Angular

Capital One

developintelligence.com

Third Day Topics

- Controllers vs .Services
- Services: Factory & Service
- Services: Constant & Value
- Services: Provider
- Ajax with $http
- Ajax with $resource
- Ajax with Restangular
- Promises & $q
- Thinning out Controllers

Services

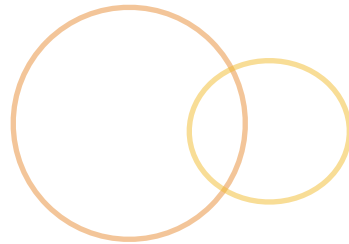
Controllers vs. Services

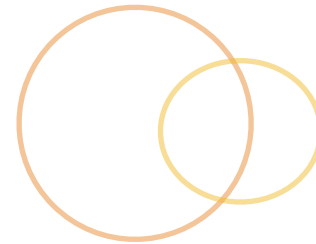
- Controllers
  - Don’t reference the DOM: Directives do that
  - Have view logic interaction via $scope
    - Handles actions from the view’s behavior
    - If a user clicks on an element the controller will directly handle the interaction passed from the view
    - Handles providing data to the view so it can be rendered
    - glue logic
  - New instance per view
    - Controllers get instantiated when they are needed
    - Garbage collected when not in use

Controllers vs. Services [cont.]

- Services
  - Mostly won't reference the DOM: Directives do that
  - Logic not interacting with the view's behavior
    - Handles the applications operation
    - Processes the business logic
    - Gives an abstraction API to low-level interactions
  - Services are singletons
    - Instantiate once by injection
    - Exist for the lifetime of the application
    - Great for caching information between view changes

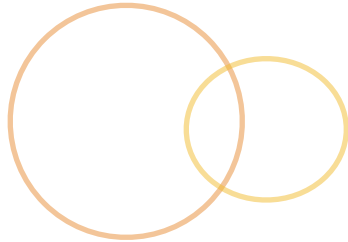
Services

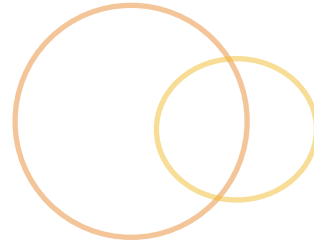




- They are great for bundling similar business logic
- They are helpful in handling information across controllers
- They are responsible for server calls
  - If they are doing ajax make them return promises
- Services don’t have their own scope

Services





- We interact with Services via dependency injection
  - Instead of creating an instance of Service we simply ask for the created instance
- Angular handles all the hidden dependencies
  - Angular automatically creates the entire chain of services before injecting the service in our code

Custom Service

- Let's create a lower order math service
  - Create a couple input fields for numbers
  - Create some buttons to perform lower order math
  - Bind an output variable to the screen for us to see the answer

```
<section ng-controller="MathController">
  <h2>Math</h2>
  <input type="text" ng-model="calculate.valueA" />
  <input type="text" ng-model="calculate.valueB" />
  <button ng-click="addValues()">Add</button>
  <button ng-click="subtractValues()">Subtract</button>
  <button ng-click="multiplyValues()">Multiply</button>
  <button ng-click="divideValues()">Divide</button>
  <h3>Output: {{calculate.calculation}}</h3>
</section>
```

Factory Creation

- We will use the **factory** method to create the service
  - Factory services are created using the .**factory** method off of a module
  - We create an object literal to handle the public api of the service
  - The returned object literal is the instance of the service
  - Any methods or variables that utilize “var” (i.e. not attached to the object literal) will be held privately within the service
- Remember a service is only instantiated once during an application lifecycle
  - Singleton

Factory Creation

- Factory Services create an object literal

```
//MathLowerService
app.factory('MathLowerService', function() {
  var factory = {};
  factory.add = function(valueA, valueB) {
    return Number(valueA) + Number(valueB);
  };
  factory.subtract = function(valueA, valueB) {
    return valueA - valueB;
  };
  factory.multiply = function(valueA, valueB) {
    return valueA * valueB;
  };
  factory.divide = function(valueA, valueB) {
    return valueA / valueB;
  };
  return factory;
});
```

Module Pattern

```
var aModule = (function() {
  var time = 8;

  return {
    doWork: function() {
      console.log("working " + time + " hours");
    },
    doMoreWork: function() {
       time += 2;
      console.log("working more " + time + " hours");
    }
  };
}());

console.log("time: " + aModule.time);
aModule.doWork();
aModule.doMoreWork();
```

Revealing Module

```
var aModule = (function () {
  var time = 8;
  function doWork() {
    console.log("working " + time + " hours");
  };
  function doMoreWork() {
      time += 2;
      console.log("working more " + time + " hours");
  };

  return {
    doWork: doWork,
    doMoreWork: doMoreWork
  };
}());

console.log("time: " + aModule.time);
aModule.doWork();
aModule.doMoreWork();
```

Service Interaction

- How do we use the service?
  - First we need to inject it into the controller utilizing it

```
//Math Controller
app.controller('MathController', ['$scope',
  'MathLowerService', function($scope, MathLowerService) {

  $scope.calculate = {
    calculation: '',
    valueA: '',
    valueB: ''
  };

  //… Actions called from the view

}]);
```

Custom Service [cont.]

- How do we use the service?
  - Second we use it like any other service

```
//Inside Math Controller

  $scope.addValues = function() {
    $scope.calculate.calculation =
      MathLowerService.add($scope.calculate.valueA,
      $scope.calculate.valueB);
  };

  $scope.subtractValues = function() {
    $scope.calculate.calculation =
      MathLowerService.subtract($scope.calculate.valueA,
      $scope.calculate.valueB);
  };

//Inside Math Controller
```

Custom Service [cont.]

- How do we use the service?
  - Second we use it like any other service [cont.]

```
//Inside Math controller

  $scope.multiplyValues = function() {
    $scope.calculate.calculation =
      MathLowerService.multiply($scope.calculate.valueA,
      $scope.calculate.valueB);
  };

  $scope.divideValues = function() {
    $scope.calculate.calculation =
      MathLowerService.divide($scope.calculate.valueA,
      $scope.calculate.valueB);
  };

//Inside Math controller
```

Custom Service [cont.]

- HTML Output

Service Creation

- We could also use the **.service** method off a module to create the service
  - "Service" Services are instantiated with the **new** operator
  - We don't return an object literal as the service API like the "Factory" Services do
  - We treat a service the same as creating a JavaScript Object utilizing the Constructor for creation
  - Any methods we add to the object's "this" scope will be exposed as our Service API
  - Any methods or variables that utilize "var" will be held privately within the service

Service Creation

- We will use the **service** method to create the service
  - Takes a Constructor function

```
app.service('MathLowerService', function() {
  this.add = function(valueA, valueB) {
    return Number(valueA) + Number(valueB);
  };
  this.subtract = function(valueA, valueB) {
    return valueA - valueB;
  };
  this.multiply = function(valueA, valueB) {
    return valueA * valueB;
  };
  this.divide = function(valueA, valueB) {
    return valueA / valueB;
  };
});
```

Service Creation

- It's like we are saying…

```
function Math() {
  this.add = function(valueA, valueB) {
    return Number(valueA) + Number(valueB);
  };
  this.subtract = function(valueA, valueB) {
    return valueA - valueB;
  };
  this.multiply = function(valueA, valueB) {
    return valueA * valueB;
  };
  this.divide = function(valueA, valueB) {
    return valueA / valueB;
  };
});

app.service('MathLowerService', Math);
```

.factory() vs. .service()

- The .factory() and the .service() both create services for us
- .factory()
  - Takes a service name and a factory function returning an object that functions as our API
- Using the .service() is instantiated behind the scenes using the new operator
  - Takes a service name and a constructor function used to create the instance
- When it is all said and done they will give us the exact same API to interact with
- No performance difference between these

Service Dependency

- We have a lower order math service
  - Now let's create a higher order math service

```
<section ng-controller="MathController">
  <h2>Math</h2>
  <input type="text" ng-model="calculate.valueA" />
  <button ng-click="squareValue()">Square</button>
  <button ng-click="cubeValue()">Cube</button>
  <h3>Output: {{calculate.calculation}}</h3>
</section>
```

Service Dependency [cont.]

- We create the math higher order service based on the low level functionality of math lower order service
  - mathLowerService is dependency injected

```
app.service('MathHigherService', ['MathLowerService',
  function(MathLowerService) {
    this.square = function(value) {
      return MathLowerService.multiply(value, value);
    };
    this.cube = function(value) {
      return MathLowerService.multiply(value,
        MathLowerService.multiply(value, value));
    };
}]);
```

Service Dependency [cont.]

