First Demo

Layered Pattern and MVC Pattern Combo

In this First of two App Demo I prepared, I used the Angular framework, which is in itself pushes developers to use Layered and MVC patterns to develop Web Applications.

In all the researches I have done, most conclusions come to the fact that Layered Pattern is a superset of MVC since it is layers of components achieving independence, making Layered higher level and reinforcing the theory of major overlapping between Architectural Patterns.

Four layers in this Angular Demo:

1. Presentation layer or View (in MVC) – html and CSS of the Angular Demo
2. Application layer or Controller (in MVC) – view component controller of the Angular Demo
3. Business Logic layer or Controller (in MVC) – services and service level models of the Angular Demo
4. Data Access layer or Model (in MVC) – APIs and models of the Angular Demo

The Demo is a simple Dictionary with a partially implemented User Signup/login due to time issues. Aside from the navigational perks added through Angular Packages to help with the UI and user navigation, it showcases the basics of a NoSQL Data store manipulation through the very intense abstraction achieved by the Layered and MVC Patterns.

At the DAL (Data Access Layer), Models represent the MongoDB database I used for the easy visualization and manipulation the actual persistent Data Store and these Models can then be used in the basic get/post APIs to change the states of Data view or Data Store.

At the Business Logic Layer, Models are still build to bridge the gap between the persistent layer (DAL) and the Application layer. These models are basically a much higher representation of the MongoDB, specifically built to handle the view data. Services built at this layer, make use of the high level models to achieve business logic for the manipulation of Data.

At the Application layer, the view component controller basically dictates how the view should handle any changes.

At the Presentation layer, the html and CSS make up the user interface of the dictionary demo.

Second Demo

Broker Pattern

In this second Demo, the Broker Pattern is used, which constitutes a middle man (broker) between a typical communications between parties. As these parties grow in size and one party starts demanding more lines of communication, the cluster in the middle can become chaotic and unmanageable. Hence the need for the Broker Pattern which regulates the exchange in the middle, by identifying what the sender is sending and what the receiver is waiting for.

The App Demo I prepared is a very basic and simple broker implementation, by basically avoiding any of the source (publisher) and receiver (subscriber) parties and going straight to the middle man (broker component) logic. The Demo is what I hope to be the most basic of brokers, including a publisher, subscriber and Unsubscribe methods.