Angular

- **Angular.**
- JavaScript Framework for web development.
- Single page application (no refresh on clicks).
- Developed by Google.
- **Angular 19 Setup in Windows OS.**
- Install Node.js.
- Npm install -g @angular/cli.
- Ng new app.
- Choose CSS.
- NO
- Cd my-app
- Ng serve.
- **Angular Interpolation Example.**
- **App.component.html.**

```
<h1>{{title}}</h1>
<h2>{{name}}</h2>
<h2>{{age}}</h2>
<h2>{{Email}}</h2>
<h2>{{address}}</h2>
```

- **App.component.ts.**

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

@Component({
  selector: 'app-root',
  imports: [],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'Angular';
  name = 'Shanelhai';
  age = 22;
  Email = 'shanelhai7@gmail.com';
  address = 'Haldwani';
}
```

- **Components in Angular.**
- A component is a building block of your Angular application. It controls a part of the screen, and each section, page, or UI element is usually managed by its own component.
- **Command of create components.**
- ng generate component login.
- **How to use.**
- **App.component.html**
  <app-login></app-login>
- **App.component.ts.**
  ```
  import { Component} from '@angular/core';
  import { LoginComponent } from './login/login.component';
  @Component({
   selector: 'app-root',
   imports: [LoginComponent],
   templateUrl: './app.component.html',
   styleUrl: './app.component.css'
  })
  export class AppComponent {
  }
  ```
- **Decorator is a @.**
- **Custom Component in Angular.**
- Custom component is a building block of Angular that you create yourself to display the user interface and handle related logic.
- **Profile.component.ts**
  ```
  import { Component } from '@angular/core';

  @Component({
   selector: 'app-profile',
   templateUrl: './profile.component.html',
  })
  export class ProfileComponent {
   // Component logic here
  }
  ```

- **Profile.component.html.**
  <h1>Hello Shaan I am a custom component</h1>
- **App.component.html.**
  <app-login></app-login>
  <app-profile></app-profile>
- **App.component.ts.**
  import { Component} from '@angular/core';
  import { LoginComponent } from './login/login.component';
  import { ProfileComponent } from './profile/profile.component';

  @Component({
   selector: 'app-root',
   imports: [LoginComponent, ProfileComponent],
   templateUrl: './app.component.html',
   styleUrl: './app.component.css'
  })
  export class AppComponent {
  }
- **Recap Module and Routing in Angular.**
- create new module.
- ng generate module users
- ng generate component users/login

- **Function Call on Button Click in Angular.**
- Calls a function when user clicks a button in Angular.
- **App.component.html.**

```
<h1>Function Call on Button Click.</h1>
<button (click)="funname()">Click Me</button>
```

- **App.component.ts.**

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

@Component({
 selector: 'app-root',
 imports: [],
 templateUrl: './app.component.html',
 styleUrl: './app.component.css'
})
export class AppComponent {
      funname(){
        alert("function called Hello World");
        // when call the function one to another function using this
keyword.
        this.otherfun()
      }
      otherfun(){
        alert("function called Hii other function");
      }
}
```

- **Data Types in Angular.**
- **string** – Text values
- Example: "Hello", "Angular"
- **number** – Numeric values (whole or decimal)
- Example: 25, 99.5
- **boolean** – True or false
- Example: true, false
- **array** – List of items of the same type
- Example: ["red", "blue"], [1, 2, 3]
- **object** – Group of values using keys
- Example: { name: "John", age: 30 }
- **any** – Can be any type (not recommended often)
- Example: "Hello", 10, true (all are valid)
- **null** – No value (empty on purpose)
- Example: null
- **undefined** – Variable declared but not assigned any value
- Example: let x: string | undefined;
- **tuple** – Fixed-length array with different types
- Example: ["Alice", 25]
- **enum** – A set of named constant values
- Example:
- **void** – No value returned (used in functions)
- Example:
- **typescript**
  Copy code
  function log(): void {
  console.log("Logging");
  }

- **app.component.html**

```
<h1>Welcome, {{ name }}</h1>
<p>Age: {{ age }}</p>
<p>Login status: {{ isLoggedIn }}</p>
<p>Your hobbies:</p>
<ul>
 <li *ngFor="let hobby of hobbies">{{ hobby }}</li>
</ul>
<p>Email: {{ user.email }}</p>
```