- Using the higher order math service
  - We need to inject MathHigherService into our controller
  - We won't need to inject the MathLowerService into the controller unless we are going to specifically use it
    - AngularJS automatically takes care of resolving any dependencies needed behind the scenes

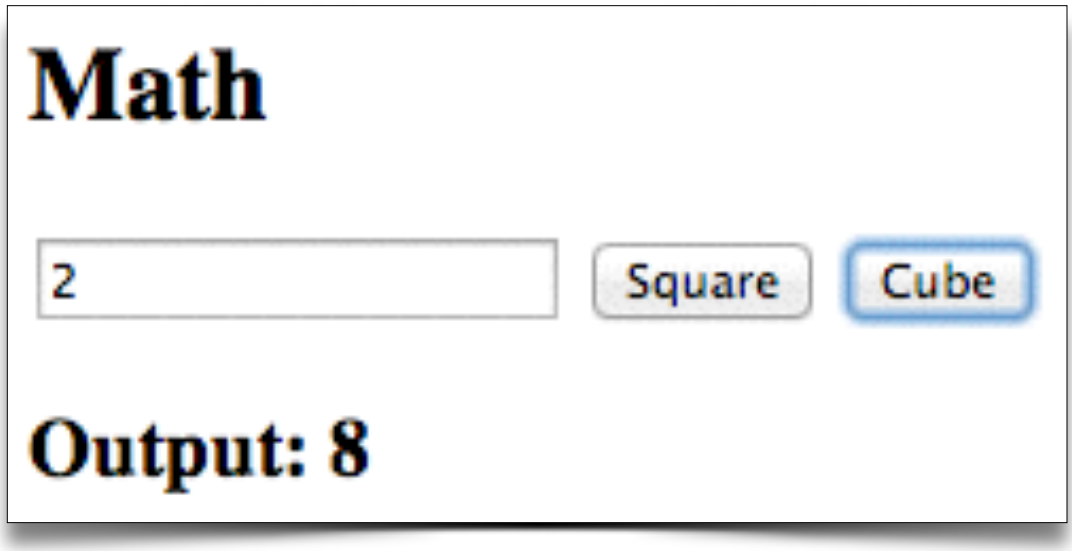
Service Dependency [cont.]

- Using the higher order math service
  - We need to inject mathHigherService into our controller

```
app.controller('MathController',
  ['$scope', 'MathHigherService',
  function($scope, MathLowerService) {
  $scope.calculate = {
    calculation: '',
    valueB: ''
  };
  $scope.squareValue = function() {
    $scope.calculate.calculation =
      MathHigherService.square($scope.calculate.valueA);
  };
  $scope.cubeValue = function() {
    $scope.calculate.calculation =
      MathHigherService.cube($scope.calculate.valueA);
  };
}]);
```

Service Dependency [cont.]

- HTML Output



Syntactic sugar

- Interesting to note: Angular is doing some work behind the scenes for this **.factory()** service

```
//MathLowerService
app.factory('MathLowerService', function() {
  var factory = {};
  factory.add = function(valueA, valueB) {
    return Number(valueA) + Number(valueB);
  };
  return factory;
});
```

Syntactic sugar [cont.]

- In reality Angular is providing some helper functionality for us

```
//MathLowerService
app.config(function($provide) {
  $provide.factory('MathLowerService', function() {
    var factory = {};
    factory.add = function(valueA, valueB) {
      return Number(valueA) + Number(valueB);
    };
    return factory;
  });
});
```

$provide

- $provide Service used for creating services in Angular
  - The $provide service is utilized to create all services in angular
  - $provide registers our services with the $injector
    - $injector creates one instance of the service per application
  - $provide gives us the different ways of creating our services
    - $provide.factory(factoryFunction)
    - $provide.service(constructorFunction)
    - $provide.constant(object)
    - $provide.value(object)
    - $provide.provider(providerFunction)

Constant Service

- Constant services are good to use for values that aren't ever going to change
  - We can set constants as primitives or objects
  - We can inject the constant into our services, controllers… and other application components
  - Can be injected into a .config function
    - Use it for configuration data

```
app.constant('months', {
 'JAN': 'January',
 'FEB': 'February',
 'MAR': 'March'
 });
```

```
app.constant('PI', 3.14);
```

Value Service

- Value services are similar to constants
  - We can set values as primitives or objects
  - We can inject the constant into our services, controllers… and other application components
- Value services can be overwritten
  - Values that are primitives aren't meant to be overwritten
- Values can't be used in the .config of a module
- Values should be used for service objects

Value Service [cont.]

- Value service

```
app.value('names', {
  'KZ': ‘Kam Zorgdrager’,
  ‘DI’: ‘Develop Intelligence
 });
```

```
app.value(‘PI', 3.14);
```

```
app.factory(‘PhilanthropistApiService', function(names) {
  names.KZ = ‘Kamren Zorgdrager’;
});
```

Value Service Object

- Using Constructor Objects with value services

```
var Product = function(type, quantityAvailable, cost) {
  //Adding the variable type to each object instance
  this.type = type;
  //Adding the available quantity to each object instance
  this.quantityAvailable = quantityAvailable;
  //Adding the cost to each object instance
  this.cost = cost;
}
Product.prototype = {
  //Resetting the prototype constructor property to point Product not Object
  constructor: Product,
  //Get the type of the product.
  getType: function() {
    return this.type;
  },
  //Get the cost of the product.
  getCost: function() {
    return this.cost;
  },
  //Get the available quantity of the product.
  getAvailableQuantity: function() {
    return this.quantityAvailable;
  }
};
```

app.value('product', Product);

Provider Service

- $provide Service used for creating services in Angular
- Useful if you need to provide configuration data for your service
  - Helps to make your services more reusable
- Must expose $get method for $injector to utilize

Provider Service [cont.]

- $provide Service used for creating services in Angular

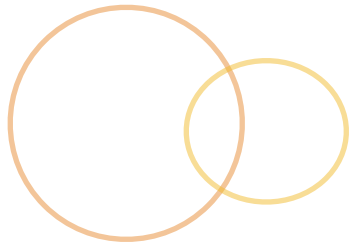
```
//MathLowerService
app.provider('MathLowerService', function() {
  var factor = 0;
  return {
    setFactor: function(pFactor) {
      factor = pFactor;
    },
    $get: function() {
      var multiply = function(value) {
        return Number(value) * factor;
      };
      return {multiply: multiply};
    }
});
```

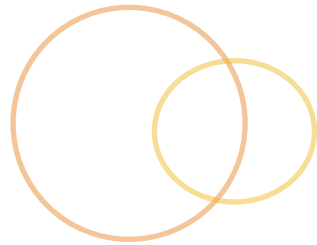
Provider Service [cont.]

- Now we can configure our service within .config for the module
  - Our multiply function off the MathLowerService will now multiply all our values times 2

```
//MathLowerServiceProvider
app.config(MathLowerServiceProvider, function() {
  MathLowerServiceProvider.setFactor(2);
});
```

Lab 7





- Our transaction logic should be placed in a service
  - Have the transaction service do all the accounting
    - Set default prices in the transaction service
  - Have our controllers invoke methods on the transaction service for:
    - Incrementing product sales
    - Getting product sales
    - Clearing out the transaction

