Usage Guidelines

Do not forward this document to any non-Infosys mail ID. Forwarding this document to a non-Infosys mail ID may lead to disciplinary action against you, including termination of employment.

Contents of this material cannot be used in any other internal or external document without explicit permission from ETA@infosys.com.

Angular JS

Day 3

Education, Training and Assessment

We enable you to leverage knowledge anytime, anywhere!

Infosys® | Building Tomorrow's Enterprise

Copyright Guideline

Confidential Information

- This Document is confidential to Infosys Limited. This document contains information and data that Infosys considers confidential and proprietary ("Confidential Information").
- Confidential Information includes, but is not limited to, the following:
  - Corporate and Infrastructure information about Infosys
  - Infosys' project management and quality processes
  - Project experiences provided included as illustrative case studies
- Any disclosure of Confidential Information to, or use of it by a third party, will be damaging to Infosys.
- Ownership of all Infosys Confidential Information, no matter in what media it resides, remains with Infosys.
- Confidential information in this document shall not be disclosed, duplicated or used – in whole or in part – for any purpose other than reading without specific written permission of an authorized representative of Infosys.
- This document also contains third party confidential and proprietary information. Such third party information has been included by Infosys after receiving due written permissions and authorizations from the party/ies. Such third party confidential and proprietary information shall not be disclosed, duplicated or used – in whole or in part – for any purpose other than reading without specific written permission of an authorized representative of Infosys.

Course Information

Course Code	: LA1211
Course Name	: AngularJS
Version Number	: 1.2

Day 3 Objectives

- Dependency injection
- Services

References

- Brad Green, Shyam Seshadri, AngularJS , O'Reilly Media, 2013.
- http://docs.Angularjs.org/guide/
- https://github.com/Angular

Services and Dependency Injection

Dependency injection

- AngularJS comes with an in-built dependency injection subsystem.
- DI makes the application development and testing much easier.
- DI allows the developer to ask for the dependencies from Angular. There is no need for the developer to explicitly create/instantiate them.
- Creation of dependent modules and providing the same to the developer is automatically taken care of via DI.

Dependency Injection

- Dependency injection defines how objects can get hold of their dependencies.
- We can do this in three ways:
  - Creating dependency using new operator.
  - Referring to a global variable to look up a dependency.
  - Passing the dependency in to where it is needed.
- For the first two options, we need to hardcode the dependencies and hence are not considered optimal.
- The third option is the most viable as we need not bother about locating the dependency from the component. The dependency is simply handed over to the component.
- Dependency injection is available by default in Angular.
- It is used mostly in controllers and factory methods.

Services

- Services in Angular are singleton objects or functions
- Services carry out specific tasks common to web apps.
- Angular services are a mechanism of abstracting shared code and functionality throughout the application
- Angular Services come as objects which are wired together using dependency injection.
- Angular provides few inbuilt services.
- We can also create custom services.
- To gain access to core AngularJS services, we just need to mention the required service as a parameter. Angular will automatically detect the required service and instantiate and provide the same.

```
function Ctrl($scope, $location, $routeParams) {
// Something code here...
}
```

Services

- $compile
  - This service will help in compiling an HTML string or DOM into a template
  - produces a template function, which can be used to link scope and the template together.
- $controller
  - This service takes care of instantiating controllers.
- $document
  - A jQuery or jqLite wrapper for the browser's window.document object.
- $exceptionHandler
  - Responsible for handling any uncaught exception in Angular expressions. The default implementation simply delegates to $log.error which logs it into the browser console.

Services

- $http service- Ajaxifying Angular
  - $http service helps in making asynchronous calls to server through $xhr.
  - The data retrieved from backend can thus be rendered in Angular view by partial rendering using Ajax.
  - Angular being a popular framework for SPA, has implicit support for Ajax and doesn't require any other JavaScript Ajax library support.

Services

- $http service
  - Helps in communication with the remote HTTP servers via the browser's XMLHttpRequest object or via JSONP.

```
$http.get(<url>)
        .success(function(data, status, headers, config){ // do something with data })
        .error(function(data, status, headers, config){ // some bad happened });
```

  There are similar methods for post, put, delete & head.

- Generic method:

```
$http({ url: <url>, method <method>, headers:{ <headerName>: <headerValue>},  cache: true/false, timeout: <time in millis>, params: <url parameters>)
```

- Setting cache as true sets up caching for all GET requests.
- Once response from a url is obtained it will not be requested again

Services

- $http or $http.<method>() functions return a promise
- This promise object provides following 3 functions:
  - then(successCallback, failureCallback)
  - success (data, status, headers, config, statusText)
  - error(data, status, headers, config, statusText)
- Headers can also be set for all the http requests:

```
$httpProvider.defaults.headers.common = { ‘header' : 'value' }
//headers for all calls

$httpProvider.defaults.headers.post = { ‘header' : 'value' }
            //headers for only POST calls
```

Ajaxifying Checklist Case study

- Lets create *CheckListItems.json* file inside a folder called *data*. The json file has the below content.

```
[
    {"text":"Wheat Flour", "done":true},
    {"text":"Tooth Paste", "done":false},
    {"text":"Rice Flour", "done":true},
    {"text":"Talcum Powder", "done":false},
    {"text":"Ragi", "done":true},
    {"text":"Maida", "done":false}
]
```