- **app.component.ts.**

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

@Component({
 selector: 'app-root',
 imports: [],
 templateUrl: './app.component.html',
 styleUrl: './app.component.css'
})
export class AppComponent {
      // Different data types used in a component

 name: string = 'Alice';        // String
 age: number = 25;              // Number
 isLoggedIn: boolean = true;    // Boolean
 hobbies: string[] = ['Reading', 'Music'];  // Array of strings
 user: { id: number; email: string } = {    // Object
  id: 1,
  email: 'alice@example.com'
 };
 anything: any = 'Can be anything';         // Any type

}
```

- **Make Counter in Angular.**
- **App.componet.html.**

```
<h1>Counter</h1>
<div class="text-align: center;">
   <h2 class="count">{{count}}</h2>
</div>
<div>
   <button class="btn btn-red"
(click)="handleCount('plus')">Increment</button>
   <button class="btn btn-black"
(click)="handleCount('')">Reset</button>
   <button class="btn btn-green"
(click)="handleCount('mins')">Decrement</button>
</div>
```

- **App.component.ts.**

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

@Component({
 selector: 'app-root',
 imports: [],
 templateUrl: './app.component.html',
 styleUrl: './app.component.css'
})
export class AppComponent {
  count = 0;
 handleCount(val:string){
   if(val == 'plus') {
     this.count = this.count + 1
   } else
   if(val == 'mins') {
     this.count = this.count - 1
   }
   else{
     this.count = this.count = 0
   }         }        }
```

- **Event in Angular.**
- An event in Angular is something that happens, like a button click or a key press, and lets you run some code when it happens.
- **App.component.html.**

```
<h1>Event in Angular</h1>
<button  name="login" (click)="handleEvent($event)">Click
me</button>
<input type="text"  (input)="handleEvent($event)"
(focus)="handleEvent($event)" name="user">
<div style="background-color: red; width: 100px; height: 100PX;"
(mouseenter)="handleEvent($event)" ></div>
<div style="background-color: blue; width: 100px; height: 100PX;"
(mouseleave)="handleEvent($event)" ></div>
```

- **App.componet.ts.**

```
handleEvent(event:Event){
   // event use access details.
   console.log('Event handled',event.type);
   // console.log('Event handled',(event.target as
Element).className);
   console.log("value",(event.target as HTMLInputElement).value);
 }
```

- **Get and Set Input filed Value.**
- **Get**: Reading the value entered by the user in an input field.
- **Set**: Changing or assigning a value to the input field using code.
- **App.component.html.**

```
<h1>Get and Set Input Filed Value</h1>
<h1>User name : {{displayName}}</h1>
<input type="text" value="{{name}}" (input)="getName($event)"
placeholder="Enter the value">
<br>
<button (click)="showName()">Get Name</button>
<button (click)="setName()">set Name</button>
<hr>
<h1>Email: {{ email }}</h1>
<input type="text" placeholder="Enter the email" #emailField />
<button (click)="getEmail(emailField.value)">Get Email</button>
```

```
<button (click)="setEmail()">Set Email</button>
```

- **App.component.ts**

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

@Component({
 selector: 'app-root',
 imports: [],
 templateUrl: './app.component.html',
 styleUrl: './app.component.css'
})
export class AppComponent {
 name="";
 displayName = "";
 email = ""
 getName(event:Event){
  const name = (event.target as HTMLInputElement).value;
  //console.log(name);
  this.name = name;
 }
 showName(){
  this.displayName = this.name;
 }
 setName(){
  this.name = "John Doe";
 }
 getEmail(value: string) {
  this.email = value;
 }
 setEmail() {
  this.email = "john.doe@example.com";
}
}
```

- **Style Option.**
- **Inline styles – Defined in the component:**
- Ts Copy code
- styles: [`p { color: red; }`]
- **External styles – Linked CSS/SCSS files:**
- Ts Copy code
- styleUrls: ['./component.css']
- **Global styles –** In styles.css or styles.scss (set in angular.json).
- **View Encapsulation –** Controls style scope:
- **Emulated (default) –** Scoped to component.
- **None – Global**
- ShadowDom – Native encapsulation
- **Dynamic styles –** With ngStyle or ngClass:
- Html Copy code
- <div [ngStyle]="{color: 'blue'}">
- <div [ngClass]="{'active': isActive}">
- Preprocessors – Use SCSS, LESS, etc., by configuring project.

- **IF Else | Control Flow.**
- If-else lets a program choose between actions based on conditions.
- **App.component.html.**

