About AngularJS

Let’s talk about what it is

But first…

Who am I?

Anders Skarby

Indepedant Software Developer / Consultant
My own co-owned company - Innotech Solutions ApS
Tweet at @askarby - E-mail me at abs@inno-tech.dk

Tonights Plan

- 17:00’ish - Presentation and Live Coding
- 18:00’ish - Sandwishes and stretching your legs
- 18:15’ish - Finishing the presentation, and questions
- 19:00’ish - Goodbye (and thanks for all the fish)

What is AngularJS

Does anyone have any ideas?

Modular

But not at runtime

MVC

Model View Controller

Dependency Injection

Ask for dependencies, don’t create them

Testability

We don’t get payed to test, we get payed for quality

Productivity

It’s what it’s all about!

Controllers and Binding

In AngularJS

Three Terms

That all you need know to comprehend AngularJS

Model

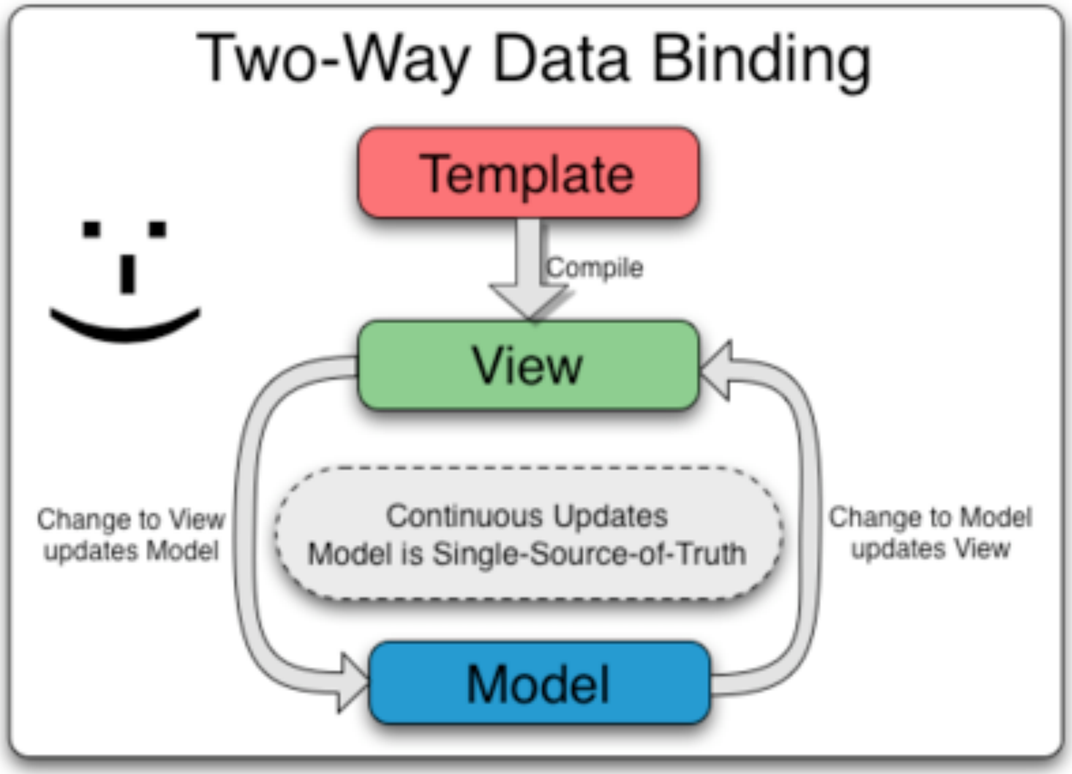
View

In AngularJS it’s called provided via a Template

Controller

Scope

The glue between the Model / Controller and the View



Template, View and Model

How they work together to provide Data binding

Data Binding Example

It’s easier than you think

Controller

```
function MyController($scope) {
    $scope.name = 'Anders';
}
```

