# Bruno Scopelliti

Husband, almost-dad.
Web Developer. Risiko Strategist.

@brunoscopelliti

**Ladies, if a man says he will fix it, he will.**

**There is no need to remind him every 6 months about it.**


Balance
A RESTful, CRUD web app
built on AngularJS

# AngularJS

## HTML enhanced for web apps!

- Framework for Single Page Application

  42KB min, **no jQuery dependency**

- AngularJS Core Principles

  Decouple DOM manipulation from app logic

  Decouple the client side of an app from the server side

  App testing is as important as app writing

# AngularJS

## Key features

- Modules + Dependency Injection
- MVC Design Pattern
- Two ways data bindings
- Enhanced HTML
- A lot of built-in services

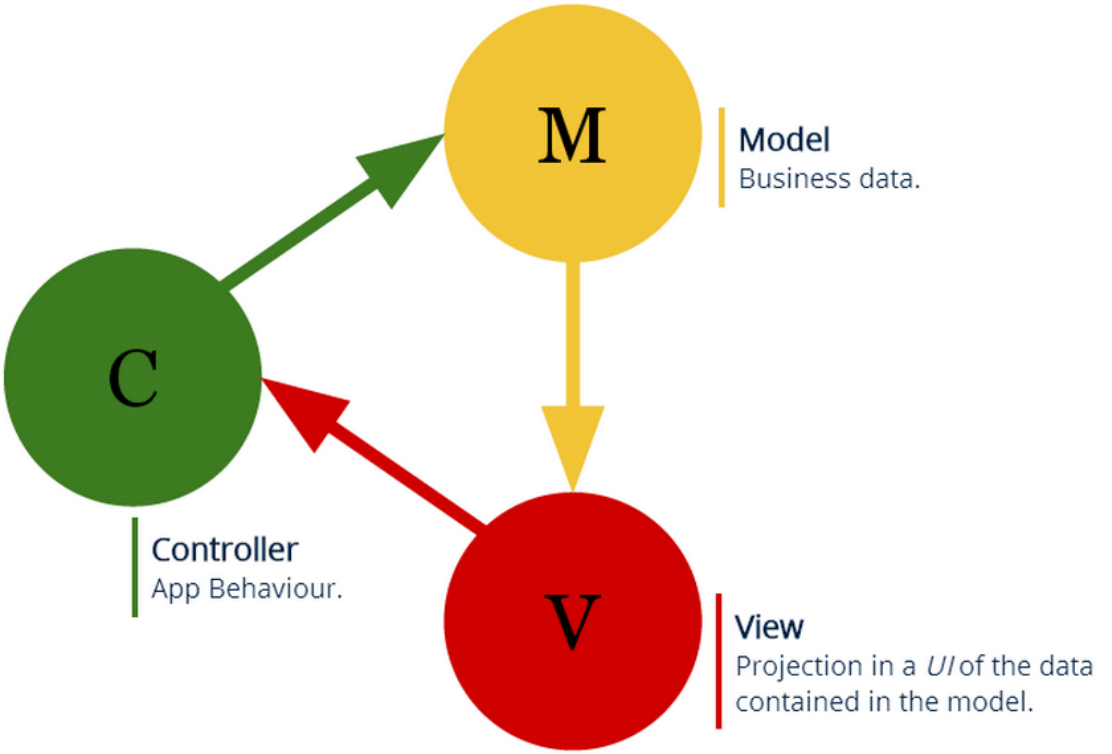
# Modules e DI

AngularJS apps are structured in **modules**.

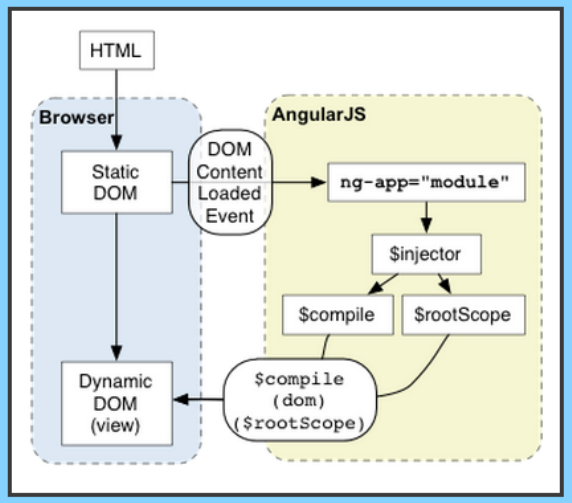
```
1 | // Create a new module without any dependencies
2 | var myModule = angular.module('myHelperModule', []);
```

