object Type  
- Object is used to keep all related data and logic together.  
- Alan Kay in early 1960's introduced the concept of "Object" in computer programming.  
- Johan Olay, Kristian Nygaard introduced reusability of object, they introduced the first OOP language called "SIMULA".  
- TypeScript uses JSON technique.  
  [JavaScript Object Notation]  
- Object is a set of Keys and Values  
  
            {  
              Key:value,  
              Key:value  
            }  
  
- Key is always string type.  
- Value can be any type.  
    a) Primitive  
    b) Non-Primitive  
  
Syntax:  
        {  
          Name:string,  
          Price:number,  
          Stock:boolean,  
          Cities:string[],  
          Rating: { Rate:number, Count:number }  
        }  
  
Ex: index.ts  
  
let product:{Name:string, Price:number, Stock:boolean, ShippedTo:string[], Rating:{Rate:number, Count:number}};  
product = {  
    Name: "Samsung TV",  
    Price: 34000.44,  
    Stock: true,  
    ShippedTo: ["Delhi", "Hyd"],  
    Rating: {Rate:3.5, Count:5600}  
}  
console.log(`Name=${product.Name}\nPrice=${product.Price}\nRating=${product.Rating.Rate} [${product.Rating.Count}]\nShipped To=${product.ShippedTo.toString()}\nStock=${(product.Stock==true)?"Available":"Out of Stock"}`);  
  
                                    Array of Objects  
                                    ----------------------  
- Array that can handle multiple objects.  
  
Syntax:  
        let  name:[{key:dataType}];  
  
Ex: Index.ts  
let products:[{Name:string, Price:number}];  
products = [  
    {  
        Name: "Samsung TV",  
        Price: 45000.44  
    },  
    {  
        Name: "Mobile",  
        Price: 23000.44  
    }  
];  
console.log(`Mobile Price : ${products[1].Price}`);  
  
                                        Date Type  
- Date is stored and configure by using "Date" class.  
  
Syntax:  
    let  mfd:Date  = new Date();                                        // load current date  
    let  mfd:Date  = new Date("YY-MM-DD HR:MIN:SEC")    // load specific  
  
- Date and Time can be accessed by using JavaScript date methods.  
    getHours()                    : Hour Number in 24 hr format  
    getMinutes()                : 0 to 59  
    getSeconds()                : 0 to 59  
    getMilliSeconds()            : 0 to 99  
    getDay()                        : 0 to 6    => 0 = Sunday  
    getDate()                    : 1 to 31, 30, 29, 28  
    getMonth()                    : 0 to 11  => 0 = January  
    getFullYear()                : 2022  
    toLocaleDateString()  
    toLocaleTimeString()  
  
- JavaScript allows to set a new date or time by using methods  
    setHours()  
    setMinutes()  
    setSeconds()  
    setMilliSeconds()  
    setDate()  
    setMonth()  
    setYear()  
  
Summary : TypeScript Data Types  
- number  
- string  
- boolean  
- null  
- undefined  
- array  
- object  
- array of object  
- date  
  
                                    TypeScript Operators  
- All are same as JavaScript operators  
    Arithematic  
    Logical  
    Assignment  
    Comparision  
    Bitwise  
    Special Operators  
        [delete, in, of, new, typeof, instanceof, yeild]  
  
                                TypeScript Statements  
- Selection Statements  
    if, else, switch, case, default  
- Looping Control  
    for, while, do while  
- Iteration Statements  
    for..in, for..of  
- Jump Statements  
    break, continue, return  
- Exception Handling  
    try, catch, throw, finally  
  
     
                                 TypeScript OOP  
- Real world application development used 3 types of programming systems  
    a) POPS  
    b) OBPS  
    c) OOPS  
  
- Process Oriented Programming System  
    \* It supports low level features  
    \* It can directly interact with hardware services  
    \* It uses less memory  
    \* It is faster  
     
    Ex: C, Pascal, COBOL  
  
     \* Code reusability issue  
    \* Code separation issue  
    \* No dynamic memory  
    \* Extensibility issues  
  
- Object Based Programming System  
    \* It supports code reusability  
    \* It supports separation  
    \* It support extensibility [limited]  
    \* Dynamic memory  
  
    Ex: JavaScript, Visual Basic  
  
    \* No dynamic polymorphism  
    \* No code level security  
    \* Issues in handling dynamic memory  
  
- Object Oriented Programming System  
    \* Supports reusability  
    \* Separation  
    \* Extensibility  
    \* Security  
    \* Dynamic Memory  
  
    Ex: C++ , Java, C#, TypeScript etc..  
  
    \* Not support low level features  
    \* Can't directly interact with hardware  
    \* Slow  
    \* More memory  
    \* Complex

TypeScript OOP  
- POPS  
- OBPS  
- OOPS  
  
Features of OOP:  
- Code reusability  
- Code separation  
- Extensibility  
- Code Security  
- Dynamic memory  
  
Characterstics of OOP:  
- Inheritance  
- Polymorphism  
- Abstraction  
- Encapsulation  
  
Drawbacks of OOP:  
- Tidious  
- Complex  
- More memory  
- Slow  
- Don't support low level features  
- Can't directly interact with hardware  
  
  
                                Contracts in OOP  
- Contract defines rules for designing a component.  
- Contracts are designed by using "interface".  
- Every rule in contract is mandatory.  
- You can't ignore any specified rule or add new functionality which is not defined in contract.  
  
Syntax:  
    interface  IName  
    {  
        //rules  
    }  
  
- Interface can contain only rules and implementation.  
  
Syntax:  
    interface  IProduct  
    {  
      Name:string = "TV";        // invalid  
      Name:string;                // valid  
    }  
  
  
Ex:  
interface IProduct {  
    Name:string;  
    Price:number;  
    Stock:boolean;  
}  
let product:IProduct = {  
    Name: "Samsung TV",  
    Price: 34000.33,  
    Stock: true,  
    Rating: 4.5                    // invalid - not in contract  
}  
  
- TypeScript contracts can define optional rules  
- Optional rules are defined by using null reference character "?".  
  
Syntax:      
interface IProduct  
{  
  Name:string;  
  Price:number;  
  Rating?:number;                // optional  
}  
  
FAQ: Why you need optional rules in contract?  
Ans : To configure goal of module.  
  
FAQ: What are the mandatory rules in contract?  
Ans : They are module objective.  
  
- Every rule defined in contract will allow assignment of values.  
- You can re-assign value any number of times.  
- You restrict assignment of value after initialization by using "readonly".  
- Contract can have readonly rules.  
  
Syntax:  
interface IProduct  
{  
 Name:string;  
 readonly Price:number;  
}  
  
Ex:  
interface IProduct {  
    Name:string;  
    readonly Price:number;  
    Stock:boolean;  
}  
let tv:IProduct = {  
    Name: "Samsung TV",  
    Price: 34000.33,                // intialized  
    Stock: true  
}  
tv.Price = 56000.44;            // invalid - Price is readonly  
  
console.log(`Name=${tv.Name}\nPrice=${tv.Price}\nStock=${tv.Stock}`);  
  
- Contract can have rules not only for property but also for methods.  
- Method rule comprises of method name and its return type.  
  
                Total():number  
                Print():void  
  
- Method rule can have only declaration not definition.  
  
Ex:  
interface IProduct {  
    Name:string;  
    readonly Price:number;  
    Qty:number;  
    Total():number;  
    Print?():void;  
}  
let tv:IProduct = {  
    Name: "Samsung TV",  
    Price: 34000.33,    
    Qty: 2,  
    Total() {  
        return this.Qty \* this.Price;  
    },  
    Print(){  
        console.log(`Name=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${this.Total()}`);  
    }  
}  
tv.Print();  
  
  
- The type of property or methods can be primitive or non primitive.  
  
Ex:  
interface IProduct {  
    Name:string;  
    readonly Price:number;  
    Qty:number;  
    Cities:string[];  
    Rating:{Rate:number, Count:number};  
    Mfd:Date;  
    Total():number;  
    Print?():void;  
}  
let tv:IProduct = {  
    Name: "Samsung TV",  
    Price: 34000.33,    
    Qty: 2,  
    Cities: ["Delhi", "Hyd"],  
    Rating: {Rate: 4.3, Count:3400},  
    Mfd: new Date("2022-10-22"),  
    Total() {  
        return this.Qty \* this.Price;  
    },  
    Print(){  
        console.log(`Name=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${this.Total()}`);  
    }  
}

Contracts in OOP  
            ------------------------  
- interface  
- rules for property  
- rules for method  
- optional rules [?]  
- readonly rules  
  
                                Extending Rules  
- You can extend a contract without disturbing existing contract.  
- TypeScript uses "extends" keyword.  
  
Syntax:  
        interface old\_version  
        {  
        }  
        interface new\_version extends old\_version  
        {  
        }  
  
- New version contract can have acess to all old\_version rules.  
- This refers to "Extensibility" of contract, which is technically known as  
        "Inheritance"  
- Inheritance offers extensibility and reusability.  
  
FAQ: Why to extend a contract without modifying existing contract?  
Ans:  To handle  
            a) Side By Side Execution  
            b) Backward Compatibility  
  
FAQ: What is side-by-side execution?  
Ans : Allowing multiple versions of same software to run on a device is  
        known as side-by-side execution.  
  
Ex:  
interface HDFCBank\_Version1  
{  
    Personal:string;  
    NRI:string;  
}  
interface HDFCBank\_Version2 extends HDFCBank\_Version1 {  
    Loans:string;  
}  
  
let hdfc\_latest:HDFCBank\_Version2 = {  
    Personal: "Personal Banking - General Customers",  
    NRI: "NRI Banking - NRI Customers",  
    Loans: "Loans - Car , Bike Loans"  
}  
  
let hdfc\_legacy:HDFCBank\_Version1 = {  
    Personal: "Personal Banking - General Customers",  
    NRI: "NRI Banking - NRI Customers",  
}  
  
- Contract supports all types of inheritance  
        a) Single  
        b) Multi level  
        c) Multiple  
  
  
Ex:  
interface HDFC\_Name {  
    BankName:string;  
}  
interface HDFCBank\_Version1  
{  
    Personal:string;  
    NRI:string;  
}  
interface HDFCBank\_Version2 extends HDFCBank\_Version1, HDFC\_Name {  
    Loans:string;  
}  
  
let hdfc\_latest:HDFCBank\_Version2 = {  
    Personal: "Personal Banking - General Customers",  
    NRI: "NRI Banking - NRI Customers",  
    Loans: "Loans - Car , Bike Loans",  
    BankName: "HDFC Bank"  
}  
  
let hdfc\_legacy:HDFCBank\_Version1 = {  
    Personal: "Personal Banking - General Customers",  
    NRI: "NRI Banking - NRI Customers",  
}  
  
  
                                     Class in OOP  
- Class is a program template.  
- A template comprises of sample data and logic, which you can implement and customize according requirements.  
- If a class is designed for mapping to data requirements then it is known as "Model".  
- If a class is designed for business requirements then it is known as "Entity".  
- Class is also known as a blue print in general behaviour.  
- TypeScript a class is defined by using "class" keyword and it have 2 types of declaration techniques  
  
            a) class definition  
            b) class expression  
  
Class Definition  
  
Syntax:  
            class  Product  
            {  
            }  
  
 - Definition is for Lazy Loading of class.  
  
Class Expression  
  
Syntax:  
        let  Employee = class {  
  
        }  
  
- Expression is for Eager loading of class.  
- It is anonymous class.  
- class without name.  
  
                                    Class Members  
- Every typescript class can have only following members  
    a) Property  
    b) Accessor  
    c) Method  
    d) Constructor  
  
FAQ: Can we declare variable as class member?  
Ans : No.  
  
FAQ: Why variable not allowed as class member?  
Ans: Variables are immutable. And class can't have immutable members.  
  
FAQ: can we have a variable in class?  
Ans : Yes. But as a member of method not as class.  
  
FAQ: can we define a function as class member?  
Ans: No.  
  
FAQ: can we define function in class?  
Ans: Yes.  
  
FAQ: What is difference between property and variable?  
Ans:  Property is mutable and variable is immutable.  
           
Ex: Index.ts  
  
let username:string|null = prompt("Enter UserName");  
let role:string|null = prompt("Enter Your Role");  
let productname:string|null = prompt("Enter Product Name");  
  
class Product  
{  
    \_productName:string|undefined|null;  
  
    get ProductName(){  
        return this.\_productName;  
    }  
  
    set ProductName(newName:string|undefined|null){  
        if(role=="admin") {  
            this.\_productName = newName;  
        } else {  
            document.write(`${username} - You are not authorized to set Product Name`);  
        }  
    }  
}  
let tv = new Product();  
tv.ProductName = productname;  
if(tv.ProductName){  
    document.write(`Name=` + tv.ProductName);  
}  
  
  
> tsc index.ts  
  
Index.html  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>Index</title>  
    <script src="src/index.js"></script>  
</head>  
<body>  
</body>  
</html>  
  
  
- Property  
- Accessor

25/10/2022

TypeScript OOP

- Contracts [Interface]  
- Class  
    a) Class Members  
            Property  
            Method  
            Constructor  
            Accessor  
  
  
                                    Class Property  
- Property describes data.  
- In a class data is stored in property.  
- Property is mutable.  
  
Syntax:  
    class  className  
    {  
        property:dataType;  
    }  
  
                                     Accessors  
  
- Accessor gives a fine grained control over property.  
- It allows or restricts read and write operations on property.  
- Accessors are 2 types  
        a) getter  
        b) setter  
- getter is used to read a value from property.  
- setter is used to store value into property.  
  
Syntax:  
        get aliasName()  
        {  
          return value;  
        }  
  
        set aliasName(newValue)  
        {  
          value = newValue;  
        }  
  
Ex:  
let username:string|null = prompt("Enter UserName");  
let role:string|null = prompt("Enter Your Role");  
let productname:string|null = prompt("Enter Product Name");  
  
class Product  
{  
    \_productName:string|undefined|null;  
  
    get ProductName(){  
        return this.\_productName;  
    }  
  
    set ProductName(newName:string|undefined|null){  
        if(role=="admin") {  
            this.\_productName = newName;  
        } else {  
            document.write(`${username} - You are not authorized to set Product Name`);  
        }  
    }  
}  
let tv = new Product();  
tv.ProductName = productname;  
if(tv.ProductName){  
    document.write(`Name=` + tv.ProductName);  
}  
  
Index.html  
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>Index</title>  
    <script src="src/index.js"></script>  
</head>  
<body>  
</body>  
</html>  
  
- Accessors are also used to access any property from a multi level hierarchy.  
  
Ex:  
class Product  
{  
    Name:string = "Samsung TV";  
    Price:number = 35000.55;  
    Rating:{VendorRating:{Rate:number, Count:number}, CustomerRating:{Rate:number, Count:number}} = {  
        VendorRating: {Rate:5.0, Count:60},  
        CustomerRating: {Rate:4.3, Count:6200}  
    };  
    get VendorRating(){  
        return this.Rating.VendorRating.Rate;  
    }  
}  
let tv = new Product();  
console.log(`Vendor Rating : ${tv.VendorRating}`);  
console.log(`Customer Rating: ${tv.Rating.CustomerRating.Rate}`);  
  
  
                            Static and Non Static Members  
  
- A property or method can be static or non-static in TypeScript.  
- The terms static and non-static refers to memory.  
  
Static  
- Static refers to continous memory.  
- Memory allocated for first object will continue for next.  
- It uses more memory  
- It is not safe.  
- Static members in class are declared by using "static" keyword.  
- They are accessed within or outside class by using class name.  
  
Syntax:  
        class Demo  
        {  
         static s = 0;  
        }  
  
Non-Static or Dynamic  
- It refers to discreet memory. [disconnected]  
- Memory is newly allocated for every object.  
- Memory is destroyed after every object.  
- It provides disconnected access to data.  
- Non static members are accessed with in class by using "this" keyword and outside class by using instance of class.  
  
Syntax:  
        class Demo  
        {  
          n = 0;  
        }  
  
Ex: Index.ts  
  
class Demo  
{  
    static s = 0;  
    n = 0;  
    constructor(){  
        Demo.s=  Demo.s + 1;  
        this.n = this.n + 1;  
    }  
    Print(){  
        console.log(`s=${Demo.s} n=${this.n}`);  
    }  
}  
let obj1 = new Demo();  
obj1.Print();  
  
let obj2 = new Demo();  
obj2.Print();  
  
let obj3 = new Demo();  
obj3.Print();  
  
FAQ: When to configure static members?  
Ans: When you need continous operations. Connected Access  
  
FAQ: When to configure non-static members?  
Ans: Whe you need discreet operations. Disconnected Access  
  
     
                                 Access Modifiers  
- Access modifier sets code level scope restrictions.  
- TypeScript access modifiers are  
    a) public  
    b) private  
    c) protected  
  
public  
    - It is accessible with in the class or outside class.  
    - It is accessible outiside class by using a dervied class object or super class  
      object.  
    - It is accessible in derived class or outside derived class.  
  
private  
    - It is accessible only with in class.  
  
protected  
    - It is accessible with in class.  
    - It is accessible to derived class only by using derived class object | refernce.  
  
Ex:  
  
class SuperClass  
{  
    public Name:string = "TV";  
    private Price:number = 45000.33;  
    protected Stock:boolean = true;  
}  
class DerivedClass extends SuperClass  
{  
    Print(obj:DerivedClass){  
        obj.Name;  
        obj.Stock;  
    }  
}  
let obj = new DerivedClass();  
obj.Name;  // public  
  
  
                                   Constructor  
- A constructor in OOP is used for instatiation.  
- It is used for constructing a object for class.  
- Constructor is a design pattern, which comes under "Creational Patterns".  
- Creational patterns in Software Engineering deals with creation of object.  
- The various creational patterns  
        - Abstract Factory  
        - Builder  
        - Single Ton etc..  
- If you wan any action to perform at the time of creating object then provide that functionality to constructor.  
  
Ex:  
class Database  
{  
    constructor(){  
        console.log(`Connected with Database`);  
    }  
    Insert(){  
        console.log(`Record Inserted`);  
    }  
}  
let oracle = new Database();  
oracle.Insert();

26/10/2022

Class Members  
- Property  
- Accessor  
- Method  
- Constructor  
- Static and Non Static  
- Access Modifiers  
  
                                        Constructor  
- It is used for instantiation. [Creating Object]  
- It is a design pattern. [Creational Patterns]  
- Defines actions to perform while creating an object for class.  
- TypeScript constructor is anonymous. [Without Name]  
- It can't be private or protected.  
- It can't overload.  
- It can be parameterized or parameter less.  
- It can have optional parameters.  
  
Ex:  
class Database  
{  
    constructor(dbname:string, UserName:string){  
        console.log(`Connected to ${dbname} as User ${UserName}`);  
    }  
    public Insert():void {  
        console.log("Record Inserted..");  
    }  
}  
let oracle = new Database("Oracle","Scott");  
oracle.Insert();  
  
                                       Inhertiance  
- Extensibility is defined by using Inheritance.  
- Class can implement contract and extend another class.  
  
Ex: Implementation of Contracts  
  
interface ProductContract  
{  
    Name:string;  
    Price:number;  
    Qty:number;  
    Total():number;  
    Print():void;  
}  
interface CategoryContract  
{  
    CategoryName:string;  
}  
class ProductComponent implements ProductContract, CategoryContract  
{  
    public Name: string = "Samsung TV";  
    public Price: number = 24000.44;  
    public Qty: number = 2;  
    public CategoryName: string = "Electronics";  
    public Total():number {  
        return this.Qty \* this.Price;  
    }  
    public Print(): void {  
        console.log(`  
            Name : ${this.Name} \n  
            Price: ${this.Price} \n  
            Qty  : ${this.Qty} \n  
            Total: ${this.Total()} \n  
            Category: ${this.CategoryName}  
        `)  
    }  
}  
let obj = new ProductComponent();  
obj.Print();  
  
Extending Classes  
- It is the process of creating a new class and extending the features of existing class.  
- Existing class is known as Super Class  
- New Class is known as Derived Class  
- "extends" is the keyword used for Inheritance.  
  
Syntax:  
  class  SuperClass  
  {  
  }  
  class  Derived extends SuperClass  
  {  
  }  
  
- You can access the members of super class in derived class by using "super" keyword.  
- Inheritance relation is known as "Is-a-Relation".  
  
  
Ex: Index.ts  
class HDFCBank\_Version1    
{  
    public Personal:string = "Personal Banking Services - Indian Customers";  
    public NRI:string = "NRI Banking Services - NRI Customers";  
    public Print():void {  
         
        document.write(`${this.Personal}<br>${this.NRI}<br>`);  
    }  
}  
class HDFCBank\_Version2 extends HDFCBank\_Version1  
{  
    public Loans:string = "Loans - Personal, Car, Bike, House";  
    public Print(): void {  
         
        super.Print();  
        document.write(`${this.Loans}<br>`);  
    }  
}  
class HDFCBank\_Version3 extends HDFCBank\_Version2  
{  
    public Govt:string = "Govt Schemes - Indian Govt. Schemes";  
    public Print(): void {  
         
        super.Print();  
        document.write(`${this.Govt}<br>`);  
    }  
}  
  
let ver1 = new HDFCBank\_Version1();  
let ver2 = new HDFCBank\_Version2();  
let ver3 = new HDFCBank\_Version3();  
  
let enterVersion = prompt("Enter Version Number");  
switch(enterVersion){  
    case "ver1":  
      document.write("<h2>HDFC Bank Version 1 Features</h2>")  
      ver1.Print();  
      break;  
    case "ver2":  
      document.write("<h2>HDFC Bank Version 2 Features</h2>")  
      ver2.Print();  
      break;  
    case "ver3":  
      document.write("<h2>HDFC Bank Version 3 Features</h2>")  
      ver3.Print();  
      break;  
    default:  
      document.write("Please Enter ver1, ver2 or ver3 only");  
      break;  
}  
  
  
  
  
  
FAQ: What is Aggregation?  
Ans : It is the process of accessing the members of one class in another without creating any relation between classes.  
        It is reffered as "Has-a-Relation".  
        It uses "Object-to-Object" communication.  
  
                                        Inheritance Rule  
- Super class constructor must be called first followed by derived class constructor.  
- In many OOP languages super constructor is called before derived class constructor implicitly.  
- JavaScript and TypeScript can't call super constructor. hence you have  
  to explicitly call super constructor in derived class constructor.  
  
Ex:  
class SuperClass  
{  
    constructor(){  
        console.log("Super Class Constructor");  
    }  
}  
class DerivedClass extends SuperClass  
{  
   constructor(){  
    super();  
      console.log("Derived Class Constructor");  
   }  
}  
  
FAQ: Why multiple inheritance is not supported for classes?  
Ans:  Constructor Deadlock  
        Deadlock is a situation for constructors of super class, where  
        all constructors a waiting for each other to start the process.  
  
FAQ: Why multiple inheritance is supported for interface?  
Ans:  No constructor in interface, No deadlock.  
  
             
                                  Polymorphism  
  
  
Ex:  
class Employee  
{  
    public FirstName:string|undefined;  
    public LastName:string|undefined;  
    public Designation:string|undefined;  
    public Print():void {  
        document.write(`${this.FirstName} ${this.LastName} - ${this.Designation} <br>`);  
    }  
}  
class Developer extends Employee  
{  
    public FirstName: string = "Raj";  
    public LastName: string = "Kiran";  
    public Designation: string = "Developer";  
    public Role:string = "Developer Role : Building, Debug, Testing";  
    public Print(): void {  
        super.Print();  
        document.write(`${this.Role}`);  
    }  
}  
class Admin extends Employee  
{  
    public FirstName: string = "Kiran";  
    public LastName: string = "Kumar";  
    public Designation: string = "Admin";  
    public Role:string = "Admin Role : Authorization and Authentication";  
    public Print(): void {  
        super.Print();  
        document.write(`${this.Role}`);  
    }  
}  
class Manager extends Employee  
{  
    public FirstName: string = "Tom";  
    public LastName: string = "Hanks";  
    public Role:string = "Manager Role : Approvals";  
    public Designation: string = "Manager";  
    public Print(): void {  
        super.Print();  
        document.write(`${this.Role}`);  
    }  
}  
  
let employees = new Array(new Developer(), new Admin(), new Manager());  
let designation:string | null = prompt("Enter Designation");  
for(var employee of employees){  
    if(employee.Designation==designation){  
        employee.Print();  
    }  
}

27/10/2022

Polymorphism  
- Ploy means Many  
- Morphos mens Forms  
- The availablity of any component in multiple forms is reffered as Polymorphism.  
- It is a behaviour given for components, so that one component can handle multiple situations.  
- Polymorphism refers to single base class object that can use the memory of multiple derived classes.  
  
