**Contents:**

1. Tech Stack Used
2. Framework Choice
   1. Page factory model (over page object model)
   2. Extent reporting and Logging
3. Troubleshooting and maintenance

**Tech stack used:**

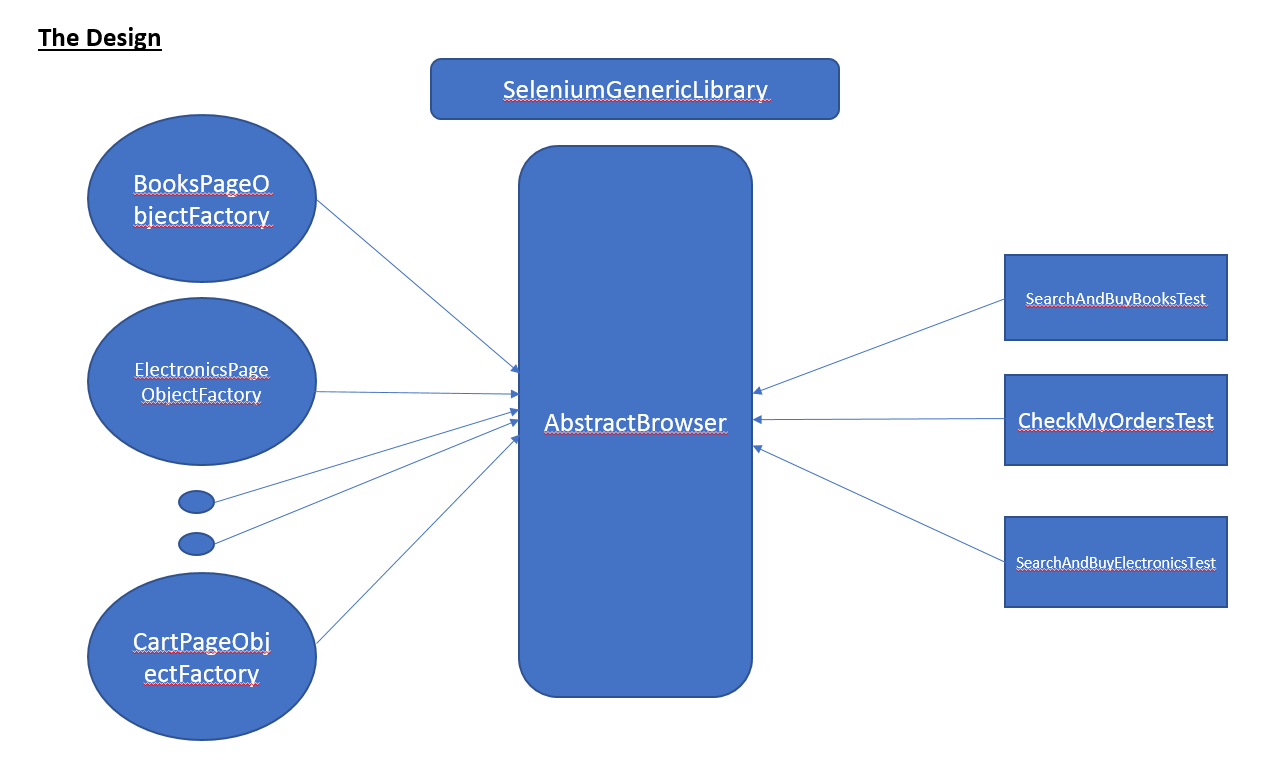
1. Maven – POM based architecture enables the ease of adding dependencies to the project, also sharing the project is easy, when imported as a maven project the dependencies are added to the project automatically which gives flexibility to the users. Ease of use with command line/terminal. Platform independent.
2. WebDriver – A proven framework for UI automation.
3. TestNG – TestNG goes very well with Selenium Java WebDriver. Provides the convenience to run based on groups.
4. Extent Reporting – The new era of reporting, enables configuration based on our needs. Renders a very structured and useful dashboard.
5. Log4j – the logging functionality is easy to plug and use within Java classes.

