Internet Programming

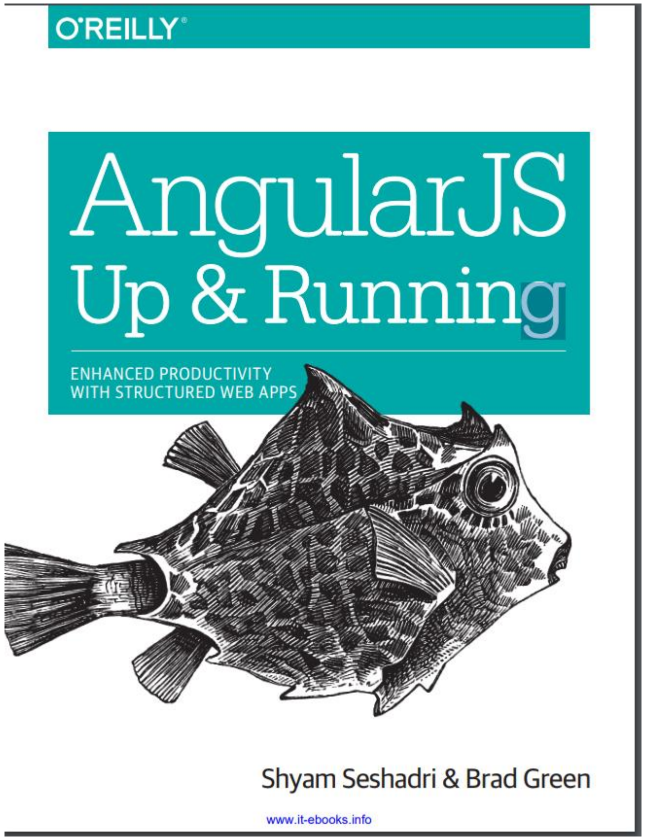
Practice #10

AngularJS - Lab 1

Dept. of Software and Information Systems Engineering

By Shir Frumeramn based on Hasidi Netanel slides

Excellent book



Available to you on moodle

https://moodle1370/php.elifnigulp/eldoom/li.ca.ugb.2%20SJralugnA/2/tnetnoc/ecruoser_dom/620Up%20And%20Running.pdf

Why AngularJS?
Official Documentation

“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 currently have to write. And it all happens within the browser, making it an ideal partner with any server technology…”

Why AngularJS? Features

- AngularJS is perfect for Single Page Applications (SPAs)
- AngularJS provides developers options to write client side application (using JavaScript) in a clean MVC (Model View Controller) way.
- Application written in AngularJS is cross-browser compliant. AngularJS automatically handles JavaScript code suitable for each browser.
- AngularJS is open source, completely free, and used by thousands of developers around the world

What is AngularJS?

- It is a client-side JavaScript framework
- It extends HTML DOM with additional attributes and makes it more responsive to user actions
- Our HTML code runs each time different JavaScript code (view) results in dynamic web page
- It works in a concept of MVC (Model-View-Controller) (Actually implement more like MVVM)

MVC in AngularJS

- **Model:**
  - The data that we work with:

    Can be a primitive type such as string, number, boolean or a complex type such as object
- **View:**
  - Display content and data to a user in a browser ng-app, ng-view
- **Controller:**
  - Preforms the functionality and the interaction between the view and the model - $scope
  - Different controllers for different areas in our application

How to use it?

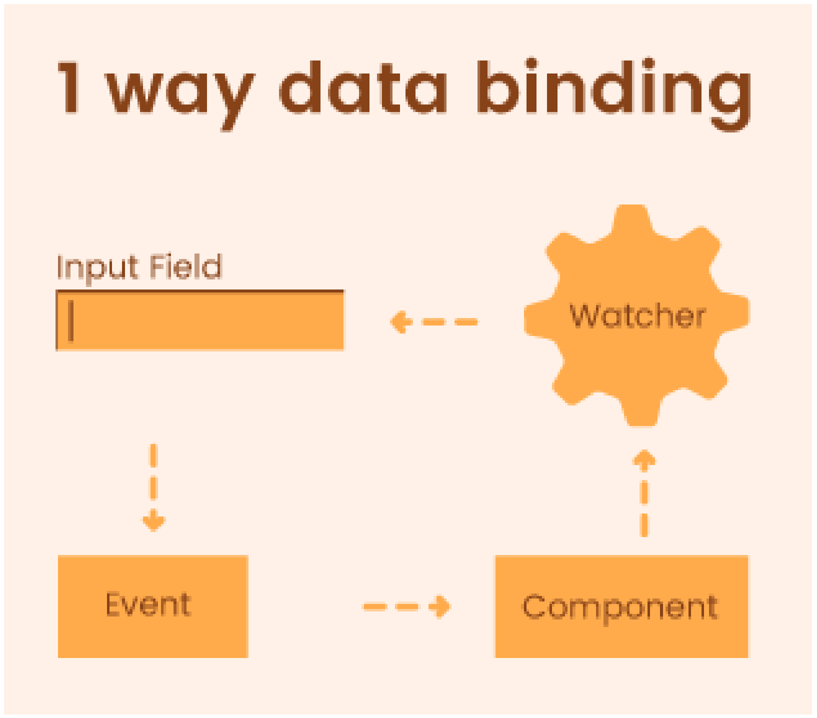
- AngularJS is distributed as a JavaScript file, and can be added to a web page with a script tag:

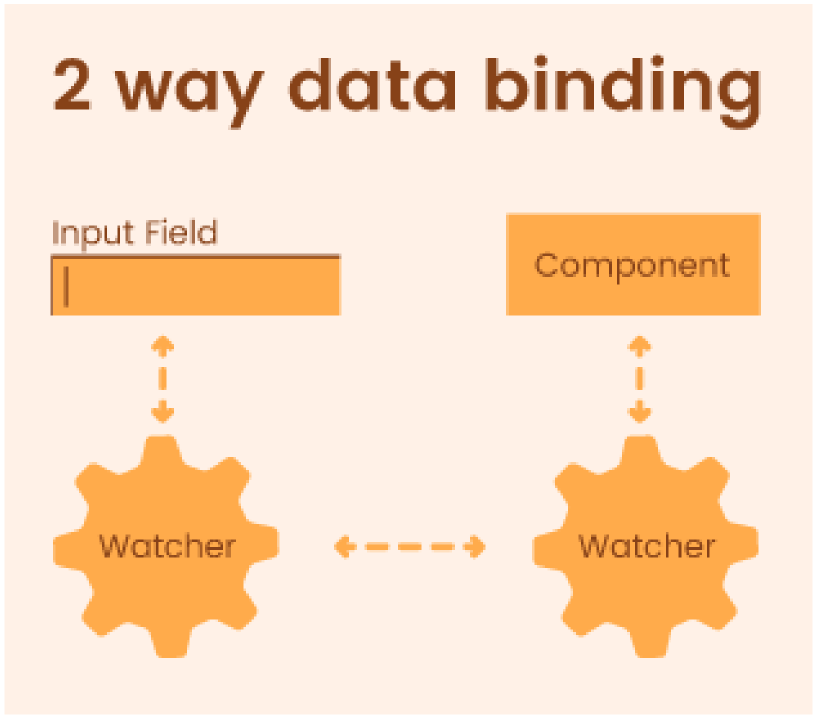
```
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
</script>
```

Data Binding

- **One way** - we take data coming from the server (or any other source), and update the Document Object Model (DOM)
- **two-way** - data-binding ensures that our controller and the UI share the same model, so that updates to one (either from the UI or in our code) update the other automatically

Data Binding

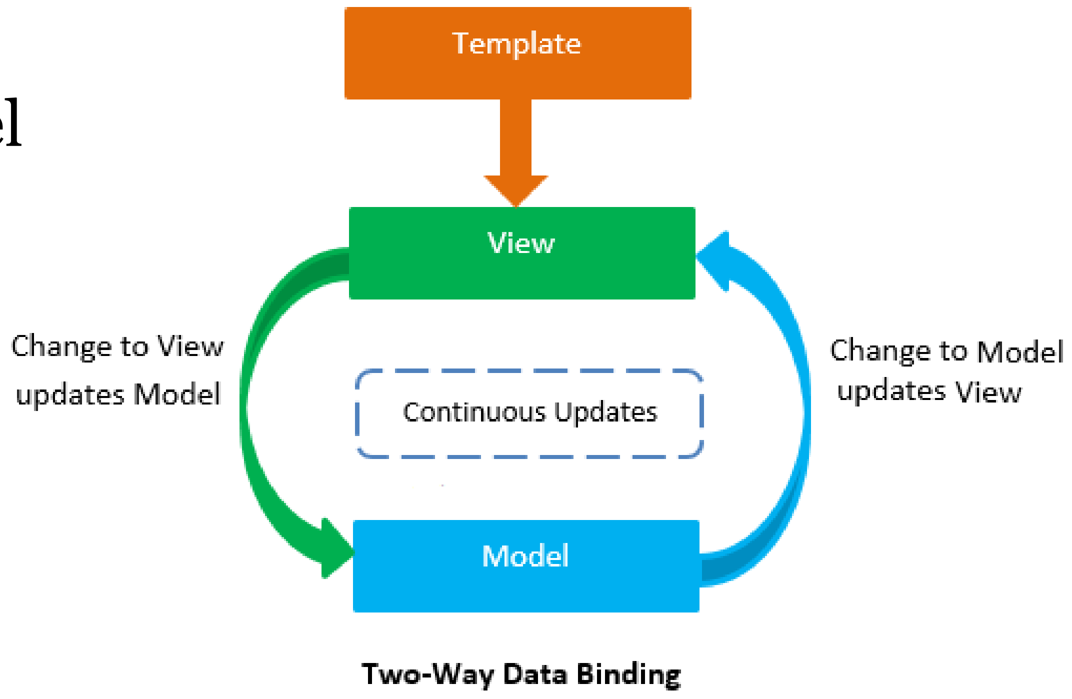




2 way Data Binding

- Allows automatic synchronization of the data between the model and the view

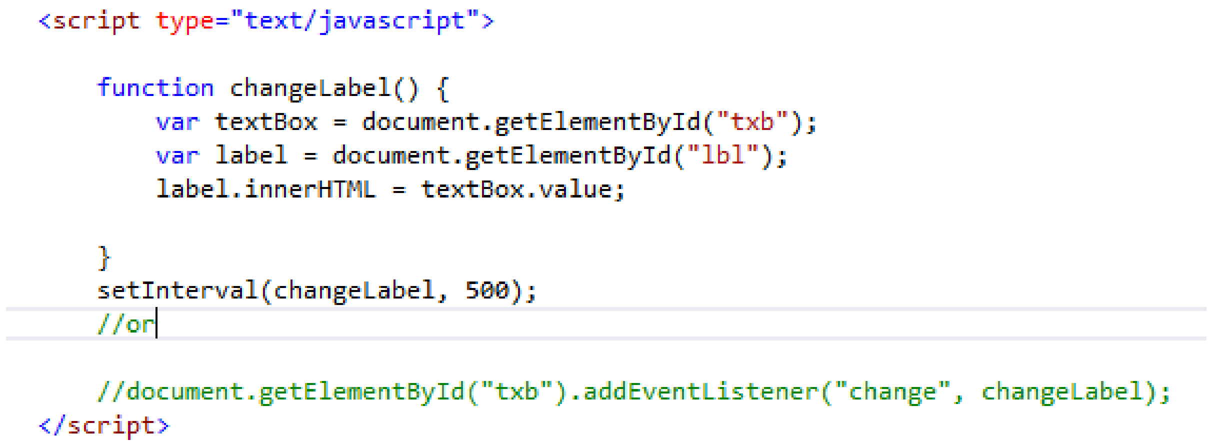
- When model changes, the view reflects the change, and vice versa (two way binding).



