## Investment Calculator Using Angular

### Project Overview

This project is an Investment Calculator built using Angular. It consists of three primary components: Header, User Input, and Investment Results. The calculator allows users to input their investment details and calculates the growth of the investment over time.

#### 1. Angular Components

**Purpose**: Components are the fundamental building blocks of Angular applications. They encapsulate the template, style, and logic for a portion of the user interface.

**Key Concepts**:

* **Component Decorator**: Defines metadata for the component using @Component. This includes the selector, template URL, and style URLs.
* import { Component } from '@angular/core';  
    
  @Component({  
   selector: 'app-header',  
   templateUrl: './header.component.html',  
   styleUrls: ['./header.component.css']  
  })  
  export class HeaderComponent {}
* **Template**: HTML structure that defines what the view for the component looks like.
* **Style**: CSS or SCSS files that define the appearance of the component.

**Important Points**:

* **Encapsulation**: Angular's encapsulation ensures that styles defined in a component do not affect other components.
* **Lifecycle Hooks**: Components have lifecycle hooks (e.g., ngOnInit, ngOnDestroy) that allow you to tap into different phases of a component's existence.

#### 2. Dependency Injection (DI)

**Purpose**: DI is a design pattern used to implement IoC (Inversion of Control). It allows services and other dependencies to be injected into components or other services.

**Key Concepts**:

* **Injectable Decorator**: Marks a class as available to be injected using the dependency injection system.
* import { Injectable } from '@angular/core';  
    
  @Injectable({ providedIn: 'root' })  
  export class InvestmentService {}
* **Constructor Injection**: Dependencies are injected via the component's or service's constructor.
* constructor(private investmentService: InvestmentService) {}

**Important Points**:

* **Singleton Services**: By default, services provided in the root are singletons, meaning only one instance of the service exists in the application.
* **Providers**: You can specify providers in the component or module to limit the scope of the service.

#### 3. Two-way Data Binding

**Purpose**: Two-way binding allows synchronization of data between the component class and the template. Changes in the template update the component, and changes in the component update the template.

**Key Concepts**:

* **ngModel Directive**: Facilitates two-way data binding in template-driven forms.
* <input [(ngModel)]="enteredInitialInvestment" />

**Important Points**:

* **FormsModule**: Must be imported to use ngModel.
* import { FormsModule } from '@angular/forms';

#### 4. Angular Directives

**Purpose**: Directives are special markers in the DOM that extend HTML's capabilities. They can be structural (e.g., \*ngIf, \*ngFor) or attribute directives.

**Key Concepts**:

* **Structural Directives**: Change the structure of the DOM by adding or removing elements.
* <div \*ngIf="results">Content goes here</div>  
  <tr \*ngFor="let result of results">{{ result.year }}</tr>

**Important Points**:

* **Syntax**: Structural directives use \* before the directive name.
* **Context Variables**: \*ngFor can use local variables like let result of results.

#### 5. Services

**Purpose**: Services in Angular are used to share data and logic across multiple components. They often handle business logic and data retrieval.

**Key Concepts**:

* **Service Implementation**: Contains methods and properties used by components.
* export class InvestmentService {  
   resultData?: { year: number; ... }[];  
    
   calculateInvestmentResults(data: InvestmentInput) {  
   // Implementation logic  
   }  
  }
* **State Management**: Services can hold state that components can subscribe to or access.

**Important Points**:

* **Separation of Concerns**: Services help keep components lean by moving business logic out of the components.
* **Reusability**: Services can be reused across different components.

#### 6. Angular Modules

**Purpose**: Angular modules help organize an application into cohesive blocks of functionality.

**Key Concepts**:

* **NgModule Decorator**: Defines an Angular module with metadata about its components, directives, services, and other modules.
* import { NgModule } from '@angular/core';  
  import { BrowserModule } from '@angular/platform-browser';  
  import { FormsModule } from '@angular/forms';  
    
  @NgModule({  
   declarations: [AppComponent, HeaderComponent, UserInputComponent, InvestmentResultsComponent],  
   imports: [BrowserModule, FormsModule],  
   bootstrap: [AppComponent]  
  })  
  export class AppModule {}

**Important Points**:

* **Bootstrap**: The root module specifies which component to bootstrap when the application starts.
* **Imports and Declarations**: Modules import other modules and declare the components, directives, and pipes they contain.

#### 7. Pipes

**Purpose**: Pipes are used to transform data in the template.

**Key Concepts**:

* **Built-in Pipes**: Angular provides several built-in pipes such as currency, date, and uppercase.
* {{ result.valueEndOfYear | currency: "INR" }}

**Important Points**:

* **Chaining Pipes**: You can chain multiple pipes together.
* **Custom Pipes**: You can create custom pipes to handle specific transformations.

By understanding these key concepts in detail, you'll have a solid foundation for developing and maintaining Angular applications, including your Investment Calculator project.