

# Lecture 9: Angular.js, DI and modules

---

Olivier Liechti  
TWEB

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud



# Dependency injection & modules

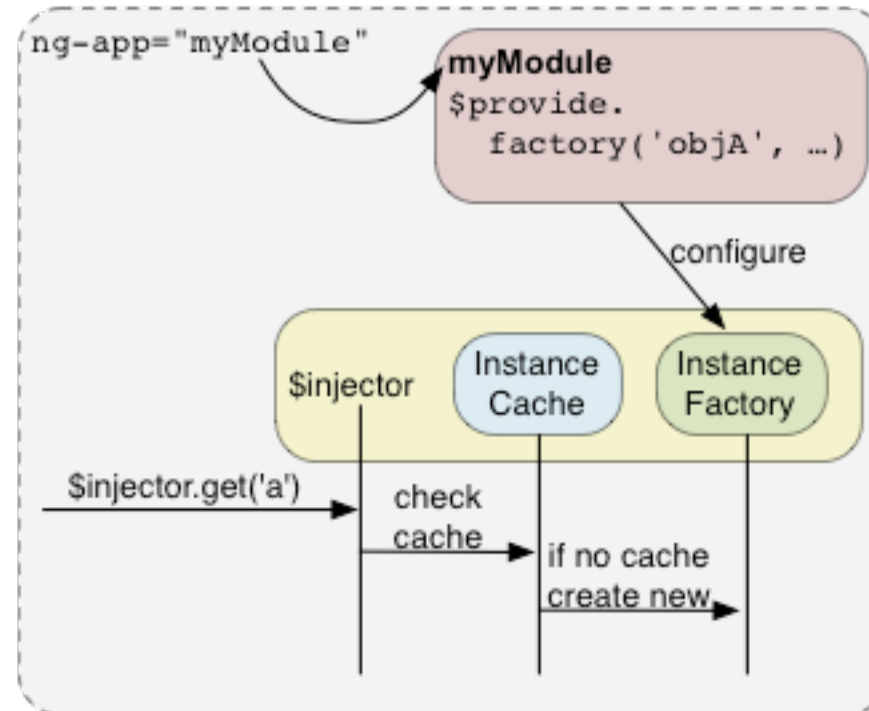
# Dependency injection (1)

---

- Remember the notion of **dependency injection**:
  - When you build your application as a **collection of services**, some services depend on other services.
  - For instance, an application-level “User Service” may depend on a lower-level “http” service.
  - When using dependency injection, a service **does NOT lookup** for an existing service. Instead, it **receives a reference** to this service from the environment.
- **Angular.js uses dependency injection a lot.** The “injector” is the framework component that deals with the injection (keeps track of available services, etc.).
- As an application developer, you have to use **special conventions** and **syntactic rules** to use the dependency injection framework.

# Dependency injection (2)

- Dependency injection is **described in several parts** of the documentation. Some of the key resources are:
  - <https://docs.angularjs.org/guide/di>
  - <https://github.com/angular/angular.js/wiki/Understanding-Dependency-Injection>



- (Unfortunately?), Angular.js provides different ways to express dependencies...

# Dependency injection (3)

This is a name

This is a variable

Inline Array Annotation (preferred)



```
someModule.controller('MyController', ['$scope', 'greeter', function($scope, greeter) {  
    // ...  
}]);
```

\$inject Property Annotation

```
var MyController = function($scope, greeter) {  
    // ...  
}  
MyController.$inject = ['$scope', 'greeter'];  
someModule.controller('MyController', MyController);
```

← These are the names, in the  
same order

Implicit Annotation (dangerous, will break if you minify your scripts!)

```
someModule.controller('MyController', function($scope, greeter) {  
    // ...  
});
```

“Given a function the injector can **infer the names of the services to inject by examining the function declaration and extracting the parameter names**. In the above example \$scope, and greeter are two services which need to be injected into the function.”

- Modules are used to **organize** your application components.
- A module is a “**collection of services, directives, controllers, filters, and configuration information**”.
- A module can have **dependencies** on other modules.
- Angular.js comes with a collection of **built-in modules** (provided in different .js files that must be loaded).
- In addition, a **large collection of third-party modules** is available. They provide rich functionality (e.g. file upload) and integration with other frameworks (e.g. Bootstrap).
- When you create an application, you create at **least one module**.

# Built-in modules

- Angular provides several modules “out-of-the-box”.
- Except for the **ng “core module”**, which is automatically loaded, you have to include a specific javascript file if you want to use one of these modules.

ng	Provided <b>by default</b> and contains the core components of AngularJS.
ngRoute	Use ngRoute to enable <b>URL routing</b> to your application.
ngAnimate	Use ngAnimate to enable animation features into your application.
ngAria	Use ngAria to improve the experience for users with disabilities.
ngResource	Use the ngResource module when calling a <b>REST API</b> .
ngCookies	Use the ngCookies module to handle cookie management.
ngTouch	Use ngTouch when developing for mobile browsers/devices.
ngSanitize	Use ngSanitize to securely parse and manipulate HTML data in your
ngMock	Use ngMock to inject and mock objects within your unit tests

# Third-party modules

---

- **Other developers**, including you, can create additional modules.
- The following web site provides a **list of popular angular modules**:  
<http://ngmodules.org/>
- **UI-Bootstrap**, **UI-Router** and **angular-file-upload** are examples of third-party modules that we used in the project.



# Modules

- **Creating a new module:**

```
var newModule = angular.module('myModule', []);
```

**angular.module** is a function provided by the default ng module

**myModule** is a name that we choose and that we will use to lookup the module later on.

This array contains the **name of the modules** that we depend on.