Ajax

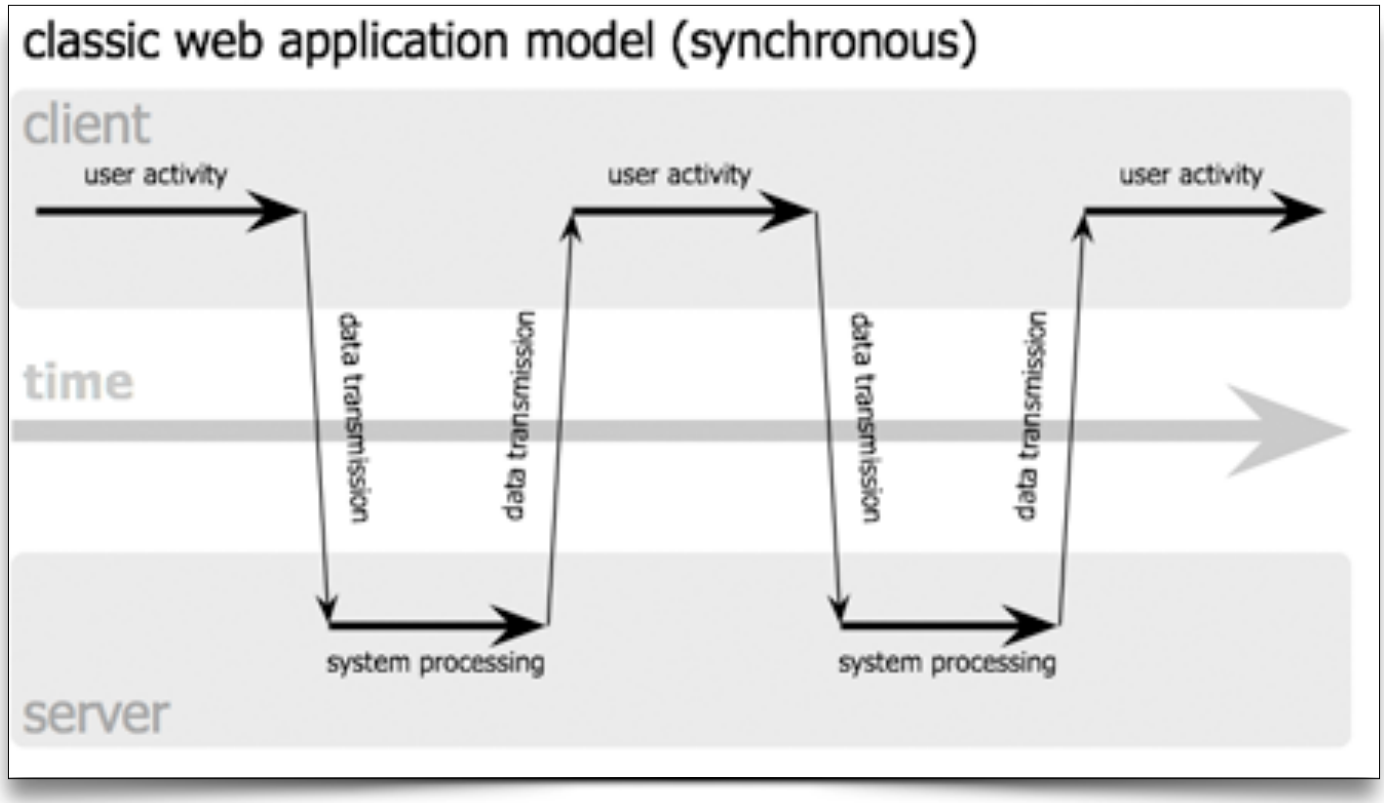
What is Ajax?

- Combination of technologies and concepts:
  - Asynchronous: A way of interacting that allows a program to flow unhindered with the sending and receiving of data
  - JavaScript: The scripting language holding Ajax together
  - eXtensible Markup Language (XML): How transmitted data is formatted
    - Today we usually use JSON or HTML fragments
- Purpose: To create more of a web application feel vs. web page feel

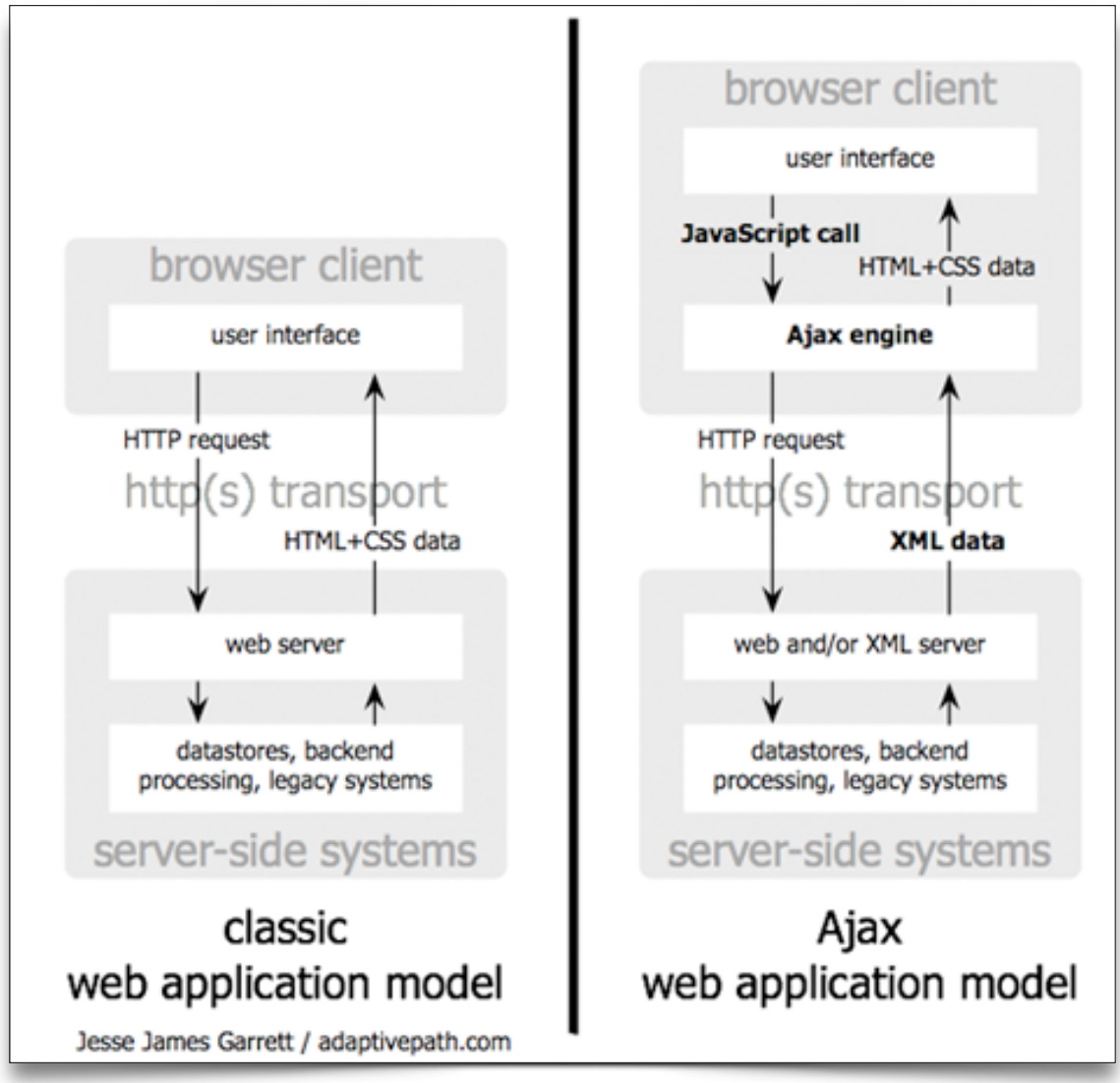
The Traditional Approach

- The whole web page is reloaded
- All content (structure & data) sent from server
- State is saved on the server and changed on the server
- The user experience is disrupted as pages change
- JavaScript is of little importance

The Traditional Approach [cont.]



