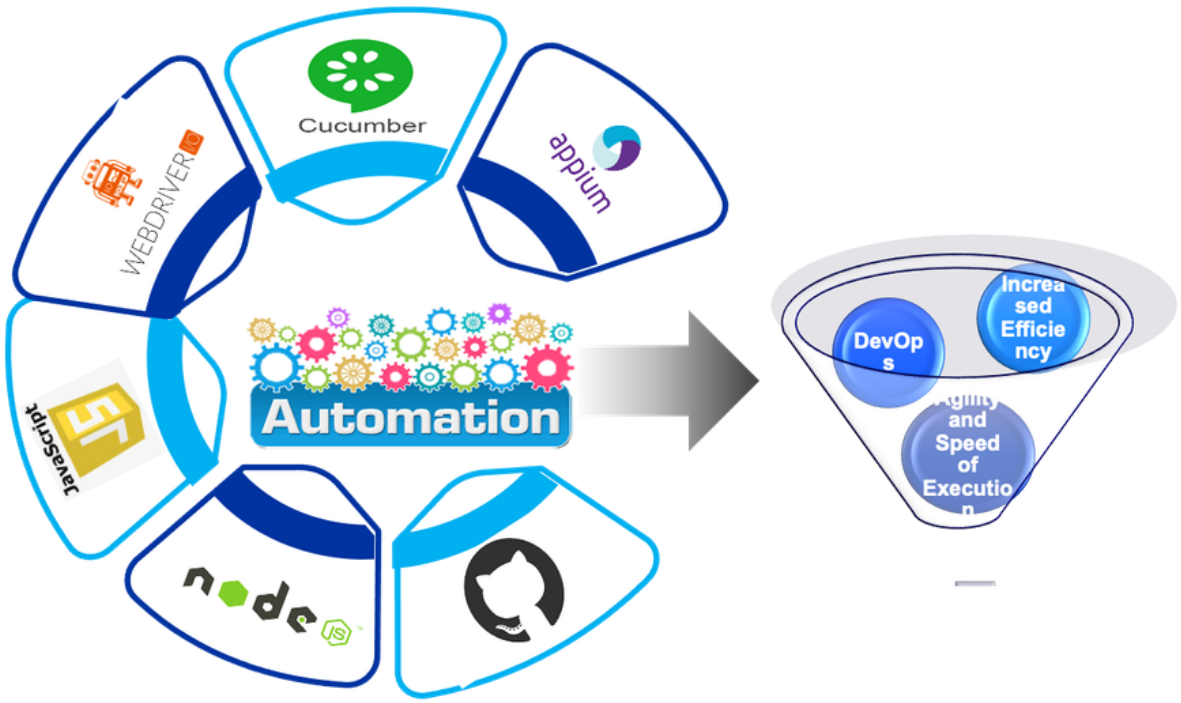
Mobile Automation - Architecture

Framework:

- **WebDriverIO** is a Next-Gen browser and mobile automation framework offering comprehensive functional assurance for Nodejs platform applications.
- **WebDriverIO Features**
  - **Web Testing**: Supports web based technologies including modern, digital, enterprise applications & Mobile apps for functional assurance for UI / API / RWD / Compatibility / visual validation needs.
  - **Tools Integration**: Supports additional validations like accessibility, performance using WinAppDriver, Axe, Lighthouse and supports Native desktop applications (e.g. written with Electron.js).
  - **Seamless integration**: Deploy tests in DevOps toolchains to leverage continuously.
  - **Extendable**:
    i. 📱 hybrid or native mobile applications running in an emulator/simulator or on a real device.
    ii. 💻 native desktop applications (e.g. written with Electron.js)

Framework and Design:



