AngularJS Introduction

Objectives

- What is AngularJS?
- Bootstrapping an AngularJS application
- The MVC of AngularJS
- Standard directives
- Dependency injection
- Services
- Routing and Single Page Applications

What is AngularJS?

- AngularJS is an MVC framework for browser based applications
  - Open source and originally developed at Google
  - MIT license
- The clean architecture has attracted a large following quickly
  - Version 1.0 was released in June 2012
- The goal is building CRUD style business applications
  - Not as suitable for games etc
- Use declarative programming for UI and imperative programming for the logic
  - The application is wired together in a declarative way
- Supports modern desktop and mobile browsers
  - Internet Explorer version 8 and above

Key features

- Model View Controller architecture
  - A well known and proven architecture
- Declarative two way data binding
  - Automatically synchronizes values between Model and View
- Dynamic templates
  - Makes it very easy to update the user interface
- Dependency injections
  - Code dependencies are automatically injected where needed
- Extends HTML with directives
  - Lots of powerful standard directives or create your own
- Build with testing in mind
  - Makes it much easier to unit test different parts

Bootstrapping an AngularJS application

- Automatic bootstrapping
  - Add a reference to AngularJS
  - Add the ngApp attribute

```
<!DOCTYPE html>
<html>
    <head>
        <title>A minimal AngularJS application</title>
    </head>
    <body ng-app>
        <h1>A minimal AngularJS application</h1>
        <script src="Scripts/angular.js"></script>
    </body>
</html>
```

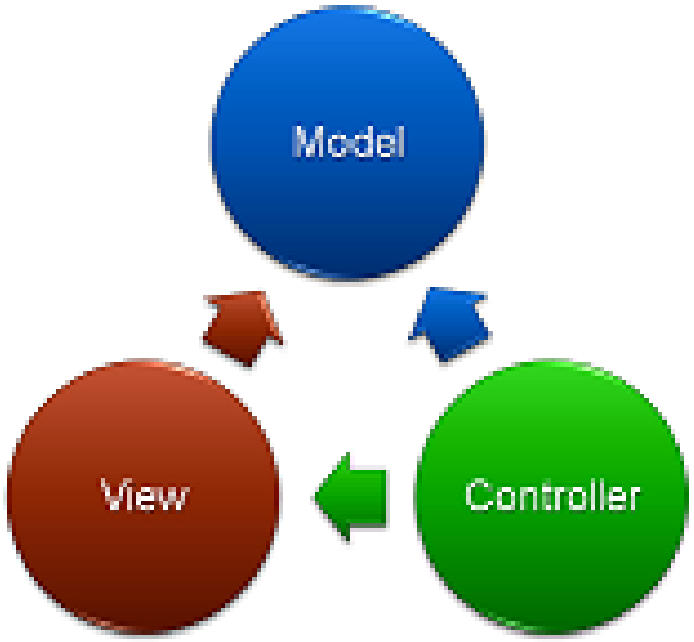
Bootstrapping an AngularJS application

- Manual bootstrapping is also possible
  - Gives you more control
- Not recommended unless required
  - For example when using AMD/RequireJS

```
<!DOCTYPE html>
<html>
    <head>
        <title>A minimal AngularJS application</title>
    </head>
    <body>
        <h1>A minimal AngularJS application</h1>
        <script src="Scripts/angular.js"></script>
        <script>
            angular.element(document).ready(function () {
                angular.bootstrap(document, []);
            });
        </script>
    </body>
</html>
```

The MVC of AngularJS

- **Provides a clear separation of concerns**



The Controller and Model

- Controller
  - Glues the view and the model together
  - Provides additional functionality
  - Uses additional services for reusable logic
- Model
  - The business data
  - Exposed to the view through the $scope

