Angular	
Date	**Topic**
Day 1 **31-Jan-2025**	1. What is Angular? 2. Angular Versions 3. Prerequisites 4. What is TypeScript? 5. Node 6. CLI 7. Environment setup 8. TypeScript Data types 9. Converting String to Number in TypeScript 10. Annotations 11. Unions 12. Inference 13. Enums
	Assignments
	1. Structure Assignment **[Quizzes/Predict output/Errors/Syntax/Coding Challenge]**
	1. What is Angular, and how is it different from AngularJS? **2.** What is Node? **3.** What is CLI **4.** What is TypeScript, and how is it different from JavaScript? **5.** What are key features of TypeScript? **6.** How do you compile TypeScript code to JavaScript? **7.** What are the different data types supported in TypeScript? **8.** How is **let** different from **var** in JavaScript? **9.** What is the purpose of the **any** type in JavaScript? **10.** What are type annotations? **11.** Explain the type interfaces? **12.** What is type inference in TypeScript? **13.** Explain how you can use union types in TypeScript? **14.** What are generics in TypeScript? **15.** What are Enums in TypeScript and why do we use them?

2. Interactive Exercise [Code debugging/Logic Building/Mock interview]

Exercise: Creating a To-Do List App

You need to create a simple To-Do list app with the following functionality:

1. **Define an Enum for task priority (Low, Medium, High).**

Priority.ts

```
enum Priority {
        Low="Low",
        Medium="Medium",
        High="High",
    export default Priority;
```

2. **Create a Task interface that includes the following properties:**

id (number)

title (string)

completed (boolean)

priority (enum: Low, Medium, High)

Task.ts

```
import Priority from "./Priority";
interface Task {
    id: number,
        title: string,
    completed: boolean;
    priority: Priority;
}
export default Task;
```

3. Functions for Task Management

Create a **task.manager.ts** file

task.manager.ts

```
import Priority from "./Priority";
import Task from "./Task";
class TaskManager{
   private tasks:Task[]=[];
//function to add a task
   addTask(title:string,priority:Priority):void{
      const newTask:Task={
         id:Date.now(),
         title,
         completed:false,
         priority,
      };
    this.tasks.push(newTask);
    console.log(`Task "${title}",added successfully`);
   }
   //function to mark a task as completed
   markTaskAsCompleted(id:number):void{
      const task=this.tasks.find(task=>task.id==id);
      if(task){
         task.completed=true;
         console.log(`Task "${task.title}" marked as completed`);
      }else{
         console.log(`Task with ID ${id} not found.`);
       }
      }
   //function to get tasks by priority
   getTasksByPriority(priority:Priority):Task[]{
      return this.tasks.filter(task=>task.priority=priority);
   }
   //list all tasks
   listTasks():void{
```

```
        this.tasks.forEach(task=>{
            console.log(`${task.id}-${task.title} [${task.priority}] completed:
${task.completed}`);
        })
    }
}
```

export default TaskManager;

4. Test the Task Manager

Create a **main.ts** to test the Task Manager

main.ts

```
import TaskManager from "./task-manager";
import Priority from "./Priority";
const taskManager=new TaskManager();
//Add task
taskManager.addTask("Complete TypeScript Exercise",Priority.High);
taskManager.addTask("Read a Book", Priority.Medium);
taskManager.addTask("Go for a walk",Priority.Low);
//List all tasks
taskManager.listTasks();
//Mark task as completed
taskManager.markTaskAsCompleted(1);
//Get tasks by priority
const highPriorityTask=taskManager.getTasksByPriority(Priority.High);
console.log("High Priority Tasks:",highPriorityTask);
```

Steps to complete the exercise:

- Define the Enum for task priority:
- Enum: Priority with values Low, Medium, High.

Create the Task interface:

- The Task interface should have id, title, completed, and priority.

Function to add a task:

- The addTask function should take a title and priority and add a new task with completed set to false and a unique id.

Function to mark a task as completed:

- The markTaskAsCompleted function should take an id and change the completed property to true.

Function to get tasks by priority:

- The getTasksByPriority function should return all tasks with the specified priority.

1. **Scenario Based Exercise [Project Simulations/Code Reviews/Task Ownership]**

Consider a **Library Management System** to demonstrate various TypeScript concepts.