```
<h1>If Else | Control Flow</h1>
<button (click)="handleColor(1)">Red</button>
<button (click)="handleColor(2)">Blue</button>
<button (click)="handleColor(3)">Green</button>
<button (click)="handleColor(4)">Black</button>
<input type="text" (input)="handleInput($event)" placeholder="Enter
the number">
@if(color == 1){
  <div style="width: 200px; height: 200px; background-color: red;">
  </div>
}
@else if(color == 2){
  <div style="width: 200px; height: 200px; background-color: blue;">
  </div>
}
@else if(color == 3){
  <div style="width: 200px; height: 200px; background-color:green;">
  </div>
}
@else{
  <div style="width: 200px; height: 200px; background-color:black;">
  </div>
}
```

- **App.component.ts.**

```
export class AppComponent {
  color = 2
  handleColor(val:number){
    this.color=val;
  }
  handleInput(event:Event){
    this.color = parseInt((event.target as HTMLInputElement).value)
  }
}
```

- **Switch Case in Angular.**
- Switch case is a way to choose one action from many options based on a value.
- **App.component.html.**

```
<h1>Switch Case</h1>
<button (click)="handleColor('red')">Red</button>
<button (click)="handleColor('blue')">Blue</button>
<button (click)="handleColor('green')">Green</button>
<button (click)="handleColor('black')">Black</button>
<input type="text" (input)="handleInput($event)" placeholder="Enter the color">
@switch(color){
  @case('red'){
  <div style="width: 200px; height: 200px; background-color: red;">
  </div>          }
  @case('blue'){
  <div style="width: 200px; height: 200px; background-color: blue;">
  </div>          }
  @case('green'){
  <div style="width: 200px; height: 200px; background-color:green;">
  </div>          }
  @case('black'){
    <div style="width: 200px; height: 200px; background-color:black;">
  </div>          }
  @default{
  <div style="width: 200px; height: 200px; background-color:white;">
  </div>          }                }
```

- **App.component.ts.**

```
export class AppComponent {
  color = 'blue';
  handleColor(val:string){
   this.color=val;                }
  handleInput(event:Event){
   this.color = (event.target as HTMLInputElement).value
  }         }
```

- **Loops in Angular.**
- A loop is a block of code that repeats a certain number of times or while a condition is true.
- **App.component.html.**

```
@for(student of students; track student; let id=$index){
   <h3>Student {{ id + 1 }}</h3>
   <p><strong>Name:</strong> {{ student.name }}</p>
   <p><strong>Age:</strong> {{ student.age }}</p>
   <p><strong>Email:</strong> {{ student.email }}</p>
   <p><strong>Address:</strong> {{ student.address }}</p>
   <button (click)="getName('student.name')">Get Name</button>
}
```

- **App.component.ts**

```
export class AppComponent {
    students = [
   { name: 'Alice', age: 20, email: 'alice@example.com', address: '123
Main St' },
   { name: 'Bob', age: 22, email: 'bob@example.com', address: '456
Oak Ave' },
   { name: 'Charlie', age: 19, email: 'charlie@example.com', address:
'789 Pine Rd' },
   { name: 'David', age: 21, email: 'david@example.com', address:
'321 Maple St' },
   { name: 'Eva', age: 23, email: 'eva@example.com', address: '654
Elm St' }
 ];
 getName(name:string){
   console.log(name);
 }
}
```

- **Signals in Angular.**
- A Signals is a Wrapper around a value.
- That give a Signals when value change.
- **Type of Signals.**
- **Writable Signals:**
- Signals you can change and update to notify Angular.
- **Computable Signals:**
- Signals that automatically calculate value based on other signals.
- **App.component.html.**

```
<h1>Signals</h1>
<button (click)="updateValue('inc')">Increment</button>
<h1>{{count()}}</h1>
<button (click)="updateValue('dec')">Decrement</button>
```

- **App.component.ts**

```
export class AppComponent {
 count = signal(10);
 updateValue(val:string){
  if(val=='inc'){
   this.count.set(this.count()+1)
  }else{
   this.count.set(this.count()-1)
  }
}
}
```

- **Data Type of Signals.**
- **App.component.html.**

```
<h1>Signals</h1>
<h1>{{data()}}</h1>
```

- **App.component.ts.**

```
<button (click)="updateValue()">Decrement</button>
export class AppComponent {
 // define the single data type.
 // data = signal<number | string>(10);
 //define the signle and value data type.
 data : WritableSignal<number | string>= signal<number |
string>(10);
```