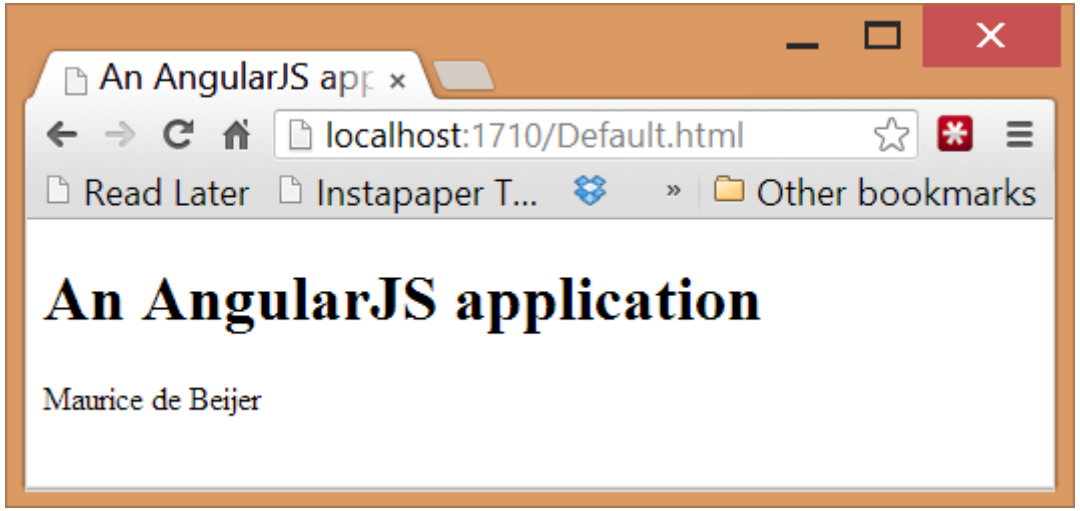
```
function SimpleCtrl($scope) {

    $scope.person = {
        firstName: "Maurice",
        lastName: "de Beijer"
    };
}
```


The View

- View
  - The user interface layer
  - Data binds to the model
  - Calls functions on the controller
  - Use declarative directives for reusable code

```
<!DOCTYPE html>
<html>
    <head>
        <title>An AngularJS application</title>
    </head>
    <body ng-app ng-controller="SimpleCtrl">
        <h1>An AngularJS application</h1>
        {{person.firstName}} {{person.lastName}}
        <script src="Scripts/angular.js"></script>
        <script src="App/SimpleCtrl.js"></script>
    </body>
</html>
```

The result in the browser



Standard directives

- Directives allow you to enhance HTML with new capabilities
  - Start using HTML as a domain specific language
- AngularJS comes with a long list of standard directives
  - ngApp
  - ngBind
  - ngModel
  - ngRepeat
  - ngClick
  - ngDisable
  - ngHide/ngShow
  - ngView
  - …
- Create your own as needed

Using directives in HTML

- Directives are named using camel case notation
  - For example: ngApp
- The ng is a prefix for AngularJS itself
  - Use another in custom directives
- Directives are invoked by splitting the name using :, -, or _
  - And optionally adding a x- or data- prefix
- Possible options for ngApp:
  - ng-app, ng:app, ng_app, x-ng-app or data-ng-app
- Directives can be added using elements, attributes, classes or in comments
  - The definition of a directive can limit its usage

Standard directives - ngBind

- Display some data in the HTML output
  - Similar to using {{expression}}

```
<!DOCTYPE html>
<html>
    <body ng-app ng-controller="SimpleCtrl">
        <h1>An AngularJS application</h1>
        {{person.firstName}} {{person.lastName}}
        <br />
        <span ng-bind="person.firstName"></span>
        <span ng-bind="person.lastName"></span>

        <script src="Scripts/angular.js"></script>
        <script src="App/SimpleCtrl.js"></script>
    </body>
</html>
```

Standard directives - ngBind

- Changes to the model are automatically refreshed
  - Provided they are made in the normal AngularJS event loop
- Changes outside the AngularJS event loop require extra work
  - setTimeout(), AJAX, DOM events etc
- Notify AngularJS using $scope.$apply()

```
function SimpleCtrl($scope) {
    setInterval(function () {
        $scope.$apply(function () {
            $scope.now = new Date().toLocaleTimeString();
        });
    }, 1000);
}
```


