AngularJS

The Super heroic JS MVW Framework

History

- Original developed by Misko Hevery and Adam Abrons in 2009.
- Adam Abrons officially abandoned the project.
- Hevery began to work with Google.
- Maintained by Google ever since.

Wait, but don't we have enough JS framework already ?

What is AngularJS

- Client-side JS Framework for SPA.
- MVC, MVVM, MV*, MVW Framework.
- Your application UI & UX on Steroids.

Why AngularJS ?

- Proper architectured and maintainable web applications.
- Decouple DOM manipulation from application logic.
- Decouple the client side of an app from the server side.
- Teach the browser new syntax.
- Two-way data-binding helps us for the automatic synchronization between models and views.


I CAN'T MOVE
ON FROM JQUERY..
BE A MAN!!
imgflip.com

NG-APP

```
<!doctype html>
<html ng-app >
    <head>
        <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.11/angular.mi
n.js"></script>
    </head>
    <body></body>
</html>
```

Boostrapping

Automatic initialization

```
<!-- index.html -->
<tag ng-app="App"></tag>


<!-- app.js -->
(function() {
    angular.module('App', []);
})();
```

Manual initialization

```
<!-- app.js -->
(function() {
    angular.module('App', []);
    angular.bootstrap(document, ['App']);
})();
```

Modularity

The framework is born to be modular.

Each ng-app is a module which depends on another module, whether its a service, factory, directive or the app config itself.

Module Declaration

```
angular
    .module('BeerModule', [])
    .service('BeerService', function() {
        this.getBeer = function() {
            return "Beer !";
        }
    });
```

Module with dependencies

```
angular
    .module('App', ['BeerModule'])
    .controller('someController', ['$scope', 'BeerService',
    function($scope, BeerService) {
        $scope.giveBeer = function() {
            alert( BeerService.getBeer() );
        }
    }]);
```

```
<div ng-app="App" data-ng-controller="someController">
    <button ng-click="giveBeer()">Bring me a beer !</button>
</div>
```

Bring me a beer !

Key Features

- Two-way data-binding
- Filter
- Controller
- Service & Factory
- Dependency Injection
- Directive
- Routing & Partial Views