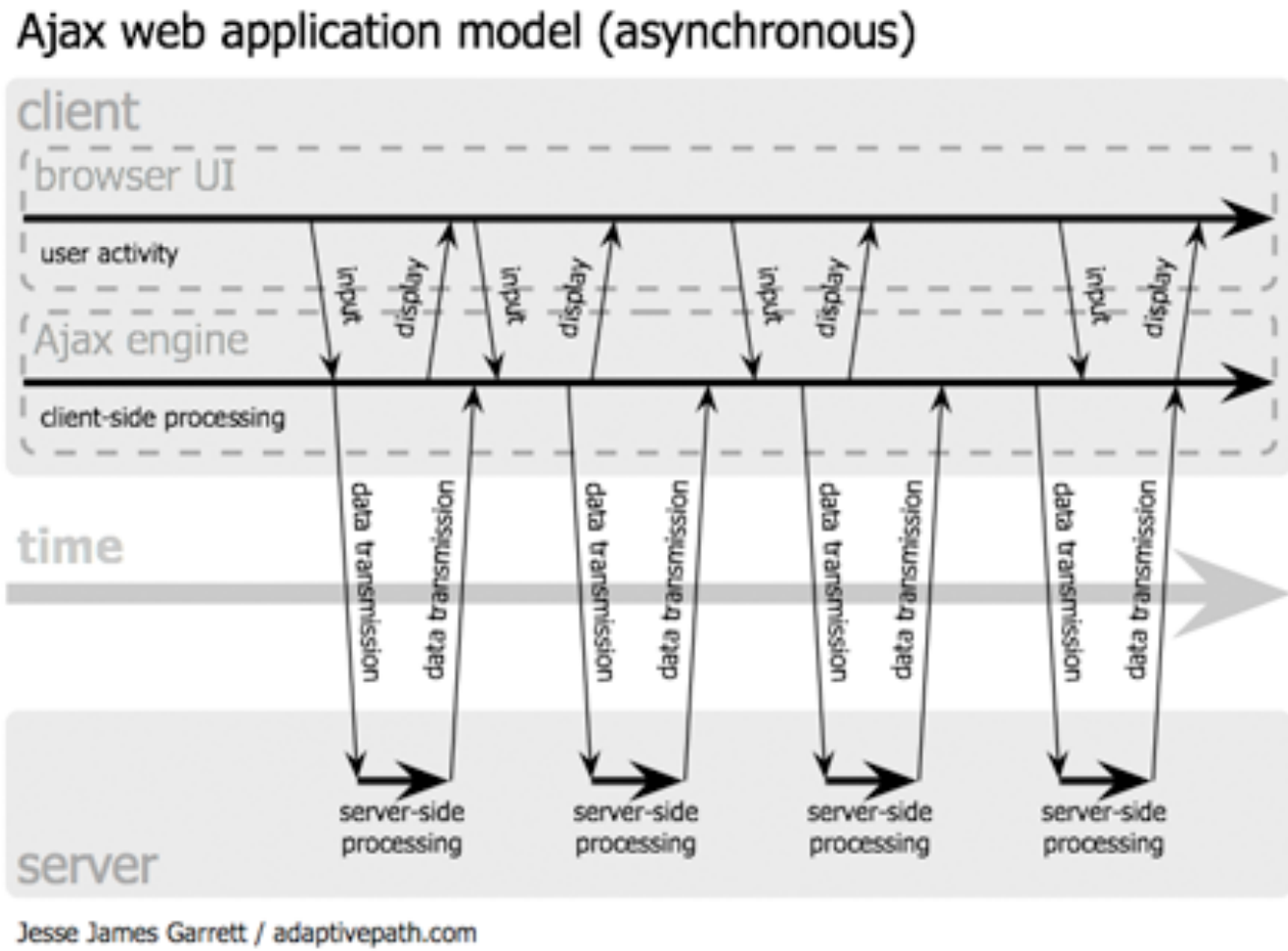
Comparison



The Ajax Approach

- Specific parts of a web page refresh
- Relevant data is sent from the server
- State is used differently
  - Users can stay on the same web page through the whole application
  - URL doesn't need to change: It can if it needs to :)
- The user experience is dynamic and fluid
- JavaScript becomes a core component
  - More design and architecting needed for JavaScript

The Ajax Approach [cont.]



Ajax Origins?

- Internet Explorer & IFrames in 1996
- Actual XHR object appeared within IE5 in 1999
  - XMLHttpRequestObject (XHR) object handles data transportation between client and server
- XHR appeared within Mozilla 1.0 in 2002

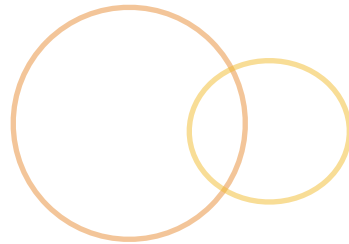
Ajax Origins? [cont.]

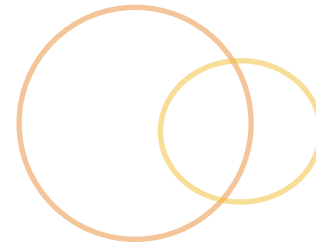
- Ajax term coined by Jesse James Garret in 2005
- Google Maps & Google Suggest started the revolution
- W3C draft specification in 2007
  - World Wide Web Consortium

How to Write Ajax

- Create an XHR object
- Send a request from client to server
- Handle server response with a callback
  - Usually :)

Status





- Property on the request object
  - Gives the HTTP status from the server
- 200: Everything is good, process the results
- 404: Server couldn't find the requested end point
- 403: Requesting a forbidden program from the server

Angular Ajax $http

$http Service

- Service used to create XHR objects for backend interaction
  - Configuration object
  - **success** and **error** are not callbacks
  - This is a promise based service
  - https://docs.angularjs.org/api/ng/service/$http

```
$http({
  method: “GET”,
  url: url
}).success(function(data, status, headers, config) {
  //On a success do this
}).error(function(data, status, headers, config) {
  //On an error do this
});
```

$http Service [cont.]

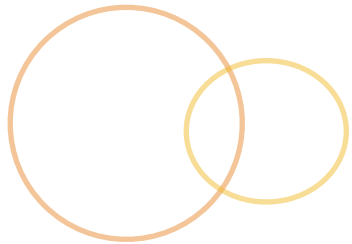
- Thinking about promises

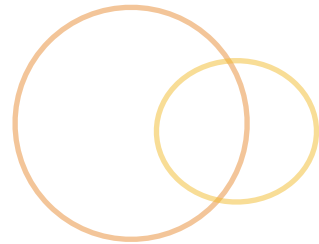
```
var promiseKept = $http({method: "GET", url: url});

promiseKept.then(function(response) {
  //Filtering for a success response
}, function(response) {
  //Filtering for an error response
});

promiseKept.success(function(data, status, headers, config) {
  //On a success do this
});

promiseKept.error(function(data, status, headers, config) {
  //On an error do this
});
```

.then

- Takes 2 parameters with $http
- success callback
  - Called if promised is resolved
  - Given whole XHR response as parameter
- failure callback
  - Called if promise is rejected
  - Given whole XHR response as parameter

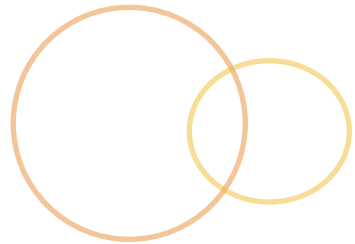
.success

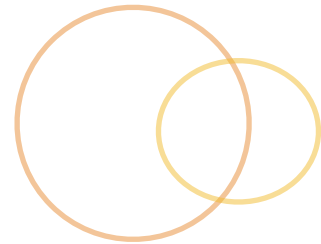




- Any 200 level status code will be considered successful
- success callback takes 4 parameters (broken out response object)
  - data: The response body
  - status: The HTTP status code
  - headers: Getter for the HTTP header
    - headers() will give you all header possibilities
    - headers('last-modified') will give you the last modified date as a string
  - config: The configuration object used to create the XHR request

.error





- Usually a 400 or 500 level status code will be considered an error
- error callback takes 4 parameters (broken out response object)
  - data: String with the error
    - "Cannot GET /your/url”
  - status: The HTTP status code
  - headers: Getter for the HTTP header
    - headers() will give you all header possibilities
    - headers(‘date') will give you the date as a string
  - config: The configuration object used to create the XHR request

$http config

- Pseudo-code for calling $http(config)

```
$http({
  method: string,
  url: string,
  params: object,
  data: string or object,
  headers: object,
  transformRequest: function transform(data, headersGetter) or
                    an array of functions,
  transformRequest: function transform(data, headersGetter) or
                    an array of functions,
  cache: boolean or Cache object,
  timeout: number,
  withCredentials: boolean
});
```

$http.get()

- Shortcut for creating a GET request
  - Retrieve an item or items (REST)
  - /ourApp/items/ … Retrieve all items
  - /ourApp/items/a … Retrieve item a
- $http.get('URL', configurationObject)
  - configurationObject
    - It could be headers, cache, timeout …

$http.post()

- Shortcut for creating a POST request
  - Replace an item or items (REST)
  - /ourApp/items/ … Replace all items
  - /ourApp/items/a … Replace item a
- $http.post('URL', data, configurationObject)
  - data: The object you have associated with your ng-model's in your form
    - We have been working with a **person** object so we could put **$scope.person** in for the data

$http.put()

- Shortcut for creating a PUT request
  - Creating a new item or items (REST)
  - /ourApp/items/ … Create a list of items
  - /ourApp/items/a … Create item a
- $http.put('URL', data, configurationObject)
  - data: The object you have associated with your ng-model's in your form
    - We have been working with a **person** object so we could put **$scope.person** in for the data

$http.delete()

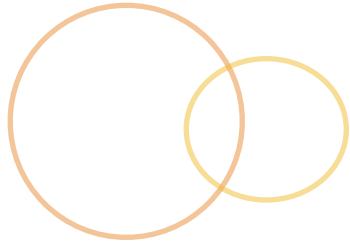
- Shortcut for creating a DELETE request
  - Delete an item or items (REST)
  - /ourApp/items/ … Delete all items
  - /ourApp/items/a … Delete item a
- $http.delete('URL', configurationObject)

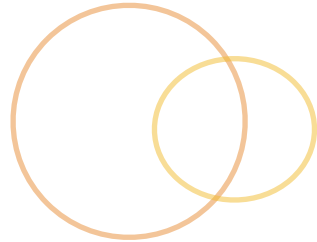
$httpProvider

- Allows us to configure the $http service
  - We could add header information inside the .config function
  - $httpProvider.defaults.headers.common
    - Allow us to send headers for each request

```
$httpProvider.defaults.headers.common['X-Requested-By'] =
        'DemoApplication';
```

Lab 8





- Create a supplies page
  - Have it list the available products we have
  - Have it list the maximum quantities
  - Get the current quantities from the backend
    - Create a json file that will serve you this information
  - Have a reset product quantity button that will fetch the backend sales
  - You could use a range element to display the information

Angular Ajax $resource

$resource Service

- Angular additional service designed specifically for RESTful applications
  - Makes less overhead than is necessary for the $http service
  - Need to dependency inject ngResource
  - https://docs.angularjs.org/api/ngResource/service/$resource

```
var app = angular.module('demo', ['ngResource']);
```

$resource Service [cont.]