•True Open Source

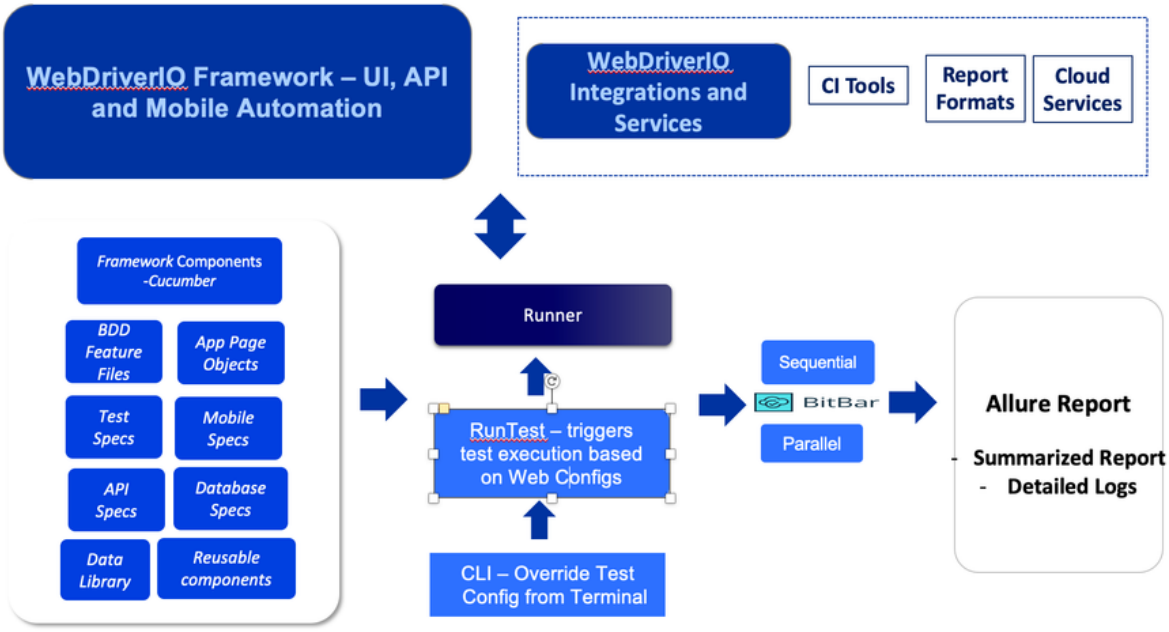
•Based on W3C Web Standards

•Committed large Community

Benefits:

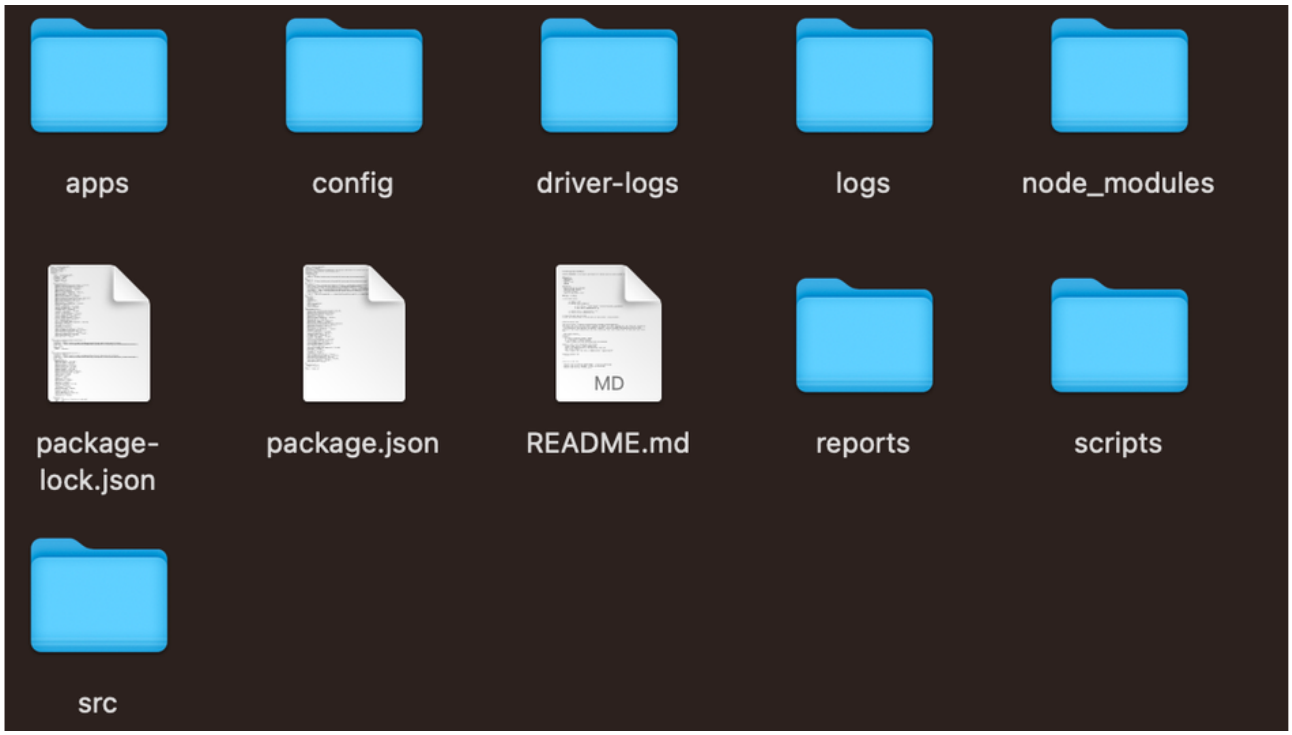
- Test execution time is reduced
- Framework is easy to use and maintain
- POM can be used for easy maintenance
- Reduction in manual test effort providing more time for exploratory testing
- Reduction in cycle time
- Reusable & scalable infrastructure components for other projects
- Bitbar supports open source object identification tool- Appium desktop versions lower than 1.20. **For iOS devices Mac machine is a pre-requisite**
- Appium Inspector can be used to identify the app locators
- Can be integrated with GitHub Actions to enable CICD pipeline

