Frank De Silva

ToDoList Architecture

An overview of the architecture and the reasons behind this choice

De Silva, Frank
8-11-2025

Contents

Introduction

Source Code repo : https://github.com/franknimaldesilva/ToDoList/tree/master

Angular having introduced signals in version 17 is working towards providing a form interface that is based on signals, as an addition to the current template forms , reactive forms. This is under active development and the source code can be found in the angular repo at https://github.com/angular/angular/blob/prototype/signal-forms/packages/forms/experimental/docs/signal-forms.md . This new forms are called Angular Signal Forms.

The key take aways from the above is given below

In Angular Signal Forms, this concept is modeled by breaking it down into four distinct parts:

1. **Data Model:** The structure and holds the current values of the data being collected by the form.
2. **Field State:** The metadata associated with each field, representing its state (e.g. valid, touched, dirty).
3. **Field Logic:** The business logic governing the form's behavior, such as validation and conditionally shown fields.
4. **UI Controls:** The actual HTML elements (e.g. <input>, <select>, custom components) that present the form fields to the user and allow interaction.

.....

A key principle of Angular Signal Forms is that it **does not maintain its own internal data**. Instead, **you, the developer, own the data model**. The data model is represented as a WritableSignal which the library uses directly as the source of truth for the values of its fields.

....

- Any changes the user makes via the userForm **directly modify** the model.

Reviews

An article written on comparing these new forms in Medium by Muhammad Awais on 27 May 2025 says the following

“At its core, Signal Forms introduce a model-first approach to building forms in Angular using WritableSignals. The form data doesn’t live in the form object – it lives in your model. That’s right the developer owns the model....

You model is the single source of truth. The form reads from it, writes to it, and reflects any changes

....

Why it matters

...

Cleaner, flatter API's (no need for FormGroups, FormControls, etc)

....

Signal Forms offer a declarative , signal-powered alternative to Reactive Forms…

Given the above trend the assessment project uses a model first approach. The models used in the assessment project have been designed to work with signals. A pathway has been created directly from the html controls to the models. This path way has all of the functionalities that would be provided Angular Signal Form. It is envisaged that when it does happen, an upgrade to use it would require minimal code change.

When it comes to models Angular leaves it coding totally up to the developer

"Of all the building blocks in an application, the data model is the one for which Angular is the least prescriptive. Elsewhere in the application, Angular requires specific decorators to be applied or parts of the API to be used, but the only requirement for the model is that it provides access to the data that the application requires; the details of how this is done and what that data looks like is left to the developer."

Adam Freeman Pro Angular

Core Architecture

On the basis of the above I based my design on my prior experience with WPF and Silverlight that have binding from XAML to the models in a similar fashion to what is shown in the Angular Signal Form source code.

A modern version of this taken from the publication Enterprise-Application-Patterns-Using.NET-MAUI from Microsoft is given below.

This Pattern is know as MVVM which consist of 3 components.

1. Model
2. View
3. ViewModel

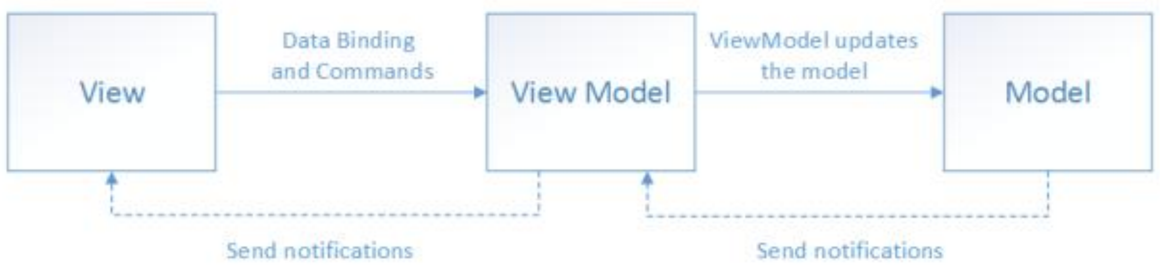


Fig 1: Take from publication Enterprise-Application-Patterns-Using.NET-MAUI page 9

Each of the components and their interactions is described as follows in this publication.

…

At a high level, the view "knows about" the view model, and the view model "knows about" the model, but the model is unaware of the view model, and the view model is unaware of the view. Therefore, the view model isolates the view from the model, and allows the model to evolve independently of the view

…

The benefits of using the MVVM pattern are as follows

• If an existing model implementation encapsulates existing business logic, it can be difficult or risky to change it. In this scenario, the view model acts as an adapter for the model classes and prevents you from making major changes to the model code.

• Developers can create unit tests for the view model and the model, without using the view. The unit tests for the view model can exercise exactly the same functionality as used by the view.