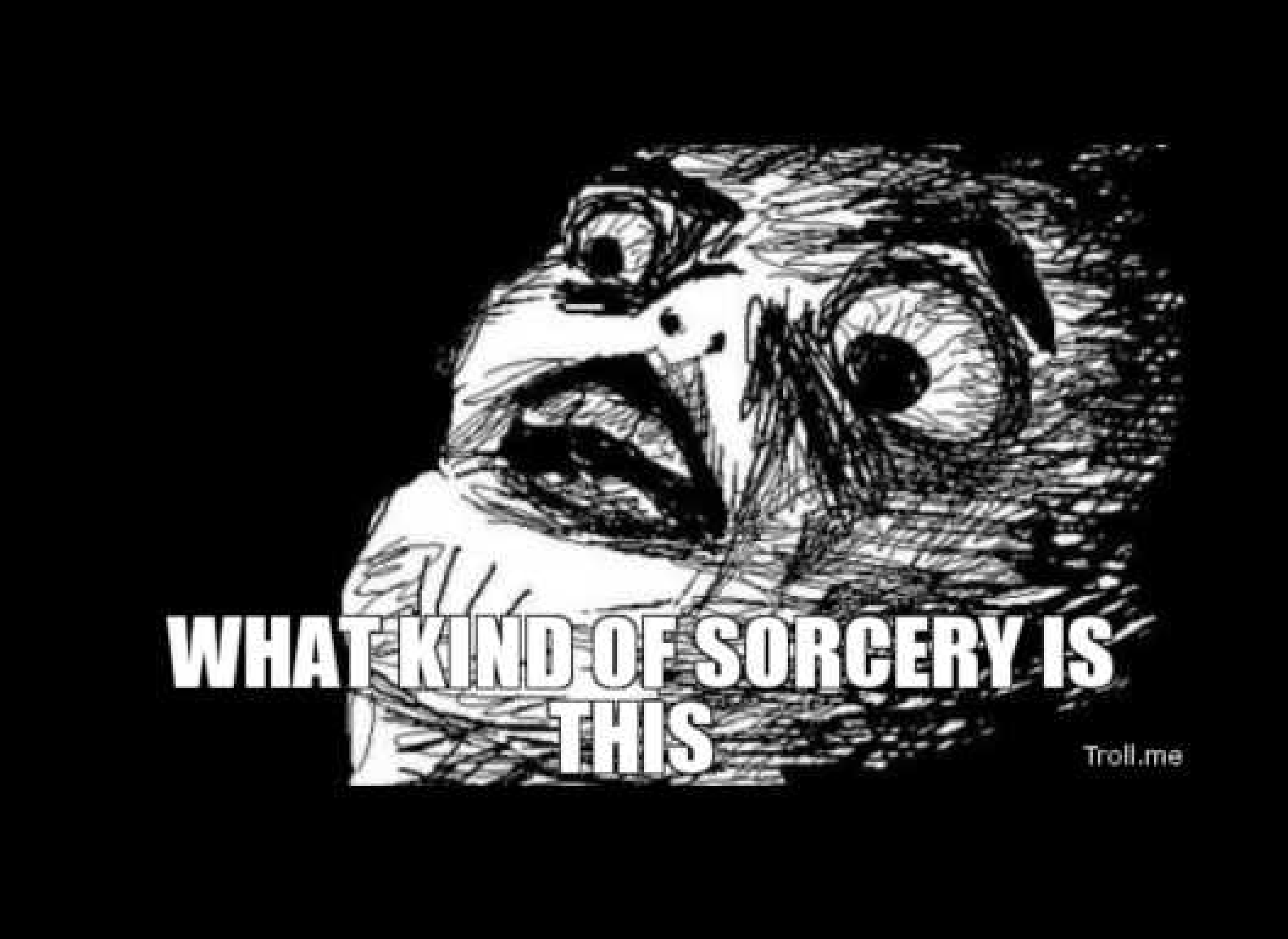
Two-way data-binding

Automatic synchronization between models and views.

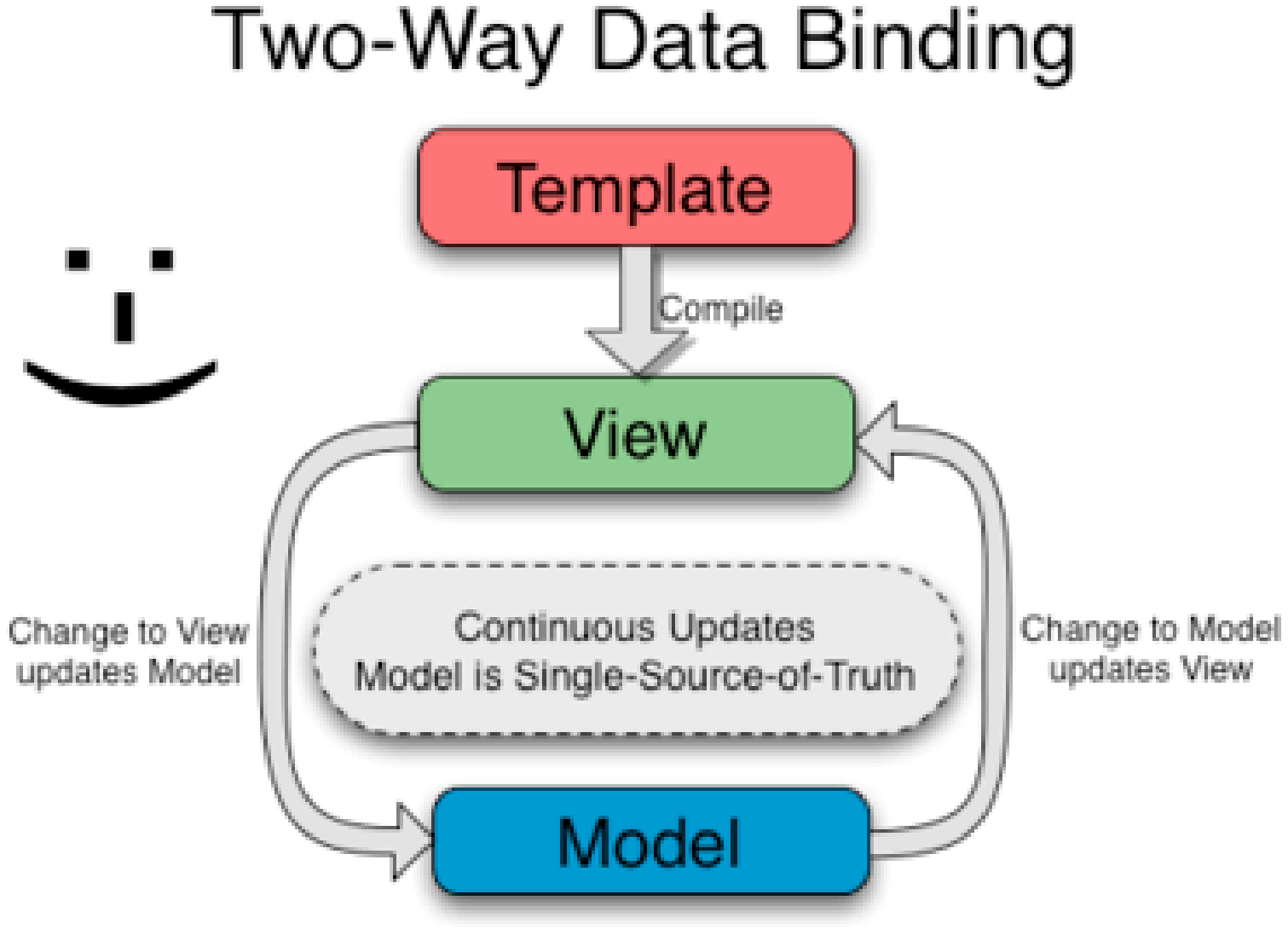
Example

```
<div>
    <input type="text" data-ng-model="name" placeholder="Your name please ?">
    <h5>Hello {{ name }}!</h5>
</div>
```

