

As a part of coding challenge, I have automated asked yelp.com

<https://github.com/automationrockstar/project>

Above repository is public, so you can access it anytime. This involves

Task 1: Goes to yelp.com and select “Restaurants” in the drop-down box in Find. Clicks Search button and append Pizza to Restaurants to make the search text as “Restaurants Pizza”. Report total no. of Search results with no. of results in the current page. Parameterize any 2 of the filtering parameters (Neighborhoods, Distance, Price, Features, etc.). Applies the filter. Reports total no. of Search results with no. of results in the current page. Report the star rating of each of the results in the first result page. Clicks and expand the first result from the search results. Logs all critical information of the selected restaurant details, for the reporting purpose.

Task 2 (Bonus): Apart from UI, using Java and API supporting libraries such as Rest-Assured and Apache HTTPComponents, automated restful web-service scenario.

Toolstack: Java, Selenium/Webdriver, Maven, Apache HTTP, and TestNG


Used same repository to accomplish both automation systems, for an instance there is one pom.xml file, with all UI automation dependencies (i.e. Selenium) and API automation dependencies.

```

<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.5</version>
</dependency>
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpclient</artifactId>
  <version>4.5.2</version>
</dependency>
<dependency>
  <groupId>com.jayway.restassured</groupId>
  <artifactId>rest-assured</artifactId>
  <version>2.9.0</version>
</dependency>
<dependency>
  <groupId>org.yaml</groupId>
  <artifactId>snakeyaml</artifactId>
  <version>1.15</version>
</dependency>


```

As I mentioned earlier, there is just repository used to automate UI and API, since they both same wide common tool-stack as as Java, TestNG and maven. At src.test.java.com/automation, you see two different packages. Package “web/test” is dedicated for UI and “webservice/test” is dedicated for API.


[automationrockstar](#) / **project**

<> Code
🔔 Issues 0
🔗 Pull requests 0
📁 Projects 0
📖 Wiki

Branch: master ▾
project / src / test / java / com / automation /


 automationrockstar armaan's first commit

..

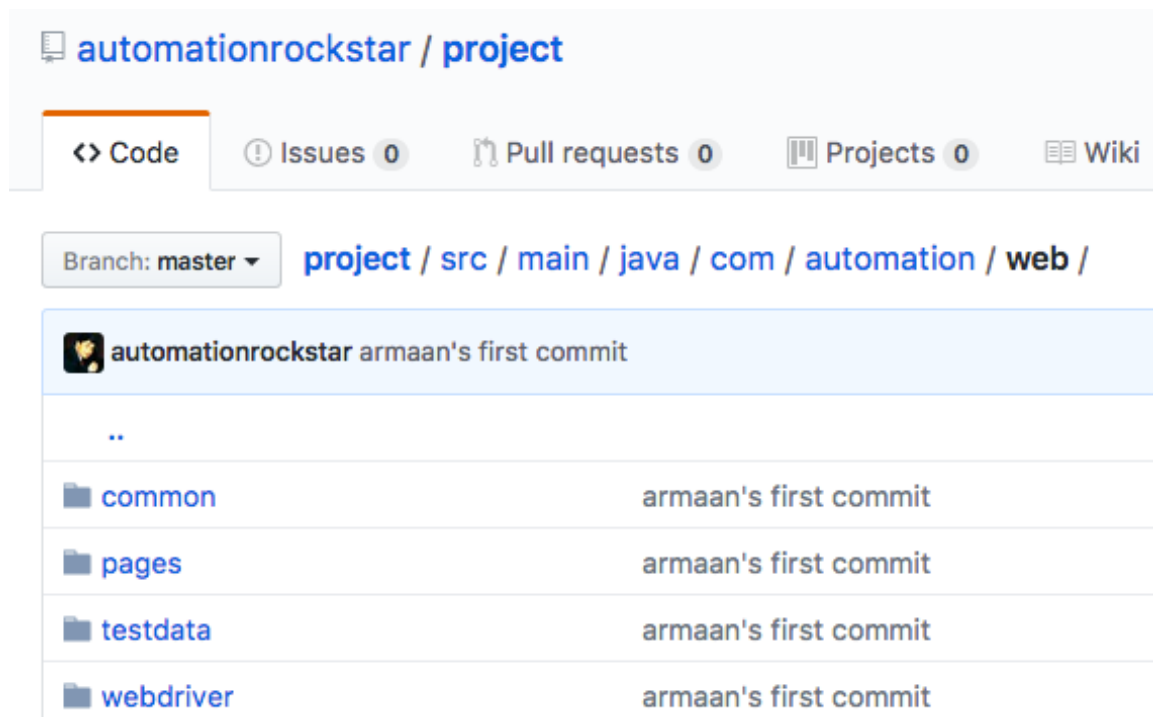
📁 web/test	armaan's first commit
📁 webservice/test	armaan's first commit

