Desarrollo de Aplicaciones Web Empresariales

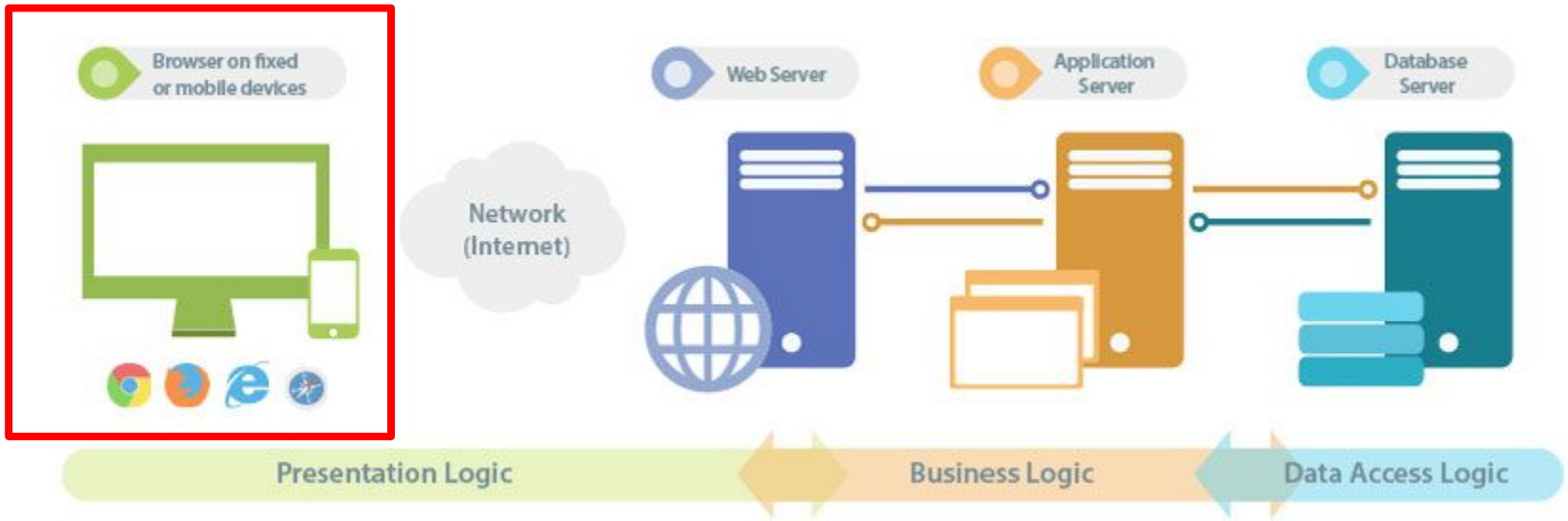
Exequiel Fuentes Lettura
exequiel.fuentes@ucn.cl

Información de contacto

- Exequiel Fuentes Lettura
  - Email: exequiel.fuentes@ucn.cl
  - Horario de Atención: Jueves y Viernes, bloque C

- Departamento de Ingeniería de Sistemas y Computación
  - Oficina: Y1 - 329
  - http://www.disc.ucn.cl

Arquitectura de aplicaciones Web: AngularJS



Material: https://docs.angularjs.org/guide

AngularJS

- Es un framework basado en Javascript para escribir aplicaciones Web.
- Utiliza el patrón Modelo-Vista-Controlador.
- Desde que se lanzó (2009) han habido algunos cambios, pero la versión 2 reescribió la mayoría de las cosas haciendo casi incompatibles las versiones.
  - En este curso se utilizará Angular 1.
- Otros frameworks:
  - Ember (Similar en muchos aspectos a AngularJS)
  - ReactJS

Conceptos y terminología

Template	HTML con marcado adicional usado para describir que se debería desplegar.
Directive	Permite extender HTML con elementos y atributos propios (reusable).
Scope	Contexto donde el modelo se almacena, así los templates y los controllers acceden.
Compiler	Proceso que hace que los templates generen HTML.
Data Binding	Sincronización entre los datos del scope y el HTML.
Module	Un contenedor para todas las partes de una aplicación.
Service	Una manera para empacar funcionalidades para que estén disponibles para cualquier vista.