• The app UI can be redesigned without touching the view model and model code, provided that the view is implemented entirely in XAML or C#. Therefore, a new version of the view should work with the existing view model.

• Designers and developers can work independently and concurrently on their components during development. Designers can focus on the view, while developers can work on the view model and model components.

….

View

The view is responsible for defining the structure, layout, and appearance of what the user sees on screen. Ideally, each view is defined in XAML, with a limited code-behind that does not contain business logic. However, in some cases, the code-behind might contain UI logic that implements visual behavior that is difficult to express in XAML, such as animations.

In a .NET MAUI application, a view is typically a ContentPage-derived or ContentView-derived class. However, views can also be represented by a data template, which specifies the UI elements to be used to visually represent an object when it's displayed. A data template as a view does not have any code-behind, and is designed to bind to a specific view model type.

ViewModel

The view model implements properties and commands to which the view can data bind to, and notifies the view of any state changes through change notification events. The properties and commands that the view model provides define the functionality to be offered by the UI, but the view determines how that functionality is to be displayed.

Model

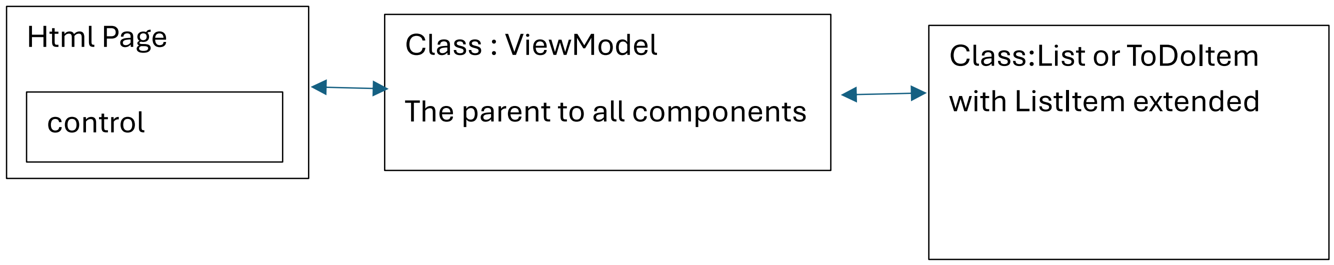
Model classes are non-visual classes that encapsulate the app's data. Therefore, the model can be thought of as representing the app's domain model, which usually includes a data model along with business and validation logic. Examples of model objects include data transfer objects (DTOs), Plain Old CLR Objects (POCOs), and generated entity and proxy objects. 12 CHAPTER 3 | Model-View-ViewModel (MVVM) Model classes are typically used in conjunction with services or repositories that encapsulate data access and caching.

Adaption MVVM to Angular

The Pattern was adapted in the assessment project as follows.

1. View : Html view in each component.
2. ViewModel : Component class with a common ViewModel to handle direct binding from Html controls to Model
3. Models : Models encapsulate the business rules and validation required by each section.

Code structure and binding pattern


Html Page
control
Class : ViewModel
The parent to all components
Class:List or ToDoItem with ListItem extended

Html Pages : View

The Html views uses the html input and textArea tags

The key feature with the binding is the id of any given html control must match the corresponding field name/property that it is editing.

Class : ViewModel

The ViewModel class that is the parent class of all components acts as the ViewModel. Providing all of the common ViewModel functionalities for each page, that is providing the channel for the binding to flow between the html controls and the corresponding properties in the associated model.

The main methods that bind the html controls in the view to the corresponding model properties is setValidity

The Models

There is two models List and ToDoItem with parent ListItem

Each property of the model has a corresponding validation function

e.g taskName => taskNameValidate

These functions are run on submit via the method on ListItem

```
 validate(): boolean {
    this.validationErrorList = [];
    let valid = true;
    let prop: keyof typeof this;
    for (prop in this) {
      if (prop.toString().toLowerCase().endsWith('validationptr')) {
        if (!(<Function>this[prop])()) {
          valid = false;
        }
      }
    }
    return valid;
  }
```

Conclusion

The ToDoLits project has been built taking into consideration the expected path of signal based form as envisaged by the Angular core development team. The approach is a model-first with the model capturing all the data from the form and implement all of the associated business rules for validation the data. A well proven pattern for this approach is that recommended by Microsoft the MVVM pattern, has been adapted into the angular framework in implementing the model-first approach. The main driver for this approach is the fact that it is what the angular team recognises as efficient approach to using signals with forms and the team itself is actively developing Signal Forms currently. The approach take is such that while independent of Signals Forms it has all of the plumbing to work efficiently, in the event that the core team were to release Signal Forms In the future, then then an upgrade to use them would be minimal as the current models are designed keeping this possibility in mind.