AngularJS and AJAX

Objectives

- Learn how to do AJAX requests in an AngularJS application
- Use the $http service
- Use the $resource service
- Use the $httpProvider to configure HTTP requests
- Unit testing AJAX code using the $httpBackend

$http service

- The basic service for doing all HTTP requests
  - The building block for all AJAX requests
- Can be used as a function
  - $http(config)
- Provides a number of shortcut methods
  - $http.post(url, config)
  - $http.get(url, config)
  - $http.put(url, config)
  - $http.delete(url, config)
- Uses the promisses API as the result
  - Provided by the $q service

$http service – Getting an array of books

```
function BooksListCtrl($scope, $http) {
    $http.get("/api/books")
        .success(function (books) {
            $scope.books = books;
        })
        .error(function (data, status) {
            $scope.data = data || "Request failed";
            $scope.status = status;
        });
}
```

$http service – Editing a single book

```
function BookEditorCtrl($scope, $routeParams, $http, $location) {
    $http.get("/api/books/" + $routeParams.id)
        .success(function (book) {
            $scope.book = book;
        });

    $scope.save = function () {
        $http.put("/api/books/" + $routeParams.id, $scope.book)
            .success(function () {
                $location.path("/books");
            })
            .error(function (errorText, status) {
                $scope.errorText = errorText || "Request failed";
                $scope.status = status;
            });
    };
    $scope.cancel = function () {
        $location.path("/books");
    };
}
```

REST using the $resource

- Creates a service for working with RESTful services
  - Much easier than using the $http object
- Standard functions are already preconfigured
  - Only the common HTTP PUT is missing
- Requires a dependency on ngResource
  - Located in angular-resource.js

```
function BooksListCtrl($scope, Books) {
    $scope.books = Books.query();
}
```

$resource – Creating the Books $resource

- Create a factory function
  - Take a dependency on ngResource
- Always specify the URL template
  - Optionally parameters and extra methods

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

app.factory("Books", function ($resource) {
    return $resource('/api/books/:id', {
        id: '@id'
    }, {
        update: {
            method: 'PUT'
        }
    });
});
```

$resource – Editing a single book

```
function BookEditorCtrl($scope, $routeParams, Books, $location) {
    $scope.book = Books.get({ id: $routeParams.id });

    $scope.save = function () {
        $scope.book.$update(function () {
            $location.path("/books");
        }, function (error, status) {
            $scope.errorText = error.data || "Request failed";
            $scope.status = error.status;
        });
    };

    $scope.cancel = function () {
        $location.path("/books");
    };
}
```

$httpProvider

- Used to configure default behavior for all $http requests
- Add default HTTP headers
- Intercept and transform HTTP requests
- Intercept and transform HTTP responses

Add or replace HTTP headers

- Changes the headers for each $http request
  - Also possible on a per request basis

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

app.config(function ($httpProvider) {
    $httpProvider.defaults.headers.common['Angular'] = 'Cool';
    $httpProvider.defaults.headers.put['X-Requested-With'] = 'Angular';
});
```

ResponseInterceptor as an AJAX busy indicator

```
app.config(function ($httpProvider) {
    var requests = 0;
    function show() { requests++; }
    function hide() {
        requests--;
        if (!requests) {
        }
    }

    $httpProvider.interceptors.push(function ($q) {
        return {
            'request': function (config) {
                show();
                return $q.when(config);
            }, 'response': function (response) {
                hide();
                return $q.when(response);
            }, 'responseError': function (rejection) {
                hide();
                return $q.reject(rejection);
            }
        };
    });
});
```

Unit testing AJAX code using the $httpBackend

- The $httpBackend is the service that is responsible
  - Use the XMLHttpRequest object
- There is a second version in the ngMock module
  - Used for unit testing code that does HTTP requests
- Can be configured to fake HTTP requests
  - Or verify that HTTP calls where made

Unit testing the Books $resource using Jasmine

```
describe("Books service", function () {
    var $httpBackend, service;

    beforeEach(module('booksApp'));

    beforeEach(inject(function (_$httpBackend_, Books) {
        $httpBackend = _$httpBackend_;
        $httpBackend.when("GET", "/api/books").respond([{}]);

        service = Books;
    }));

    it("should return data from query", function () {
        var books = service.query();
        $httpBackend.flush();
        expect(books.length).toBe(1);
    });
});
```

Summary

- Doing any AJAX request is easy
  - Use the $http service
- REST services are even easier
  - Use the $resource to wrap $http calls
- Common AJAX setup can be done just once
  - Using the $httpProvider
- AJAX requests can be faked in unit tests
  - With the mock $httpBackend service