WebDriverIO Framework Architecture:



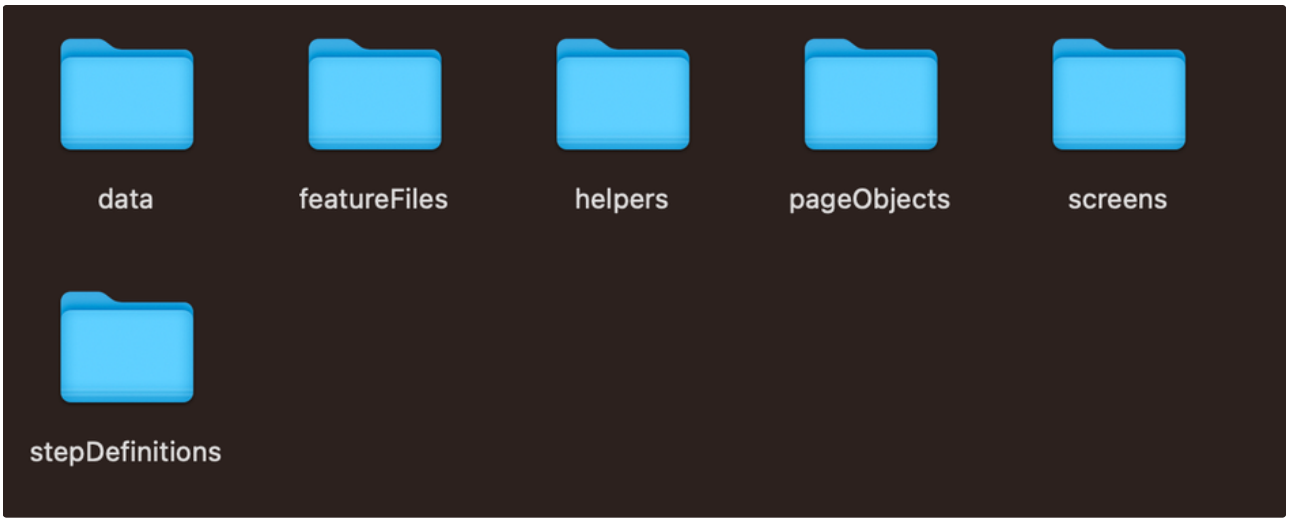
Framework Design:

Base Framework Folder Structure :


apps
config
driver-logs
logs
node_modules
package-lock.json
package.json
README.md
MD
reports
scripts
src

- Apps : contains apk files
- Config : configurations files related to each type of platform like android or ios
- Logs : log files
- Node_modules : all there quired node modules and dependencies
- Reports : Execution repots
- Package.json : contains list of components that need to be installed and npm install will refer this file to install
- README : Instructions on how to use this framework

Src folder structure:



- data : contains input data files
- featureFiles : Feature files translated from JIRA manual tests
- helpers : re-usable common functions/methods

