Gruntfile.js

```
module.exports = function (grunt) {
  grunt.loadNpmTasks('grunt-html2js');
  grunt.loadNpmTasks('grunt-contrib-less');
  grunt.loadNpmTasks('grunt-contrib-connect');
  grunt.loadNpmTasks('grunt-contrib-watch');
  grunt.loadNpmTasks('grunt-contrib-concat');
  grunt.loadNpmTasks('grunt-concat-sourcemap');
  grunt.loadNpmTasks('grunt-contrib-copy');
  grunt.loadNpmTasks('grunt-contrib-clean');
  grunt.loadNpmTasks('grunt-karma');


  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    html2js: {
      /**
       * These are the templates from `src/app`.
       */
      app: {
        options: {
          base: 'src'
        },
        src: ['src/**/*.tpl.html'],
        dest: 'build/templates-app.js'
      }
    },
    less: {
      all: {
        src: 'style.less',
        dest: 'build/style.css',
        options: {
          report: 'gzip'
        }
      }
    },
    connect: {
      serve: {
        options: {
          port: 8080,
          base: 'build/',
          hostname: '*',
          debug: true
        }
      }
    },
    watch: {
      options: {
        atBegin: true
      },
      templates: {
        files: ['src/**/*.tpl.html'],
        tasks: ['html2js']
      },
      less: {
        files: ['style.less', 'src/**/*.less'],
        tasks: ['less']
      },
      sources: {
        files: ['src/**/*.js', 'src/*.js'],
        tasks: ['concat_sourcemap:app']
      },
      index: {
        files: 'index.html',
        tasks: ['copy:index']
      }
      // Useful for watching / rerunning karma tests
      // jsTest: {
      //    files: ['test/spec/{,*/}*.js'],
      //    tasks: ['karma']
      //}
```

```
    },
    concat_sourcemap: {
      options: {
        sourcesContent: true
      },
      app: {
        src: ['src/**/*.js', 'src/*.js'],
        dest: 'build/app.js'
      },
      libs: {
        src: [
          'libs/angular/angular.js',
          'libs/angular-animate/angular-animate.js',
          'libs/angular-mocks/angular-mocks.js',
          'libs/angular-ui-router/release/angular-ui-router.js'
        ],
        dest: 'build/libs.js'
      }
    },
    copy: {
      index: {
        src: 'index.html',
        dest: 'build/',
        options: {
          processContent: function (content, srcpath) {
            // Compiling index.html file!
            var packageVersion = require('./package.json').version;
            return grunt.template.process(content, {
              data: {
                version: packageVersion
              }
            });
          }
        }
      }
    },
    clean: {
      all: {
        src: ['build/']
      }
    },
    // Test settings
    karma: {
      unit: {
        configFile: 'test/karma.conf.js',
        singleRun: true
      }
    }
  });

  // Build process:
  // - clean build/
  // - creates build/templates-app.js from *.tpl.html files
  // - creates build/style.css from all the .less files
  // - concatenates all the source files in build/app.js - banner with git revision
  // - concatenates all the libraries in build/libs.js
  // - copies index.html over build/
  grunt.registerTask('build', ['clean', 'html2js', 'less', 'concat_sourcemap:app', 'concat_sourcemap:libs', 'copy']);
  grunt.registerTask('default', ['clean', 'concat_sourcemap:libs', 'connect', 'watch']);
  grunt.registerTask('test', ['karma']);
};
```

index.html

```
<!DOCTYPE html>
<html ng-app="BasicHttpAuthExample">

  <head>
    <meta charset="utf-8" />
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.6.2/css/font-awesome.min.css">
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css" />
  </head>

  <body>

    <div class="jumbotron">
      <div class="container">
        <div class="col-xs-offset-2 col-xs-8">
          <div ng-view></div>
        </div>
      </div>
    </div>



    <script src="//code.jquery.com/jquery-2.0.3.min.js"></script>
    <script src="//code.angularjs.org/1.2.20/angular.js"></script>
    <script src="//code.angularjs.org/1.2.20/angular-route.js"></script>
    <script src="//code.angularjs.org/1.2.13/angular-cookies.js"></script>
    <script src="scripts/app.js"></script>
    <script src="modules/authentication/services.js"></script>
    <script src="modules/authentication/controllers.js"></script>
    <script src="modules/home/controllers.js"></script>
  </body>

</html>
```

LICENSE

scripts/app.js

```
'use strict';

// declare modules
angular.module('Authentication', []);
angular.module('Home', []);

angular.module('BasicHttpAuthExample', [
    'Authentication',
    'Home',
    'ngRoute',
    'ngCookies'
])

.config(['$routeProvider', function ($routeProvider) {

    $routeProvider
        .when('/login', {
            controller: 'LoginController',
            templateUrl: 'modules/authentication/views/login.html',
            hideMenus: true
        })

        .when('/', {
            controller: 'HomeController',
            templateUrl: 'modules/home/views/home.html'
        })

        .otherwise({ redirectTo: '/login' });
}])

.run(['$rootScope', '$location', '$cookieStore', '$http',
    function ($rootScope, $location, $cookieStore, $http) {
        // keep user logged in after page refresh
        $rootScope.globals = $cookieStore.get('globals') || {};
        if ($rootScope.globals.currentUser) {
            $http.defaults.headers.common['Authorization'] = 'Basic ' + $rootScope.globals.currentUser.authdata; //
jshint ignore:line
        }

        $rootScope.$on('$locationChangeStart', function (event, next, current) {
            // redirect to login page if not logged in
            if ($location.path() !== '/login' && !$rootScope.globals.currentUser) {
                $location.path('/login');
            }
        });
    }]);
```