Hello !


WHAT KIND OF SORCERY IS THIS
Troll.me

How it works ?


Two-Way Data Binding
Template
Compile
View
Change to View
updates Model
Continuous Updates
Model is Single-Source-of-Truth
Change to Model
updates View
Model

Filter

Manipulate how your expressions are displayed.

```
<div ng-init="heroes = [
    { name: 'Axe', type: 'str'},
    { name: 'Sven', type: 'str'}
    { name: 'Drow Ranger', type: 'agi'},
    { name: 'Lich', type: 'int'}
]">
</div>
<div
    <input type="text" ng-model="searchText.type">
    <ul>
        <li ng-repeat="hero in heroes | filter:searchText">{{ hero.name }}</li>
    </ul>
</div>
```

- Axe
- Sven
- Drow Ranger
- Lich

Controller

Main logic handler for your application, watch model changes or register services, forget about DOM manipulation.

Declaration

```
<!-- Example controller.js -->
App.controller('ctrlName', function ($scope, [, dependencies]) {
    // your logic here
});


<!-- Example view.html -->
<div ng-controller="ctrlName">
    // do your stuff here
</div>
```

Example

```
App.controller('ctrlName', function ($scope) {
    $scope.count = 1;
    $scope.message = '';

    $scope.moreBeer = function() {
        $scope.count += 1;

        if ( $scope.count >= 5 && $scope.count < 10)
            $scope.message ="Too much beer !";

        if ( $scope.count >= 10 )
            $scope.message = "STAAAAHHHP !!!!";
    }
});


<div ng-controller="ctrlName">
    Beer count : {{ count }}
    <p class="small">{{ message }}</p>
    <button ng-click="moreBeer()">I want more !</button>
</div>
```

Beer count : 1

I want more !

Factory

allow us to retrieving values or calling another service and return as an object.

"When you're using a Factory you create an object, add properties to it, then return that same object. When you pass this service into your controller, those properties on the object will now be available in that controller through your factory."

Factory Declaration

```
App.factory('BeerFactory', function() {
    return "I'm from BeerFactory";
});
```

...or expose something a bit more

```
App.factory('BeerFactory', function() {
    return {
        name  : "beer name",
        price : "1000"
    }
});
```

Service

quite similar to Factory, except it’s instantiated with the ‘new’ keyword and returned as the service itself a.k.a `this`.

Service Declaration

```
App.service('ServiceName', function() {
    this.method = function() {
        return "something";
    }
});
```

another example with multiple function inside service

```
App.service('ServiceName', function() {
    this.method = function() {
        return "something";
    }
    this.anotherMethod = function() {
        return "something from another method";
    }
});
```

A service is a singleton to an object.

Example

```
App.service('BeerService', function() {
    this.sayBeer = function() {
        return "Beer!";
    }
    this.sayHello = function() {
        return "Hello";
    }
    this.sayBoth = function() {
        return this.sayHello() + ' ' + this.sayBeer(); // returns "Hello Beer!";
    }
});


App.controller('serviceController', function($scope, BeerService) {
    $scope.sayBeer = function() {
        alert( BeerService.sayBeer() );
    }
    $scope.sayHello = function() {
        alert( BeerService.sayHello() );
    }

    $scope.sayBoth = function() {
        alert( BeerService.sayBoth() );
    }
});


<button class="btn" ng-click="sayBeer();">Say Hello</button>
<button class="btn" ng-click="sayHello();">Say Beer!</button>
<button class="btn" ng-click="sayBoth();">Say both</button>
```

Say Hello Say Beer! Say both

Dependency Injection

The Angular injector subsystem is in charge of creating components, resolving their dependencies, and providing them to other components as requested.