Syntax:  
    let obj = new Array(new class1(), new class2(), new class3());  
  
- It is the process of overloading memory.  
  
Ex:  
Index.ts  
  
class Employee  
{  
    public FirstName:string|undefined;  
    public LastName:string|undefined;  
    public Designation:string|undefined;  
    public Print():void {  
        document.write(`${this.FirstName} ${this.LastName} - ${this.Designation} <br>`);  
    }  
}  
class Developer extends Employee  
{  
    public FirstName: string = "Raj";  
    public LastName: string = "Kiran";  
    public Designation: string = "Developer";  
    public Role:string = "Developer Role : Building, Debug, Testing";  
    public Print(): void {  
        super.Print();  
        document.write(`${this.Role}`);  
    }  
}  
class Admin extends Employee  
{  
    public FirstName: string = "Kiran";  
    public LastName: string = "Kumar";  
    public Designation: string = "Admin";  
    public Role:string = "Admin Role : Authorization and Authentication";  
    public Print(): void {  
        super.Print();  
        document.write(`${this.Role}`);  
    }  
}  
class Manager extends Employee  
{  
    public FirstName: string = "Tom";  
    public LastName: string = "Hanks";  
    public Role:string = "Manager Role : Approvals";  
    public Designation: string = "Manager";  
    public Print(): void {  
        super.Print();  
        document.write(`${this.Role}`);  
    }  
}  
  
let employees = new Array(new Developer(), new Admin(), new Manager());  
let designation:string | null = prompt("Enter Designation");  
for(var employee of employees){  
    if(employee.Designation==designation){  
        employee.Print();  
    }  
}  
  
> tsc index.ts  
  
Index.html  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>Index</title>  
    <script src="src/index.js"></script>  
</head>  
<body>  
</body>  
</html>  
  
Summary  
- Contracts  
- Classes  
- Inhertiance  
- Polymorphism  
  
                                     Templates in OOP  
- Template comprises of sample data and logic, which you can customize and implement according to your requirements.  
- Templates are designed by using "Abstract Classes"  
- Abstract class comprises of members 2 types  
        a) Implemented with functionality      
        b) Need to implement - Not Implemented  
  
- Abstract class can contain  
        a) Incomplete methods     : abstract methods  
        b) completed methods    : non-abstract methods  
  
- If a class have atleast one abstract method then the class is marked as "abstract".  
  
- The process of hiding the data structure and providing only functionality to application is known as "Abstraction".  
  
  
Contracts => Templates => Components => Application  
  
- Template implements Contract.  
- Component extends Template.  
  
  
Ex: Index.js  
  
interface ProductContract  
{  
    Name:string;  
    Price:number;  
    Qty:number;  
    Total():number;  
    Print():void;  
}  
abstract class ProductTemplate implements ProductContract  
{  
    public Name: string = "";  
    public Price: number = 0;  
    public Qty: number = 0;  
    public abstract Total():number;  
    public abstract Print(): void;  
}  
class ProductComponent extends ProductTemplate  
{  
   Name = "Samsung TV";  
   Price = 45000.55;  
   Qty = 2;  
   Total(){  
     return this.Qty \* this.Price;  
   }  
   Print(){  
     console.log(`Name=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${this.Total()}`);  
   }  
}  
  
let tv = new ProductComponent();  
tv.Print();  
  
  
                                       Generics  
- Generic refers to "Type Safe"  
- Initially the data type is unknown.  
- According to the type of data passed into reference, it can be strongly typed.  
- In Type Script Generic can be defined for  
  
        a) Property  
        b) Method  
        c) Class  
        d) Parameters

28/10/2022

Generics  
- Type Safe  
- Strongly Typed  
- Generic can be  
    Property  
    Parameter  
    Method  
    Class  
     
                                 Distributed Computing  
  
- Distributed computing architecture allows 2 different applications running on 2 different mechines to share information.  
  
- Two different applications running on same mechine in 2 different process can share information.  
  
- There are various distributed computing technologies  
  
    CORBA            - Common Object Request Broken Architecture  
    DCOM            - Distributed Component Object Model  
    RMI                - Remote Method Invocation  
    EJB                - Enterprise Java Beans  
    Webserivces  
    Remoting  
  
- Web Service Specifications  
    a) SOAP            Consumer => XML Request  
                        Provider => XML Response  
    b) REST  
                        Consumer => Query Request  
                        Provider => XML Response | JSON  
  
    c) JSON  
                         Consumer => JSON Request  
                        Provider => JSON Response  
  
  
Ex:  
interface IProduct{  
    Name:string;  
    Price:number;  
}  
interface IEmployee{  
    FirstName:string;  
    Designation:string;  
}  
  
  
class Service<T>  
{  
    public Data:T|undefined;  
    public Print(){  
        for(var property in this.Data){  
            console.log(`${property} : ${this.Data[property]}`);  
        }  
    }  
}  
console.log(`------Product Details---------`);  
let tv = new Service<IProduct>();  
tv.Data = {Name: "Samsung TV", Price:56000.55};  
tv.Print();  
  
console.log(`------Employee Details---------`);  
let emp = new Service<IEmployee>();  
emp.Data = {FirstName:"John", Designation:"Developer"};  
emp.Print();  
  
Note: You can't perform any operation directly on Generic types.  
        You need functions.  
  
EX:  
function Sum(a:any,b:any){  
    return a + b;  
}  
  
class Demo  
{  
    public Addition<T>(a:T, b:T) {  
        return Sum(a,b);  
    }  
}  
let obj = new Demo();  
console.log(obj.Addition<number>(10,30));  
  
Note: The technologies that support Generics can have a "Generic Constructor"  
          but not TypeScript.  
  
        TypeScript can't have Generic Constructor.  
  
  
        constructor<T>()            // invalid  
        {      
        }  
  
Ex:  
interface IProduct  
{  
    Name:string;  
    Price:number;  
}  
class Service  
{  
    public GetData<T>(data:T){  
        console.log(data);  
    }  
}  
let tv = new Service();  
console.log(`-One Product--`);  
tv.GetData<IProduct>({Name: "Samsung TV", Price: 34000.44});  
console.log(`--Products List--`);  
tv.GetData<IProduct[]>([{Name:"Mobile", Price:24000.44},{Name:"Watch", Price:2400.44}]);  
  
                            Dependency Injection  
- It is a software design pattern.  
- It specifies how an object get holds of all its dependencies.  
- It comprises of 2 major components  
    a) Provider  
    b) Injector  
- Provider is responsible for locating the data in memory.  
- Injector is responsible for injecting data into component.  
  
Ex:  
class DataService{  
    public GetData(){  
  
    }  
}  
class Component  
{  
    constructor(private data: DataService){  
        console.log(data);  
    }  
    public Print(){  
        console.log(this.data);  
    }  
}  
  
Ex:  
class CaptchaService  
{  
    public GetCode(){  
        return Math.random();  
    }  
}  
  
class LoginComponent{  
    constructor(private captcha?:CaptchaService){  
  
    }  
    public PrintCode(){  
        return this.captcha?.GetCode();  
    }  
}  
let obj = new LoginComponent();  
obj.PrintCode();  
obj.PrintCode();  
  
                                   Enum, Modules

01/11/2022

Enumerations  
                                    --------------------  
- Enum is a collection of constants.  
- Enum constant can be  
    a) number  
    b) string  
    c) expression  
- Enum number type have auto implementation for values.  
- Previous value is incremented and applied to next if it is not initialized.  
- If there is no previous value then value starts with "0".  
  
Syntax:  
enum StatusCodes  
{  
    RoundTrip,                        // 0  
    NotFound = 404,  
    OK = 200,  
    Method,                            // 201  
}  
  
- Enum is accessed by reference of Key.  
  
    StatusCodes.NotFound                => 404  
  
- If Enum have string constant then auto implementation of value will not work.  
- Auto implementation works only when previous is a number.  
  
Syntax:  
enum Demo  
{  
   A = "string",  
   B,                        // invalid  
}  
  
enum Demo  
{  
 A,                            0  
 B                            1  
}      
  
- Enum can have an expression  
  
Syntax:  
enum StatusCodes  
{  
    A = 10,  
    B = 20,  
    C = A + B  
}  
console.log(`Addition=${StatusCodes.C}`);  
  
- Enum can't have any expression that returns a boolean.  
  
    A  > B        // invalid  
  
- Any operator that returns boolean can't be used in enum.  
  
FAQ: Why booleans are not supported?  
Ans :  Booleans are not constants.  
  
FAQ: What is enum reverse mapping?  
Ans : It is a technique used to access the key with reference of value.  
  
Ex:  
enum StatusCodes  
{  
   NotFound = 404  
}  
console.log(`${StatusCodes.NotFound} : ${StatusCodes[404]}`);  
  
  
                                     Module Systems  
- Module is a set of variables, functions, components, contracts and templates.  
- Modules are required to build a library for application.  
- Library defines a factory that you can import and implement in any project.  
- It enables reusability, maintainability and testability.  
- JavaScript based technologies requires module system installed  
        CommonJS  
        RequireJS  
        AMD [Asynchronous Module Distribution]  
        UMD [Universal Module Distribution]  
  
Ex: JavaScript Module System  
  
1. Add a new folder "js"  
2. Add following files  
  
            functions.js  
 export function PrintTitle(str){  
    return str;  
}  
  
            maths.js  
export function Addition(a, b){  
    return a + b;  
}  
  
            sample.html  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>Document</title>  
     
    <script type="module">  
        import { PrintTitle } from './functions.js';  
        import { Addition } from './maths.js';  
        document.querySelector("p").innerHTML = PrintTitle("Module System of JavaScript") + "<br>" + "Addition=" + Addition(20,20);  
    </script>  
</head>  
<body>  
    <p></p>  
</body>  
</html>  
  
- JavaScript module is private for all its members, in order to access them outside you have to mark as "export".  
  
            export function Name();  
            export class Name();  
            export const name = function() { };  
  
- Every module can have one default export, which loads eagerly.  
  
            export default function Name() { }  
  
- You have to import the members  
     
            import  default\_member  from  "moduleName";  
            import  { member } from "moduleName";  
            import  default\_member, { member } from "moduleName";  
  
- Script type must be "module" in order use import and export statements.

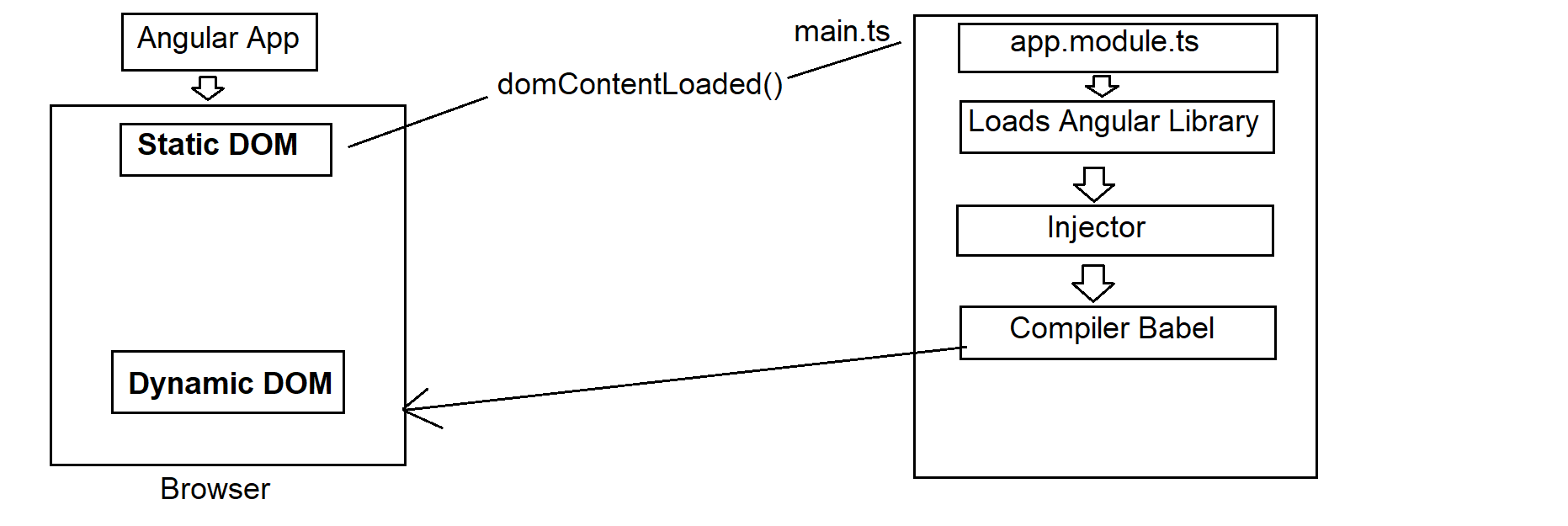
02/11/2022

Angular 14  
  
- Angular is a developers platform.  
- Developers platform provides end-to-end solution for developer.  
- It provides all tools and languages that are required for  
    a) Building  
    b) Debug  
    c) Testing  
    d) Deploying  
- Google introduced Angular as open source and it is mantained by large community of developers and organizations.  
- Google started with Angular JS  
- The last version of Angular JS is 1.8  
- Google started Angular as alternative for Angular JS  
- Latest version of Angular is 14x.  
- Upto Angular 11 versions - They are no longer under support.  
- Angular is used for building SPA [Single Page Applications].  
  
Setup Environment for Angular  
------------------------------------------  
1. Download and Install NodeJS on your PC  
2. Download and Install TypeScript  
3. Download and Install Editor [Visual Studio Code]  
4. Download and Install Angular CLI  
    [Command Line Tool -  Used for creating and managing angular projects]  
  
        C:\> npm install  -g  @angular/cli  
  
Note: If your PC is already having angular/cli older version, then update to new.  
  
        - Check existing version  
                > ng version  
  
        - Un-install existing version  
                > npm uninstall -g @angular/cli  
                > npm cache verify  
  
        - Install New version  
  
                > npm install -g  @angular/cli  
                > ng version  
  
        - To Install any specific version  
  
                > npm install -g  @angular/cli@11  
  
Setup Workspace for Angular Projects  
-----------------------------------------------------  
- Workspace is a set of projects.  
- Workspace provides unified environment for all projects.  
- It allows to share resources across projects.  
- To create workspace go to any location on your PC and open in command prompt  
  
            D:\> ng  new   angular-projects  --create-application=false  
  
- Workspace is created by name "angular-projects", where you can maintain multiple project.  
  
- Open Workspace Folder in VS Code  
  
            D:\angular-projects  
  
Workspace File System:  
  
File / Folder                            Description  
----------------------------------------------------------------------------------------------------  
.vscode                        It comprises of VS code extentions that are used  
                                for angular.  
  
node\_modules                It comprises of all library files.  
  
  
.editorconfig                    It comprises rules for editor.  
  
.gitignore                        It defines the folders and files that are ignored while  
                                publishing on GIT.  
  
angular.json                    It is configuration file for projects that are running  
                                in the workspace.  
                                - From where scripts are accessed  
                                - From where styles are accessed  
                                - Static resources like images are accessed  
  
package.json                It defines project meta data.  
                                >npm init -y  
                                It contains information about dependencies in project.  
                                It defines name, version, license etc..  
  
package-lock.json            It contains details information about every  
                                dependency.  
  
                                >npm install  
  
README.md                It is help document.  
  
  
tsconfig.json                It comprises of rules for TypeScript.  
  
                                > tsc init  
  
Adding Project into Workspace  
------------------------------------------  
  
1. Open Terminal in workspace  
  
2. Run the command  
  
    > ng  generate application  shopping  
         
        Add routing to project ?  N  
        Which style sheet format ? CSS    [sass, less]  
  
3. Test your project by serving on server  
  
        > ng  serve  --project=shopping  
  
4. Your project start on    
        "[http://localhost:4200&quot](http://localhost:4200&quot/" \t "https://classroom.google.com/c/NTU2MTgyMTg5NzM0/m/NTY3NTY1NTcxNDEx/_blank);  
  
5. Open any browser and request the url  
  
        [http://localhost:4200](http://localhost:4200/" \t "https://classroom.google.com/c/NTU2MTgyMTg5NzM0/m/NTY3NTY1NTcxNDEx/_blank)

03/11/2022

Angular 14  
  
- Angular is a developers platform.  
- Developers platform provides end-to-end solution for developer.  
- It provides all tools and languages that are required for  
    a) Building  
    b) Debug  
    c) Testing  
    d) Deploying  
- Google introduced Angular as open source and it is mantained by large community of developers and organizations.  
- Google started with Angular JS  
- The last version of Angular JS is 1.8  
- Google started Angular as alternative for Angular JS  
- Latest version of Angular is 14x.  
- Upto Angular 11 versions - They are no longer under support.  
- Angular is used for building SPA [Single Page Applications].  
  
Setup Environment for Angular  
------------------------------------------  
1. Download and Install NodeJS on your PC  
2. Download and Install TypeScript  
3. Download and Install Editor [Visual Studio Code]  
4. Download and Install Angular CLI  
    [Command Line Tool -  Used for creating and managing angular projects]  
  
        C:\> npm install  -g  @angular/cli  
  
Note: If your PC is already having angular/cli older version, then update to new.  
  
        - Check existing version  
                > ng version  
  
        - Un-install existing version  
                > npm uninstall -g @angular/cli  
                > npm cache verify  
  
        - Install New version  
  
                > npm install -g  @angular/cli  
                > ng version  
  
        - To Install any specific version  
  
                > npm install -g  @angular/cli@11  
  
Setup Workspace for Angular Projects  
-----------------------------------------------------  
- Workspace is a set of projects.  
- Workspace provides unified environment for all projects.  
- It allows to share resources across projects.  
- To create workspace go to any location on your PC and open in command prompt  
  
            D:\> ng  new   angular-projects  --create-application=false  
  
- Workspace is created by name "angular-projects", where you can maintain multiple project.  
  
- Open Workspace Folder in VS Code  
  
            D:\angular-projects  
  
Workspace File System:  
  
File / Folder                            Description  
----------------------------------------------------------------------------------------------------  
.vscode                        It comprises of VS code extentions that are used  
                                for angular.  
  
node\_modules                It comprises of all library files.  
  
  
.editorconfig                    It comprises rules for editor.  
  
.gitignore                        It defines the folders and files that are ignored while  
                                publishing on GIT.  
  
angular.json                    It is configuration file for projects that are running  
                                in the workspace.  
                                - From where scripts are accessed  
                                - From where styles are accessed  
                                - Static resources like images are accessed  
  
package.json                It defines project meta data.  
                                >npm init -y  
                                It contains information about dependencies in project.  
                                It defines name, version, license etc..  
  
package-lock.json            It contains details information about every  
                                dependency.  
  
                                >npm install  
  
README.md                It is help document.  
  
  
tsconfig.json                It comprises of rules for TypeScript.  
  
                                > tsc init  
  
Adding Project into Workspace  
------------------------------------------  
  
1. Open Terminal in workspace  
  
2. Run the command  
  
    > ng  generate application  shopping  
         
        Add routing to project ?  N  
        Which style sheet format ? CSS    [sass, less]  
  
3. Test your project by serving on server  
  
        > ng  serve  --project=shopping  
  
4. Your project start on    
        "[http://localhost:4200&quot](http://localhost:4200&quot/" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTA0MTc4ODQ4ODk2/_blank);  
  
5. Open any browser and request the url  
  
        [http://localhost:4200](http://localhost:4200/" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTA0MTc4ODQ4ODk2/_blank)

Install Angular CLI  
    > npm install -g @angular/cli  
Create Workspace  
    > ng new workspaceName --create-application=false  
Add a new Project  
    > ng generate application  shopping  
Start Project  
    > ng serve --project=shopping  
  
                                    Project File System  
  
File/Folder                        Description  
------------------------------------------------------------------------------------------------------  
src                            It comprises all application resources.  
                            - components  
                            - services  
                            - guards etc..  
  
.browserslistrc            It is used to configure the browsers supported for  
                            applications.  
  
karma.conf.js            It is a testing configuration file.  
                            Karma is a testing framework used by angular.  
  
tsconfig.app.json        It is TypeScript configuration file for current project.  
  
tsconfig.spec.json        "spec" files are related to testing.  
  
  
                    Angular Resources folder "SRC"  
  
File/Folder                            Description  
---------------------------------------------------------------------------------------------  
app                    It comprises of application resources like  
                        - components  
                        - services  
                        - factories  
                        - pipes  
                        - gaurds etc..  
  
assets                It is a folder that contains static resources  
                        - images, text docs, pdf, audio, video etc..  
  
environments        It is used to configure all evironments for application  
                        a) Development  
                        b) Quality | Testing  
                        c) Production  
  
favicon.ico            It is shortcut icon or favicon for application.  
  
index.html            It is the startup page for every project.  
  
main.ts                It is entry point for starting your application.  
  
pollyfills.ts            It is used to handle differential loading.  
                        Differential loading is a technique of configuring library  
                        suitable for browser.  
                        \* Legacy Library for Legacy Browser  
                        \* Modular Library for Modern Browser  
  
styles.css            It comprises of global styles that are used for all  
                        component in your application.  
  
test.ts                    It comprises of test attributes used for testing your  
                        resources.  
  
  
                           Angular High level Architecture  
- Angular uses 2 types of compiling techniques  
    a) JIT  
    b) AOT  
- JIT is Just-in-Time ,  It compiles the code in browser.  
- AOT is Ahead-of-Time, It compiles the code at application level.  
- Babel is the compiler with "Ivy" as engine for Angular.  
- Angular Introduced Ivy from version 11x.  
- The process of converting static DOM into dynamic DOM is known as "bootstrapping".  
  
1. Client Requests Angular App  => [http://localhost:4200](http://localhost:4200/" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTY3ODUxMTk1MDI1/_blank)  => Index.html  
2. main.ts => app.module.ts => compiled => load component into index.html

[](https://drive.google.com/file/d/1F_dIl8hPTHu8KmF2lN_kntn62NxRScwJ/view?usp=drive_web&authuser=0" \o "arch1.png" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTY3ODUxMTk1MDI1/_blank)

[arch1.png](https://drive.google.com/file/d/1F_dIl8hPTHu8KmF2lN_kntn62NxRScwJ/view?usp=drive_web&authuser=0" \o "arch1.png" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTY3ODUxMTk1MDI1/_blank)

04/11/2022

Angular Components  
- A component is considered as Building Block for SPA.  
 [Single Page Application]  
- Component comprises of  
    a) Presentation  
    b) Styles  
    c) Logic  
- Presentation is defined by using HTML.  
- Styles are defined by using CSS.  
- Logic is defined by using TypeScript.  
- Web components enable reusability, maintainability, extensibility and testability.  
- Popular web components that you come accross in web development  
        - Carousel  
        - Cards  
        - Modals  
        - Navbar  
        - Accordion  
        - Dropdown etc..  
  
- Technically angular component comprises of 3 phases  
    a) Markup  
    b) Styles  
    c) Logic  
- Angular component provides 2 types design techniques  
    a) Inline documentation  
    b) Code behind documentation  
  
Inline Documentation  
- In this techniques markup, styles and logic all are kept in one file.  
- The component file will have extention ".ts".  
  
            getbootstrap.com => themes  
  
- Inline technique encapsulates everything into one file, hence it reduces the number of requests and improves page load time.  
  
- If everything is in one file then it is hard to extend and test.  
  
- If your component doesn't require regular extentions and have a stable functionality then better use "Inline Documentation".  
  
