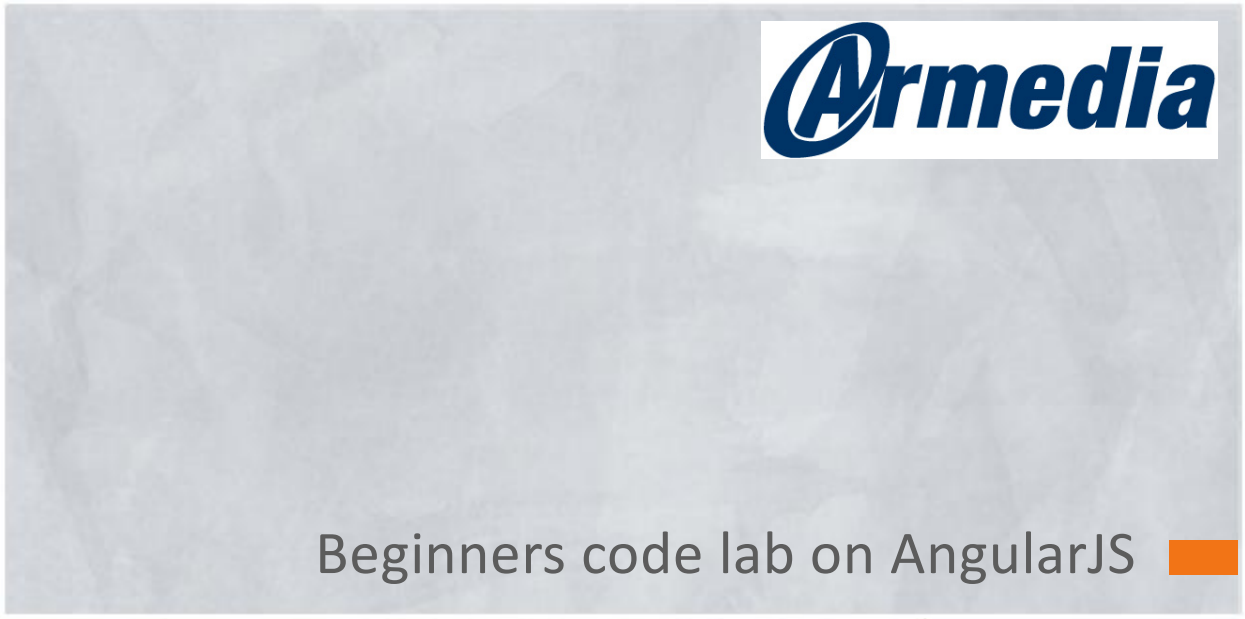


Beginners code lab on AngularJS

Code lab 2016

AngularJS

AngularJS Codelab introduction

This code lab is for those who

- Want to learn and develop applications using AngularJS
- Want to evaluate the AngularJS platform for their next projects

You will need on your laptops

- AngularJS 1.5.8 (latest stable release)
- Text editor of your choice – Eclipse, SublimeText, WebStorm
- Node.js and npm preinstalled (latest versions) - http://nodejs.org/downloads

Prerequisites

- Moderate knowledge of HTML, CSS, and JavaScript
- Basic Model-View-Controller (MVC) concepts
- The Document Object Model (DOM)
- JavaScript functions, events, and error handling

Introduction to AngularJS

AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you would otherwise have to write.

Features

- AngularJS is powerful JavaScript based development framework to create RICH Internet Application
- AngularJS is open source, completely free framework used by more than thousands of developers which provides developers options to write client side applications in MVC(Model View Controller) way
- Application written in AngularJS is cross-browser complaint

Environment Setup

Tools needed for developing applications using AngularJS

Getting Started with AngularJS

- Install JDK 1.8 - http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- Install Git client - https://git-scm.com/download/gui/linux
  - Make sure git/bin and git/cmd are added to system Path variable.
  - to check if git is installed type below command:
    git --version
- Install Node.js from https://nodejs.org/en/download/
  - Node.js is JavaScript runtime built on Chrome's V8 JavaScript engine which uses an event-driven, non-blocking I/O model.
  - with installing Node.js you are getting also package manager npm – largest package manager of open source libraries
  - to check if node is installed type below command:
    $ node -v
  - to check if npm is installed type below command:
    $ npm –v
- Clone the code lab project locally
  git clone https://gitlab.com/armedia/angular-codelab.git