A module can be dependent by one, or more others modules, and can contain controllers, services, directives, and/or filters.

```
1 | // Create a new module,
2 | // that is dependent on ngRoute, and myHelperModule
3 | var myApp = angular.module('myApp',
4 |     ['ngRoute', 'myHelperModule']);
```




# MVC Design Pattern



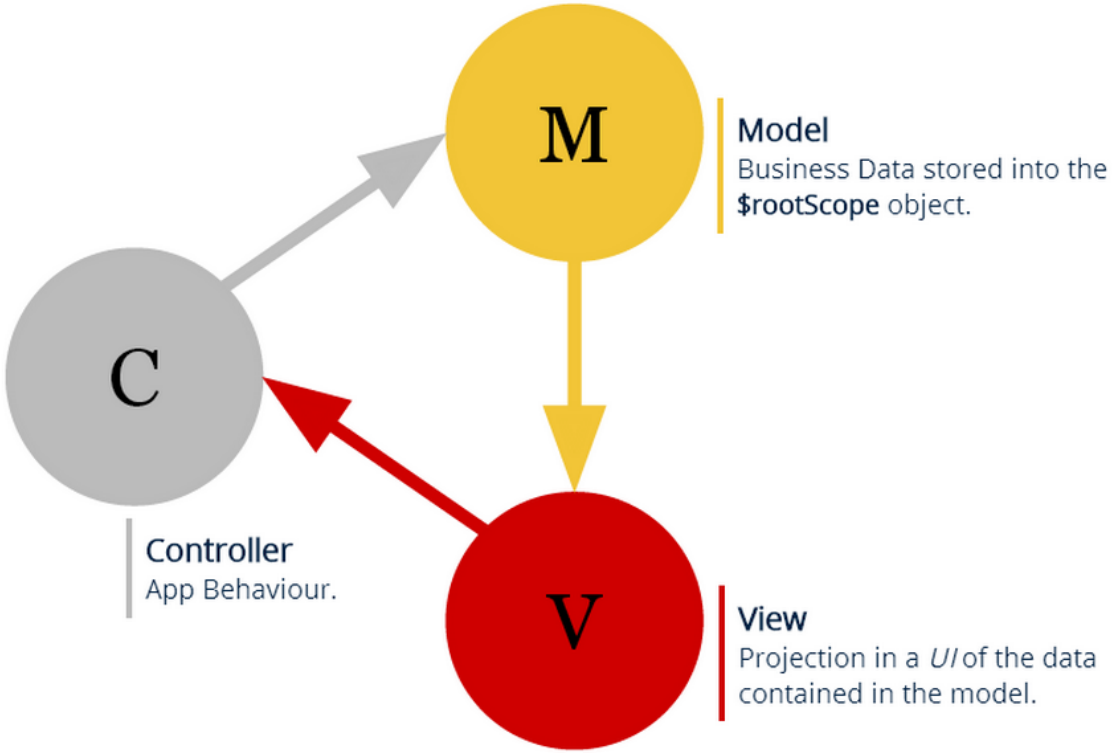

```html
<html ng-app="myApp">
<head>
    <!-- ... -->
</head>
<body>
    <h1>Hello {{who}}!</h1>
    <script src="js/angular.js"></script>
    <script src="js/hello-world-app.js"></script>
</body>
</html>
```

**Template**
An HTML document with extra markup: *directives*, and *expressions*.

```javascript
// Define the myApp module
var app = angular.module('myApp', []);

// Code which needs to run to kickstart the app
app.run(['$rootScope', function($rootScope) {

    // It is executed only after all of the service
    // have been configured and
    // the injector has been created
    $rootScope.who = 'world';

}]);
```

The **Model** exposes its data to **Controller** and **View** through the $scope object.

# Application Startup




# MVC Design Pattern

```html
<html ng-app="myApp">
<head> <!-- ... --> </head>
<body>
    <h1 ng-controller="myCtrl">Hello {{who}}!</h1>
    <script src="js/angular.js"></script>
    <script src="js/hello-world-app.js"></script>
</body>
</html>
```

The *ngController* directive attaches a controller class to the view.

```javascript
var app = angular.module('myApp', []);

// Register a new controller
app.controller('myCtrl', ['$scope', function($scope) {

    $scope.who = 'world';

}]);
```

