

**Stacked Invest Automation Framework Overview**

***—--—--—--—-- Specifications —--—--—--—--***

**Integrated Development Environment (IDE):** IntelliJ IDEA CE

(IntelliJ is preferred. VScode, Eclipse, and etc. can be used)

**Languages:** Java (JDK 18)

**Integrated Tools:** Cucumber, Maven, Selenium

(Tools to be added: Appium for mobile testing, Rest Assured for API testing)

**Version Control Tools:** Git/GitHub

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

***--—--—--—-- Specifications Explained*** ***—--—--—--—***

**Cucumber:** A cucumber is a tool based on Behavior Driven Development (BDD) framework which is used to write acceptance tests for the web application. It allows automation of functional validation in easily readable and understandable format (like plain English) to Business Analysts, Developers, Testers, etc.

**Maven:** Maven is chiefly used for Java-based projects, helping to download dependencies, which refers to the libraries or JAR files. The tool helps get the right JAR files for each project as there may be different versions of separate packages.

**Selenium:** Selenium is a free (open-source) automated testing framework used to validate web applications across different browsers and platforms. You can use multiple programming languages like Java, C#, Python etc to create Selenium Test Scripts.

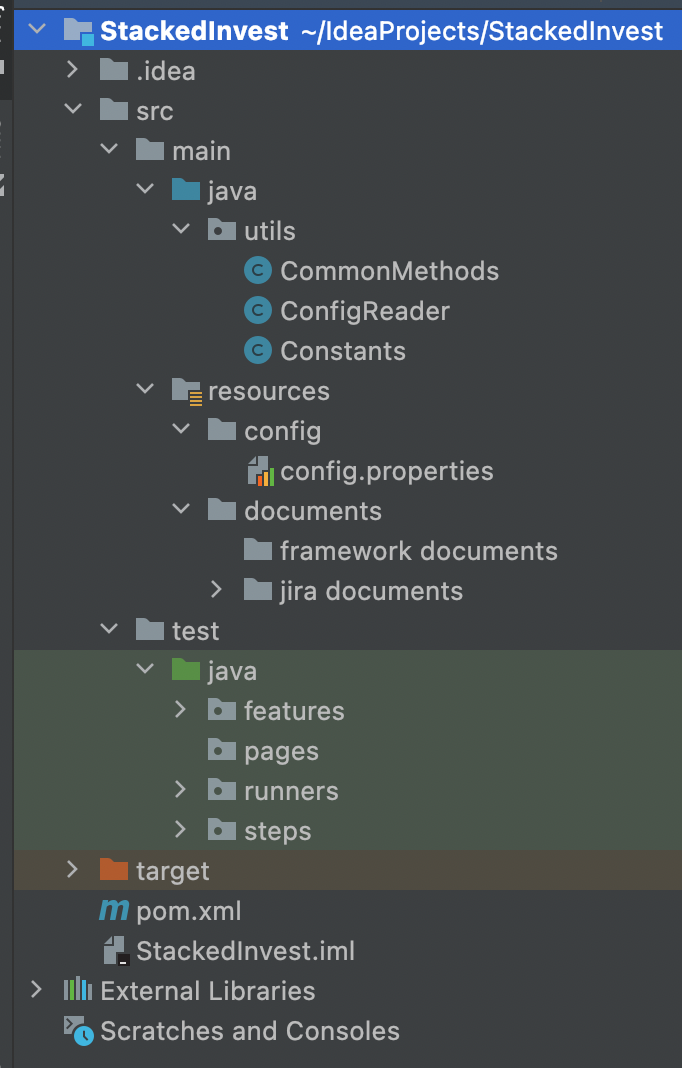
**Appium:** Appium is an open-source tool for automating native, mobile web, and hybrid applications on iOS mobile, Android mobile, and Windows desktop platforms.

**Rest Assured:** REST Assured is a Java library that provides a domain-specific language (DSL) for writing powerful, maintainable tests for RESTful APIs.

**Git/Github:** Git is a DevOps tool used for source code management. Git is used to track changes in the source code, enabling multiple developers to work together on non-linear development. GitHub is a web-based version-control and collaboration platform for software developers.