Getting Started with AngularJS

- On the command line utility, run the following command:
  sudo npm -g install gulp bower
  - This will install bower dependency management tool which is used for managing front end components html, css, js etc.. and gulp which is task/build runner for developers
- Go to the root folder of project and run
  - npm install - this will install following tools into node_modules directory:
    - Bower - client side code package manager
    - Http-Server - simple local static web server
    - Karma - unit test runner
    - Protractor - end to end (E2E) test runner etc..
  - bower install - this will install packages to bower_components
    - A package can be a GitHub shorthand, a Git endpoint, a URL, and more
  - gulp serve

  You are now ready to go, your application is available at **http://127.0.0.1:3000**.

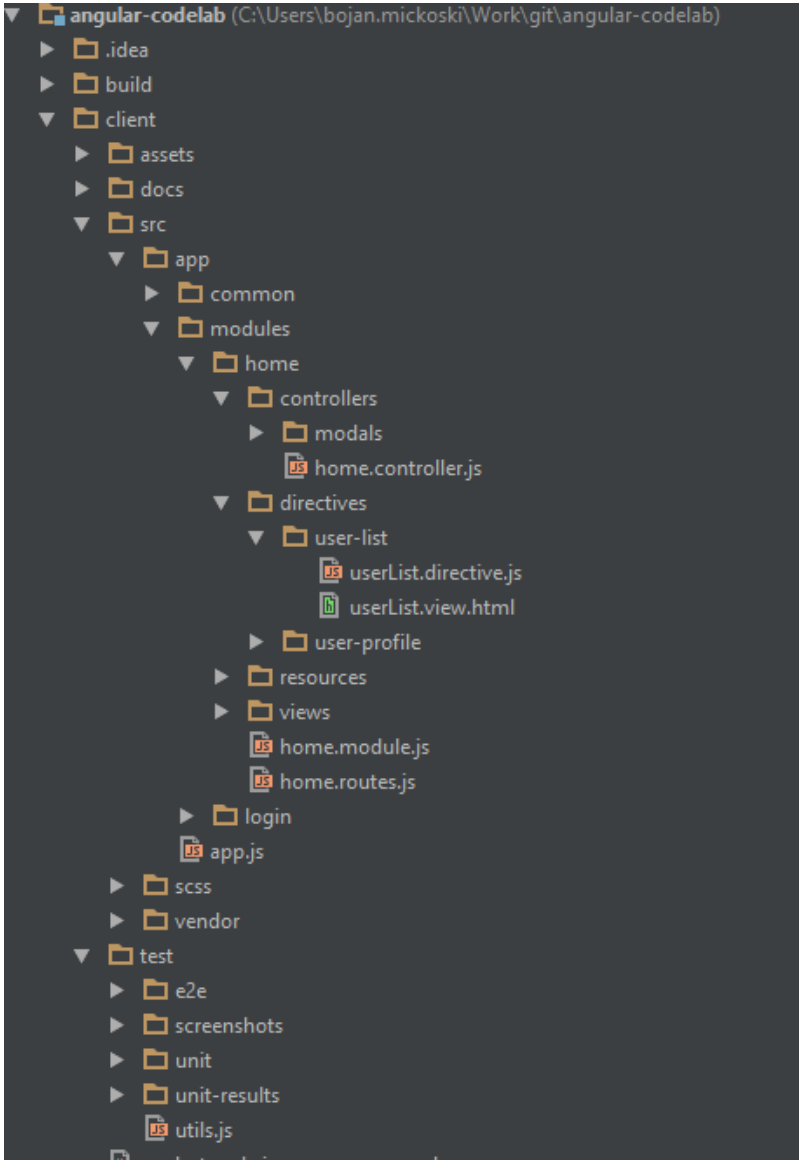
AngularJS application

Application Development – Core Concepts

Concept	Description
Data Binding	sync data between the model and the view components
Module	a container for the different parts of an app including controllers, services, filters, directives which configures the Injector
Controller	the business logic behind views
Service	reusable business logic independent of views
Directive	extend HTML with custom attributes and elements
Dependency Injection	Built in dependency injection for making application easier to develop, test.. etc..
Routing	It is concept of switching views
Testing	Unit testing helps maintain clean code

AngularJS application

Structure



AngularJS application

Data Binding

Data-binding in Angular apps is the automatic synchronization of data between the model and view components.