src/app.js

```
angular.module('angular-login', [
  // login service
  'loginService',
  'angular-login.mock',
  'angular-login.directives',
  // different app sections
  'angular-login.home',
  'angular-login.pages',
  'angular-login.register',
  'angular-login.error',
  // components
  'ngAnimate'
])
.config(function ($urlRouterProvider) {
  $urlRouterProvider.otherwise('/');
})
.run(function ($rootScope, $window) {
  // google analytics
  $rootScope.$on('$stateChangeSuccess', function (event, toState, toParams) {
    var realURL = toState.url;
    if (!!$window.ga) {
      // resolves variables inside urls, ex: /error/:error in /error/unauthorized
      for (var v in toParams) {
        realURL = realURL.replace(':' + v, toParams[v]);
      }
      $window.ga('send', 'pageview', realURL);
    }
  });
  /**
   * $rootScope.doingResolve is a flag useful to display a spinner on changing states.
   * Some states may require remote data so it will take awhile to load.
   */
  var resolveDone = function () { $rootScope.doingResolve = false; };
  $rootScope.doingResolve = false;

  $rootScope.$on('$stateChangeStart', function () {
    $rootScope.doingResolve = true;
  });
  $rootScope.$on('$stateChangeSuccess', resolveDone);
  $rootScope.$on('$stateChangeError', resolveDone);
  $rootScope.$on('$statePermissionError', resolveDone);
})
.controller('BodyController', function ($scope, $state, $stateParams, loginService, $http, $timeout) {
  // Expose $state and $stateParams to the <body> tag
  $scope.$state = $state;
  $scope.$stateParams = $stateParams;

  // loginService exposed and a new Object containing login user/pwd
  $scope.ls = loginService;
  $scope.login = {
    working: false,
    wrong: false
  };
  $scope.loginMe = function () {
    // setup promise, and 'working' flag
    var loginPromise = $http.post('/login', $scope.login);
    $scope.login.working = true;
    $scope.login.wrong = false;

    loginService.loginUser(loginPromise);
    loginPromise.error(function () {
      $scope.login.wrong = true;
      $timeout(function () { $scope.login.wrong = false; }, 8000);
    });
    loginPromise.finally(function () {
      $scope.login.working = false;
    });
  };
  $scope.logoutMe = function () {
    loginService.logoutUser($http.get('/logout'));
  };
```

```
});
```

src/grandfather.js

```
angular.module('angular-login.grandfather', ['ui.router', 'templates-app'])
.config(function ($stateProvider) {
  $stateProvider
    .state('app', {
      abstract: true,
      template: '<ui-view></ui-view>',
      resolve: {
        'login': function (loginService, $q, $http) {
          var roleDefined = $q.defer();

          /**
           * In case there is a pendingStateChange means the user requested a $state,
           * but we don't know yet user's userRole.
           *
           * Calling resolvePendingState makes the loginService retrieve his userRole remotely.
           */
          if (loginService.pendingStateChange) {
            return loginService.resolvePendingState($http.get('/user'));
          } else {
            roleDefined.resolve();
          }
          return roleDefined.promise;
        }
      }
    });
});
```

src/login-service.js

```
angular.module('loginService', ['ui.router'])
.provider('loginService', function () {
  var userToken = localStorage.getItem('userToken'),
      errorState = 'app.error',
      logoutState = 'app.home';

  this.$get = function ($rootScope, $http, $q, $state) {

    /**
     * Low-level, private functions.
     */
    var setHeaders = function (token) {
      if (!token) {
        delete $http.defaults.headers.common['X-Token'];
        return;
      }
      $http.defaults.headers.common['X-Token'] = token.toString();
    };

    var setToken = function (token) {
      if (!token) {
        localStorage.removeItem('userToken');
      } else {
        localStorage.setItem('userToken', token);
      }
      setHeaders(token);
    };

    var getLoginData = function () {
      if (userToken) {
        setHeaders(userToken);
      } else {
        wrappedService.userRole = userRoles.public;
        wrappedService.isLogged = false;
        wrappedService.doneLoading = true;
      }
    };

    var managePermissions = function () {
      // Register routing function.
      $rootScope.$on('$stateChangeStart', function (event, to, toParams, from, fromParams) {

        /**
         * $stateChangeStart is a synchronous check to the accessLevels property
         * if it's not set, it will setup a pendingStateChange and will let
         * the grandfather resolve do his job.
         *
         * In short:
         * If accessLevels is still undefined, it let the user change the state.
         * Grandfather.resolve will either let the user in or reject the promise later!
         */
        if (wrappedService.userRole === null) {
          wrappedService.doneLoading = false;
          wrappedService.pendingStateChange = {
            to: to,
            toParams: toParams
          };
          return;
        }

        // if the state has undefined accessLevel, anyone can access it.
        // NOTE: if `wrappedService.userRole === undefined` means the service still doesn't know the user role,
        // we need to rely on grandfather resolve, so we let the stateChange success, for now.
        if (to.accessLevel === undefined || to.accessLevel.bitMask & wrappedService.userRole.bitMask) {
          angular.noop(); // requested state can be transitioned to.
        } else {
          event.preventDefault();
          $rootScope.$emit('$statePermissionError');
          $state.go(errorState, { error: 'unauthorized' }, { location: false, inherit: false });
        }
      });
```

```
    /**
     * Gets triggered when a resolve isn't fulfilled
     * NOTE: when the user doesn't have required permissions for a state, this event
     *       it's not triggered.
     *
     * In order to redirect to the desired state, the $http status code gets parsed.
     * If it's an HTTP code (ex: 403), could be prefixed with a string (ex: resolvename403),
     * to handle same status codes for different resolve(s).
     * This is defined inside $state.redirectMap.
     */
    $rootScope.$on('$stateChangeError', function (event, to, toParams, from, fromParams, error) {
      /**
       * This is a very clever way to implement failure redirection.
       * You can use the value of redirectMap, based on the value of the rejection
       * So you can setup DIFFERENT redirections based on different promise errors.
       */
      var errorObj, redirectObj;
      // in case the promise given to resolve function is an $http request
      // the error is a object containing the error and additional informations
      error = (typeof error === 'object') ? error.status.toString() : error;
      // in case of a random 4xx/5xx status code from server, user gets loggedout
      // otherwise it *might* forever loop (look call diagram)
      if (/^[45]\d{2}$/.test(error)) {
        wrappedService.logoutUser();
      }
      /**
       * Generic redirect handling.
       * If a state transition has been prevented and it's not one of the 2 above errors, means it's a
       * custom error in your application.
       *
       * redirectMap should be defined in the $state(s) that can generate transition errors.
       */
      if (angular.isDefined(to.redirectMap) && angular.isDefined(to.redirectMap[error])) {
        if (typeof to.redirectMap[error] === 'string') {
          return $state.go(to.redirectMap[error], { error: error }, { location: false, inherit: false });
        } else if (typeof to.redirectMap[error] === 'object') {
          redirectObj = to.redirectMap[error];
          return $state.go(redirectObj.state, { error: redirectObj.prefix + error }, { location: false, inherit: false
});
        }
      }
      return $state.go(errorState, { error: error }, { location: false, inherit: false });
    });
  };

  /**
   * High level, public methods
   */
  var wrappedService = {
    loginHandler: function (user, status, headers, config) {
      /**
       * Custom logic to manually set userRole goes here
       *
       * Commented example shows an userObj coming with a 'completed'
       * property defining if the user has completed his registration process,
       * validating his/her email or not.
       *
       * EXAMPLE:
       * if (user.hasValidatedEmail) {
       *   wrappedService.userRole = userRoles.registered;
       * } else {
       *   wrappedService.userRole = userRoles.invalidEmail;
       *   $state.go('app.nagscreen');
       * }
       */
      // setup token
      setToken(user.token);
      // update user
      angular.extend(wrappedService.user, user);
      // flag true on isLogged
      wrappedService.isLogged = true;
      // update userRole
```

```
        wrappedService.userRole = user.userRole;
        return user;
      },
      loginUser: function (httpPromise) {
        httpPromise.success(this.loginHandler);
      },
      logoutUser: function (httpPromise) {
        /**
         * De-registers the userToken remotely
         * then clears the loginService as it was on startup
         */
        setToken(null);
        this.userRole = userRoles.public;
        this.user = {};
        this.isLogged = false;
        $state.go(logoutState);
      },
      resolvePendingState: function (httpPromise) {
        var checkUser = $q.defer(),
            self = this,
            pendingState = self.pendingStateChange;

        // When the $http is done, we register the http result into loginHandler, `data` parameter goes into
loginService.loginHandler
        httpPromise.success(self.loginHandler);

        httpPromise.then(
          function success(httpObj) {
            self.doneLoading = true;
            // duplicated logic from $stateChangeStart, slightly different, now we surely have the userRole informations.
            if (pendingState.to.accessLevel === undefined || pendingState.to.accessLevel.bitMask & self.userRole.bitMask)
{
              checkUser.resolve();
            } else {
              checkUser.reject('unauthorized');
            }
          },
          function reject(httpObj) {
            checkUser.reject(httpObj.status.toString());
          }
        );
        /**
         * I setted up the state change inside the promises success/error,
         * so i can safely assign pendingStateChange back to null.
         */
        self.pendingStateChange = null;
        return checkUser.promise;
      },
      /**
       * Public properties
       */
      userRole: null,
      user: {},
      isLogged: null,
      pendingStateChange: null,
      doneLoading: null
    };

    getLoginData();
    managePermissions();

    return wrappedService;
  };
});
```