Creating Angular Component  
----------------------------------------  
- Every component  must have the suffix name as "Component".  
  
                    Home.Component.ts  
                    Login.Component.ts  
  
- Every component must be inside "src\app" folder  
  
- Add a new folder into "app" by name  "components"  
  
- Add a new file into components  
                     
                    "header.component.ts"  
  
- Technically every component is a TypeScript class.  
  
   export class  HeaderComponent  
   {  
  
   }  
  
- Angular uses "@Component()" decorator [directive] to configure a class as component.  
  
- @Component() is a member of "@angular/core" library  
  
Syntax:  
    import { Component }  from  "@angular/core";  
  
    @Component()  
     export class HeaderComponent  
     {    
  
     }  
  
- @Component() is a contract [interface] that specifies the meta data for component.  
- Every component meta data comprises of various attributes  
            a) selector  
            b) template  
            c) styles  etc..  
- Every component must have 2 attributes defined  
            a) selector  
            b) template  
- Selector specifies how your component can be accessed and used in any page.  
- Template specifies the markup to render when component is injected.  
  
Syntax:  
    import { Component } from "@angular/core";  
  
    @Component({  
        selector : "app-header",  
        template: "<h2>{{ title }}</h2>"  
    })  
    export class HeaderComponent  
    {  
        public title:string = "Shopper.";  
    }  
  
- Go to "app.module.ts"  and import your component  
  
import { HeaderComponent }  from "./components/header.component";  
  
- You have to set your component as startup component  
  
        bootstrap : [ HeaderComponent ]  
  
- Go to Index.html  and inject the component in body  
  
    <body>  
        <app-header> </app-header>  
    </body>  
  
Note:  "app.module.ts" is is managed by  "@NgModule()" decorator  
  
         declarations: [ ]       It contains information about components  
                                  to use in your app.  
  
                                  Make sure that every component you  
                                  designed is declared in declarations[].  
  
         imports : []              It contains information about modules.  
  
         providers:[]              It contains information about Services.  
  
         bootstrap:[]              It defines the components to start with.  
  
Note:  The components defined in bootstrap must match with  
         components defined in index.html  
  
Start your project:  
  
  > ng serve --project=shopping  
  
Browser :  
   
   [http://localhost:4200](http://localhost:4200/" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTY3ODUwMDA5MDQ1/_blank)

05/11/2022

Adding component by using Code Behind Technique:  
- In code behind technique every component comprises of 4 files  
     
        .html                    : Presentation  
        .css                    : Styles  
        .ts                        : Logic  
        .spec.ts                : Testing  
  
- It allows reusability, maintainability, testability and extensibility.  
- Issue with code behind is, having multiple files will increase the number of requests and also page load time.  
- If you are using code behind then "@Component()" directive uses following attributes  
  
            @Component({  
                selector : "app-login",  
                templateUrl: "login.component.html",  
                stylesUrl : ["login.component.css", "..."]  
            })  
  
Ex:  
1. Add a new folder into components  
            "login"  
  
2. Add following files into login folder  
  
        login.component.ts  
        login.component.html  
        login.component.css  
  
3. login.component.ts  
  
import { Component } from "@angular/core";  
  
@Component({  
    selector: "app-login",  
    templateUrl: "./login.component.html",  
    styleUrls: ["./login.component.css"]  
})  
export class LoginComponent  
{  
    public title:string = "User Login";  
}  
  
4. login.component.html  
  
<div id="container">  
  <form>  
    <h3>{{title}}</h3>  
    <dl>  
        <dt>User Name</dt>  
        <dd><input type="text"></dd>  
        <dt>Password</dt>  
        <dd><input type="password"></dd>  
    </dl>  
    <button>Login</button>  
  </form>  
</div>  
  
5. login.component.css  
  
#container {  
    display: flex;  
    justify-content: center;  
    align-items: center;  
    height: 400px;  
}  
form {  
    border:2px solid gray;  
    border-radius: 20px;  
    padding: 20px;  
}  
  
6. Go to app.module.ts  
  
 declarations : [  
                .............  
                LoginComponent  
                ]  
  
  bootstrap : [  
                 LoginComponent  
                ]  
  
7. Go to Index.html  
  
        <body>  
            <app-login> </app-login>  
        </body>  
  
Adding components by using Scaffolding technique:  
  
- Angular CLI provides commands that can scaffold and generate component required for your requirements.  
  
Commands in Terminal  
  
 > ng  generate component  name  
 > ng  g c  name  
  
  --dry-run                 It shows the preview of command without  
                             really executing the command.  
  
  --inline-style             It will not generate css file.  
  
  --inline-template         It will not generate html file.  
  
  --skip-tests                 It will not generate test file.  
  
   
Setup Bootstrap for Angular Project  
-------------------------------------------------  
  
1. Open Workspace location in Terminal  
  
        D:\angular-projects> npm install bootstrap --save  
                                   npm install bootstrap-icons --save  
  
2.  Goto "src\styles.css"  file and import files  
  
  
@import "../../../node\_modules/bootstrap/dist/css/bootstrap.css";  
  
@import "../../../node\_modules/bootstrap-icons/font/bootstrap-icons.css";  
  
3.  login.component.html  
  
  
<div class="container-fluid d-flex justify-content-center align-items-center" style="height:400px">  
  <form class="border border-2 rounded rounded-2 p-3 border-primary">  
    <h3> <span class="bi bi-person-fill"></span> {{title}}</h3>  
    <dl>  
        <dt>User Name</dt>  
        <dd><input type="text" class="form-control"></dd>  
        <dt>Password</dt>  
        <dd><input type="password" class="form-control"></dd>  
    </dl>  
    <button class="btn btn-primary w-100">Login</button>  
  </form>  
</div>  
  
Ex:  
amazon.component.html  
  
<div class="container-fluid">  
    <h2>Amazon Search</h2>  
    <div class="input-group">  
        <select class="input-group-text">  
            <option>All</option>  
        </select>  
        <input type="text" class="form-control">  
        <button class="btn btn-warning">  
            <span class="bi bi-search"></span>  
        </button>  
    </div>  
    <h2>Netflix Register</h2>  
    <div class="bg-dark p-4">  
        <div class="input-group input-group-lg">  
            <input type="email" placeholder="Your email address" class="form-control">  
            <button class="btn btn-danger">  
                Get Started <span class="bi bi-chevron-right"></span>  
            </button>  
        </div>  
    </div>  
</div>

07/11/2022

Multiple Components  
- Angular allows index.html to start with more than one component.  
  
    <body>  
        <app-header> </app-header>  
        <app-footer> </app-footer>  
    </body>  
  
- All components configured in index.html must be defined in "bootstrap" of app.module.ts  
  
     bootstrap :  
        [  
          HeaderComponent, FooterComponent  
        ]  
  
- You can access a component in another component.  
- You just need the selector name.  
  
                            Data Binding in Angular

08/11/2022

Angular Data Binding  
  
- Data Binding is a technique used in web applications to access data from any source and bind to UI.  
- JavaScript and jQuery depend on lot of DOM manipulations.  
                getElementById()  
                createElement()  
                appendChild() etc...  
 and Events  
                onload,  
                onblur  
                onchange  
                onkeyup etc..  
  
- Angular is completely de-coupled for DOM manipulations.  
- Angular implility uses a framework for data binding called  
        "MVC"  
  [Model - View - Controller]  
  
What is MVC?  
- It is a software architectural pattern.  
- Software architectural pattern can build application and also can control its flow.  
- Introduced in early 1970's by "Trygve".  
- Formulated with a language called "Small Talk".  
- Code separation concerns.  
  
        Java                Spring  
        PHP                Cake PHP, Code Igniter      
        Python            Django, Flask, Grock  
        Ruby                Ruby on Rails  
        .NET                ASP.NET MVC  
        JavaScript        SPINE, Angular  
  
- MVC separates application into 3 components  
  
        a) Model  
        b) View  
        c) Controller  
  
Model  
- It represents the data we are working with  
- It contains data and its validations.  
- Angular data will be in JSON format.  
 [JavaScript Object Notation]  
  
View  
- It represents the UI  
- It configures HTML UI.  
  
Controller  
- It is the core MVC component.  
- It handles overall application flow.  
- It comprises of application specific logic.  
  
                            Binding Technique  
- Angular uses data binding in 3 ways  
  
        a) Interpolation  
        b) Attribute Binding  
        c) Property Binding  
  
- Interpolation is a technique where data is not binded to any element it is just kept as a literal.  
  Interpolation is defined by using "{{ }}" [Data Binding Expression]  
  
- Property binding is a techique that allows to bind any value to the property of element.  
  Property is defined by using "[ ]"  
  
    <element  [property]=value>  
    <element> {{ value }} </element>  
  
- Interpolation is only one way binding [uni-directional]  
- It is forward only.  
- It is one time rendering.  
  
Syntax:  
            public  isSelected:boolean = true;  
  
            <input type="checkbox" checked={{isSelected}}>  
  
- Property binding is also one way  
- It is also forward only  
- It is not one time, it is continous update.  
  
Syntax:  
            <input type="checkbox"  [checked]="isSelected">  
  
Attribute Binding  
- Every HTML element is not having a suitable property.  
  
<img src class>  
document.getElementById("imgId").className  
  
- All attributes of HTML element are not available as properties.  
- Then you have to binding value directly to attribute by using "attr"  
  
Syntax:  
         public  theight:number = 200;  
         public  twidth:number = 400;  
  
        <table [height]="theight">        // invalid  
        <table [attr.height]="theight">  
        <table [width] ="twidth">  
  
Summary:  
- Angular Data binding is classified into 2 types  
        a) One Way Binding  
        b) Two Way Binding  
- One Way Binding can be defined using  
        a) Interpolation {{ }}  
        b) Property  [propertyName]  
        c) Attribute  [attr.propertyName]

09/11/2022

Angular Data Binding  
- One Way Binding  
  a) Interpolation  
  b) Property Binding  
  c) Attribute Binding  
  
                            Two Way Binding  
- It is the process of accessing data from source and updating to UI.  
- Identifying the changes in UI and updating back into data source.  
- Angular two way binding is managed by "Model".  
- Model is reffered as "Single-Source-of-Truth".  
- Model keeps the traces of data  
  
        previousValue == currentValue        => No Change Detected  
        previousValue !== currentValue    => Change Detected  
  
- When ever change is detected it updates the changes into data.  
- Change Detection and Updating Changes in Angular is managed by "Angular Model".  
- Angular Model is a component which is configured as "ngModel".  
- It is a member of "FormsModule".  
- FormsModule is a member of "@angular/forms" library.  
- ngModel uses "Property" and "Event" binding techniques  
  
        [ ]        => Property Binding        [value]=""  
        ( )        => Event Binding            (change)="function()"  
  
Syntax:  
        <input type="text"  [(ngModel)]="UserName">  
            [value]  
        <input type="checkbox"  [(ngModel)]="IsValid">  
            [checked]  
  
Ex:  
1. Go to "app.module.ts"  
  
  import  { FormsModule }  from  "@angular/forms";  
  
  imports : [  
               FormsModule  
            ]  
  
2. data-binding.component.ts  
  
      public  UserName:string = "John";  
  
3. data-binding.component.html  
  
     UserName : <input type="text" [(ngModel)]="UserName">  
  
    <p> Hello ! {{ UserName }} </p>  
  
  
Note :  [ ] = value property  
          ( ) = change event  
  
  
Ex:  
1. Add a new folder into app folder  
    "contracts"  
  
2. Add a new file  
        ProductContract.ts  
  
export interface ProductContract  
{  
    Name:string;  
    Price:number;  
    Stock:boolean;  
    City:string;  
}  
  
3. Data-Binding.Component.ts  
  
import { Component, OnInit } from '@angular/core';  
import { ProductContract } from '../../contracts/ProductContract';  
  
@Component({  
  selector: 'app-data-binding',  
  templateUrl: './data-binding.component.html',  
  styleUrls: ['./data-binding.component.css']  
})  
export class DataBindingComponent implements OnInit {  
  
  public Product:ProductContract = {  
     Name: "",  
     Price: 0,  
     Stock: false,  
     City: "Select City"  
  }  
  
  public UpdatedProduct:ProductContract = {  
     Name: "",  
     Price: 0,  
     Stock: false,  
     City : ""  
  }  
  
  public UpdateClicked():void{  
      this.UpdatedProduct = {  
          Name : this.Product.Name,  
          Price: this.Product.Price,  
          Stock: this.Product.Stock,  
          City : this.Product.City  
      }  
  }  
  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  
}  
  
  
4. Data-Binding.Component.html  
  
  
<div class="container-fluid">  
   <div class="row">  
     <div class="col-3">  
        <h3>Register Product</h3>  
        <dl>  
            <dt>Name</dt>  
            <dd><input type="text" [(ngModel)]="Product.Name" class="form-control"></dd>  
            <dt>Price</dt>  
            <dd><input type="text" [(ngModel)]="Product.Price" class="form-control"></dd>  
            <dt>Stock</dt>  
            <dd class="form-switch">  
                <input class="form-check-input" [(ngModel)]="Product.Stock" type="checkbox"> Available  
            </dd>  
            <dt>City</dt>  
            <dd>  
                <select [(ngModel)]="Product.City" class="form-select">  
                    <option>Select City</option>  
                    <option>Delhi</option>  
                    <option>Hyd</option>  
                </select>  
            </dd>  
        </dl>  
        <button (click)="UpdateClicked()" class="btn btn-primary w-100">Update Product</button>  
     </div>  
     <div class="col-9">  
        <div class="position-absolute end-0 w-50">  
            <h3>Product Details</h3>  
            <dl class="row">  
                <dt class="col-3">Name</dt>  
                <dd class="col-9" [innerText]="UpdatedProduct.Name"></dd>  
                <dt class="col-3">Price</dt>  
                <dd class="col-9">{{UpdatedProduct.Price}}</dd>  
                <dt class="col-3">Stock</dt>  
                <dd class="col-9">{{(UpdatedProduct.Stock==true)?"Available":"Out of Stock"}}</dd>  
                <dt class="col-3">City</dt>  
                <dd class="col-9">{{UpdatedProduct.City}}</dd>  
            </dl>  
        </div>  
     </div>  
   </div>  
</div>

10/11/2022

Data Binding using MVVM  
  
- It is an architectural pattern that handle communication between model and view by using "View-Model" instead of controller.  
  
            [ Model-View-ViewModel]  
  
- Model comprises of data  
- View describes the UI.  
- ViewModel is memory allocated for data inside view, without any controller.  
- It is good for component which are using inline-design technique.  
- You can use MVVM in code behind component, so that controller burden is reduced.  
  
Configuring Elements to Use in MVVM approach:  
  
1. Every HTML element must have "ngModel" configured as attribute.  
  
        <input type="text" ngModel>  
        <select ngModel>  
  
2. Every HTML element must have a reference name in order to submit its value.  
  
        <input type="text"  ngModel  name="UserName">  
        <select ngModel name="City">  
  
  
3. ngModel will not use HTML "name" to access element, It requires a reference name in memory defined with "#Name".  
  
<input type="text" ngModel name="UserName"   #UserName="ngModel">  
  
    # refers to ID - It must be unique.  
  
  
4. Reference Name of Model and Element Name must match.  
  
        #UName="ngModel"    name="UserName"   // invalid  
        #UserName="ngModel"  name="UserName" // valid  
  
5. You can access any model reference with various attributes.  
    Every model reference returns various values.  
  
            UserName.value        [returns the value contained]  
            UserName.valid  
            UserName.invalid  
            UserName.pristine  
            UserName.dirty  
            UserName.touched  
            UserName.untouched  
            UserName.errors  
  
Ex:  
1. Add a new component  
  
        > ng g c  mvvm-demo  --skip-tests  
  
2. mvvm-demo.component.html  
  
<div class="container-fluid">  
    <div class="row">  
        <div class="col-3">  
            <h3>Register Product</h3>  
            <dl>  
                <dt>Name</dt>  
                <dd><input type="text" ngModel name="Name" #Name="ngModel"></dd>  
                <dt>Price</dt>  
                <dd><input type="text" ngModel name="Price" #Price="ngModel"></dd>    
                <dt>City</dt>  
                <dd>  
                    <select ngModel name="City" #City="ngModel">  
                        <option>Delhi</option>  
                        <option>Hyd</option>  
                    </select>  
                </dd>  
                <dt>Stock</dt>  
                <dd>  
                    <input type="checkbox" #Stock="ngModel" name="Stock" ngModel> Available  
                </dd>  
            </dl>  
        </div>  
        <div class="col-9">  
            <h3>Product Details</h3>  
            <dl>  
                <dt>Name</dt>  
                <dd>{{Name.value}}</dd>  
                <dt>Price</dt>  
                <dd>{{Price.value}}</dd>  
                <dt>City</dt>  
                <dd>{{City.value}}</dd>  
                <dt>Stock</dt>  
                <dd>{{(Stock.value==true)?"Available":"Out of Stock"}}</dd>  
            </dl>  
        </div>  
    </div>  
</div>    
  
                             Angular Directives  
- Technically directive is a function that can handle various interactions.  
- Angular directive can  
            a) Return Markup  
            b) Extend Markup  
            c) Make markup more interactive and responsive  
- Angular directive can be used as  
            a) Element  
            b) Attribute  
            c) Class  
  
Ex: Element  
            <app-login> </app-login>  
  
Ex: Attribute  
            <input type="text" ngModel>  
            <div \*ngIF="">  
Ex: Class  
        css file  
        .ng-valid {  }  
        .ng-pristine { }  
        .ng-dirty { }  
  
- Angular Provides several built-in directives and also allows to create custom directives.  
  
- Angular Built-in Directives are classified into 2 major groups  
  
        a) Structural Directives  
        b) Attribute Directives  
  
  
                             Structural Directives  
                             ----------------------------  
- Structural directives are used for DOM Structure.  
    a) Adding Element into DOM  
     b) Removing Elements from DOM  
    c) Iterating Elements in DOM etc..  
  
- Angular Structural Directives are  
    a) ngIf  
    b) ngSwitch  
    c) ngFor  
  
  
                                    NgIF  
- It is a structural directive used to Add or Remove any element from DOM hierarchy.  
- It uses a boolean value or expression.  
- Every structural directive is added to HTML element by using "\*"  
  
        <div \*ngIf="true/false">  
  
Ex:  
if-demo.component.html  
  
<div class="container-fluid">  
    <h2>Product Details</h2>  
    <dl>  
        <dt>Name</dt>  
        <dd>Nike Casuals</dd>  
        <dt>Price</dt>  
        <dd>4500.56</dd>  
        <dt class="form-switch"> <input class="form-check-input" ngModel name="Preview" #Preview="ngModel" type="checkbox"> Show Preview</dt>  
        <dd \*ngIf="Preview.value">  
            <img src="assets/shoe.jpg"  width="100" height="100">  
        </dd>  
    </dl>  
</div>

11/11/2022

Data Binding  
 - MVC  
 - MVVM  
Angular Directives  
- Structural Directives  
    a) NgIF  
  
NgIF  
  <div  \*ngIf="booleanValue/booleanExpression">  
  
                    NgIF with Then and Else Block  
  
- NgIf defines condition.  
- Then specifies the block to execute when condition is true.  
- Else specifies the block to execute when condition is false.  
- Angular "NgIF"  
        a) Adds a container or element using "then" block  
        b) Removes a container or element using "else" block  
- NgIF can switch between then and else block  
  
Syntax:  
        <div  \*ngIf="condition; then  thenBlockId else elseBlockId">  
         
- NgIf can't handle HTML container dynamically.  
        <div> <span> <p> ...  
  
- You have to use Dynamic container provided by angular  
  
        <ng-template> </ng-template>  
  
- "then" and "else" can handle only dynamic container. Hence NgIF requires "<ng-template>" to keep all the content.  
  
Note: Make sure that the dynamic templates are configured outside condition container.  
  
Syntax:  
 <div \*ngIf="true; then  trueBlock else  falseBlock"> </div>  
 <ng-template  #trueBlock>  
        to display when true  
 </ng-template>  
 <ng-template #elseBlock>  
         to display when false  
 </ng-template>  
  
Ex:  [http://fakestoreapi.com](http://fakestoreapi.com/" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTY5ODIxMDAxNjUw/_blank)  
  
Accessing Data from API:  
- JavaScript provides a promise called "fetch()".  
- "fetch()" uses an Ajax request for fetching data from any API.  
  
Syntax:  
    fetch("url")  
    .then(function(response) {  
       return response.json();  
    })  
    .then(function(data) {  
       // handle data  
    })  
   .catch(function(err) {  
       // throw error  
   }  
  
Ex:  
1. Create a  new Contract  
        FakestoreProduct.ts  
  
export interface FakeStoreProduct  
{  
    id:number;  
    title:string;  
    description:string;  
    price:number;  
    image:string;  
    rating:{rate:number, count:number}  
}  
  
2. Add component  
  
    > ng g c if-demo --skip-tests  
  
3. if-demo.component.ts  
  
import { Component, OnInit } from '@angular/core';  
import { FakeStoreProduct } from '../../contracts/FakeStoreProduct';  
  
@Component({  
  selector: 'app-if-demo',  
  templateUrl: './if-demo.component.html',  
  styleUrls: ['./if-demo.component.css']  
})  
export class IfDemoComponent implements OnInit {  
  
  public Product:FakeStoreProduct = {  
    id: 0,  
    title: '',  
    description: '',  
    image: '',  
    price: 0,  
    rating: {rate: 0, count: 0}  
  };  
  
  public Count:number = 1;  
   
  public LoadProduct(id:number):void {  
      fetch(`[http://fakestoreapi.com/products/$](http://fakestoreapi.com/products/$" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTY5ODIxMDAxNjUw/_blank){id}`)  
      .then(response=> response.json())  
      .then(data=> {  
         this.Product = data;  
      })  
  }  
  
  constructor() { }  
  
  ngOnInit(): void {  
     this.LoadProduct(1);  
  }  
  
  public NextClick():void{  
    this.Count++;  
    this.LoadProduct(this.Count);  
  }  
  public PreviousClick():void {  
    this.Count--;  
    this.LoadProduct(this.Count);  
  }  
  
}  
  
4. if-demo.component.html  
  
<div class="container-fluid">  
    <h2>Product Details</h2>  
    <dl>  
        <label>Preview</label>  
        <div class="form-switch">  
            <input type="checkbox" ngModel name="Toggle" #Toggle="ngModel"  class="form-check-input">  
        </div>  
        <label>Details</label>  
        <div \*ngIf="Toggle.value; then detailsBlock else previewBlock"></div>  
        <ng-template #detailsBlock>  
            <dt>Title</dt>  
            <dd>{{Product.title}}</dd>  
            <dt>Price</dt>  
            <dd>{{Product.price}}</dd>  
            <dt>Rating</dt>  
            <dd>  
                <span class="bi bi-star-fill text-success"></span>  
                {{Product.rating.rate}} [{{Product.rating.count}}]  
            </dd>  
        </ng-template>  
        <ng-template #previewBlock>  
            <dt>Preview</dt>  
            <dd>  
                <img [src]="Product.image" width="100" height="100">  
            </dd>  
        </ng-template>  
         
    </dl>  
    <button (click)="PreviousClick()" class="btn btn-primary me-2" aria-label="previous">  
        <span class="bi bi-chevron-left"></span>  
    </button>  
    <button (click)="NextClick()" class="btn btn-primary">  
        <span class="bi bi-chevron-right" aria-label="next"></span>  
    </button>  
</div>

17/11/2022

Angular Directives:  
NgIF  
    then  
    else  
  
FAQ: Can we define multiple then blocks?  
Ans : No. Angular allows only one then block in "NgIF" directive.  
  
FAQ: How to handle multiple conditions?  
Ans:  By using a technique called "Content Projection".  
  
FAQ: What is Content Projection?  
Ans:  It is the process of creating a template explicitly and rendering  
        into UI.  
  