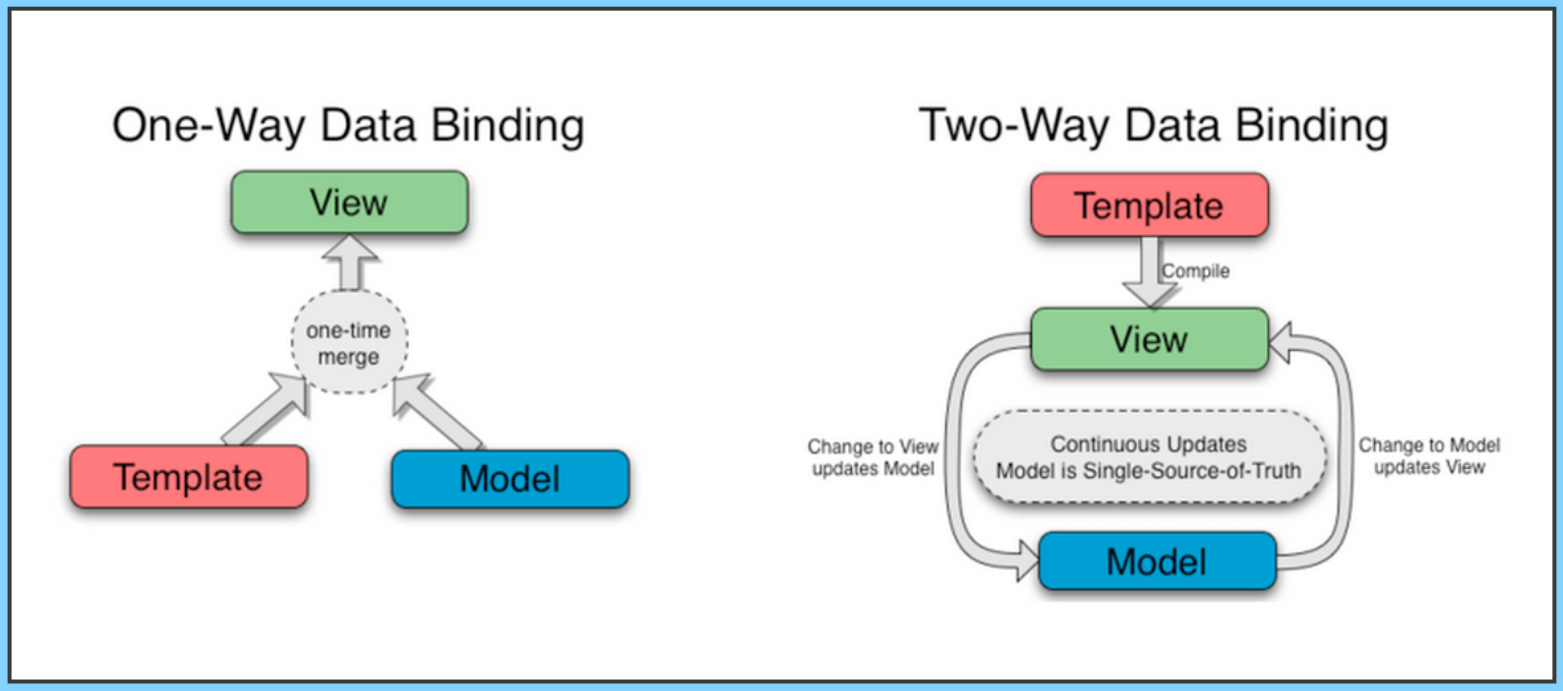
A new child $scope is available as injectable parameter to the controller constructor function.

# MVC Design Pattern

Toggle case!