src/mockhttp.js

```
/* jshint -W084 */
angular.module('angular-login.mock', ['ngMockE2E'])
.factory('delayHTTP', function ($q, $timeout) {
  return {
    request: function (request) {
      var delayedResponse = $q.defer();
      $timeout(function () {
        delayedResponse.resolve(request);
      }, 700);
      return delayedResponse.promise;
    },
    response: function (response) {
      var deferResponse = $q.defer();

      if (response.config.timeout && response.config.timeout.then) {
        response.config.timeout.then(function () {
          deferResponse.reject();
        });
      } else {
        deferResponse.resolve(response);
      }

      return $timeout(function () {
        deferResponse.resolve(response);
        return deferResponse.promise;
      });
    }
  };
})
// delay HTTP
.config(['$httpProvider', function ($httpProvider) {
  $httpProvider.interceptors.push('delayHTTP');
}])
.constant('loginExampleData', {
  version: '0.2.0'
})
.run(function ($httpBackend, $log, loginExampleData) {
  var userStorage = angular.fromJson(localStorage.getItem('userStorage')),
      emailStorage = angular.fromJson(localStorage.getItem('emailStorage')),
      tokenStorage = angular.fromJson(localStorage.getItem('tokenStorage')) || {},
      loginExample = angular.fromJson(localStorage.getItem('loginExample'));

  // Check and corrects old localStorage values, backward-compatibility!
  if (!loginExample || loginExample.version !== loginExampleData.version) {
    userStorage = null;
    tokenStorage = {};
    localStorage.setItem('loginExample', angular.toJson(loginExampleData));
  }

  if (userStorage === null || emailStorage === null) {
    userStorage = {
      'johnm': { name: 'John', username: 'johnm', password: 'hello', email: 'john.dott@myemail.com', userRole:
userRoles.user, tokens: [] },
      'sandrab': { name: 'Sandra', username: 'sandrab', password: 'world', email: 'bitter.s@provider.com', userRole:
userRoles.admin, tokens: [] }
    };
    emailStorage = {
      'john.dott@myemail.com': 'johnm',
      'bitter.s@provider.com': 'sandrab'
    };
    localStorage.setItem('userStorage', angular.toJson(userStorage));
    localStorage.setItem('emailStorage', angular.toJson(emailStorage));
  }

  /**
   * Generates random Token
   */
  var randomUUID = function () {
    var charSet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
    var randomToken = '';
    for (var i = 0; i < 36; i++) {
      if (i === 8 || i === 13 || i === 18 || i === 23) {
```

```
      randomToken += '';
      continue;
    }
    var randomPoz = Math.floor(Math.random() * charSet.length);
    randomToken += charSet.substring(randomPoz, randomPoz + 1);
  }
  return randomToken;
};

// fakeLogin
$httpBackend.when('POST', '/login').respond(function (method, url, data, headers) {
  var postData = angular.fromJson(data),
      user = userStorage[postData.username],
      newToken,
      tokenObj;
  $log.info(method, '->', url);

  if (angular.isDefined(user) && user.password === postData.password) {
    newToken = randomUUID();
    user.tokens.push(newToken);
    tokenStorage[newToken] = postData.username;
    localStorage.setItem('userStorage', angular.toJson(userStorage));
    localStorage.setItem('tokenStorage', angular.toJson(tokenStorage));
    return [200, { name: user.name, userRole: user.userRole, token: newToken }, {}];
  } else {
    return [401, 'wrong combination username/password', {}];
  }
});

// fakeLogout
$httpBackend.when('GET', '/logout').respond(function (method, url, data, headers) {
  var queryToken, userTokens;
  $log.info(method, '->', url);

  if (queryToken = headers['X-Token']) {
    if (angular.isDefined(tokenStorage[queryToken])) {
      userTokens = userStorage[tokenStorage[queryToken]].tokens;
      // Update userStorage AND tokenStorage
      userTokens.splice(userTokens.indexOf(queryToken));
      delete tokenStorage[queryToken];
      localStorage.setItem('userStorage', angular.toJson(userStorage));
      localStorage.setItem('tokenStorage', angular.toJson(tokenStorage));
      return [200, {}, {}];
    } else {
      return [401, 'auth token invalid or expired', {}];
    }
  } else {
    return [401, 'auth token invalid or expired', {}];
  }
});

// fakeUser
$httpBackend.when('GET', '/user').respond(function (method, url, data, headers) {
  var queryToken, userObject;
  $log.info(method, '->', url);

  // if is present in a registered users array.
  if (queryToken = headers['X-Token']) {
    if (angular.isDefined(tokenStorage[queryToken])) {
      userObject = userStorage[tokenStorage[queryToken]];
      return [200, { token: queryToken, name: userObject.name, userRole: userObject.userRole }, {}];
    } else {
      return [401, 'auth token invalid or expired', {}];
    }
  } else {
    return [401, 'auth token invalid or expired', {}];
  }
});

// fakeRegister
$httpBackend.when('POST', '/user').respond(function (method, url, data, headers) {
  var postData = angular.fromJson(data),
      newUser,
```

```
        errors = [];
    $log.info(method, '->', url);

    if (angular.isDefined(userStorage[postData.username])) {
      errors.push({ field: 'username', name: 'used' });
    }

    if (angular.isDefined(emailStorage[postData.email])) {
      errors.push({ field: 'email', name: 'used' });
    }

    if (errors.length) {
      return [409, {
        valid: false,
        errors: errors
      }, {}];
    } else {
      newUser = angular.extend(postData, { userRole: userRoles[postData.role], tokens: [] });
      delete newUser.role;

      userStorage[newUser.username] = newUser;
      emailStorage[newUser.email] = newUser.username;
      localStorage.setItem('userStorage', angular.toJson(userStorage));
      localStorage.setItem('emailStorage', angular.toJson(emailStorage));
      return [201, { valid: true, creationDate: Date.now() }, {}];
    }
  });

});
```