- $resource creates a resource Class object
- The resource object's API is better suited for REST
  - get: GET
  - save: POST
  - query: GET (i.e. GET all resources)
  - remove: DELETE
  - delete: DELETE

```
//Buyer is our created resource object
var Buyer = $resource('/buyer/:buyerId');
```

$resource .get() API

- Parameters
  - Parameter object: Used to pass parameters in the URL
  - Success function
  - Error function
- Expects 1 resource returned
- URL: GET /buyers/13579?accessLevel=42

```
var Buyer = $resource('/buyers/:buyerId');
var buyer = Buyer.get({
  buyerId: 13579,
  accessLevel: 42
}, function (response) {
  //Interact with properties on the buyer object instance
  buyer.name = 'Kamren';
}, function(error) {
  //Do things because of erring
});
```

$resource .query() API

- Uses GET to retrieve a list of resources
- Expects a list of resources returned
- URL: GET /buyers

```
var buyers = Buyer.query(function (response) {
  //Interact with properties on the buyer object instance
  var buyer = buyers[0];
  buyer.name = 'Kamren';
});
```

$resource .save() API

- Uses POST to save a resource
- Parameters
  - Parameters
  - Body payload
  - Successful function
  - Error function
- POST /buyers

```
Buyer.save({}, {
  name: 'Kamren'
}, function (response) {
  //Handle a successful response
}, function (response) {
  //Handle a non-successful response
});
```

$resource .delete() API

- Uses DELETE to remove a resource
- Parameters
  - Parameters
  - Body payload
  - Successful function
  - Error function
- URL: DELETE /buyers/13579

```
Buyer.delete({
  buyerId: 13579
},
{}, function (response) {
  //Handle a successful response
}, function (response) {
  //Handle a non-successful response
});
```

$resource .remove() API

- Uses DELETE to remove a resource
- Why 2 DELETE methods?
  - **delete** is a keyword in JavaScript **.delete()** can cause issues with IE
  - .delete() and .remove() are synonymous
- Parameters: Same as .delete()

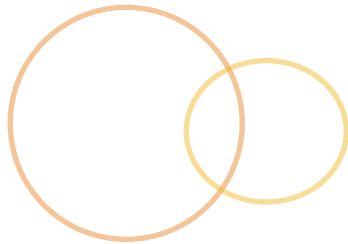
```
Buyer.remove({
  buyerId: 13579
},
{}, function (response) {
  //Handle a successful response
}, function (response) {
  //Handle a non-successful response
});
```

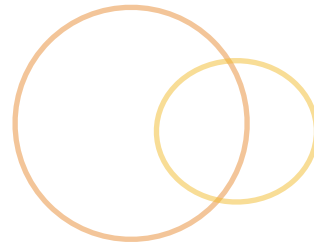
Resource Instances

- Convenience methods on returned server data
- Called on a single instance
  - $save()
  - $remove()
  - $delete()

```
var Buyer = $resource('/buyers/:buyerId');
var buyer = Buyer.get({
  buyerId: 13579
}, function (response) {
  //Interact with properties on the buyer object instance
  buyer.name = 'Kamren';
  buyer.$save();
});
```

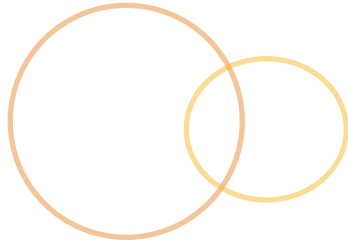
Lab 9

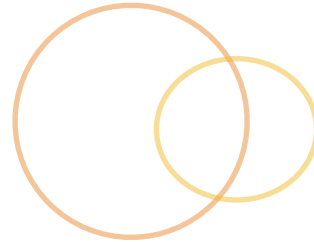




- On the **supplies page**
  - Remove the reset button
  - On initialization of the supplies check to see if the supplies service has been instantiated
  - If it hasn't, fetch the supply quantities and set it
- On initialization of the **sell page** check to see if we have initialized our transaction service
  - If the price is already set then we don't need to go to the backend
  - Create a json file for cost initialization
  - Use the $resource service
  - Get the price to show in our sub views: only show a price if there is one

Lab 9 [cont.]





- On the **sell page** create the ability to make a purchase
  - Only allow the transaction to go through if we have:
    - Something to transact
    - Enough product for the transaction
  - At this point: you will need to visit the supplies page before you go to the sell page
    - We are only initializing the supplies service on the supplies page
    - We will fix this next lab

Transforms

- Sometimes the data coming to us from an API is not configured the way we need it to be
- $http & $resource gives us transforming capabilities
  - transformResponse: transform incoming data
- Write a factory service that will return a transforming function
  - It will take the data from the response
  - Return the changed data

Transforms $http

- The $http service

```
$http({
  method: 'GET',
  url: ‘/some/api',
  transformResponse: transformResponseService
});
```

Transforms $resource

- $resource allows us to rename its base server interaction methods
  - Below is the default configuration for the different server interactions

```
 {
    'get':    {method:'GET'},
    'save':   {method:'POST'},
    'query':  {method:'GET', isArray:true},
    'remove': {method:'DELETE'},
    'delete': {method:'DELETE'}
};
```

Transforms $resource [cont.]

- To override or add our own method we need to specify it in our own actions object literal
  - It is also where we can specify the transformResponseService

```
var Buyer = $resource('/buyers/:buyerId', {}, {
  //Actions object defining what we want to happen
  {
    'retrieve': {
      method:'GET',
      transformResponse: tansformResponseService
    }
  }
});
```

Router Helps

- Sometimes we have needed to get our controllers to run a service interaction when the controller is initialized
  - We have used IIFE or simply invoked a method to handle this
  - That is a bit awkward

Router Helps [cont.]

- Both the ngRouter & UI Router have the ability to handle that initialization for us
- Within their providers we are able to inject dependencies into the route
- This is useful when we have services that take time to get initialized
  - (i.e. an API call)
- The controller won't be initialized until the injected resources are finished loading

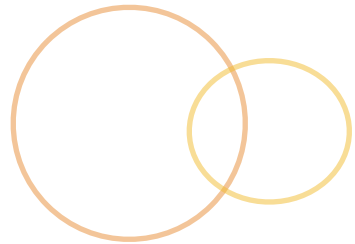
Router Helps [cont.]

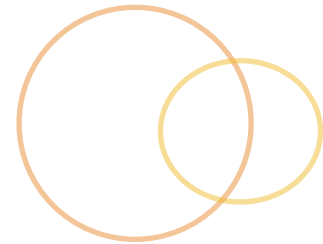
- Both the ngRoute & UI Router have the ability to handle that initialization for us
  - The information returned from our service call will then be available for us to inject into our controller

```
resolve: {
  //Needed service that will take time to resolve
  aNeededService: 'ANeededService',
  //Returned supplies if needed
  importantValues: function(aNeededService) {
      return aNeededService.init();
  }
}
```

Promises

Promises





- Sometimes asynchronous interactions can take a long time to complete
  - (e.g. Geolocation)
- When we don’t want to have to monitor a background task, like fetching the geolocation
  - We don’t want to simply rely on callbacks because they don’t guaranteed a response
  - We don’t want to rely on events because they also have no guarantee

Promises [cont.]

- They assist in decoupling code
- They allow for receiving information in an asynchronous way
- They guarantee a returned interaction
  - We will create a deferred
  - It will immediately returns a promise object
- They specify an easily understandable flow of code
  - Making debugging easier

Deferreds

- Used within a service
- Creates our promise object
- We use its API to invoke resolve / reject on the promise
  - https://docs.angularjs.org/api/ng/service/$q

Deferreds [cont.]