- Use ng-model to bind data from the model to the view

```
function LoginCtrl($state, loginService, sessionService) {
        var loginvm = this;
        loginvm.user = {};
}

<input ng-model="loginvm.user.password" >
```

- Two – way Binding

When data in the model changes, the view reflects the change, and when data in the view changes, the model is updated as well.

AngularJS application

Modules

Module is a collection of services, directives, controllers, filters, and configuration information

- Declare modules without variable using setter syntax

```
angular
        .module('app', [
                'ui.router',
                'ui.bootstrap',
                'home',
                'login',
                'getting-started',
                'common',
                'templates'
    ])
```

Modules

- When using module avoid using variable

```
angular
      .module('home')
      .controller('HomeCtrl', HomeCtrl);

function HomeCtrl) { }
```

- angular.module(home', []); sets a module, whereas angular.module(‘home'); gets the module. Only set once and get for all other instances.

- Use named functions instead of passing an anonymous function in as a callback.

- This produces readability and reduces the volume of code ‘wrapped’ inside the Angular framework

AngularJS application

Modules

```
// dataService.js
angular
        .module('common')
        .factory('DataService', dataService);
function dataService() { }

// userList.directive.js
angular
    .module(common')
    .directive('userList', userList);

 function userList($uibModal) {
```

- Use **IIFE** (Immediately Invoked Function Expression) scoping

```
(function() {

    angular
        .module('login', []);
})();
```

AngularJS application

Controllers

In Angular, a Controller is defined by a JavaScript **constructor function** that is used to augment the angular scope.

- Use controllerAs View syntax over classic controller with $scope syntax

```
<h3 Conversation between {{ userselectionvm.loggedUser}} and
   {{ userselectionvm.user.name}}</h3>
```

- Use controllerAs with vm Controller syntax over classic controller with $scope syntax. Use a capture variable for this with this syntax.

```
function UserSelectionModalController(user, sessionService) {
   var userselectionvm = this;

   userselectionvm.user = user;
   userselectionvm.loggedUser = sessionService.getUser();
}
```

Controllers

The this keyword is contextual and when used with a function inside a controller may change its context. Capturing the context of this (using variable name such as vm) avoids encountering this problem.

This syntax is more contextual, easier to read and avoids any reference issues that may occur without using ‘dotting’(userselectionvm.loggedUser).

- Bindable members Up Top

Bindable members should be placed at the top of the controllers, alphabetized and not spread through the controller code. This makes easy to read and helps instantly identify which members of controller can be bound and used in the View.

Setting functions in-line is fine as long as those functions are not more then 1 line. Otherwise functions should be defined below bindable members.

AngularJS application

Controllers

```
function UserSelectionModalController (sessionService, user,
conversationService) {
    var userselectionvm = this;
    userselectionvm.currentTime = new Date().getTime();
    userselectionvm.user = user;
    userselectionvm.enteredText = "";

    userselectionvm.dismissUserSelectionModal =
dismissUserSelectionModal;
    userselectionvm.concatConversation = concatConversation;

    userselectionvm.loggedUser = sessionService.getUser();//1 liner func()
    userselectionvm.currentConversation =
conversationService.getConversation(userselectionvm.user.id);// 1 liner
func()

function concatConversation () { … }

function dismissUserSelectionModal () { … };
```

Controllers

- Presentational logic only (MVVM)

Presentational logic only inside a controller, avoid Business logic(delegate to Services)

```
function HomeCtrl($state, $http, $uibModal, sessionService, dataService) {
        function getUsers () {
                dataService.getUsers().then(function (users){
                    homevm.users = users;
                });
        }
}
```

- When controller must be paired with a view and either component may be re-used by other controller or views define controllers along with routes

Pairing the controller in the route allows different routes to invoke different pairs of controllers and views. When controllers are assigned in the view using ng-controller, that view is always associated with the same controller.

