Test Automation

Test Automation Framework

For DSS we have two types of automation

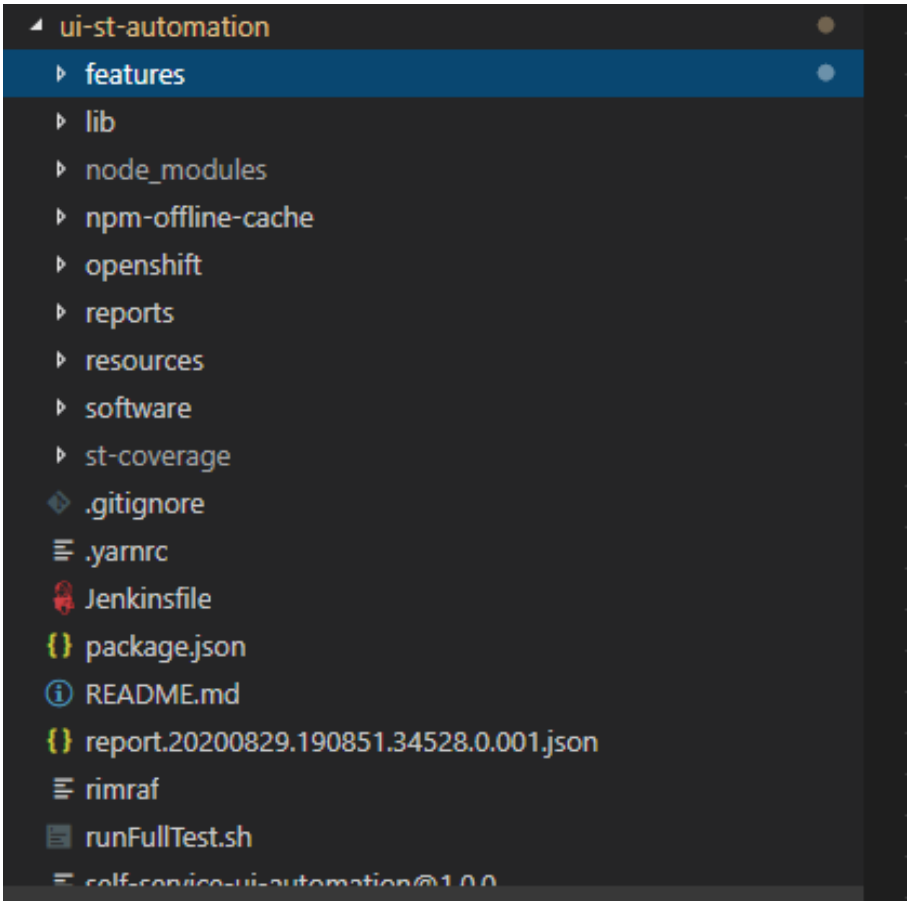
1-ST/MOCK UI - where the DSS UI runs on dev openshift/local where pretty much the UI functionalities are automated.
2- End to End - This actually runs on the TG non prod env (E2E/E2E2) and covers all the Srs the at are delivered.Verification includes upto whether the SRs are created in downstream (DCS) by checking the status in DSS.

The frame work is common for both the automation suites.

We follow the Behaviour driven Framework (BDD) using PUPPETEER is a chrome's team Testing tool, Cucumber JS is a test framework approach.

java script is used as the coding language for the underlying test functions.

Folder structure:



Tests are written in the human-readable Gherkin language using GIVEN WHEN AND THEN keywords and are stored in feature files that have the feature extension.

```
@Smoke1
  Scenario:Add Legal Entity to Digital Channel - Create and View service request
  Given I am an authorised user with permission to create an "Add Legal Entity to Digital Channel" service request
  When The user navigate to the "Add Legal Entity to Digital Channel" service request screen
  Then The application should display the "service request Header details" for "Add Legal Entity to Digital Channel"
    | Title       | Add Legal Entity to Digital Channel (AU)                                                    |
    | Description | Add legal entities to your ANZ Transactive – Global digital channel for Cash Management Service Requests. |
  Then The application shows the "cancel" button
  Then The application should display the "Instructions to download and complete the form" for "Add Legal Entity to Digital Channel"
```