Scenario: Library Management System
We want to manage books in a library, where we can:

- Add a new book
- Borrow a book
- Return a book
- Check book availability
- Get the list of books by genre or author

Key Features:

1. **Book Class**: Represents each book with properties like title, author, genre, and isAvailable.
2. **Library Class**: Contains an array of books and methods to perform actions such as borrowing, returning, and listing books.
3. **Enums and Types**: To ensure valid genres and statuses.
4. **Generics**: For filtering books by properties like genre or author.

Explanation:

1. **Enum for Genre**: We define a set of possible genres to restrict book types to predefined values (Fiction, Non-Fiction, etc.).
2. **BookClass**: This class implements the Book interface, representing each book in the library. We can borrow and return books using the borrowBook and returnBook methods.
3. **Library Class**: This class manages an array of books and provides methods to add books, list books, and check availability. It also allows filtering books by genre or author using methods like listBooksByGenre and listBooksByAuthor.
4. **Methods**:
    - addBook: Adds a new book to the library.
    - listBooks: Displays all books in the library.
    - listBooksByGenre: Lists books of a specific genre.

- listBooksByAuthor: Lists books by a specific author.
- checkAvailability: Checks whether a book is available for borrowing.

2. Daily coding Exercise

Exercise User Profile Management (Using Classes and Interfaces)

Objective: Create a `User` class with methods to update the user's information.

- **Step 1:** Create a User interface with the following properties:
  - id (number)
  - name (string)
  - email (string)
  - age (optional number)

- **Step 2: Implement a class UserProfile that implements the User interface.**
  - Add a method to update the user's name and email.
  - Add a method to display the user's profile information.

3. Gamification

Creating a gamification system for GT's to improve engagement, productivity, enhance teamwork, Encourage healthy competition.

Gamification Elements

1. Points
   a. Award 10 points for completing the assignments on time
   b. Extra points for presentation
2. Badges
   a. Graduate Trainee Champion for consistent performance
3. Leader boards
   a. Monthly leader board to recognize top performer

Angular	
Date	**Topic**
Day 2 **01-Feb-2025**	1. What is npm? 2. Creating Angular application 3. Folder Structure 4. Running your angular app 5. Adding Bootstrap to your application 6. What are components? 7. Angular Component Life Cycle Hooks 8. Modules in angular 9. Data Binding 10. NgModel 11. Routing 12. Location Strategies
	Assignments
	1. Structure Assignment [Quizzes/Predict output/Errors/Syntax/Coding Challenge]
	1. What is npm? 2. Steps to create angular application? 3. Explain the folder structure of Angular application? 4. How do we add bootstrap to Angular application? 5. What are components in Angular? Can you explain the structure of a typical Angular component? 6. What is data binding in Angular? What types of data binding are available? 7. How does Angular handle events in templates? Can you provide an example? 8. What are the various lifecycle hooks in Angular? Name and explain a few. 9. What is the purpose of the ngOnInit lifecycle hook in Angular components? 10. Explain the difference between ngIf and ngFor directives in Angular. 11. What is the purpose of the @Input() and @Output() decorators in Angular?

12. How would you pass data from a parent component to a child component in Angular?
13. How Angular Routing works?
14. What is Location Strategies? Explain each Strategies, Give Examples.

2. Interactive Exercise [Code debugging/Logic Building/Mock interview]

Interactive example of **data binding** in Angular. We'll cover the main types of data binding: **Interpolation**, **Property Binding**, **Event Binding**, and **Two-way Binding**.

Follow the steps

- **Step 1: Set Up an Angular Component**

  Assume you already have a basic Angular application set up. If not, you can quickly generate a new Angular project using the Angular CLI:

  Code

```
ng new - -no-standalone data-binding-demo
cd data-binding-demo
ng serve
```

- **Step 2: Create a Component**

  Create a new component where we will demonstrate data binding:

  Code

```
ng generate component data-binding-demo
```

  This will generate the component in the src/app/data-binding-demo directory. We will modify this component to showcase different types of data binding.

