Every [AngularJs](http://conceptf1.blogspot.com/2014/04/learning-angularjs-part1.html) application communicates with remote server by making http calls and get the data in form of JSON or XML from remote server and than this data will be show to users with html.  
  
If your application need to interact with the remote HTTP server then AngularJs has ***$http*** service for you. It allows to communicate with backend remote web APIs using JSON or XMLHttpRequest.  
  
  
Here is the general get call using $http service:

|  |  |  |
| --- | --- | --- |
| 01 | // Simple GET request example | |
| 02 | $http.get('/api/url'). |

|  |  |  |
| --- | --- | --- |
| 03 | success(function(data, status, headers, config) { | |
| 04 | // asynchronous callback event will be trigger |

|  |  |  |
| --- | --- | --- |
| 05 | // when the call to URL is successful. | |
| 06 | }). |

|  |  |  |
| --- | --- | --- |
| 07 | error(function(data, status, headers, config) { | |
| 08 | // asynchronous callback event will be trigger |

|  |  |  |
| --- | --- | --- |
| 09 | // when en error occurred calling URL or server returns | |
| 10 | // response with error status. |

|  |  |
| --- | --- |
| 11 | }); |

post call using $http service:

|  |  |
| --- | --- |
| 01 | // Simple POST request example |
| 02 | $http.post('/api/url', {data:'hello! Its post call data!'}). | |

|  |  |  |
| --- | --- | --- |
| 03 | success(function(data, status, headers, config) { | |
| 04 | // asynchronous callback event will be trigger |

|  |  |  |
| --- | --- | --- |
| 05 | // when the call to URL is successful. | |
| 06 | }). |

|  |  |  |
| --- | --- | --- |
| 07 | error(function(data, status, headers, config) { | |
| 08 | // asynchronous callback event will be trigger |

|  |  |  |
| --- | --- | --- |
| 09 | // when en error occurred calling URL or server returns | |
| 10 | // response with error status. |

|  |  |
| --- | --- |
| 11 | }); |

There are many scenarios when you need to capture and make some changes to each request for example you want to insert session token to each web request for the authorization similarly you may need to capture each response to perform some actions on data like global error handling for API calls Interceptors are created for these scenarios.

**Interceptors**

***$httpProvider***contains an array of registered interceptors. Interceptor is an AngularJs factory, you can register it by pushing to httpProvider interceptor array in your application configurations.  
  
There are four different interceptors you can handle, and these four functions should be in your interceptor [factory](http://conceptf1.blogspot.com/2014/05/learning-angularjs-part5.html) if you need to perform custom operations in it:

1. **Request Interceptor:**

A request interceptor will be invoke on each request initialization, you can change request data here like adding authorization token.

1. **Response Interceptor:**

A response interceptor will be invoke on each response from remote server, you can manipulate response here like checking pushing some data to response perform some operations on response values.

1. **Request Error Interceptor:**

A request error interceptor will be invoke if there is some error while requesting remote server, like missing header or internet disconnection. Here you can validate request and resend the request to remote server.

1. **Response Error Interceptor:**

A response error interceptor will be invoke if there is error on backend remote calls like some unhandled exception on server. Here you can handle the request by showing proper message to user or resend the request to same url or alternate if available.

Here is the example of Interceptor factory with all above interceptor functions:

|  |  |
| --- | --- |
| 01 | // Interceptor example for angularJs. |
| 02 | angular.module('app').factory('customInterceptor', ['$q', function($q) { | |

|  |  |
| --- | --- |
| 03 |  |
| 04 | var myInterceptor = { | |

|  |  |
| --- | --- |
| 05 | request : request, |
| 06 | requestError : requestError, | |

|  |  |
| --- | --- |
| 07 | response : response, |
| 08 | responseError : responseError | |

|  |  |  |
| --- | --- | --- |
| 09 | }; | |
| 10 |  |

|  |  |
| --- | --- |
| 11 | // On request success |
| 12 | request: function (config) { | |

|  |  |  |
| --- | --- | --- |
| 13 | // Contains the data about the request before it is sent. | |
| 14 | console.log(config); |

|  |  |
| --- | --- |
| 15 |  |
| 16 | // Return the config or wrap it in a promise if blank. | |

|  |  |  |
| --- | --- | --- |
| 17 | return config || $q.when(config); | |
| 18 | }; |

|  |  |
| --- | --- |
| 19 |  |
| 20 | // On request failure | |

|  |  |
| --- | --- |
| 21 | requestError: function (rejection) { |
| 22 | // Contains the data about the error on the request. | |

|  |  |  |
| --- | --- | --- |
| 23 | console.log(rejection); | |
| 24 |  |

|  |  |  |
| --- | --- | --- |
| 25 | // Return the promise rejection. | |
| 26 | return $q.reject(rejection); |

|  |  |  |
| --- | --- | --- |
| 27 | }; | |
| 28 |  |

|  |  |
| --- | --- |
| 29 | // On response success |
| 30 | response: function (response) { | |

|  |  |  |
| --- | --- | --- |
| 31 | // Contains the data from the response. | |
| 32 | console.log(response); |