Every test-case is very independent, but extends super class called “BaseClass”, which is taking care of before and after classes. It takes care of loading up property, of browser. Based upon what browser user chose to execute test on, this basic information provided to the test-case through super class.

```
17  *   author: armaan (automationrockstar@gmail.com)   *
18  *   location: dublin, ca, usa                         *
19  *   date: june 1st, 2017                             *
20  *****/
21
22  public class BaseTest {
23
24      private static final Logger logger = Logger.getLogger(CustomerConfig.class);
25      protected WebDriver webDriver;
26      protected Browser browser;
27      protected String websiteUrl;
28
29      @BeforeClass(alwaysRun = true)
30      public void init() {
31          browser = new Browser();
32          websiteUrl=CustomerConfig.loadProperty("site.url");
33          browser.setName(CustomerConfig.loadProperty("browser.name"));
34          browser.setVersion(CustomerConfig.loadProperty("browser.version"));
35          browser.setPlatform(CustomerConfig.loadProperty("browser.platform"));
36          webDriver = WebDriverFactory.getInstance(browser);
37          webDriver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
38          webDriver.manage().timeouts().pageLoadTimeout(60, TimeUnit.SECONDS);
39          webDriver.manage().window().maximize();
40          logger.info("Test case Config setup complete.");
41      }
42
43      @AfterTest(alwaysRun = true)
44      public void tearDown() {
45          if (webDriver != null) {
46              webDriver.quit();
47          }
48      }
49  }
```

Followed standard guidelines, while automating such as:

1. Testcase is nothing but collection of methods. When there is change in any step or web-element, user don't need to go through painful process of modifying all the scripts
2. Keep data and web-element declaration aside of test-case or test-class
3. Script should able to run on various platforms, and any browser

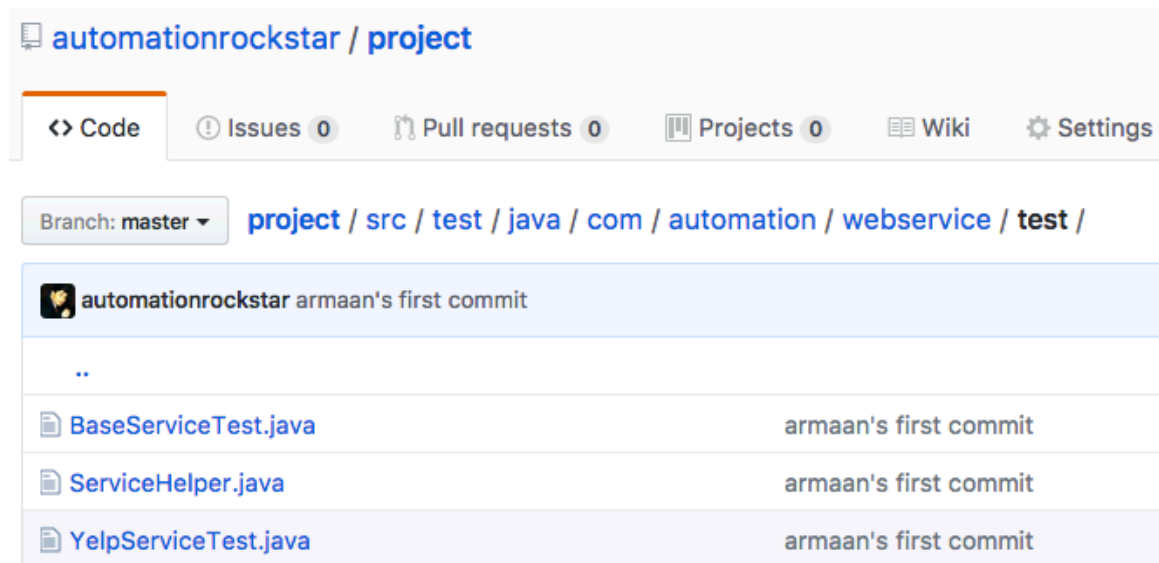


Folder “common” holds a class which is taking care of all the common utilities. Folder “pages” made for all the application pages, automation is dealing with. It holds all the methods, with page-factory model, where all web-elements defined with xpath, or id. Folder “testdata” holds testdata, but when we will run this automation in production, this is going to be based in external JSON file, which can be uploaded to Jenkins job. When user do

not upload any JSON testdata file, in that case it runs with default testdata and pick up from “testdata” folder. It also holds testconstants.java which provides constants to suite XML file (through which user can run group of test-cases).

Let’s talk about bonus task, which is my favorite API automation. I have created 3 different files which their own purpose. Baseservice holds init() method which deals with yelp.com authentications.

