Introduction to AngularJS

Introduction

AngularJS is a structural framework for building dynamic web applications. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular was developed by Google around 2009 and is maintained by them on an ongoing basis.

Angular is one of a growing number of what is called **MVC** type frameworks which includes along with angular, ember.js, backbone.js and knockout.js. The term **MVC** standing for Model View Controller and up until recently referred mainly to server side applications. We will look at this 'Design Pattern' in this lesson.

AngularJS is a framework and as such imposes a certain **structure** on a developer. This structure can roughly be described by referring to common application design patterns which we will look at in more detail in this lesson. These design patterns attempt to 'separate' out components of an application into some form of logical structure. This is often referred to as a 'separation of concerns'. In AngularJS these components or constructs are referred to as controllers, services, directives, constants and filters. A framework is then responsible for managing the initialization between these components and the interaction between them. The developer's task is then to 'fill' the components with their own code.

There are many production web applications out there you that have been built with Angular . You can see the growing number of them on
https://builtwith.angularjs.org/

Framework vs Library

We stated in the introduction that Angular is a framework. You may have also heard libraries being referred to when discussing libraries such as JQuery. What is the difference between a library and a framework ?

A library such as JQuery is essentially a library full of javascript functions which a developer can use once they have included the JQuery library javascript file in their HTML file. Once we have access to the JQuery library we can make use of the available functions by doing something like

```
// jQuery
$('#btn').one('click', function() {
  $('#msg').append(' World');
});
```

The functions one() and append() are JQuery functions. In the case of a library it is up to the developer when and how they use the functions available to them from a library. No structure or organization is imposed. In a sense the developer is **'in control'** , not the library.

Contrast the above to the situation with a framework. A framework imposes a structure on the developer which they must buy into. There could be said to be an '**Inversion of Control**' where the control is imposed by the framework and the developer need to follow by 'plugging' in their code into the components of the framework structure. In this way, a developer's code will be called by the framework when appropriate.

Design Patterns

Before discussing the specifics of the Angular Framework will discuss a few design patterns that can be used to describe the angular approach to building an application which we referred to in the introduction.

The MVC Design Pattern

MVC, Model-View-Controller, at a high level, this design pattern refers to keeping a **clean separation** between

- The code in an application or component for the **view** or user interface presentation
- The **controller** an intermediary between the view and the model, marshalling model data for passing onto a view or presentation of the model data
- The **model** or the data representation.

Clean refers to limiting interdependencies between the three layers. Code written in this fashion is easier to maintain and upgrade, as any layer can be changed without affecting the rest of the code. The goal of MVC and related patterns is to separate data management and presentation.

MVC DESIGN PATTERN

MVVM Pattern

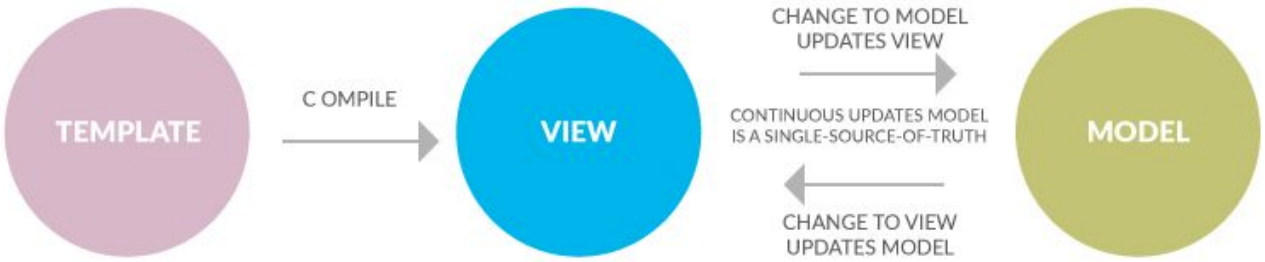
Another design pattern that is closely aligned to the way the Angular framework is structured is the **Model-View View-Model** design pattern (MVVM).

MVVM DESIGN PATTERN

It is similar to MVC but in this pattern the **View-Model** is the object representing the state of both the application data and the views representation of it.

Two Way Binding

Another concept that the Angular framework encapsulates is the the concept of two-way data binding.



TWO WAY BINDING

Angular's implementation of data-binding lets you treat the model as the **single-source-of-truth** in your application. The view is a projection of the model at all times. When the model changes, the view reflects the change, and vice versa.

