Front-end RESTful Back-end Connection

Connecting AngularJS Front-end with RESTful Express-Node Back-end

CORS

- **CORS**: Cross-Origin Resource Sharing
- **Cross-Origin Request**: A resource makes a cross-origin HTTP request when it requests a resource from a different domain than the one which served itself
- **Idea:** a mechanism that *allows restricted resources* (e.g. fonts, AJAX requests) on a web page to be *requested from another domain* outside the domain from which the resource originated

Adding CORS support to back-end server

- Most of the heavy lifting for CORS is handled between the browser and the server
- The *browser adds* some *additional headers*, and sometimes makes additional requests, during a CORS request on behalf of the client
- These additions are hidden from the client (but can be discovered using a packet analyzer such as Wireshark)
- Server needs to be configured to support CORS

Setup back-end to support CORS

- Install CORS middleware in back-end application

  > npm install cors --save

- Add these lines in **app.js** to reference and use the cors middleware

```
var cors = require('cors');

        // AND

app.use(cors());
```

Setup front-end app to interact with back-end

- Define environment constant in front-end app

```
.constant('environment', 'staging')
```

- Install ng-resource module to interact with RESTful server-side data source

```
> npm install ng-resource --save
```

- Configure $resourceProvider to disable trailing slash stripping in front-end app

```
app.config(['$resourceProvider', function($resourceProvider) {
        $resourceProvider.defaults.stripTrailingSlashes = false;
}]);
```

Using $resource service

- Load the ***ng-resource*** module in **index.html**

- In the module where services are defined (e.g., src/js/services.js) configure environment for interaction with REST api
  - Use factory method on the module to create EnvConfig service
  - EnvConfig returns the appropriate environment configuration with baseURL and suffix for api enpoints

- Create Custom service using EnvConfig and $resource
  - $resource -- A factory that creates a resource object that lets you interact with RESTful server-side data sources
  - The resource object has action methods that provide **high-level behaviors** without the need to interact with the **low level $http service**

Using $resource service

- $resource service is used as a function that takes up to 4 arguments

  $resource(url, [paramDefaults], [actions], options);

- **url** is a string -- a parameterized URL template with parameters prefixed by : as in /user/:username
  - E.g., EnvConfig.api.baseUrl + 'contact/:id' + EnvConfig.api.suffix;

- **paramDefaults** is an object that provides default values for url parameters. These can be overridden in actions method

Using $resource service (2)

- $resource service is used as a function that takes up to 4 arguments

  $resource(url, [paramDefaults], [actions], options);

- **actions** is an object with declaration of custom actions that should extend the default set of resource actions
  - {action1: {method:?, params:?, isArray:?, headers:?, ...},
    action2: {method:?, params:?, isArray:?, headers:?, ...}, ...}

- **options** is an object that provides custom settings that should extend the default $resourceProvider behavior
  - stripTrailingSlashes AND cancellable are the supported options

$resource params

- **url** is required
- **paramDefaults** is optional
- **actions** is optional
- **options** is optional

Resource object returned

- A resource "class" object with methods for the default set of resource actions optionally extended with custom actions

- The default set contains these actions:

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

Custom PUT request

- The params with id is optional since the default value of id is already specified
- id value can be overridden in update action

```
return $resource(url, {id: '@_id'}, {
        update: {
                method: 'PUT'
        }
 });

                    USAGE

Contact.update({id: contact._id}, contact);
```

Instance of resource "class" object

- Calling these methods invoke an $http with the specified http method, destination and parameters
- When the data is returned from the server then the **object is an instance of the resource class**
- **The actions save, remove and delete are available on it as methods with the $ prefix**
- This allows you to easily perform CRUD operations (create, read, update, delete) on server-side data

Example

- If the parameter value is **prefixed with @** then the value for that parameter will be extracted from the corresponding property on the data object (provided when calling an action method)

```
var User = $resource('/user/:userId', {userId:'@id'});

var user = User.get({userId:123}, function() {
      user.abc = true; user.$save();
});
```

Resources

- http://www.html5rocks.com/en/tutorials/cors/
- https://www.npmjs.com/package/cors
- http://viralpatel.net/blogs/angularjs-service-factory-tutorial/
- https://docs.angularjs.org/api/ngResource/service/$resource
- http://www.sitepoint.com/creating-crud-app-minutes-angulars-resource/