- What the code looks like

```
//Creates a deferred object
var deferred = $q.defer();

//Used to resolve the promise
deferred.resolve(successObject)

//Used to reject the promise
deferred.reject(failureMessage)

//Used for the immediately resolved promise object
return deferred.promise
```

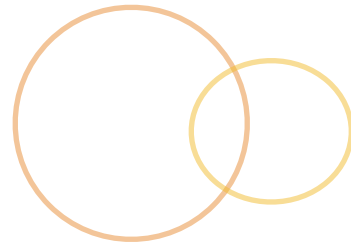
Promise Object

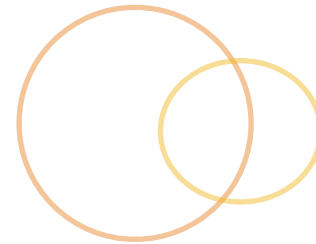
- Returned from our Service utilizing a deferred object
- **then**:
  - Handles the callbacks for success/resolve and error/reject
- **catch**:
  - Shorthand way to interact with a reject
- **finally**:
  - Handles promise fulfillment whether rejected or resolved

.then

- Takes three parameters
- success callback
  - Called if promised is resolved
- failure callback
  - Called if promise is rejected
- notify callback
  - Called when promise is notified
  - Could be used to give updates to users about how long things
- With $http each callback is provided the whole response

Promises

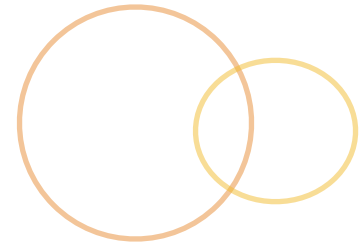


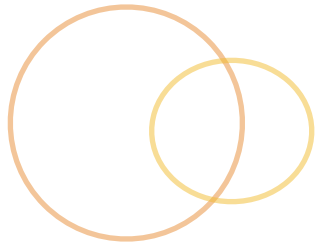


- What the code looks like

```
//Creates a deferred object
promise.then(successFunction, errorFunction, notifyFunction);

//Used to resolve the promise
promise.catch(errorFunction)

//Used to reject the promise
//.finally(alwaysFunction) doesn't work with IE
promise['finally'](function() {});
```

Geolocation Basics





Why Geolocation

- Allows people to know where they are and where stuff is around them
- Wouldn't it be nice to know the barometric pressure?
- Wouldn't it be nice to know where the nearest Redbox is?

Geolocation

- Not a first class member of HTML5
- A W3C standard
- A JavaScript API

Position Makeup

- Coordinate system is based on longitude and latitude
  - Latitude:
    - Measures how far north or south from the equator
    - Equator is at 0
    - North Pole is at 90 or 90 North
    - South Pole is at -90 or 90 South
    - The Latitude range is from -90 to 90

Position Makeup [cont.]

- Coordinate system is based on longitude and latitude
  - Longitude:
    - Measured how far east or west from the Prime Meridian
    - The Prime Meridian is 0 (i.e. Greenwich, England)
    - The Longitude range is from -180 to 180 or 180 W to 180 E
    - West longitudes are preceded with a negative sign

Position Markup

- Where is Boulder CO?
- Normal every day usage is in degrees/minutes/seconds format with cardinal direction
  - (Latitude, Longitude)
  - (40º1’3”, -105º16’47”) … 40º1’3” N / 105º16’47” W
- Programmatically we get decimals
  - (Latitude, Longitude)
  - 40.0176, -105.2797

Get the Device Position

Sounds Complicated

- Not really … Your browser does the hard work
  - It “rolls” through the options available and picks the most accurate positioning it has to offer

```
var getMyLocation = (function (displayLocation) {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(function(position) {
      console.log(“latitude: " + position.coords.latitude +
        “\nlongitude: " + position.coords.longitude);
    });
  } else {
      console.log(“no location available");
  }
}());
```

Get the Current Location

- Let's dissect this method signature
  - **successHandler**: Function invoked if the location was found
  - **errorHandler**: Function invoked if the browser couldn't get a location
  - **options object:** Extra options to tweak geolocation

```
navigator.gelolocation.getCurrentPosition(
  successHandler, errorHandler, options);
```

Success Handler

- Function invoked if the location was found
  - Passed a **position** object, as an argument, containing the location information
  - Position contains coords and timestamp
    - **coords**: A coordinates object
    - **timestamp**: A timestamp in milliseconds
      - You can simply create a new Date() out of it
      - Let's you know how old the location is

```
new Date(position.timestamp);
```

Success Handler [cont.]

- Coordinates object
  - **latitude**: North / South measurement
  - **longitude**: East / West measurement
  - **accuracy**: Accuracy of longitude and latitude specified in meters
  - **speed** (optional): Current ground speed of the device
  - **heading** (optional): Direction of travel of the device, measured in degrees from 0^{o} to 360^{o} clockwise
  - **altitude** (optional): Height of the device above sea level
  - **altitudeAccuracy** (optional): Accuracy of height specified in meters

Error Handler

- Function invoked if the browser couldn't get a location
- There is an **error** object argument given to the errorHandler
  - That error object has a **message** and a **code** property
  - Code 0: Unknown error
    - Catchall error used when nothing else makes sense
    - The **error.message** property gives more information
  - Code 1: Permission denied by the user
    - The user shut you down and they don't want you to know where they are

```
navigator.gelolocation.getCurrentPosition(
  successHandler, errorHandler, options);
```

Error Handler [cont.]

- There is an **error** object argument given to the errorHandler
  - Code 2: Position is not available
    - The browser tried to get the position, but failed =(
    - The **error.message** property gives more information
  - Code 3: Request timed out
    - The browser had an internal timeout that was triggered before it was able to retrieve a location
    - This timeout can be set to a longer/shorter interval
  - Errors with codes 0 or 2 have extra information

```
navigator.gelolocation.getCurrentPosition(
  successHandler, errorHandler, options);
```

Position Options

- Three properties that can be set (all optional)
  - **enableHighAccuracy**: boolean
    - Defaults to false
    - Might make location resolution slower
    - Tries to retrieve exact location
    - iPhones and Androids have separate permissions for high-accuracy positioning

```
navigator.gelolocation.getCurrentPosition(
  successHandler, errorHandler, options);
```

Position Options [cont.]

- Three properties that can be set (all optional)
  - **timeout**:
    - Defaults to infinity
    - Defined in milliseconds
    - How long the application is willing to wait for the position
    - Starts counting after user gives permission for the search

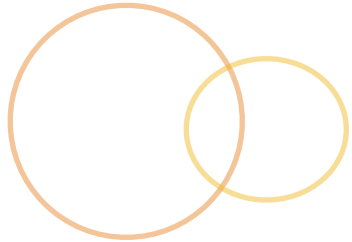
```
navigator.gelolocation.getCurrentPosition(
  successHandler, errorHandler, options);
```

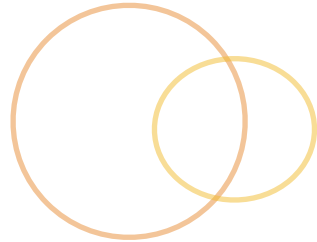
Position Options [cont.]

- Three properties that can be set (all optional)
  - **maximumAge**:
    - Defaults to 0
    - Defined in milliseconds
    - Allows the browser to answer with a cached position if it is not longer than the allotted time

```
navigator.gelolocation.getCurrentPosition(
  successHandler, errorHandler, options);
```

Lab 10





- Create a Supplies Service API
  - Have a $http service to hit /data/initial-supplies for product quantities
    - Transform the data coming from the back-end for the Supplies Service
  - Utilize the Supplies Service API within the Supplies Service for initialization of quantities
  - Have a $http service to hit /data/cost.json for product cost
  - Utilize the Supplies Service API within the Transaction Service for initialization of costs

Lab 10 [cont.]

- Let your app make a transaction without having to go to the supplies page first
  - Utilize the Transaction Service and Supplies Service within the Sell Controller
- Add error messages in a constant service
  - Allow for all the services to have the same display errors (i.e. Server is down)
- Use router resolves for route controller injection
  - Have the Supplies Service be initialized within the resolve property for the **supplies ui-router state**

Lab 10 [cont.]

- Create a Geolocation services
  - Factory/Service: that displays user current latitude/longitude on the form page
- Transform the data coming from the back-end for the Supplies Service
  - **/data/initial-supplies** is the supplies back-end service you will need

Lab 10 [cont.]

- Develop a service to handle the donor form submission
  - At this point only give it the ability to add a donor to the backend
  - /data/philanthropists/:donor is the back-end service

UI Router

- Not only can the UI Router give us nested states it also allows for multiple views on a page
  - These can help to slim down the size of the templates we are using
- In order to keep track of what view is what we are given named views
  - There can only be 1 un-named view in a state

UI Router [cont.]

