RestAngular

**Using Restangular**

**Creating Main Restangular object**

There are 3 ways of creating a main Restangular object. The first one and most common one is by stating the main route of all requests. The second one is by stating the main route and object of all requests.

// Only stating main route

Restangular.all('accounts')

// Stating main object

Restangular.one('accounts', 1234)

// Gets a list of all of those accounts

Restangular.several('accounts', 1234, 123, 12345);

This is all you need to start using all the basic Restangular features.

// Add Restangular as a dependency to your app

angular.module('your-app', ['restangular']);

// Inject Restangular into your controller

angular.module('your-app').controller('MainCtrl', function($scope, Restangular) {

// ...

});

### Let's code!

Now that we have our main Object let's start playing with it.

// First way of creating a Restangular object. Just saying the base URL

var baseAccounts = Restangular.all('accounts');

// This will query /accounts and return a promise.

baseAccounts.getList().then(function(accounts) {

$scope.allAccounts = accounts;

});

var newAccount = {name: "Gonto's account"};

// POST /accounts

baseAccounts.post(newAccount);

// GET to http://www.google.com/ You set the URL in this case

Restangular.allUrl('googlers', 'http://www.google.com/').getList();

// GET to http://www.google.com/1 You set the URL in this case

Restangular.oneUrl('googlers', 'http://www.google.com/1').get();

// You can do RequestLess "connections" if you need as well

// Just ONE GET to /accounts/123/buildings/456

Restangular.one('accounts', 123).one('buildings', 456).get()

// Just ONE GET to /accounts/123/buildings

Restangular.one('accounts', 123).getList('buildings')

// Here we use Promises then

// GET /accounts

baseAccounts.getList().then(function (accounts) {

// Here we can continue fetching the tree :).

var firstAccount = accounts[0];

// This will query /accounts/123/buildings considering 123 is the id of the firstAccount

$scope.buildings = firstAccount.getList("buildings");

// GET /accounts/123/places?query=param with request header: x-user:mgonto

$scope.loggedInPlaces = firstAccount.getList("places", {query: param}, {'x-user': 'mgonto'})

// This is a regular JS object, we can change anything we want :)

firstAccount.name = "Gonto"

// If we wanted to keep the original as it is, we can copy it to a new element

var editFirstAccount = Restangular.copy(firstAccount);

editFirstAccount.name = "New Name";

// PUT /accounts/123. The name of this account will be changed from now on

firstAccount.put();

editFirstAccount.put();

// PUT /accounts/123. Save will do POST or PUT accordingly

firstAccount.save();

// DELETE /accounts/123 We don't have first account anymore :(

firstAccount.remove();

var myBuilding = {

name: "Gonto's Building",

place: "Argentina"

};

// POST /accounts/123/buildings with MyBuilding information

firstAccount.post("Buildings", myBuilding).then(function() {

console.log("Object saved OK");

}, function() {

console.log("There was an error saving");

});

// GET /accounts/123/users?query=params

firstAccount.getList("users", {query: params}).then(function(users) {

// Instead of posting nested element, a collection can post to itself

// POST /accounts/123/users

users.post({userName: 'unknown'});

// Custom methods are available now :).

// GET /accounts/123/users/messages?param=myParam

users.customGET("messages", {param: "myParam"})

var firstUser = users[0];

// GET /accounts/123/users/456. Just in case we want to update one user :)

$scope.userFromServer = firstUser.get();

// ALL http methods are available :)

// HEAD /accounts/123/users/456

firstUser.head()

});

}, function errorCallback() {

alert("Oops error from server :(");

})

// Second way of creating Restangular object. URL and ID :)

var account = Restangular.one("accounts", 123);

// GET /accounts/123?single=true

$scope.account = account.get({single: true});

// POST /accounts/123/messages?param=myParam with the body of name: "My Message"

account.customPOST({name: "My Message"}, "messages", {param: "myParam"}, {})