AngularJS application

Controllers

```
(function() {
 'use strict';

 function config($stateProvider) {
  $stateProvider
   .state('root.home', {
    url: '/home',
    views: {
     '@': {
      templateUrl: 'src/app/modules/home/views/home.view.html',
      controller: 'HomeCtrl as homevm'
     }
    }
   });
 }

 angular
          .module('home')
          .config(config);
})();
```

AngularJS application

Services

Services are substitutable objects that are wired together using dependency injection. They are used to organize and share code across application.

- All services are singletons, there is only one instance of a given service per injector

```
function sessionService() {

  function getUser() {
            return this.user.username;
  }
  return {
       getUser: getUser,
  };
}
angular
      .module('common')
      .factory('SessionService', sessionService);
```

Services

- Refactor logic for making data operations and interacting with data to a factory. This will hide implementation from outside controllers, directives etc.. and also will make easier to change that implementation

```
function dataService($http) {

  function getUsers() {

    return $http.get('assets/home/users.json')
        .then(function(result){
            return result.data;
      });
  }
  return {
   getUsers: getUsers
  };
}

angular
      .module('common')
      .factory('DataService', dataService);
```

AngularJS application

Directives

At a high level, directives are markers on a DOM element that tell AngularJS's HTML compiler ($compile) to attach a specified behavior to that DOM element

- Create one directive per file

One directive per file is easy to maintain.

- Naming conventions

Avoid using ng-* prefix custom directive, they might conflict future native directives.

- Restrict to Elements and Attributes

When creating directive only use custom element (E) and custom attribute (A) method. EA is default for angular 1.3 +.

AngularJS application

Directives

```
angular
    .module('home')
    .directive('userList', userList);

function userList() {
    var directive = {
            link: link,
            restrict: 'E',
            scope: {
                  userList: '='
            },
      templateUrl: 'src/app/modules/home/directives/user-
list/userList.view.html‘;
    };

    return directive;

    function link(scope, element, attr, table) {
    }
}
```

AngularJS application

Dependency Injection

Dependency Injection (DI) is a software design pattern that deals with how components get hold of their dependencies.

- Avoid using the shortcut syntax of declaring dependencies

- Use $inject to manually identify your dependencies for Angular components

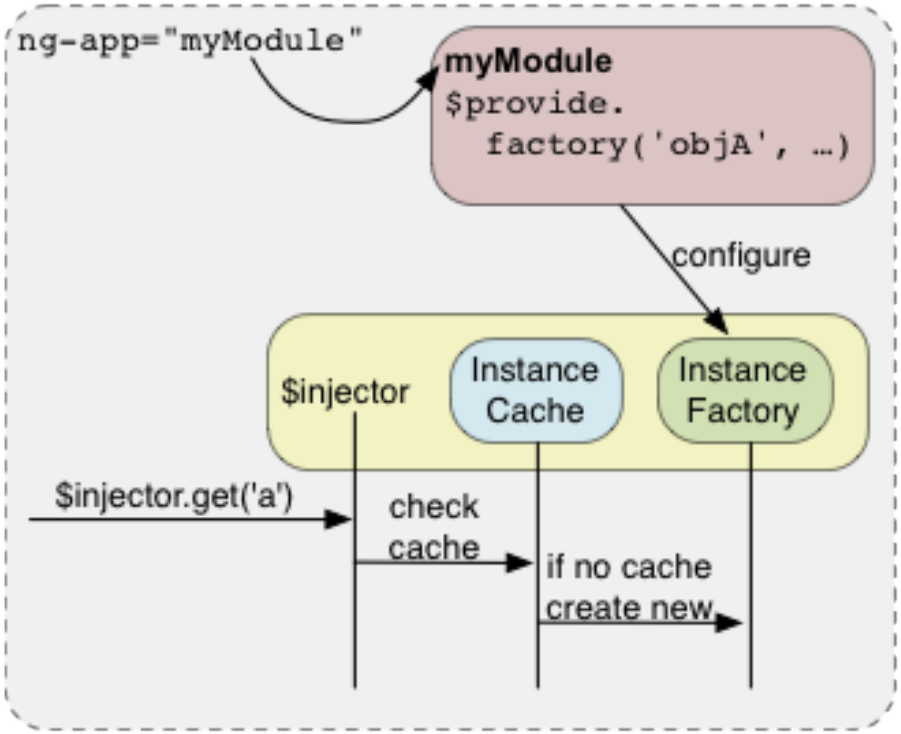
```
angular
        .module('login')
        .controller('LoginCtrl', LoginCtrl);

    LoginCtrl.$inject = ['$state', 'LoginService', 'SessionService'];

    /* @ngInject */
    function LoginCtrl($state, loginService, sessionService) {}
]);
```