Dependency Injection Example

```
App.controller('demoCtrl', function($scope, BeerService) {
    // $scope, BeerService now accessible here.
});
```

Dude, minification breaks my application, WTF !?

---

we need angular dependency annotation, so that the injector knows what services to inject into the function.

```
App.controller('demoCtrl', ['$scope', 'BeerService',
function(a, b) {
    // a is $scope
    // and b is BeerService since we already annotated them.
}]);
```

Here we pass an array whose elements consist of a list of strings (the names of the dependencies) followed by the function itself.

Directive

Agular directives are markers on a DOM element which attach a special behavior to it.

Directive

At a high level, directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS's HTML compiler to attach a specified behavior to that DOM element or even transform the DOM element and its children.

Angular comes with a set of these directives built-in, like:

- ngModel
- ngView
- ngBind
- And many more...

Directive Types

All of the Angular-provided directives match attribute name, tag name, comments, or class name. The following demonstrates the various ways a directive (beerDir in this case) can be referenced from within a template:

```
<beer-dir></beer-dir>
<span data-beer-dir="exp"></span>
<!-- directive: beer-dir exp -->
<span class="beer-dir: exp;"></span>
```

See how Angular normalize the directive name? The normalization process is as follows:

1. Strip x- and data- from the front of the element/attributes.
2. Convert the :, -, or _-delimited name to camelCase.

Directive Example

```
App.directive('beerDir', function() {
  return {
    template: 'This beer is not so bad ;)'
  };
});


<beer-dir></beer-dir>
```

This beer is not so bad ;)

More Directive Example

```
App.controller('beerController', ['$scope', function($scope) {
  $scope.myBeerCollection = [
    { name: 'Carlsberg', grade: 'good'},
    { name: 'Bintang', grade: 'great'},
    { name: 'Guinness', grade: 'not my taste'}];
}])
.directive('myBeer', function() {
  return {
    controller: 'beerController',
    restrict: 'E', // 'A' = attribute, 'E' = element, 'C' = class
    templateUrl: 'includes/my-beer.html'
  };
});
```

includes/my-beer.html

```
<ul class="list">
    <li ng-repeat="beer in myBeerCollection">
        {{ beer.name }} is {{ beer.grade }}
    </li>
</ul>
```

```
<my-beer></my-beer>
```

- Carlsberg is good
- Bintang is great
- Guinness is not my taste

Routing & Partial Views

Angular let you slice your view template into a smaller and managable piece.

Routing & Partial Views Example

With AngularJS we can turn the index.html template into what we call a "layout template". This is a template that is common for all views in Angular application. Other "partial templates" are then included into this layout template depending on the current "route" — the view that is currently displayed to the user.

app.js

```
App.config(['$routeProvider', function($routeProvider){
    $routeProvider
    .when('/', {
        templateUrl: 'views/home.html'
    })
    .when('/todo', {
        templateUrl: 'views/main.html',
        controller: 'MainCtrl'
    })
    .when('/about', {
        templateUrl: 'views/about.html'
    })
    .otherwise({
        redirectTo: '/'
    });
}])
.controller('MainCtrl', function ($scope, localStorageService) {

    var todosInStore = localStorageService.get('todos');

    $scope.todos = todosInStore || [];
```

Routing & Partial Views Example

views/home.html

```
<div class="jumbotron"><h1>Hello World!</h1></div>
```

views/about.html

```
<div class="jumbotron"><h1>This is about beer.</h1></div>
```

views/main.html

```
<div class="container">
  <h2>My todos</h2>

  <!-- Todos input -->
  <form role="form" ng-submit="addTodo()">
    <div class="row">
      <div class="input-group">
        <input type="text" ng-model="todo" placeholder="What needs to be done?" clas
s="form-control">
        <span class="input-group-btn">
          <input type="submit" class="btn btn-primary" value="Add">
        </span>
      </div>
    </div>
  </form>
  <p></p>

  <!-- Todos list -->
  <div ui-sortable="" ng-model="todos">
    <p class="input-group" ng-repeat="todo in todos" style="padding:5px 10px; cursor:
move;">
```

Routing & Partial Views Example

index.html

```
<div class="collapse navbar-collapse" id="js-navbar-collapse">
    <ul class="nav navbar-nav">
        <li class="active"><a href="#/">Home</a></li>
        <li><a ng-href="#/todo">To-do</a></li>
        <li><a ng-href="#/about">About</a></li>
    </ul>
</div>
<ng-view></ng-view>
```

See the example. (../v.2/app/#/)


WHAT IF I TOLD YOU, THIS PRESENTATION WAS OVER?
THANKYOU FOR WATCHING


THE PRESENTATION IS DONE
Y U NO APPLAUSE?!
quickmeme.com