Standard directives - ngModel

- Two way data binding between view and model
  - Bindings are live
- Don't bind to the $scope directly!
  - Always bind to a property of a child object

```
<html>
    <body ng-app ng-controller="SimpleCtrl">
        <h1>An AngularJS application</h1>
        {{person.firstName}} {{person.lastName}}
        <br />
        First: <input type="text" ng-model="person.firstName"/>
        <br />
        Last: <input type="text" ng-model="person.lastName"/>

        <script src="Scripts/angular.js"></script>
        <script src="App/SimpleCtrl.js"></script>
    </body>
</html>
```

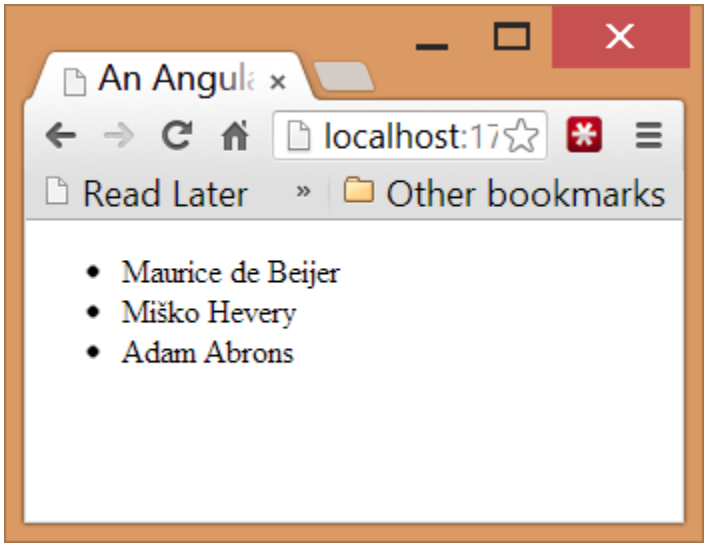
Standard directives - ngRepeat

- Repeats the UI template for each element in a collection
- A new linked scope is created for each item!

```
<!DOCTYPE html>
<html>
    <head>
        <title>An AngularJS application</title>
    </head>
    <body ng-app ng-controller="SimpleCtrl">
        <ul>
            <li ng-repeat="person in people">
            {{person.firstName}} {{person.lastName}}</li>
        </ul>

        <script src="Scripts/angular.js"></script>
        <script src="App/SimpleCtrl.js"></script>
    </body>
</html>
```

Standard directives - ngRepeat

```
function SimpleCtrl($scope) {
    $scope.people = [{
        firstName: "Maurice",
        lastName: "de Beijer"
    }, {
        firstName: "Miško",
        lastName: "Hevery"
    }, {
        firstName: "Adam",
        lastName: "Abrons"
    }];
}
```



Standard directives - ngShow

- Only show the content when the expression is true
  - ngHide, ngEnable and ngDisable are similar

```
<!DOCTYPE html>
<html>
<body ng-app ng-controller="SimpleCtrl">
    Country: <input type="text" ng-model="person.country" />
    <br />
    <div ng-show="person.country == 'USA'">
        State: <input type="text" ng-model="person.state" />
    </div>

    <script src="Scripts/angular.js"></script>
    <script src="App/SimpleCtrl.js"></script>
</body>
</html>
```

Filters

- Filters can be used to transform an expression
- Some work on simple values
  - {{now | date:'fullDate' }} => Saturday, February 8, 2014
  - {{123.45 | currency }} => $123.45
  - {{ object | json }}
- Others work on collections
  - Filter
  - orderBy
  - limitTo

```
ng-repeat="movie in movies |
           filter:filterText |
           orderBy:sortOrder |
           limitTo:numLimit"
```

Dependency injection

- AngularJS uses dependency injection to decouple modules
  - Dependencies are automatically injected by the framework
- Based on the parameter name

```
var myApp = angular.module("myApp", []);

myApp.service("myService", function ($http) {
    this.doSomeWork = function () {
        // Do something
    }
});

function TheController($scope, myService) {
    myService.doSomeWork();
}
```