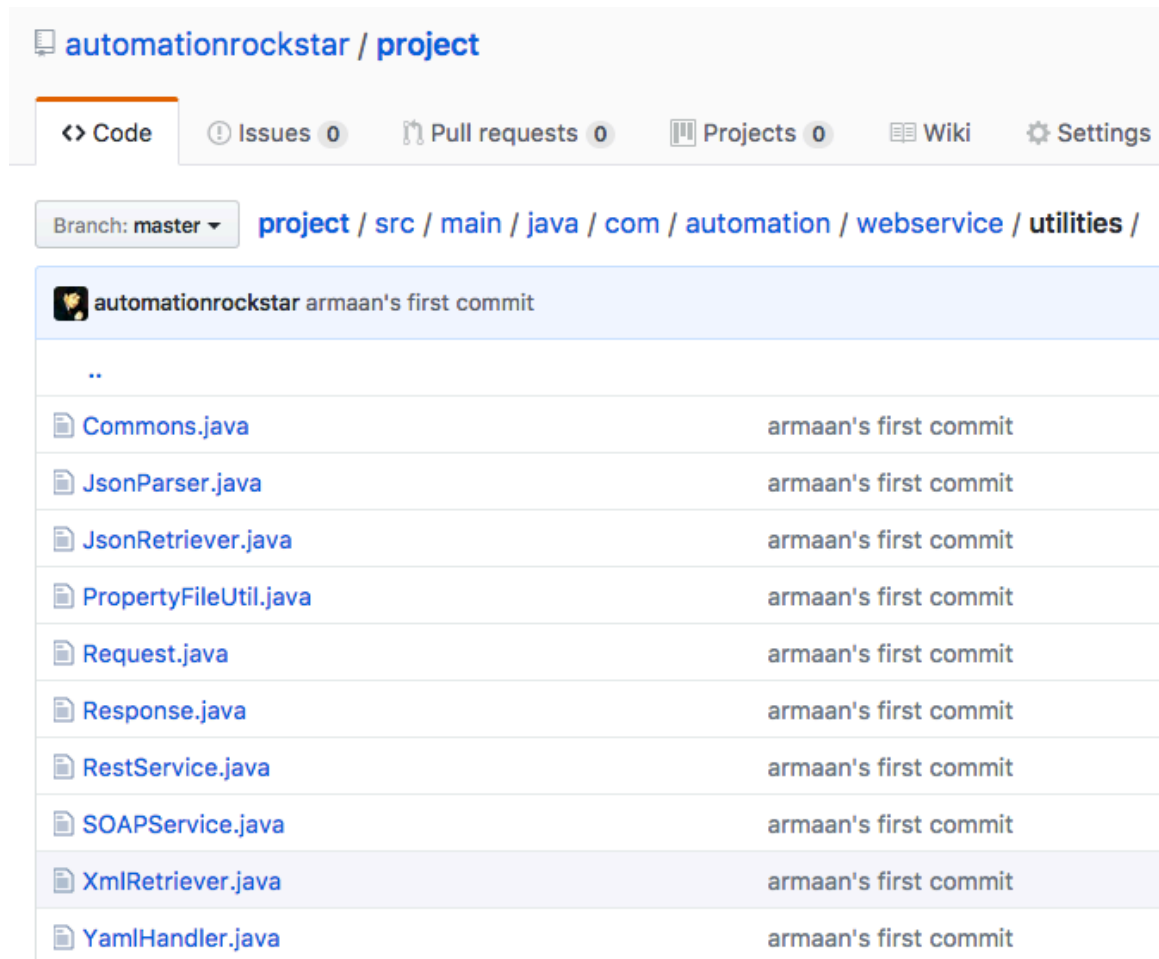
Like UI automation, YelpServiceTest extends superclass BaseServiceTest. We can reach an objective through this way, but when it comes to production code, we can have multiple libraries and wrappers which will make our API tests robust.



In src.main.java (not test), there you have same directory structure pattern followed as com.automation.webservice.base holds 3 different classes.

Service base provides logger, environment information to the API test classes.

Class comonbase plays key role of @SuppressWarnings for the classes.



automationrockstar / project	
<> Code ! Issues 0 🔗 Pull requests 0 📁 Projects 0 📖 Wiki ⚙ Settings	
Branch: master ▾ project / src / main / java / com / automation / webservice / utilities /	
automationrockstar armaan's first commit	
..	
📄 Commons.java	armaan's first commit
📄 JsonParser.java	armaan's first commit
📄 JsonRetriever.java	armaan's first commit
📄 PropertyFileUtil.java	armaan's first commit
📄 Request.java	armaan's first commit
📄 Response.java	armaan's first commit
📄 RestService.java	armaan's first commit
📄 SOAPService.java	armaan's first commit
📄 XmlRetriever.java	armaan's first commit
📄 YamlHandler.java	armaan's first commit

Utilities holds all API utilities, break down into small pieces to ease debugging. For instance, response holds methods which deals with http response such as readResponse and getSoapHeader etc.

Other important class Request.java holds all the http calls such as Get, Post, Delete etc. wrapped in the form of methods such as doCall, doGet, doPost etc.