src/routing-config.js

```
/**
 * Directly from fnakstad
 * https://github.com/fnakstad/angular-client-side-auth/blob/master/client/js/routingConfig.js
 */

(function (exports) {

  var config = {

    /* List all the roles you wish to use in the app
    * You have a max of 31 before the bit shift pushes the accompanying integer out of
    * the memory footprint for an integer
    */
    roles: [
      'public',
      'user',
      'admin'
    ],

    /*
    Build out all the access levels you want referencing the roles listed above
    You can use the "*" symbol to represent access to all roles
     */
    accessLevels: {
      'public' : '*',
      'anon': ['public'],
      'user' : ['user', 'admin'],
      'admin': ['admin']
    }

  };

  /*
    Method to build a distinct bit mask for each role
    It starts off with "1" and shifts the bit to the left for each element in the
    roles array parameter
   */
  function buildRoles(roles) {

    var bitMask = "01";
    var userRoles = {};

    for (var role in roles) {
      var intCode = parseInt(bitMask, 2);
      userRoles[roles[role]] = {
        bitMask: intCode,
        title: roles[role]
      };
      bitMask = (intCode << 1).toString(2);
    }

    return userRoles;
  }

  /*
  This method builds access level bit masks based on the accessLevelDeclaration parameter which must
  contain an array for each access level containing the allowed user roles.
   */
  function buildAccessLevels(accessLevelDeclarations, userRoles) {

    var accessLevels = {},
        resultBitMask,
        role;
    for (var level in accessLevelDeclarations) {

      if (typeof accessLevelDeclarations[level] === 'string') {
        if (accessLevelDeclarations[level] === '*') {

          resultBitMask = '';

          for (role in userRoles) {
```

```
            resultBitMask += "1";
          }
          //accessLevels[level] = parseInt(resultBitMask, 2);
          accessLevels[level] = {
            bitMask: parseInt(resultBitMask, 2),
            title: accessLevelDeclarations[level]
          };
        }
        else {
          console.log("Access Control Error: Could not parse '" + accessLevelDeclarations[level] + "' as access
definition for level '" + level + "'");
        }
      }
      else {

        resultBitMask = 0;
        for (role in accessLevelDeclarations[level]) {
          if (userRoles.hasOwnProperty(accessLevelDeclarations[level][role])) {
            resultBitMask = resultBitMask | userRoles[accessLevelDeclarations[level][role]].bitMask;
          }
          else {
            console.log("Access Control Error: Could not find role '" + accessLevelDeclarations[level][role] + "' in
registered roles while building access for '" + level + "'");
          }
        }
        accessLevels[level] = {
          bitMask: resultBitMask,
          title: accessLevelDeclarations[level][role]
        };
      }
    }

    return accessLevels;
  }


  exports.userRoles = buildRoles(config.roles);
  exports.accessLevels = buildAccessLevels(config.accessLevels, exports.userRoles);

})(typeof exports === 'undefined' ? this : exports);
```

test/karma.conf.js

```
// Karma configuration
// http://karma-runner.github.io/0.12/config/configuration-file.html
module.exports = function(config) {
  config.set({
    // enable / disable watching file and executing tests whenever any file changes
    autoWatch: true,

    // base path, that will be used to resolve files and exclude
    basePath: '../',

    // testing framework to use (jasmine/mocha/qunit/...)
    frameworks: ['jasmine'],

    // list of files / patterns to load in the browser
    files: [
      'libs/angular/angular.js',
      'libs/angular-animate/angular-animate.js',
      'libs/angular-mocks/angular-mocks.js',
      'libs/angular-ui-router/release/angular-ui-router.js',
      'src/**/*.js',
      'test/spec/**/*.js'
    ],

    // list of files / patterns to exclude
    exclude: [],

    // web server port
    port: 8080,

    // Start these browsers, currently available:
    // - Chrome
    // - ChromeCanary
    // - Firefox
    // - Opera
    // - Safari (only Mac)
    // - PhantomJS
    // - IE (only Windows)
    browsers: [
      'PhantomJS'
    ],

    // Which plugins to enable
    plugins: [
      'karma-phantomjs-launcher',
      'karma-jasmine'
    ],

    // Continuous Integration mode
    // if true, it capture browsers, run tests and exit
    singleRun: false,

    colors: true,

    // level of logging
    // possible values: LOG_DISABLE || LOG_ERROR || LOG_WARN || LOG_INFO || LOG_DEBUG
    logLevel: config.LOG_INFO

    // Uncomment the following lines if you are using grunt's server to run the tests
    // proxies: {
    //   '/': 'http://localhost:9000/'
    // },
    // URL root prevent conflicts with the site root
    // urlRoot: '_karma_'
  });
};
```