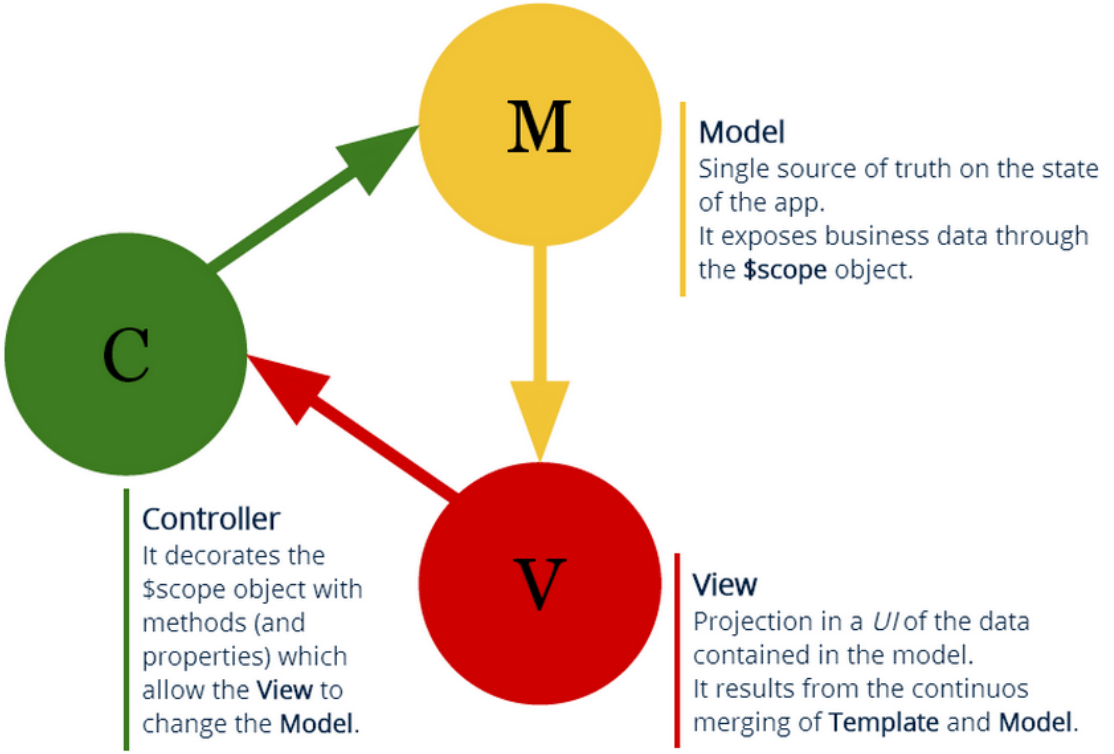
# Hello world!

Angu




Hello Angu!

# Two ways data binding




# MVC Design Pattern

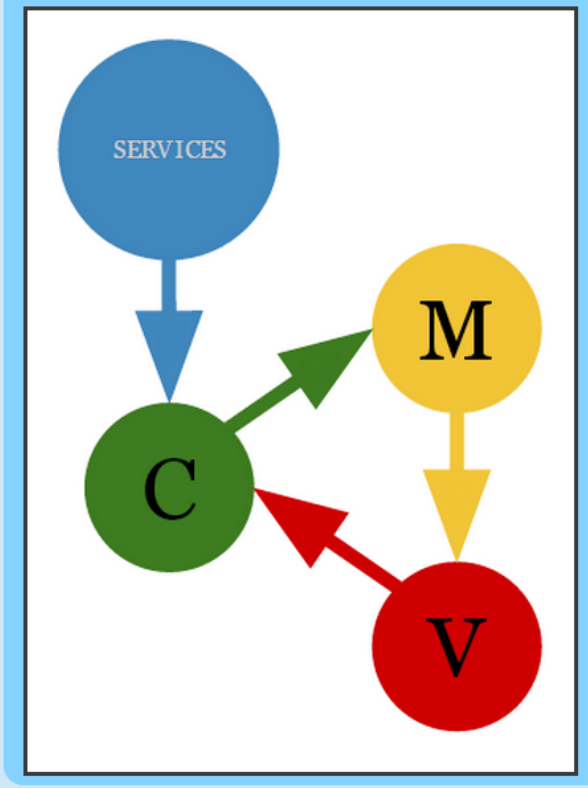
# Directives

## Enhanced HTML

> Directives are markers on DOM elements, which tell AngularJS's HTML compiler to attach a specified behavior to that DOM element or even transform the DOM element and its children.

```
<!-- ngClick -->
<span ng-click="doSomething();">Click here</span>

<!-- custom directive-->
<date-picker></date-picker>
```

# Services




- Services are *singleton objects*, responsables for specific tasks.
- Services can be injected into controllers via AngularJS DI.
- AngularJS provides a lot of built-in services
  *$http*, *$animate*, etc.
- It is also possible to define custom services.

This is a quite good moment to take a look at Balance.

# Balance

## Balance features

- Register a new account
- Access using account credentials

- Create a new flow
- Read saved flows
- Update data about a flow - *soon*
- Delete a flow

# $http vs ngResource

## Work with the XMLHttpRequest Object

- $http is a core AngularJS service

  It helps with XHR requests.

  It is based on the deferred/promise APIs exposed by the $q service.

- ngResource is a stand-alone module

  4KB min. It is built upon **$http**.

  The **$resource** service provides support for **RESTful** web services.