- **Lookup an existing module:**

```
var existingModule = angular.module('myModule');
```

# Angular full-stack generator

```
angular.module('generatorAngularFullstackApp', [  
  'ngCookies',  
  'ngResource',  
  'ngSanitize',  
  'btford.socket-io',  
  'ui.router',  
  'ui.bootstrap',  
  'luegg.directives',  
  'pdf'  
])
```

← The name of our  
module

← The list of  
dependencies. Each  
is loaded in  
index.html from a .js  
file.

app.js

```
<script src="bower_components/angular-resource/angular-resource.js"></script>  
<script src="bower_components/angular-cookies/angular-cookies.js"></script>  
<script src="bower_components/angular-sanitize/angular-sanitize.js"></script>  
<script src="bower_components/angular-bootstrap/ui-bootstrap-tpls.js"></script>
```

index.html

```
angular.module('ngResource', ['ng']).
```

angular-resource.js

# Recipes for adding “stuff” to a module

---

- When you have created a new module, you will want to create **services** and **specialized objects** (directives, filters, etc.).
- To do that, Angular.js has a notion of “recipe”. There are 5 types of recipes:
  - The Value Recipe
  - The Factory Recipe
  - The Service Recipe
  - The Provider Recipe
  - The Constant Recipe

# The Value Recipe

```
var myApp = angular.module('myApp', []);  
myApp.value('clientId', 'a12345654321x');
```

← Create a new module  
← Use a recipe to add a  
new value to the  
module

```
myApp.controller('DemoController', ['clientId', function  
DemoController(clientId) {  
    this.clientId = clientId;  
}]);
```

↑  
Access the value

↑  
Declare a dependency  
on the value

# The Factory Recipe

Dependency name



Dependency reference



```
myApp.factory('apiToken', ['clientId', function apiTokenFactory(clientId) {  
    var encrypt = function(data1, data2) {  
        // NSA-proof encryption algorithm:  
        return (data1 + ':' + data2).toUpperCase();  
    };  
  
    var secret = window.localStorage.getItem('myApp.secret');  
    var apiToken = encrypt(clientId, secret);  
  
    return apiToken;  
}]);
```



We return a String value, but we could return any kind of object (e.g. a function). The same is true for the Value Recipe.

# The Provider Recipe

```
myApp.provider('unicornLauncher', function UnicornLauncherProvider() {  
  var useTinfoilShielding = false;  
  
  this.useTinfoilShielding = function(value) {  
    useTinfoilShielding = !!value;  
  };  
  
  this.$get = ["apiToken", function unicornLauncherFactory(apiToken) {  
    return new UnicornLauncher(apiToken, useTinfoilShielding);  
  }];  
});
```

```
myApp.config(["unicornLauncherProvider",  
function(unicornLauncherProvider) {  
  unicornLauncherProvider.useTinfoilShielding(true);  
}]);
```

The Provider Recipe is the most generic (and verbose) recipe. All others are shortcuts built on top of it. One interesting feature is that it is possible to configure the factory (nice for generic services used in different applications)

# Angular full-stack generator (1)

```
angular.module('generatorAngularFullstackApp')
  .factory('User', function ($resource) {
    return $resource('/api/users/:id/:controller', {
      id: '@_id'
    },
    {
      changePassword: {
        method: 'PUT',
        params: {
          controller: 'password'
        }
      },
      get: {
        method: 'GET',
        params: {
          id: 'me'
        }
      }
    }
  ));
});
```


user.service.js

← 'User' is the name of our service.  
We have a dependency on the  
\$resource angular service

The Factory Recipe is used in other  
files as well.

# Angular full-stack generator (2)

The **Provider Recipe** has been used to create these services. For this reason, it is possible to configure them based on the needs of our application.



```
'use strict';

angular.module('generatorAngularFullstackApp', [
  'ngCookies',
  'ngResource',
  'ngSanitize',
  'btford.socket-io',
  'ui.router',
  'ui.bootstrap',
  'luegg.directives',
  'pdf'
])
.config(function ($stateProvider, $urlRouterProvider, $locationProvider, $httpProvider) {
  $urlRouterProvider
    .otherwise('/');

  $locationProvider.html5Mode(true);
  $httpProvider.interceptors.push('authInterceptor');
})
```



# Angular full-stack generator (3)

The **Provider Recipe** has been used to create the \$stateProvider service. For this reason, it is possible to configure them based on the needs of our application.



```
angular.module('generatorAngularFullstackApp')
  .config(function ($stateProvider) {
    $stateProvider
      .state('main', {
        url: '/',
        templateUrl: 'app/main/main.html',
        controller: 'MainCtrl'
      });
  });
```

main.js

```
angular.module('generatorAngularFullstackApp')
  .config(function ($stateProvider) {
    $stateProvider
      .state('login', {
        url: '/login',
        templateUrl: 'app/account/login/login.html',
        controller: 'LoginCtrl'
      })
      .state('signup', {
        url: '/signup',
        templateUrl: 'app/account/signup/signup.html',
        controller: 'SignupCtrl'
      })
      .state('settings', {
        url: '/settings',
        templateUrl: 'app/account/settings/settings.html',
        controller: 'SettingsCtrl',
        authenticate: true
      });
  });
```

account.js

# Topics for next lecture

---

- Suggested topics
  - Directives
  - UI Router
- Do you have other topics or questions that you would like to get more information about?
- Are there specific issues that you did not understand well when using the framework?
- This could be about Angular.js, but also about other frameworks and tools.
- Send me an email, as early as possible, so that I can integrate the content in the upcoming lecture.