Dependency injection

- JavaScript is often minified in production
  - Need to provide AngularJS with some extra hints

```
var myApp = angular.module("myApp", []);

myApp.service("myService", ["$http", function ($http) {
    this.doSomeWork = function () {
        // Do something
    }
}]);

function TheController($scope, myService) {
     myService.doSomeWork();
}
TheController.$inject = ["$scope", "myService"];
```


Modules & Services

- Modules are groupings of related functionality
  - Also used to bootstrap the application
- Services are reusable pieces of business logic
  - Separation results in reuse and testability
- Created as singleton objects
  - Inject by AngularJS using dependency injection
- Services are created as part of a module
  - One module can take a dependency on another module

Services – The module and service

```
angular.module("myData", []).service("peopleLoader", function () {
    this.load = function () {
        return [{
            firstName: "Maurice",
            lastName: "de Beijer"
        }, {
            firstName: "Miško",
            lastName: "Hevery"
        }, {
            firstName: "Adam",
            lastName: "Abrons"
        }];
    };
});

angular.module("myApp", ["myData"]);

function PeopleCtrl($scope, peopleLoader) {
    $scope.people = peopleLoader.load();
}
```

Services – The markup

```
<!DOCTYPE html>
<html>
<head>
    <title>An AngularJS application</title>
</head>
<body ng-app="myApp" ng-controller="PeopleCtrl">

    <ul>
        <li ng-repeat="person in people">{{person.firstName}}</li>
    </ul>

    <script src="Scripts/angular.js"></script>
    <script src="App/SimpleCtrl.js"></script>
</body>
</html>
```


Standard Services

- Many general purpose services provided by AngularJS
- $http
  - Used for XMLHttpRequest handling
- $location
  - Provide information about the current URL
- $q
  - A promise/deferred module for async requests
- $routeProvider
  - Configure routes in an SPA
- $log
  - Logging service
- Many more

Routing

- Used to create SPA style application
  - The page can change without using the server
- The ngView is often used to render a template
  - HTML templates loaded when needed
  - Can also be pre loaded as script with type="text/ng-template"
- The $routeProvider service is used to configure the route
- The $location service can be used to navigate
  - Using an anchor tag is also possible
- The $routeParams service can be used to retrieve parameters
  - Properties named in the route URL template
- Requires a reference to angular-route.js
  - Starting with Angular 1.2

SPA Routing – The markup

```
<body ng-app="spaApp">
    <div ng-view></div>

    <script id="/templates/people.html" type="text/ng-template">
        <ul><li ng-repeat="person in people">
                <span ng-click="edit(person.id)">
                    {{person.firstName}} {{person.lastName}}
                </span></li></ul>
    </script>

    <script id="/templates/person.html" type="text/ng-template">
        First name:<input ng-model="person.firstName" />
        <br />
        Last name:<input ng-model="person.lastName" />
        <br />
        <button ng-click="toList()">To list</button>
    </script>

    <script src="Scripts/angular.js"></script>
    <script src="Scripts/angular-route.js"></script>
    <script src="App/SimpleCtrl.js"></script>
</body>
```

SPA Routing – The routing configuration

```
var app = angular.module("spaApp", ["myData"]);
app.config(function ($routeProvider) {
    $routeProvider
        .when("/people", {
            templateUrl: "/templates/people.html",
            controller: 'PeopleCtrl'
        })
        .when("/person/:id", {
            templateUrl: "/templates/person.html",
            controller: 'PersonCtrl'
        })
        .otherwise({ redirectTo: '/people' })
});
```


SPA Routing – The controllers

```
function PeopleCtrl($scope, peopleLoader, $location) {
    $scope.people = peopleLoader.load();

    $scope.edit = function (id) {
        $location.path("/person/" + id);
    };
}

function PersonCtrl($scope, peopleLoader, $routeParams, $location) {
    $scope.person = peopleLoader.get($routeParams.id);

    $scope.toList = function () {
        $location.path("/people");
    };
}
```


Summary

- AngularJS is a complete framework for client side applications
  - Based on the standard MVC design pattern
- Two way data binding makes it easy to build data entry forms
- Dependency injection makes it easy to separate modules
- Build with testing in mind