        Templates are created by using "TemplateRef<>"  
        Templates are rendered by using "ViewChild()"  
  
  
Ex:  
content-projection.component.ts  
  
import { Component, OnInit, TemplateRef, ViewChild } from '@angular/core';  
  
@Component({  
  selector: 'app-content-projection',  
  templateUrl: './content-projection.component.html',  
  styleUrls: ['./content-projection.component.css']  
})  
export class ContentProjectionComponent implements OnInit {  
  
  public thenBlock:TemplateRef<any>|null = null;  
  public blockName:string = '';  
  
  @ViewChild('block1',{static:true}) public block1:TemplateRef<any>|null = null;  
  @ViewChild('block2',{static:true}) public block2:TemplateRef<any>|null = null;  
  @ViewChild('block3',{static:true}) public block3:TemplateRef<any>|null = null;  
   
  
  constructor() { }  
  
  ngOnInit(): void {  
     this.thenBlock = this.block1;  
  }  
  public BlockChanged():void {  
     switch(this.blockName){  
        case "block1":  
         this.thenBlock = this.block1;  
         break;  
        case "block2":  
        this.thenBlock = this.block2;  
        break;  
        case "block3":  
        this.thenBlock = this.block3;  
        break;  
     }  
  }  
  
}  
  
content-projection.component.html  
  
<div class="container-fluid">  
   <h2>Content Project Demo</h2>  
   <select [(ngModel)]="blockName" (change)="BlockChanged()">  
    <option value="block1">Block-1</option>  
    <option value="block2">Block-2</option>  
    <option value="block3">Block-3</option>  
   </select>  
   <div \*ngIf="true; then thenBlock  else elseBlock"></div>  
   <ng-template #block1>  
        <h2>Then Block-1</h2>  
   </ng-template>  
   <ng-template #block2>  
        <h2>Then Block-2</h2>  
   </ng-template>  
   <ng-template #block3>  
        <h2>Then Block-3</h2>  
    </ng-template>  
   <ng-template #elseBlock>  
        <h2>Else Block</h2>  
   </ng-template>  
</div>  
  
  
Ex:  
 product-view.component.ts  
  
import { Component, OnInit, TemplateRef, ViewChild } from '@angular/core';  
  
@Component({  
  selector: 'app-product-view',  
  templateUrl: './product-view.component.html',  
  styleUrls: ['./product-view.component.css']  
})  
export class ProductViewComponent implements OnInit {  
  
  public thenBlock:TemplateRef<any>|null = null;  
  
  @ViewChild("View1", {static:true}) public View1:TemplateRef<any>|null = null;  
  @ViewChild("View2", {static:true}) public View2:TemplateRef<any>|null = null;  
  @ViewChild("View3", {static:true}) public View3:TemplateRef<any>|null = null;  
  @ViewChild("View4", {static:true}) public View4:TemplateRef<any>|null = null;  
  @ViewChild("View5", {static:true}) public View5:TemplateRef<any>|null = null;  
  
  
  constructor() { }  
  
  ngOnInit(): void {  
    this.thenBlock = this.View1;  
  }  
  public MouseOverProduct(viewName:string){  
      switch(viewName){  
         case "View1":  
         this.thenBlock = this.View1;  
         break;  
         case "View2":  
         this.thenBlock = this.View2;  
         break;  
         case "View3":  
         this.thenBlock = this.View3;  
         break;  
         case "View4":  
         this.thenBlock = this.View4;  
         break;  
         case "View5":  
         this.thenBlock = this.View5;  
         break;  
      }  
  }  
  
}  
  
  
product-view.component.html  
  
<div class="container-fluid">  
<h3>Printer</h3>  
<div class="row">  
    <nav class="col-2">  
        <div class="border border-2 border-primary mb-2" style="width: 55px;">  
            <img (mouseover)="MouseOverProduct('View1')" src="assets/View1.png" width="50" height="50">  
        </div>  
        <div class="border border-2 border-primary mb-2" style="width: 55px;">  
            <img (mouseover)="MouseOverProduct('View2')" src="assets/View2.png" width="50" height="50">  
        </div>  
        <div class="border border-2 border-primary mb-2" style="width: 55px;">  
            <img (mouseover)="MouseOverProduct('View3')" src="assets/View3.png" width="50" height="50">  
        </div>  
        <div class="border border-2 border-primary mb-2" style="width: 55px;">  
            <img (mouseover)="MouseOverProduct('View4')" src="assets/View4.png" width="50" height="50">  
        </div>  
        <div class="border border-2 border-primary mb-2" style="width: 55px;">  
            <img (mouseover)="MouseOverProduct('View5')" src="assets/View5.png" width="50" height="50">  
        </div>  
    </nav>  
    <main class="col-10">  
        <div \*ngIf="true; then thenBlock"></div>  
        <ng-template #View1>  
            <img src="assets/View1.png" width="400" height="400">  
        </ng-template>  
        <ng-template #View2>  
            <img src="assets/View2.png" width="400" height="400">  
        </ng-template>  
        <ng-template #View3>  
            <img src="assets/View3.png" width="400" height="400">  
        </ng-template>  
        <ng-template #View4>  
            <img src="assets/View4.png" width="400" height="400">  
        </ng-template>  
        <ng-template #View5>  
            <img src="assets/View5.png" width="400" height="400">  
        </ng-template>  
    </main>  
</div>  
</div>  
  
Task: Thendemo  
  
thendemo.component.ts  
  
import { Component, OnInit, TemplateRef, ViewChild } from '@angular/core';  
import { FakeStoreProduct } from '../../contracts/FakeStoreProduct';  
import { ProductContract } from '../../contracts/ProductContract';  
  
@Component({  
  selector: 'app-thendemo',  
  templateUrl: './thendemo.component.html',  
  styleUrls: ['./thendemo.component.css']  
})  
export class ThendemoComponent implements OnInit {  
  
  public Product:FakeStoreProduct = {  
     id: 0,  
     title: '',  
     price: 0,  
     description: '',  
     image: '',  
     rating: {rate:0, count: 0}  
  };  
  
  public LoadProduct():void {  
      fetch("[http://fakestoreapi.com/products/1&quot](http://fakestoreapi.com/products/1&quot" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTcxNTU1MzY4ODEx/_blank);)  
      .then(response=> response.json())  
      .then(data=>{  
        this.Product = data;  
      })  
  }  
  
  public thenBlock:TemplateRef<any>|null = null;  
  
  @ViewChild('preview',{static:true}) public preview:TemplateRef<any>|null = null;  
  @ViewChild('details',{static:true}) public details:TemplateRef<any>|null = null;  
   
  
  constructor() { }  
  
  ngOnInit(): void {  
    this.LoadProduct();  
    this.thenBlock = this.preview;  
  }  
  
}  
  
thendemo.component.html  
  
<div class="container-fluid">  
    <h2>Product Details</h2>  
    <button>Preview</button>  
    <button>Details</button>  
    <div \*ngIf="true; then thenBlock"></div>  
    <ng-template #preview>  
        <img [src]="Product.image" width="200" height="200">  
    </ng-template>  
    <ng-template #details>  
        <dl>  
            <dt>Title</dt>  
            <dd>{{Product.title}}</dd>  
            <dt>Price</dt>  
            <dd>{{Product.price}}</dd>  
        </dl>  
    </ng-template>  
</div>

18/11/2022

NgSwitch  
- It is a structural directive, which is used to add or remove elements from DOM.  
- It improves performance when page have to render multiple blocks.  
  
FAQ: What is issue with NgIF?  
Ans:  Rendering multiple blocks requires Content Projection.  
         It is heavy on application and slow in access.  
  
- The directives are  
        ngSwitch  
        ngSwitchCase  
        ngSwitchDefault  
  
Syntax:  
        <div [ngSwitch]="Value">  
                <div \*ngSwitchCase="A">  
                        Block-A  
                </div>  
                <div \*ngSwitchCase="B">  
                        Block-B  
                </div>  
                <div  \*ngSwitchDefault>  
                        Default Block  
                </div>  
        </div>  
  
Note: NgSwitchCase can directly operator on HTML Elements.  
        NgIf requires dynamic template <ng-template>  
  
Ex:  
 switch-demo.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-switch-demo',  
  templateUrl: './switch-demo.component.html',  
  styleUrls: ['./switch-demo.component.css']  
})  
export class SwitchDemoComponent implements OnInit {  
  
  public ViewName:string = 'View1';  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  public ChangeView(viewName:string){  
     this.ViewName = viewName;  
  }  
  
}  
  
switch-demo.component.html  
  
<div class="container-fluid">  
  <h2>Product Details</h2>  
  <div class="row">  
    <nav class="col-2">  
        <div class="mb-2">  
            <img (mouseover)="ChangeView('View1')" src="assets/View1.png" class="border border-2 border-primary" width="50" height="50">  
        </div>  
        <div class="mb-2">  
            <img (mouseover)="ChangeView('View2')" src="assets/View2.png" class="border border-2 border-primary" width="50" height="50">  
        </div>  
        <div class="mb-2">  
            <img (mouseover)="ChangeView('View3')" src="assets/View3.png" class="border border-2 border-primary" width="50" height="50">  
        </div>  
        <div class="mb-2">  
            <img (mouseover)="ChangeView('View4')" src="assets/View4.png" class="border border-2 border-primary" width="50" height="50">  
        </div>  
        <div class="mb-2">  
            <img (mouseover)="ChangeView('View5')" src="assets/View5.png" class="border border-2 border-primary" width="50" height="50">  
        </div>  
    </nav>  
    <main class="col-10">  
        <div [ngSwitch]="ViewName">  
            <div \*ngSwitchCase="'View1'">  
                <img src="assets/View1.png" width="400" height="400">  
            </div>  
            <div \*ngSwitchCase="'View2'">  
                <img src="assets/View2.png" width="400" height="400">  
            </div>  
            <div \*ngSwitchCase="'View3'">  
                <img src="assets/View3.png" width="400" height="400">  
            </div>  
            <div \*ngSwitchCase="'View4'">  
                <img src="assets/View4.png" width="400" height="400">  
            </div>  
            <div \*ngSwitchCase="'View5'">  
                <img src="assets/View5.png" width="400" height="400">  
            </div>  
        </div>  
    </main>  
  </div>  
</div>  
  
Switch-Demo.component.css  
  
img:hover {  
    cursor: grab;  
}  
  
Summary  
- NgIF  
- NgSwitch  
  
                                    NgFor  
- It is a repeater.  
- It is used to repeat any HTML element.  
- NgFor uses an "iterator" for repeating.  
  
            for..in  
            for..of  
            forEach  
            map()  
  
- NgFor uses "for..of" iterator.  
  
Syntax:  
        <li \*ngFor="let item of collection">  
  
Ex:  
for-demo.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-for-demo',  
  templateUrl: './for-demo.component.html',  
  styleUrls: ['./for-demo.component.css']  
})  
export class ForDemoComponent implements OnInit {  
  
  public categories:string[] = ["All","Electronics", "Footwear", "Fashion"];  
  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  
}  
  
for-demo.component.html  
  
<div class="container-fluid">  
    <div class="row">  
        <div class="col">  
            <h2>Categories</h2>  
            <ol>  
                <li \*ngFor="let item of categories">{{item}}</li>  
            </ol>  
        </div>  
        <div class="col">  
            <h2>Select Category</h2>  
            <select>  
                <option [value]="item" \*ngFor="let item of categories">{{item}}</option>  
            </select>  
        </div>  
        <div class="col">  
            <table class="table table-hover table-dark">  
                <thead>  
                    <tr>  
                        <th>Categories</th>  
                    </tr>  
                </thead>  
                <tbody>  
                    <tr \*ngFor="let item of categories">  
                        <td>{{item}}</td>  
                    </tr>  
                </tbody>  
            </table>  
        </div>  
        <div class="col">  
            <h2>NavBar</h2>  
            <ul>  
                <li \*ngFor="let item of categories">  
                    <a href="#">{{item}}</a>  
                </li>  
            </ul>  
        </div>  
        <div class="col">  
            <button class="btn btn-danger mb-2 w-100" \*ngFor="let item of categories">  
                {{item}}  
            </button>  
        </div>  
    </div>  
    <div class="row mt-3">  
        <div class="col">  
            <h2>Select Category</h2>  
            <ul id="choose" class="list-unstyled">  
                <li \*ngFor="let item of categories">  
                    <input type="checkbox"> <label>{{item}}</label>  
                </li>  
            </ul>  
        </div>  
    </div>  
</div>  
  
  
Ex: Table  
  
for-demo.component.ts  
  
public products:any[] = [  
    {Name: "Samsung TV", Price:45000.44},  
    {Name: "Nike Casuals", Price:2600.44}  
  ];  
  
  
for-demo.component.html  
  
<div class="container-fluid">  
    <h2>Products List</h2>  
    <table class="table table-hover table-dark">  
        <thead>  
            <tr>  
                <th>Name</th>  
                <th>Price</th>  
            </tr>  
        </thead>  
        <tbody>  
            <tr \*ngFor="let product of products">  
                <td>{{product.Name}}</td>  
                <td>{{product.Price}}</td>  
            </tr>  
        </tbody>  
    </table>  
</div>  
  
Ex: Nested Iterations  
  
for-demo.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-for-demo',  
  templateUrl: './for-demo.component.html',  
  styleUrls: ['./for-demo.component.css']  
})  
export class ForDemoComponent implements OnInit {  
  
   public menu:any[] = [  
     {Category:"Electronics", Products:["TV", "Mobile"]},  
     {Category:"Footwear", Products:["Casuals","Sneakers","Boots"]}  
   ]  
   
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  
}  
  
  
for-demo.component.html  
  
<div class="container-fluid">  
   <div class="row">  
     <div class="col">  
        <h2>Menu</h2>  
        <ol>  
            <li \*ngFor="let item of menu">  
                 {{item.Category}}  
                 <ul>  
                    <li \*ngFor="let product of item.Products">  
                        {{product}}  
                    </li>  
                 </ul>  
            </li>  
        </ol>  
     </div>  
     <div class="col">  
        <h2>Select Product</h2>  
        <select class="form-select">  
            <optgroup \*ngFor="let item of menu" [label]="item.Category">  
                <option \*ngFor="let product of item.Products">  
                    {{product}}  
                </option>  
            </optgroup>  
        </select>  
     </div>  
     <div class="col">  
        <h2>Menu</h2>  
        <details \*ngFor="let item of menu">  
            <summary>{{item.Category}}</summary>  
            <ul>  
                <li \*ngFor="let product of item.Products">  
                    {{product}}  
                </li>  
            </ul>  
        </details>  
     </div>  
   </div>  
</div>

19/11/2022

api.nasa.gov  
  
Ex: Presenting Nasa API in Table  
  
nasa-api.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-nasa-api',  
  templateUrl: './nasa-api.component.html',  
  styleUrls: ['./nasa-api.component.css']  
})  
export class NasaApiComponent implements OnInit {  
  
  public marsObject:any = {};  
  
  public GetApiData(){  
     fetch("[https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos?sol=1000&api\_key=DEMO\_KEY&quot](https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos?sol=1000&api_key=DEMO_KEY&quot" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTA3Mzg1MTAyMzU4/_blank);)  
     .then(response=> response.json())  
     .then(data => {  
        this.marsObject = data;  
     })  
  }  
  
  constructor() { }  
  
  ngOnInit(): void {  
     this.GetApiData();  
  }  
  
}  
  
  
nasa-api.component.html  
  
<div class="container-fluid">  
  <h2>Mars Rover Photos</h2>  
  <table class="table table-hover">  
     <thead>  
        <tr>  
            <th>Photo Id</th>  
            <th>Camera Name</th>  
            <th>Preview</th>  
            <th>Rover Name</th>  
        </tr>  
     </thead>  
     <tbody>  
         <tr \*ngFor="let item of marsObject.photos">  
             <td>{{item.id}}</td>  
             <td>{{item.camera.full\_name}}</td>  
             <td>  
                <img [src]="item.img\_src" width="100" height="100">  
             </td>  
             <td>  
                {{item.rover.name}}  
             </td>  
         </tr>  
     </tbody>  
  </table>  
</div>  
  
Ex: Card-Style  
  
nasa-api.component.html  
  
<div class="container-fluid">  
  <h2>Mars Rover Photos</h2>  
  <div class="d-flex flex-wrap">  
      <div \*ngFor="let item of marsObject.photos" class="card m-2 p-2" style="width:200px;">  
        <img [src]="item.img\_src" class="card-img-top" height="150">  
        <div class="card-header">  
            <h3>{{item.id}}</h3>  
        </div>  
        <div class="card-body">  
            <dl>  
                <dt>Camera Name</dt>  
                <dd>{{item.camera.full\_name}}</dd>  
                <dt>Rover Name</dt>  
                <dd>{{item.rover.name}}</dd>  
            </dl>  
        </div>  
        <div class="card-footer">  
            <button class="btn btn-dark w-100">More Details..</button>  
        </div>  
      </div>  
  </div>  
</div>  
  
                                    fakestoreapi.com  
  
/products                                    [{ }]  
/products/1                                { }  
/products/categories                    [" "]  
/products/category/jewelery            [{ }]  
  
  
Ex:  
shop.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-shop',  
  templateUrl: './shop.component.html',  
  styleUrls: ['./shop.component.css']  
})  
export class ShopComponent implements OnInit {  
  
  public Categories:string[] = [];  
  public Products:any[] = [];  
  
  public LoadCaegories(){  
     fetch("[http://fakestoreapi.com/products/categories&quot](http://fakestoreapi.com/products/categories&quot" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTA3Mzg1MTAyMzU4/_blank);)  
     .then(response=> response.json())  
     .then(categories => {  
        categories.unshift("all");  
        this.Categories = categories;  
     })  
  }  
  
  public LoadProducts(url:string){  
    fetch(url)  
    .then(response=>response.json())  
    .then(products=>{  
       this.Products = products;  
    })  
  }  
  
  
  constructor() { }  
  
  ngOnInit(): void {  
    this.LoadCaegories();  
    this.LoadProducts("[http://fakestoreapi.com/products&quot](http://fakestoreapi.com/products&quot" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTA3Mzg1MTAyMzU4/_blank););  
  }  
  
  public CategoryChanged(category:string):void{  
     if(category=="all"){  
       this.LoadProducts("[http://fakestoreapi.com/products&quot](http://fakestoreapi.com/products&quot" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTA3Mzg1MTAyMzU4/_blank););  
     } else {  
      this.LoadProducts(`[http://fakestoreapi.com/products/category/$](http://fakestoreapi.com/products/category/$" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTA3Mzg1MTAyMzU4/_blank){category}`);  
     }  
  }  
  
}  
  
  
shop.component.html  
  
<div class="container-fluid">  
    <header class="d-flex bg-dark text-white  p-2 justify-content-between">  
        <div class="brand-name">Shopper.</div>  
        <div>  
            <span><a (click)="CategoryChanged('all')">Home</a></span>  
            <span><a (click)="CategoryChanged('electronics')">Electronics</a></span>  
            <span><a (click)="CategoryChanged('jewelery')">Jewelery</a></span>  
            <span><a>Men's Clothing</a></span>  
            <span><a>Women's Clothing</a></span>  
        </div>  
        <div>  
            <span class="bi bi-search"></span>  
            <span class="bi bi-person"></span>  
            <span class="bi bi-heart"></span>  
            <span class="bi bi-cart4"></span>  
        </div>  
    </header>  
    <section class="row mt-2">  
        <nav class="col-2">  
          <div>  
            <div>  
                <select name="Category" ngModel #Category="ngModel" (change)="CategoryChanged(Category.value)" class="form-select">  
                    <option \*ngFor="let category of Categories" [value]="category">  
                        {{category | uppercase}}  
                    </option>  
                </select>  
            </div>  
          </div>  
        </nav>  
        <main class="col-10 d-flex flex-wrap overflow-auto" style="height: 500px;">  
            <div \*ngFor="let item of Products" class="card m-2 p-2" style="width: 200px;">  
                <img [src]="item.image" class="card-img-top" height="150">  
                <div class="card-header" style="height:140px">  
                    <p>{{item.title}}</p>  
                </div>  
                <div class="card-body">  
                    <dl>  
                        <dt>Price</dt>  
                        <dd>{{item.price}}</dd>  
                        <dt>Rating</dt>  
                        <dd>  
                            <span class="bi bi-star-fill text-success"></span>  
                            {{item.rating.rate}} [{{item.rating.count}}]  
                        </dd>  
                    </dl>  
                </div>  
                <div class="card-footer">  
                    <button class="btn btn-danger w-100">  
                        <span class="bi bi-cart2"></span>  
                        Add to Cart  
                    </button>  
                </div>  
            </div>  
        </main>  
    </section>  
</div>  
  
shop.component.css  
  
.brand-name {  
    font-size: 20px;  
    font-weight: bold;  
}  
header span {  
    margin-right: 20px;  
    font-weight: bold;  
}  
a {  
    cursor: grab;  
}

21/11/2022

NgFor Properties  
  
Property                Type                Description  
----------------------------------------------------------------------------------  
index                    number            It returns repeating item index number.  
odd                    boolean]            It returns true if repeating item is at odd  
even                    boolean]            even, first, last occurance  
first                    boolean]      
last                    boolean]  
trackBy                function            It returns the changes in iteration.  
                        pointer            NgFor will perform iteration only over the  
                                            changed index.  
  