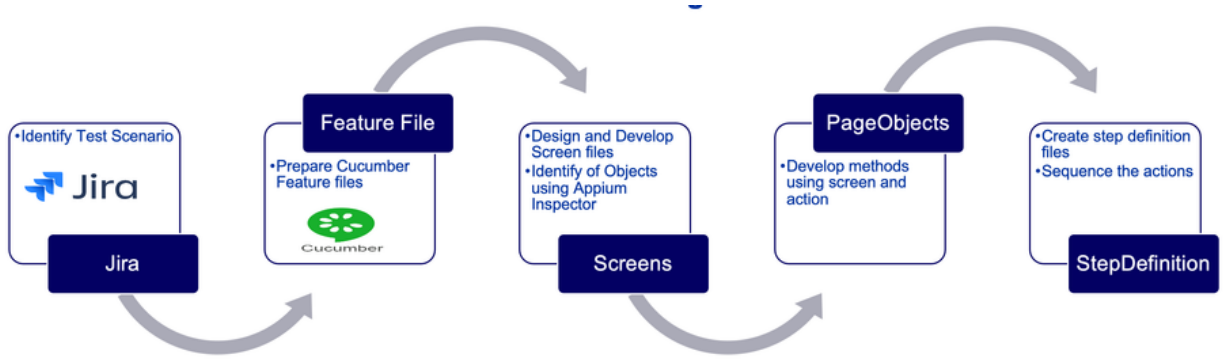
- Logs : log files
- pageObjects : pageObject files based on the application flow
- screens : application screens which will hold the object repository of the elements with locator’s
- stepDefinition : contains stepDefinition files

Tool Stack :

WebdriverIO	Open Source	**WebdriverIO** allows you to automate any mobile application written with modern web frameworks such as React, Angular, Polymeror Vue.js as well as native mobile applications for Android and iOS.
Cucumber	Open Source	**Cucumber is a testing library that supports Behavior-Driven Development (BDD). It** executes specifications written in plain language and produces reports indicating whether the software behaves according to the specification or not.
Node JS	Open Source	**Node.js** is an open-source, cross-platform, back-end JavaScript runtime environment that runs on a JavaScript Engine and executes JavaScript code outside a web browser, which was designed to build scalable network applications.
Appium	Open Source	**Appium** is an open-source tool for automating native and hybrid mobile applications on iOS and Android mobiles.

Automation script development:

Flow:



Jira

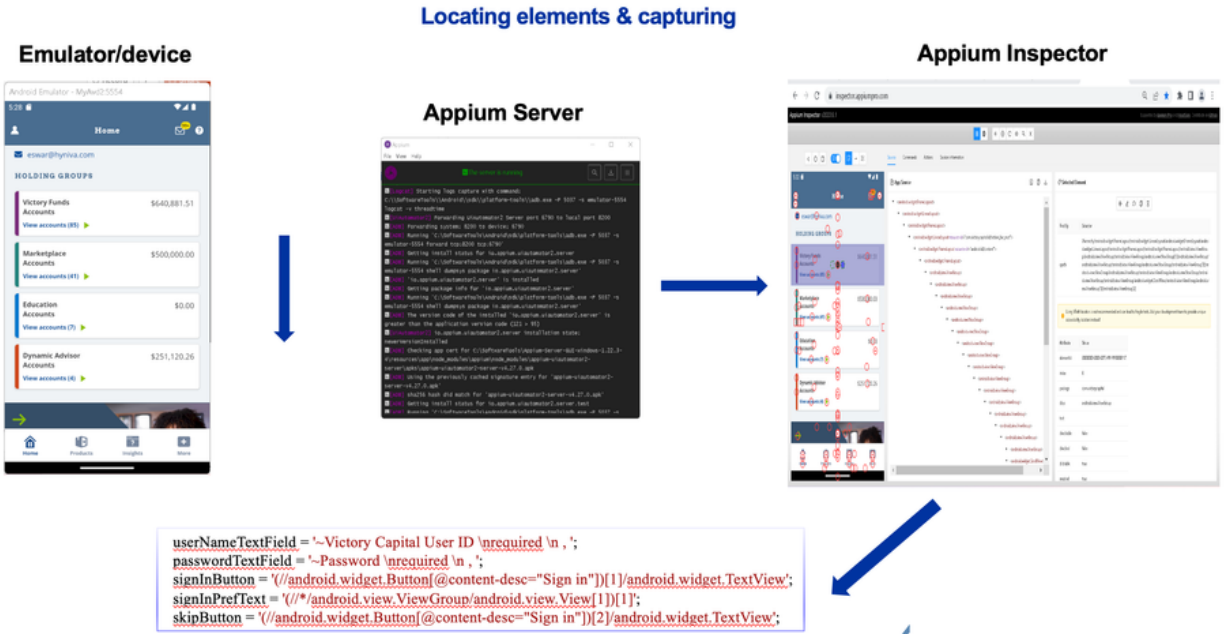
1. Click on theLogin with any brokerage-enabled user id, and the user lands on the dashboard.
2. Market Place card and select View/Open a new account.
3. *View accounts Option* will appear if there are existing FIS accounts.
4. *Open a new account option* will appear if there are no existing FIS accounts.
5. Click on the View Accounts option and the user is able to see the list of existing FIS accounts.
6. Click on any account you want to perform the functionality and click on Manage Services and Preferences card.


```
Feature: Market Place New Account Open

  @androidApp
  Scenario Outline: DFPS6074-As a user I can submit FIS NAO account
    Given I launch the app
    When I login with <username> and <password>
    Then I should land on VCM Dashboard
    Then account should get created susscessfully

    Examples:
      | username       | password     |
      | onlineid31     | Pass@123     |
```