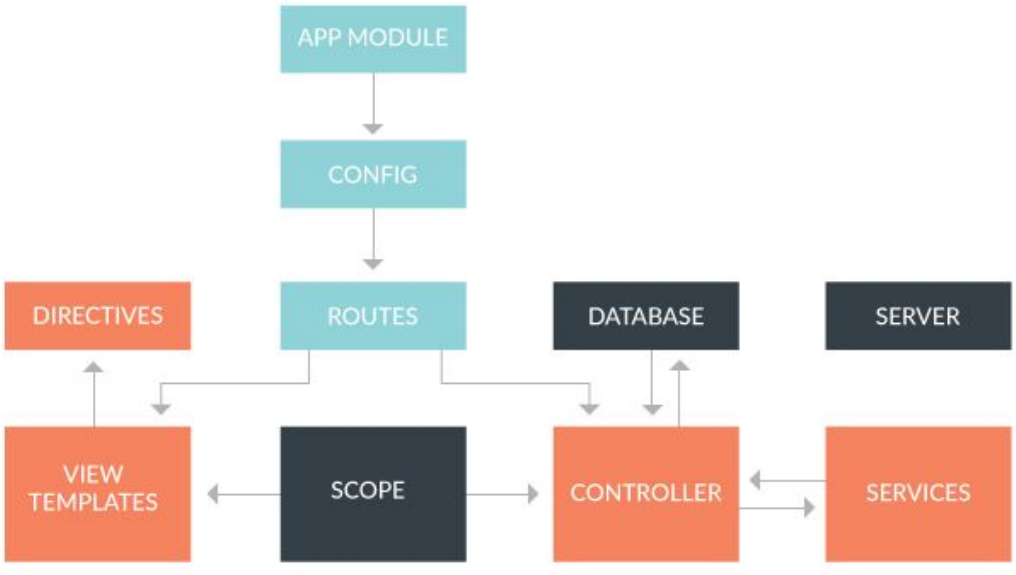
Data-binding is an automatic way of updating the view whenever the model changes, as well as updating the model whenever the view changes.

Behind the two way data binding is the VM or View-Model component of MVVM. The view-model is the object representing the state of both the application data and the views representation of it. The flow is bi-directional. A change on one side necessitates a change on the other automatically.

The VM in AngularJS is represented by the **$scope** object which the framework *injects* into controller functions as a parameter.

AngularJS Application Structure

We can depict an angular application diagrammatically as below:

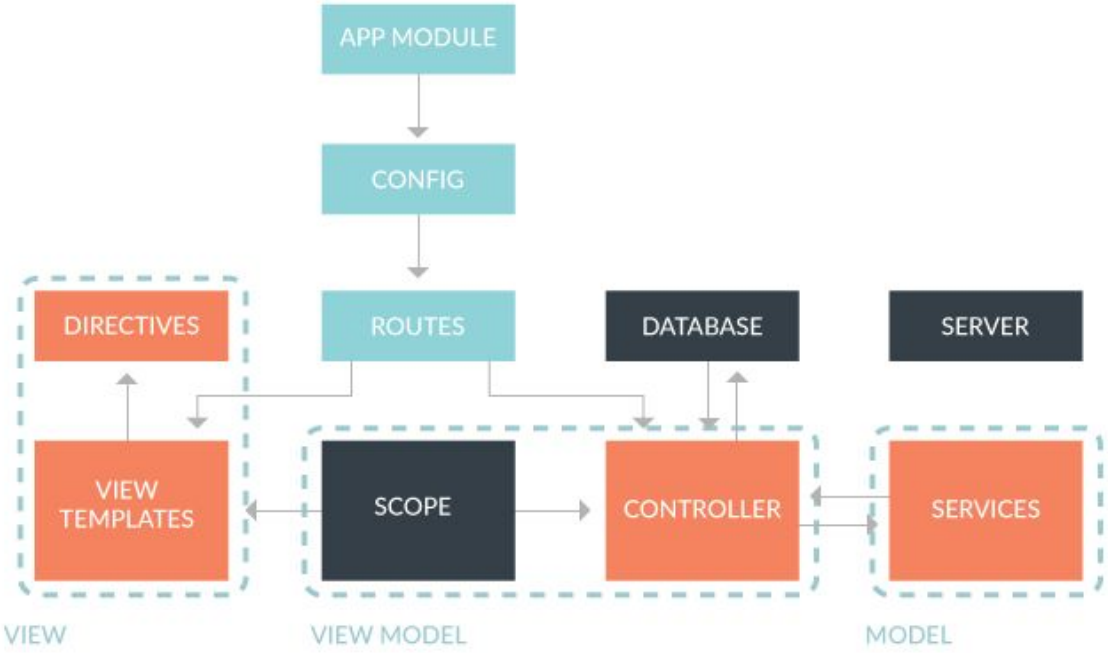


ANGULAR JS APPLICATION STRUCTURE

An angular application consists of

- **controllers** - Controller should be just an **interlayer** or the **'glue'** between model and view. In angular's case between the service layer and the view/template layer. Try to make controllers as **thin** as possible i.e. as little coding as possible. All the work should be done by the service layer and passed to the controller.
- **templates** - html files, have an associated controller which will define the data to be presented to the template for viewing in the browser. this can also be known as the **view** or presentation layer
- **directives** - UI components that can be included in templates or views as enhancements to existing html elements or new html elements that can have dynamic behaviour
- **services** - allow model data and business logic to be retrieved from some source (external server process) and passed on to controllers via dependency injection.

So how does the Angular Application structure map to our discussion on design patterns, particularly the MVVM pattern.



ANGULAR JS APPLICATION STRUCTURE (MVVM)