AngularJS application

Dependency Injection

This mirrors the technique used by ng-annotate. If ng-annotate detects injection has already been made, it will not duplicate it. This safeguards your dependencies from being vulnerable to minification issues when parameters may be mangled

Avoid creating in-line dependencies as long as lists can be difficult to read in the array.



AngularJS application

Routing

Routing is important for creating a navigation flow between views and composing views.

- Use angular ui-router

Angular UI-Router is a client-side Single Page Application routing framework for AngularJS. UI Router offers all the features of the Angular router plus a few additional ones including nested routes and states.

- Define routes for views in the module where they exis

Each module should be able to stand on its own.

When removing a module or adding a module, the app will only contain routes that point to existing views.

AngularJS application

Routing

```
function config($stateProvider) {
   $stateProvider
    .state('root.home', {
      url: '/home',
      views: {
       '@': {
         templateUrl: 'src/app/modules/home/views/home.view.html',
         controller: 'HomeCtrl as homevm'
       }
      }
    });
 }

 angular.module('home')
  .config(config);
```

AngularJS application

Testing

Unit testing improves code's orthogonality.
Fundamentally, code is called "orthogonal" when it's easy to change.

- Write a set of tests for every story. Start with an empty test and fill them in as you write the code for the story.

When removing a module or adding a module, the app will only contain routes that point to existing views.

```
it("should call checkAuthentication with return value true and get users",
function() {
});

it("should call checkAuthentication with return value true and fail for
getting users", function() {
});
```

Testing

- Use Jasmine for unit testing

Jasmine is widely used in the Angular community. It is stable, well maintained, and provide robust testing features.

- Use Karma as a test runner

Karma is easy to configure to run once or automatically when you change your code

Karma hooks into your Continuous Integration process easily on its own or through Grunt or Gulp.

- Use PhantomJS ro run test pm a server

PhantomJS is a headless browser that helps run your tests without needing a "visual" browser.

AngularJS application

Testing

```
describe('loginUser with bad credentials',function(){
     beforeEach(function(){
       spyOn(mockLoginService ,
'userLogin').and.callFake(function(mockupBadUser) {
         var deferred = q.defer();
          deferred.reject();

         return deferred.promise;
       });
     });
     it("should call loginUser from LoginService with bad credentials",
function() {
       loginController.loginUser();

       rootScope.$digest();

       expect(mockLoginService.userLogin).toHaveBeenCalled();
     });
   });
```

NgDocs

Documenting code is important – if you're working in a team, it allows colleagues who perhaps weren't involved in the original development to see what goes where.

- Angular has its own style of JSDocs called ngDocs

The greatest thing about ngDocs, is that there are generators to build documentation sites which are very similar to the official Angular documentation.

- Use grunt or gulp

In codelab we are using gulp. Gulp is a fast and intuitive streaming build tool built on Node.js

To run unit tests go to root folder of project and run:

$ gulp

Server will be run at **http://127.0.0.1:8000**.

AngularJS application

NgDocs

```
/**
  * @ngdoc controller
  * @name home.controller: HomeCtrl
  *
  * @description
  *
  * {@link https://gitlab.armedia.com/solutions-engineering/angular-codelab/blob/master/client/src/app/modules/home/controllers/home.controller.js home/controllers/home.controller.js}
  *
  * This controller is responsible for showing home page, logged in users.
  *
  * @requires $rootScope
  * @requires $state
  * @requires $uibModal
  * @requires $translate
  * @requires SessionService
  * @requires DataService
  *
  */
```

Ready for coding !!!

Thank You …

That’s all folks …

Contact:
Bojan Mickoski
bojan.mickoski@armedia.com

Vishal Deshpande
vishal.deshpande@armedia.com