CS 454 – AngularJS & Node.js

AngularJS

Albert Cervantes
Cydney Auman

What is AngularJS

- The official documentation of AngularJS says:

  "client-side technology, written entirely in JavaScript. It works with the long-established technologies of the web (HTML, CSS, and JavaScript) to make the development of web apps easier and faster than ever before."
- It is a framework that is primarily used to build single-page web applications.

How is AngularJS different?

- Other JavaScript frameworks force developers to extend from custom JavaScript objects and manipulate the DOM from the outside in.
- Example: jQuery-DOM.html
  - In jQuery, to add a button into the DOM, we have to *know* where we are putting it, and then insert it into place.
- Requires the developer to have knowledge of the entire DOM.

How is AngularJS different?

- Augments HTML
  - Grants a native Model-View-Controller (MVC) capability.
  - Makes building impressive and expressive client-side applications quick and enjoyable.
- Allows developers to encapsulate a portion of an entire page as one application, rather than forcing the entire page to be an AngularJS application.

Data Binding

- Data-binding is an automatic way of updating the view whenever the model changes, as well as updating the model whenever the view changes.
- This is awesome because it eliminates DOM manipulation from the list of things you have to worry about.
- Example: The Basics

Angular App Observations

- The *ng-app* attribute declares that everything inside of it belongs to this Angular app
  - This is how we can nest an Angular app inside of a web app.
  - The only components that will be affected by Angular are the DOM elements that we declare inside of the one with the *ng-app* attribute.
- We bound the *name* property to the input field using *ng-model*.
- The $scope object is simply a JavaScript object whose properties are all available to the view and with which the controller can interact.

Bi-Directional Binding

- When Angular thinks that a value might change, it will run its own event loop to check if the value is dirty.
  - A value is considered dirty if it has changed since the last time the event loop ran.
- If the view changes the value, the model observes the change through *dirty checking*.
- If the model changes the value, the view updates with the change.
- Example: Todo

Directives

- Directives are Angular's method of creating new HTML elements that have their own custom functionality.
- The simplest way to think about a directive is that it is simply a function that we run on a particular DOM element.
  - The *directive* function can provide extra functionality on the element.
- A built-in directive is one that ships out of the box with Angular.
- All built-in directives are prefixed with the **ng** namespace.
  - In order to avoid namespace collisions, do not prefix the name of your own directives with **ng**.
- Example: Directives/index.html, myDirective

Directive Observations

- ng-app=“myApp”
  - This line lets Angular know which **module** to use.
    - In this case, the module name is **myApp**.
- angular.module(‘myApp’, [])
  - If the ‘myApp’ module does not exist, this will create it and return a reference to the *new* module.
  - If the module already exists, this will return a reference to the *existing* module object.

Directive Definition

- With the *.directive()* method, provided by the Angular module API, we can register new directives by providing a name as a *string* and *function*.
- The name of the directive should always be *camelCased*, and the function we provide should return an object.

Directive Definition

- By default, Angular nests the HTML provided by our template string inside of our custom HTML tag, <my-directive>, in the generated source code.
  - To remove, we simply add one more option to our definition.
    - *replace: true*

Declaring Directives

- Declaring a directive is the act of placing a function within our HTML as an element, attribute, class, or comment.
- The following are valid declarations:

```
<my-directive></my-directive>
<div my-directive></div>
<div class="my-directive"></div>
<!-- directive: my-directive -->
```

- The **restrict** option tells Angular which declaration format(s) to look for when compiling our HTML.
- We can specify that we want our directive to be invoked if it is:

  an element (**E**), an attribute (**A**), a class (**C**), a comment (**M**)

Passing Data into Directives

- Notice in our template:

```
template: '<a href="https://github.com/cydneymikel/CS454">Click me to go to the CS 454 Homepage</a>'
```

- We are hard-coding the link and the text to be displayed.
- Let's change our template to:

```
template: '<a href="{{myUrl}}">{{myLinkText}}</a>’
```

- And our HTML to:

```
<div my-directive
        my-url="https://github.com/cydneymikel/CS454"
        my-link-text="Click me to go to the CS 454 Homepage"
 ></div>
```

Passing Data into Directives

- Setting attributes in the DOM is like passing arguments to a function.
- The directive has it’s own scope.
- To copy the DOM value of the attribute into a scope property, we use the **@** binding strategy.
- Example: index-directive-with-values.html

Binding Directive Inputs

- Sometimes, we want to bind directive inputs to models created elsewhere.
- This prevents us from having to hard-code attribute values in our HTML.
- Example: index-dynamic-attr.html

Built-In Directives

- Several built-in directives in AngularJS have similar HTML names
  - ng-href
  - ng-src
  - ng-disabled
  - ng-checked
  - ng-readonly
  - ng-selected
  - ng-class
  - ng-style

Controllers

- Controllers in AngularJS exist to augment the view of an AngularJS application.
- In our Hello world example application, we did not use a controller, but only an implicit controller.
- Example: Todo with Filters