|  |  |
| --- | --- |
| 33 |  |
| 34 | // Return the response or promise. | |

|  |  |  |
| --- | --- | --- |
| 35 | return response || $q.when(response); | |
| 36 | }; |

|  |  |
| --- | --- |
| 37 |  |
| 38 | // On response failture | |

|  |  |
| --- | --- |
| 39 | responseError: function (rejection) { |
| 40 | // Contains the data about the error. | |

|  |  |  |
| --- | --- | --- |
| 41 | console.log(rejection); | |
| 42 |  |

|  |  |  |
| --- | --- | --- |
| 43 | // Return the promise rejection. | |
| 44 | return $q.reject(rejection); |

|  |  |  |
| --- | --- | --- |
| 45 | }; | |
| 46 |  |

|  |  |  |
| --- | --- | --- |
| 47 | return myInterceptor; | |
| 48 | }]); |

and then register it to $httpProvider interceptor array.

|  |  |  |
| --- | --- | --- |
| 1 | angular.module('app').config(['$httpProvider', function($httpProvider) { | |
| 2 | $httpProvider.interceptors.push('customInterceptor'); |

|  |  |
| --- | --- |
| 3 | }]); |

### Session Injector (request interceptor)

There are two ways of implementing server side authentication. The first one is to use the traditional Cookie-Based Authentication that uses server side cookies to authenticate the user on each request. The other approach is Token-Based Authentication. When the user logs in, he gets sessionToken from the backend. This sessionToken identifies the user in the server and is sent to the server on each request.   
The following sessionInjector adds x-session-token header to each intercepted request (in case the current user is logged in):

Session Injector

module.factory('sessionInjector', ['SessionService', function(SessionService) {

var sessionInjector = {

request: function(config) {

if (!SessionService.isAnonymus) {

config.headers['x-session-token'] = SessionService.token;

}

return config;

}

};

return sessionInjector;

}]);

module.config(['$httpProvider', function($httpProvider) {

$httpProvider.interceptors.push('sessionInjector');

}]);

And now creating a get request:

Creating a request

$http.get('https://api.github.com/users/naorye/repos');

The configuration object before intercepted by sessionInjector:

Before interceptor

{

"transformRequest": [

null

],

"transformResponse": [

null

],

"method": "GET",

"url": "https://api.github.com/users/naorye/repos",

"headers": {

"Accept": "application/json, text/plain, \*/\*"

}

}

The configuration object after intercepted by sessionInjector:

After interceptor

{

"transformRequest": [

null

],

"transformResponse": [

null

],

"method": "GET",

"url": "https://api.github.com/users/naorye/repos",

"headers": {

"Accept": "application/json, text/plain, \*/\*",

"x-session-token": 415954427904

}

}

### Timestamp Marker (request and response interceptors)

Let’s measure the time it takes to get a backend response using interceptors. It is done by adding a timestamp for each request and response:

Timestamp Marker

module.factory('timestampMarker', [function() {

var timestampMarker = {

request: function(config) {

config.requestTimestamp = new Date().getTime();

return config;

},

response: function(response) {

response.config.responseTimestamp = new Date().getTime();

return response;

}

};

return timestampMarker;

}]);

module.config(['$httpProvider', function($httpProvider) {

$httpProvider.interceptors.push('timestampMarker');

}]);

And now we can do:

Timestamp Marker usage

$http.get('https://api.github.com/users/naorye/repos').then(function(response) {

var time = response.config.responseTimestamp - response.config.requestTimestamp;

console.log('The request took ' + (time / 1000) + ' seconds.');

});

**Examples**

**Authentication Token Injector**

If you are using token based authentication for web APIs in which on authentication call server return you a token & this token is required for all the further calls so server. So now you need to provide this authentication token to all the request so here we can use the interceptor. For this we need request interceptor and need to insert token to request.

|  |  |  |
| --- | --- | --- |
| 01 | angular.module('app').factory('authTokenInjector', ['authenticationService', function(AuthenticationService) { | |
| 02 | var authTokenInjector = { |

|  |  |
| --- | --- |
| 03 | request: function(config) { |
| 04 | if (!AuthenticationService.isAnonymus) { | |

|  |  |  |
| --- | --- | --- |
| 05 | config.headers['x-session-token'] = AuthenticationService.securityToken; | |
| 06 | } |

|  |  |  |
| --- | --- | --- |
| 07 | return config; | |
| 08 | } |

|  |  |
| --- | --- |
| 09 | }; |
| 10 | return authTokenInjector; | |

|  |  |
| --- | --- |
| 11 | }]); |

Now register it to interceptors by pushing it to $httpProvider interceptor array. After this each call will be intercepted and authToken get injected to header. It is global handling for authentication now no need to handle it for individual call.

**Overlay/Pre-loader to show**

You want to show overlay on your page unless all the calls gets completed. For this instead of handling it manually you can write it in interceptors and let this work for your complete application.  
  