- **Step 3: Modify the Component**

  Open the data-binding-**demo.component.ts** file and add some properties and methods to demonstrate data binding:

data-binding-demo.component.ts

Code

```
import { Component } from '@angular/core';
@Component({
      selector: 'app-data-binding-demo',
      templateUrl: './data-binding-demo.component.html',
      styleUrls: ['./data-binding-demo.component.css']
})

export class DataBindingDemoComponent {
 // Property Binding Example
    title = 'Angular Data Binding Demo';
 // Event Binding Example
     count = 0;
 // Two-way Binding Example
    userName = '';
// Method for event binding
   incrementCount() {
      this.count++;
    }
}
```

- **Step 4: Update the Template**

  Now, open the **data-binding-demo.component.html** file and add the necessary bindings to demonstrate each type of data binding:

  data-binding-demo.component.html

  Code

```
<h1>{{ title }}</h1> <!-- Interpolation -->
<!-- Property Binding -->
<p [innerText]="title"></p>
<!-- Event Binding -->
<button (click)="incrementCount()">Click me! {{ count }}</button>
<!-- Two-way Binding -->
<input [(ngModel)]="userName" placeholder="Enter your name">
<p>Your name is: {{ userName }}</p>
```

- **Step 5: Ensure FormsModule is imported**

  For **two-way binding** with ngModel to work, make sure you have imported FormsModule in your app module (app.module.ts).

  ## app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';  // Import FormsModule
import { AppComponent } from './app.component';
import { DataBindingDemoComponent } from './data-binding-demo/data-binding-demo.component';

@NgModule({
 declarations: [
   AppComponent,
       DataBindingDemoComponent
 ],
imports: [
 BrowserModule,
 FormsModule  // Add FormsModule here

 ],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

- **Step 6: Explanation of Binding Types**

  Now let's walk through what each type of data binding does:

1. **Interpolation ({{ }}):**
   - The {{ title }} syntax allows you to bind the value of a component property to a part of the template.
   - In this case, it will display the value of the title property ("Angular Data Binding Demo") inside the <h1> element.
2. **Property Binding ([ ]):**
   - The [innerText]="title" syntax binds the innerText property of the <p> element to the title property in the component.
   - This is a **one-way binding** from the component property to the DOM element.
3. **Event Binding (( )):**

- The (click)="incrementCount()" syntax binds the click event of the button to the incrementCount() method in the component.
- Each time the button is clicked, the method increments the count property, which is then displayed next to the button.

4. **Two-way Binding ([(ngModel)]):**
   - The [(ngModel)]="userName" syntax binds the value of the input field to the userName property in the component.
   - As you type in the input field, the value of userName is updated automatically, and vice versa. This is **two-way data binding**.

- **Step 7: Run the Application**

  Now, run your Angular app to see the data binding in action. You should be able to:

  1. See the title displayed using **Interpolation**.
  2. Modify the title text using **Property Binding**.
  3. Increment the count using the **Event Binding** (by clicking the button).
  4. Update your name using **Two-way Binding** (in the input field).

  Resulting Output
  Your app will display:

- A title at the top of the page.
- A button that shows and increments the count when clicked.
- An input field where you can type your name, and it will be displayed dynamically below the input

3. **Scenario Based Exercise [Project Simulations/Code Reviews/Task Ownership]**

create a **Shopping Cart Application** in Angular

Step 1: Set Up the Angular Project

1. Create a new Angular project using the Angular CLI with `ng new shopping-cart-app`.
2. Navigate to the project directory and start the development server using `ng serve`.

Step 2: Generate Components

3. Use Angular CLI to generate the following components:
   - **ProductListComponent**: To display the list of available products.

- **CartComponent**: To display the items in the cart.

Step 3: Define the Product Model

1. Create a model to represent a product. The model will contain properties like:
   - `id`: A unique identifier for the product.
   - `name`: The name of the product.
   - `price`: The price of the product.
   - `quantity`: The quantity of the product in the cart (default to 1).

Step 4: Set Up Routing

1. Define routes to navigate between the Product List page and the Cart page in the `app-routing.module.ts`:
   - `path: ''`: Displays the `ProductListComponent` by default.
   - `path: 'cart'`: Displays the `CartComponent`.

Step 5: Create a Service for Cart Management

1. Create a service to manage the shopping cart, which will allow:
   - Adding products to the cart.
   - Removing products from the cart.
   - Updating the quantity of a product in the cart.
   - Calculating the total price of items in the cart.

Step 6: Implement Product List Component

1. **ProductListComponent** should display a list of products available for purchase.
2. The component will have an "Add to Cart" button for each product, allowing users to add the selected product to the shopping cart.
3. Use data binding to show the list of products.