CodesJava.com

Data Binding <u>Without</u> Angular

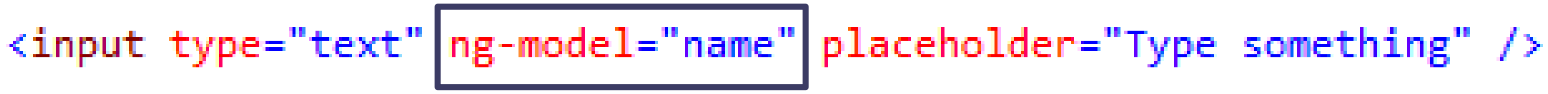
- We need to use getters and setters from and to DOM elements.
- We need to handle the changing events.
- This is for just one field !



```
<script type="text/javascript">

    function changeLabel() {
        var textBox = document.getElementById("txb");
        var label = document.getElementById("lbl");
        label.innerHTML = textBox.value;

    }
    setInterval(changeLabel, 500);
    //or

    //document.getElementById("txb").addEventListener("change", changeLabel);
</script>
```

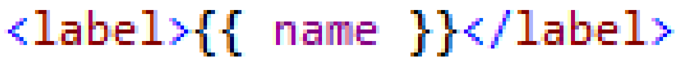
Data Binding With Angular

- We use the ‘ng-model’ to create model on the view that allows binding to the model (data).



```
<input type="text" ng-model="name" placeholder="Type something" />
```

- We use expressions to evaluate the model on the view



```
<label>{{ name }}</label>
```

Directives

- It's a marker on the HTML tag and tells Angular to run or reference some JavaScript code
- AngularJS:
  - lets you extend HTML with new attributes called **Directives**
  - has a set of built-in directives which offers functionality
  - AngularJS also lets you define your own directives
  - directives are extended HTML attributes with the prefix **ng-**

The most basic

The AngularJS framework can be divided into following three major parts –

- **ng-app** – directive that initializes an AngularJS application
- **ng-model** – directive that binds the value of HTML controls (input, select, textarea) to application data
- **ng-bind** – directive that initializes application data

Directives - Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

<body>

    <div ng-app="">
        <p>Name:
            <input type="text" ng-model="name">
        </p>
        <p ng-bind="name"></p>
    </div>

</body>

</html>
```

AngularJS starts automatically when the web page has loaded

The **ng-app** directive tells AngularJS that the <div> element is the "owner" of an AngularJS **application -** only allowed to work on, control, and modify that particular section of the HTML

The **ng-model** directive binds the value of the input field to the application variable **name**

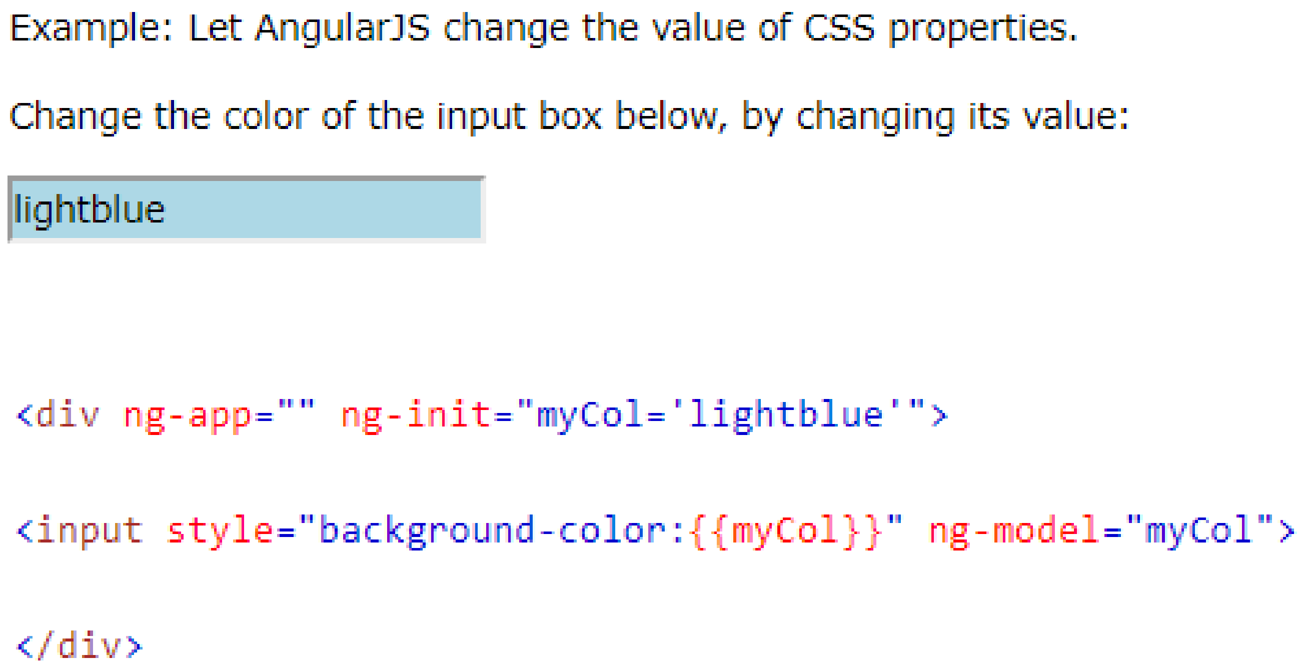
The **ng-bind** directive binds the innerHTML of the <p> element to the application variable **name**.

https://www.w3schools.com/angular/tryit.asp?filename=try_ng_intro

Expressions

- AngularJS binds data to HTML using **Expressions**
- can be written inside double braces: {{ expression }}
- AngularJS expressions can also be written inside a directive: ng-bind="expression“
- AngularJS will resolve the expression, and return the result exactly where the expression is written

Expressions - Example



Example: Let AngularJS change the value of CSS properties.

Change the color of the input box below, by changing its value:

lightblue

```
<div ng-app="" ng-init="myCol='lightblue'">

<input style="background-color:{{myCol}}" ng-model="myCol">

</div>
```

https://www.w3schools.com/angular/tryit.asp?filename=try_ng_expression_3

Expressions - Example

- **Objects:**

```
<div ng-app="" ng-init="person={firstName:'John',lastName:'Doe'}">
 <p>The name is {{ person.lastName }}</p>
  <p>The name is <span ng-bind="person.lastName"></span></p>
</div>
```

- **Arrays:**

```
<div ng-app="" ng-init="points=[1,15,19,2,40]">
 <p>The third result is {{ points[2] }}</p>
</div>
```

- **For More examples:**

  https://www.w3schools.com/angular/angular_expressions.asp

Module

- An AngularJS module defines an application
- Modules are AngularJS’s way of packaging relevant code under a single name (package)
- The module is a container for the application controllers, services etc.
- The module can also depend on other modules as dependencies ([])

Module

- A module is created by using the AngularJS function **angular.module**

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

- The first argument is the *name* of the module
- The second argument is an *array* of module names that this *module depends on*

Now you can add controllers, services and more, to your AngularJS application

Module

Creating a module

```
let app = angular.module('myApp', []);
```

app.js file

module as container

```
app.controller("myCtrl", [function(){

  ...

}])
```

myCtrl.js file

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/
angularjs/1.6.9/angular.min.js"></script>

<script src="myCtrl.js"></script>
<script src="app.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">
</div>

</body>

</html>
```

HTML file

Controller

- Fetching the right data from the server for the current UI
- Deciding which parts of that data to show to the user Presentation logic, such as how to display elements, which parts of the UI to show, how to style them, etc.
- User interactions, such as what happens when a user clicks something or how a text input should be validated
- Gateway between the Model (data) and the View (UI)
- We use it through the 'ng-controller' directive.