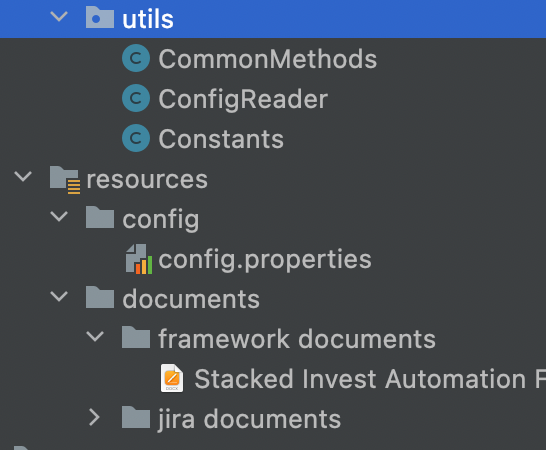
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

***--—--—--—-- Framework Structure Explained*** ***—--—--—--—***

Our framework follows a page object model. A page object model, also known as POM, is a design pattern in Selenium that creates an object repository for storing all web elements. It is useful in reducing code duplication and improves test case maintenance. In Page Object Model, consider each web page of an application as a class file. 

Our **src**, otherwise known as source folder, holds important directories and packages that contain most of the code for our automation framework.

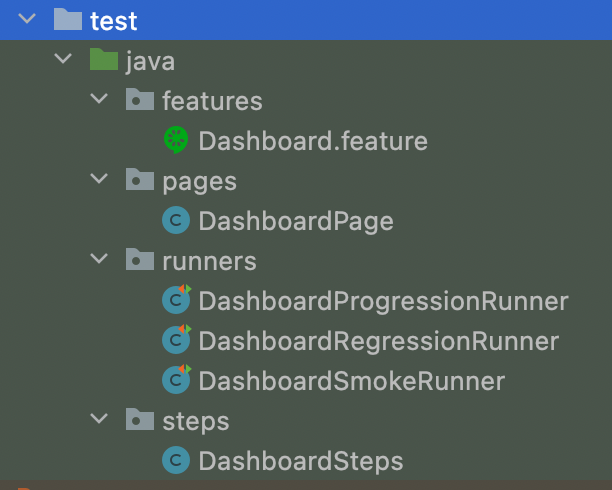
Within src, there is a main directory folder and a test directory folder**. Main** contains a folder named **java** that contains **utils** (utilities), which are general methods/code for the framework. For example, common methods is a class that contains methods of code that will be commonly used throughout our project. Constants is a class where we will store final constant variables. ConfigReader is a class that has methods to read our data from our config.properties file. Main also contains a **resources** folder that holds important data.



The **config** folder contains configuration data such as the name of the browser we are using or the url we will be accessing.

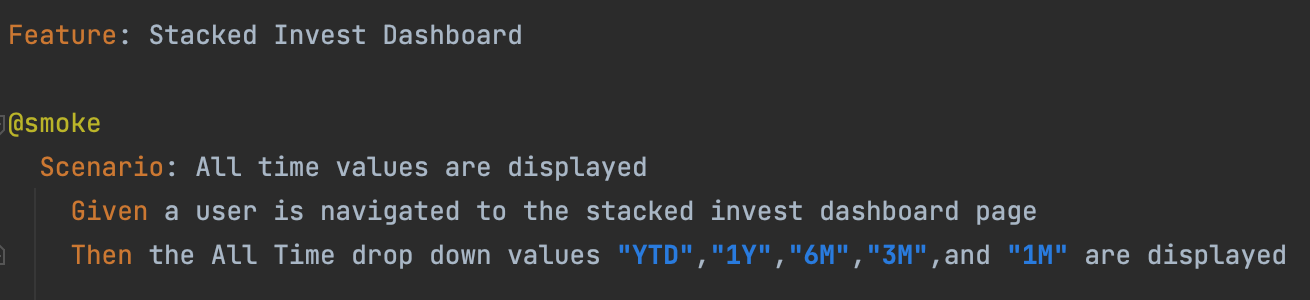


In the **test** folder, we have a java folder that contains organized packages that hold specific code related to different aspects of the application we are testing.



The **features** package is where we create, write, and maintain our “features” or test cases. Our test cases will be written in BDD Gherkin language.

