Super-simple Model-View-What-not? Framework!

Siddique Hameed
https://github.com/siddii

STL-JS Meetup

08/15/2013

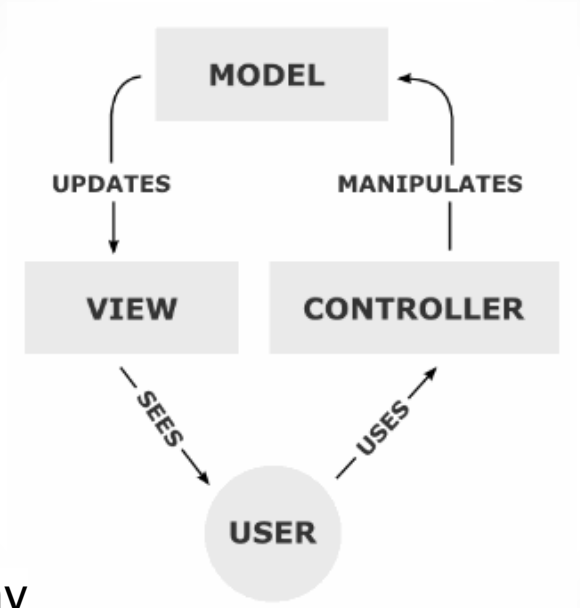
Highlights...

- Synchronized *Model*<->*View*<->*Controller* (MVC) *
- **D**ata binding. Powerful 2-way/bi-directional!
- **D**irectives/Declarative based
- **D**ependency Injection (DI). Don't confuse this with AMD style script loading like *requirejs* or *commonjs*
- Out-of-box *client-side* HTML templating
- CSP (*Content Security Policy*) support (using *ng-csp* directive)
- SPA (*Single Page Application*) support
- Designed with CRUD/RESTful apps in mind.
- TDD focused
- etc...

Think Outside the DOM!

Traditional MVC...

Source: http://en.wikipedia.org/wiki/Model-view-controller



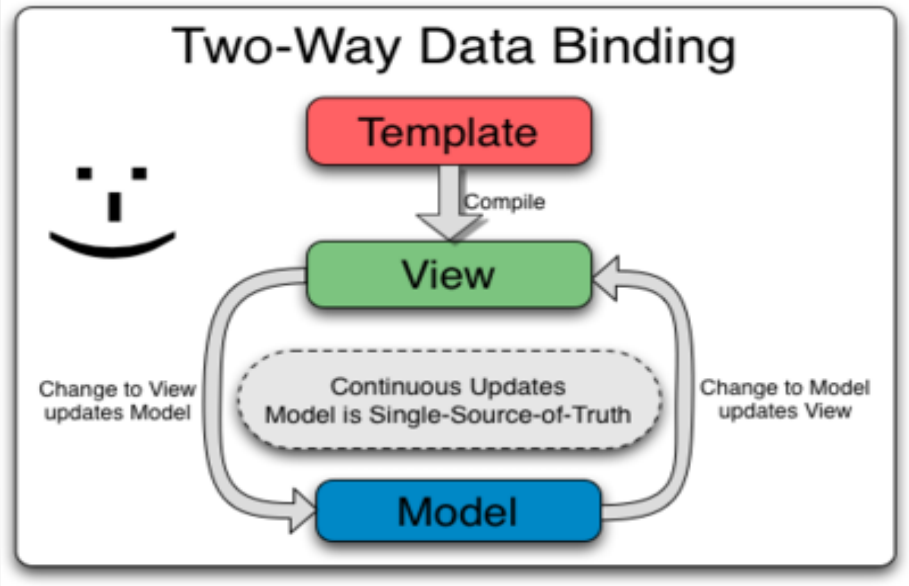
Dependency hierarchy

...

The View however, knows about the Model. It will poll the Model about the state, to know what to display. That way, the View can display something that is based on what the Model has done. But the View knows nothing about the Controller.

...

MVC in AngularJS



- Transforms static DOM to dynamic DOM
- Compiles directives & produce linking functions hence making the DOM live view of the model & vice-versa.

Some Key Concepts...

Directives are ways to...