```
  //Computable Signals readonly you can not change:
  // count:signal<number> = computed(()=>10);
  updateValue(){
    this.data.set("Hello");
  }
}
```

- **Computed Signals.**
- Computed signals read-only you can't change force fully.
- **App.component.html.**

```
 <h1>Computed Signals</h1>
<h1>{{z()}}</h1>
<button (click)="showValue()">show value</button>
<button (click)="updateX()">update x</button>
```

- **App.component.ts.**

```
export class AppComponent {
  x = signal(10);
  y = signal(20);
  z = computed(()=>this.x()+this.y());
  showValue(){
    console.log(this.z());
    this.x.set(100);
    console.log(this.z());
  }
  updateX(){
    this.x.set(200);
  }
}
```

- **Effect in Angular.**
- A function that runs automatically when signals it uses change.
- **App.component.html.**

```
<h1>Effect in Angular</h1>
<h1>{{userName()}}</h1>
<h1>{{count()}}</h1>
<button (click)="userName.set('sam')">Update user name</button>
<button (click)="toggleValue()">Update user name</button>
@if(displayHeading){
   <h1 style="background-color: green;">Page Heading</h1>
}
```

- **App.component.ts.**

```
export class AppComponent {
  userName = signal('Shaan');
 count = signal(0);
 displayHeading = false; // initialize properly

 constructor() {
  effect(() => {
   if (this.count() === 2) {
    this.displayHeading = true;
   setTimeout(()=>{
    this.displayHeading = false;
   },2000)
   } else {
    this.displayHeading = false;
   }
  });
 }
 toggleValue() {
  this.count.set(this.count() + 1);
 }
}
```

- **For Loop Contextual Variables.**
- Contextual variables are special variables you can use inside *ngFor loops in Angular.
- **Common Contextual Variables:**
- index Position of the item (starting from 0)
- first true if it's the first item
- last true if it's the last item
- even true if the index is even
- odd true if the index is odd
- **app.component.html**

```
<h1>For Loop Contextual Variables.</h1>
@for (user of user; let i = $index; let isFirst = $first; let isLast = $last;
let isOdd = $odd; let isEven = $even; track user) {
 <h3>User {{ i + 1 }}: {{ user }}</h3>
 <p>Index: {{ i }}</p>
 <p>First: {{ isFirst }}</p>
 <p>Last: {{ isLast }}</p>
 <p>Odd: {{ isOdd }}</p>
 <p>Even: {{ isEven }}</p>
}
```

- **app.component.ts.**

```
export class AppComponent {
 user = ['Alice', 'Bob', 'Charlie', 'David', 'Eva'];
}
```

- **To Way Binding in Angular.**
- Two-way binding means data flows both ways.
- **App.component.html.**

```
<h1>Two Way Binding</h1>
<input [(ngModel)]="name" type="text" placeholder="Enter the
name">
<h1>{{name}}</h1>
```

- **App.component.ts.**

```
import { Component, computed, effect, signal, WritableSignal} from
'@angular/core';
import { FormsModule, NgModel  } from '@angular/forms';

@Component({
 selector: 'app-root',
 imports: [FormsModule],
 templateUrl: './app.component.html',
 styleUrl: './app.component.css'
})
export class AppComponent {
 name="Shaan";
 changeName(event:Event){
  const val = (event.target as HTMLInputElement).value;
  this.name=val
 }
}
```

- **Directives in Angular.**
- Directive is a class that adds additional behavior to elements in your application.
- A features in angular that help you to provide more power to Dom elements.
- If-else condition.
- For Loop.
- Add Style.
- ngif, ngfor, ngstyle, ngswitch, ng class.
- **Types of Directives.**
- **Component Directives.**
- The most common type of directive.
- Used in Component template file.
- **Structural Directives.**
- Change the structure of the DOM by adding, removing, or manipulating elements.
- **Attribute Directives.**
- Modify the appearance or behavior of an existing element.
- **Simple Directives.**
- **App.component.html.**
  <h1 *ngIf="show">Heading Tag</h1>
- **App.component.ts.**
- Import: [NgIf]
  export class AppComponent {
    show=true
  }
- ***NgFor Directives in Angular.**
- *ngFor is a directive in Angular used to loop through a list and display each item in HTML.
- **App.component.html.**
  <h1>Heading Tag</h1>
   <li *ngFor="let x of students">{{x}}
   <li *ngFor="let student of studentData">{{student.name}}, {{student.age}}, {{student.email}}, {{student.address}}

- **App.component.ts.**

