AngularJS Introduction

Mendel Rosenblum
AngularJS

- JavaScript framework for writing web applications
  - Handles: DOM manipulation, input validation, server communication, URL management, etc.

- Uses Model-View-Controller pattern
  - HTML Templating approach with two-way binding

- Minimal server-side support dictated

- Focus on supporting for programming in the large and single page applications
  - Modules, reusable components, testing, etc.

- Widely used framework (Angular 1 - 2009) with a major rewrite coming out (Angular 2)
  - CS142 will use Angular 1
# Angular Concepts and Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template</td>
<td>HTML with additional markup used to describe what should be displayed</td>
</tr>
<tr>
<td>Directive</td>
<td>Allows developer to extend HTML with own elements and attributes (reusable pieces)</td>
</tr>
<tr>
<td>Scope</td>
<td>Context where the model data is stored so that templates and controllers can access</td>
</tr>
<tr>
<td>Compiler</td>
<td>Processes the template to generate HTML for the browser</td>
</tr>
<tr>
<td>Data Binding</td>
<td>Syncing of the data between the Scope and the HTML (two ways)</td>
</tr>
<tr>
<td>Dependency Injection</td>
<td>Fetching and setting up all the functionality needed by a component</td>
</tr>
<tr>
<td>Module</td>
<td>A container for all the parts of an application</td>
</tr>
<tr>
<td>Service</td>
<td>A way of packaging functionality to make it available to any view</td>
</tr>
</tbody>
</table>
Angular Example

<!doctype html>
<html ng-app>
  <head>
    <script src="/angular.min.js"></script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text" ng-model="yourName" placeholder="Enter a name here">
      <h1>Hello {{yourName}}!</h1>
    </div>
  </body>
</html>
Angular Bootstrap

<!doctype html>
<html ng-app>
  <head>
    <script src="./angular.min.js"></script>
  </head>
  <body>
    <div>
      <label> Name: </label>
      <input type="text" ng-model="yourName" placeholder="Enter a name here">
      <h1>Hello {{yourName}}!</h1>
    </div>
  </body>
</html>
Angular Bootstrap

```html
<!doctype html>
<html ng-app>
  <head>
    <script src="/angular.min.js"></script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text" ng-model="yourName" placeholder="Enter a name here">
      <h1>Hello {{yourName}}!</h1>
    </div>
  </body>
</html>
```

Once ready, scans the html looking for a ng-app attribute - Creates a scope.
Angular Bootstrap

<!doctype html>
<html ng-app>
    <head>
        <script src="/angular.min.js"></script>
    </head>
    <body>
        <div>
            <label>Name:</label>
            <input type="text" ng-model="yourName" placeholder="Enter a name here">
            <h1>Hello {{yourName}}!</h1>
        </div>
    </body>
</html>

Compiler - Scans DOM covered by the ng-app looking for templating markup - Fills in with information from **scope**.
Angular Compiler Output

Changes to template HTML in red. Classes:

- **ng-scope** - Angular attached a scope here.
- **ng-binding** - Angular bound something here.
- **ng-pristine/ng-dirty** - User interactions?
- **ng-untouched/ng-touched** - Blur event?
- **ng-valid/ng-invalid** - Valid value?

Note: `{{yourName}}` replaced with value of yourName
Two-way binding: Type 'D' character into input box

```html
<!doctype html>
<html ng-app class="ng-scope">
  <head>
    <script src="./angular.min.js"></script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text" ng-model="yourName" placeholder="Enter a name here" class="ng-valid ng-dirty ng-valid-parse ng-touched">
      <h1 class="ng-binding">Hello D!</h1>
    </div>
  </body>
</html>
```

The scope variable `yourName` is updated to be "D" and the template is rerendered with `yourName = "D"`. Note angular validation support.
Two-way binding: Type 'a', 'n' into input box

<!doctype html>
<html ng-app class="ng-scope">
  <head>
    <script src="./angular.min.js"></script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text" ng-model="yourName" placeholder="Enter a name here" class="ng-valid ng-dirty ng-valid-parse ng-touched">
      <h1 class="ng-binding">Hello Dan!</h1>
    </div>
  </body>
</html>
angular.module