Controller

- Creating the controller is done by attaching it to our module

```
angular.module('myApp')
.controller('MainCtrl', [function() {
    // Controller-specific code goes here
    console.log('MainCtrl has been created');
}]);
```

- Using through the ‘ng-controller’ directive.

```
<div ng-app="myApp">
<head> <title>Notes App</title> </head>
<body ng-controller="MainCtrl as ctrl">
        // View-specific code goes here
</body>
</div>
```

Controller

controller1

- We will use the : *controller as syntax*, and a proxy name (*self*) *for this keyword*

index.html

```
<body ng-controller="MainCtrl as ctrl">
    {{ctrl.helloMsg}} AngularJS.
```

MainCtrl.js

```
app.controller('MainCtrl', [function() {
    let self = this
    self.helloMsg = 'Hello ‘;
    …
}])
```

Controller - Example

controller2

- The controller we wrote has no direct access to the view or any of the DOM elements that it needs to update. It is pure JavaScript. Creating Our First Controller
- When the user clicked the button and changeMessage was triggered, we did not have to tell the UI to update. It happened automatically.
- The HTML connects parts of the DOM to controllers, functions, and variables, and not the other way around.
- our whole aim in an AngularJS application should be to manipulate and modify the model (pure JavaScript), and let AngularJS do the heavy lifting of updating the UI accordingly

Built in Directives

- ng-repeat
  - one of the most versatile and heavily used directives of AngularJS, because it allows us to iterate over an array or over the keys and values of an object and display them in the HTML
  - the contents of the element on which the directive is applied is considered the template of the ng-repeat.
  - AngularJS picks up this template, makes a copy of it, and then applies it for each instance of the ng-repeat

Built in Directives

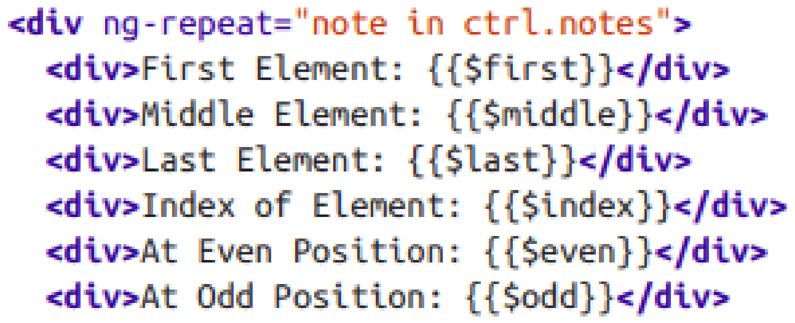
Array + object

- Notes on ng-repeat
  - For arrays use: `ng-repeat="note in ctrl.notes"`
  - For objects use: `ng-repeat="(key, value) in ctrl.notes"`
  - When we use the ng-repeat directive over an object, the keys of the object will be sorted in a case-sensitive, alphabetic order (Upper-case first)

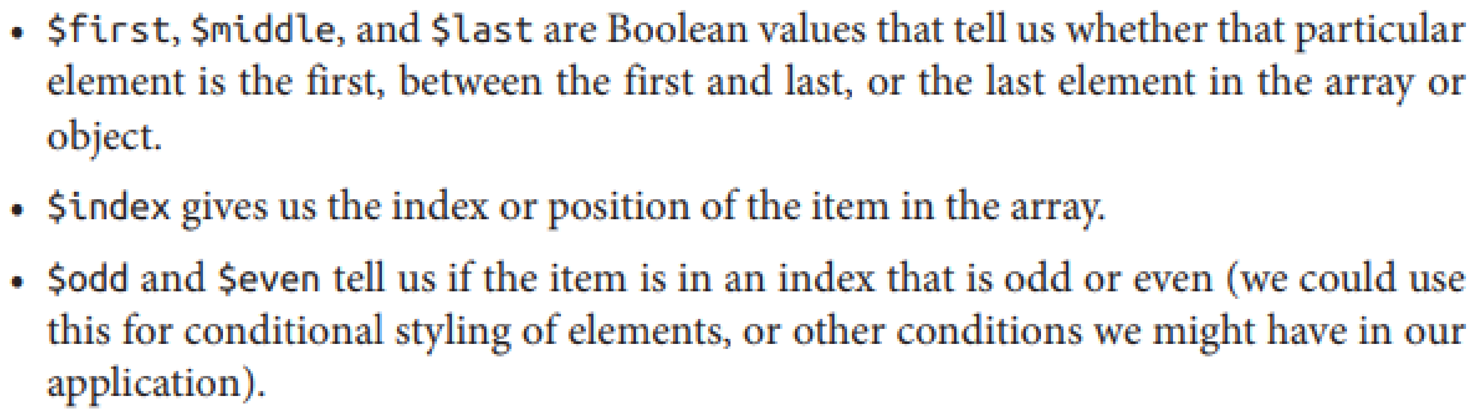
Built in Directives

helpers

- Notes on ng-repeat



```
<div ng-repeat="note in ctrl.notes">
  <div>First Element: {{$first}}</div>
  <div>Middle Element: {{$middle}}</div>
  <div>Last Element: {{$last}}</div>
  <div>Index of Element: {{$index}}</div>
  <div>At Even Position: {{$even}}</div>
  <div>At Odd Position: {{$odd}}</div>
```



- `$first`, `$middle`, and `$last` are Boolean values that tell us whether that particular element is the first, between the first and last, or the last element in the array or object.
- `$index` gives us the index or position of the item in the array.
- `$odd` and `$even` tell us if the item is in an index that is odd or even (we could use this for conditional styling of elements, or other conditions we might have in our application).

Lab Exercise

- הוסיפו לקובץ הcontroller אובייקט של ערים, החזיקו עבור כל עיר: שם, מדינה, לינק לתמונה
- חברו את הcontroller לview, והציגו באמצעות ng-repeat אלמנט של פסקה עבור כל עיר, כך שבתוך הפסקה יופיעו פרטי העיר והתמונה

Any Questions?

And I'll come shortly..

Internet Programming

Practice #11

AngularJS - Lab 2

Dept. of Software and Information Systems Engineering

by Shir Frumerman Base on Hasidi Netanel Slides

Today

- Some more built-in directives
- Routing
- Forms

Built in Directives

- AngularJS built-in directives list and examples :

https://www.w3schools.com/angular/angular_ref_directives.asp

Built in Directives

- ng-show / ng-hide

  Used to show or hide a given element based on expressions, booleans and functions

  Examples on : scotch.io/tutorials/how-to-use-ngshow-and-nghide

- ng-click

  Used to fire a method or expression when element is clicked.

Built in Directives

- ng-options

  The ng-options directive fills a <select> element with <options>.

  The ng-options directive uses an array to fill the dropdown list.

```
<select ng-model="selectedName" ng-options="item for item in names"></select>
```

Routing

- ngRoute - handle routing
- In single page app url is called *hashbang* URL
- Traditional URL: http://www.myApp.com/first/page
- Hashbang URL: http://www.myApp.com/#/first/page
- When the hash fragment changes, the JavaScript responds and loads only the relevant data and HTML – faster app

Routing

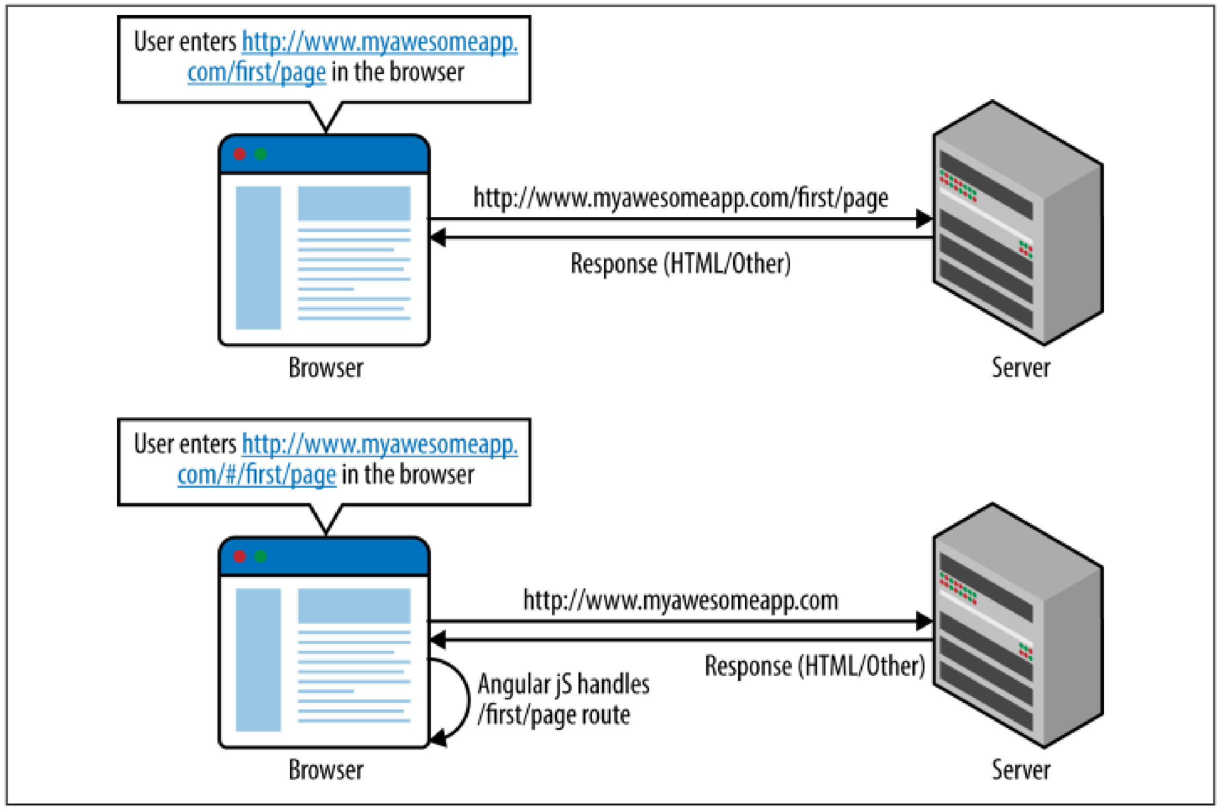

User enters http://www.myawesomeapp.com/first/page in the browser
http://www.myawesomeapp.com/first/page
Response (HTML/Other)
Browser
Server
User enters http://www.myawesomeapp.com/#/first/page in the browser
http://www.myawesomeapp.com
Response (HTML/Other)
Angular jS handles /first/page route
Browser
Server