```
import { NgFor, NgIf } from '@angular/common';
import { Component, computed, effect, signal, WritableSignal} from
'@angular/core';
import { FormsModule, NgModel  } from '@angular/forms';

@Component({
 selector: 'app-root',
 imports: [FormsModule, NgIf, NgFor],
 templateUrl: './app.component.html',
 styleUrl: './app.component.css'
})
export class AppComponent {
  students = ['Alice', 'Bob', 'Charlie'];
  studentData = [
 { name: 'Alice', age: 20, email: 'alice@example.com', address: '123
Apple St' },
 { name: 'Bob', age: 22, email: 'bob@example.com', address: '456
Banana Ave' },
 { name: 'Charlie', age: 21, email: 'charlie@example.com', address:
'789 Mango Blvd' }
];
 }
```

- ***ngif Directive in Angular.**
- The `*ngIf` directive conditionally includes or removes elements in templates.
- **App.component.html.**

```
<h1>*ngif Directive in Angular</h1>
<!-- two conditions -->
<div *ngIf="login; else elseBlock">
   <a href="">Logout</a>
</div>
<ng-template #elseBlock>
<a href="/">login</a>
</ng-template>
<!-- multip conditons -->
<div *ngIf="block==0">
```

```
  <h1>Heading 0</h1>
</div>
<div *ngIf="block==1">
  <h1>Heading 1</h1>
</div>
<div *ngIf="block==2">
  <h1>Heading 2</h1>
</div>
<button (click)="updateBlock()">Update Block</button>
```

- **App.component.ts.**

```
export class AppComponent {
  login = false
  block = 0
  updateBlock(){
   this.block++;          } }
```

- ***ngSwitch in Angular.**
- **App.component.html.**

```
<div [ngSwitch]="color">
 <p *ngSwitchCase="'red'" style="background-color: red;">Red color
selected</p>
 <p *ngSwitchCase="'blue'" style="background-color: blue;" >Blue
color selected</p>
 <p *ngSwitchCase="'green'" style="background-color:green;">Green
color selected</p>
 <p *ngSwitchDefault>Unknown color</p>
</div>
```

- App.component.ts.

```
import { NgFor, NgIf, NgSwitch, NgSwitchCase, NgSwitchDefault } from
'@angular/common';
import { Component, computed, effect, signal, WritableSignal} from
'@angular/core';
import { FormsModule, NgModel  } from '@angular/forms';
@Component({
 imports: [FormsModule, NgSwitch, NgSwitchCase,NgSwitchDefault],
})
export class AppComponent {
 color = "green";  }
```

- **Basic Routing in Angular.**
- ng generate component home.
- **Step 1 : app.routes.ts.**

```
import { Routes } from '@angular/router';
import { AboutComponent } from './about/about.component';
import { HomeComponent } from './home/home.component';

export const routes: Routes = [
 { path: '', redirectTo: 'home', pathMatch: 'full' },
 {path:'home',component:HomeComponent},
 {path:'about',component:AboutComponent},
 {path:'**',component:PageNotFoundComponent}

];
```

- **Step 2 : app.component.ts.**

```
import { Component, computed, effect, signal, WritableSignal} from
'@angular/core';
import { FormsModule, NgModel  } from '@angular/forms';
import { RouterOutlet, RouterLink } from '@angular/router';

@Component({
 selector: 'app-root',
 imports: [RouterOutlet, RouterLink],
})
export class AppComponent {
}
```

- **Step 3 : app.component.html.**

```
<router-outlet></router-outlet>
```

- **Header with Routing.**
- Step 1 : Header create a Navar.
- <a routerLink="/home" >Home
- Step 2: Header.component.ts add this imports: [RouterLink].
- Step 3 : app.component.html add this <app-header></app-header>
- **4O4 Page.**
- ng generate component page-not-found

- **Pass Data From one Page to Other.**
- Pass Data with Router Link.
- Pass Data with Button Click.
- Pass Data with Router.
- **Home.component.html.**