<!doctype html>
<html ng-app="cs142App">
  <head>
    <script src="/angular.min.js"></script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text" ng-model="yourName" placeholder="Enter a name here">
      <h1>Hello {{yourName}}!</h1>
    </div>
  </body>
</html>

In a JavaScript file:

angular.module("cs142App", []);

or to fetch existing module:

angular.module("cs142App");

Module - Container of everything needed under ng-app
Controllers

<!doctype html>
<html ng-app="cs142App">
  <head>
    <script src="/angular.min.js"></script>
  </head>
  <body ng-controller="MyCntrl">
    <div>
      <label>Name:</label>
      <input type="text" ng-model="yourName" placeholder="Enter a name here">
      <h1>{{greeting}} {{yourName}}!</h1>
    </div>
  </body>
</html>

In a JavaScript file:

```
angular.module("cs142App", [])
  .controller('MyCntrl', function($scope) {
    $scope.yourName = "";
    $scope.greeting = "Hola";
  });
```

Will define a new scope and call controller MyCntrl.
Templates, Scopes & Controllers

- Best practice: Each **template** component gets a new **scope** and is paired with a **controller**.

- **Expressions** in templates:

  ```javascript
  {{foo + 2 * func()}}
  ```

  are evaluated in the context of the scope. Controller sets up scope:

  ```javascript
  $scope.foo = ... ;
  $scope.func = function() { ... };
  ```

- Best practice: Keep expressions simple put complexity in controller

- Controllers make model data available to view template
Scope inheritance

- A scope object gets its prototype set to its enclosing parent scope

```html
<div ng-controller="crtl1">
  <div ng-controller="crtl2">
    ...
  </div>
</div>
```

- $rootScope is parent of all

- ScopeB's prototype points at ScopeA
- Mostly does what you want (all properties of A appear in B)
- Useful since scopes are frequently created (e.g. ng-repeat, etc.)
- $rootScope is parent of all
"There should always be a dot in your model"

Common advice to beginning AngularJS programmers. Why?

Consider: `<input type="text" ng-model="yourName" placeholder="Enter a name here">`

Model reads will go up to fetch properties from inherited scopes. Writes will create the property in the current scope!

`<input type="text" ng-model="model.yourName" placeholder="Enter a name here">`

Read of object `model` will locate it in whatever inherited scope it is in. `yourName` will be create in that object in the right scope.
Scope digest and watches

- Two-way binding works by watching when expressions in view template change and updating the corresponding part of the DOM.
- Angular adds a **watch** for every variable or function in template expressions.
- During the **digest** processing all watched expressions are compared to their previously known value and if different the template is reprocessed and the DOM update.
  - Angular automatically runs digest after controller run, etc.

It is possible to:

- Add your own watches: `($scope.$watch(..))` (e.g. caching in controller)
- Trigger a digest cycle: `($scope.$digest())` (e.g. model updates in event)
Example of needing scope $watch

Name: {{firstName}} {{lastName}}

vs

Name: {{fullName}}

```javascript
$scope.fullName = $scope.firstName + " " + $scope.lastName;

$scope.$watch('firstName',
    function() {
        $scope.fullName = $scope.firstName + " " + $scope.lastName;
    });
```
A digression: camelCase vs dash-case

Word separator in multiword variable name

- Use dash: active-buffer-entry
- Capitalize first letter of each word: activeBufferEntry

Issue: HTML is case-insensitive so camelCase is a problem

AngularJS solution: You can use either, Angular will map them to the same thing.

Use dash in HTML and camelCase in JavaScript

Example: ng-model and ngModel
ngRepeat - Directive for looping in templates

- **ngRepeat** - Looping for DOM element creation (tr, li, p, etc.)

```
<ul>
  <li ng-repeat="person in peopleArray">
    <span>{{person.name}} nickname {{person.nickname}}</span>
  </li>
</ul>
```

- Powerful but opaque syntax. From documentation:

```
<div ng-repeat="model in collection | orderBy: 'id' as filtered_result track by model.id">
```

ngIf/ngShow - Conditional inclusion in DOM

- **ngIf** - Include in DOM if expression true (dialogs, modals, etc.)
  