This is where testers will be writing the actual test cases in the Gherkin language using Gherkin Keywords: ‘Background’, ‘Given’, ‘When’, ‘And’, ‘Or’, ‘But’, and ‘Then’ (Cucumber scenarios). You can have multiple scenarios in one feature file. Please have scenarios reviewed by the Automation Team before implementing. Every feature file needs to end with ‘.feature’. *Example:* login.feature



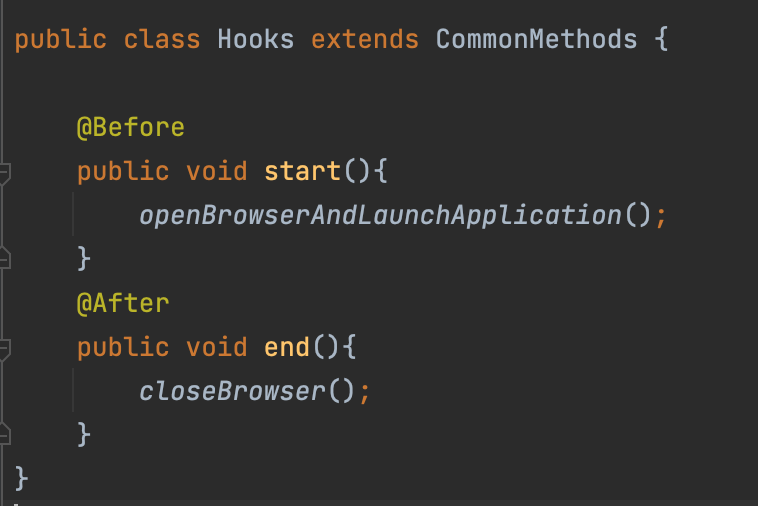
For more information on BDD Gherkin language and how to write it, please refer to this link:

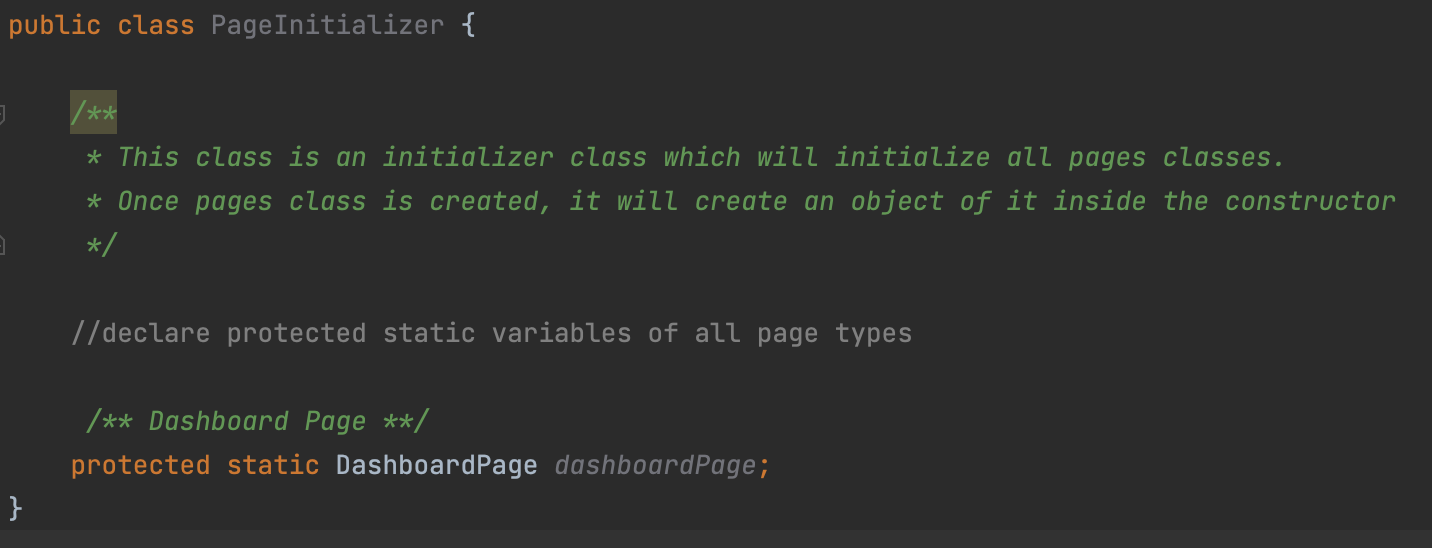
**https://cucumber.io/docs/gherkin/reference/**

The **pages** package contains page classes where we create, write, store and maintain our elements that pertain to specific pages of our application. Our pages classes extend our common methods class. We store page elements in our pagesclasses using selenium locators, mainly XPaths. In every page class tester will need to create a constructor to initialize the stored web elements using PageFactory class.