- Extend HTML vocabulary by adding more syntax & semantics (*tags, attributes, classnames & even comments*)
- Transforms HTML markup to **D**omain **S**pecific **L**anguage (DSL)

  For ex. <*book isbn=""/*>, <*screenshot url=""*
/>
- Build reusable widgets/components

Model

- Represented as Javascript objects/properties attached to a Scope ($scope)

```
function GreetingsController($scope) {
      $scope.greetings = 'Hello';

      $scope.user = {firstName:'John',lastName:'Doe'};

      $scope.reverseGreetings = function (){

        return $scope.greetings.split('').reverse().join('');

      };

    }
```

```
<div>{{greetings}} {{user.firstName}}, {{user.lastName}}!</div>
```

View

- This is what the user sees and responds to
- Dynamic DOM or *live view* generated after compiling directives & applying templates, filters etc.
- Manages event handlers & delegates to controller & model internally(No more event handlers/callback recursions or chaining!)

Controller

- Glues or connects Model & View
- Initializes state of Scope
- Invoke services to perform reusable or externalized operations

Services

- Borrowed phrase from server-side technology
- Encapsulated *object* for commonly performed operations
- Wired up using DI injection
- Not necessarily used only for client/server interactions or async operations
- Examples include $http, $q, $window, $timeout etc.

Filters

- Helps formatting or filtering data
- Can also be used to limit or control what is displayed like in pagination etc.
- Unix's all time favorite *pipe(|)* expression to chain filter calls

Examples

```
{{amount | currency:"USD$"}}
```

```
myArray | orderBy:'timestamp':true | limitTo:10
```

Routing

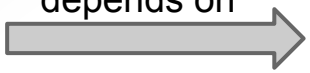
- SPA (Single Page Application) support
- Used for routing to multiple views (or Template files)
- Provides deep-linking URLs
- Bookmarking support

```
$routeProvider.when('/questions', {templateUrl: 'templates/questions.html', controller: QuestionsController});

$routeProvider.when('/questions/:questionNo', {templateUrl:'templates/questions.html', controller: QuestionsController});

$routeProvider.when('/results', {templateUrl:'templates/results.html', controller: ResultsController});
```

Why Dependency Injection?

```
function Driver{
        driveCar: function () {
        }
}
```

depends on



```
function Car{
        drive: function() {
        }
}
```

There are 3 ways a **Driver** can drive the **Car**

1.)
```
driveCar: function () {
        var car = new Car();
        car.drive();
}
```

2.)
```
var car = new Car();  //as global or shared state
driveCar: function () {
        car.drive();
}
```

3.)
```
driveCar: function (car) {     //dependency injected to the driver
        car.drive();
 }
```

TDD in Angular

- TDD in Client side JS is lot harder especially dealing with XHR & browser based components (DOM, document, window etc.)
- Strong emphasis on both Unit Testing & E2E (End-to-End testing)
- Based on Jasmine BDD
- Based on Karma (formerly called Testacular) Test runner
- Angular supplied Mock & Test harness classes

Are you sure, it's all good?

- Oh no, yet another framework to learn!
- Declarative/directive style may not be everyone's cup of coffee/tea/whatever :)
- Very comprehensive aka *structured framework*, so lot to absorb!
- Evolving documentation. But, its a very active community, doesn't seem hard to find answers online easily
- No UI widgets/skins/styling. Probably good that way?
- Watch for variables & methods with ‘$’ prefixes

May not be suited for...

- Game development
- Heavy screen or pixel based operations
- Anything involving excessive DOM manipulation
- Visualization (charts, graphs etc.)

Obviously, we can use other libraries on top or along with Angular.

OK, Show me the <code/>

Q&A

So, what next?

Checkout these projects/links...

- Google I/O 2013 - Design Decisions in AngularJS - http://www.youtube.com/watch?v=HCR7i5F5L8c
- https://github.com/siddii/angular-timer
- https://github.com/siddii/wikipedia-chrome-extension
- http://stephenplusplus.github.io/meangular/
- http://angular-ui.github.io/
- http://www.localytics.com/blog/2013/angularjs-at-localytics/

& Crack some <code/>...

Happy Coding :)