modules/authentication/controllers.js

```
'use strict';

angular.module('Authentication')

.controller('LoginController',
    ['$scope', '$rootScope', '$location', 'AuthenticationService',
    function ($scope, $rootScope, $location, AuthenticationService) {
        // reset login status
        AuthenticationService.ClearCredentials();

        $scope.login = function () {
            $scope.dataLoading = true;
            AuthenticationService.Login($scope.username, $scope.password, function(response) {
                if(response.success) {
                    AuthenticationService.SetCredentials($scope.username, $scope.password);
                    $location.path('/');
                } else {
                    $scope.error = response.message;
                    $scope.dataLoading = false;
                }
            });
        };
    }]);
```

modules/authentication/services.js

```
'use strict';

angular.module('Authentication')

.factory('AuthenticationService',
    ['Base64', '$http', '$cookieStore', '$rootScope', '$timeout',
    function (Base64, $http, $cookieStore, $rootScope, $timeout) {
        var service = {};

        service.Login = function (username, password, callback) {

            /* Dummy authentication for testing, uses $timeout to simulate api call
             ----------------------------------------------*/
            $timeout(function(){
                var response = { success: username === 'test' && password === 'test' };
                if(!response.success) {
                    response.message = 'Username or password is incorrect';
                }
                callback(response);
            }, 1000);


            /* Use this for real authentication
             ----------------------------------------------*/
            //$http.post('/api/authenticate', { username: username, password: password })
            //    .success(function (response) {
            //        callback(response);
            //    });

        };

        service.SetCredentials = function (username, password) {
            var authdata = Base64.encode(username + ':' + password);

            $rootScope.globals = {
                currentUser: {
                    username: username,
                    authdata: authdata
                }
            };

            $http.defaults.headers.common['Authorization'] = 'Basic ' + authdata; // jshint ignore:line
            $cookieStore.put('globals', $rootScope.globals);
        };

        service.ClearCredentials = function () {
            $rootScope.globals = {};
            $cookieStore.remove('globals');
            $http.defaults.headers.common.Authorization = 'Basic ';
        };

        return service;
    }])

.factory('Base64', function () {
    /* jshint ignore:start */

    var keyStr = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=';

    return {
        encode: function (input) {
            var output = "";
            var chr1, chr2, chr3 = "";
            var enc1, enc2, enc3, enc4 = "";
            var i = 0;

            do {
                chr1 = input.charCodeAt(i++);
                chr2 = input.charCodeAt(i++);
                chr3 = input.charCodeAt(i++);
```

```
                enc1 = chr1 >> 2;
                enc2 = ((chr1 & 3) << 4) | (chr2 >> 4);
                enc3 = ((chr2 & 15) << 2) | (chr3 >> 6);
                enc4 = chr3 & 63;

                if (isNaN(chr2)) {
                    enc3 = enc4 = 64;
                } else if (isNaN(chr3)) {
                    enc4 = 64;
                }

                output = output +
                    keyStr.charAt(enc1) +
                    keyStr.charAt(enc2) +
                    keyStr.charAt(enc3) +
                    keyStr.charAt(enc4);
                chr1 = chr2 = chr3 = "";
                enc1 = enc2 = enc3 = enc4 = "";
            } while (i < input.length);

            return output;
        },

        decode: function (input) {
            var output = "";
            var chr1, chr2, chr3 = "";
            var enc1, enc2, enc3, enc4 = "";
            var i = 0;

            // remove all characters that are not A-Z, a-z, 0-9, +, /, or =
            var base64test = /[^A-Za-z0-9\+\/\=]/g;
            if (base64test.exec(input)) {
                window.alert("There were invalid base64 characters in the input text.\n" +
                    "Valid base64 characters are A-Z, a-z, 0-9, '+', '/',and '='\n" +
                    "Expect errors in decoding.");
            }
            input = input.replace(/[^A-Za-z0-9\+\/\=]/g, "");

            do {
                enc1 = keyStr.indexOf(input.charAt(i++));
                enc2 = keyStr.indexOf(input.charAt(i++));
                enc3 = keyStr.indexOf(input.charAt(i++));
                enc4 = keyStr.indexOf(input.charAt(i++));

                chr1 = (enc1 << 2) | (enc2 >> 4);
                chr2 = ((enc2 & 15) << 4) | (enc3 >> 2);
                chr3 = ((enc3 & 3) << 6) | enc4;

                output = output + String.fromCharCode(chr1);

                if (enc3 != 64) {
                    output = output + String.fromCharCode(chr2);
                }
                if (enc4 != 64) {
                    output = output + String.fromCharCode(chr3);
                }

                chr1 = chr2 = chr3 = "";
                enc1 = enc2 = enc3 = enc4 = "";

            } while (i < input.length);

            return output;
        }
    };

    /* jshint ignore:end */
});
```

modules/home/controllers.js

```
'use strict';

angular.module('Home')

.controller('HomeController',
    ['$scope',
    function ($scope) {

    }]);
```

src/directives/form-helpers.js

```
angular.module('angular-login.directives', [])
/**
 * Simple directive to check password equality
 *
 * usage:
 * <input type="password" ng-model="password" password-match="password2">
 * <input type="password" ng-model="password2">
 */
.directive('passwordMatch', function () {
  return {
    restrict: 'A',
    scope: false,
    require: 'ngModel',
    link: function (scope, elem, attrs, controller) {
      var checker = function () {
        // get the value of the first password
        var pwd = scope.$eval(attrs.ngModel);
        // get the value of the other password
        var pwd2 = scope.$eval(attrs.passwordMatch);
        return pwd === pwd2;
      };
      scope.$watch(checker, function (pwdMatch) {
        controller.$setValidity('match', pwdMatch);
      });
    }
  };
})
/**
 * Directive to manage valid/invalid states of remote-validated Data.
 * It stores an internal array of values declared invalid by the server.
 * Generates the form error specified in case the user re-types the same invalid values,
 * clears the errors in case the user changes the ngModel.
 *
 * usage:
 * <input type="email" ng-model="email" remote-validated="used">
 *
 * NOTE: Your controllers have to make the field invalid in case *your* server says so.
 */
.directive('remoteValidated', function () {
  return {
    restrict: 'A',
    scope: false,
    require: 'ngModel',
    link: function (scope, elem, attrs, controller) {
      var invalidItems = [];
      scope.$watch(attrs.ngModel, function (newValue, oldValue) {
        if (newValue) {
          // Check the array of already-bad items
          if (invalidItems.indexOf(newValue) !== -1) {
            return controller.$setValidity(attrs.remoteValidated, false);
          }
          // When the model changes, it checks if the previous value was
          // triggering the error from server-side
          if (controller.$error[attrs.remoteValidated]) {
            invalidItems.push(oldValue);
          }
          controller.$setValidity(attrs.remoteValidated, true);
        }
      });
    }
  };
});
```