Step 7: Implement Cart Component

1. **CartComponent** should display the list of items in the cart.
2. Each cart item should show the product's name, price, quantity, and total cost (quantity * price).
3. Provide input fields for users to update the quantity of each product in the cart.
4. Allow users to remove items from the cart.
5. Display the total price of all the items in the cart.

Step 8: Update the Main App Component

1. In the main app component (`app.component.html`), include a navigation menu with links to the **Product List** and **Cart** pages.

2. Use `<router-outlet>` to display the routed components.

Step 9: Add FormsModule for Two-Way Binding

1. Import `FormsModule` in `app.module.ts` to enable two-way data binding with `ngModel` (for input fields like quantity in the cart).

Step 10: Run the Application

1. Use `ng serve` to start the application.
2. Navigate through the app, adding products to the cart, modifying quantities, and removing items from the cart.
3. The total price should update dynamically as items are added, removed, or quantities are changed.

4. Daily coding Exercise

Exercise
create the **Employee Management System** in Angular, focusing on **routing**, **components**, and **navigation**,

Step 1: Set Up the Angular Project

1. Create a new Angular project using the Angular CLI.
2. Start the development server to view your app in the browser.

Step 2: Generate Components

1. Generate two components:
   - **Employee List Component**: This component will display a list of employees.
   - **Employee Details Component**: This component will display detailed information for a selected employee.

Step 3: Define Routing

1. Define routes for the application in the routing module:
   - Set up a default route that will display the Employee List page.
   - Set up a route for displaying the Employee Details page, which should accept an employee ID as a parameter.

Step 4: Implement Employee List Page

1. In the Employee List component, display a list of employees.
2. Provide a link for each employee that navigates to their details page.

3. Use Angular's router to dynamically generate the URL for each employee using their ID.

Step 5: Implement Employee Details Page

1. In the Employee Details component, retrieve the employee ID from the route parameters.
2. Use this ID to display detailed information about the selected employee (such as their name, position, and department).

Step 6: Add Navigation to the App

1. Create a navigation bar in the main component.
2. Include links to navigate between the **Employee List** and **Employee Details** pages.
3. Use Angular's **router-outlet** to display the component based on the current route.

Step 7: Run the Application

1. Start the Angular application.
2. Test the navigation by:
    - Viewing the employee list.
    - Clicking on an employee's name to view their details on a separate page.

5. Gamification

Creating a gamification system for GT's to improve engagement, productivity, enhance teamwork, Encourage healthy competition.

Gamification Elements

1. Points
    a. Award 10 points for completing the assignments on time
    b. Extra points for presentation
2. Badges
    a. Graduate Trainee Champion for consistent performance
3. Leader boards
    a. Monthly leader board to recognize top performer

Angular	
Date	**Topic**
Day 3 **03-Feb-2025**	1. Directives 2. Inbuilt directives and types 3. Pipes 4. Inbuilt pipes 5. Classes 6. Interfaces
	Assignments
	1. Structure Assignment **[Quizzes/Predict output/Errors/Syntax/Coding Challenge]**
	1. What are directives in Angular? 2. What are different types of directives in Angular? 3. How are attribute directives different from structural directives? 4. What is the purpose of the @Directive decorator in Angular? **Attribute Directives** 1. How does the ngClass directive work? 2. Explain the use of the ngStyle directive with an example. 3. How do you create custom directive in Angular? **Structural Directives** 4. What does the *ngIf directive do, and how is it used? 5. How does the *ngFor directive work? Can you provide an example? 6. How does *ngSwitch differ from *ngIf? 7. Why do structural directives use an Asterix (*) in Angular? 8. Can you explain the purpose of ElementRef and Renderer2 in custom directives? 9. How does @HostListener work in custom directives? 10. What is @HostBinding, and how is it used?

11. How can you pass input properties to a custom directive?
12. How can you use multiple directives on the same element?
13. How do you manage performance concerns when using directives in Angular?
14. What is the difference between a directive and a component in Angular?

5. Interactive Exercise [Code debugging/Logic Building/Mock interview]

Create an Angular component that uses attribute and structural directives, and a build a custom directive to dynamically change the text color of elements based on user interactions.

