specialist ICT learning

GURUTEAM

H&R BLOCK

AngularJS: Modules

GuruTeam Instructor: Sang Shin

Topics

- Modules
- Multi-module application
- Angular application directory structure
- Module loading and dependencies
- Angular initialization (bootstrapping)

Modules

What is a Module?

- You can think of a module as a container for the different parts of your app – controllers, services, filters, directives, etc.
  - From AngularJS 1.3.0, a controller cannot exist without being a part of a module

- You create a module via
  - var app = angular.module('myApp', []);
- Yon can reference a previously created module via
  - var app = angular.module('myApp');

Module and Dependency Modules

- Your application can be constructed with other helper modules (dependency modules)
  - Increases modularity and reusability of your application

```
var app = angular.module('myApp', ['myModule1', 'myModule2']);
```

Multi-Module Application

How to Construct Multi-Module App?

- For non-simple application, you want to break your application to multiple modules
  - A module for each feature
  - A module for each reusable directive, component and filter
  - And an application level module which depends on the above modules and contains any initialization code

Why Modulization?

- Reusability
  - > Other applications can use the modules – it is a matter of copying the module directory to the other application
- Self-contained Context
  - > Each module provides a context in which related components (services, directives, filters) are grouped together
- Testability
  - > Each module should be tested on its own

Angular App Directory Structure

Possible App Directory Structure #1

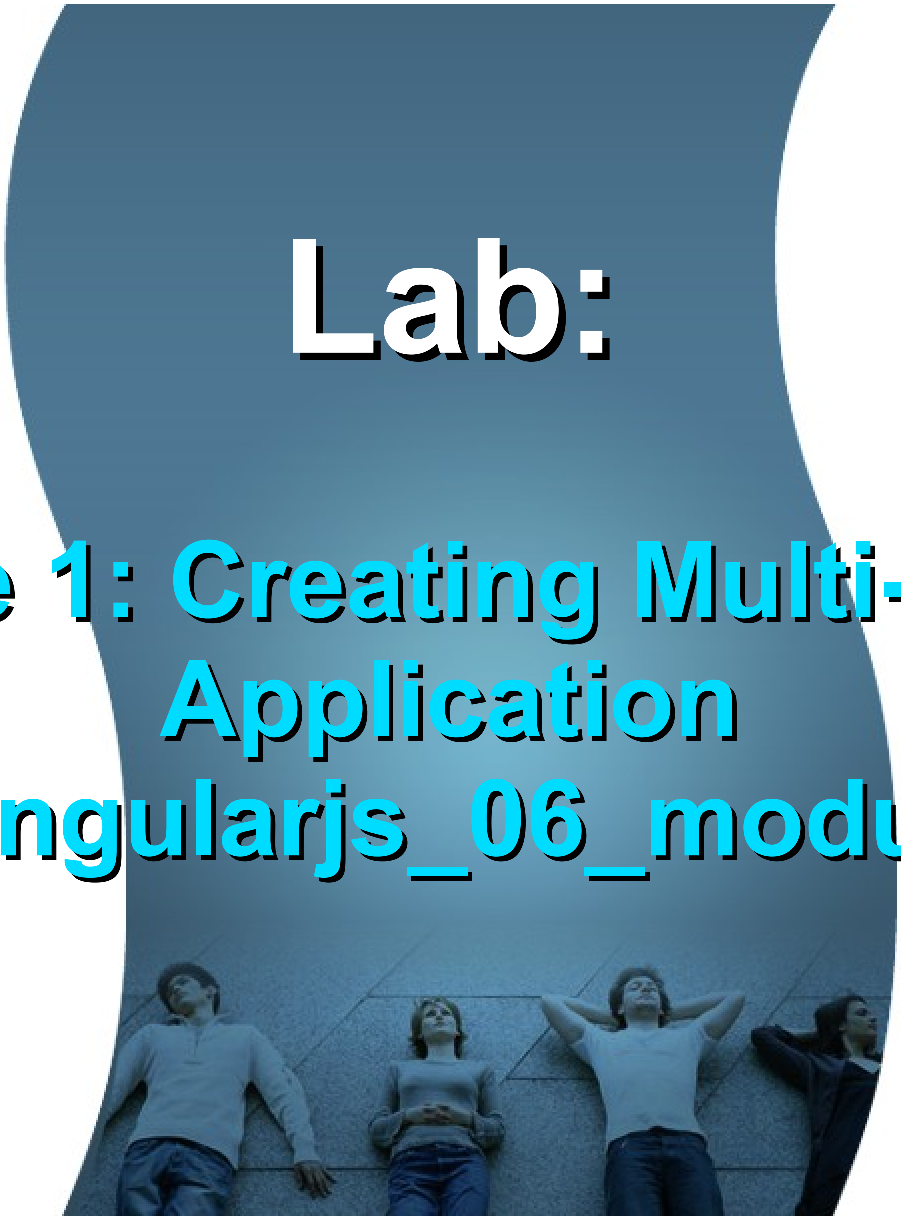
```
app/
------controllers/
--------------------mainController.js
--------------------anotherController.js
------directives/
--------------------mainDirective.js
--------------------anotherDirective.js
------services/
--------------------mainService.js
--------------------anotherService.js
------filters/
--------------------filter1.js
------views/
--------------------mainView.html
--------------------anotherView.html
------styles/
--------------------main.css
--------------------another.css
------app.js
------index.html
```