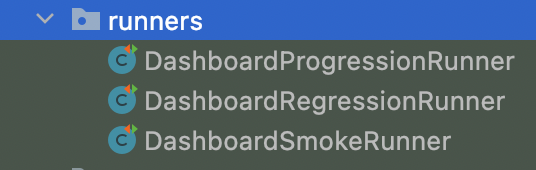


The **steps** package contains steps classes where we create, write, and maintain our methods/lines of code in Java and Selenium that execute our test cases. Our steps classes extend our common methods class. This is where the tester will add additional cucumber steps after using the runners to generate any unimplemented steps. Our steps package also contains our **Hooks class**, which contains the methods to set up and close down our browser to begin testing. We use cucumber annotations of @Before and @After in the Hooks class in order to implement methods that run before and after our tests. In addition, we have a Page Initializer class in our steps package, that creates objects of our pages for further framework organization.





The **runners** package contains runners classes where we create, write, and maintain the code that will run/execute our test cases. We use cucumber options to add specifications to our runner. The Runners class is a class that allows us to run certain test cases depending on Tags given to the feature files. This way we can group certain test cases that are being worked on, finished or have to run in certain sequences/order. We have different names for our runners that we tag such as progression, regression, and smoke.



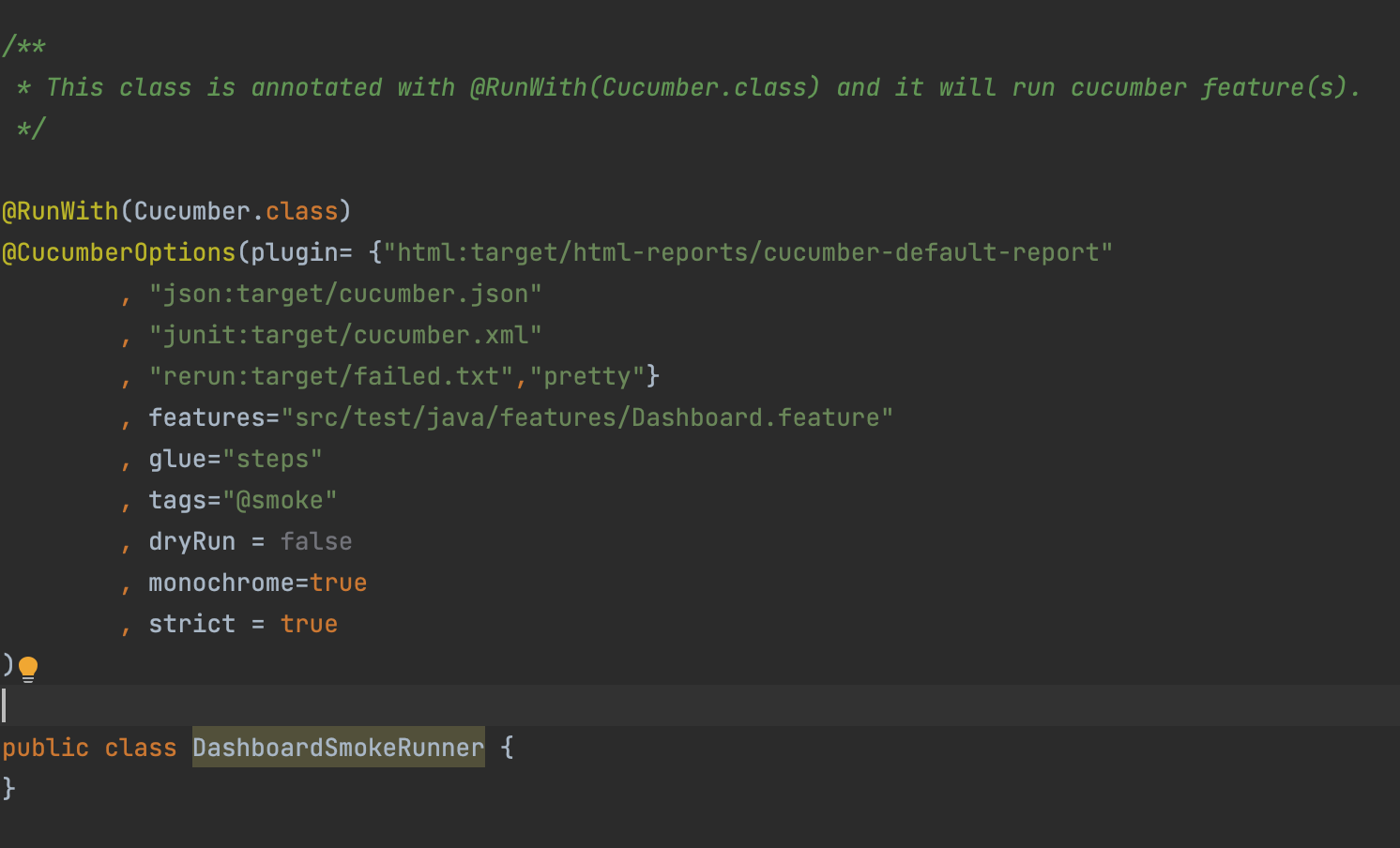
***Difference between Progression Smoke and Regression Runners:***