Template

```
<div>{{name}}</div>
```

Services

‘cause Controllers won’t suffice for everything

Various kinds

All singletons, but their constructs are different

Factory

You instantiate what you want - yourself.

```
module.factory('user', function(){
    var svc = {};
    svc.create = function () { … };
    svc.remove = function () { … };
    return svc;
});
```

Service

You provide behavior for the construct

```
module.service('user', function(){
    this.create = function () { … };
    this.remove = function () { … };
});
```

Provider

A configurable factory

```
module.provider('user', function() {
    var replicateToBackend = false;

    this.setReplicateToBackend = function () { ... };

    this.$get = function () {
        ... same as factory
    };
});
```

Honerable mentions

we won’t forget about values and constants

Singletons

They all create a single instance that is shared across all injections to eg. Controllers

Filters

A badly named construct

Able to modify content

Build in filters

- filter
- currency
- number
- date
- json
- lowercase
- uppercase
- limitTo
- orderBy

Easy to use

```
<div>{{name | uppercase}}</div>
```

They also take arguments

```
<ul>
    <li ng-repeat="name in names | filter: 'a'"></li>
</ul>
```

Custom Filters

```
module.filter('reverseArray', function () {
    return function (input, limit) {
        input = angular.copy(input);
        if (angular.isArray(input)) {
            input.reverse();
            if (limit) {
                input = input.slice(0, limit);
            }
        }
        return input;
    };
});
```

Directives

Adding functionality to markup

Everything is a directive

We’ve seen a few already

Examples we've seen

- ngApp
- ngBind - seen as {{ … }}
- ngModel
- ngRepeat

Create your own

Leads to reusable components

Naming convention

HTML is case-insensitive, JavaScript is not

Example

Directive

```
module.directive('gnHello', function () {
    return {
        restrict: 'E',
        scope: {
            name: '=',
        },
        template: '<div>Hello {{name}}</div>'
    };
});
```

Usage

```
<div>
    <input ng-model="firstname">
    <gn-hello name="firstname"></gn-hello>
</div>
```

Testing

A way to improve the quality of your delivery

Two kinds of tests

Well, that we’re going to talk about

Unit tests

- They test a single unit of code
- Are fast (nano - milliseconds to execute)
- Easy to write
- Mocks are used to separate from dependencies

Unit tests

- They are executed (in the AngularJS world) through Karma (a test runner)
- Jasmine is the default test / mocking framework (spec based)
    - It can be replaced with other frameworks, such as CucumberJS, Mocha, etc.
- Should be run repeatedly (through grunt / gulp) on every code change

Unit test example

```
describe('PasswordController', function() {
  beforeEach(module('app'));

  var $controller;

  beforeEach(inject(function(_$controller_){
    // The injector unwraps the underscores (_) from around the parameter names when matching
    $controller = _$controller_;
  }));

  describe('$scope.grade', function() {
    it('sets the strength to "strong" if the password length is >8 chars', function() {
      var $scope = {};
      var controller = $controller('PasswordController', { $scope: $scope });
      $scope.password = 'longerthaneightchars';
      $scope.grade();
      expect($scope.strength).toEqual('strong');
    });
  });
});
```

End-to-end tests

- Test the entire application
- From the browser, to the backend
- Slower (seconds - minutes to execute)
- Behaviour is tested
- Must have a full setup, cannot mock
- Need a predictable data set / setup

End-to-end tests

- They are executed (in the AngularJS world) through the Protractor test framework.
- Executed in real browsers
- Using web driver
- Has extended selectors to select based on AngularJS constructs

End-to-end test example

```
describe('angularjs homepage todo list', function() {
  it('should add a todo', function() {
    browser.get('http://www.angularjs.org');

    element(by.model('todoText')).sendKeys('write a protractor test');
    element(by.css('[value="add"]')).click();

    var todoList = element.all(by.repeater('todo in todos'));
    expect(todoList.count()).toEqual(3);
    expect(todoList.get(2).getText()).toEqual('write a protractor test');
  });
});
```