Figure 10-1. Flow of normal URLs versus hash URLs

Taken from : AngularJS: Up And Running by Shyam Seshadri and Brad Green

Routing - set up

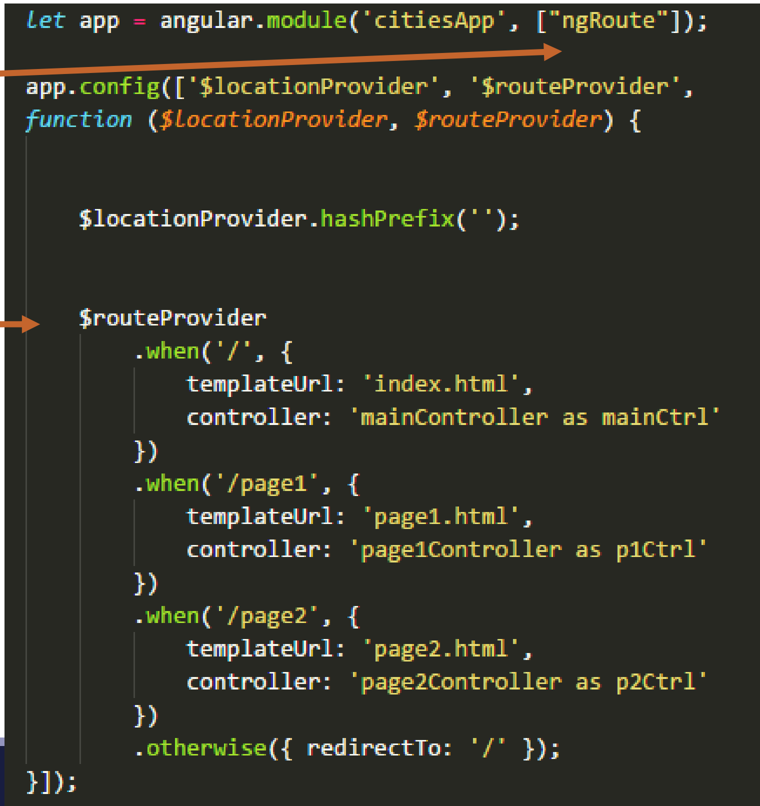
- AngularJS routing is not part of the core library
  Include in index.html :

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-route.js"></script>
```

Include the module as a dependency of our main AngularJS app module :

Define our routes in the config section using the $routeProvider service

- Define for each view it's controller



```
let app = angular.module('citiesApp', ["ngRoute"]);

app.config(['$locationProvider', '$routeProvider',
function ($locationProvider, $routeProvider) {


    $locationProvider.hashPrefix('');


    $routeProvider
        .when('/', {
            templateUrl: 'index.html',
            controller: 'mainController as mainCtrl'
        })
        .when('/page1', {
            templateUrl: 'page1.html',
            controller: 'page1Controller as p1Ctrl'
        })
        .when('/page2', {
            templateUrl: 'page2.html',
            controller: 'page2Controller as p2Ctrl'
        })
        .otherwise({ redirectTo: '/' });
}]);
```

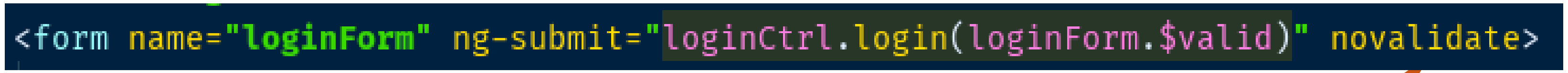
Routing - ng-view

- ng – view : Mark which section of the page AngularJS should change when the route changes
- AngularJS application that uses ngRoute, there can be one and only one ng-view directive for that application
- Only the content inside the ng-view tags will be changed

```
<div ng-view></div>
```

Working with Forms

- heavily leverage the ng-model directive to get our data into and out of the form
- We use the 'ng-submit' directive to trigger the submission functionality.
- Handle forms states and validate easily



```
<form name="loginForm" ng-submit="loginCtrl.login(loginForm.$valid)" novalidate>
```

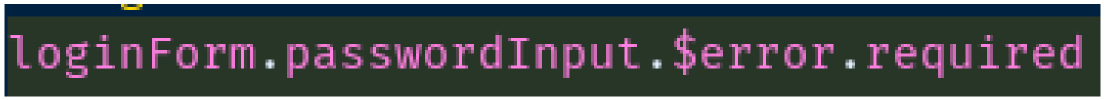
This will prevent the default HTML5 validations (since we'll be doing that ourselves)

Form States

- AngularJS creates a FormController that holds the current state of the form as well as some helper methods
- Access the FormController for a form using the form’s name
- Each of the states (except $error) are Booleans and can be used to conditionally hide, show, disable, or enable HTML elements in the UI

Form States

- $valid: True if a form item is valid.
- $invalid: The opposite of the above.
- $pristine: True if the form/input **has not** been used yet.
- $dirty: True if the form/input has been used.
- $touched: True if an input has lost focus.
- $error: Expression that checks if an input has an error.

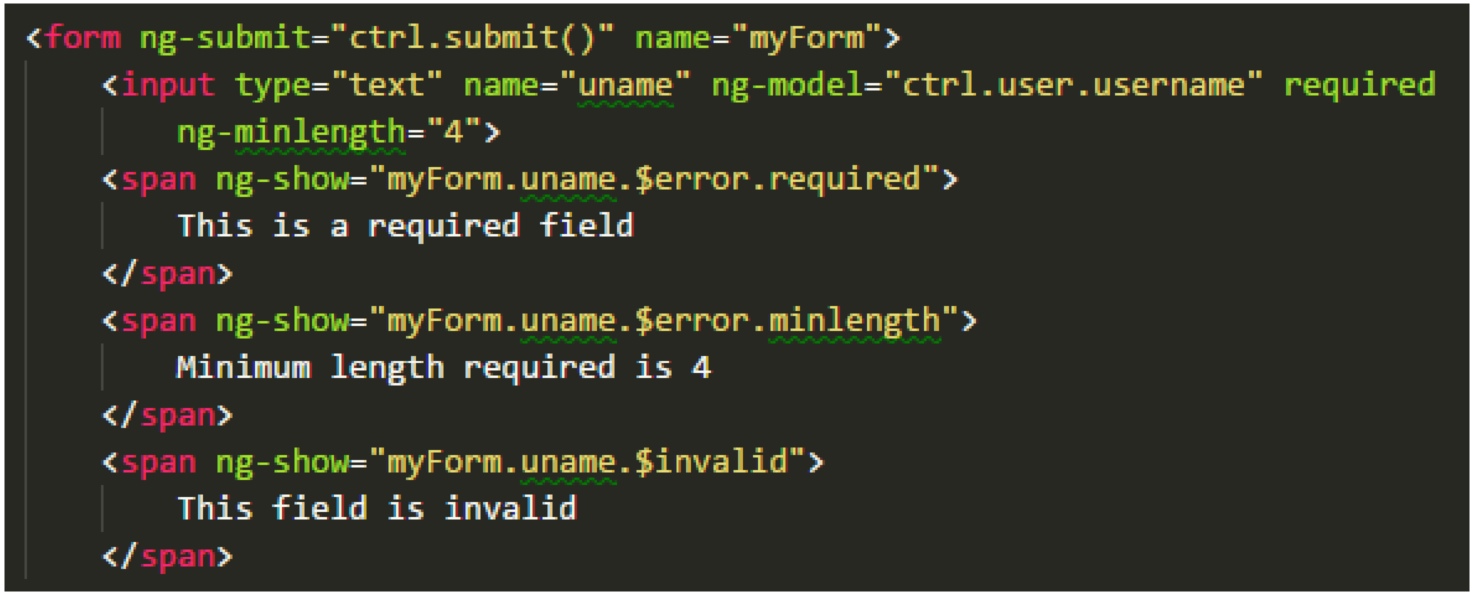


```
loginForm.passwordInput.$error.required
```

Form Directives

- ng-required: required by expression
- ng-disabled: disable the submit button when our form is not valid.
- ng-minlength: controls the min length of the input.
- ng-maxlength: controls the max length of the input.
- ng-pattern: regular expression validation.

Form - Displaying Error Messages