  
Syntax:  
    <li  \*ngFor="let  item of  collection; let i=index; let odd=odd">  
  
  
Syntax:  
    public TrackChange(index:number)  
    {  
     return index;  
    }  
  
    <tr  \*ngFor="let item of collection; trackBy:TrackChange">  
  
Ex:  
 forproperties.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-forproperties',  
  templateUrl: './forproperties.component.html',  
  styleUrls: ['./forproperties.component.css']  
})  
export class ForpropertiesComponent implements OnInit {  
  
  public products:any[] = [  
    {Name:"TV", Price:46000.44},  
    {Name:"Mobile", Price:13000.44},  
    {Name:"Shoe", Price:2300.44}  
  ];  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  public DeleteClick(index:number):void {  
     let flag = confirm("Are you sure?\nWant to Delete?");  
     if(flag==true){  
      this.products.splice(index,1);  
     }  
  }  
  public AddMoreClick():void {  
      alert("Add Clicked");  
      this.products = [  
        {Name:"TV", Price:46000.44},  
        {Name:"Mobile", Price:13000.44},  
        {Name:"Shoe", Price:2300.44},  
        {Name:"Watch", Price:5000.44}  
      ];  
  }  
  public TrackChange(index:number){  
    return index;  
  }  
}  
  
  
forproperties.component.html  
  
<div class="container-fluid">  
    <h2>Product Details <button (click)="AddMoreClick()" class="btn btn-danger">Add More</button></h2>  
    <table class="table table-hover">  
        <thead>  
            <tr>  
                <th>Name</th>  
                <th>Price</th>  
                <th>Index</th>  
                <th>Odd</th>  
                <th>Even</th>  
                <th>First</th>  
                <th>Last</th>  
                <th>Actions</th>  
            </tr>  
        </thead>  
        <tbody>  
            <tr [class.odd]="o" [class.even]="e" \*ngFor="let item of products; let i=index; let o=odd; let e=even; let f=first; let l=last; trackBy:TrackChange">  
                <td>{{item.Name}}</td>  
                <td>{{item.Price}}</td>  
                <td>{{i}}</td>  
                <td>{{o}}</td>  
                <td>{{e}}</td>  
                <td>{{f}}</td>  
                <td>{{l}}</td>  
                <td>  
                    <button (click)="DeleteClick(i)" class="btn btn-danger">  
                        <span class="bi bi-trash-fill"></span>  
                    </button>  
                </td>  
            </tr>  
        </tbody>  
    </table>  
</div>  
  
forproperties.component.css  
  
.odd {  
    background-color: aquamarine;  
}  
.even {  
    background-color: burlywood;  
}  
  
  
                                    Shopping Example  
  
shop.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-shop',  
  templateUrl: './shop.component.html',  
  styleUrls: ['./shop.component.css']  
})  
export class ShopComponent implements OnInit {  
  
  public Categories:string[] = [];  
  public Products:any[] = [];  
  public CarItems:any[] = [];  
  public CartItemsCount:number = 0;  
  public ToggleCart:boolean = false;  
  
  public ToggleClick():void{  
    this.ToggleCart = (this.ToggleCart==false)?true:false;  
  }  
  
  public LoadCaegories(){  
     fetch("[http://fakestoreapi.com/products/categories&quot](http://fakestoreapi.com/products/categories&quot" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTcyNDQxNTAzNDUx/_blank);)  
     .then(response=> response.json())  
     .then(categories => {  
        categories.unshift("all");  
        this.Categories = categories;  
     })  
  }  
  
  public LoadProducts(url:string){  
    fetch(url)  
    .then(response=>response.json())  
    .then(products=>{  
       this.Products = products;  
    })  
  }  
  
  
  constructor() { }  
  
  ngOnInit(): void {  
    this.LoadCaegories();  
    this.LoadProducts("[http://fakestoreapi.com/products&quot](http://fakestoreapi.com/products&quot" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTcyNDQxNTAzNDUx/_blank););  
  }  
  
  public CategoryChanged(category:string):void{  
     if(category=="all"){  
       this.LoadProducts("[http://fakestoreapi.com/products&quot](http://fakestoreapi.com/products&quot" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTcyNDQxNTAzNDUx/_blank););  
     } else {  
      this.LoadProducts(`[http://fakestoreapi.com/products/category/$](http://fakestoreapi.com/products/category/$" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTcyNDQxNTAzNDUx/_blank){category}`);  
     }  
  }  
  public AddToCartClick(id:number):void {  
     fetch(`[http://fakestoreapi.com/products/$](http://fakestoreapi.com/products/$" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTcyNDQxNTAzNDUx/_blank){id}`)  
     .then(res=>res.json())  
     .then(product=>{  
        this.CarItems.push(product);  
        alert(`${product.title}\nAdd to Cart`);  
        this.GetItemsCount();  
     })  
  }  
  
  public GetItemsCount():void {  
    this.CartItemsCount = this.CarItems.length;  
  }  
}  
  
  
shop.component.html  
  
<div class="container-fluid">  
    <header class="d-flex bg-dark text-white  p-2 justify-content-between">  
        <div class="brand-name">Shopper.</div>  
        <div>  
            <span><a (click)="CategoryChanged('all')">Home</a></span>  
            <span><a (click)="CategoryChanged('electronics')">Electronics</a></span>  
            <span><a (click)="CategoryChanged('jewelery')">Jewelery</a></span>  
            <span><a>Men's Clothing</a></span>  
            <span><a>Women's Clothing</a></span>  
        </div>  
        <div>  
            <span class="bi bi-search"></span>  
            <span class="bi bi-person"></span>  
            <span class="bi bi-heart"></span>  
            <button (click)="ToggleClick()" class="btn btn-light"><span class="bi bi-cart4"></span> [{{CartItemsCount}}]</button>  
             
        </div>  
    </header>  
    <section class="row mt-2">  
        <nav class="col-1">  
          <div>  
            <div>  
                <select name="Category" ngModel #Category="ngModel" (change)="CategoryChanged(Category.value)" class="form-select">  
                    <option \*ngFor="let category of Categories" [value]="category">  
                        {{category | uppercase}}  
                    </option>  
                </select>  
            </div>  
          </div>  
        </nav>  
        <main class="col-7 d-flex flex-wrap overflow-auto" style="height: 500px;">  
            <div \*ngFor="let item of Products" class="card m-2 p-2" style="width: 200px;">  
                <img [src]="item.image" class="card-img-top" height="150">  
                <div class="card-header" style="height:140px">  
                    <p>{{item.title}}</p>  
                </div>  
                <div class="card-body">  
                    <dl>  
                        <dt>Price</dt>  
                        <dd>{{item.price}}</dd>  
                        <dt>Rating</dt>  
                        <dd>  
                            <span class="bi bi-star-fill text-success"></span>  
                            {{item.rating.rate}} [{{item.rating.count}}]  
                        </dd>  
                    </dl>  
                </div>  
                <div class="card-footer">  
                    <button (click)="AddToCartClick(item.id)" class="btn btn-danger w-100">  
                        <span class="bi bi-cart2"></span>  
                        Add to Cart  
                    </button>  
                </div>  
            </div>  
        </main>  
        <aside class="col-4">  
           <div \*ngIf="ToggleCart">  
            <h3>Your Cart Items</h3>  
            <table class="table table-hover">  
                <thead>  
                    <tr>  
                        <th>Title</th>  
                        <th>Preview</th>  
                        <th>Price</th>  
                    </tr>  
                </thead>  
                <tbody>  
                    <tr \*ngFor="let item of CarItems">  
                        <td>{{item.title}}</td>  
                        <td><img [src]="item.image" width="50" height="50"></td>  
                        <td>{{item.price}}</td>  
                    </tr>  
                </tbody>  
            </table>  
           </div>  
        </aside>  
    </section>  
</div>

23/11/2022

Structural Directives  
- NgIF  
- NgSwitch  
- NgFor  
  
Attribute Directives  
- NgModel  
- NgClass  
- NgStyle  
  
                                        NgModel  
  
- It is a member of "FormsModule" defined in "@angular/forms" library.  
- NgModel transforms a static HTML element into Dynamic Element.  
- Angular allocates memory for element in order to store its value and use across requests.  
  
Syntax:  
    <input type="text" ngModel  name="UserName"  #UserName="ngModel">  
    <input type="text" [(ngModel)]="UserName">  
  
                                        NgClass  
                                    [Class Binding]  
- It is used to configure a CSS class to any element dynamically.  
- CSS classes are defined in 3 ways  
  
        a) String Reference  
        b) Array Reference  
        c) Object Reference  
  
String Reference: It is used to apply any specific class to element.  
  
        <h2  [ngClass]=" 'className' ">  
  
Array Reference: It is used to to apply multiple classes to element.  
  
        <h2 [ngClass]="['class1', 'class2', ...]">  
  
Object Reference: It is used to turn ON or OFF the class dynamically.  
  
        <h2 [ngClass]="{'class1':true, 'class2':false}">  
  
  
Ex:  
class-demo.component.css  
  
.back-style {  
    background-color: yellow;  
}  
.text-style {  
    color:red;  
    text-align: center;  
}  
.border-style {  
    border:2px solid red;  
}  
  
class-demo.component.html  
  
<div class="container-fluid">  
    <h1 [ngClass]="'back-style'">String Reference</h1>  
    <h1 [ngClass]="['back-style','text-style', 'border-style']">Array Reference</h1>  
    <h1 [ngClass]="{'back-style':false, 'text-style':true, 'border-style':true}" >Object Reference</h1>  
</div>  
  
Ex: Validation Styles  
  
class-demo.component.css  
  
.invalid-style {  
    border: 1px solid red;  
    box-shadow: 2px 2px 2px red;  
}  
.valid-style {  
    border: 1px solid green;  
    box-shadow: 2px 2px 2px green;  
}  
  
class-demo.component.html  
  
<div class="container-fluid">  
   <h2>Login</h2>  
   <dl>  
    <dt>User Name</dt>  
    <dd><input type="text" [ngClass]="{'valid-style':UserName.valid, 'invalid-style':UserName.invalid}" ngModel name="UserName" #UserName="ngModel" required></dd>  
   </dl>  
</div>  
  
  
Ex: String Array Reference  
  
class-demo.component.css  
  
.text-style {  
    text-align: center;  
    color:red;  
}  
.border-style {  
    border: 2px solid red;  
}  
.back-style {  
    background-color: yellow;  
}  
  
class-demo.component.html  
  
<div class="container-fluid">  
   <fieldset>  
    <legend>Apply Styles</legend>  
    <dl>  
        <dt><input type="text" placeholder="eg: text-style, border-style, back-style" ngModel name="Effects" #Effects="ngModel" class="form-control"></dt>  
    </dl>  
   </fieldset>  
   <h1 [ngClass]="Effects.value">Sample Text</h1>  
</div>  
  
  
  
Ex: CheckBox  
  
class-demo.component.css  
  
.text-style {  
    text-align: center;  
    color:red;  
}  
.border-style {  
    border: 2px solid red;  
}  
.back-style {  
    background-color: yellow;  
}  
  
class-demo.component.html  
  
<div class="container-fluid">  
   <fieldset>  
    <legend>Apply Styles</legend>  
     <ul class="list-unstyled">  
        <li><input ngModel name="BackStyle" #BackStyle="ngModel" type="checkbox">Background Effect</li>  
        <li><input ngModel name="TextStyle" #TextStyle="ngModel" type="checkbox">Text Effect</li>  
        <li><input ngModel name="BorderStyle" #BorderStyle="ngModel" type="checkbox">Border Effect</li>  
     </ul>  
   </fieldset>  
   <h1 [ngClass]="{'back-style':BackStyle.value, 'text-style':TextStyle.value, 'border-style':BorderStyle.value}">Sample Text</h1>  
</div>  
  
  
Ex: Theme  
  
class-demo.component.css  
  
.dark-mode {  
    background-color: black;  
    color:white;  
    padding: 10px;  
}  
  
class-demo.component.html  
  
<div class="container-fluid">  
   <div class="d-flex justify-content-center align-items-center" style="height: 500px;">  
        <div  [ngClass]="{'dark-mode':Theme.value}"  class="border border-3 p-2 border-dark">  
            <div class="form-switch">  
                <input class="form-check-input" name="Theme" ngModel #Theme="ngModel" type="checkbox"> Dark Mode  
            </div>  
            <dl>  
                <h3>User Login</h3>  
                <dt>User Name</dt>  
                <dd><input type="text" class="form-control"></dd>  
                <dt>Password</dt>  
                <dd><input type="password" class="form-control"></dd>  
            </dl>  
            <button [ngClass]="{'btn-light':Theme.value}" class="btn btn-dark w-100">Login</button>  
        </div>  
   </div>  
</div>  
  
  
Ex: Dynamic Class for User Validation  
  
class-demo.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-class-demo',  
  templateUrl: './class-demo.component.html',  
  styleUrls: ['./class-demo.component.css']  
})  
export class ClassDemoComponent implements OnInit {  
  
  public Users:any[] = [  
     {UserName: "john"},  
     {UserName: "john12"},  
     {UserName: "john\_nit"},  
     {UserName: "david"}  
  ];  
  public UserName:string = '';  
  public Msg:string = '';  
  public IsSuccess:boolean = false;  
  public IsError:boolean = false;  
   
  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  public VerifyUser(){  
     for(var user of this.Users)  
     {  
        if(user.UserName==this.UserName){  
           this.Msg = "User Name Taken - Try Another";  
           this.IsSuccess = false;  
           this.IsError = true;  
           break;  
        } else {  
           this.Msg = "User Name Available";  
           this.IsSuccess = true;  
           this.IsError = false;  
        }  
     }  
  }  
  
}  
  
  
class-demo.component.html  
  
<div class="container-fluid">  
    <h2>Register User</h2>  
    <dl class="w-25">  
        <dt>User Name</dt>  
        <dd><input type="text" (keyup)="VerifyUser()" [(ngModel)]="UserName" class="form-control"></dd>  
        <dd [ngClass]="{'text-success':IsSuccess, 'text-danger':IsError}">{{Msg}}</dd>  
    </dl>  
</div>  
  
                                         NgStyle  
                                        [Style Binding]  
- It configure inline style dynamically for any element.  
  
Syntax:  
        <div  [ngStyle]="{attribute:value, attribute:value}">  
  
Ex:  
class-demo.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-class-demo',  
  templateUrl: './class-demo.component.html',  
  styleUrls: ['./class-demo.component.css']  
})  
export class ClassDemoComponent implements OnInit {  
  
  public styleObject:any = {  
    'position':'fixed',  
    'top': '',  
    'left': ''  
  }  
  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  
  public MouseMove(e:any):void {  
       this.styleObject = {  
          'position': 'fixed',  
          'left': e.clientX + 'px',  
          'top': e.clientY + 'px'  
       }  
  }  
  
}  
  
  
class-demo.component.html  
  
<div (mousemove)="MouseMove($event)" class="container-fluid d-flex justify-content-center align-items-center" style="height:500px">  
    <p>Move mouse pointer to test</p>  
    <div style="height: 1000px;"></div>  
    <img [ngStyle]="styleObject" src="assets/flag.gif" width="50" height="50">  
</div>

24/11/2022

Summary  
- Components  
- Data Binding  
    a) One Way  
    b) Two Way  
- Class Binding  
- Style Binding  
- Content Projection  
- Directives  
    a) Structural  
    b) Attribute  
    c) Component Directives  
        <app-login>  
  
                                Angular Event Binding  
                                ------------------------------  
FAQ: What is an Event?  
Ans : Event is a message sent by sender to its subscriber in order to notify the                
         change.  
        Event follows a software design pattern called "Observer".  
        Observer is a communication pattern.  
        Event uses delegate mechanism. [Function Pointer]  
  
Syntax:  
        public  InsertClick():void            => Subscriber  
        {  
  
        }  
  
        <button  (click)="InsertClick()">        => Sender  
  
        (click)                        : Event  
        (click)="InsertClick()"    : Event Handler  
  
- Every Event handler by have an default argument "$event".  
- JavaScript default event args are  
        a) this  
        b) event  
- $event sends information about    
        a) object  
        b) event  
  
Syntax: JavaScript  
  
            <button  onclick="InsertClick(this, event)">  
  
           Angular  
  
            <button  (click)="InsertClick($event)">  
  
             $event.clientX, clientY, keyCode, charCode, shiftKey, ctrlKey etc..  
  
             $event.target.id, name, class, value, width, height, src etc..  
  
Ex:  
 component.ts  
  
 import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-class-demo',  
  templateUrl: './class-demo.component.html',  
  styleUrls: ['./class-demo.component.css']  
})  
export class ClassDemoComponent implements OnInit {  
  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  public InsertClick(eventArgs:any){  
    document.write(`  
       Button Id :  ${eventArgs.target.id}  <br>  
       Button Name: ${eventArgs.target.name} <br>  
       Button Class: ${eventArgs.target.className} <br>  
       X Position  : ${eventArgs.clientX} <br>  
       Ctrl Key    : ${eventArgs.ctrlKey}  
  
    `);  
  }  
  
  
}  
  
  
component.html  
  
<div class="container-fluid d-flex justify-content-center align-items-center" style="height:500px">  
    <button (click)="InsertClick($event)" id="btnInsert" name="Insert" class="btn btn-primary">Insert</button>  
</div>  
  
- Angular allows custom args.  
- Angular allows any type of args.  
- Angular allows multiple args.  
- Angular allows both default and custom args.  
  
Ex: Multiple Parameters  
  
.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-class-demo',  
  templateUrl: './class-demo.component.html',  
  styleUrls: ['./class-demo.component.css']  
})  
export class ClassDemoComponent implements OnInit {  
  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  public InsertClick(eventArgs:any, id:number, name:string, stock:boolean, cities:string[], rating:{rate:number, count:number}){  
     document.write(`  
        id    : ${id} <br>  
        Name  : ${name}<br>  
        Stock : ${stock}<br>  
        Cities : ${cities.toString()} <br>  
        Rating  : Rate ${rating.rate} [${rating.count}] <br>  
        Button Name : ${eventArgs.target.name}  
     `);  
  }  
  
  
}  
  
component.html  
  
<div class="container-fluid d-flex justify-content-center align-items-center" style="height:500px">  
    <button (click)="InsertClick($event, 1,'TV',true,['Delhi','Hyd'],{rate:3.4, count:3000})" id="btnInsert" name="Insert" class="btn btn-primary">Insert</button>  
</div>  
  
Ex: Rest Parameters as Event Args  
  
.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-class-demo',  
  templateUrl: './class-demo.component.html',  
  styleUrls: ['./class-demo.component.css']  
})  
export class ClassDemoComponent implements OnInit {  
  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  public InsertClick(...args:any[]){  
     let [id, name, stock, cities, rating] = args;  
     document.write(`  
        Id   : ${id} <br>  
        Name : ${name} <br>  
        Stock : ${stock} <br>  
        Cities : ${cities.toString()} <br>  
        Rating : ${rating.rate} [${rating.count}]  
     `);  
  }  
  
  
}  
  
  
component.html  
  
<div class="container-fluid d-flex justify-content-center align-items-center" style="height:500px">  
    <button (click)="InsertClick(1,'TV',true,['Delhi','Hyd'],{rate:3.4, count:3000})" id="btnInsert" name="Insert" class="btn btn-primary">Insert</button>  
</div>  
  
- Angular can use all JavaScript browser events in the same style.  
- Browser Events are categorized into various groups  
  
1. Mouse Events  
        mouseover  
        mouseout  
        mousedown  
        mouseup  
        mousemove  
  
2. Keyboard Events  
        keyup  
        keydown  
        keypress                        john  
  
3. Button Events  
        click  
        dblclick  
        contextmenu  
  
4. Clipboard Events  
        cut  
        copy  
        paste  
  
5. Element State Events  
        focus  
        blur  
        change  
        select  
        selectstart  
  
6. Form Events  
        submit  
        reset  
  
7. Touch Events  
        touchstart  
        touchend  
        touchmove  
  
8. Timer Events  
        setInterval  
        clearInterval  
        setTimeout  
        clearTimeout  
  
etc...  
  
  
  
  
Ex:  
component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-event-demo',  
  templateUrl: './event-demo.component.html',  
  styleUrls: ['./event-demo.component.css']  
})  
export class EventDemoComponent implements OnInit {  
  
  
  public Users:any[] = [  
    {UserName:"john"},  
    {UserName:"john12"},  
    {UserName:"john\_nit"},  
    {UserName:"david"}  
  ];  
  public userError:string = '';  
  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  public VerifyUser(eventArgs:any){  
      for(var user of this.Users)  
      {  
          if(user.UserName==eventArgs.target.value) {  
              this.userError = "User Name Taken - Try Another";  
              break;  
          } else {  
              this.userError = "User Name Available";  
          }  
      }  
  }  
  
}  
  
  
component.html  
  
<div class="container-fluid">  
    <h3>Register User</h3>  
    <dl>  
        <dt>User Name</dt>  
        <dd><input type="text" (keyup)="VerifyUser($event)"></dd>  
        <dd>{{userError}}</dd>  
    </dl>  
</div>

25/11/2022

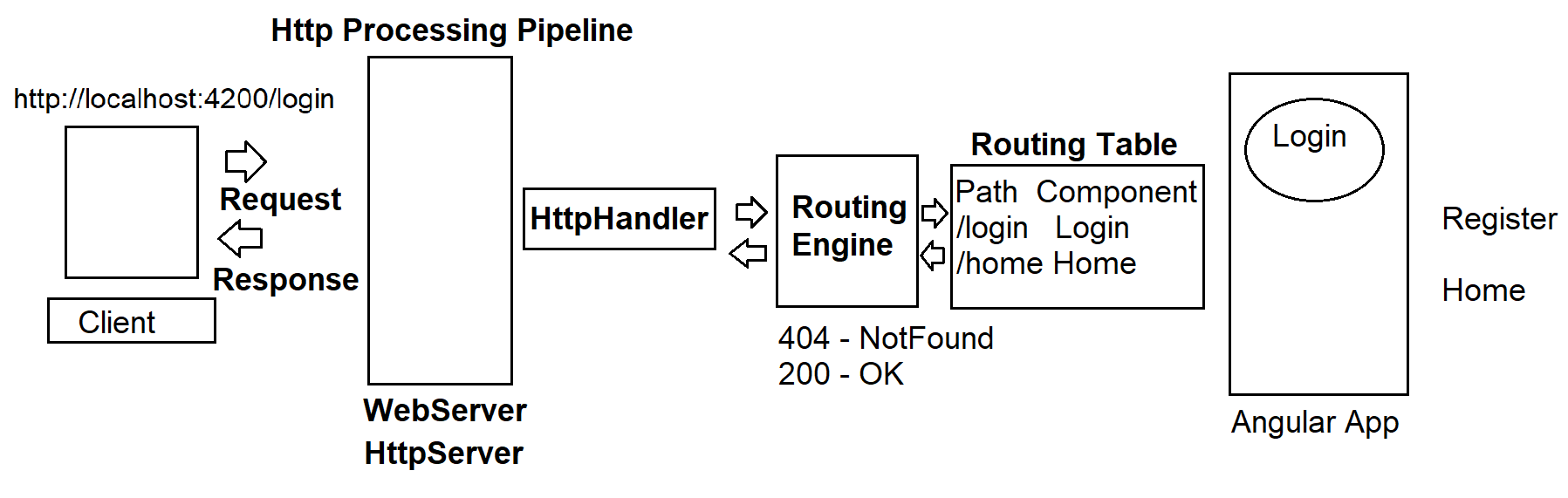
Custom Events  
- Custom Events are user defined events.  
- Usually custom events are required while designing components.  
  
Step-1: Accessing a component in another component.  
  
          <app-parent>  
             <app-child> </app-child>      
             </app-parent>  
  
     
Step-2: Transporting Data from parent to child  
- You have to configure a property in child component that can collect data from parent.  
         
                public  MsgFromParent:string = " ";  
  
- Every property of child component is not suitable to collect data from parent.  
- The property which is marked with "@Input()" directive can access the parent data.  
  
            @Input()  public  MsgFromParent:string = " ";  
  
- Input property can by any type  
        a) Primitive  
        b) Non Primitive  
  
- Child component can configure property binding and access data from parent.  
  
        <app-child  [MsgFromParent]=" ' ' "> </app-child>  
  
  
Step-3: Transporting data from child to parent  
- You need an "EventEmitter"  
- Every Angular event is derived from EventEmitter.  
  
        onclick => click:EventEmitter  
  
Syntax:  
            public  CustomEvent:EventEmitter<T> = new EventEmitter<T>();  
  
  
- Events are configured by using "Event Binding".  
  
            <app-child [propertyName]=""   (CustomEvent)="Method()">  
  
  
- Custom Events are not accessible outside scope.  
  
- Event Emitter emits the value by using "@Output()" directive.  
  