Ejemplo

```
<!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>
```

Cómo funciona

```
<!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>
```

Script loads and runs on when browser signals context is loaded and ready

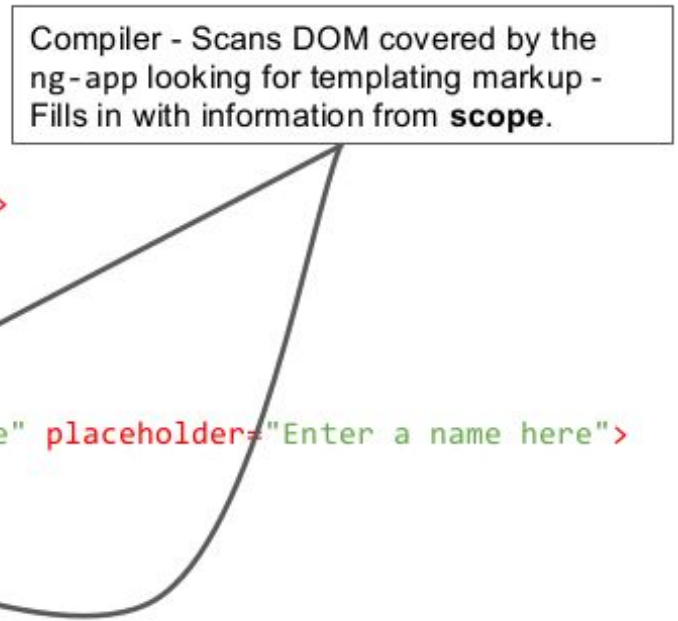
Cómo funciona

```
<!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**.

Cómo funciona

```
<!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>
```



Cómo funciona

```
<!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-pristine ng-untouched ng-valid">
         <h1 class="ng-binding">Hello !</h1>
      </div>
   </body>
</html>
```

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

Cómo funciona

```
<!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

Cómo funciona

```
<!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>
```

Template updated with each change (i.e. key stroke)!

Continuando...

In a JavaScript file:

```
angular.module("cs142App", []);
    or to fetch existing module:
angular.module("cs142App");
```

```
<!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>
```

Module - Container of everything needed under ng-app

Continuando...

In a JavaScript file:

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

```
<!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>
```

Will define a new scope and call controller MyCntrl.

Templates, Scopes y Controllers

- Cada **template** tiene un nuevo **scope** y debe estar asociado con un **controller**.
- La expresiones en los templates son de la forma:
  - {{foo + 2 * func()}}
  - Que son evaluados en el contexto del scope. El controller cambia el scope:
  - $scope.foo = ... ;
    $scope.func = function() { ... };
- Sugerencia: mantener las expresiones simples, poner la complejidad en el controller.
- Los controllers hacen que los datos estén disponibles en el template.

Herencia en el scope

- El scope de un objeto está encapsulado al scope de su padre:

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

Creates new scope (ScopeA)

Creates new scope (ScopeB)

- Todas la propiedades de A están en B.
- Útil dado que scopes son frecuentemente creados, ejemplo: ng-repeat
- $rootScope es el padre de todos.

Sugerencia para scope y model

- Considera:
  - <input type="text" ng-model="yourName" placeholder="Enter a name here">
- Esto obtendrá las propiedades del actual scope, escribirá la propiedad en el actual scope!
- La solución es:
  - <input type="text" ng-model="**model.**yourName" placeholder="Enter a name here">
- Entonces, localizará las propiedades del modelo ***model***. Se escribirá la propiedad en el objeto en el correcto scope.