Exercise Steps:

1. Setup

Create a new Angular project if you don't have one:

Code

```
ng new DirectiveExercise
cd DirectiveExercise
ng serve
```

2. Create a Component

Generate new component

Code

```
ng generate component directive-exercise
```

Update the **app.component.html**:

Code

```
<app-directive-exercise></app-directive-exercise>
```

3. Task 1: Use Attribute Directives (ngStyle, ngClass)

In **directives-expercise-component.html:**

Code

```
<h2>Attribute Directives Exercise </h2>
<p [ngStyle]="{'color' : textColor, 'font-weight':isBold?
'bold': 'normal'}">
      This text has dynamic styles.
</p>
<button (click)="toggleTextColor()">Toggle Text Color
</button>
<button (click)="toggleBold()">Toggle Text Bold </button>
```

In **directive-exercise-component.ts**

Code

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

```
      selector: 'app-directive-exercise',
      templateUrl: './directive-exercise.component.html`,
      styleUrls: ['./directive-exercise.component.css]
})

export class DirectiveExerciseComponent{
     textColor:string='blue';
     isBold:boolean=false;

     toggleTextColor(){
     this.textColor=this.textColor==='blue' ? 'red' : 'blue';
     }
     toggleBold()
     {
          this.isBold=!this.isBold;
     }
}
```

4. **Task 2: Use Structural Directives (*ngIf, *ngFor)**

   Extend the template:

```
<h2>Structural Directive Exercise </h2>
<button (click)="toggleList()">Toggle Item List
</button>
<ul *ngIf="showList">
  <li *ngFor="let item of items; let i=index">
          Item {{ i+1}}:{{ Item}}
 </li>
</ul>
```

   In **directive-exercise.component.ts:**

```
Items:string[]=['Apple', 'Banana','Cherry'];
showList:boolean=true;

toggleList(){
    this.showList=!this.showList;
}
```

5. **Task 3: Create a Custom Directive (appHighLight)**

   Generate the directive:

```
ng generate directive highlight
```

   In **highlight.directive.ts**

```
Import { Directive, ElementRef, HostListener, Render2}
from '@angular/core`;

@Directive({
      Selector:`[appHighLight]`
})

Export class HighlightDirective {
  Constructor(private el:ElementRef, private
render:Renderer2{}


@HostListener ('mousenter') onMouseEnter () {
```

```
this.renderer.setStyle(this.el.nativeElement,'color','gr
een');
}
@HostListener ('mouseleave') onMouseLeave () {
                    This.renderer.setStyle
(this.el.nativeElement,'color','black');
                }
            }
```

6. Task 4: Use the Custom Directive

Add this to the template:

```
<p appHighLight> Hover over this text to change its
color! </p>
```

7. Scenario Based Exercise [Project Simulations/Code Reviews/Task Ownership]

1. Introduction

The **Expense Tacker System** is simple yet powerful tool to help users manage and monitor their expenses. The system will allow users to categorize expenses, view them in structured format, and identify high-value expenses using visual cues. This solution will be built using Angular for the front-end and TypeScript to ensure type-safe and scalable code.

2. Solution Architecture

The Expense Tracker System consists of the following components:

1. Front End

- Angular -based user interface for input and display
- Component-based architecture for maintainability.

2. Data Management

- Use TypeScript Class and Interface for defining the Expense data model
- In-memory data storage for tracking expenses.

3. Business Logic:

- Adding, categorizing and viewing expenses.
- Highlighting high-value expenses using custom directives.

Expense Model (Class & Interface)

1. **Interface Expense** (id, description, amount, category ['Food |Travel |Entertainment |Others'])
2. **Class ExpenseClass**
3. Create **Component expense-list**
4. **Create directive** highExpense [use color red and black]
5. **Create Custom Pip for Currency Formatting**
6. **Do Application Setup [**app.module.ts**]**
7. **Run the application**

3. Daily coding Exercise

Exercise 1 Pipe Exercise: Temperature Converter

Objective

Create a custom pipe that converts temperatures between Celsius and an optional parameter ('F' or 'C').

Instructions:

- Define a custom pipe named TemperatureConverterPipe.
- The pipe should accept a value (temperature in Celsius) and an optional parameter ('F' or 'C').
- Convert the temperature to the request unit:
  - If 'F' is passed, convert, Celsius to Fahrenheit.
  - If 'C' is passed, convert Fahrenheit to Celsius.
- Display a list of temperature reading in both units.

Expected Output:

25^0C is 77^0F
0^0C is 32^0F

Exercise 2 Classes and Interface exercise

E-learning Platform

Objective:

Build an e-learning platform with courses and Enrolments using interfaces and classes.

