*Read this file to understand and know how the framework is designed, how does it work, how was it built, what are the test scenarios, what are the limitations of the framework.*

***Author*** *- Seshadri Shome Chowdhuri*

**About:**

This project is a Selenium Automation Framework written in JAVA to test various functionalities on mapSYNQ.com.

**Architecture:**

Components used to build this framework –

1. Selenium WebDriver.

2. Java.

3. TestNG

4. Extent Report.

5. Apache POI API.

6. Maven.

It is a hybrid framework combining Page Object Model and Data Driven Approach. The code base is written in Java. Apache POI API is in used to read / write data from / to excel files. The project is built with Maven for automatic jar libraries injection by declaring the respective dependencies. Extent Report is used to generate the test execution report. Web Driver Event Listener is used to generate run time logs.

**Approach:**

mapSYNQ.com landing page is a single page that was in-scope for testing **(As Log in and Registration was not functioning).** As this framework follows Page Object Model, the liberty is taken to group alike functionalities inside landing page together and segregate them. Each of these group of tightly coupled functionalities are treated as a page. That has been done to get the flavor of POM. Otherwise only one webpage would have been in-scope and POM couldn’t justify it’s full potential with a single webpage.

**Design:**

In this framework the mapSYNQ.com webpage is divided in to four different groups / classes – Landing Page, Directions Tab, Live Tab, Personal Tab. Among these four groups - Landing Page, Directions Tab, Live Tab are implemented in this framework.

Below are the design of the modules used in framework -

***Folder - src/main/java***

* TestBase.java -> WebDriver Connection, Launching URL.
* config.properties -> Storing URL, Browser name (if not using TestNG) and other constants.
* ExtentReportListener.java -> Report Generator
* com.mapsynq.qa.pages.DirectionsTab.java -> Store all the web elements on Directions tab and methods to action on them.
* com.mapsynq.qa.pages.LandingPage.java -> Store all the web elements on Landing Page and methods to action on them.
* com.mapsynq.qa.pages.LiveTab.java -> Store all the web elements on Live Tab and methods to action on them.
* mapSYNQ\_Test\_Data.xlsx -> Test data excel file.
* com.mapsynq.qa.util.TestUtil.java -> Stores Test data location, static variables to verify results, Test data reader utility, Screenshot utility, explicitly defined wait utilities.
* com.mapsynq.qa.util.WebEventListener.java -> Web Event Listener class to listen to web events and generate run time logs.
* com.mapsynq.qa.util.Xls\_Reader.java -> (Not used in this framework) Excel Data reader / wrtier / manipulator using Apache POI API.

***Folder - src/test/java***

* com.mapsynq.qa.testcases.DirectionTabTest.java -> Test Cases to be performed on Directions Tab.
* com.mapsynq.qa.testcases.LandingPageTest.java -> Test Cases to be performed on Landing Page.
* com.mapsynq.qa.testcases.LiveTabTest.java -> Test Cases to be performed on Live Tab.

***Folder - src/main/resources***

* parallelbrowser.xml -> Runner xml to run the test cases in one go in different browsers parallelly. Currently Chrome, Firefox and Microsoft EDGE are included in it. **Though at present it isn’t functioning properly due to session errors.**
* singlebrowser.xml -> Runner xml to run the test cases in one go in a particular browsers. Currently Chrome is fitted into it, but if need be it can be any browser supported by selenium. **It’s working fine. So suggestion would be to run test cases with this runner xml.**
* screenshots -> Stores the screenshots if any exception occurs.

***Folder - test-output***

* Extent.html -> Generated report. After test cases are run, refresh project folder and copy it’s location and paste in a browser. It will display the detailed test run report.
* pom.xml -> As this is a maven project all the dependencies of the required APIs (TestNG, Selenium, Apache POI, Extent Report Generator etc.) are put into this. It automatically brings the necessary jars into project under folder name ‘Maven Dependencies’.

**Test Cases and methods:**

DirectionTabTest.Java -

1. Swap From To Point -> *swapFromToPointTest(String From\_Point, String To\_Point)*
2. Get Direction based on From To Point -> *getDirectionsTest(String From\_Point, String To\_Point)*
3. Clear Route -> *clearRouteTest(String From\_Point, String To\_Point)*
4. Leave At Time -> *leaveAtTime(String From\_Point, String To\_Point)*

LandingPageTest.Java -

1. Validate Landing Page Title -> *LandingPageTitleTest()*
2. Validate if mapSYNQ logo is displaying -> *landingPagelogoTest()*
3. Enter Search text in search box and select an option from suggested options displaying below -> *searchLocationFromSuggestedOptionTest(String Search\_Location)*
4. Enter Search text search box and click on search location button -> *searchLocationBySearchButtonTest(String Search\_Location)*
5. Select a Location on the returned search result -> *selectLocationTest(String Search\_Location)*
6. Select the dropdown of a Location on the returned search result and select To Here from the dropdown -> *selectToHereTest(String Search\_Location)*
7. Select the dropdown of a Location on the returned search result and select From Here from the dropdown -> *selectFromHereTest(String Search\_Location)*
8. Select a Location on the returned search result and select From Here from the Point of Inetrest popup -> *selectFromHerePOITest(String Search\_Location)*
9. Select a Location on the returned search result and select To Here from the Point of Inetrest popup -> *selectToHerePOITest(String Search\_Location)*
10. Close Point of Inetrest popup -> *closePOIPopupTest(String Search\_Location)*

LiveTabTest.java -

1. Verify if Camera Footage is Displaying for a particular camera link is clicked -> *getCameraImageTest()*
2. Close Camera footage popup -> *closeCameraPopupTest()*
3. Verify if Toll Chart Displaying for a particular Toll link is clicked -> *getTollsListTest()*
4. Close Toll popup -> *closeTollPopupTest()*
5. Verify Toll Chart against a selected vehicle group and Time frame -> *viewTollTest()*

**Limitations:**

1. Parallel Browser testing is not currently available as when test suite is run via parallelbrowser.xml it is encountering some session related issues.
2. Only one page was available to test (in-scope). So to use Page Object Model, the main landing page is divided into four parts and each of them are being treated as different objects.
3. Among four parts Landing Page, Directions Tab, Live Tab is implemented in this framework. Personal tab is not included in this framework.