Scope digest y watches

- AngularJS agrega a **watch** por cada variable o función en las expresiones del **template**.
- Durante el proceso de **digest** todas las expresiones monitoreadas son comparadas con su valor previo en busca de diferencias, si las hay se actualiza el DOM.
- Es posible:
  - Agregar tus propios watches: ($scope.$watch(..)
  - Lanzar un digest: ($scope.$digest())

Ejemplo de $watch

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

     vs

Name: {{fullName}}
```

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

$scope.$watch('firstName',
 function() {
  $scope.fullName =
     $scope.firstName +
      " " + $scope.lastName;
  });
```

camelCase vs dash-case

- Formas de separar variables con nombre compuesto:
  - Usando guión: active-buffer-entry
  - Primera mayúscula por cada palabra: activeBufferEntry
- Problema: HTML es case-insensitive.
- Solución en AngularJS: ambas son válidas.
- Convención:
  - Usar guión en HTML: ng-model
  - Usar camelCase en Javascript: ngModel

ngRepeat: Directiva para ciclos

- ngRepeat: ciclo para crear elementos en el DOM (tr, li, p, etc)

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

ngIf/ngShow: Condiciones

- ngIf: Se incluye en el DOM si la expresión es verdadera.
  - Se crea el scope y el controller cuando es verdadero.

```
<div class="center-box" ng-if="showTrialOverWarning">
   {{buyProductAdmonishmentText}}
</div>
```

- ngShow: Similar a ngIf excepto que es usado para esconder o mostrar elementos en el DOM.
  - Ocupa espacio en el DOM (pero no en la pantalla) cuando está escondido
  - El scope y el controller se crea cuando en el inicio.

ngClick/ngModel: Une la entrada al scope

- ngClick: Ejecuta el código en el scope cuando el elemento es clickeado.

```
<button ng-click="count = count + 1" ng-init="count=0">
    Increment
</button>
<span> count: {{count}} </span>
```

- ngModel: Une elementos como input, select, textarea.

```
<select name="singleSelect" ng-model="data.singleSelect">
    <option value="option-1">Option 1</option>
    <option value="option-2">Option 2</option>
</select>
```

Directivas: ngHref, ngSrc y ngInclude

- Cambiar atributos en los elementos **a** y **img**:
  - <a ng-href="{{linkHref}}">link1
  -
- Incluir HTML parcial desde otros templates:
  -

Directivas

- Es el método preferido en AngularJS para construir componentes reutilizables.
  - Permite extender el lenguaje usando templates HTML y Controllers.
  - ng es el prefijo que indica que es una directiva.
- Las directivas pueden:
  - Ser insertadas por el compilador HTML como atributo o elemento.
  - Especificar el template y controller a ser usado.
  - Aceptar argumentos desde los templates.
  - Ejecutar un scope como un hijo o completamente aislado.

Servicios

- Usado para proporcionar módulos a través de componentes
  - Ejemplo: librerías Javascript compartidas.
- AngularJS tiene muchos servicios preestablecidos:
  - Comunicación: $http, $resource, $xhrFactory
  - Acceso al DOM: $location, $window, $document, $timeout, $interval
  - Algunas funcionalidades Javascript: $animate, $sce, $log
  - Acceso interno a Angular: $rootScope, $parse, $compile

Algunas cosas a considerar

- Un framework no puede comenzar hasta que se descarga e inicializa.
  - Particularmente relevante en conexiones móviles.
- Los módulos en AngularJS se obtienen cuando son requeridos.
- AngularJS no es pequeño:

1.5.8/angular.js	1.187.559 bytes
1.5.8/angular.min.js	160.048 bytes
1.5.8/angular.min.js.gzip	56.165 bytes

- Se recomienda el uso de CDN:
  - https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js

AngularJS v2

- Se hizo un rediseño casi completo, haciendo que la versión 2 sea considerada otro framework.
- Se puede encontrar las diferencias en este link:
  - https://angular.io/docs/ts/latest/cookbook/a1-a2-quick-reference.html

¿Preguntas?