**Framework choice*:***

***Page Factory model (over page object model):***

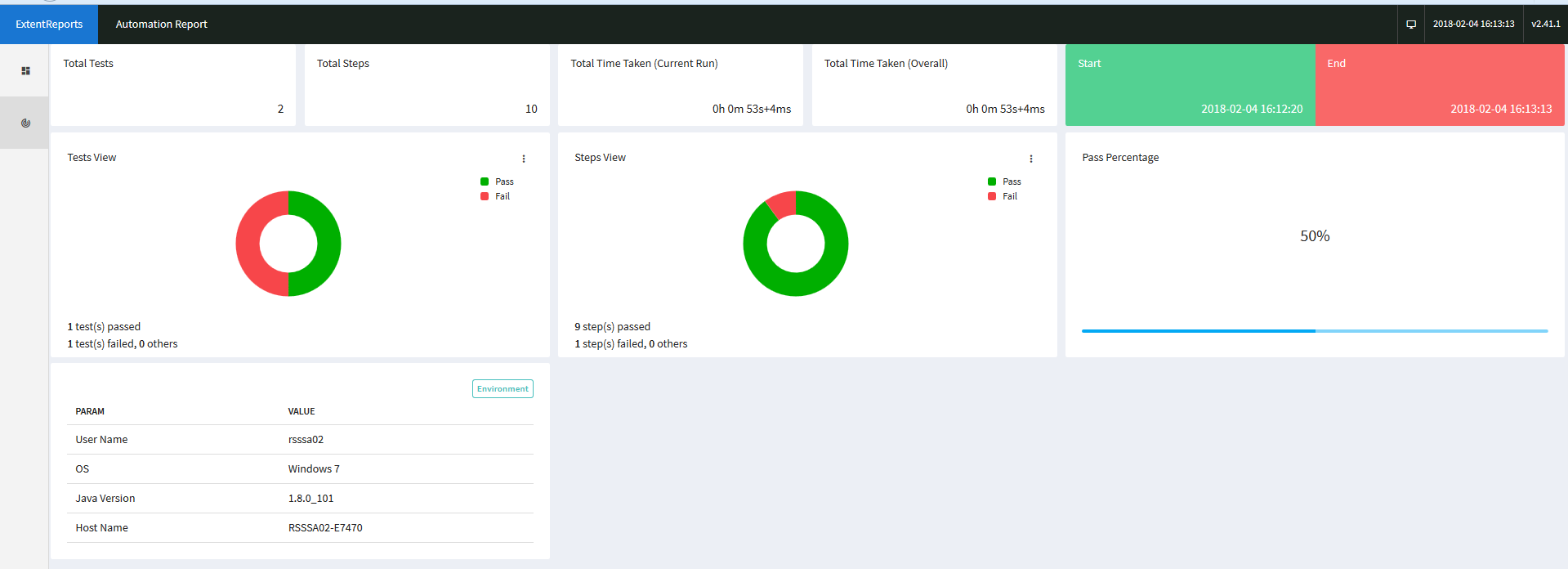
It provides the ease of maintaining Object Repository for UI elements in the page. Page Object Patten maintains operations and flows in the UI should be separated from verification. This concept makes our code cleaner and easy to understand. Code becomes less and optimized because of the reusable page methods in the POM classes.