Based on AngularJS structural component category

Possible App Directory Structure #2

```
app/
------main/
--------------------mainController.js
--------------------mainDirective.js
--------------------mainService.js
--------------------mainView.html
--------------------main.css
------functionality1/
--------------------anotherController.js
--------------------anotherDirective.js
--------------------anotherService.js
--------------------anotherView.html
--------------------another.css
------shared/
--------------------sharedFilter.js
--------------------shared.css
------app.js
------index.html
```

Based on
Feature/Functionality
Recommended



Lab:

Exercise 1: Creating Multi-module Application 3306_angularjs_06_modules.zip

Module Loading & Dependencies

Configuration & Run Blocks of a Module

- A module is a collection of configuration and run blocks which get applied to the application during the bootstrap process
- Configuration block
  - Gets executed during the provider registrations and configuration phase
  - Only providers (not instances) and constants can be injected into configuration blocks
  - Prevents accidental instantiation of services before they have been fully configured.
- Run block
  - Gets executed after the injector is created and are used to kick start the application
  - Only instances and constants can be injected into run blocks.

Configuration & Run Blocks

```
angular.module('myModule', []).
config(function(injectables) { // provider-injector
  // This is an example of config block.
  // You can have as many of these as you want.
  // You can only inject Providers (not instances)
  // into config blocks.
}).
run(function(injectables) { // instance-injector
  // This is an example of a run block.
  // You can have as many of these as you want.
  // You can only inject instances (not Providers)
  // into run blocks
});
```

Configuration Blocks

```
angular.module('myModule', []).
  value('a', 123).
  factory('a', function() { return 123; }).
  directive('directiveName', ...).
  filter('filterName', ...);

// is same as

angular.module('myModule', []).
  config(function($provide, $compileProvider, $filterProvider) {
    $provide.value('a', 123);
    $provide.factory('a', function() { return 123; });
    $compileProvider.directive('directiveName', ...);
    $filterProvider.register('filterName', ...);
  });
```

Run Blocks

- Run blocks are the closest thing in Angular to the main method
  - A run block is the code which needs to run to kick-start the application
- It is executed after all of the services have been configured and the injector has been created
- Run blocks typically contain code which is hard to unit-test, and for this reason should be declared in isolated modules, so that they can be ignored in the unit-tests.

Dependencies Between Modules

- Modules can list other modules as their dependencies
- "Depending on a module" implies that the required module needs to be loaded before the requiring module is loaded
  - The configuration blocks of the required modules execute before the configuration blocks of the requiring module
  - The same is true for the run blocks
- Each module can only be loaded once, even if multiple other modules require it.