To achieve this you can write html on your main index page for loader:

|  |  |  |
| --- | --- | --- |
| 01 | <style type="text/css"> | |
| 02 | div#preloader { |

|  |  |  |
| --- | --- | --- |
| 03 | position: absolute; | |
| 04 | opacity: 0.5; |

|  |  |  |
| --- | --- | --- |
| 05 | display: none; | |
| 06 | left: 0; |

|  |  |
| --- | --- |
| 07 | top: 0; |
| 08 | z-index: 9996; | |

|  |  |
| --- | --- |
| 09 | width: 100%; |
| 10 | height: 100%; | |

|  |  |
| --- | --- |
| 11 | overflow: visible; |
| 12 | background: #333 url('loaderImage.png') no-repeat center center; | |

|  |  |
| --- | --- |
| 13 | } |
| 14 | </style> | |

and html:

|  |  |  |
| --- | --- | --- |
| 1 | <div id="preloader"> | |
| 2 | </div> |

Now write the interceptor factory, which will show the loader when call get started and hide when all the calls get executed.But As you know there are multiple calls on a page so we will use a counter which will be incremented on each request and loader will be hide when request counter is zero:

|  |  |
| --- | --- |
| 01 | // Interceptor example for angularJs. |
| 02 | angular.module('app').factory('overlay', ['$q', function($q) { | |

|  |  |
| --- | --- |
| 03 |  |
| 04 | //initialize counter | |

|  |  |  |
| --- | --- | --- |
| 05 | var requestCounter=0; | |
| 06 |  |

|  |  |  |
| --- | --- | --- |
| 07 | var myInterceptor = { | |
| 08 | request : request, |

|  |  |  |
| --- | --- | --- |
| 09 | requestError : requestError, | |
| 10 | response : response, |

|  |  |  |
| --- | --- | --- |
| 11 | responseError : responseError | |
| 12 | }; |

|  |  |
| --- | --- |
| 13 |  |
| 14 | // On request success | |

|  |  |  |
| --- | --- | --- |
| 15 | request: function (config) { | |
| 16 |  |

|  |  |  |
| --- | --- | --- |
| 17 | //will be incremented on each request | |
| 18 | requestCounter++; |

|  |  |
| --- | --- |
| 19 |  |
| 20 | //show loader if not visible already | |

|  |  |  |
| --- | --- | --- |
| 21 | if(!$('#preloader').is(':visible')){ | |
| 22 | $('#preloader').show(); |

|  |  |  |
| --- | --- | --- |
| 23 | } | |
| 24 |  |

|  |  |  |
| --- | --- | --- |
| 25 | // Return the config or wrap it in a promise if blank. | |
| 26 | //it is required to return else call will not work |

|  |  |  |
| --- | --- | --- |
| 27 | return config || $q.when(config); | |
| 28 | }; |

|  |  |
| --- | --- |
| 29 |  |
| 30 | // On request failure | |

|  |  |  |
| --- | --- | --- |
| 31 | requestError: function (rejection) { | |
| 32 |  |

|  |  |  |
| --- | --- | --- |
| 33 | //decrement counter as request is failed | |
| 34 | requestCounter--; |

|  |  |  |
| --- | --- | --- |
| 35 | hideLoaderIfNoCall(); | |
| 36 |  |

|  |  |  |
| --- | --- | --- |
| 37 | // Return the promise rejection. | |
| 38 | return $q.reject(rejection); |

|  |  |  |
| --- | --- | --- |
| 39 | }; | |
| 40 |  |

|  |  |
| --- | --- |
| 41 | // On response success |
| 42 | response: function (response) { | |

|  |  |
| --- | --- |
| 43 |  |
| 44 | //decrement counter as request is failed | |

|  |  |
| --- | --- |
| 45 | requestCounter--; |
| 46 | hideLoaderIfNoCall(); | |

|  |  |
| --- | --- |
| 47 |  |
| 48 | // Return the response or promise. | |

|  |  |  |
| --- | --- | --- |
| 49 | return response || $q.when(response); | |
| 50 | }; |

|  |  |
| --- | --- |
| 51 |  |
| 52 | // On response failture | |

|  |  |  |
| --- | --- | --- |
| 53 | responseError: function (rejection) { | |
| 54 |  |

|  |  |
| --- | --- |
| 55 |  |
| 56 | //decrement counter as request is failed | |

|  |  |
| --- | --- |
| 57 | requestCounter--; |
| 58 | hideLoaderIfNoCall(); | |

|  |  |
| --- | --- |
| 59 |  |
| 60 | // Return the promise rejection. | |

|  |  |  |
| --- | --- | --- |
| 61 | return $q.reject(rejection); | |
| 62 | }; |

|  |  |
| --- | --- |
| 63 |  |
| 64 | function(){ | |

|  |  |
| --- | --- |
| 65 |  |
| 66 | // check if counter is zero means | |

|  |  |  |
| --- | --- | --- |
| 67 | // no request is in process | |
| 68 |  |

|  |  |  |
| --- | --- | --- |
| 69 | // use triple equals see why | |
| 70 | if(requestCounter === 0) |

|  |  |  |
| --- | --- | --- |
| 71 | $('#preloader').hide(); | |
| 72 | } |

|  |  |
| --- | --- |
| 73 |  |
| 74 | return myInterceptor; | |

|  |  |
| --- | --- |
| 75 | }]); |