- Event Related Details are collected by using event args "$event"  
  
  
Ex:  
child.component.ts  
  
import { Component, Input, OnInit, EventEmitter, Output } from '@angular/core';  
  
@Component({  
  selector: 'app-child',  
  templateUrl: './child.component.html',  
  styleUrls: ['./child.component.css']  
})  
export class ChildComponent implements OnInit {  
  
  @Input() public MsgFromParent:string = "";  
  @Input() public ProductDetails:{Name:string, Price:number} = {Name: '', Price: 0};  
  
  @Output() public ChildComponentClick:EventEmitter<string> = new EventEmitter<string>();  
  @Output() public ChildComponentChange:EventEmitter<any> = new EventEmitter<any>();  
  
  public SendClick():void {  
     this.ChildComponentClick.emit('Hello ! from Child');  
     this.ChildComponentChange.emit({Name:'TV', Price:4500.44});  
  }  
  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  
}  
  
child.component.html  
  
<div style="height:250px" class="container-fluid m-4 p-4 bg-light text-dark">  
   <h4>Child Component <button (click)="SendClick()" class="btn btn-dark">Msg to Parent</button> </h4>  
   <p>{{MsgFromParent}}</p>  
   <dl>  
    <dt>Name</dt>  
    <dd>{{ProductDetails.Name}}</dd>  
    <dt>Price</dt>  
    <dd>{{ProductDetails.Price}}</dd>  
   </dl>  
</div>  
  
parent.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-parent',  
  templateUrl: './parent.component.html',  
  styleUrls: ['./parent.component.css']  
})  
export class ParentComponent implements OnInit {  
  
  constructor() { }  
  
  public msg:string = "";  
  public product:{Name:string, Price:number} = {Name:'', Price: 0};  
  public msgFromChild:string = "";  
  public obj:any = {};  
  
  ngOnInit(): void {  
  }  
  public SendClick(){  
    this.msg = "Hello ! from Parent";  
    this.product = {  
      Name: 'Samsung TV',  
      Price: 46000.44  
    }  
  }  
  public CollectMsg(e:string){  
    this.msgFromChild = e;  
  }  
  public CollectObject(e:any){  
    this.obj = e;  
  }  
  
}  
  
  
parent.component.html  
  
<div style="height:400px" class="container-fluid m-2 p-4 bg-dark text-white">  
    <p>{{msgFromChild}}</p>  
    <div>  
        Name: {{obj.Name}}  <br>  
        Price: {{obj.Price}}  
    </div>  
   <h2>Parent Component <button (click)="SendClick()" class="btn btn-light">Send Msg to Child</button> </h2>  
   <app-child [MsgFromParent]="msg" [ProductDetails]="product" (ChildComponentChange)="CollectObject($event)"  (ChildComponentClick)="CollectMsg($event)" ></app-child>  
</div>

26/11/2022

Angular Custom Events  
1. @Input()  
2. @Output()  
3. EventEmitter  
4. emit()  
5. $event  
  
Ex:  
 filter.component.ts  
  
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';  
  
@Component({  
  selector: 'app-filter',  
  templateUrl: './filter.component.html',  
  styleUrls: ['./filter.component.css']  
})  
export class FilterComponent implements OnInit {  
  
 @Input() public ElectronicsCount:number = 0;  
 @Input() public JeweleryCount:number = 0;  
 @Input() public MensCount:number = 0;  
 @Input() public WomensCount:number = 0;  
  
 @Output() public FilterClick:EventEmitter<any> = new EventEmitter<any>();  
 @Output() public FilterChange:EventEmitter<any> = new EventEmitter<any>();  
  
  public ButtonClick(e:any){  
     this.FilterClick.emit(e.target.name);  
  }  
  public SelectChange(e:any){  
     this.FilterChange.emit(e.target.value);  
  }  
  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  
}  
  
  
filter.component.html  
  
<div>  
   <div class="btn-group-vertical w-100">  
    <button name="electronics" (click)="ButtonClick($event)" class="btn btn-dark mb-2 d-flex justify-content-between">Electronics <span class="badge bg-primary">{{ElectronicsCount}}</span> </button>  
    <button name="jewelery" (click)="ButtonClick($event)" class="btn btn-dark mb-2 d-flex justify-content-between">Jewelery <span class="badge bg-primary">{{JeweleryCount}}</span> </button>  
    <button name="men's clothing" (click)="ButtonClick($event)" class="btn btn-dark mb-2 d-flex justify-content-between">Men's Clothing <span class="badge bg-primary">{{MensCount}}</span></button>  
    <button name="women's clothing" (click)="ButtonClick($event)" class="btn btn-dark mb-2 d-flex justify-content-between">Women's Clothing <span class="badge bg-primary">{{WomensCount}}</span></button>  
   </div>  
   <div class="mt-2">  
      <select (change)="SelectChange($event)" class="form-select">  
        <option value="electronics">Electronics</option>  
        <option value="jewelery">Jewelery</option>  
        <option value="men's Clothing">Mens Clothing</option>  
        <option value="women's Clothing">Womens Clothing</option>  
      </select>  
   </div>  
</div>  
  
product-catalog.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-product-catalog',  
  templateUrl: './product-catalog.component.html',  
  styleUrls: ['./product-catalog.component.css']  
})  
export class ProductCatalogComponent implements OnInit {  
  
  public Products:any[] = [];  
  public eCount:number = 0;  
  public jCount:number = 0;  
  public mCount:number = 0;  
  public wCount:number = 0;  
  
  public LoadProducts(url:string):void {  
     fetch(url)  
     .then(response=>response.json())  
     .then(products=>{  
       this.Products = products;  
       this.eCount = products.filter((product:any)=> product.category=='electronics').length;  
       this.jCount = products.filter((product:any)=> product.category=='jewelery').length;  
       this.mCount = products.filter((product:any)=> product.category=="men's clothing").length;  
       this.wCount = products.filter((product:any)=> product.category=="women's clothing").length;  
     })  
  }  
  
  public GetCategoryName(e:any) {  
     this.LoadProducts(`[http://fakestoreapi.com/products/category/$](http://fakestoreapi.com/products/category/$" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTA3NzI5NTI2NDcz/_blank){e}`);  
  }  
  public CategoryChange(e:any){  
    this.LoadProducts(`[http://fakestoreapi.com/products/category/$](http://fakestoreapi.com/products/category/$" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTA3NzI5NTI2NDcz/_blank){e}`);  
  }  
  
  constructor() { }  
  
  ngOnInit(): void {  
    this.LoadProducts('[http://fakestoreapi.com/products&#39](http://fakestoreapi.com/products&" \l "39" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTA3NzI5NTI2NDcz/_blank););  
  }  
  
}  
  
product-catalog.component.html  
  
<div class="container-fluid">  
    <header class="bg-dark p-1 text-white text-center">  
        <h2>Shopping Online</h2>  
    </header>  
    <section class="mt-3 row">  
        <nav class="col-3">  
            <h3>Filter Products</h3>  
            <app-filter (FilterChange)="CategoryChange($event)" (FilterClick)="GetCategoryName($event)" [WomensCount]="wCount" [MensCount]="mCount" [ElectronicsCount]="eCount" [JeweleryCount]="jCount"></app-filter>  
        </nav>  
        <main class="col-9">  
            <h3>Product Catalog</h3>  
            <div class="d-flex flex-wrap overflow-auto" style="height: 500px;">  
                <div style="width: 200px;" class="card m-2 p-2" \*ngFor="let item of Products">  
                    <img [src]="item.image" class="card-img-top" height="140">  
                    <div class="card-header" style="height:130px">  
                        <p>{{item.title}}</p>  
                    </div>  
                </div>  
            </div>  
        </main>  
    </section>  
</div>

27/11/2022

Component Life Cycle Hooks  
  
1. Angular Request Flow  
  
- Client Makes a request to angular application on Web Server  
  
        [http://localhost:4200/login](http://localhost:4200/login" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTM5OTQzNDk4Mzkw/_blank)  
  
- Request is processed in Web Server by HttpHandler within the "Processing Pipeline".  
  
- Handler uses a Routing Engine which maps to the Route Table in application.  
  
- Routing Engine verifies the requested URL path with the path configured in Route Table.  
  
- If requested path is not matching with Route Path then it returns "404" status.  
  [Not Found]  
  
- It requested path is found then it loads the requested component into memory.  
  
- The component life cycle is initiated.  
  
- The component life cycle comprises of various phases from initialization to destroying.  
  
- The life cycle phases are managed by a set of methods known as Life Cycle Hooks.  
  
2. Life Cycle Phases  
- These are a set of phases that occur in a sequential order.  
- Life cycle of component starts when component is requested.  
- Life Cycle of component ends when another component is requested or the application is closed.  
- These phases are managed by a set of methods.  
  
Phase-1:    
- It is managed by "ngOnChanges()"  
- Initialization of references are managed in this phase.  
  
            public Name:string = "John";  
            public Name:string = undefined = "John"  
  
- Binding the value to UI is also managed in this phase.  
  
            <input type="text" [value]="Name">      
  
- It identifies the changes in UI and updates to reference.  
                     
- Angular built-in events are managed in this phase.  
    [click, keyup, mouseover etc..]  
  
- NgModel manages data binding  
  
- Styles and Classes are also binded in this phase.  
  
Summary of NgOnChanges  
- Initialization  
- Property Binding  
- Attribute Binding  
- Interpolation  
- Data Binding  
- Event Binding [Not Custom Events]  
- Class Binding  
- Style Binding  
  
- All the above techniques are reffered as "Change Detection".  
- The changes are identified by "SimpleChanges" object.  
            PreviousValue , CurrentValue  
  
  
Ex:  
 output-demo.component.ts  
  
 import { Component, OnInit, Input, OnChanges, SimpleChanges } from '@angular/core';  
  
@Component({  
  selector: 'app-output-demo',  
  templateUrl: './output-demo.component.html',  
  styleUrls: ['./output-demo.component.css']  
})  
export class OutputDemoComponent implements OnChanges {  
  
 @Input() public UserName:string|undefined;  
 public PreviousValue:any;  
 public CurrentValue:any;  
 public msg:any;  
  
  constructor() { }  
  
  ngOnChanges(changes: SimpleChanges): void {  
     for(var property in changes)  
     {  
        let change = changes[property];  
        this.PreviousValue = change.previousValue;  
        this.CurrentValue = change.currentValue;  
     }  
     if(this.PreviousValue==this.CurrentValue) {  
       this.msg = "No Change Detected";  
     } else {  
      this.msg = "Change Detected";  
     }  
  }  
  
}  
  
  
output-demo.component.html  
  
<p>Hello ! {{UserName}}</p>  
<h3>Changes</h3>  
{{msg}}  
<h3>Change Details</h3>  
<dl>  
    <dt>Previous Value</dt>  
    <dd>{{(PreviousValue==undefined)?"Undefined":PreviousValue}}</dd>  
    <dt>Current Value</dt>  
    <dd>{{CurrentValue}}</dd>  
</dl>  
  
input-demo.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-input-demo',  
  templateUrl: './input-demo.component.html',  
  styleUrls: ['./input-demo.component.css']  
})  
export class InputDemoComponent implements OnInit {  
  
  public Uname:string|undefined;  
  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  
}  
  
  
input-demo-component.html  
  
<div class="container-fluid">  
    <h2>Your Name</h2>  
    <input type="text" [(ngModel)]="Uname">  
    <h3>Output Component</h3>  
    <app-output-demo [UserName]="Uname"></app-output-demo>  
</div>  
  
Phase-2:  
- This phase is managed by "ngOnInit()"  
- It allocates memory for current component  
- It is shared to components which are working within context of current component.  
- It configure explicit memory for component, which is known as "Context Memory".  
  
FAQ: What is difference between Constructor and Context?  
Ans : Constructor is private to the current component.  
        Context is shared to another component that runs within the component.  
  
- Child Component uses "@Input()" to accept data from the context memory of parent.  
  
Phase-3  
- This phase is managed by "ngDoCheck()"  
- Some actions are not implicitly managed.  
- You have to create explicit actions.  
- EventEmitter is configured  
- Custom Events are trigged.  
- @Output()  
  
Phase-4  
- This is managed by ngAfterContentInit()  
- The output is generated for all actions.  
- The layout is generated.  
- The Render Process completed.  
- This phase manages only content defined by implicit actions.  
  
Phase-5  
- This is managed by  ngAfterContentChecked()  
- Explicit actions related [custom events] layout is generated  
- It is rendered.  
  
  
Phase-6  
- This phase is managed by  ngAfterViewInit()  
- The implicit templates are rendered into UI.  
- The painting process for implicit actions is completed.  
- Output is ready in View. [UI]  
        NgIf, ngSwitch, NgFor  
  
  
Phase-7  
- This phase is managed by ngAfterViewChecked()  
- Explicit templates are rendered and painted.  
        <ng-template>  
- TemplateRef<>  
- Content Projection  
  
Phase-8  
- This phase is managed by "ngOnDestroy()"  
- It will un subscribe the methods.  
  
        public InsertClick():void { }  
        <button (click)="InsertClick()">  
  
- Events are disabled  
- Memory is destroyed  
- Values are erased  
- Component is cleaned up.  
  
FAQ: When a component is destroyed?  
Ans:  When any another component is requested.



28/11/2022

Angular Pipes  
            -------------------  
- Pipe is used to transform data.  
- Angular Pipe is a class which have a "transform()" method.  
- It takes a value as input and transforms the value according to requirement and returns the transformed value.  
- Angular Provides pre-defined pipes and also allows to create custom pipes.  
  
Angular Built-Pipes:  
---------------------------

Angular Pipes

- Pipes are used to “Transform” data.

- Data accessed from various sources and

presented in UI will not be exactly as

expected.

- Hence, we have to transform data to

display in User Friendly Format.

- Angular provides “Built-in-Pipes” to

transform data.

- Angular also allows to create custom

pipes.

- Every Pipe in angular is class implements

“PipeTransform” base

- Every pipe related functionality is defined

by using “transform()”.

Angular Built-in Pipes

- AsyncPipe

- CurrencyPipe

- DatePipe

- DecimalPipe

- I18nPluralPipe

- I18mSelectPipe

- JsonPipe

- KeyValuePipe

- LowerCasePipe

- UpperCasePipe

- TitleCasePipe

- SlicePipe

- Pipes are configured for data in order to format, filter, sort or evaluate.  
- A pipe is attached to your data by using " | " .  
  
        data | pipeName  
  
  
1. UppercasePipe        : It converts all letters to capital letter.  
2. LowerCasePipe        : It converts to lower case.  
3. TitleCasePipe        : It converts the first letter of every word to capital letter.  
  
    Syntax:  
                {{ title | uppercase }}  
  
4. DecimalPipe            : It is used to display numeric values.  
  
    Syntax:  
                public  Price = 45000.50;  
  
                {{ Price | number }}  
                {{ Price | number:'.2' }}  
                {{ Price | number:'5.2' }}  
  
5. CurrencyPipe        : It is similar to number pipe but have currency symbol.  
  
    Syntax:  
                {{ Price | currency }}  
                {{ Price | currency:'INR' }}            [USD, GB]  
                {{ Price | currency:'&#8377;'}}  
  
  
6. DatePipe                : It can display date in various formats.  
  
                 
Syntax:      
            public  Mfd:Date = new Date("2022-10-20");  
  
            {{ Mfd | date }}  
            {{ Mfd | date: 'formatName' }}    fullDate, shortDate, fullTime shortTime  
            {{ Mfd | date: 'customFormat' }}  
  
            Custom Format:  "MM = Month", "dd = date", "yyyy = year"  
  
            M        - month number 1 digit      
            MM    - month number 2 digits  
            MMM    - short month  
            MMMM - long month  
         
            d        - date  
            dd        - date 2 digits  
  
            yy        - year 2 digits  
            yyyy    - year 4 digits  
  
Syntax:  
            {{ Mfd | data : 'dd-MMM-yy' }}  
  
  
7. SlicePipe        :  It can extract the value between sepcified index and return.  
  
Syntax:  
        public Name:string = "Samsung Tv";  
  
        {{ Name | slice:0:7 }}  
  
        <li \*ngFor="let item of collection | slice:0:3">  {{ item }} </li>  
  
8. JsonPipe        : It converts the data into JSON format.  
  
  
        {{ data | json }}  
  
  
9. KeyValuePipe :  It is used to return all key and value from an iterator.  
  
Syntax:  
            <li  \*ngFor="let item of collection | keyvalue">  
  
                {{ item.key }}  -  {{ item.value }}  
  
            </li>  
  
10. i18nSelectPipe :  It performs iteration over the collection and checks the  
                            given condition so that it can return suitable value.  
  
Syntax:  
            {{ item.city | i18nselect:statusMessageObject }}  
  
  
11. i18nPluralPipe  
  
12. Async

29/11/2022

i18nPlural  
- It can identify the collection is refering to one or multiple elements.  
- If it is one then Singular.  
- If it is many then Plural.  
  
                let  product;  
                let  products;  
  
Ex:  
pipe-demo.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-pipe-demo',  
  templateUrl: './pipe-demo.component.html',  
  styleUrls: ['./pipe-demo.component.css']  
})  
export class PipeDemoComponent implements OnInit {  
  
  public Messages:any[] = [];  
  public Message:string = "";  
  public ShowMessages:boolean = false;  
  
  
  public SendClick():void {  
     let now = new Date();  
     let time = now.toLocaleTimeString();  
     this.Messages.push(this.Message + "-" + time);  
     this.Message = "";  
  }  
  public MessageClick():void {  
     this.ShowMessages = (this.ShowMessages==false)?true:false;  
  }  
  public ClearClick():void {  
     this.Messages = [];  
  }  
  
  public msgStatus = {  
     "=0" : "No Messages",  
     "=1" : "You have one New Message",  
     "other": "# Messages"  
  }  
  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  
}  
  
  
pipe-demo.component.html  
  
<div class="container-fluid">  
   <div>  
      <label>Your Message</label>  
      <div>  
         <textarea [(ngModel)]="Message" rows="4" cols="40"></textarea>  
      </div>  
      <div class="mt-2">  
         <button (click)="SendClick()" class="btn btn-primary">Send Message</button>  
      </div>  
   </div>  
   <div class="d-flex justify-content-center align-items-center" style="height:400px">  
      <div>  
         <div>  
            <button (click)="MessageClick()" style="width:500px" class="btn btn-primary btn-lg position-relative">  
               {{Messages.length | i18nPlural:msgStatus}}  <span class="bi bi-envelope"></span> <span class="badge bg-dark position-absolute start-100 top-0">{{Messages.length}}</span>  
             </button>  
         </div>  
         <div class="mt-3">  
            <button (click)="ClearClick()"  style="width:500px" class="btn btn-danger">  
               <span class="bi bi-trash"></span> Clear Messages  
            </button>  
         </div>  
         <ul \*ngIf="ShowMessages" class="list-unstyled">  
            <li \*ngFor="let item of Messages">  
               {{item}}  
            </li>  
         </ul>  
      </div>  
   </div>  
</div>  
  
Summary  
- uppercase  
- lowercase  
- titlecase  
- decimal / number  
- date  
- json  
- slice  
- currency  
- i18nselect  
- i18nplural  
- async  
  
                                     Custom Pipe  
- Pipe is a class  
- To create a custom pipe, you have to add a new class.  
- The pipe behaviour is implemented by using "PipeTransform" contract.  
  
Syntax:  
            class  Name implements PipeTransform  
            {  
            }  
  
- Angular uses class as Pipe by using "@Pipe()" directive.  
  
Syntax:  
 import  { Pipe, PipeTransform } from "@angular/core";  
  
@Pipe({  
     name: "pipeName"  
 })  
export class Name implements PipeTransform  
{  
  
}  
  
- Every Pipe returns a format using "transform()" method.  
  
    transform(yourValue:any)  
    {  
         transform your value;  
         return your value;  
    }  
  
- Every pipe that you design explicitly must be registered in "app.module.ts".  
  
    declarations : [  
                      YourPipeName  
                    ]  
  
Ex:  
1. Add a new folder "pipes" into app  
  
2. Add a new file  
  
        "sentence-case.pipe.ts"  
  
import { Pipe, PipeTransform } from "@angular/core";  
  
@Pipe({  
    name: "sentence"  
})  
export class SentenceCasePipe implements PipeTransform  
{  
    transform(value:any) {  
        let firstChar = value.charAt(0);  
        let restChars = value.substring(1);  
        let sentence = firstChar.toUpperCase() + restChars.toLowerCase();  
        return sentence;  
    }  
}  
  
3. Go to app.module.ts  
  
    declarations : [  
                SentenceCasePipe  
        ]  
  
4. component.ts  
             
        public title:string = "wElComE to ANguLAr";  
  
5. component.html  
  
            {{ title | sentence }}  
  
  
Note: You can generate pipe using CLI command  
  
        > ng  g  pipe  pipeName --skip-tests  
  
  
Ex:  
1. Open Pipes folder in Integrated Terminal  
  
    > ng  g  pipe  sorting --skip-tests  
  
2. sorting.pipe.ts  
  
import { Pipe, PipeTransform } from '@angular/core';  
  
@Pipe({  
  name: 'sorting'  
})  
export class SortingPipe implements PipeTransform {  
  
  transform(list:any, reverse?:boolean) {  
    if(reverse==true){  
      list.sort();  
      list.reverse();  
      return list;  
    } else {  
      list.sort();  
      return list;  
    }  
  }  
}  
  
3. component.ts  
  
public cities:any[] = ["Mumbai","Delhi","Hyd","Chennai","Bangalore"];  
  
4. component.html  
  
<div class="container-fluid">  
   <h2>Cities </h2>  
   <div>  
      <input type="checkbox" name="reverse" ngModel #reverse="ngModel"> Reverse  
   </div>  
   <ol class="mt-4">  
      <li \*ngFor="let item of cities | sorting:reverse.value">{{item}}</li>  
   </ol>  
</div>  
  
Task: Create a pipe for filtering data.  
- Input must be from textbox  
- find the value from list which is matching with textbox text.  
- return only the value that matches.

30/11/2022

ngular Services  
  
What is a Service?  
- Service is a pre-defined business logic, which you can inject and implement in any component.  
- Service enables Reusability, Maintainability and Testability.  
- Service is a set of factories.  
- Factory is a set of functions and values.  
- Factory uses "Single Call" mechanism.  
- Single call is a design pattern where an object is created every time when a method is requested.  
- Service uses "Single Ton" pattern.  
- Single ton creates an object for first request and it will use the same object across requests.  
- Service uses "DI Mechanism" for implementation.  
  [Dependency Injection]  
- DI Comprises of 2 components  
  
        a) Provider  
        b) Injector  
  
Provider:  
- It is used to locate a service in memory.  
- It looks into  
        a) root  
        b) child  
        c) platform  
- root is memory allocated for project.  
- child is memory allocated for module in project.  
- platform is memory allocated for workspace.  
  
Injector  
- It is used to inject a service into component.  
- Service is injected into the constructor of component.  
- It uses single ton pattern.  
  
Configuring a Service in Angular:  
- Angular Service is a class.  
- Service class comprises of methods known as service methods.  
- The directive for service is "@Injectable()"  
  
Syntax:  
    import  { Injectable } from  "@angular/core";  
  
    @Injectable()  
     export class  ServiceName  
    {  
          public ServiceMethod() {  
             return somedata;  
          }  
    }  
  
- Every service requires a scope to be defined, which is used by "Provider".  
  
     @Injectable(  
         {  
           providedIn: "root | child | platform"  
        }  
     )  
                (or)  
  
     app.module.ts  
  
       providers: [ ServiceName ]  
  
- Service is injected into any component  
  
   export class ComponentClass  
   {  
        constructor(private refName: ServiceName) { }  
   }  
  
- Angular provide several pre-defined services  
  
        a) Routing Service  
        b) Form Builder Service  
        c) Validation Service  
        d) Cookie Service (3rd Party)  
        e) HttpService [GET, POST...] API  
  
Ex:  
1. Add a new folder  
    "services" into app  
  
2. Add a new file  
    "captcha.service.ts"  
  
import { Injectable } from "@angular/core";  
  
@Injectable({  
    providedIn:"root"  
})  
export class CaptchaService  
{  
    public GenerateCode(){  
        let a = Math.random() \* 10;  
        let b = Math.random() \* 10;  
        let c = Math.random() \* 10;  
        let d = Math.random() \* 10;  
        let e = Math.random() \* 10;  
        let f = Math.random() \* 10;  
        let code = `${Math.round(a)} ${Math.round(b)} ${Math.round(c)} ${Math.round(d)} ${Math.round(e)} ${Math.round(f)}`;  
        return code;  
    }  
}  
  
3. Go to component  
  
    login.component.ts  
  
import { Component, OnInit } from "@angular/core";  
import { CaptchaService } from "../../services/captcha.service";  
  
@Component({  
    selector: "app-login",  
    templateUrl: "./login.component.html",  
    styleUrls: ["./login.component.css"]  
})  
export class LoginComponent implements OnInit  
{  
    constructor(private captcha:CaptchaService){  
    }  
    public code:string = '';  
  
    ngOnInit(){  
        this.code = this.captcha.GenerateCode();  
    }  
  
    public NewCode():void{  
        this.code = this.captcha.GenerateCode();  
    }  
    public title:string = "User Login";  
}  
  
4. login.component.html  
  
<div class="container-fluid d-flex justify-content-center align-items-center" style="height:400px">  
  <form class="border border-2 rounded rounded-2 p-3 border-primary">  
    <h3> <span class="bi bi-person-fill"></span> {{title}}</h3>  
    <dl>  
        <dt>User Name</dt>  
        <dd><input type="text" class="form-control"></dd>  
        <dt>Password</dt>  
        <dd><input type="password" class="form-control"></dd>  
        <dt>Verify Code <button (click)="NewCode()" class="btn"> <span class="bi bi-arrow-clockwise"></span> </button> </dt>  
        <dd>{{code}}</dd>  
    </dl>  
    <button class="btn btn-primary w-100">Login</button>  
  </form>  
</div>

02/12/2022

FAQ's  
Pipes  
1. What is a Pipe? What is the role of Pipe?  
A. It is a class, used to transform data.  
  
2. Why to transform data?  
A. To display is user friendly format. And to handle operations like sorting and filtering the data.  
  