Creation vs Retrieval

- Creation of a module
  - *angular.module('myModule', [])* will create the module myModule and overwrite any existing module named myModule
- Retrieval of a module
  - *angular.module('myModule')* to retrieve an existing module

Creation vs Retrieval Example

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

// add some directives and services
myModule.directive('myDirective', ...);
myModule.factory('myService', ...);

// overwrites both myService and myDirective by creating a new module
var myModule = angular.module('myModule', []);

// throws an error because myOtherModule has yet to be defined
var myModule = angular.module('myOtherModule');
```

Recipes

Module contains recipes

- In order for the injector to know how to create and wire together all of these objects, it needs a registry of "recipes"
  - Each recipe has an identifier of the object and the description of how to create this object
- Each recipe belongs to an Angular module
  - An Angular module is a bag that holds one or more recipes
- When an Angular application starts with a given application module, Angular creates a new instance of injector, which in turn creates a registry of recipes as a union of all recipes defined in the core "ng" module
- The injector then consults the recipe registry when it needs to create an object for your application.

Value Recipe

- Let's say that we want to have a very simple service called "clientId" that provides a string representing an authentication id

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

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

<html ng-app="myApp">
  <body ng-controller="DemoController as demo">
    Client ID: {{demo.clientId}}
  </body>
</html>
```

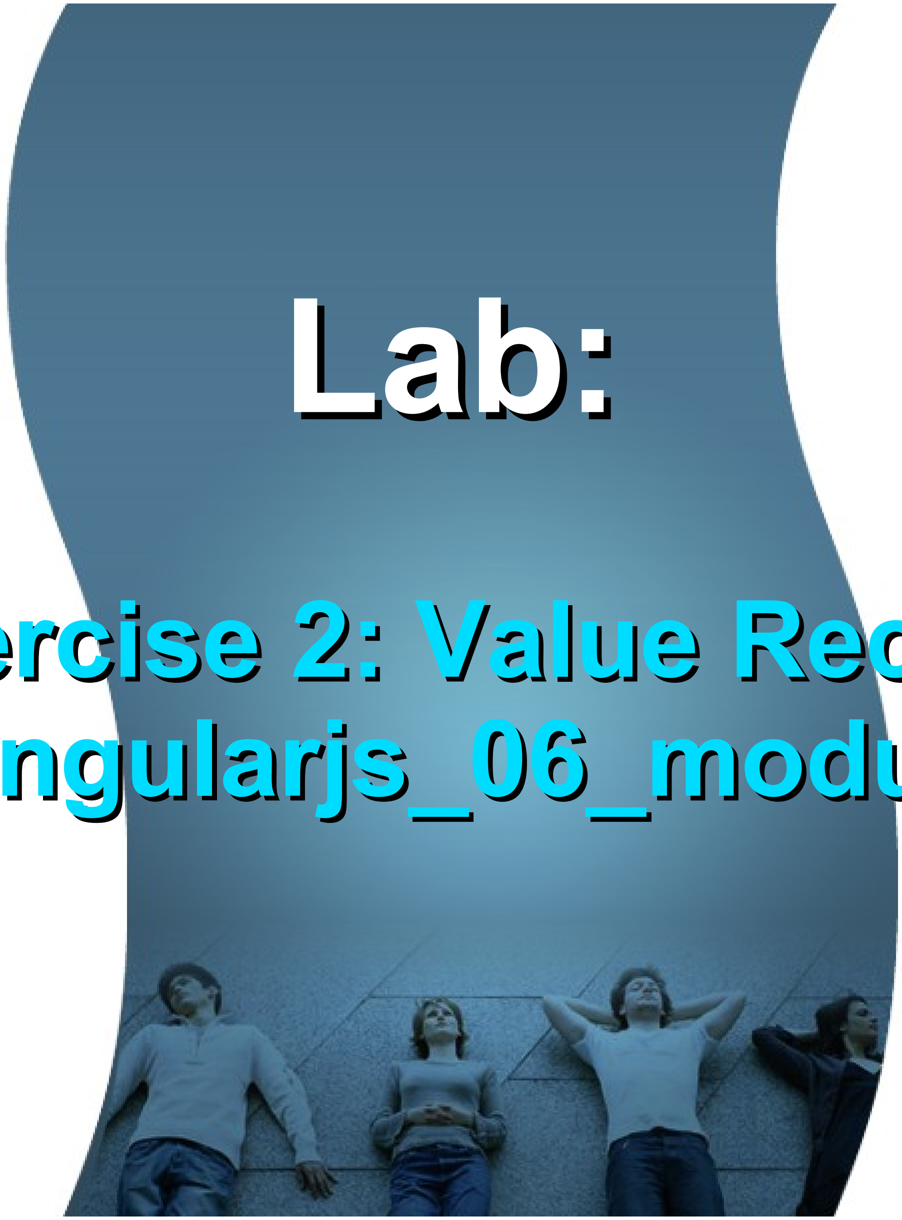
Factory Recipe

- The Factory recipe adds the following abilities to the value recipe
  - ability to use other services (have dependencies)
  - service initialization
  - delayed/lazy initialization

```
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;
}]);
```