***Progression Runner* –** We use Progression Runners to run our current code allowing us to see what our current code under development is functioning. Used only for testing purposes to run current, in progress code.

***Smoke Runner –*** We use the Smoke Runner to ensure that the application we are using is stable and all functionalities are working as expected with no issues.

***Regression Runner –*** We use the Regression Runner to run our test cases to ensure that bug fixes or changes in requirements don’t affect the other functionalities of the application.

***Cucumber options explained:***



**plugin:** In plugin, we set the path and specifications to generate our test reports.

**glue:** In glue, we write the name of the package that contains our step’s classes and methods.

**tags:** In tags, we write the name of the tag that corresponds to the type of test case we will be running/testing.

**dryRun:** In dryRun, we set it to true in order to generate any correlating step methods that have not yet been implemented. Cucumber dry run is used for compilation of the Step Definition and Feature files and to verify the compilation errors. The value of dry run can be either true or false. The default value of dry run is false and it is a part of the Test Runner Class file.

**monochrome:** In cucumber, monochrome is used to display console output in color format or not.In monochrome, we can set it to true or false for our desired results.

**strict:** If strict option is set to false then at execution time if cucumber encounters any undefined/pending steps then cucumber does not fail the execution and undefined steps are skipped and build is successful.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

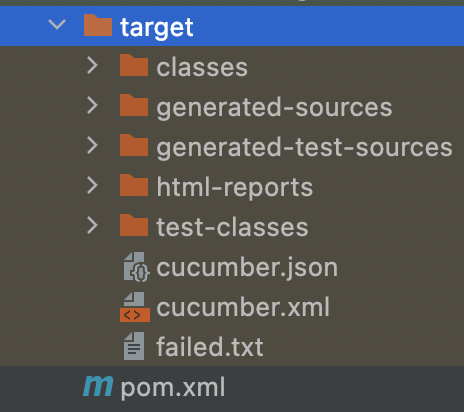
***——---—-- Other Aspects of Framework Explained*** ***—--—-—--***

**Naming Conventions:** Java follows camel-case syntax (example of camel-case= camelCase or helloWorld) for naming the class, interface, method, and variable. If the name is combined with two words, the second word will start with uppercase letters such as actionPerformed(), firstName, ActionEvent, ActionListener, etc.

In terms of naming conventions with classes, class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations

In terms of naming conventions with packages, package names are written in all lower case to avoid conflict with the names of the classes of interfaces.

**Target directory folder:** The target directory is used to house all output of the build. In the target folder, our generated test reports are stored. We run a clean command in maven in order to clear our target folder to store newer updated reports.



**Pom.xml file:** A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects. We place dependencies in our pom.xml file, which allow us to access and use different testing related tools/libraries.

**Organization and Commenting:** Organization, proper spelling, proper spacing, and proper commenting are all important aspects to a high quality automation framework. Commenting gives framework users further context to different pages, classes, and aspects of the framework. Testers must follow established organizational conventions that exist within the framework.