- Named Views

```
<div ui-view="sidebar"></div>
<div ui-view></div>
```

- We assign multiple views through the **views** property on the state
  - No url property is assigned to the multiple views that property is set on the state holding these views
  - Each of our views will be able to get its own controller

```
views: {
  '': {
    templateUrl: '/home.html'
  },
  'sidebar': {
    templateUrl: '/sidebar.html',
    controller: 'SidebarController'
  }
}
```

UI Router [cont.]

- Sometimes we don't want the state we create to be reachable by a URL
  - Abstract State
- When we have a state that doesn't have a resolvable URL we use the **abstract** property on that state and set it to **true**
  - If someone tries to land on the state URL directly nothing will happen
  - It is a good opportunity to use a redirect to a nested state

UI Router [cont.]

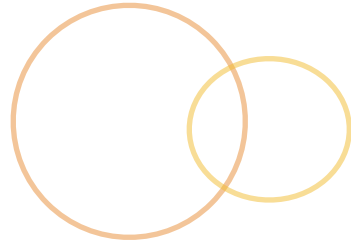
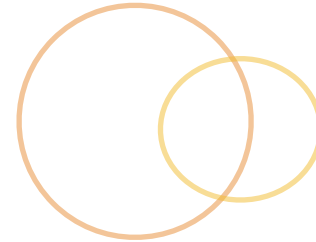
- There is another way for nested states to interact with their parents
  - Via the **parent** property
  - A bit simpler to keep track of

```
$stateProvider.state('nested', {
  url: '/nested',
  parent: 'parent',
  templateUrl: 'parent-nested.html,
  controller: NestedController
});
```

State Parameters

- When wanting to deal with state parameters we will utilize the $stateParams object
  - It will allow us to use . notation to grab parameters we need

State


- Some thought is needed to get our active css class addition working with nested states
  - At this point the simplest way is to utilize the ng-class directive to add an active class
  - However, if our link is not to a parent, but a child the **ui-sref-active** directive won't quite do it
- $state.**includes**('someState')
  - The way ui-router gives us for determining the active state
  - Allows us to check if the active state includes part of the state
    - We can check to see if the parent or child state is included in the current state
    - We can evaluate it in our ng-class directive

State

- To have access to the $state in the view we will need to add it to the $rootScope
  - We can set this up in a run block of the application
  - We want it right at the beginning of our application kick-off

Setting id HTML

- Instead of statically writing our “id” for each page in our single page application we can do it dynamically
  - ng-attr-id directive allows you to dynamically add an id

Underscore

- Sometimes its useful to grab extra JavaScript functionality
- Underscore.js: Very helpful with data structures
  - http://underscorejs.org/
- Lo-Dash: Drop in replacement for Underscore
  - http://lodash.com/
  - Runs a bit quicker

Restangular

Restangular

- Angular service specifically built to interact with RESTful services
- It is promise based so you don't have to call the $promise object off of $resource interaction
  - Simplifies router resolvers
  - Similar to the $http service
- All the HTTP methods are supported
  - $resource doesn't utilize PUT
- Allows for creation of custom methods

Restangular

- Restangular web site
  - https://github.com/mgonto/restangular

- bower install restangular
- Inject it into our app as ‘restangular’

- Note: Restangular needs underscore/lo-dash as a dependency

Restangular

- Restangular is object based
  - When we create our Restangular instance we are setting it up to grab objects from a route
- Setting up Restangular to interact with cars route
  - i.e. /cars for the back-end service

```
Cars = Restangular.all('cars')
```

Restangular

- To get all the resources we use getList()
  - Uses GET and interacts with /cars
  - It will return a promise

```
Cars = Restangular.all('cars');

Cars.getList();
```

Restangular

- We can get single objects
  - Instead of using **all** we use **one**
- Restangular is able to construct URLs for us
  - Allowing us to focus on the objects

- Get all the Mazda models
  - /cars/mazda/models
  - We didn't need to store a URL because it is created for us automatically

```
mazda = Restangular.one('cars', 'mazda');

mazda.get().then(function(car) {
  car.getList('models');
});
```

Restangular

- We access HTTP methods by their lower-case verb
  - .get() : GET
  - .getList() : GET all
  - .post() : POST
  - .put() : PUT
  - .remove() : DELETE

Restangular

- If our web service has a base URL we can configure this with our RestangularProvider
  - RestangularProvider.setBaseUrl('/back/end/data');
- It will attach that base URL to every request

Restangular Interceptors

- Restangular allows us to intercept incoming and out going requests
  - Useful when we need to massage the request/response data
  - Configure this with the RestangularProvider

Restangular Interceptors [cont.]

- setResponseInterceptor:
  - data - The data retrieved from server
  - operation - Lower-case HTTP method (i.e. put)
  - what - The string of the model we are getting (i.e. ‘cars’)

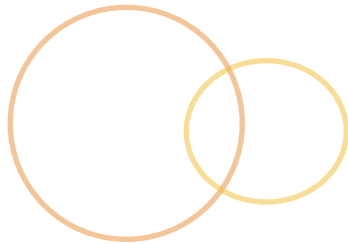
```
RestangularProvider.setResponseInterceptor(
        function(data, operation, what) {
        //Do some transformations
        return tranformedData;
});
```

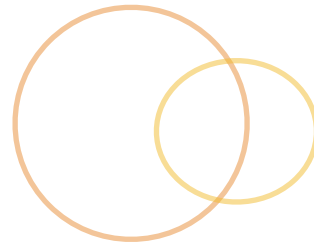
Restangular Interceptors [cont.]

- setRequestInterceptor:
  - element - The object being sent
  - operation - Lower-case HTTP method (i.e. put)
  - what - The string of the model we are getting (i.e. ‘cars’)

```
RestangularProvider.setRequestInterceptor(
        function(element, operation, what) {
        //Do some transformations
        return tranformedElement;
});
```

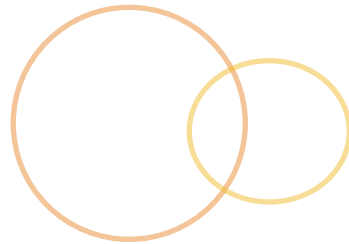
Lab 11 [cont.]

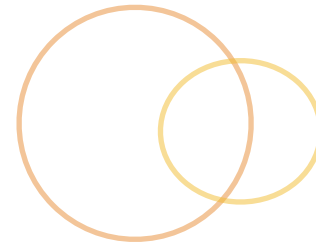




- Place 3 overarching views in our app
  - Header view, Footer view, ‘’ (i.e. the content view)
- Make a transaction API service
  - It has been started as a $resource move it to use Restangular
  - Utilize the provider to set the base url for all Restangular requests to use ‘/data’ as the root
  - Utilize the Restangular request interceptor to transform our request of parameter names going to the back-end
  - /data/transactions/:transaction is the backend service
  - Have our **Sell page** post completed transactions to the backend

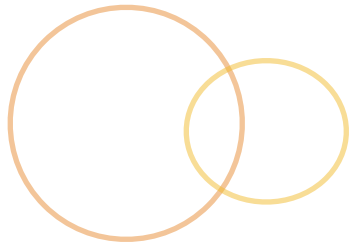
Lab 11 [cont.]

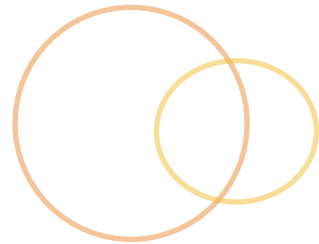




- Modify our reports page
  - The reports state should be abstract
  - Make it have **sales history nested state** (you already have this)
  - Make it have a **transactions nested state** that lists out all the running transactions
    - Show the transaction information
    - Use the resolve property to for getting the Transactions from the Transaction API service before the view is loaded
    - Set the returned transactions onto the $scope via this resolved dependency

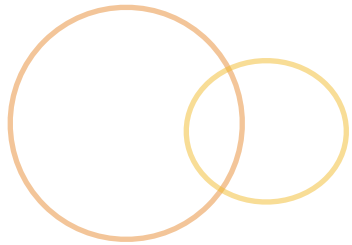
Lab 11 [cont.]