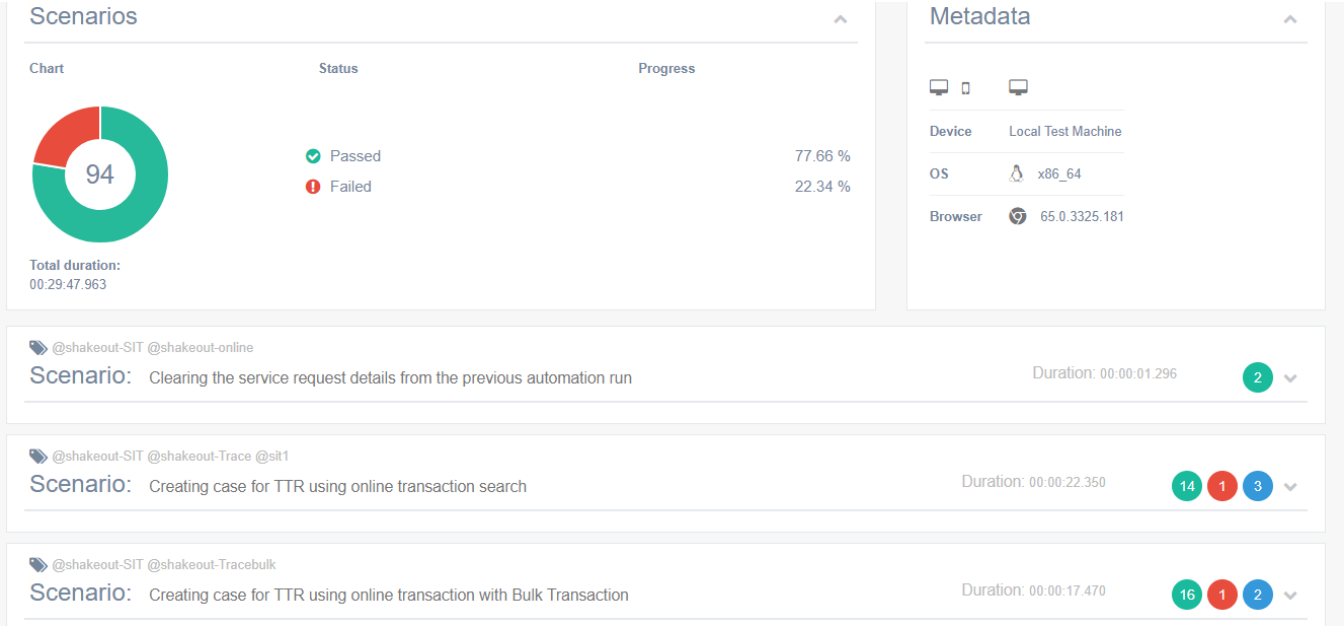
In turn each of the these test scenarios steps are mapped to a underlying js functions.

```
Then(/^The application should display the (.*) (?:for|on|in|under|the|) "(.*)"$/, async function (component, servicetyp, datatable) {
  await this.waitForLoading()
  const data = datatable.rowsHash();
  for (let i = 0; i < Object.keys(data).length; i++) {
    const value = data[Object.keys(data)[i]];
    const key = Object.keys(data)[i];
    await this.checkElementExist(pageobj[key]);
    await this.checkHTMLText(pageobj[key], value);
  }
  //await this.attachScreenShot();
});
```

The tests/functions are written in a common approach, so that they can be leveraged for other features.

Common page object model is used where are the page element details are stored and used across functions.

Multiple-cucumber-html-reporter is a reporting module for Cucumber to parse the JSON output to a detailed report of the pass and the failures.



Mock UI Automation

1.Test executed against on MOCK UI (eg:ST)

2.Test coverage includes page(s), validate elements, components, and UI related functions (including error cases), by leveraging mocks.

3.Re-usability of code using shared utilities/steps/regular expressions across other env.

4. Tests run every build/releases and commit pushed to a branch

5. Pass percentage is achieving 95% on every release.

End to End Automation

The purpose of the End to End Automation here is to eliminate the manual execution of Regression of basic flows at the end of each release and to check status of the environment by running it everyday.

The tests are executed on the non prod environment (E2E/E2E2).

Following flows are covered in the End to End Automation.

1. Creation of service request
2. Viewing created service Request
3. Audit history validations of a service request

4. Token Approval flow (Both single and dual)
5. Cancel request flow (Both Instant cancel and withdraw flow)

The scripts are written in a way that any new service request can be automated by inheriting the steps which was used earlier for other service requests based on the SPA used for the new service request.

Basic API automation framework is also integrated to the script which would fire a GET request to get OTP value which would be user for Token login.