# Movie Search Application Instruction Set

## Introduction:

It contains feature to login and move to dashboard page which provides facility to search movie in given search box that generates corresponding results. Use of directives is done for search box, movie details and rating elements

## Technologies Used:

AngularJS, Bootstrap, HTML5, CSS, nodeJS.
**Plugins/Libraries:** Toastr Module, Error Handler, Decorator etc.
**Stack:** Yeoman, Grunt, Bower, Karma, npm (node modules) etc.
**Movie API:** https://api.themoviedb.org/3/search/movie?api_key=<api_key>&language=en-US&query=<movie_name>

## Folder Structure:

1. Created using Yeoman, which is the scaffolding tool that helps you kick start new projects with best practices and tools ( http://yeoman.io/ ).
2. Project structure created by Yeoman is a bit complex and according to me doesn't suite big websites with lots of reusable components like directives, filters and services. To accomplish the requirement of creating a generic folder structure which is suitable for small as well as large websites I have tweaked the yeoman structure to an extent so that it suffices all the basic needs of an Angular website.
3. I have moved the app folder of Yeoman inside the public folder.
4. Also if you notice the app folder of yeoman is quite jumbled up as it contains everything right from htmls to images and all other assets.
5. To avoid this, I have created a separate assets folder besides app folder so that the assets like images, css and libraries don't mix up with the actual code that we write.
6. Within the app folder there are folders created based on the modules present in the app. Since we have login and home modules I have created separate folders with the same name along with other supportive modules in the app folder.
7. Created separate directives for search box, movie details and rating elements (for star ratings of movie) and stored in shared folder
8. There is an utils folder which contains the cross-cutting-concerns (common components) of the site.
9. Individual module folder contains all the module specific code files like controllers, services, filters and directives.
10. If something is to be used at multiple places it will go in the utils folder.
11. Along with development structure, I have created few test cases and facility to minify the build (That you can use using grunt)

## Use Case:

1. I have created 2 pages – **Login and Home page.**
   - User is by default redirected to the **login page**. On entering valid credentials user will be redirected to the home page. I have also involved **nodeJS server** having service to verify login details. User is validated against this service and also Angular validations are applied.
   - **Home page** will have movie search feature that displays search box and Find button.

   **Note:** nodeJS service is only written for login

2. Whenever user type some text to search as movie and hit "Find" button or "press Enter". There is service call which hits **Movie API** with query="text_to_search" and gives the result.

3. Created directives as:
   - **<search-input>:** It contains one textbox that has model "query", keypress event especially during enter key press and button "Find" which have click event. Both event handlers have call to **Movie API** through movie search service.

   - **<movie-details>:** For creating view for individual movie that is generated from list of movies which are received from **Movie API** during search.

   - **<div star-rating>:** For creating star rating view for individual movie based on popularity rating value. This directive is included within template of <movie-details>

4. CSS classes for Angular validations are used to display form errors
5. Some test cases are created for login and home pages using Karma.
6. Grunt is there to run server, run test cases, minify and build the app. Refer to **Gruntfile.js**
7. Bower is there to install dependencies through **bower.json.**

## Components:

### Service Layer: AngularUIFramework\public\app\utils\httpErrorHandler.service.js

1. I have created an Angular interceptor to handle services in the application. Because of the use of this interceptor all the services hit in the sites using $http provider gets redirected through this file.
2. All the controls are included in the utils folder.
3. We have functions to handle requests, response and errors.
4. Please note that the generic error handler is injected in this file so that errors are thrown from a common file.
5. You can always extend the functionality of this file to handle the services being hit through the application.

### Error Handler: AngularUIFramework\public\app\utils\errorHandler.dacorator.js

1. I have used AngularJS decorator here. Decorators allow us to separate out cross-cutting concerns and allow services to preserve SRP without worrying about "infrastructure" code.
2. A good use case of $provide.decorator is when you need to do minor "tweak" on some third-party/upstream service, on which your module depends, while leaving the service intact (because you are not the owner/maintainer of the service).
3. Error handler accepts the errors from any controller of the application and throws it to the UI.
4. Here you can even manage how the errors should be displayed to the user.

### Session Management: AngularUIFramework\public\app\utils\storageUtil.service.js

1. This is a generic session manager.
2. It includes just the setters and getters for variables or objects to be stored in the session.
3. It is been used to store the user details after login.

## To Run Commands: Open command prompt under **movieApp** folder

**Note:** Ignore the '$' while running command on windows

## Installing the application:

1. **Install node** from (https://nodejs.org/en/download/)
2. Run the following command to load all the module dependencies for app.
   **$ npm install**
3. Install the grunt cli and bower if not installed on the dev machine **$ npm install -g grunt-cli bower**
4. Install the bower dependencies using **$ bower install**

## Executing the App Commands:

1. **$ grunt serve**

   This loads the app dependencies (bower components) and runs the app on localhost server (http://localhost:9000). Lastly it watches for the file changes of app folder and reloads the files on server automatically.

2. **$ node server / $ npm start**

   It will start the node server which is ready to accept the API hits, it provides login service to verify the credentials to log into application.
   Server will run at port 8001.

3. Login Page at localhost:9000:
   Username: test@test.com
   Password: test123

## Other Commands:

1. **$ grunt test**

   It runs the test cases of test folder. The test configuration loads from the **karma.conf.js** file. Note the files name which ends with ***.specs.js** is only executed for test cases.

2. **$ grunt build**

   This command minifies the complete app and outputs the result to **dist** folder. It minifies the vendor files (bower components, 3rd party libraries.) to **vendor.REV.js** and application's .js files (controllers, services) to **scripts.REV.js** file. Same way for css files also. This command also takes care of angular dependencies injections and html minification.