Instructions:

1. Define an ICourse interface with properties: id, title, duration and category.
2. Create a Course class implementing the ICourse interface.
3. Write a service to manage course enrolments and display enrolled courses.

[Note: only class and interface need in this example no methods]

Exercise 3 List the Built-in Pipes in Angular [Give Examples]

4. Gamification

Creating a gamification system for GT's to improve engagement, productivity, enhance teamwork, Encourage healthy competition.

Gamification Elements

1. Points
    a. Award 10 points for completing the assignments on time
    b. Extra points for presentation
2. Badges
    a. Graduate Trainee Champion for consistent performance
3. Leader boards
    a. Monthly leader board to recognize top performer

Angular	
Date	**Topic**
Day 4 04-Feb-2025	1. Template Driven form 2. Reactive Components 3. Form 4. Data Sharing Between

Assignments

1. Structure Assignment [Quizzes/Predict output/Errors/Syntax/Coding Challenge]

1. What are two types of form in Angular?
2. What is the difference between Template-driven and Reactive forms in Angular?
3. How do you define a Template-driven form in Angular?
4. How do you define a Reactive form in Angular?
5. How can you apply validations to Angular forms?
6. What are the common validators in Angular forms?
7. How do you create custom validators in Angular?
8. What is FormGroup and FormControl in Angular?
9. How do you get the value of form in Angular?
10. How do you handle form submission in Angular?
11. What is ngModel in Angular?
12. What is the FormBuilder service in Angular?
13. How do you dynamically add form controls to a form?
14. How do you disable or enable form controls?
15. How do you check if a form is valid or invalid in Angular?

2. Interactive Exercise [Code debugging/Logic Building/Mock interview]

Interactive Exercise on Angular forms.
Creating simple reactive form with validation and dynamic form control creation. We will also learn to submit the form, validate the inputs, and display the form data.

Here the step by step exercise

Objective

We will create a **registration form** with the following fields:

1. Username (required)
2. Password (required, minimum 6 characters)
3. Confirm password (should match the password)
4. Age (required, should be a number, min 18)

Step 1 Set up your Angular Application

Generate a new component called registration:

```
ng new angular-forms-example
```

cd angular-forms-example

ng add @angular/forms

Step 2 Create the Component

```
ng generate component registration
```

Step 3 Update the Module

```
import {NgModule} from '@angular/core';
import {BrowserModule} from '@angular/platform-browser';
import {ReactiveFormsModule} from '@angular/forms'; // Add
this import

import {AppComponent} from './app.component';
import {RegistrationComponent} from './registration/registrat
ion.component';

@NgModule{
  declarations: [
    AppComponent,
    RegistrationComponent
  ],
  imports: [
    BrowserModule,
    ReactiveFormsModule// Add this in imports array
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule{}
```

Step 4 Build the Registration Form

In **registration.component.ts,** create the form with validation.

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/
forms';

@Component({
  selector: 'app-registration',
  templateUrl: './registration.component.html',
  styleUrls: ['./registration.component.css']
})
export class RegistrationComponent implements OnInit {
  registrationForm: FormGroup;

  constructor(private fb: FormBuilder) {}

  ngOnInit() {
    this.registrationForm = this.fb.group({
      username: ['',
[Validators.required, Validators.minLength(3)]],
      password: ['',
[Validators.required, Validators.minLength(6)]],
      confirmPassword: ['', [Validators.required]],
```

```
      age: ['', [Validators.required, Validators.min(18)]],
    }, { validators: this.passwordMatchValidator });
  }

  // Custom validator for password matching
  passwordMatchValidator(control: FormGroup): {
[key: string]: boolean } | null {
    if (control.get('password')?.value !==
control.get('confirmPassword')?.value) {
      return { 'passwordMismatch': true };
    }
    return null;
  }

  onSubmit() {
    if (this.registrationForm.valid) {
      console.log('Form Data:', this.registrationForm.value);
    } else {
      console.log('Form is invalid');
    }
  }
}
```

Step 5 Create the HTML Template

In **registration.component.html**, define the form layout:

```
<div class="container">
  <h2>Registration Form</h2>
  <form [formGroup]="registrationForm"
(ngSubmit)="onSubmit()">