Script Development:



Login.screen.js

```
class loginScreen {
  userNameTextField = '//XCUIElementTypeTextField[@r
  passwordTextField = '//XCUIElementTypeSecureTextFi
  signInButton = '~Sign in';
  signInPrefText = '~Sign in Preferences';
  skipButton = '~Skip';

  invalidCredError = '~Error,Please enter valid cred
}

module.exports = new loginScreen();
```

Login.page.js

```
class LoginPage extends BasePage {
  getObjectLocator () {
    platform = browser.capabilities.platformName.tol
    return require(`../screens/native/${platform}/lo
  }
  async login (username, password) {
    password = credentials[username];
    await this.sendText(this.getObjectLocator().user
    await this.setText(this.getObjectLocator().passv
    await this.click(this.getObjectLocator().signIn
    await this.pause(10);
```

```
12
```

```
12     await this.waitForElement(this.getObjectLocator(
13   }
14 }
15 module.exports = new LoginPage();
```

Step Definition:

Cucumber feature file

```
1  Feature: Login verification
2
3    @androidApp @iosApp @sanity @l
4    Scenario Outline: As a user, I
5      Given I launch the app
6      When I login with <username>
7      And I skip preferences
8      Then I land on VCM Dashboard
9
10     Examples:
11       | username   | password |
12       | onlineid31 | ******** |
```

Screen objects

```
1  class loginScreen {
2    userNameTextField = '//XCUIEle
3    passwordTextField = '//XCUIEle
4    signInButton = '~Sign in';
5    signInPrefText = '~Sign in Pre
6    skipButton = '~Skip';
7
8    invalidCredError = '~Error,Ple
9  }
10
11 module.exports = new loginScreen
```

Page Actions

```
1  class LoginPage extends BasePage
2    getObjectLocator () {
3      platform = browser.capabilit
4      return require(`../screens/r
5    }
6    async login (username, passwor
7      password = credentials[usern
8      await this.sendText(this.get
9      await this.setText(this.getO
10     await this.click(this.getObj
11     await this.pause(10);
12     await this.waitForElement(th
13   }
14 }
15 module.exports = new LoginPage()
```

- The feature file contains all the scenario with the required steps and the filename needs to be in ““feturename“.feature” format.
- All the required screen objects for a step or screen will be captured in “xxxx.screen.js“ file.
- With captured screen objects for the give page all action related to the step will be written in the “xxx.page.js” file.

Example step Definition

Given.js

```
1  const {Given} = require('@wdio/c
2  const LoginPage = require('./../
3  const SignInPage = require('./../
4  const pages = {
5   login: LoginPage,
6   signIn: SignInPage
7  };
8  Given(/^I am on the (\w+) page$/
9   await pages[page].open();
10 });
11 Given(/^I launch the app$/, asyn
12 await SignInPage.launchApp();
13 await SignInPage.signIn();
14 });
15
```