src/error/error.js

```
angular.module('angular-login.error', ['angular-login.grandfather'])
.config(function ($stateProvider) {
  $stateProvider
    .state('app.error', {
      url: '/error/:error',
      templateUrl: 'error/error.tpl.html',
      accessLevel: accessLevels.public
    });
});
```

src/error/error.tpl.html

```
<div class="jumbotron">
  <h1>Error</h1>
  <div ng-switch="$stateParams.error">
    <p class="text-danger" ng-switch-when="unauthorized">You are not authorized</p>
    <p class="text-danger" ng-switch-when="401">You are not authorized</p>
    <p class="text-danger" ng-switch-default>Some error has occurred</p>
  </div ng-switch>
</div>
```

src/home/home.js

```
angular.module('angular-login.home', ['angular-login.grandfather'])
.config(function ($stateProvider) {
  $stateProvider
    .state('app.home', {
      url: '/',
      templateUrl: 'home/home.tpl.html',
      controller: 'HomeController'
    });
})
.controller('HomeController', function ($scope) {
  $scope.users = angular.fromJson(localStorage.getItem('userStorage'));
});
```

src/home/home.tpl.html

```
<div class="jumbotron">
  <h1>This is home!</h1>
  <p>Everybody can access this page, the other credentials stored are:</p>
  <div ng-repeat="user in users">
    <h2>{{ user.name }}</h2>
    <p ng-class="{ 'text-info': user.userRole.title === 'user', 'text-danger': user.userRole.title === 'admin' }">
      username: {{ user.username }}, password: {{ user.password }}, email: {{ user.email }}, permission: {{
user.userRole.title }}
    </p>
  </div>
</div>

<h2>native json</h2>
<p>angular provides the real json object</p>
<pre>
  <code>
{{ users | json }}
  </code>
</pre>

<h2>You can keep track of mocked http requests</h2>
<p>Just open the console of your favourite browser and the ngMock will print out the requests as console.info.</p>
```

src/pages/admin.tpl.html

```
<div class="jumbotron">
  <h1>Admin interface</h1>
  <p class="text-danger">Only accessible by <b>admins</b></p>
</div>
```

src/pages/pages.js

```
angular.module('angular-login.pages', ['angular-login.grandfather'])
.config(function ($stateProvider) {
  $stateProvider
    .state('app.admin', {
      url: '/admin',
      templateUrl: 'pages/admin.tpl.html',
      accessLevel: accessLevels.admin
    })
    .state('app.user', {
      url: '/user',
      templateUrl: 'pages/user.tpl.html',
      accessLevel: accessLevels.user
    });
});
```

src/pages/user.tpl.html

```
<div class="jumbotron">
  <h1>Page for registered users</h1>
  <p class="text-info">Both <b>users</b> and <b>admins</b> can access to this page!</p>
</div>
```

src/register/register.js

```
angular.module('angular-login.register', ['angular-login.grandfather'])
.config(function ($stateProvider) {
  $stateProvider
    .state('app.register', {
      url: '/register',
      templateUrl: 'register/register.tpl.html',
      controller: 'RegisterController',
      accessLevel: accessLevels.anon
    });
})
.controller('RegisterController', function ($scope, $http, $timeout, $state) {
  $scope.xhr = false;
  $scope.redirect = false;

  $scope.registerObj = {
    role: 'user'
  };

  $scope.submit = function (formInstance) {
    // xhr is departing
    $scope.xhr = true;
    $http.post('/user', $scope.registerObj)
    .success(function (data, status, headers, config) {
      console.info('post success - ', data);
      $scope.xhr = false;
      $scope.redirect = true;
      $timeout(function () {
        $state.go('app.home');
      }, 2000);
    })
    .error(function (data, status, headers, config) {
      data.errors.forEach(function (error, index, array) {
        formInstance[error.field].$error[error.name] = true;
      });
      formInstance.$setPristine();
      console.info('post error - ', data);
      $scope.xhr = false;
    });
  };
});
```

src/register/register.less

```less
@slate-color: rgb(39, 43, 48);

.form-signin {
  max-width: 400px;
  padding: 15px;
  margin: 0 auto;
  background-color: lighten(@slate-color, 10%);
  box-shadow: 1px 1px 2px 2px lighten(@slate-color, 20%);
  border-radius: 5px;
  .form-control:focus {
    z-index: 2;
  }
  .form-control {
    position: relative;
    font-size: 16px;
    height: auto;
    padding: 10px;
  }
  input {
    // margin-bottom: 10px;
    &.password {
      margin-bottom: -1px;
      border-bottom-left-radius: 0;
      border-bottom-right-radius: 0;
    }
    &.password2 {
      border-top-left-radius: 0;
      border-top-right-radius: 0;
    }
  }
  .form-input {
    margin-bottom: 10px;
    .errors {
      background-color: softlight(@slate-color, white);
      border: 1px solid transparent;
      border-radius: 4px;
      transition: border-color 1s;

      // hides borders if .errors is empty
      &.active {
        border-color: #666666;
      }
      .error {
        margin: 3px 5px;
        white-space: nowrap; // white-space div align
        overflow: hidden;

        // FIXME: needs rework.
        &.ng-hide-add {
          transition: max-height .5s;
          max-height: 60px;
          display: block!important;

          &.ng-hide-add-active {
            max-height: 0px;
          }
        }
        &.ng-hide-remove {
          transition: max-height .5s;
          max-height: 0px;
          display: block!important;

          &.ng-hide-remove-active {
            max-height: 60px;
          }
        }

        // Message inside the .error div
        &:before {
          content: 'Error:';
          display: inline-block;
```

```
          color: #FA3F3C;
          font-weight: 800;
          width: 40px;
          vertical-align: top;
        }
        p {
          display: inline-block;
          white-space: normal; // reset it on children
          margin: 0;
          &:first-child {
            width: 40px;
            vertical-align: top;
          }
          &:last-child {
            width: 316px;
          }
        }
      }
    }
  }
  .form-signin-heading {
    font-weight: 800;
    text-shadow: 2px 2px 4px black;
    margin-bottom: 10px
  }
  .checkbox {
    font-weight: normal;
    margin-bottom: 10px;
  }
}
```