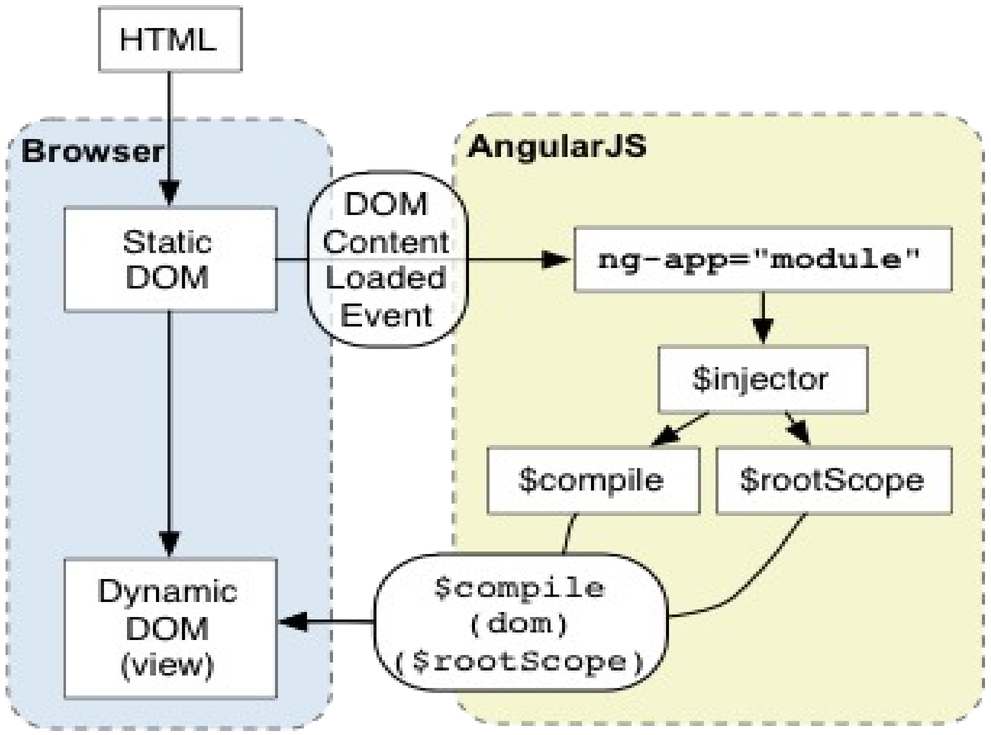
Lab:

Exercise 2: Value Recipe
3306_angularjs_06_modules.zip

Angular Initialization (Bootstrap)

Automatic Initialization

- Angular initializes automatically upon *DOMContentLoaded* event
- At this point Angular looks for the *ng-app* directive which designates your application root



Fired when the initial HTML document has been completely loaded and parsed, without waiting for stylesheets, images,

The compilation is a process of walking the DOM tree and matching DOM elements to directives

After “ng-app” is found, Angular will

- Load the module associated with the directive
  - ng-app=”myApp”
- Create the application injector
  - The injector is responsible for actually creating instances
  - There is only a single injector per application
  - Can be referred to as *$injector*
- Compile the DOM treating the *ng-app* directive as the root of the compilation
  - The compilation is a process of walking the DOM tree and matching DOM elements to directives
  - This allows you to tell it to treat only a portion of the DOM as an Angular application

Who are GuruTeam?

- Specialist onsite training in Linux, Cloud, Database, Architecture, Software and Web Development Technologies
- Accredited by the LPI, CompTIA, Hortonworks and the Cloud Credential Council to deliver training, examinations and certifications.
- Over 230 courses available
- All GuruTeam instructors have extensive real-world experience in their technologies
- Clients are indigenous Irish Companies and Multinationals
- We can bring high spec preconfigured equipment for deliveries in Ireland, the UK and Europe.

Exceed your expectations...
GURUTEAM