

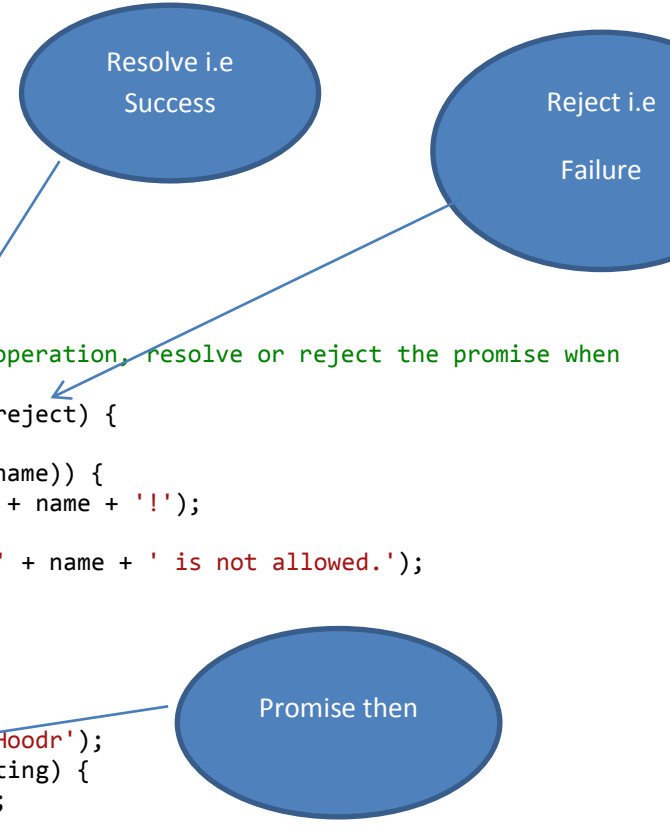
\$q

A service that helps you run functions asynchronously, and use their return values (or exceptions) when they are done processing.

```
angular.module("MyApp").controller("MYCtrl", ["$q", "$scope", function ($q, $scope) {

    $scope.okToGreet = function (name)
    {
        if (name == 'Robin Hood')
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    $scope.gh= function(name) {
        // perform some asynchronous operation, resolve or reject the promise when
        // appropriate.
        return $q(function (resolve, reject) {
            setTimeout(function () {
                if ($scope.okToGreet(name)) {
                    resolve('Hello, ' + name + '!');
                } else {
                    reject('Greeting ' + name + ' is not allowed.');
```



```
            }, 1000);
        });
    }

    $scope.promise = $scope.gh('Robin Hoodr');
    $scope.promise.then(function (greeting) {
        alert('Success: ' + greeting);
    }, function (reason) {
        alert('Failed: ' + reason);
    });
}]);
```

Interceptors

The interceptors are service factories that are registered with the `$httpProvider` by adding them to the `$httpProvider.interceptorsarray`. The factory is called and injected with dependencies (if specified) and returns the interceptor.

There are two kinds of interceptors (and two kinds of rejection interceptors):

- `request`: interceptors get called with a `http config` object. The function is free to modify the `config` object or create a new one. The function needs to return the `config` object directly, or a promise containing the `config` or a new `config` object.
- `requestError`: interceptor gets called when a previous interceptor threw an error or resolved with a rejection.
- `response`: interceptors get called with `http response` object. The function is free to modify the `response` object or create a new one. The function needs to return the `response` object directly, or as a promise containing the `response` or a new `responseobject`.
- `responseError`: interceptor gets called when a previous interceptor threw an error or resolved with a rejection.

```
// register the interceptor as a service
```

The next technique that I want to discuss is creating an interceptor that will modify all requests made with the **\$http** service as well as intercept any response errors. We want to modify the outgoing requests so that we can add an authorization token to the header of our call. Doing it in the interceptor allows us to do it one place and not have to worry about setting it at the individual service level. If the session token has expired and we receive a 401 error from the server we also want to broadcast that status so we can redirect the user to log back in again.

```
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');
}]);
```

```
(
function () {
```

```

var CurrentUserFactoryModule = angular.module("CurrentUserFactoryModule", []);
angular.module("CurrentUserFactoryModule").factory("CurrentUser", CurrentUser);
function CurrentUser() {
    var profile = {
        isLoggedIn: false,
        username: "",
        token: ""
    };
    var setProfile = function (username, token) {
        profile.username = username;
        profile.token = token;
        profile.isLoggedIn = true;
    };

    var getProfile = function () {
        return profile;
    }

    return {
        setProfile: setProfile,
        getProfile: getProfile
    }
}
})();

```

```

var app = angular.module("MyApp", ["ui.router", "CurrentUserFactoryModule"]);

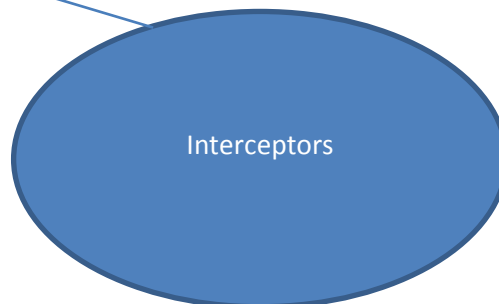
```

```

app.factory('AuthorizationInjector', ["CurrentUser", function (CurrentUser) {
    var sessionInjector = {
        request: function (config) {
            config.headers['Authorization'] = CurrentUser.getProfile().username +
            ":" + CurrentUser.getProfile().token;

            return config;
        }
    };
    return sessionInjector;
}]);
app.config(['$httpProvider', function ($httpProvider) {
    $httpProvider.interceptors.push('AuthorizationInjector');
}]);

```



.config, .run, AppCtrl

Configuration blocks (registered with `module.config()`) get executed during provider registration, and can only be injected providers and constants (see `module.provider()` and `module.constant()`). This is typically where you would configure application-wide stuff, such as the `$routeProvider`. Stuff that needs to be configured before the services are created.

Run blocks (registered with `module.run()`) get executed after the injector has all the providers. Now, all instances and constants can be injected. This is typically where you would configure services, `$rootScope`, events and so on.