src/register/register.tpl.html

```
<form class="form-signin" name="registerForm" role="registration" ng-submit="submit(registerForm)">
  <div class="alert alert-warning">
    <p class="text-center"><strong>Please NOTE</strong></p>
    <p class="text-center">All the data here is fake, it gets stored in your localStorage, it will
<strong>NEVER</strong> leave your browser.</p>
  </div>
  <h2 class="form-signin-heading"><i class="fa fa-user"></i> New User</h2>
  <!--div class="form-group">
    <label for="username" class="col-lg-2 control-label">Email</label>
    <div class="col-lg-10">
      <input type="text" class="form-control" id="username" placeholder="Username">
    </div>
  </div-->
  <div class="form-input username">
    <input type="text" class="form-control" placeholder="Username" name="username" ng-model="registerObj.username"
autofocus="true"
      ng-minlength="4" ng-maxlength="16" ng-required="true" remote-validated="used">
    <div class="errors" ng-class="{ active: registerForm.username.$invalid && registerForm.username.$dirty }">
      <div class="error ng-hide" ng-show="registerForm.username.$error.minlength">
        <p>Username is too short!</p>
      </div>
      <div class="error ng-hide" ng-show="registerForm.username.$error.maxlength">
        <p>Max username length is 16, please shorten it.</p>
      </div>
      <div class="error ng-hide" ng-show="registerForm.username.$error.used">
        <p>Username is already taken.</p>
      </div>
    </div>
  </div>
  <div class="form-input name">
    <input type="text" class="form-control" placeholder="Real Name" name="name" ng-model="registerObj.name" ng-
minlength="4" ng-maxlength="32" ng-required="true">
    <div class="errors" ng-class="{ active: registerForm.name.$invalid && registerForm.name.$dirty }">
      <div class="error ng-hide" ng-show="registerForm.name.$error.minlength">
        <p>Provided name is too short!</p>
      </div>
      <div class="error ng-hide" ng-show="registerForm.name.$error.maxlength">
        <p>Max name length is 32, please shorten it.</p>
      </div>
    </div>
  </div>
  <div class="form-input password">
    <input type="password" class="form-control password" placeholder="Password" name="password" ng-
model="registerObj.password" ng-minlength="4" ng-maxlength="16" ng-required="true" password-
match="registerObj.password2">
    <input type="password" class="form-control password2" placeholder="Repeat Password" name="password2" ng-
model="registerObj.password2">
    <div class="errors" ng-class="{ active: registerForm.password.$invalid && registerForm.password.$dirty }">
      <div class="error ng-hide" ng-show="registerForm.password.$error.match">
        <p>Passwords do not match.</p>
      </div>
      <div class="error ng-hide" ng-show="registerForm.password.$error.minlength">
        <p>For your own safety, use a password longer than 4 characters.</p>
      </div>
      <div class="error ng-hide" ng-show="registerForm.password.$error.maxlength">
        <p>For your own <b>SANITY</b>, use a password shorter than 16 characters.</p>
      </div>
    </div>
  </div>
  <div class="form-input email">
    <input type="email" class="form-control email" placeholder="E-Mail" name="email" ng-model="registerObj.email" ng-
required="true" remote-validated="used">
    <div class="errors" ng-class="{ active: registerForm.email.$invalid && registerForm.email.$dirty }">
      <div class="error ng-hide" ng-show="registerForm.email.$error.email">
        <p>E-Mail seems invalid.</p>
      </div>
      <div class="error ng-hide" ng-show="registerForm.email.$error.used">
        <p>E-Mail is already taken.</p>
      </div>
    </div>
  </div>
  <div class="form-input role">
```

```
    <select class="form-control" name="role" ng-model="registerObj.role">
      <option value="user" selected>User</option>
      <option value="admin">Administrator</option>
    </select>
  </div>
  <button class="btn btn-lg btn-block" type="submit"
    ng-class="{ 'btn-primary': registerForm.$valid && registerForm.$dirty, 'btn-success': redirect }"
    ng-disabled="registerForm.$invalid || registerForm.$pristine || xhr || redirect">
    <span ng-hide="redirect">Register <i class="fa fa-repeat fa-spin" ng-show="xhr"></i></span>
    <span ng-show="redirect">Redirecting... <i class="fa fa-repeat fa-spin"></i></span>
  </button>
</form>
```

test/spec/login-service.js

```
describe('Provider: login-service', function() {
  'use strict';

  var loginService;

  beforeEach(module('loginService'));

  // Initialize the controller and a mock scope
  beforeEach(inject(function(_loginService_) {
    loginService = _loginService_;
  }));

  describe('loginHandler', function() {
    it('should create loginService.user with JSON from first argument', function() {
      var user = {
        foo: 'bar'
      };

      expect(loginService.user).toEqual({});
      loginService.loginHandler(user);
      expect(loginService.user).toEqual(user);
    });

    it('should extend loginService.user with JSON from subsequent calls', function() {
      var user1 = {
        foo: 'bar'
      };
      var user2 = {
        'baz': 'qux'
      };
      var combined = {
        'foo': 'bar',
        'baz': 'qux'
      };

      expect(loginService.user).toEqual({});
      loginService.loginHandler(user1);
      expect(loginService.user).toEqual(user1);
      loginService.loginHandler(user2);
      expect(loginService.user).toEqual(combined);

    });

    it('should set the user as logged in when called', function() {
      //TODO this is not secure!
      var user = {
        foo: 'bar'
      };

      expect(loginService.isLogged).toBeFalsy();
      loginService.loginHandler(user);
      expect(loginService.isLogged).toBeTruthy();

    });

    it('should set the user role when called', function() {
      var user = {
        userRole: userRoles.admin
      };

      expect(loginService.userRole).toEqual(userRoles.public);
      loginService.loginHandler(user);
      expect(loginService.userRole).toBe(user.userRole);

    });

    it('should set a token', function() {
      var user = {
        token: 'supersecret'
      };
```