    <div>
      <label for="username">Username:</label>
      <input id="username" formControlName="username"
type="text">
      <div *ngIf="registrationForm.get('username')?.invalid
&& registrationForm.get('username')?.touched">
        <small
*ngIf="registrationForm.get('username')?.hasError('required')
">Username is required.</small>
        <small
*ngIf="registrationForm.get('username')?.hasError('minlength'
)">Username must be at least 3 characters.</small>
      </div>
    </div>

    <div>
      <label for="password">Password:</label>
      <input id="password" formControlName="password"
type="password">
      <div *ngIf="registrationForm.get('password')?.invalid
&& registrationForm.get('password')?.touched">
        <small
*ngIf="registrationForm.get('password')?.hasError('required')
">Password is required.</small>
        <small
*ngIf="registrationForm.get('password')?.hasError('minlength'
)">Password must be at least 6 characters.</small>
      </div>
```

```
    </div>

    <div>
      <label for="confirmPassword">Confirm Password:</label>
      <input id="confirmPassword"
formControlName="confirmPassword" type="password">
      <div
*ngIf="registrationForm.hasError('passwordMismatch') &&
registrationForm.get('confirmPassword')?.touched">
        <small>Passwords must match.</small>
      </div>
    </div>

    <div>
      <label for="age">Age:</label>
      <input id="age" formControlName="age" type="number">
      <div *ngIf="registrationForm.get('age')?.invalid &&
registrationForm.get('age')?.touched">
        <small
*ngIf="registrationForm.get('age')?.hasError('required')">Age
is required.</small>
        <small
*ngIf="registrationForm.get('age')?.hasError('min')">Age must
be at least 18.</small>
      </div>
    </div>

    <button type="submit"
[disabled]="registrationForm.invalid">Submit</button>
  </form>
</div>
```

Step 6 Add Styles

In **registration.component.css**, add some basic styles:

```
.container {
  width: 300px;
  margin: 0 auto;
  padding: 20px;
  border: 1px solid #ccc;
}

input {
  width: 100%;
  padding: 8px;
  margin-bottom: 10px;
}

button {
  width: 100%;
  padding: 10px;
  background-color: #007bff;
  color: white;
  border: none;
  cursor: pointer;
}