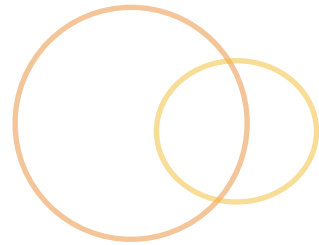




- Modify our reports page …
  - Make it have a **donors nested state** that lists out all the donors
    - Will need to resolve the Donors/Philanthropists API Service
    - Make your donor list filterable by **name** and **best contact**
  - Make a **nested donor state within the donors state** to show a specific donor with full information
    - Allow that donor to be deleted
    - After deletion send the user to the donors state
    - Will need to use the parent's resolved donors
    - Will need to find the donor object utilizing **$stateParams**

Lab 11 [cont.]

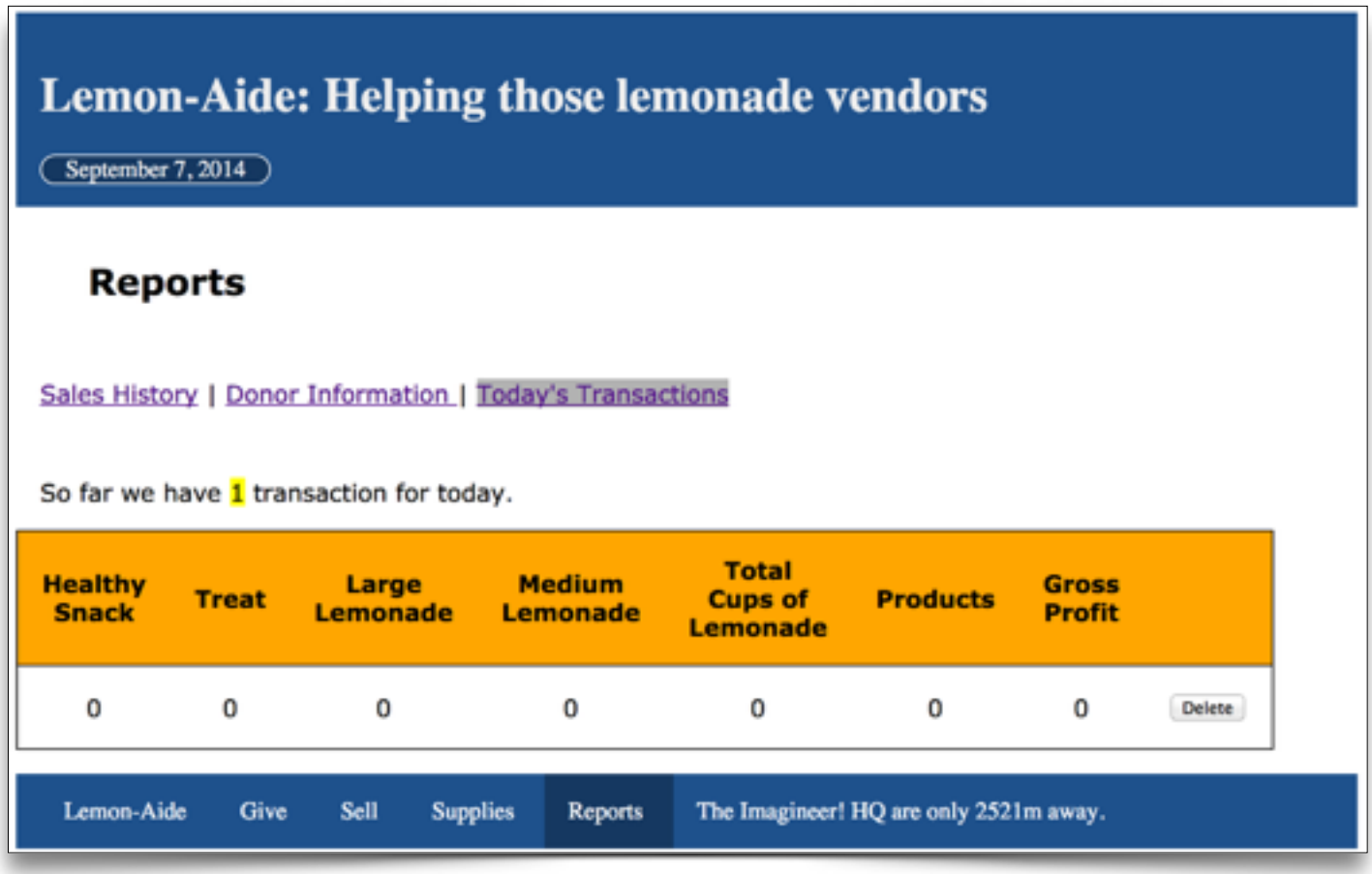




- Dynamically add your page title to your HTML element **id**
  - Make it lowercase so our CSS still works
- Make your routes utilize the “parent” property
- Create another Geolocation service as a provider
  - Have it find the distance between us and Alex’s Lemonade Stand Headquarters that will display in footer
  - Within a config block set the coordinates for Alex’s Lemonade stand to pass into the Geolocation service

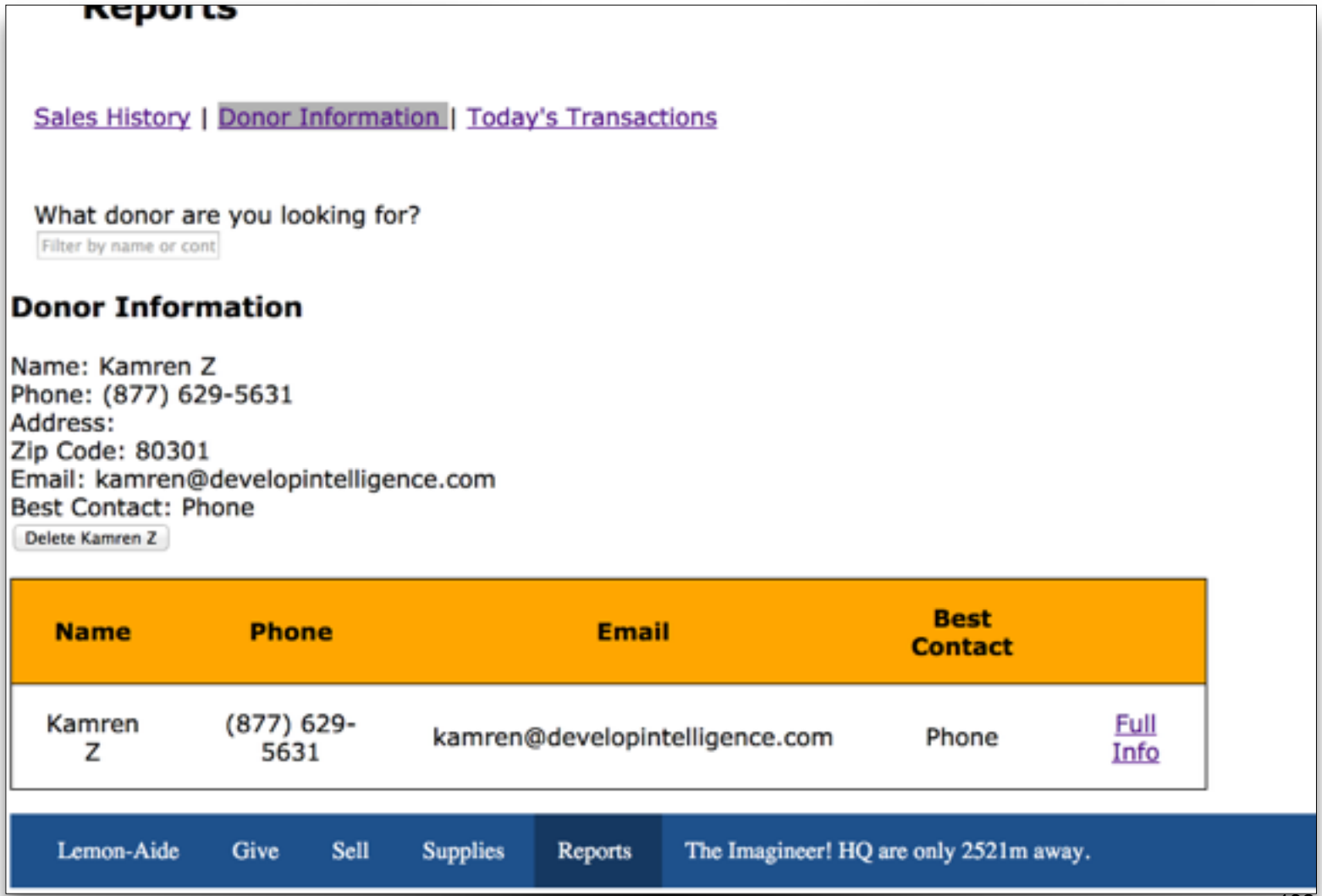
Lab 11 [cont.]

Transaction View



Lab 11 [cont.]

- Donors View & Donor Sub-view



Lab 11 [cont.]

- Sales History View

End