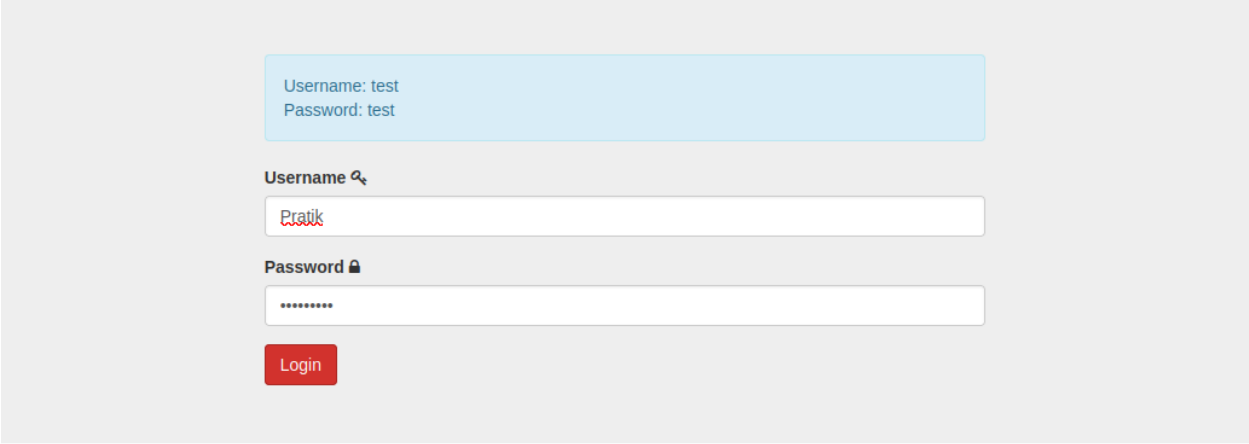
```
      expect(localStorage.getItem('userToken')).toEqual(null);
      loginService.loginHandler(user);
      expect(localStorage.getItem('userToken')).toEqual(user.token);

    });
  });
});
```

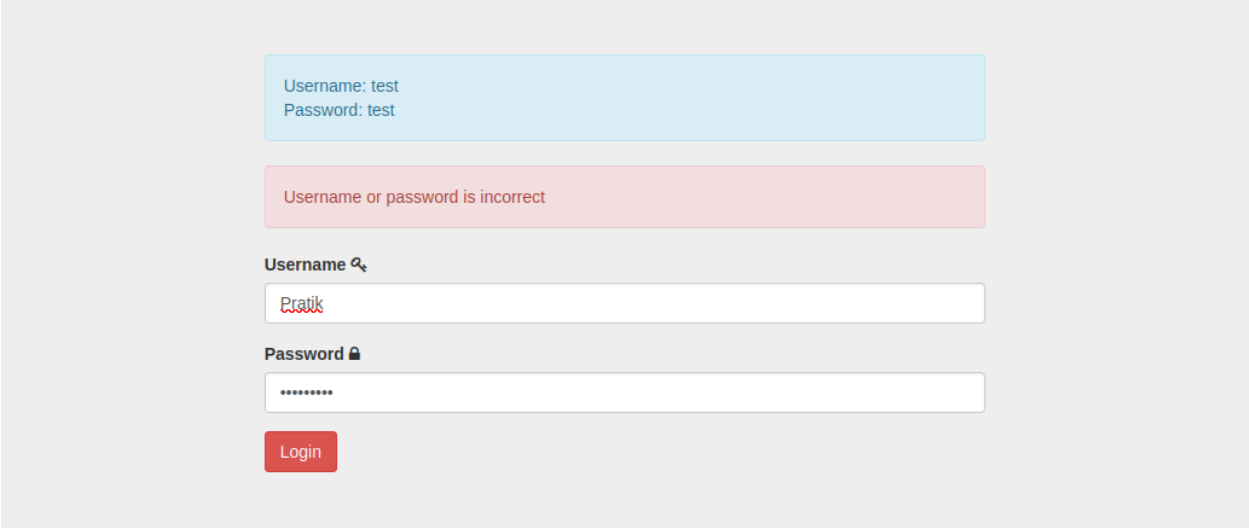
modules/authentication/views/login.html

```
<div class="alert alert-info">
    Username: test<br />
    Password: test
</div>
<div ng-show="error" class="alert alert-danger">{{error}}</div>
<form name="form" ng-submit="login()" role="form">
    <div class="form-group">
        <label for="username">Username</label>
        <i class="fa fa-key"></i>
        <input type="text" name="username" id="username" class="form-control" ng-model="username" required />
        <span ng-show="form.username.$dirty && form.username.$error.required" class="help-block">Username is required</spa
    </div>
    <div class="form-group">
        <label for="password">Password</label>
        <i class="fa fa-lock"></i>
        <input type="password" name="password" id="password" class="form-control" ng-model="password" required />
        <span ng-show="form.password.$dirty && form.password.$error.required" class="help-block">Password is required</spa
    </div>
    <div class="form-actions">
        <button type="submit" ng-disabled="form.$invalid || dataLoading" class="btn btn-danger">Login</button>
        <img ng-if="dataLoading" src="
/hpDcmVhdGVkIHdpdGggYWpheGxvYWQuaW5mbwAh+QQJCgAAACwAAAAAEAAQAAADMwi63P4wyklrE2MIOggZnAdOmGYJRbExwroUmcG2LmDEwnHQLVsYOd2mBzk
/IpHI5TAAAIfkECQoAAAAsAAAAABAAEAAAAzIIunInK0rnZBTwGPNMgQwmdsNgXGJUlIWEuR5oWUIpz8pAEAMe6TwfwyYsGo/IpFKSAAAh+QQJCgAAACwAAAAE
    </div>
</form>
```

modules/home/views/home.html

```
<h1>Home</h1>
<p>You're logged in!!</p>
<p><a href="#/login">Logout</a></a></p>
```


Username: test
Password: test
Username
Pratik
Password
Login


Username: test
Password: test
Username or password is incorrect
Username
Pratik
Password
Login

Home

You're logged in!!

Logout