3. What is the base for creating a custom Pipe?  
A. PipeTransform  
  
4. Which method returns a format?  
A. transform()  
  
5. What is a parameterized pipe?  
A. It is a pipe that allows parameters to modify its functionality.  
  
                {{ price | currency: 'INR' }}  
  
6. What is Chaining of Pipes?  
A. It is the process of configuring multiple pipes to any specific content.  
  
                {{ message | uppercase | slice:0:6 }}  
  
7. What is a Pure or Impure Pipe?  
A. Pure Pipe changes the value on change detection.  
        percent, slice, i18nselect, i18nplural  
    Impure Pipe will not change value on change detection.  
        uppercase, number, currency, etc..  
  
8. What are built-in pipes and custom pipes?  
  
9. Where to register a Pipe?  
A. Declarations[ ]  
  
10. What  is directive for Pipe?  
A.  @Pipe()  
  
11. What is "async" pipe?  
A. It handle async requests.  
  
Services  
  
1. What is a service?  
  
2. What a service enables?  
A. Reusability, Maintainability, Testablility.  
  
3. What is a factory?  
A. It is a set of functions and values.  
  
4. Why not a factory to use directly in application?  
A. It uses "single call"  
  
5. What is single call?  
A. An object is required every time in order to access any method.  
  
6. Which design pattern service follows?  
A. Single ton  
  
7. What is a Provider?  
A. It is a component of DI that locates a service in memory.  
  
8. What are the Provider Locations?  
A. root, child, platform, any  
  
9. What is difference between configuring a provider in service and configuring in app.module.ts?  
A. app.module uses Eager Loading.  
    Inside Service it is Lazy Loading.  
  
10. What is an Injector?  
A.  It is a component of DI that is responsible for injecting a service into  
     any component of another service.  
  
  
11. Who manages the Injector in service?  
A.  It is managed by the directive "@Injectable()"  
  
12. Why to Inject a service into constructor and not to create an object?  
A. To handle Single Ton.  
  
13. Name any built-in service of Angular?  
A.      HttpClient, Routing, Form Validation etc..  
  
14. Where the services are registered if not configured with provider?  
A.   In app.module.ts  
        providers : [ ]  
  
  
  
  
  
  
  
                                Angular Forms  
- Form is a container that provides an UI from where user can interact with our application.  
- It is used to handle "CRUD Operations"  
            C     - Create  
            R     - Read  
            U   - Update  
            D     - Delete  
- Form is a set of elements like buttons, textbox, checkbox, radio etc..  
- Form is an Array or Elements[]  
- Angular can handle forms without any DOM techiques.  
  
Note: "Angular" is completely de-coupled from DOM.  
  
- Angular uses built-in library and framework to handle forms.  
- Angular Forms are categories into 2 types  
  
        1. Template Driven From  
        2. Model Driven or Reative Forms  
  
- Driven refers to an approach followed by developer to handle any component.  
  
  
Template Driven  
- It is an approach followed by developers where the complete configuration of form and its elements are defiend in UI level.  
- In template driven approach form is configured by using "ngForm"  
  
            <form  #frmRegister="ngForm">  
  
- The elements are configured by using  "ngModel"  
  
    <input type="text"  name="UserName"  ngModel  #UserName="ngModel">  
  
- The template drive forms library is configured by "FormsModule" defined in "@angular/forms".  
  
Features:  
- It reduces the number of requests.  
- It improves the page load time.  
- It is good for inline documentation.  
  
Issues:  
- It is hard to extend.  
- It is hard to test.  
- Separation issues  
- Heavy on UI.  
- Rendering will be slow.  
  
Configuring a Form and Validations

03/12/2022

Configuring a Template Driven Form  
  
1. Go to "app.module.ts" and import the library  
  
    import  { FormsModule } from "@angular/forms";  
  
    imports : [  
                FormsModule  
            ]  
  
2. Form is configured by using "NgForm"  
  
    <form  #refName="ngForm"> </form>  
  
  
3. Elements are configured by using "NgModel"  
  
     <input type="text" ngModel  #Name="ngModel">  
     <select  ngModel   #Name="ngModel">  
  
4. NgForm provides properties that used to handle form and its elements.  
  
            value    [Is an object that collects all form data]  
            invalid  
            valid  
            pristine  
            dirty  
            submitted  
  
Ex:  
template-form.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-template-form',  
  templateUrl: './template-form.component.html',  
  styleUrls: ['./template-form.component.css']  
})  
export class TemplateFormComponent implements OnInit {  
  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  public SubmitClick(formdata:any){  
     alert(JSON.stringify(formdata));  
  }  
  
}  
  
template-form.component.html  
  
<div class="container-fluid">  
    <form #frmRegister="ngForm" (submit)="SubmitClick(frmRegister.value)">  
        <h2>Register User</h2>  
        <dl>  
            <dt>User Name</dt>  
            <dd><input name="UserName" ngModel #UserName="ngModel" type="text"></dd>  
            <dt>Mobile</dt>  
            <dd><input name="Mobile" ngModel #Mobile="ngModel" type="text"></dd>  
            <dt>City</dt>  
            <dd>  
                <select name="City" ngModel #City="ngModel">  
                    <option>Delhi</option>  
                    <option>Hyd</option>  
                </select>  
            </dd>  
        </dl>  
        <button>Register</button>  
    </form>  
    <h3>Data Preview</h3>  
    <pre>  
        {{frmRegister.value | json}}  
    </pre>  
    <h2>User Details</h2>  
    <dl>  
        <dt>User Name</dt>  
        <dd>{{frmRegister.value.UserName}}</dd>  
        <dt>Mobile</dt>  
        <dd>{{Mobile.value}}</dd>  
    </dl>  
</div>  
  
                                Validating Template Form  
- Validation is the process of verify user input.  
- Validation is required to ensure that contradictionary and unauthorized data is not get stored into database.  
- In Web application you can validate  
        a) Server Side  
        b) Client Side  
- Server Side validation is done by using server side scripting  
    JSP, PHP, ASP, Node JS etc..  
- Client Side Validation is done by using  
    JavaScript, jQuery, React, Angular etc..  
- Client Side validation will reduce the burden on server.  
- Client Side validation can be by-passed by browser.  
- Client Side validations in JavaScript and jQuery are defined by using lot of functions.  
                JavaScript  
                Verify a Number        : isNaN()  
                Verify string size        : string.length  
  
- Angular provides Validation services  
- Angular validation services can use HTML validation attributes and verify your values.  
                 HTML 5 Validations  
                 -required  
                 -minlength  
                 -maxlength  
                 -pattern  
                 -number  
                 -email  
                 -url  
  
- Angular Validation services are categorized to 2 groups  
  
        a) Form State Validation Service  
        b) Input State Validation Service  
  
Form State Services:  
- They are used to verify all fields in the form simultaneously at the same time.  
- The service are  
  
    Service Name                Property                Description  
    ------------------------------------------------------------------------------  
    NgPristine                    pristine        It is untouched or not modified [true]  
     NgDirty                        dirty            Any one field modified [true]      
    NgValid                        valid            All fields are valid.  
    NgInvalid                    invalid        Any one field is invalid.  
     NgSubmitted                submitted    Form is submitted  
  
  
Ex:  
template-form.component.html  
  
<div class="container-fluid">  
  <div class="row">  
    <div class="col-3">  
        <form #frmRegister="ngForm" (submit)="SubmitClick(frmRegister.value)">  
            <h2>Register User</h2>  
            <dl>  
                <dt>User Name</dt>  
                <dd><input name="UserName" required ngModel #UserName="ngModel" type="text"></dd>  
                <dt>Mobile</dt>  
                <dd><input name="Mobile" required  pattern="\+91\d{10}" ngModel #Mobile="ngModel" type="text"></dd>  
                <dt>City</dt>  
                <dd>  
                    <select name="City" ngModel #City="ngModel">  
                        <option>Delhi</option>  
                        <option>Hyd</option>  
                    </select>  
                </dd>  
            </dl>  
            <button [disabled]="frmRegister.invalid">Register</button>  
            <button [disabled]="frmRegister.pristine">Save</button>  
        </form>  
    </div>  
    <div class="col-9">  
         <h3>Form State Services</h3>  
         <dl>  
            <dt>Pristine [Form is Untouched]</dt>  
            <dd>{{frmRegister.pristine}}</dd>  
            <dt>Dirty [Form Modified]</dt>  
            <dd>{{frmRegister.dirty}}</dd>  
            <dt>Valid [All Fields are valid]</dt>  
            <dd>{{frmRegister.valid}}</dd>  
            <dt>Invalid [Any one Field is not Valid]</dt>  
            <dd>{{frmRegister.invalid}}</dd>  
            <dt>Form Submitted</dt>  
            <dd>{{frmRegister.submitted}}</dd>  
         </dl>  
    </div>  
  </div>  
</div>  
  
template-form.component.css  
  
.invalid-style {  
    background-color: rgb(255, 198, 198);  
}  
.valid-style {  
    background-color: rgb(221, 255, 221);  
}  
form {  
    padding: 20px;  
}  
  
  
<form [ngClass]="{'valid-style':frmRegister.valid, 'invalid-style':frmRegister.invalid}" #frmRegister="ngForm" (submit)="SubmitClick(frmRegister.value)">  
  
</form>

05/12/2022

Angular Services  
- Provider  
- Injector  
  
                            Angular Http Service  
- Http service it provides a set of properties and methods that are use to communicate with API.  
            JavaScript        fetch()  
            jQuery            $.ajax()  
            Angular            HttpClient  
            React                Axios  
  
- HttpClient is an Angular Service used to communicate with API  
            - Returns data in JSON  
            - Handles CORS  
            - Good Error Handling  
            - Manage Cross Site Script Attacks  
            - Prevent Request Forgery  
            - Can use Async methods with a library "RxJS"  
  
- HttpClient class is a member of "HttpClientModule" defined in  
    "@angular/common/http"  
  
- HttpClient is a service, you have to inject into any component or into any another service.  
  
  
- RxJS is "Reactive Extention JavaScript", It is async library for JavaScript apps.  
  
- RxJS provides  
        a) Observer  
        b) Subscriber  
  
- Observer defines the actions to perform. Methods are configured as type "Observer".  
  
                public MethodName():Observer<T>{  
  
                }  
  
- Async methods can't execute directly.  
- You have to execute aync methods using "Subscriber"  
  
                MethodName()                        // sync  
                MethodName().subscribe()        // async  
  
                subscribe(function(data){  
  
                 })  
  
Ex:  
1. Import Module  
  
import { HttpClientModule } from '@angular/common/http';  
  
2. Add service  
  
    > ng g s  apidata --skip-tests  
  
apidata.service.ts  
  
import { Injectable } from '@angular/core';  
import { HttpClient } from '@angular/common/http';  
import { Observable, retry } from 'rxjs';  
  
@Injectable({  
  providedIn: 'root'  
})  
export class ApidataService {  
  
  constructor(private http:HttpClient) { }  
  GetCategories():Observable<string[]>{  
    return this.http.get<string[]>('[http://fakestoreapi.com/products/categories&#39](http://fakestoreapi.com/products/categories&" \l "39" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTc2Mzg2MzkzNDgw/_blank););  
  }  
  GetProducts():Observable<any[]>{  
    return this.http.get<any[]>('[http://fakestoreapi.com/products&#39](http://fakestoreapi.com/products&" \l "39" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTc2Mzg2MzkzNDgw/_blank););  
  }  
}  
  
  
4. Add component  
  
.component.ts  
  
import { Component, OnInit } from '@angular/core';  
import { ApidataService } from '../../services/apidata.service';  
  
@Component({  
  selector: 'app-datademo',  
  templateUrl: './datademo.component.html',  
  styleUrls: ['./datademo.component.css']  
})  
export class DatademoComponent implements OnInit {  
  
  public Categories:string[] = [];  
  public Products:any[] = [];  
  
  constructor(private apidata: ApidataService) { }  
  
  ngOnInit(): void {  
    this.apidata.GetCategories().subscribe(data=> this.Categories=data );  
    this.apidata.GetProducts().subscribe(data=> this.Products=data);  
  }  
  
}  
  
  
component.html  
  
<div class="container-fluid">  
  <h2>Categories</h2>  
  <ol>  
    <li \*ngFor="let item of Categories">  
        {{item}}  
    </li>  
  </ol>  
  <h2>Products</h2>  
  <ol>  
    <li \*ngFor="let item of Products">  
        {{item.title}}  
    </li>  
  </ol>  
</div>

Template Driven Form  
Validating Template Form  
    a) Form State Validation  
    b) Input State Validation  
  
                                    Input State Validation  
- It is the process of verifying every field in form individually.  
- The validation services are  
  
Service                Property                Description  
------------------------------------------------------------------------------------------  
NgPristing            pristine                Not modified  
NgDirty                dirty                    Modified  
NgInvalid            invalid                Not Valid [it returns true if any one of the  
                                                              validation fails]  
NgValid                valid                    Valid [it returns true if all validations pass]  
NgErrors                errors                    It is used to verify specific validation.  
NgTouched            touched                It returns true when element gets focus.  
NgUnTouched        untouched            It returns true if elements is not in focus.  
  
  
- Errors object is used to verify every individual valiation property.  
- Upto Angular 13 "Errors" is an object with validation properties as "Keys".  
  
            errors.required  
            errors.minlength etc..  
  
- Angular 14 introduces new Errors log, which is an array of errors.  
  
             errors?.['required', 'minlength', 'pattern']  
  
- Angular can identify HTML validation attributes  
        required  
        minlength  
        maxlength  
        pattern  
        email  
        url etc..  
  
- HTML validation attributes are not for all types of elements.  
- You have to create custom validations.  
- Angular provides Validation CSS Classes which are used to set validation styles for element, without using "ngClass"  
  
        .ng-valid  
        .ng-invalid  
        .ng-pristine  
        .ng-dirty  
        .ng-touched  
        .ng-untouched  
  
Syntax:  
        input.ng-invalid {  
        }  
        input.ng-valid {  
        }  
  
Ex: Template Form Validation  
  
template-form.component.ts  
  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-template-form',  
  templateUrl: './template-form.component.html',  
  styleUrls: ['./template-form.component.css']  
})  
export class TemplateFormComponent implements OnInit {  
  
  constructor() { }  
  
  public CityError:string = "";  
  public EvenError:string = "";  
  
  ngOnInit(): void {  
  }  
  public SubmitClick(formdata:any){  
     alert(JSON.stringify(formdata));  
  }  
  public CityChange(cityName:string){  
    if(cityName=="notcity") {  
       this.CityError = "Please Select a City";  
    } else {  
       this.CityError = "";  
    }  
  }  
  
  public VerifyEven(n:any) {  
      if(parseInt(n) % 2 ==0){  
        this.EvenError = "";  
      } else {  
        this.EvenError = "Not an Even Number";  
      }  
  }  
  
}  
  
  
template-form.component.css  
  
#terms~button {  
    display: none;  
}  
#terms:checked~button {  
    display: block;  
}  
#terms+span {  
    color:red;  
}  
#terms:checked+span {  
    color:green;  
}  
input.ng-invalid {  
    border:1px solid red;  
    box-shadow: 2px 2px 2px red;  
}  
input.ng-valid {  
    border:1px solid green;  
    box-shadow: 2px 2px 2px green;  
}  
form.ng-invalid {  
    background-color: lightpink;  
}  
form.ng-valid {  
    background-color: lightgreen;  
}  
form {  
    padding: 20px;  
}  
  
template-form.component.html  
  
<div class="container-fluid">  
     
    <form class="w-25" #frmRegister="ngForm">  
        <h2>Register User</h2>  
        <dl>  
            <dt>User Name</dt>  
            <dd><input type="text" required   minlength="4" name="UserName" ngModel #UserName="ngModel" class="form-control"></dd>  
            <dd class="text-danger" \*ngIf="(frmRegister.submitted && UserName.invalid)||(UserName.touched && UserName.invalid)">  
                <div \*ngIf="UserName.errors?.['required']">Name Required</div>  
                <div \*ngIf="UserName.errors?.['minlength']">Name too short min 3 chars</div>  
            </dd>  
            <dt>Mobile</dt>  
            <dd><input type="text" required  pattern="\+91\d{10}" name="Mobile" ngModel #Mobile="ngModel" class="form-control"></dd>  
            <dd class="text-danger" \*ngIf="(frmRegister.submitted && Mobile.invalid)|| (Mobile.touched && Mobile.invalid) ">  
                <div \*ngIf="Mobile.errors?.['required']">Mobile Required</div>  
                <div \*ngIf="Mobile.errors?.['pattern']">Invalid Mobile</div>  
            </dd>  
            <dt>Password</dt>  
            <dd><input type="password" name="Password" ngModel #Password="ngModel" required pattern="(?=.\*[A-Z])\w{4,15}" class="form-control"></dd>  
            <dd class="text-danger" \*ngIf="(frmRegister.submitted && Password.invalid) || (Password.touched && Password.invalid)">  
                <div \*ngIf="Password.errors?.['required']">Password Required</div>  
                <div \*ngIf="Password.errors?.['pattern']">At least 1 upper case letter required</div>  
            </dd>  
            <dt>Verify Even</dt>  
            <dd>  
                <input type="text" (blur)="VerifyEven(Even.value)" name="Even" ngModel #Even="ngModel" class="form-control" placeholder="Enter Even Number">  
            </dd>  
            <dd class="text-danger">  
                {{EvenError}}  
            </dd>  
            <dt>City</dt>  
            <dd>  
                <select name="City" (change)="CityChange(City.value)" ngModel #City="ngModel" class="form-select">  
                    <option value="notcity">Select Your City</option>  
                    <option value="Delhi">Delhi</option>  
                    <option value="Hyd">Hyd</option>  
                </select>  
            </dd>  
            <dd class="text-danger">  
                {{CityError}}  
            </dd>  
            <dt>Terms of Service</dt>  
            <dd>  
                <textarea rows="4" cols="30" disabled>Our Terms of Service</textarea>  
                <div>  
                    <input type="checkbox" id="terms" class="form-check-input">  <span class="form-check-label">I Accept</span>  
                    <button class="btn btn-primary">Register</button>  
                </div>  
            </dd>  
        </dl>  
         
    </form>  
</div>  
  
                                         
                            Model Driven / Reactive Form

06/12/2022

Template Forms and Validation  
  
                            Model Driven / Reactive Forms  
                            -------------------------------------------  
- Clean separation of code and UI.  
- Easy to Extend  
- Easy to Test  
- Less Burden on UI  
- Renders Fast  
- Library required for model driven forms "ReactiveFormsModule" in "@angular/forms".  
  
Step-1: Import ReactiveFormsModule in app.module.ts  
  
import { FormsModule, ReactiveFormsModule } from '@angular/forms';  
  
imports : [  
        ReactiveFormsModule  
  ]  
  
Step-2: Entire form is configured in controller.  
  
           FormGroup            : It configures <form> element  
           FormControl            : It configures <input> <select> <textarea> etc..  
           FormBuilder            : It is a service  
  
Syntax:  
          public  frmRegister = new FormGroup({  
                FieldName : new FormControl('value', Validators[]),  
                FieldName : new FormControl('value', Validators[])  
          })  
  
Step-3: Bind the Model Form with UI  
  
        <form [FormGroup]="frmRegister"]>  
          <input type="text" formControlName="FieldName">  
  
Ex: Simple Form  
  
reactive-demo.component.ts  
  
import { Component, OnInit } from '@angular/core';  
import { FormGroup, FormControl } from '@angular/forms';  
  
@Component({  
  selector: 'app-reactive-demo',  
  templateUrl: './reactive-demo.component.html',  
  styleUrls: ['./reactive-demo.component.css']  
})  
export class ReactiveDemoComponent implements OnInit {  
  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  
  public frmRegister = new FormGroup({  
     Name: new FormControl(''),  
     Price: new FormControl(0),  
     City:  new FormControl(''),  
     Stock: new FormControl(false)  
  })  
  
  public RegisterClick(formData:any){  
   alert(JSON.stringify(formData));  
  }  
  
}  
  
  
reactive-demo.component.html  
  
<div class="container-fluid">  
     
    <form [formGroup]="frmRegister" (submit)="RegisterClick(frmRegister.value)">  
        <h2>Register Product</h2>  
        <dl>  
            <dt>Name</dt>  
            <dd><input type="text" formControlName="Name"></dd>  
            <dt>Price</dt>  
            <dd><input type="text" formControlName="Price"></dd>  
            <dt>City</dt>  
            <dd>  
                <select formControlName="City">  
                    <option>Delhi</option>  
                    <option>Hyd</option>  
                </select>  
            </dd>  
            <dt>Stock</dt>  
            <dd>  
                <input formControlName="Stock" type="checkbox"> Available  
            </dd>  
        </dl>  
        <button class="btn btn-primary">Register</button>  
    </form>  
</div>  
  
                                 Nested Forms - Async  
  
- You can configure a FormGroup inside another FormGroup.  
  
Syntax:  
         frmRegister = new FormGroup({  
  
                Field : new FormControl('', []),  
  
                ChildForm: new FormGroup({  
                     Field : new FormControl('', [])  
                })  
  
        })  
  
- You have to bind parent FormGroup to <form> element.  
- You have to bind child FormGorup to any HTML container. <div> <aside>  
  
Syntax:  
            <form [formGroup]="frmRegister">  
  
                <div formGroupName="ChildForm">  
  
                </div>  
     
            </form>  
  
Ex:  
reactive-demo.component.ts  
  
import { Component, OnInit } from '@angular/core';  
import { FormGroup, FormControl } from '@angular/forms';  
  
@Component({  
  selector: 'app-reactive-demo',  
  templateUrl: './reactive-demo.component.html',  
  styleUrls: ['./reactive-demo.component.css']  
})  
export class ReactiveDemoComponent implements OnInit {  
  
  constructor() { }  
  
  ngOnInit(): void {  
  }  
  
  public frmRegister = new FormGroup({  
     Name: new FormControl(''),  
     Price: new FormControl(0),  
     City:  new FormControl(''),  
     Stock: new FormControl(false),  
     frmVendor : new FormGroup({  
        VendorName: new FormControl(''),  
        Rating: new FormControl(0)  
     })  
  })  
  
  get VendorName() {  
    return this.frmRegister.controls.frmVendor.controls.VendorName as FormControl;  
  }  
  
  get Rating(){  
    return this.frmRegister.controls.frmVendor.controls.Rating as FormControl;  
  }  
  
  public RegisterClick(formData:any){  
   alert(JSON.stringify(formData));  
  }  
  
  public UpdateClick(){  
     this.frmRegister.patchValue({  
         Price: 45000.44,  
         frmVendor: {  
           Rating: 4.5  
         }  
     })  
  }  
  
}  
  
  
reactive-demo.component.html  
  
<div class="container-fluid">  
     
    <form [formGroup]="frmRegister" (submit)="RegisterClick(frmRegister.value)">  
        <h2>Register Product</h2>  
        <dl>  
            <dt>Name</dt>  
            <dd><input type="text" formControlName="Name"></dd>  
            <dt>Price</dt>  
            <dd><input type="text" formControlName="Price"></dd>  
            <dt>City</dt>  
            <dd>  
                <select formControlName="City">  
                    <option>Delhi</option>  
                    <option>Hyd</option>  
                </select>  
            </dd>  
            <dt>Stock</dt>  
            <dd>  
                <input formControlName="Stock" type="checkbox"> Available  
            </dd>  
            <div formGroupName="frmVendor">  
                <h3>Vendor Form</h3>  
                <dl>  
                    <dt>Vendor Name</dt>  
                    <dd>  
                        <select formControlName="VendorName">  
                            <option>Reliance Digital</option>  
                            <option>Bajaj Electronics</option>  
                        </select>  
                    </dd>  
                    <dt>Rating</dt>  
                    <dd><input formControlName="Rating" type="range" min="1" value="1" max="5"></dd>  
                </dl>  
            </div>  
        </dl>  
        <button class="btn btn-primary">Register</button>  
        <button type="button" (click)="UpdateClick()" class="btn btn-info ms-2">Update</button>  
    </form>  
</div>  
  
                                Form Builder Service  
- It is a service provided by "ReactiveFormsModule".  
- It uses single ton pattern for configure form and its elements.  
  
        private fb : FormBuilder  
  
- Methods provides  
  
            fb.group()            <form>  
            fb.control()            <input> <select> <textarea> etc..  
            fb.array()                collection of controls.  
  
