**MVVM (Model-View-ViewModel)**

### ****MVVM Architecture in Angular****

* **Model**  
  Represents the application data and business logic (e.g., Services, Interfaces, APIs)
* **View**  
  The UI elements (HTML templates + Angular bindings)
* **ViewModel**  
  The component class (.ts) that connects View and Model

### 📊 ASCII Diagram

+----------------+ Bind Data +-------------------+ API/Data Logic +--------------------+

| View | <--------------------> | ViewModel | <------------------------> | Model |

| (HTML, UI) | (Two-way binding) | (Component .ts) | (Service, Interface) | (API, Business Logic)|

+----------------+ +-------------------+ +--------------------+

Example: home.component.html home.component.ts user.service.ts

------------------- ------------------ ------------------

<input [(ngModel)]="name"> name: string = ''; getUser(), saveUser()

### 🔄 Flow Explanation

| Layer | Role |
| --- | --- |
| **Model** | Talks to backend via services (HttpClient), handles business logic |
| **ViewModel** | Component class: holds UI data, methods, and handles events |
| **View** | HTML template: binds to component variables, emits events |

#### ****View (HTML)**** – app.component.html

<h1>Hello {{ name }}</h1>

<input [(ngModel)]="name" />

<button (click)="save()">Save</button>

#### ****ViewModel (Component)**** – app.component.ts

export class AppComponent {

name = '';

constructor(private userService: UserService) {}

save() {

this.userService.saveUser(this.name);

}

}

**Model (Service)** – user.service.ts

@Injectable({ providedIn: 'root' })

export class UserService {

saveUser(name: string) {

console.log("Saving user:", name);

// HTTP logic goes here

}

}

### ✅ Summary Table

| MVVM Role | Angular Entity | Responsibility |
| --- | --- | --- |
| Model | Service, Interface | Data fetching, persistence, business logic |
| ViewModel | Component Class (.ts) | Handles data & logic for the template |
| View | HTML Template | UI and user interactions |

## 🧩 MVVM in Angular (Simplified View)

pgsql

CopyEdit

+-------------+ +----------------+ +------------------+

| Template | <---> | Component (TS) | <---> | Model / Data |

| (HTML) | View | ViewModel | | Business Logic |

+-------------+ +----------------+ +------------------+

## 🧠 Roles of Each Layer in Angular MVVM

| Layer | Angular Equivalent | Responsibility |
| --- | --- | --- |
| **Model** | Data classes, interfaces, services, APIs | Represents business data and logic |
| **View** | \*.component.html | UI markup bound to component properties using binding |
| **ViewModel** | \*.component.ts class | Contains UI logic, acts as glue between view & model |

## ✅ Why the Component is the ViewModel

The Component in Angular:

* Exposes **data to the view** via public properties
* Responds to **user events** (e.g., click, input) via methods
* **Subscribes to services** to fetch/update data (the "model")
* Implements **form logic**, **validation**, and **UI state**

So it:

* **Translates raw data → UI-friendly structure**
* **Processes events from the view**
* **Keeps the view reactive** via property bindings

### 🔍 Quick Example

**product.model.ts**

ts

CopyEdit

export interface Product {

id: number;

name: string;

price: number;

}

**product.component.ts** (ViewModel)

ts

CopyEdit

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

import { Product } from './product.model';

@Component({

selector: 'app-product',

templateUrl: './product.component.html'

})

export class ProductComponent {

product: Product = { id: 1, name: 'Laptop', price: 75000 };

applyDiscount() {

this.product.price = this.product.price \* 0.9;

}

}

**product.component.html** (View)

html

CopyEdit

<h2>{{ product.name }}</h2>

<p>Price: ₹{{ product.price }}</p>

<button (click)="applyDiscount()">Apply Discount</button>

## 💡 Summary

| Angular Part | MVVM Role | Notes |
| --- | --- | --- |
| \*.component.html | **View** | Handles user interface and binding |
| \*.component.ts | **ViewModel** | Contains logic, event handlers, glue code |
| Model classes/services | **Model** | Business logic, persistence, API access |