```
<form ng-submit="ctrl.submit()" name="myForm">
    <input type="text" name="uname" ng-model="ctrl.user.username" required
        ng-minlength="4">
    <span ng-show="myForm.uname.$error.required">
        This is a required field
    </span>
    <span ng-show="myForm.uname.$error.minlength">
        Minimum length required is 4
    </span>
    <span ng-show="myForm.uname.$invalid">
        This field is invalid
    </span>
```

Lab Excersice

1. Open “routing“ file in Examples folder in VSCode
2. Add a new button in the navigation bar to Point of interests page
3. Add the sol form EX1 to the routing directory (component/POI/)
4. Clean the redundant code, edit all the files in order to combine the new files to the existed ones
5. Add a new button in the navigation bar to : Register
6. Create a form with validation according to our register form of the project
7. Add register controller that log to console the form details
8. Set the routing to the form view and controller

Any Questions?

And I’ll come shortly..

Internet Programming

AngularJS - Lab 3

Dept. of Software and Information Systems Engineering

By Shir Frumerman Based on Hasidi Netanel slides

Some Notes

- Head Controller in index.html
- Different inputs with ng-repeats
- Developer Tool

- If a controller needs to store some state for the entire application, it should be stored
  in a service, and not $rootScope. Never $rootScope.

Services - Why?

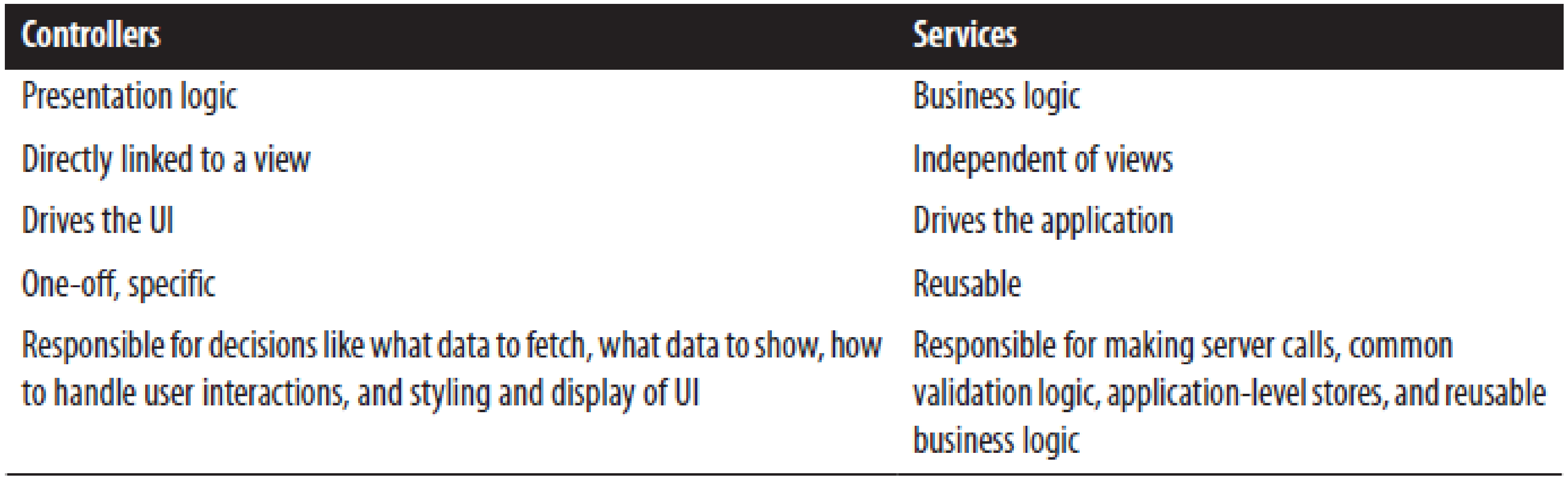
- Controllers:
  - instances that get created and destroyed as we navigate across our application
  - one controller cannot directly communicate with another controller to share state or behavior

Services

- AngularJS services are functions or objects that can hold behavior across our application
- Each AngularJS service is instantiated only once (so each part of our application gets access to the same instance of the AngularJS service) - Singeltons

Services vs. Controllers

- Controllers responsible to control the presentation logic and handle user interaction.
- Services responsible to perform global and more big functionality.



Controllers	Services
Presentation logic	Business logic
Directly linked to a view	Independent of views
Drives the UI	Drives the application
One-off, specific	Reusable
Responsible for decisions like what data to fetch, what data to show, how to handle user interactions, and styling and display of UI	Responsible for making server calls, common validation logic, application-level stores, and reusable business logic

Built-in Services

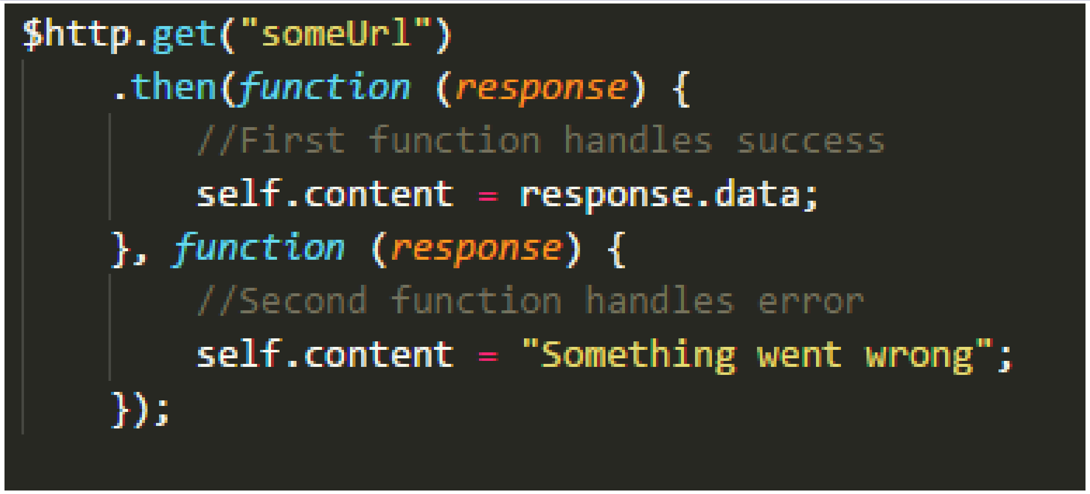
- AngularJS prefixes all the services that are provided by the AngularJS library with the ‘$’ sign

- Examples:
  - $scope
  - $rootScope

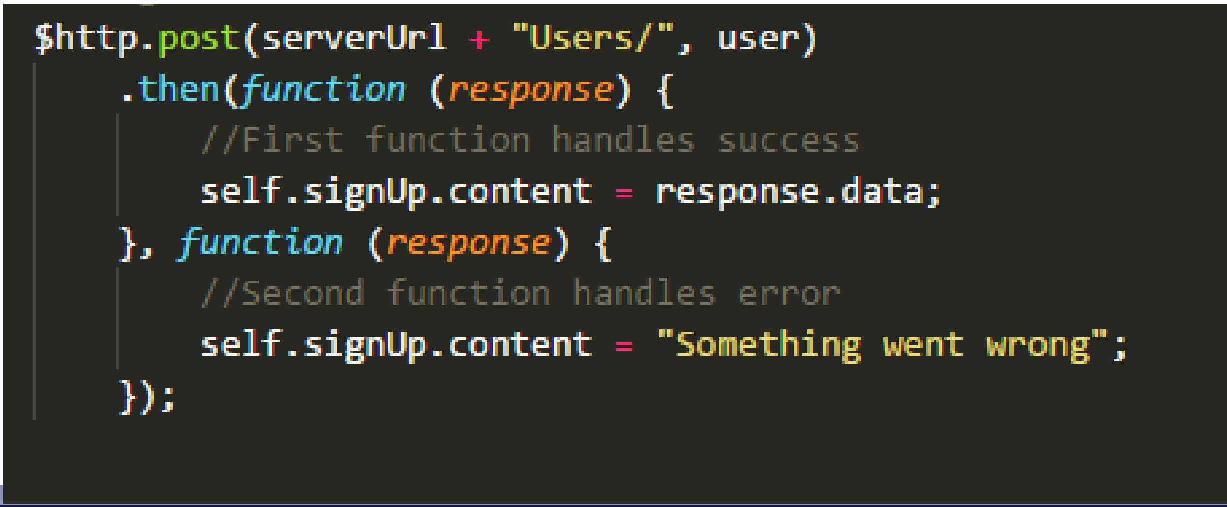
  - $location – $location.path() for hashbang url
    Allows us to interact with the URL in the browser bar, and get and manipulate its value

  - $http
    Make requests to the server from the application

$http Service



```
$http.get("someUrl")
    .then(function (response) {
        //First function handles success
        self.content = response.data;
    }, function (response) {
        //Second function handles error
        self.content = "Something went wrong";
    });
```

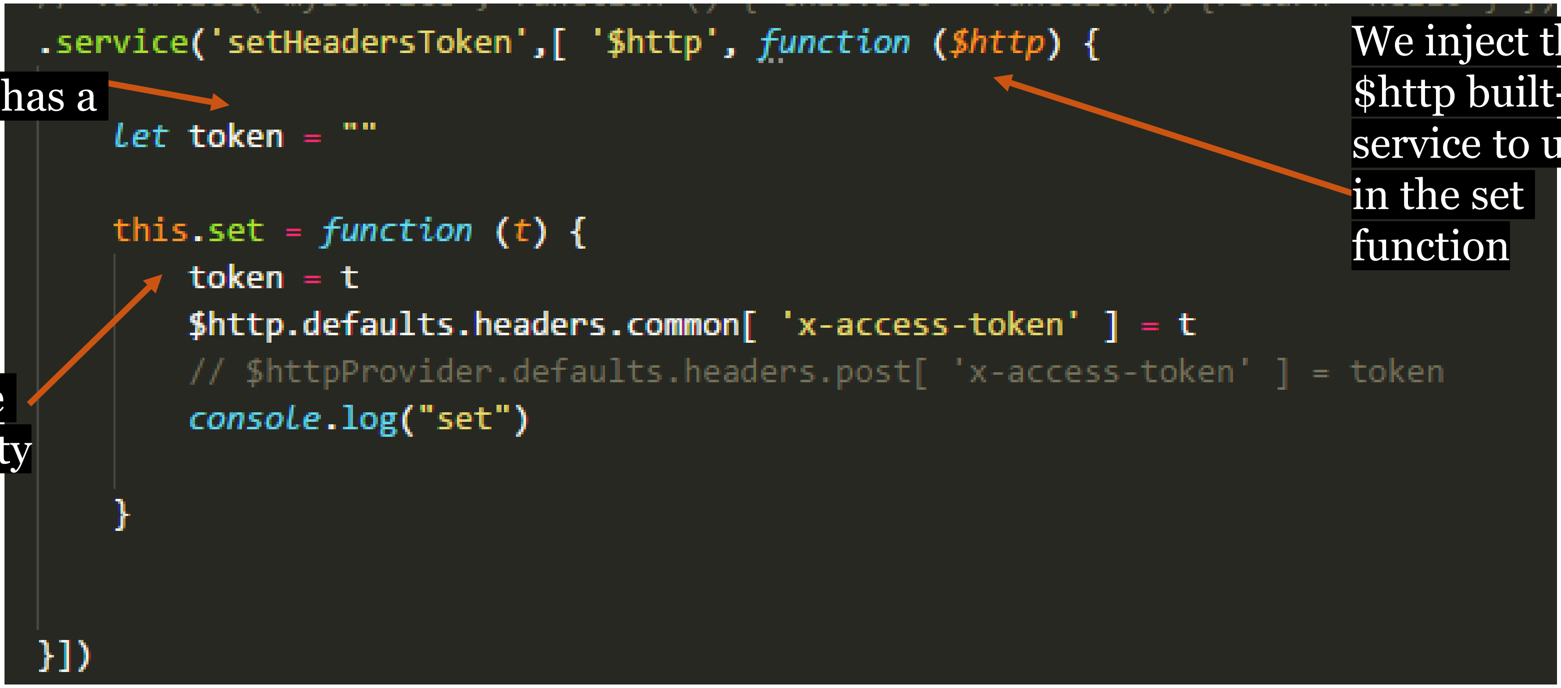