                                    Array & Validation  
  
  
  
                                    Routing - SPA

07/12/2022

FormBuilder Service  
- It is a service for configuring form and its elements.  
- It uses a single ton pattern for form and elements.  
- It comprises of various methods  
        a) group()  
        b) control()  
        c) array()  
  
- group() configures <form>  
- control() configures <input> <select> <textarea>  
- array() configures a set of controls.  
  
Syntax:  
       constructor(private fb:FormBuilder) { }  
  
       frmRegister = fb.group({  
             Field : fb.control(' '),  
             ChildForm: fb.group({  
                Field: fb.control(' ')  
             })  
        })  
  
Ex:  
   reactive-demo.component.ts  
  
 public frmRegister = this.fb.group({  
     Name: this.fb.control(''),  
     Price: this.fb.control(0),  
     City: this.fb.control(''),  
     Stock: this.fb.control(false),  
     frmVendor: this.fb.group({  
         VendorName: this.fb.control(''),  
         Rating: this.fb.control(0)  
     })  
  })  
  
Ex:  
reactive-demo.component.ts  
  
import { Component, OnInit } from '@angular/core';  
import { FormGroup, FormControl, FormBuilder, FormArray } from '@angular/forms';  
  
@Component({  
  selector: 'app-reactive-demo',  
  templateUrl: './reactive-demo.component.html',  
  styleUrls: ['./reactive-demo.component.css']  
})  
export class ReactiveDemoComponent implements OnInit {  
  
  constructor(private fb: FormBuilder) { }  
  
  ngOnInit(): void {  
  }  
  
  public frmRegister = this.fb.group({  
     Name: this.fb.control(''),  
     Price: this.fb.control(0),  
     City: this.fb.control(''),  
     Stock: this.fb.control(false),  
     frmVendor: this.fb.group({  
         VendorName: this.fb.control(''),  
         Rating: this.fb.control(0)  
     }),  
     Photos : this.fb.array([this.fb.control('')])  
  })  
  
  get VendorName() {  
    return this.frmRegister.controls.frmVendor.controls.VendorName as FormControl;  
  }  
  
  get Rating(){  
    return this.frmRegister.controls.frmVendor.controls.Rating as FormControl;  
  }  
  
  get Photos(){  
    return this.frmRegister.controls.Photos as FormArray;  
  }  
  
  public RegisterClick(formData:any){  
   alert(JSON.stringify(formData));  
  }  
  
  public UpdateClick(){  
     this.frmRegister.patchValue({  
         Price: 45000.44,  
         frmVendor: {  
           Rating: 4.5  
         }  
     })  
  }  
  public AddControls(){  
     this.Photos.push(this.fb.control(''));  
  }  
  public RemoveControl(index:number){  
     this.Photos.removeAt(index);  
  }  
  
}  
  
reactive-demo.component.html  
  
<div class="container-fluid">  
     
    <form [formGroup]="frmRegister" (submit)="RegisterClick(frmRegister.value)">  
       <div class="row">  
          <div class="col-4">  
            <h2>Register Product</h2>  
            <dl>  
                <dt>Name</dt>  
                <dd><input type="text" formControlName="Name"></dd>  
                <dt>Price</dt>  
                <dd><input type="text" formControlName="Price"></dd>  
                <dt>City</dt>  
                <dd>  
                    <select formControlName="City">  
                        <option>Delhi</option>  
                        <option>Hyd</option>  
                    </select>  
                </dd>  
                <dt>Stock</dt>  
                <dd>  
                    <input formControlName="Stock" type="checkbox"> Available  
                </dd>  
                <div formGroupName="frmVendor">  
                    <h3>Vendor Form</h3>  
                    <dl>  
                        <dt>Vendor Name</dt>  
                        <dd>  
                            <select formControlName="VendorName">  
                                <option>Reliance Digital</option>  
                                <option>Bajaj Electronics</option>  
                            </select>  
                        </dd>  
                        <dt>Rating</dt>  
                        <dd><input formControlName="Rating" type="range" min="1" value="1" max="5"></dd>  
                    </dl>  
                </div>  
            </dl>  
            <button class="btn btn-primary">Register</button>  
            <button type="button" (click)="UpdateClick()" class="btn btn-info ms-2">Update</button>  
          </div>  
          <div class="col-8">  
              <dl class="mt-4">  
                <dt>Upload Photo <button type="button" (click)="AddControls()" class="btn btn-link">more..</button> </dt>  
                <dd>  
                   <div \*ngFor="let item of Photos.controls; let i = index" class="mb-2">  
                      <input type="file" formControlName="i"> <button>Upload</button> <button (click)="RemoveControl(i)" class="btn"><span class="bi bi-trash"></span></button>  
                   </div>  
                </dd>  
              </dl>  
          </div>  
       </div>  
    </form>  
</div>  
  
                                Validating Reactive Forms  
- ReactiveFormsModule provides "Validators[]".  
- Validators is a collection of validation properties like  
        required  
        minlength  
        pattern  
        maxlegth  
- No need to configure validation properties in UI for elements.  
  
Syntax:  
        fb.control(' ', [Validators.required, Validators.pattern])  
        FormControl(' ', [Validators.required, Validators.pattern])  
  
Ex:  
reactive-validation.component.ts  
  
import { Component, OnInit } from '@angular/core';  
import { FormBuilder, FormControl, Validators } from '@angular/forms';  
  
@Component({  
  selector: 'app-reactive-validation',  
  templateUrl: './reactive-validation.component.html',  
  styleUrls: ['./reactive-validation.component.css']  
})  
export class ReactiveValidationComponent implements OnInit {  
  
  constructor(private fb:FormBuilder) { }  
  
  ngOnInit(): void {  
  }  
  
  public frmRegister = this.fb.group({  
      UserName : this.fb.control('',[Validators.required, Validators.minLength(4)]),  
      Mobile: this.fb.control('', [Validators.required, Validators.pattern(/\+91\d{10}/)])  
  })  
  get UserName(){  
    return this.frmRegister.controls.UserName as FormControl;  
  }  
  get Mobile(){  
    return this.frmRegister.controls.Mobile as FormControl;  
  }  
  
}  
  
reactive-validation.component.html  
  
<div class="container-fluid">  
    <form [formGroup]="frmRegister">  
        <h2>Register User</h2>  
        <dl>  
            <dt>User Name</dt>  
            <dd><input type="text" formControlName="UserName"></dd>  
            <dd \*ngIf="UserName.touched && UserName.invalid" class="text-danger">  
              <div \*ngIf="UserName.errors?.['required']">User Name Required</div>  
              <div \*ngIf="UserName.errors?.['minlength']">Name too Short...</div>  
            </dd>  
            <dt>Mobile</dt>  
            <dd><input type="text" formControlName="Mobile"></dd>  
            <dd \*ngIf="Mobile.touched && Mobile.invalid" class="text-danger">  
                <div \*ngIf="Mobile.errors?.['required']">Mobile Required</div>  
                <div \*ngIf="Mobile.errors?.['pattern']">Invalid Mobile</div>  
              </dd>  
        </dl>  
        <button>Register</button>  
    </form>  
</div>  
  
  
Note: If you are using Angular validation services, then disable HTML form validations. By using "novalidate" attribute for <form>.  
  
                <form [formGroup]="frmRegister"  novalidate>  
  
                </form>  
  
FAQ  
1. Can we have multiple submit buttons in a form?  
A. Yes  
  
2. Why you need multiple submit buttons?  
A. They allow to handle various actions.  
  
3. All submit buttons use same "submit()" event, then how they perform different funcitonality?  
A.

09/12/2022

Angular Routing  
- Routing is a technique used in web applications to create and configure User Friend and SEO friendly url.  
  
 [http://www.amazon.in/electronics.php?category=mobiles&model=samsung](http://www.amazon.in/electronics.php?category=mobiles&model=samsung" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTc3Nzc4NTMyMzY3/_blank)  
  
 [http://www.amazon.in/electronics/mobiles/samsung](http://www.amazon.in/electronics/mobiles/samsung" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTc3Nzc4NTMyMzY3/_blank)  
  
- User friendly URL allows the user to query any content directly from URL.  
- SEO friendly URL allows crawlers to exactly know your location.  
  
- Routing uses implicity AJAX calls, New details are added to page without reloading the complete page.  
  
- It is the primary feature required to build SPA [Single Page Application]  
  
- User can stay on one page and get access to everything on to page.  
  
- Routing is a technique used both  
        a) Client Side'  
  
        b) Server Side  
  
- Angular is using routing client side.  
  
- Angular provides a built-in library for configuring routes.  
  
            "@angular/router"  
  
- Routing for application is defined by using a "RouterModule"  
  
- Modules are created by using "@NgModule()"  directive.  
  
Syntax: app-routing.module.ts  
  
        import    { NgModule }  from  "@angular/core"  
  
        @NgModule( { })  
  
        export class AppRoutingModule {  }  
  
- "Routes" class is used to configure a virtual Route Table  
  
        const routes:Routes = [  
             { path: " ",  component: " "}  
        ]  
  
- "RouterModule" class is used to import and export routes for application, so that application can start using routes.  
  
- Route Links are configure by using <a> element with angular attribute  
    "routerLink"  
  
    <a  routerLink="routePathName"> Text | Image </a>  
  
- Route Result is rendered using   
   <router-outlet>

10/12/2022

 Routes  
- RouterModule  
- Import  
- Export  
- RouterLink  
- RouterOutlet  
  
                                Route Parameters  
- A parameter can transport data from one component to another.  
- Parameter is configure in Angular route  
  
          { path : "details/:id/:name/:price", component : DetailsComponent }  
  
- Parameter values are passed in the URL or through router link.  
  
          [http://localhost:4200/details/1/tv/34000.44](http://localhost:4200/details/1/tv/34000.44" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTA4NTkwNTY3MDYw/_blank)  
  
            routeParams = {  
                   id : 1,      
                   name: "tv",  
                   price : 34000.44  
            }  
  
          <a routerLink="details/1/tv/45000"> </a>  
  
- You have to access the parameters and use in any component by using Angular "ActivatedRoute" service.  
  
           constructor(private route:ActivatedRoute) { }  
  
           this.route.snapShot.paramMap.get("id");            => 1  
  
Ex:  
1. Add a new component  
    > ng g c shopper-details --skip-tests  
  
2. app-routing.module.ts  
  
{path: "details/:id/:name/:price", component: ShopperDetailsComponent},  
  
3. shopper-details.component.ts  
  
import { Component, OnInit } from '@angular/core';  
import { ActivatedRoute } from '@angular/router';  
  
@Component({  
  selector: 'app-shopper-details',  
  templateUrl: './shopper-details.component.html',  
  styleUrls: ['./shopper-details.component.css']  
})  
export class ShopperDetailsComponent implements OnInit {  
  
  constructor(private route:ActivatedRoute) { }  
  
  public Product:any = {};  
  ngOnInit(): void {  
    this.Product = {  
       Id: this.route.snapshot.paramMap.get("id"),  
       Name: this.route.snapshot.paramMap.get("name"),  
       Price: this.route.snapshot.paramMap.get("price")  
    }  
  }  
  
}  
  
4. shopper-details.component.html  
  
<h2>Product Details</h2>  
<dl>  
    <dt>Product Id</dt>  
    <dd>{{Product.Id}}</dd>  
    <dt>Price</dt>  
    <dd>{{Product.Price}}</dd>  
    <dt>Name</dt>  
    <dd>{{Product.Name}}</dd>  
</dl>

17/12/2022

Angular Material  
- CDK  
- Material Components  
  
1. Form-Field  
    It is just like a container with set of elements. [Field Set]  
     
 - visit  "material.angular.io"  
 - go to components  
 - select any component  
 - go to "API" tab for Module & dependency  
 - If you want to explore the properties then switch to "API"  
 - If you want to know the property values then track the property type class.  
  
        @Input() appearance: MatFormFieldAppearance  
  
        MatFormFieldAppearance = "fill, outline"  
  
Ex:  Form Field and Input  
  
1. app.module.ts  
  
import {MatFormFieldModule} from '@angular/material/form-field';  
import { MatInputModule } from '@angular/material/input';  
  
imports: [  
    MatFormFieldModule,  
    MatInputModule  
]  
  
2. material-demo.component.html  
  
<div class="container-fluid">  
    <h2>Bootstrap Input</h2>  
     <div class="w-25">  
        <label class="form-label">User Name</label>  
        <div>  
            <input type="text" placeholder="User Name" class="form-control">  
        </div>  
     </div>  
    <h2>Material Input</h2>  
    <mat-form-field  appearance="outline" >  
        <mat-label>User Name</mat-label>  
        <input name="UserName" ngModel #UserName="ngModel" type="text" placeholder="Enter User Name" matInput>  
    </mat-form-field>  
    <h3>Hello ! {{UserName.value}}</h3>  
</div>  
  
Ex: Calendar  
  
import {MatDatepickerModule} from '@angular/material/datepicker';  
import { MatNativeDateModule } from '@angular/material/core';  
  
imports: [  
MatDatepickerModule,  
 MatNativeDateModule  
]  
  
  
material-demo.component.html  
  
<mat-form-field>  
        <mat-label>Departure</mat-label>  
        <input name="Departure" [matDatepicker]="picker" ngModel #Departure="ngModel" type="text" matInput>  
        <mat-datepicker-toggle matSuffix [for]="picker"></mat-datepicker-toggle>  
        <mat-datepicker #picker></mat-datepicker>  
</mat-form-field>  
  
  
                                    Angular Testing  
- Testing is to process of verifying  
        "AS-IS and TO-BE"  
  
- AS-IS refers to what developer have designed.  
- TO-BE refers to what client wants.  
  
        AS-IS === ToBE        Test Pass  
        AS-IS !==  TOBE        Test Fail  
  
- Angular Inbuilt Provides "Jasmine-Karma" framework  
  
- Tests are defined in a "spec file" [.spec.ts]  
  
- Jasmine provides various test methods  
         
        a) describe()  
        b) it()  
        c) expect()  
        d) assert related methods..  
  
- Testing every method includes 3 phases  
  
        a) Arrange        describe()  
        b) Act                it()  
        c) Assert            expect()  
  
  
Ex:  
> ng g c cart  
  
.spec.ts  
  
import { ComponentFixture, TestBed } from '@angular/core/testing';  
  
import { CartComponent } from './cart.component';  
  
describe('CartComponent', () => {  
  let component: CartComponent;  
  let fixture: ComponentFixture<CartComponent>;  
  
  beforeEach(async () => {  
    await TestBed.configureTestingModule({  
      declarations: [ CartComponent ]  
    })  
    .compileComponents();  
  
    fixture = TestBed.createComponent(CartComponent);  
    component = fixture.componentInstance;  
    fixture.detectChanges();  
  });  
  
  it('should create', () => {  
    expect(component).toBeTruthy();  
  });  
  
  //Cart Title Test  
  
  describe("Cart Title Test",()=>{  
    it("Title Must Be Amazon Cart",()=>{  
       expect(component.CartTitle("Amazon Shopping Cart")).toBe("Amazon Shopping Cart");  
    })  
  })  
  
  //Cart Total Test  
  describe("Cart Total Test",()=>{  
    it("Total Must be 10000",()=>{  
      expect(component.Total(2,5000)).toBe(10000);  
    })  
  })  
  
});  
  
  
> ng  test  --project=shopping

18/12/2022

Build and Deploy  
- Building is the process of Checking  
        Syntax Errors  
        Code Snippets  
        Dependencies  
        Providers  
        Injectors  
- You can build for development or production.  
- When you build for development it will just compile and identify the issues.  
- When you build for production it will generate "dist" folder and bundle all resources which are ready for "Go-Live".  
  
            >ng build                            [ development ]  
            >ng build  --production        [ production ]  
  
- From Angular 14 "ng-build" is default production.  
  
- All components logic is bundled and resources are bundled and copied into "dist" folder.  
  
- It will generate "index.html" as startup for SPA.  
  
FAQ: Can we change default bundle folder from "dist" to any another?  
Ans: Yes.  
                > ng build  --outputDirectory=folderName  
  
FAQ: Can we change the startup page?  
Ans: Yes.  
                > ng build  --baseHref=login.html  
  
                > ng build --help  
  
- Deploying is the process of uploading and publishing all production code into server.  
        a) Local Server  
        b) Cloud Server  
  
- To Deploy of cloud servers angular provides various libraries  
- You have to use the libraries to handle deployment.  
  
Note: Don't use any manual deployment process.  
  
  
DEPLOYMENT TO                PACKAGE  
--------------------------------------------------------------------------------------  
Firebase hosting                    @angular/fire  
Vercel                                vercel init angular  
Netlify                                    @netlify-builder/deploy  
GitHub pages                        angular-cli-ghpages  
NPM                                    ngx-deploy-npm  
Amazon Cloud S3                    @jefiozie/ngx-aws-deploy  
  
>ng add  @angular/fire  
  
                             Firebase Deployment  
- Firebase is Google Cloud  
- Login into Firebase using Google Account  
  
        [https://firebase.google.com/](https://firebase.google.com/" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTc5Nzg1NTM5NTA0/_blank)  
  
- Go to Console  
  
- Create a new Project  
  
     Name :    angular-smart-shopping  
  
     Select :Google Analytics for your Firebase project  
  
     Account: Default [Free]  
  
     Create Project  
  
- Install Firebase Tools on your PC globally - [CLI] - It provides various commands that are used to publish your project into cloud.  
  
    C:\>npm install -g firebase-tools  
  
- Connect your project with cloud [firebase]  
  
    a) Add Angular Firebase library to your project  
  
        workspace > ng  add  @angular/fire --project=amazon  
  
                       >  What features to add...?  
             
                           ng deploy  --hosting  
  
                      >Which Firebase account would you like to use?  
         
                            your@account  
  
                    ? Please select a project: angular-smart-shopping  
                        ? Please select a hosting site:  
                            [https://angular-smart-shopping.web.app](https://angular-smart-shopping.web.app/" \t "https://classroom.google.com/c/NTY3NTY2MjU2MTA5/m/NTc5Nzg1NTM5NTA0/_blank)  
  
    b) Build your project for production  
  
                      > ng build --project=amazon  
  
    c) ng deploy  --project=amazon  
  
Note: After every change your have to build and deploy.  
  
        > ng build --project=amazon  
        > ng deploy --project=amazon  
  
  
                            Deploying on GIT  
  
- Create a new account on GIT  
- Create a new Repository  
- Download "GIT Hub Desktop" tool  
- Login to GIT hub Desktop  
- Clone your repository  
  
  
View material

18/12/2022

Angular State  
  
- Every web application uses Http protocol.  
- Http is a "state less" protocol.  
- State Less uses the mechanism  
  
    GO - GET - FORGET  
  
- GO            : Establish connection with server.  
- GET            : Get response from server.  
- FORGET    : Disconnect from server and clean up.  
  
- State less nature of Http is an advantage for server. As It can manage memory.  
  
- State less nature is not good for page or component as it can't remember information between pages or components.  
  
- Web Applications use various state management techniques, which are classified into  
            a) Server Side State Management  
            b) Client Side State Management  
  
- Client Side State Management allows to store data locally on client device.  
            a) Query String  
            b) Cookies  
             
- Query String  
    \* Can't handle complex data like binary  
    \* Limit for data 2048 chars  
    \* Not safe, Easy to hack  
    \* Stored in browser logs  
    \* Can be bookmarked.  
  
- Cookies  
    \* CORS  
    \* Can be disabled  
    \* Persistent [Can be deleted]  
  
- 3rd Party Library for JavaScript based  
  
    Redux => React  
     NgRx  => Angular  
  
- It can mantain local state for your application, where you can store data and make it available across components.  
  
- NgRx creates a store for application.  
  
- Store is used as a service. You can inject into any component.  
  
- You can store data centrally for application.  
  
Terms:  
1. Actions:  Defines actions used to update the store.  
  
2. Reducer:  Defines the definition for actions.  
  
3. Store : It stores the data.  
  
4. Selector: It inject the store into any component  
  
5. State : It defines the initial and final state of data.  
  
6. Observable: It belongs to RxJs [Actions Async]  
  
Ex:  
1. Install NgRx  
  
    > ng add  @ngrx/store@latest  --project=hdfcbank  
  
2. Create Actions  
  
      createAction('actionName')  
  
3. Create Reducer  
     
  
      createReducer(initialState, on((state)=> state + 1)  
  
4.  Go to app.module and configure the store  
  
    StoreModule.forRoot({count: couterReducer}, {})  
  
5.  Inject the store into any component  
  
     constructor(private likes:Store)  
  
      c =  this.likes.select('count');  
  
      <h2> {{ c | async }} </h2>  
  
        <button (click)="increment()">  
  
  
  
Ex:  
src/app/counter.actions.ts  
---------------------------------------------  
import { createAction } from '@ngrx/store';  
  
export const increment = createAction('[Counter Component] Increment');  
export const decrement = createAction('[Counter Component] Decrement');  
export const reset = createAction('[Counter Component] Reset');  
  
src/app/counter.reducer.ts  
--------------------------------------  
import { createReducer, on } from '@ngrx/store';  
import { increment, decrement, reset } from './counter.actions';  
  
export const initialState = 0;  
  
export const counterReducer = createReducer(  
  initialState,  
  on(increment, (state) => state + 1),  
  on(decrement, (state) => state - 1),  
  on(reset, (state) => 0)  
);  
  
  
src/app/app.module.ts (StoreModule)  
--------------------------------------------------------  
import { BrowserModule } from '@angular/platform-browser';  
import { NgModule } from '@angular/core';  
   
import { AppComponent } from './app.component';  
   
import { StoreModule } from '@ngrx/store';  
import { counterReducer } from './counter.reducer';  
   
@NgModule({  
  declarations: [AppComponent],  
  imports: [BrowserModule, StoreModule.forRoot({ count: counterReducer })],  
  providers: [],  
  bootstrap: [AppComponent],  
})  
export class AppModule {}  
  
  
  
  
src/app/my-counter/my-counter.component.ts  
----------------------------------------------------------------------  
import { Component } from '@angular/core';  
import { Store } from '@ngrx/store';  
import { Observable } from 'rxjs';  
import { increment, decrement, reset } from '../counter.actions';  
   
@Component({  
  selector: 'app-my-counter',  
  templateUrl: './my-counter.component.html',  
})  
export class MyCounterComponent {  
  count$: Observable<number>;  
   
  constructor(private store: Store<{ count: number }>) {  
    this.count$ = store.select('count');  
  }  
   
  increment() {  
    this.store.dispatch(increment());  
  }  
   
  decrement() {  
    this.store.dispatch(decrement());  
  }  
   
  reset() {  
    this.store.dispatch(reset());  
  }  
}  
  
  
src/app/my-counter/my-counter.component.html  
------------------------------------------------------------------  
  
<button (click)="increment()">Increment</button>  
  
<div>Current Count: {{ count$ | async }}</div>  
  
<button (click)="decrement()">Decrement</button>  
  
<button (click)="reset()">Reset Counter</button>  
  
Actions [Declaration]  
     createAction()  
    increment, decrement, reset  
Reducer [Definition]  
     initialState = 0;  
    createReducer(initialState, on(increment, (state)=> state + 1),  
                                        on(decrement, (state)=> state - 1),  
                                        on(reset), (state) => 0)  
     )  
Store [app.module.ts]  
    forRoot({ count: createReducer })  
  
Select [Component]  
  
    private store: Store  
  
    const likes: Observable<number>;           async  
  
    onIncrement() {  
          this.store.dispatch(increment())  
    }  
  
  
Async Values are executed by using "async" pipe  
  
        const count : Observerable<number>;  
  
         {{ count | async }}  
  
        GetProducts(): Observable<IProduct[]>  
         {  
         }  
  
        GetProducts().subscribe()