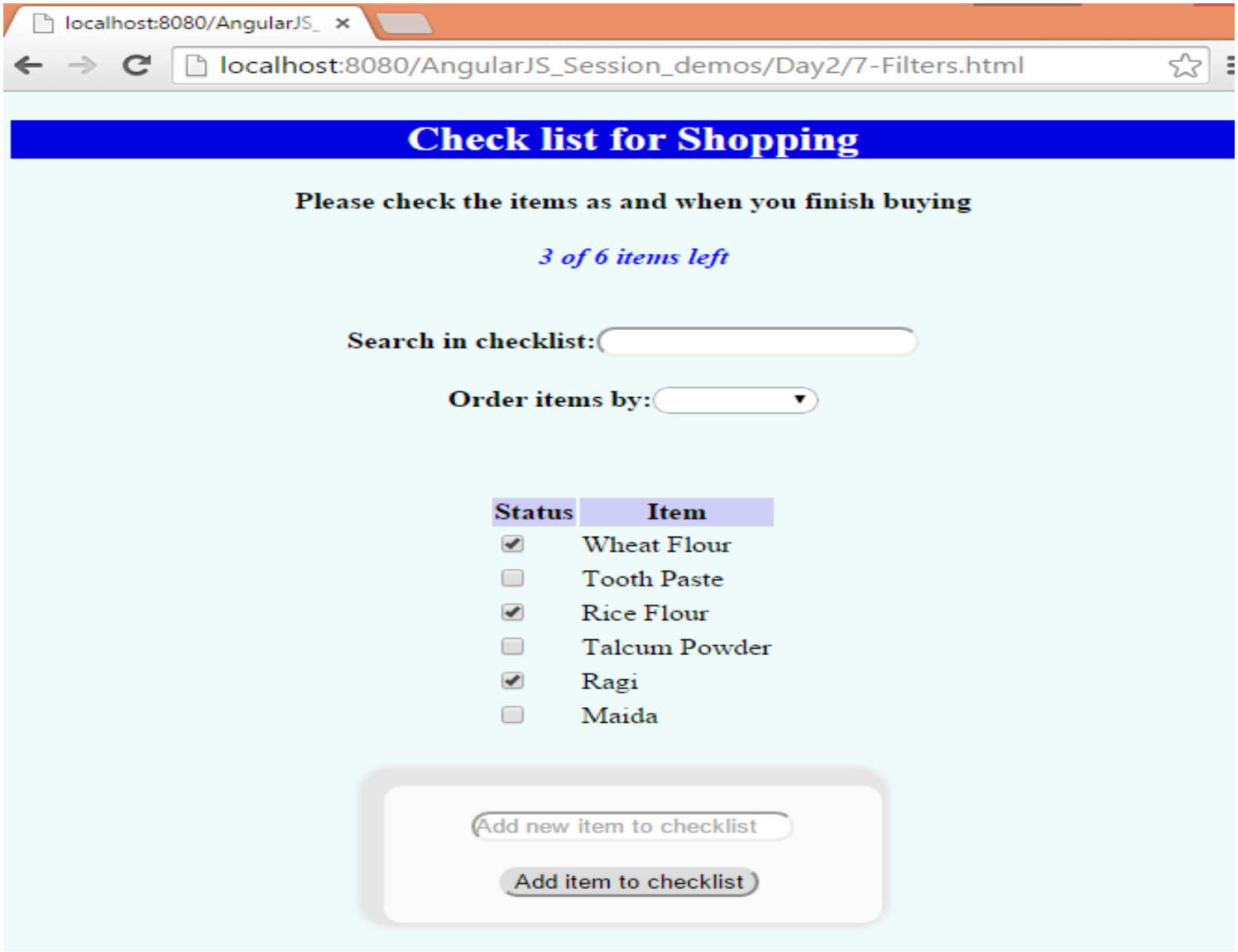
Ajaxifying Checklist Case study

- Lets modify the controllers_checklist.js to make an Ajax call to the *CheckListItems.json* file stored inside *data* folder.

```
function ItemCtrl($scope, $http) {
  $http.get('data/CheckListItems.json').success(function(data) {
  $scope.items = data;
  });
$scope.remaining = function() {
    var count = 0;
                angular.forEach($scope.items, function(item) {
               if(item.done==false)
                          count=count+1;
               });   return count;
  };
$scope.addItem = function() {
    $scope.items.push({text:$scope.itemText, done:false});
    $scope.itemText = '';    };   }
```

Making an Ajax call to CheckListItems.json and storing the output to *items* model

Ajaxifying Checklist Case study



$location service

- $location service provides access to window.location
  - **absUrl()** returns the complete url of the page
  - **protocol()** returns the protocol used http or https
  - **host()** returns the host
  - **port()** returns the port
  - **search()** returns the url parameters as an object
  - **search(param, [value])*** set/get a url parameter
  - **path([newPath])** set/get part of url after # without query params
  - **url([newUrl])** set/get part of url after # including query params
- Currently Angular does not provide two way binding for $location methods
- Changing path or url does not cause page reload
- To reload page use $window.location.href

Services

- $location service
  - This service helps in parsing the URL present in the browser's address bar
  - The mentioned URL is then made available to the application.
  - Whenever the URL in the address bar changes, the same is reflected onto the $location services and vice-versa.
- $location service broadcasts following two events on $rootScope:
  - $locationChangeStart
  - $locationChangeSuccess
- Any component interested in listening to url changes can set up a listener to these broadcasts.

$rootScope Service

- $rootScope is the top level scope of an Angular app
- Only 1 $rootScope exists in an app
- $rootScope is created automatically by Angular
- Any component can request for $rootScope via dependency injection

```
myApp.controller("myController", function($scope,
$rootScope){ //code }
```

- $rootScope can be used for making application level broadcasts
- Certain events are fired by Angular API on the $rootScope
  - E.g. $locationChangeSuccess fired by $location

Summary

- Dependency injection
- Services

Thank You

Infosys® | Building Tomorrow's Enterprise