```
$http.post(serverUrl + "Users/", user)
    .then(function (response) {
        //First function handles success
        self.signUp.content = response.data;
    }, function (response) {
        //Second function handles error
        self.signUp.content = "Something went wrong";
    });
```

Services (Custom)

- We can create our own services
- Holds :
  - *It needs to be reusable-*
    More than one controller or service will need to access the particular function that is being implemented.
  - *Application-level state*
    Controllers get created and destroyed. If we need state stored across our application, it belongs in a service.
  - *It is independent of the view*
    If what we are implementing is not directly linked to a view, it probably belongs in a service.

Services (Custom)- Example



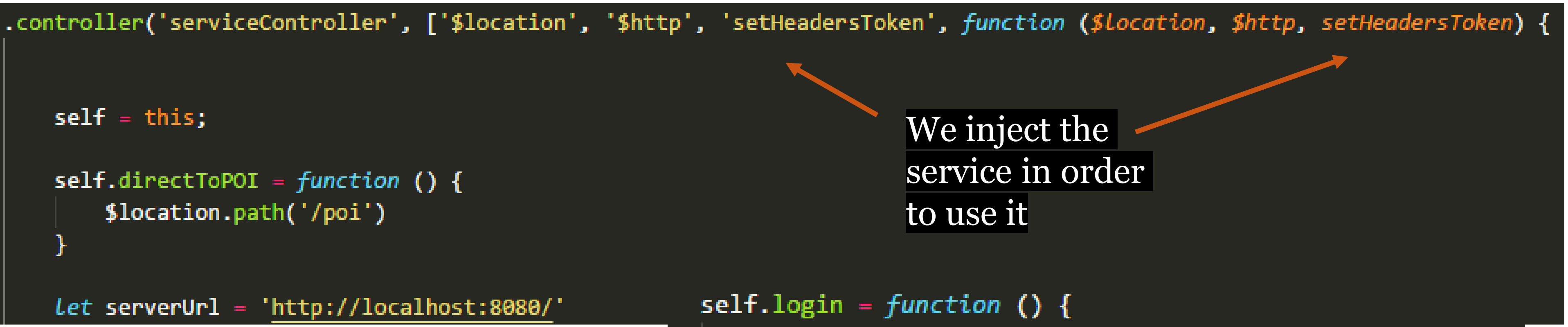
```
.service('setHeadersToken',[ '$http', function ($http) {

    let token = ""

    this.set = function (t) {
        token = t
        $http.defaults.headers.common[ 'x-access-token' ] = t
        // $httpProvider.defaults.headers.post[ 'x-access-token' ] = token
        console.log("set")

    }


}])
```

Our service has a property

We inject the $http built-in service to use it in the set function

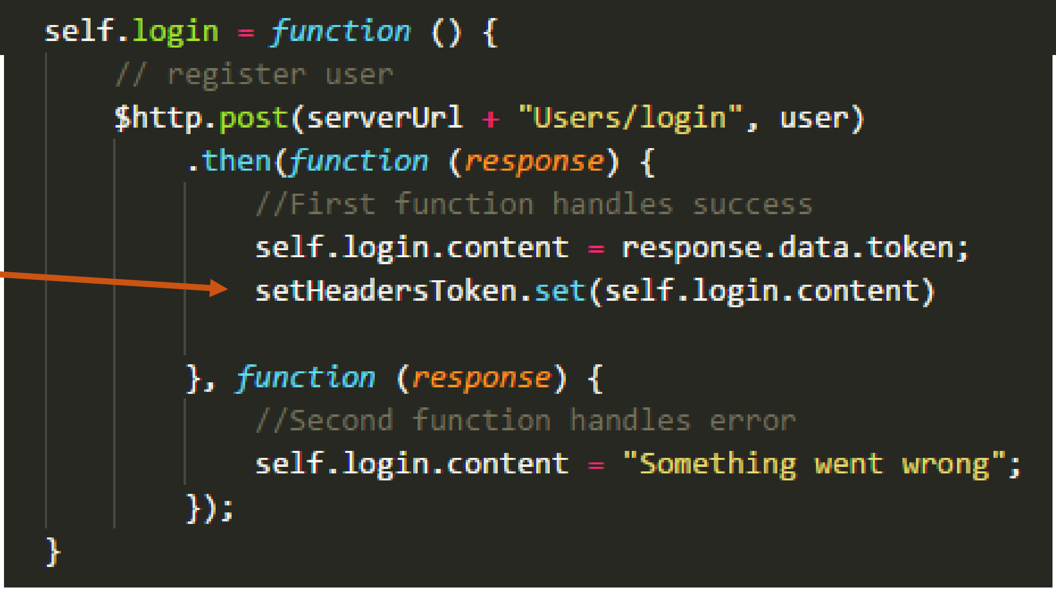
Our service functionality

Services (Custom)- Example



```
.controller('serviceController', ['$location', '$http', 'setHeadersToken', function ($location, $http, setHeadersToken) {


    self = this;

    self.directToPOI = function () {
        $location.path('/poi')
    }

    let serverUrl = 'http://localhost:8080/'
```

We inject the service in order to use it



```
self.login = function () {
    // register user
    $http.post(serverUrl + "Users/login", user)
        .then(function (response) {
            //First function handles success
            self.login.content = response.data.token;
            setHeadersToken.set(self.login.content)

        }, function (response) {
            //Second function handles error
            self.login.content = "Something went wrong";
        });
}
```

Use the service funcionality

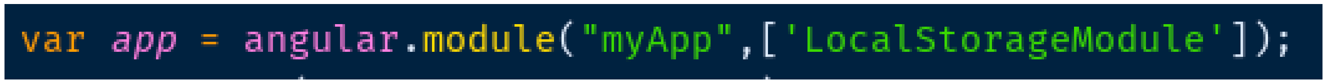
Local-Storage Service

- Allows us to save data within the browser.
- Save data objects in json format.
- Save cookies.
- Load previous session data when the app starts.

Local-Storage Service

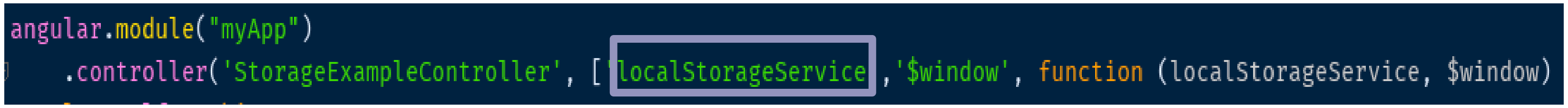
- Local-Storage service is not part of the core angular module, so we need to download it:
  npm and add in the index.html file:
  <script src="node_modules/angular-local-storage/dist/angular-local-storage.min.js"></script>

- Using the module:



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

- Inject the service wherever we want to use it (controller, custom service).



```
angular.module("myApp")
    .controller('StorageExampleController', ['localStorageService','$window', function (localStorageService, $window)
```

Local-Storage Service (Methods)

- Gets a value by key from the local storage
  localStorageService.get(key)
- Add key-value pair to the local storage
  localStorageService.set(key,value)
- Get an array of current stored keys
  localStorageService.keys()
- Remove an item by key
  localStorageService.remove(key)
- Remove all the items for this app
  localStorageService.clearAll()

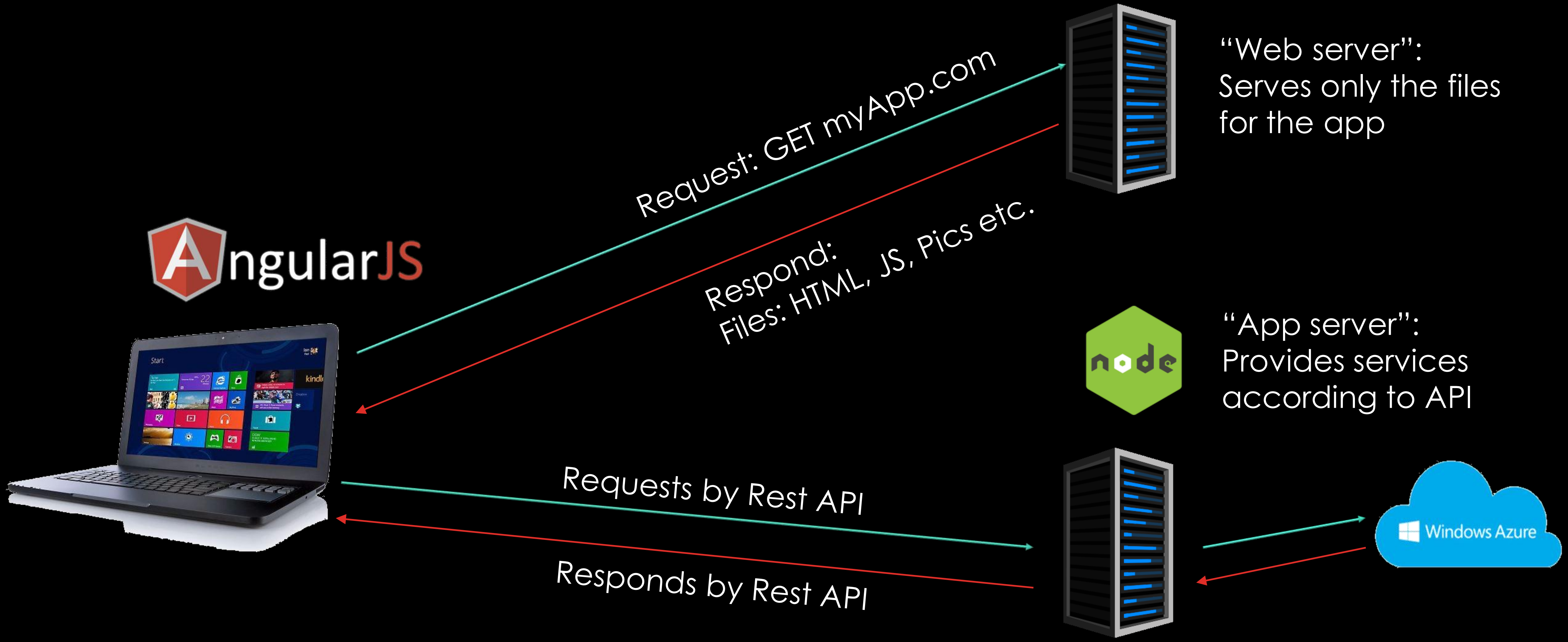
Any Questions?

And I’ll come shortly..

ANGULARJS
NOTES, TIPS AND BEST PRACTICE

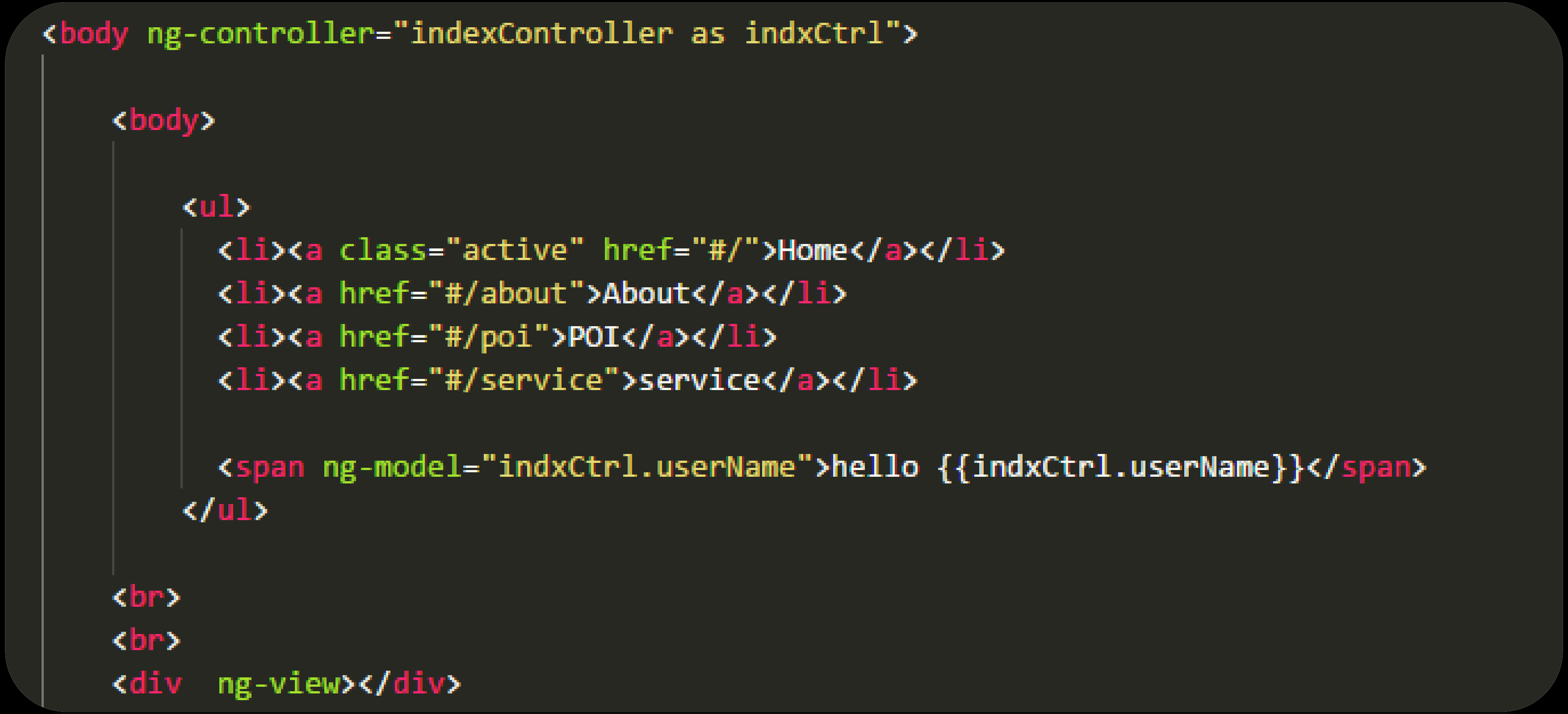
By Shir Frumerman

CLIENT-SERVER “ARCHITECTURE”



THE MAIN CONTROLLER

- The main HTML frame of the application should also have controller
- Routing between views and controllers in app only effects the ng-view scope
- We will define another controller in the body tag to control elements outside the ng-view
- Example for usage: presenting the username in the upper page bar