# Web Services

## The RESTful way

- Resources
- Interaction based on HTTP verbs, and status
  **POST**, **GET**, **UPDATE**, and **DELETE**, map the **CRUD** actions.
- Stateless communication
  Each request must contain all the information necessary to understand the request.
- HATEOAS
  Interconnected resource representations.

# Balance

## Balance features

- Register a new account
- Access using account credentials

- **C**reate a new flow
- **R**ead saved flows
- **U**pdate data about a flow - *soon*
- **D**elete a flow

# $resource

## Create the resource model

```
angular.module('balance')
.factory('flowService', ['$resource', function($resource) {

    // $resource(url[, paramDefaults][, actions]);

    // Returns an object with methods for
    // the default set of resource actions

    return $resource('./:user/flows/:year/:month/:day/:flowId',
        {
            user: '@user',
            year: '@year',
            month: '@month',
            day: '@day',
            flowId: '@flowId'
        });

}]);
```

# Web Services

```
RewriteEngine On
RewriteBase /
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d

## Create
## Rewrite POST :user/flows
## Create a new flow
RewriteCond %{REQUEST_METHOD} POST
RewriteRule flows$ /balance/server/api/create-flow.php [L]

## Read
## Rewrite GET :user/flows/year/:year
## Get the flows occurred during the specified year
RewriteCond %{REQUEST_METHOD} GET
RewriteRule flows/year/([0-9]+)$ /balance/server/api/get-year-flows.php [L]

## Update
## Rewrite PUT :user/flows/:flowId
## Put the flow with id :flowId
RewriteCond %{REQUEST_METHOD} PUT
```

# $resource - Use the resource object

```
angular.module('balance').controller('statsController', ['$scope', 'accountService', 'flowService',
    function($scope, Account, Flow) {

        var User = Account.user || {};

        $scope.type = false;
        $scope.amount = $scope.tags = $scope.date = '';

        $scope.record = function() {

            var xhr,
                _flow = { user: User.name, type: $scope.type, amount: $scope.amount, tags: $scope.tags, date: $scope.date };

            xhr = Flow.save({ user: $scope.name }, _flow).$promise;
            xhr.then(function(data) {

                /* ... */

            });

        }

}]);
```

# Protected Web Services

## Authentication to a RESTful web service

- Stateless web services

  Each requests have to carry with them the state of the client.

  For this reason the state of the client is stored as an header of the request itself.

```
$http.post('/tokenizer', { username: user, password: pw })
.success(function (data, status, headers, config) {

    if (data.status) {
        $http.defaults.headers.common['auth'] = data.token;
    }

});
```

# Automatic Test

## Grunt, Karma, and Jasmine

**Grunt**
Task runner

**Karma**
Spectacular Test Runner for Javascript

**Jasmine**
BDD testing framework

# Modules

```
describe("The balance module", function() {

  var module, deps;

  beforeEach(function() {
    module = angular.module("balance");
    deps = module.value('balance').requires;
  });

  it("should be registered", function() {
    expect(module).not.toEqual(null);
  });

  it("should have ngResource as a dependency", function() {
    expect(deps.indexOf('ngResource') >= 0).toEqual(true);
  });

});
```

# Unit Test

# Test the $scope.record() method

```
describe("statsController", function() {

  var rootScope, scope, ctrl, F;

  beforeEach(module('balance'));

  beforeEach(inject(['$controller', '$rootScope', 'flowService', function($controller, $rootScope, Flow) {
    rootScope = $rootScope;
    scope = rootScope.$new();
    F = Flow;
    ctrl = $controller('statsController', { $scope: scope });
  }]));

  it("should be able to create a new flow", inject(['$q', function($q) {

    var deferred;

    angular.extend(scope, { last30days: [], type: true, amount: 50.00, tags: 'test', date: new Date() });

    // simulate resolving a promise
    deferred = $q.defer();
    deferred.resolve({status: true, flow: {userId: 1, signedDate: new Date(), amount: 50.00, type: true, tags: 'test'}});
```

# Unit Test

## Friendly reminder

Run the unit tests

> grunt unit

# $resource

## Define custom actions

```
angular.module('balance')
.factory('flowService', ['$resource', function($resource) {

    // $resource(url[, paramDefaults][, actions]);

    return $resource('./:user/flows/:last/:year/:month/:day/:flowId',
        { user: '@user', /* ... */ flowId: '@flowId' },
        {
            'newAction': {
                method: '[POST|GET|PUT|DELETE]',
                params: {},
                isArray: false
            }, /* ... */
        }
    );

}]);
```

## Test, Fail, Fix...

Iterate


That's all Folks!