  ```html
  <div class="center-box" ng-if="showTrialOverWarning">
    {{buyProductAdmonishmentText}}
  </div>
  ```

  Note: will create scope/controllers when going to true, exit going to false

- **ngShow** - Like ngIf except uses visibility to hide/show DOM elements
  - Occupies space when hidden
  - Scope & controllers created at startup
ngClick/ngModel - Binding user input to scope

- **ngClick** - Run code in scope when element is clicked
  ```html
  <button ng-click="count = count + 1" ng-init="count=0">
    Increment
  </button>
  <span> count: {{count}} </span>
  ```

- **ngModel** - Bind with input, select, textarea tags
  ```html
  <select name="singleSelect" ng-model="data.singleSelect">
    <option value="option-1">Option 1</option>
    <option value="option-2">Option 2</option>
  </select>
  ```
ngHref & ngSrc

Sometimes need to use ng version of attributes:

- a tag
  
  ```html
  <a ng-href="{{linkHref}}">link1</a>
  ```

- img tag
  
  ```html
  <img ng-src="{{imageSrc}}" alt="Description" />
  ```
ngInclude - Fetches/compile external HTML fragment

- Include partial HTML template (Take angular expression of URL)
  
  <div ng-include="'navBarHeader.html'"</div>

- CS142 uses for components
  
  <div ng-include="'components/example/exampleTemplate.html''
  ng-controller="ExampleController"></div>
Directives

- Angular preferred method for building reusable components
  - Package together HTML template and Controller and extend templating language.
  - Ng prefixed items in templates are directives
- Directive can:
  - Be inserted by HTML compiler as:
    - attribute (`<div my-dir="foo">...</div>`)  
    - element (`<my-dir arg1="foo">...</my-dir>`)  
  - Specify the template and controller to use
  - Accept arguments from the template
  - Run as a child scope or isolated scope
- Powerful but with a complex interface
  Example: `<example arg1="fooBar"></example>`
Directives are heavily used in Angular

```html
<body layout="row" ng-controller="AppCtrl">
  <md-sidenav layout="column" ...
>  
    <md-toolbar ...
      >
      ...
    </md-toolbar>
  <md-list>
    <md-item ng-repeat="item in menu">
      <md-item-content layout="row" layout-align="start center">
        <md-button aria-label="Add" ng-click="showAdd($event)"
          >
        </md-item-content>
      </md-item>
    <md-divider></md-divider>
    <md-subheader>Management</md-subheader>
  </md-item>
  <md-divider></md-divider>
  <md-subheader>Management</md-subheader>
</body>
```
Services

- Used to provide code modules across view components
  - Example: shared JavaScript libraries
- Angular has many built-in services
  - Server communication (model fetching)
    - $http, $resource, $xhrFactory
  - Wrapping DOM access (used for testing mocks)
    - $location, $window, $document, $timeout, $interval
  - Useful JavaScript functionality
    - $animate, $sce, $log
  - Angular internal accesses
    - $rootScope, $parse, $compile
Dependency injection

- Support for programming in large
  a. Entities list what they define and what they need
  b. At runtime Angular brings entities and their dependencies together

- Example:

```javascript
var cs142App = angular.module('cs142App', ['ngRoute']);
cs142App.config(['$routeProvider', function($routeProvider) {
  cs142App.controller('MainController', ['$scope', function($scope) {
```

```javascript
...}
```
Angular APIs

- ngRoute - Client-side URL routing and URL management
  - CS142 - Passing parameters to the views
- ngResource - REST API access
  - CS142 - Fetch models
- ngCookies - Cookie management and access
- ngAria - Support for people with disabilities (Accessible Rich Internet Applications)
- ngTouch - Support for mobile devices (ngSwipeLeft, ngSwipeRight, etc.)
- ngAnimate - Support for animations (CSS & JavaScript based)
- ngSanitize - Parse and manipulate HTML safely
Some thoughts on JavaScript Frameworks

- Web app can not start until framework downloaded and initialized
  - Particular relevant for wimpy devices and networks (e.g. Mobile)
- Can lazy load Angular modules (Makes dependency tracking important)
- Core Angular is not small

<table>
<thead>
<tr>
<th>File Path</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4.8/angular.js</td>
<td>1,070,726 bytes</td>
</tr>
<tr>
<td>1.4.8/angular.min.js</td>
<td>148,199 bytes</td>
</tr>
<tr>
<td>1.4.8/angular.min.js.gzip</td>
<td>53,281 bytes</td>
</tr>
</tbody>
</table>