```
<h1>Hello home page</h1>
<a routerLink="/profile">Go to the profile page</a>
<!-- Data pass  -->
<br>
<a [routerLink]="['/profile' ,{name:'shan'}]">Go to the profile
page</a>
<br>
<button (click)="goToProfile('Shaan')" >Go to the profile
page</button>
<button (click)="goToProfile('Shanelhai')" >Go to the profile
page</button>
```

- **Home.component.ts.**

```
export class HomeComponent {
 constructor(private router:Router){
 }
 goToProfile(name:string){
  this.router.navigate(['profile'],{queryParams:{name:name}})
 }  }
```

- **Profile.component.ts.**

```
export class ProfileComponent {
 userName:string|null= "";
 constructor(private router:ActivatedRoute){}
 ngOnInit(){
  this.userName = this.router.snapshot.paramMap.get('name');
  console.log(this.userName);
  this.router.queryParams.subscribe(params=>{
   this.userName=params['name'];
  })
 }
}
```

- **Build Dynamic Routing in Angular.**
- Dynamic Routing in Angular means navigating to different pages (components) based on changing values, like an ID, name, or slug in the URL.
- **Home.component.html.**

```
<h1>Hello home page</h1>
@for(user of user; track user.id)
{
   <li><a
routerLink="/user/{{user.id}}/{{user.name}}">{{user.name}}</a></li>
}
```

- **User.component.ts.**

```
export class UserComponent {
 name:null|string="";
  constructor(private route:ActivatedRoute){
  }
  ngOnInit(){
   this.route.params.subscribe(params =>{
    console.log(params)
    this.name=params['name'];
   })
  }
}
```

- **User.component.html.**

```
<h1>User name is:{{name}}</h1>
```

- **App.routes.ts.**

```
export const routes: Routes = [
 {path: '', redirectTo: 'home', pathMatch: 'full' },
  {path:'user/:id/:name',component:UserComponent},
];
```

- **Forms in Angular.**
- Forms are used to collect user input.
- **Types of Forms in Angular.**
- **Reactive Form:**
- Use for complex logic, validations, and dynamic form controls.
- Example: Login form with real-time validation and conditionally shown fields.
- **Template-Driven Form:**
- Use for simple forms directly in HTML templates.
- Example: Contact form with name, email, and message fields.
- **Basic Reactive Forms.**
- **Reactive:**
- **App.component.html.**

```
<h1>Basic of React Forms</h1>
<form action="" method="post">
   <input type="text" [formControl]="name" placeholder="Enter your
name">
   <br>
   <br>
   <input type="password" [formControl]="password"
placeholder="Enter your password">
   <br>
   <br>
   <button type="button" (click)="displayValue()">Login</button>
    <button type="button" (click)="setValue()">Set Values</button>
   <h1>{{name.value}}</h1>
   <h1>{{password.value}}</h1>
</form>
```

- **App.component.ts.**

```
@Component({
 selector: 'app-home',
 imports: [ReactiveFormsModule],
})
export class HomeComponent {
 name = new FormControl();
 password = new FormControl();
 displayValue(){
  console.log(this.name.value);
 }
 setValue(){
  this.name.setValue('John Doe');
  this.password.setValue('ddd');
 }
}
```

- **Form Grouping in Reactive Forms.**
- Form Grouping in Reactive Forms means combining controls to manage related form fields together easily.
- **App.component.html.**

```
<h1>Basic of React Forms</h1>
<form [formGroup]="profileForm" (ngSubmit)="onSubmit()"
method="post">
  <input type="text" formControlName="name" placeholder="Enter
your name">
  <br>
  <br>
  <input type="text" formControlName="email" placeholder="Enter
your email">
  <br>
  <br>
  <input type="password"  formControlName="password"
placeholder="Enter your password">
  <br>
  <br>
  <button>submit</button>
   <button type="button" (click)="setValue()">set value</button>
```

```
</form>
<h1>{{this.profileForm.value.name}}</h1>
<h1>{{this.profileForm.value.email}}</h1>
<h1>{{this.profileForm.value.password}}</h1>
```

- **App.component.ts.**

```
@Component({
 selector: 'app-home',
 imports: [ReactiveFormsModule],
})
export class HomeComponent {
 profileForm= new FormGroup({
  name:new FormControl(),
  email:new FormControl('a@gmail.com'),
  password: new FormControl()
 });
 onSubmit(){
  console.log(this.profileForm.value);
 }
 setValue(){
  this.profileForm.setValue({
   name: 'John',
   email: 'john@gmail.com',
   password: '123456'
  })
}
}
```

- **Reactive Forms Validation.**
- **define validation.**
- Required.
- Min Character.
- Email Validation.
- Make getter method
- Define Validation message.
- Define Error Message.
- Add Style.
- **App.component.html.**