button:disabled {
  background-color: #ccc;
```

```
}

small {
  color: red;
}
```

Step 7 Run the application

```
ng serve
```

3. **Scenario Based Exercise [Project Simulations/Code Reviews/Task Ownership]**

Student Management System

Objective:

Develop a **Student Management System** using Angular to enable educational institutions or administrators to manage student information efficiently, including registration, listing student details, and tracking key statistics.

Scope:

The system should support the following operations:

- **Register Students:** Admin can add student details (name and age);
- **View Student List:** Display a real-time list of registered students.
- **Tack Statistics:** Show key metrics such as the total number of students and their average age.

Functional Requirements:

1. **Student Registration Form**
   - A **Reactive Form** for capturing student information**:**
     - **Name:** Required, minimum of 2 characters.
     - **Age:** Required, numeric, between 5 and 100
   - Show appropriate validation error messages.
   - Submit the form to add a student to the list.
2. **Display Student List**
   - Show a list of registered students with their name and age.
   - The list should update reactively when a new student is added.
3. **Statistics Dashboard**
   - Display the following metrics.
     - Total number of registered students
     - Average student age.
   - Update statistics in real-time as the list changes.

Key Features & Implementation Strategy

1. **StudentFormComponent**
   - Create a form using **ReactiveFormsModule.**
   - Handle form validation and submission.
2. **StudentListComponent**
   - Use **@Input** to accept and display the list of students.
3. **StudentStatsComponent**
   - Use **@Input** to receive student data and compute real-time statistics.

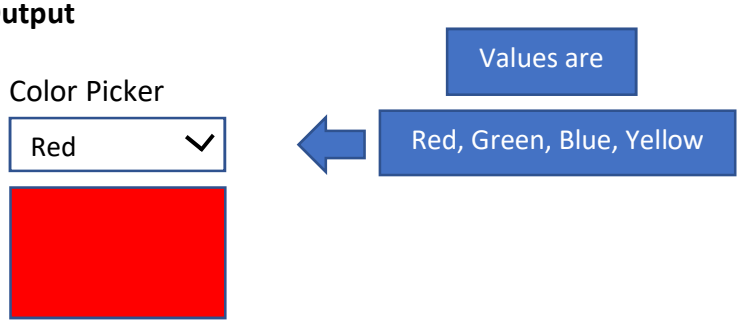
4. Daily coding Exercise

Exercise 1

Color-Picker with Event Binding and Property Binding
Create a **Color Picker App** that

1. Allows the user to select a color from a dropdown.
2. Changes the background color of a div based on the selected color.

Output



Exercise 2

Build an Employee Management form where

1. The **Parent Component (EmployeeFormComponent)** contains a reactive form to input employee details such as name, email and department.
2. The **department selection** in the child component updates the form in the parent component using event communication.

Output

Employee Management System
Employee Registration Form

Select Department:

HR

Finance

IT

Marketing

Submit

5. Gamification

Creating a gamification system for GT's to improve engagement, productivity, enhance teamwork, Encourage healthy competition.

Gamification Elements

1. Points
    a. Award 10 points for completing the assignments on time
    b. Extra points for presentation
2. Badges
    a. Graduate Trainee Champion for consistent performance
3. Leader boards
    a. Monthly leader board to recognize top performer

Bootstrap	
Date	**Topic**
Day 5 **05-Feb-2025**	1. Services 2. Providers 3. Dependency Injection 4. Rxjs
	Assignments
	1. Structure Assignment **[Quizzes/Predict output/Errors/Syntax/Coding Challenge]**

2. Interactive Exercise [Code debugging/Logic Building/Mock interview]

3. **Scenario Based Exercise [Project Simulations/Code Reviews/Task Ownership]**

4. **Daily coding Exercise**

5. Gamification

Creating a gamification system for GT's to improve engagement, productivity, enhance teamwork, Encourage healthy competition.

Gamification Elements

1. Points
    a. Award 10 points for completing the assignments on time
    b. Extra points for presentation
2. Badges
    a. Graduate Trainee Champion for consistent performance
3. Leader boards
    a. Monthly leader board to recognize top performer

Bootstrap	
Date	**Topic**
Day 6 **6-Feb-2025**	1. Data Stream 2. Observable 3. Subscription 4. Subject 5. HttpClient Module
	Assignments
	1. Structure Assignment [Quizzes/Predict output/Errors/Syntax/Coding Challenge]
	2. Interactive Exercise [Code debugging/Logic Building/Mock interview]
	3. **Scenario Based Exercise [Project Simulations/Code Reviews/Task Ownership]**
	4. **Daily coding Exercise**
	5. Gamification
	Creating a gamification system for GT's to improve engagement, productivity, enhance teamwork, Encourage healthy competition. **Gamification Elements** 1. Points a. Award 10 points for completing the assignments on time b. Extra points for presentation 2. Badges

	a. Graduate Trainee Champion for consistent performance 3. Leader boards a. Monthly leader board to recognize top performer

Bootstrap	
Date	**Topic**
Day 7 7-Feb-2025	1. Http Interceptors 2. HttpRequest 3. HttpHandler 4. AuthGuards 5. Type of guards
	Assignments
	1. Structure Assignment [Quizzes/Predict output/Errors/Syntax/Coding Challenge]
	2. Interactive Exercise [Code debugging/Logic Building/Mock interview]
	3. **Scenario Based Exercise [Project Simulations/Code Reviews/Task Ownership]**
	4. **Daily coding Exercise**
	5. Gamification

	Creating a gamification system for GT's to improve engagement, productivity, enhance teamwork, Encourage healthy competition. **Gamification Elements** 1. Points a. Award 10 points for completing the assignments on time b. Extra points for presentation 2. Badges a. Graduate Trainee Champion for consistent performance 3. Leader boards a. Monthly leader board to recognize top performer

Bootstrap	
Date	**Topic**
Day 8 **8-Feb-2025**	1. Lazy Loading 2. Angular Persistence 3. Ngrx State Management
	Assignments
	1. Structure Assignment [Quizzes/Predict output/Errors/Syntax/Coding Challenge]
	2. Interactive Exercise [Code debugging/Logic Building/Mock interview]
	3. **Scenario Based Exercise [Project Simulations/Code Reviews/Task Ownership]**

4. **Daily coding Exercise**

5. Gamification

Creating a gamification system for GT's to improve engagement, productivity, enhance teamwork, Encourage healthy competition.

Gamification Elements

1. Points
    a. Award 10 points for completing the assignments on time
    b. Extra points for presentation
2. Badges
    a. Graduate Trainee Champion for consistent performance
3. Leader boards
    a. Monthly leader board to recognize top performer