When.js

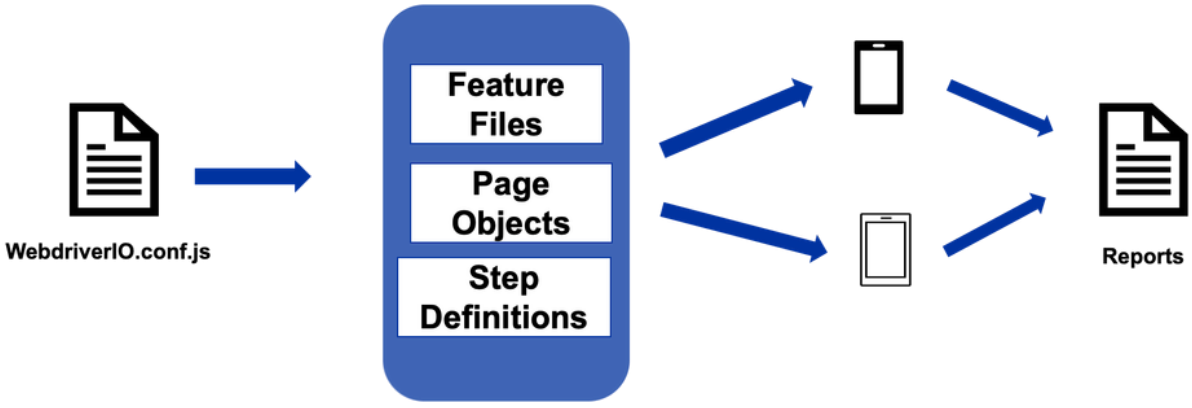
```
1  const {When} = require('@wdio/cuc
2  const expectChai = require('chai
3  const expectWDIO = require('@wdio
4  const LoginPage = require('./../p
5  When(/^I login with (\w+) and (.+
6  await LoginPage.login(username, p
7  await LoginPage.skipSignInPref();
8  });
```

Then.js

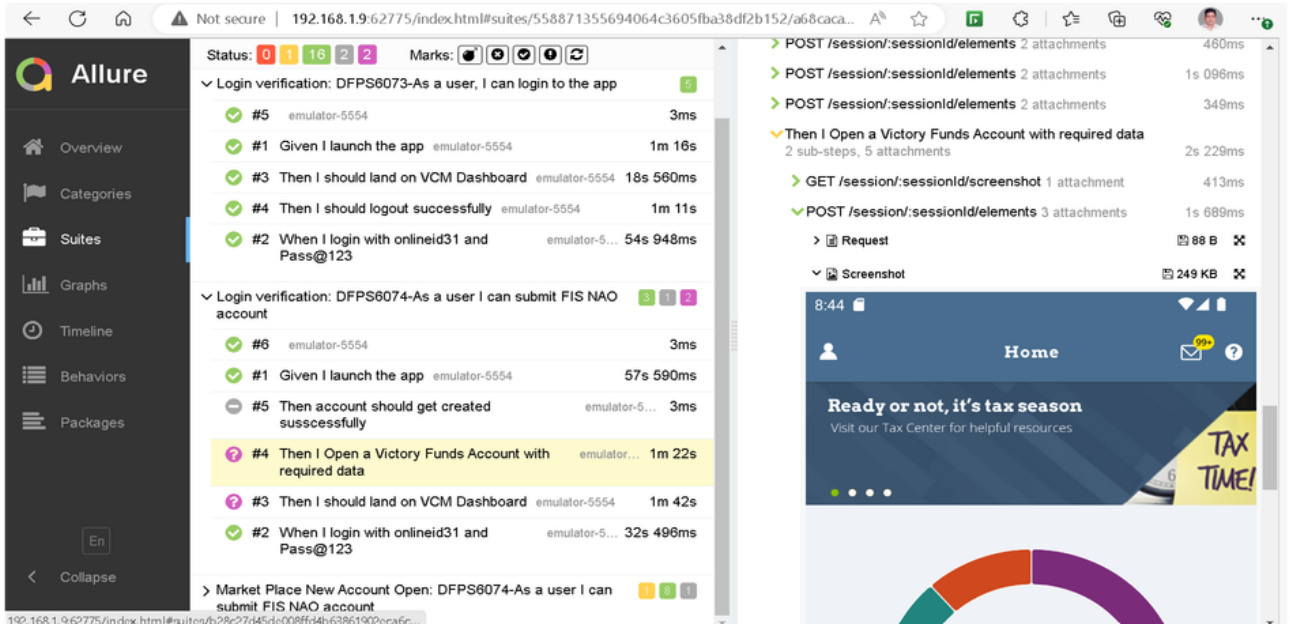
```
1  const { Then } = require("@wdio/
2  const DashBoardPage = require("
3  const signOutPage = require("../
4  Then(/^I should land on VCM Dash
5  await DashBoardPage.verifyDashBo
6  });
7  Then(/^I should logout successfu
8   await signOutPage.signOut();
9  });
10
```

Automation Execution Results:

Script Execution:



Execution Results:



Jira and Xray integration for e2e execution through github actions:

- Specify Cucumber tests using natural language, in Jira.
- Export Cucumber features from Jira to the CI environment(github actions), using the Jira/Xray REST API.
- Implement tests in code and commit them to the source code versioning system.
- Execute tests in the CI environment.
- Report results to Xray, using the Jira/Xray REST API.