```
<h1>Basic of React Forms</h1>
<form [formGroup]="profileForm" (ngSubmit)="onSubmit()"
method="post">
  <input type="text" formControlName="name" placeholder="Enter
your name">
  <span *ngIf="name?.hasError('required')">Name is
Required</span>
  <span *ngIf="name?.invalid &&(name?.dirty ||
name?.touched)">Provide the user name</span>
  <br>
  <br>
  <input type="text" formControlName="email" placeholder="Enter
your email">
  <span *ngIf="email?.hasError('required')">Email is
Required</span>
  <span *ngIf="email?.invalid &&(email?.dirty ||
email?.touched)">Provide the valid email</span>
  <br>
  <br>
  <input type="password"  formControlName="password"
placeholder="Enter your password">
  <span *ngIf="password?.hasError('required')">password is
Required</span>
  <span *ngIf="password?.invalid &&(password?.dirty ||
password?.touched)">Provide the valid password</span>
  <br>
```

```
    <br>
    <button>submit</button>
     <button type="button">set value</button>
</form>
<!-- <h1>{{this.profileForm.value.name}}</h1>
<h1>{{this.profileForm.value.email}}</h1>
<h1>{{this.profileForm.value.password}}</h1> -->
```

- **App.component.ts.**

```
export class HomeComponent {
  profileForm= new FormGroup({
    name:new FormControl('',[Validators.required]),
    email:new FormControl('',[Validators.required,
Validators.maxLength(50), Validators.pattern('^[a-z0-9._%+-]+@[a-z0-
9.-]+\.[a-z]{2,4}$')]),
    password: new FormControl('',[Validators.required,
Validators.minLength(5)])
  });
  onSubmit(){
    console.log(this.profileForm.value);
  }
  get name(){
    return this.profileForm.get('name');
  }
   get email(){
    return this.profileForm.get('email');
  }
   get password(){
    return this.profileForm.get('password');
  }
}
```

- **Template Driven Forms.**
- Define Validations.
- Required.
- Min length and max length.
- Email.
- Dropdown.
- display message on validation.
- display button if any of input field is invalid.
- **App.component.html.**

```
<h1>Basic of Angular Forms</h1>
<form #userForm="ngForm"
(ngSubmit)="addDetails(userForm.value)">
 <input type="text" #name="ngModel" required name="name"
ngModel placeholder="Enter your name" ngModel/>
 <span *ngIf="name.invalid && name.touched">Name is
Required</span>
 <br><br>
 <input type="text" #email="ngModel"  required name="email"
ngModel placeholder="Enter your email" ngModel/>
 <span *ngIf="email.invalid && email.touched">email is
Required</span>
 <br><br>
 <input type="password" #password="ngModel" required
name="password" ngModel placeholder="Enter your password"
ngModel />
  <span *ngIf="password.invalid && password.touched">Password is
Required</span>
 <br><br>

 <label>
  Gender:
  <select name="gender" ngModel>
   <option value="">Select Gender</option>
   <option value="male">Male</option>
   <option value="female">Female</option>
  </select>
```

```
  </label>
  <br><br>
  <input type="range" name="range" min="0" max="20" ngModel />
  <br><br>
  <button type="submit"
[disabled]="userForm.invalid">Submit</button>
</form>
<h1>{{userDetails?.name}}</h1>
<h1>{{userDetails?.email}}</h1>
<h1>{{userDetails?.password}}</h1>
<h1>{{userDetails?.gender}}</h1>
```

- **App.component.ts.**

```
import { NgIf } from '@angular/common';
import { Component } from '@angular/core';
import { FormsModule, NgForm } from '@angular/forms';

@Component({
  selector: 'app-home',
  imports: [FormsModule, NgIf],
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent {
  userDetails:any;
  addDetails(val:NgForm) {
    console.log(val);
    this.userDetails=val;
  }
}
```

- **Pass Date Parent to Child Component.**
- **App.component.html.**
- **that is parent component.**
- <router-outlet></router-outlet>
- <app-child [user]="userName" city="haldwani" ></app-child>
- <select #selectUser (change)="onUserChange(selectUser.value)"
  name="" id="">
- <option value="A">AAAA</option>
- <option value="B">BBBB</option>
- <option value="C">BBBB</option>
- <option value="D">DDDD</option>
- </select>
- **App.component.ts.**
  export class AppComponent {
  // Dynmic Data.
  userName = "Shanelhai";
  onUserChange(user:string){
  this.userName = user
  }
  }
- **Child.component.html.**
  <h1>Child Component {{user}}</h1>
  <h1>Child Component {{city}}</h1>
- **child.component.ts.**
  import {Input} from '@angular/core';
  export class ChildComponent {
  @Input() user:string="";
  @Input() city:string="";
  }

- **Reuse Component.**
- **App.component.html.**