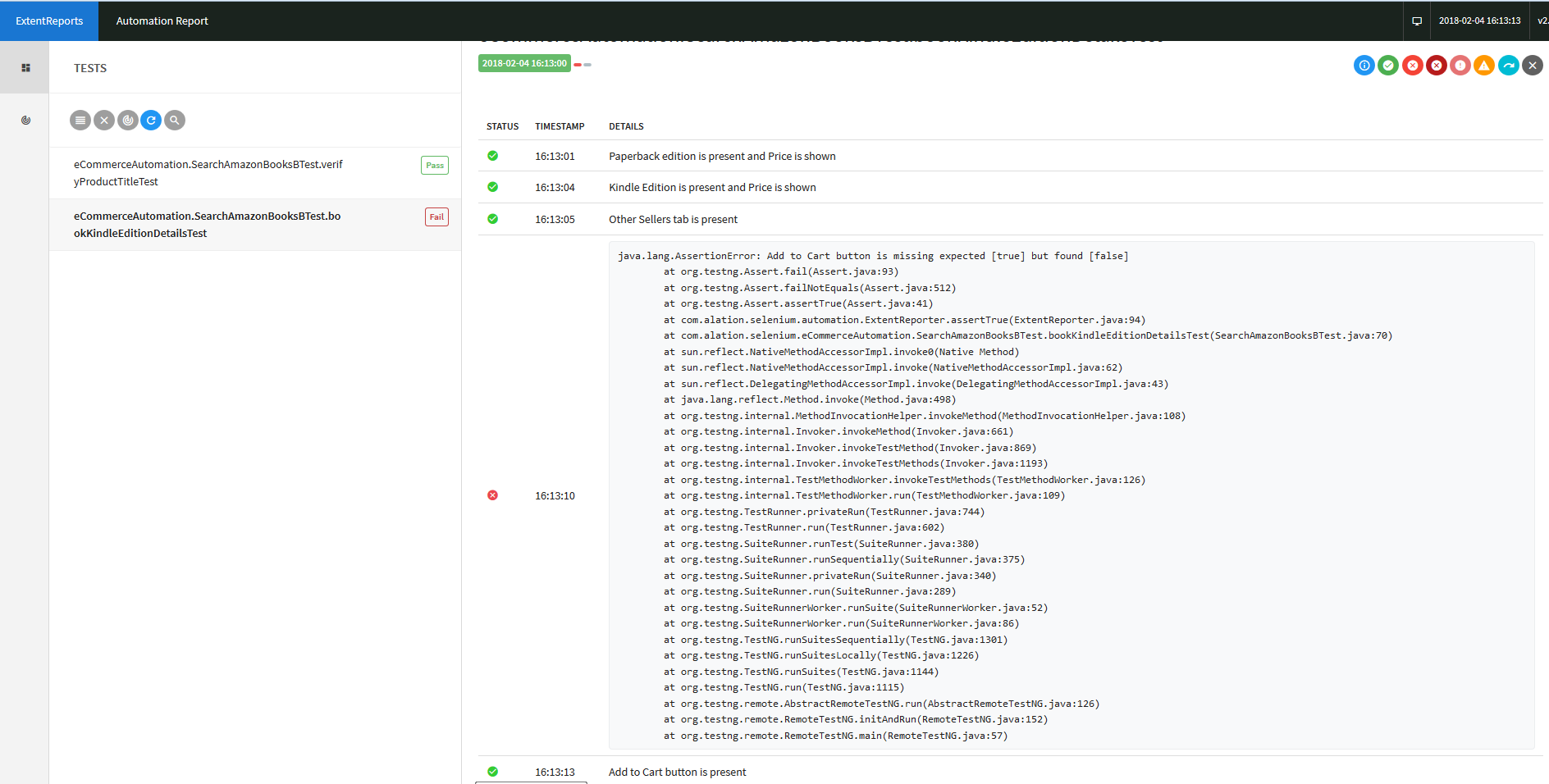
Below is the design of the test framework (reporting excluded). When I say page object, as the name refers this java class is created for every page and the respective object will be created in AbstractBrowser class. Whenever we write any test the user has to just extend this class and every other PageObjectFactory class will be available to the user. In this way we can avoid any removal or addition of new page object factory classes. Any changes at any point in time will be easy to spot and fix. Any new elements to be added or modified will be made easy.



**Extent Reporting and Logging:**

Renders a very nice dashboard, which shows granular level details in case user wishes to drill down into the results. On failure shows the full stack trace





Framework uses Log4j and the logging mechanism is made available at the wrapper classes so that log can show more details.

*INFO SeleniumGenericLibrary:143 - waiting to click on element.. - [[FirefoxDriver: firefox on XP (0404cf02-c35c-4bac-af9a-92eef9725c2e)] -> id: averageCustomerReviews]*

*INFO SeleniumGenericLibrary:143 - waiting to click on element.. - [[FirefoxDriver: firefox on XP (0404cf02-c35c-4bac-af9a-92eef9725c2e)] -> id: acrCustomerReviewLink]*

*INFO SearchAmazonBooksBTest:48 - Total number of reviews displayed - 13 customer reviews*

Possible enhancement is to show only the locator instead of printing the webelement. This enables users to identify and locate the problem easily lets you locate the problem by seeing the logger.

**Troubleshooting and maintenance**

Framework employs extentreporting assertions. The reason is to enable the ease of troubleshooting, the advantage is that the stack trace are displayed for each assert steps written inside the tests. This is achieved by the extentReport dashboard which drills down the test methods and show every assertion as steps.

Maintaining the page objects. The framework will be written in a way that every page has its own page object classes. For instance. Add to cart can be part of CartPageObjectFactory. Editions of book (kindle or paperback) can be part of BookPageObjectFactory. Actions specific to pages can be added to these specific classes. From tests the user has to invoke the respective object and the webelement or action is made available to the user.

Doing this way enables the automation team to maintain the page objects. If they change tomorrow. Removal or updating is made easy.