```html
<body ng-controller="indexController as indxCtrl">

    <body>

        <ul>
          <li><a class="active" href="#/">Home</a></li>
          <li><a href="#/about">About</a></li>
          <li><a href="#/poi">POI</a></li>
          <li><a href="#/service">service</a></li>

          <span ng-model="indxCtrl.userName">hello {{indxCtrl.userName}}</span>
        </ul>

    <br>
    <br>
    <div  ng-view></div>
```

HOW CAN WE CHANGE DATA-MODEL IN THE MAIN CONTROLLER?

- The main controller is a parent controller to all the other controllers, Why? Because it's declare on the body, that contains ng-view

- The easiest way to accesses/change data-models of a parent controller:

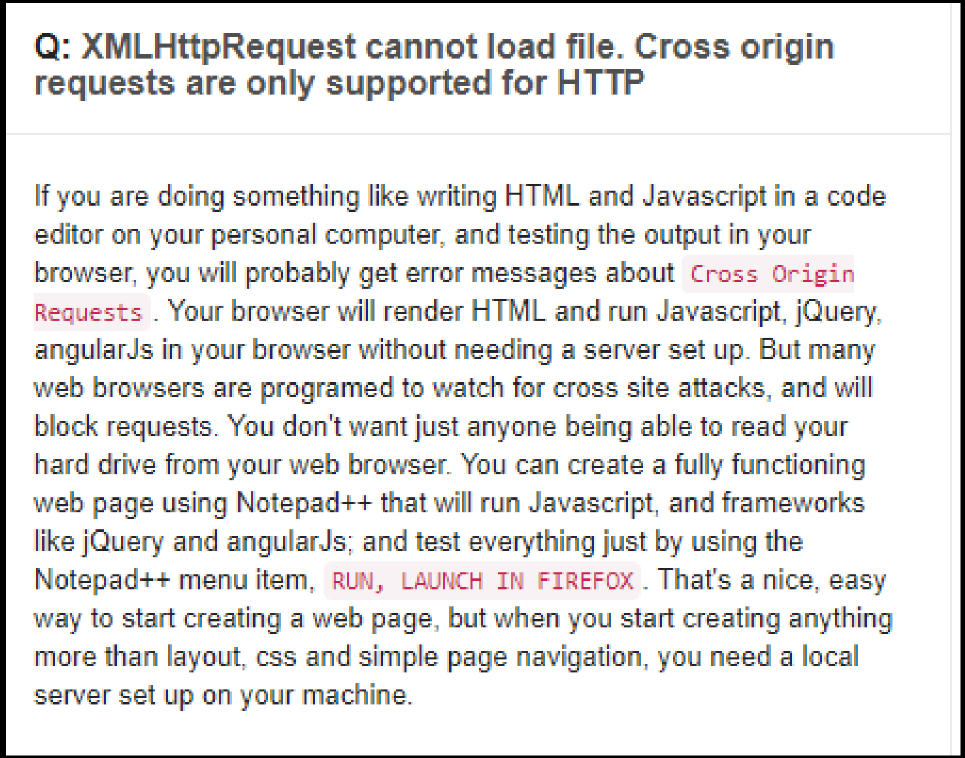
  - If :
    ```
    <body ng-controller="indexController as indxCtrl">
    <span ng-model="indxCtrl.userName">hello {indxCtrl.userName}}</span>
    ```

  - Inside child controller use : don't forget to inject $scope as dependency!!
    ```
     $scope.indxCtrl.userName= "Changed!
    ```

  - `THE SYNTAX: $SCOPE.[PRENT-CTRL-NAME].[MODEL-NAME]=[SOME-VALUE]`

HOW TO RUN/WORK WITH THE CLIENT

- We cannot open the app directly from html file on our computer, why?



Q: XMLHttpRequest cannot load file. Cross origin requests are only supported for HTTP

If you are doing something like writing HTML and Javascript in a code editor on your personal computer, and testing the output in your browser, you will probably get error messages about `Cross Origin Requests`. Your browser will render HTML and run Javascript, jQuery, angularJs in your browser without needing a server set up. But many web browsers are programed to watch for cross site attacks, and will block requests. You don't want just anyone being able to read your hard drive from your web browser. You can create a fully functioning web page using Notepad++ that will run Javascript, and frameworks like jQuery and angularJs; and test everything just by using the Notepad++ menu item, `RUN, LAUNCH IN FIREFOX`. That's a nice, easy way to start creating a web page, but when you start creating anything more than layout, css and simple page navigation, you need a local server set up on your machine.

HOW TO RUN/WORK WITH THE CLIENT

- So we will use local-server to serve our files and deal with the cross-origin problem (all the connected files will server to your browser) :
  - If you wan your client to update automatically while working (like nodemon) use:
    - “Lite-server”: Download from npm, run from terminal
    - “Live-server” : Extension for Vscode, run form UI
    - Probably there is extension for web storm, google it
  - For running your app after development:
    - “http-server”: Download from npm, run from terminal

WHAT ARE ANGULARJS FILTERS?

- Used to process data and format values to present to the user
- Applied on expressions in our HTML, or directly on data in our controllers and services
- Mostly, they are used as that final level of formatting to convert data from the way it is stored to a user-readable forma :
    - Take a timestamp and make it human-readable
    - Add the currency symbol to a number
    - Show string with upper case
    - And more

SOME SYNTAX BEFORE EXAMPLES

- The general syntax to use filters is to use the Unix syntax of piping the result of one expression to another :

```
{{expression | filter}}
```

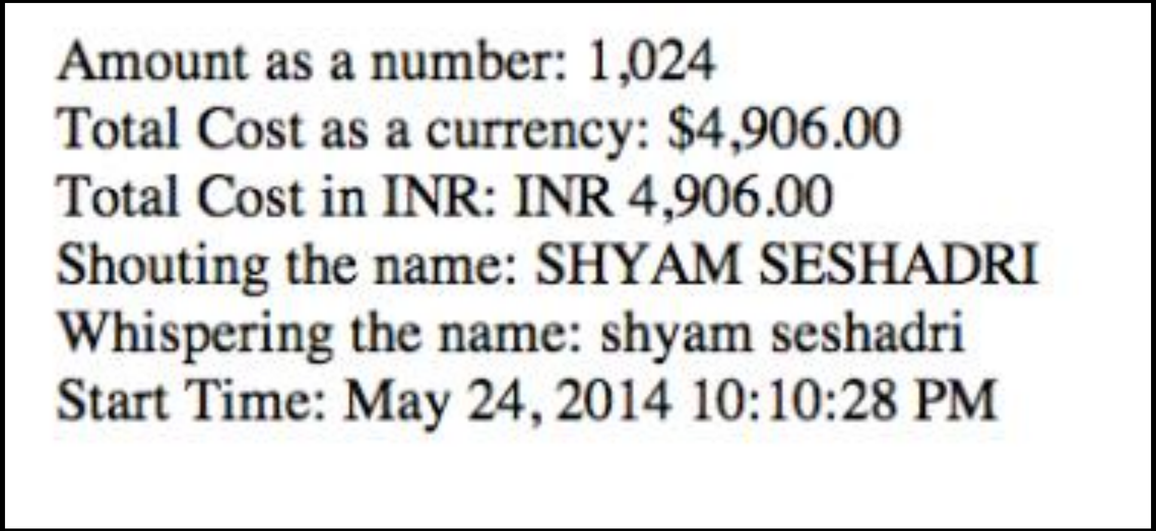
- The filter will take the value of the expression (a string, number, or array) and convert it into some other form

- We can also chain multiple filters together by piping one filter after another. The syntax would be:

```
{{expression | filter1 | filter2}}
```

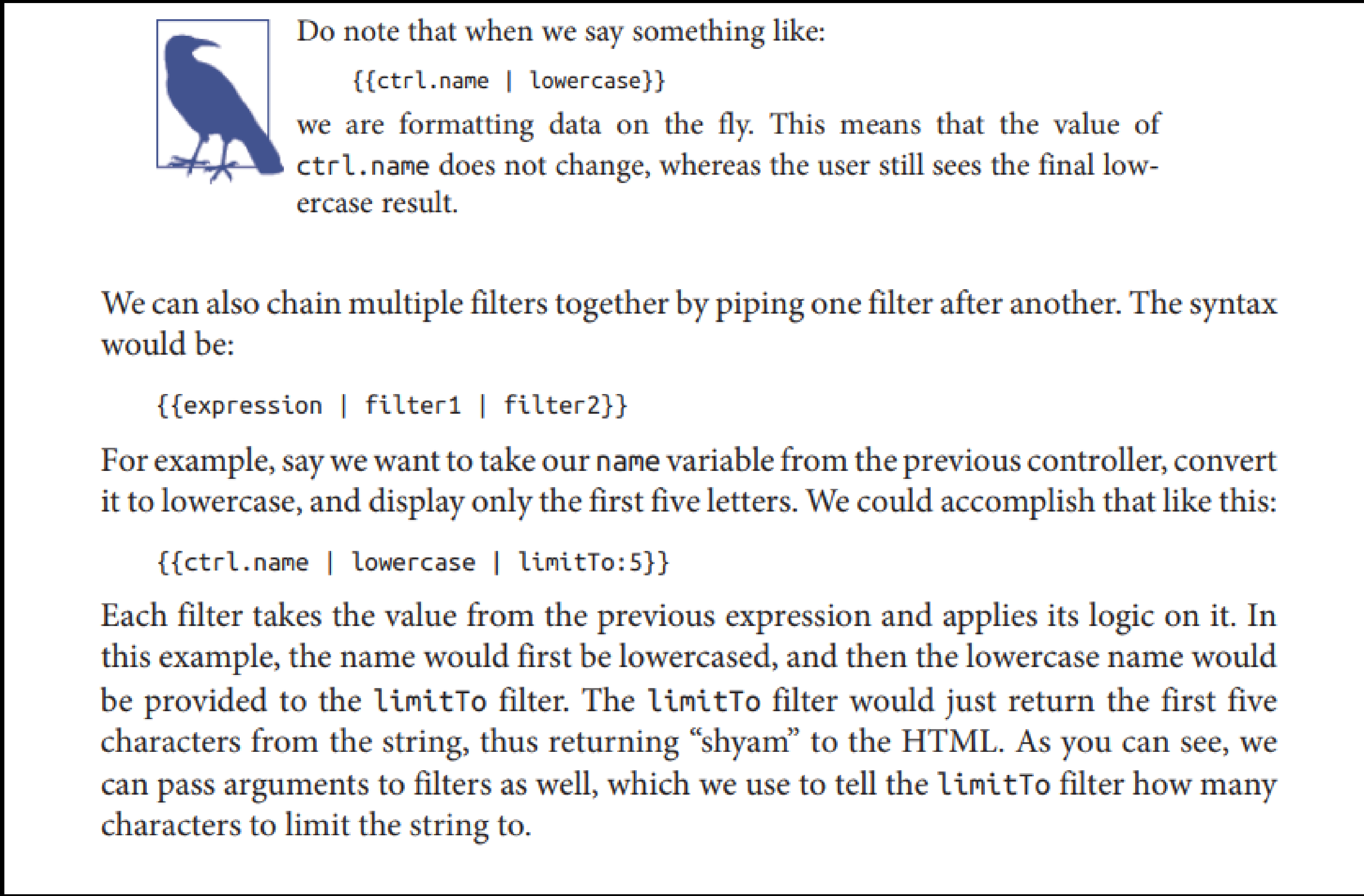
```html
<html>
<head>
  <title>Filters in Action</title>
</head>
<body ng-app="filtersApp">

<div ng-controller="FilterCtrl as ctrl">
  <div>
    Amount as a number: {{ctrl.amount | number}}
  </div>
  <div>
    Total Cost as a currency: {{ctrl.totalCost | currency}}
  </div>
  <div>
    Total Cost in INR: {{ctrl.totalCost | currency:'INR '}}
  </div>
  <div>
    Shouting the name: {{ctrl.name | uppercase}}
  </div>
  <div>
    Whispering the name: {{ctrl.name | lowercase}}
  </div>
  <div>
    Start Time: {{ctrl.startTime | date:'medium'}}
  </div>
</div>

<script
  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>

<script type="text/javascript">
  angular.module('filtersApp', [])
    .controller('FilterCtrl', [function() {
      this.amount = 1024;
      this.totalCost = 4906;
      this.name = 'Shyam Seshadri';
      this.startTime = new Date().getTime();
    }]);
</script>
</body>
</html>
```

FOR EXAMPLE
SOME BUILT-IN FILTERS



```
Amount as a number: 1,024
Total Cost as a currency: $4,906.00
Total Cost in INR: INR 4,906.00
Shouting the name: SHYAM SESHADRI
Whispering the name: shyam seshadri
Start Time: May 24, 2014 10:10:28 PM
```

MORE EXAMPLE



Do note that when we say something like:

```
{{ctrl.name | lowercase}}
```

we are formatting data on the fly. This means that the value of `ctrl.name` does not change, whereas the user still sees the final lowercase result.

We can also chain multiple filters together by piping one filter after another. The syntax would be:

```
{{expression | filter1 | filter2}}
```

For example, say we want to take our `name` variable from the previous controller, convert it to lowercase, and display only the first five letters. We could accomplish that like this:

```
{{ctrl.name | lowercase | limitTo:5}}
```

Each filter takes the value from the previous expression and applies its logic on it. In this example, the name would first be lowercased, and then the lowercase name would be provided to the `limitTo` filter. The `limitTo` filter would just return the first five characters from the string, thus returning "shyam" to the HTML. As you can see, we can pass arguments to filters as well, which we use to tell the `limitTo` filter how many characters to limit the string to.

WHAT WILL WE USE IN OUR PROJECT

- We are interested in different and very useful feature:
  - when filters are used in the view, they give us dynamic, on-the-fly data that doesn’t need to be stored

What does it means?

IF we would like for example to show a list of items to the user, sorted-by or filtered by some value, **we don’t need to create a new list for that!**
The only thing we need is to use the filter feature on the list and that list will change without adding data to out model!

FILTER & SORT-BY

- Some examples and explanations to how to use filter and sort-by on list (all that you need for the project) you can find here:
  - https://www.w3schools.com/angular/ng_filter_filter.asp
  - https://www.w3schools.com/angular/ng_filter_orderby.asp
  - https://curtistimson.co.uk/post/angularjs/filtering-arrays-in-angularjs/

FILTER & SORT-BY IMPORTANT EXAMPLE

- Some useful usage is to apply the filter using a model expression (binding to ng-model) , in order to do that you just need to write the name of the model:

  filter : [model-name]

- Run the code in the next slide for example!

FILTER & SORT-BY IMPORTANT EXAMPLE

```
<!DOCTYPE html>
<html>
     <script
     src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

     <body ng-app="myApp">
          <div ng-controller="orderCtrl">
               <ul>
                    <li ng-repeat="x in customers | orderBy : selected">
                         {{x.name + ", " + x.city}}
                    </li>
                         <select ng-options="item for item in ['city', 'name’]”
                         ng-model="selected"></select>
               </ul>
               <p>The array items are sorted by: {{selected}}</p>
          </div>

          <script>
               var app = angular.module('myApp', []);
                    app.controller('orderCtrl', function ($scope) {
                    $scope.customers = [
                    { "name": "Bottom-Dollar Marketse", "city": "Tsawassen" },
                    { "name": "Alfreds Futterkiste", "city": "Berlin" },
                    { "name": "Bon app", "city": "Marseille" },
                    { "name": "Cactus Comidas para llevar", "city": "Buenos Aires" },
                    { "name": "Bolido Comidas preparadas", "city": "Madrid" },
                    { "name": "Around the Horn", "city": "London" },
                    { "name": "B's Beverages", "city": "London" }
                    ];
               });
          </script>
     </body>
</html>
```

Copy and run the example
in your browser

DIRECTORY STRUCTURE BEST PRACTICE

```
app/
--index.html
--app.js
----- shared/   // acts as reusable components (Services)
---------- services/
--------------- setHeader-service.js
----- components/   // each component is treated as a mini Angular app
 ---------- home/
--------------- home-controller.js
--------------- home-service.js
--------------- home-view.html
--------------- home-css.css
---------- products/
--------------- products-controller.js
--------------- products-service.js
--------------- products-view.html
assets/
----- img/      // Images and icons for your app
----- js/       // JavaScript files written for your app that are not for angular
----- node-modules/
```

- Have one controller, service, directive, or filter per file. Don’t club them into large single files.

- Don’t have one giant module. Break up your applications into smaller modules. Let your main application be composed of multiple smaller, reusable modules.

- Prefer to create modules and directories by functionality (authorization, admin-services, search, etc.), over type (controllers, services, directives). This makes your code more reusable.

- Use namespaces. Namespace your modules, controllers, services, directives, and filters. Have *mycompany*-chart for directives, *myProject*AuthService, and so on. That way, anyone coming in can quickly distinguish your own code from thirdparty and core AngularJS code.

- Most of all, be consistent. This is not AngularJS-specific, but don’t change the way you name files, folders, directives, or controllers from one place in your application to another

DIRECTORY STRUCTURE BEST PRACTICE

```
app/
--index.html
--app.js
----- shared/   // acts as reusable components (Services)
---------- services/
--------------- setHeader-service.js
----- components/   // each component is treated as a mini Angular app
 ---------- home/
--------------- home-controller.js
--------------- home-service.js
--------------- home-view.html
--------------- home-css.css
---------- products/
--------------- products-controller.js
--------------- products-service.js
--------------- products-view.html
assets/
----- img/      // Images and icons for your app
----- js/       // JavaScript files written for your app that are not for angular
----- node-modules/
```