```
<h1>Reuse Component</h1>
@for(user of users; track users)
{
   <div>
      <app-child [user]="user"></app-child>
   </div>
}
```

- **App.component.ts.**

```
@Component({
 selector: 'app-root',
 imports: [RouterOutlet, HeaderComponent, ChildComponent],
})
```

-
```
export class AppComponent {
  users=['A', 'B', 'C', 'D', 'E', 'F'];
}
```

- **child.component.html.**

```
<h1>User name is : {{user}}</h1>
```

- **child.component.ts.**

```
export class ChildComponent {
 @Input() user:string="";
}
```

- **Pass data Child to Parent Component.**
- **Child.component.html.**

```
import { Component, EventEmitter, Output} from '@angular/core';
export class ChildComponent {
 @Output() getUsers = new EventEmitter();
  user = ['A', 'B', 'C', 'D', 'E', 'F'];
  loadData(){
    this.getUsers.emit(this.user);
  }           }
```

- **child.component.html**

```
<h1>Hello Child Component.</h1>
<button (click)="getUsers.emit(this.user)">pass data 1</button>
<button (click)="loadData()">pass data 2</button>
```

- **app.component.ts**

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { HeaderComponent } from "./header/header.component";
import { ChildComponent } from "./child/child.component";
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet, HeaderComponent, ChildComponent],
})
export class AppComponent {
  user:undefined|string[];
  handlerUser(user: string[]) {
    console.log(user);
    this.user=user;
  }           }
```

- **app.component.html.**

```
<app-header></app-header>
<router-outlet></router-outlet>
<app-child (getUsers)="handlerUser($event)"></app-child>
@for(user of user; track user)
{
   <h1>{{user}}</h1>            }
```

- **Pipes in Angular.**
- Pipes in Angular are used to transform data before displaying it in the template.
- **App.component.html.**

```
<h1>Pipes in Angular</h1>
<h1>{{title}}</h1>
<h1>{{title | uppercase}}</h1>
<h1>{{title | titlecase}}</h1>
<h1>{{name | lowercase}}</h1>
<h1>{{date | date:'fullDate'}}</h1>
<h1>{{date | date:'hh:mm:ss'}}</h1>
<h1>{{amount|currency}}</h1>
<h1>{{amount|currency:'EUR'}}</h1>
<h1>{{amount|currency:'INR'}}</h1>
```

- **App.component.ts.**

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { HeaderComponent } from "./header/header.component";
import { ChildComponent } from "./child/child.component";
import { CommonModule } from '@angular/common';
@Component({
 selector: 'app-root',
 standalone: true,
 imports: [RouterOutlet, HeaderComponent, ChildComponent,
CommonModule],
 templateUrl: './app.component.html',
 styleUrl: './app.component.css'
})
export class AppComponent {
 title="step by step";
 name = "SHANELHAI";
 date  = new Date();
 amount = 10;
}
```

- **automation Testing.**
- Automation Testing Documentation for Angular Project.
- **Introduction**
- Automation testing ensures the quality and stability of the Angular application by automatically running tests on components, services, and user flows. This documentation outlines the testing types, tools used, and guidelines for writing and running tests.
- **Testing Types**
- Type Description Tools
- Unit Testing Tests individual units like components and services Jasmine + Karma
- Integration Testing Tests interactions between multiple unitsJasmine + Karma
- End-to-End Testing Tests full user flows in real browsers Cypress or Playwright
- Snapshot Testing Compares component output to saved snapshots Jest
- **3. Tools Used**
- **Jasmine:** Framework for writing unit and integration tests.
- **Karma:** Test runner for running Jasmine tests in browsers.
- **Cypress:** Tool for end-to-end browser testing.
- **Jest:** Optional tool for snapshot testing.
- **Angular CLI:** Provides commands for running and generating tests.
- **Np test.**