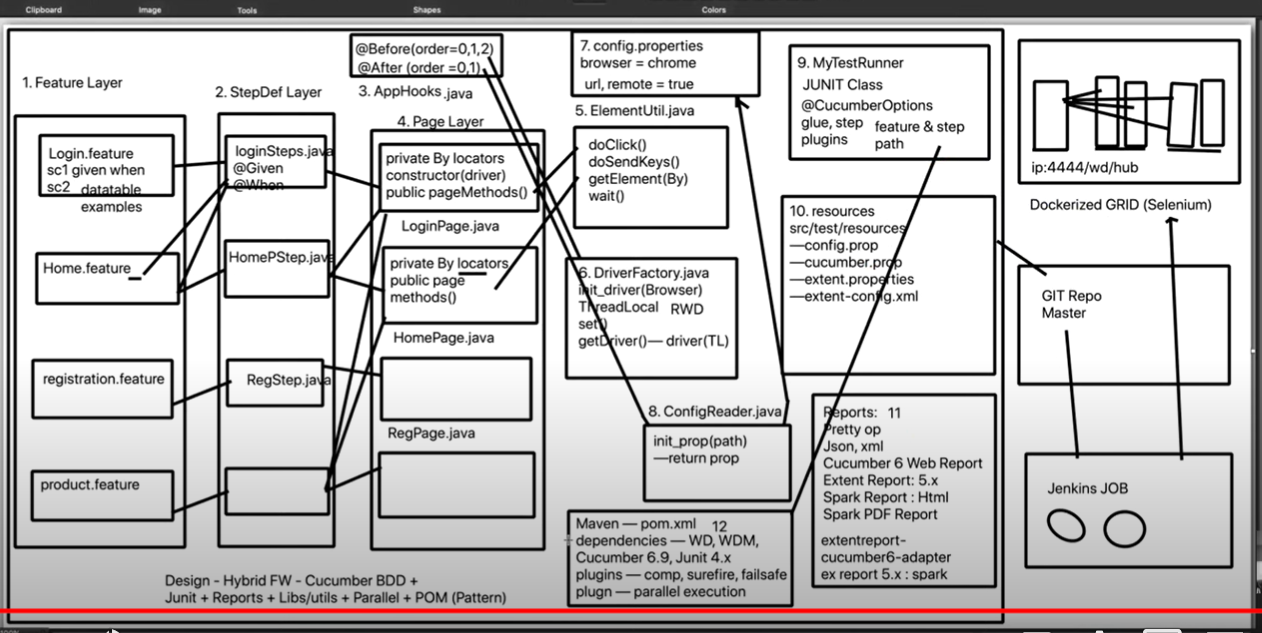
**Cucumber BDD Framework**

The document gives an insight about how a Cucumber BDD framework can be designed from scratch

The Hybrid framework is based on **Page Object Model** design pattern and runner is **Junit**. The framework will follow the SRP principle (Single Responsibility Pattern) wherein each component would be having its own specific meaning which needs to be followed.



The above image gives a brief illustration of the page object model pattern that is used to develop the framework.

1. **Section 1 – Feature Layer (src/test/resources)**

* The feature layer contains the feature files for all the pages. As the framework pattern is Page Object Model hence for each page in the application being automated there would be a separate feature file for the same.
* The feature file would be integrated into its own Step definition file. As Cucumber doesn’t allow duplicate step definitions hence the relationship would be one-many. For example the Home screen feature file will have to access the login page step definition prior to executing home page related scenarios.

1. **Section 2 – Step Definition Layer (src/test/java)**

* The step definition layer contains the step definition java classes for the respective feature files. Each page would have its own step definition file.
* The step definition classes would also act as the Test class of a TDD Page Object model pattern and these classes will have the assertions to validate the scenarios.
* The step definition class has integration to the feature files and also to the Page classes of the respective application pages.

1. **Section 3 – Hooks Class (src/test/java)**

* In the Hooks class we can add the implementation of the @Before and @After hooks. Tagged hooks can be also declared in the Hooks class.
* If multiple hooks are present then we can add order value to give the priority. For @After hook the priority is in reverse order.
* The Hooks class can be integrated with the Driver Initialization class and the Config Reader class. The objects of Driver and Config Reader class can be created in Hooks and assigned to @Before and @After methods.

1. **Section 4 – Page Layer (src/main/java)**

* The Page layer contains the page classes of all the pages in the application. This Page class does the same functionality as the Page class in a TDD framework. Only difference being in TDD the Page class is integrated to the Test Class whereas in the BDD Cucumber framework, the Page class is integrated with the Step definition class which contains the Assertions.
* The Page class contains the object repository and the action methods. Here Encapsulation concept is implemented by declaring the object repository By locators as private and later accessing them in the Action methods using getter methods.

1. **Section 5 – Utilities or Common Methods Class (src/main/java)**

* The Utilities or Common Class contains the reusable methods that can be called from the Page layer classes.
* We can also add custom methods to the Common Class which are specific to the application we are working on which would enhance code reusability.

1. **Section 6 – Driver Initialization Class (src/main/java)**

* Driver Initialization Class can be regarded as one of the most vital parts of the framework pattern. This is where the WebDriver is initialized.
* The Driver Initialization Class functionality is based on the browser that is being selected.
* The Driver Class is integrated with the Hooks Class which is integrated to the Config Reader Class as well from where the browser value is retrieved and passed to the Driver class method called in the Hooks class using the Driver Class object.
* For running the scripts later on a Grid, the initialization of RemoteWebDriver can also be included in the Driver Class.
* Additionally to run the scripts parallel on multiple threads we can implement the ThreadLocal functionality also in the Driver class.

1. **Section 7 – config.properties file (src/test/resources) and**

**Section 8 – Config Reader Class (src/main/java)**

* config.properties file is used to initialize the values of variables such as browser, url, run on remote or not, run on headless mode or not etc.
* config.properties file is integrated with the Config Reader Class which initializes the properties and returns the object of the same.
* The Config Reader class is integrated with the Hooks Class.

1. **Section 9 – Runner Class (src/test/java)**

* The Runner Class is the third major component of Cucumber apart from Feature file and Step definition Class. The Runner Class can be in either Junit or TestNG.
* The Runner Class consists primarily of Cucumber Options functionality.

1. **Section 10 – src/test/resources**

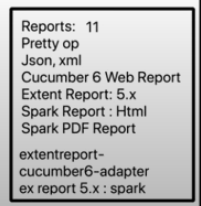
* The source folder src/test/resources is the folder structure to place all the feature files of the application along with the config.properties file.
* Other files to generate reports can also be stored in resources source folder such as –

cucumber.properties

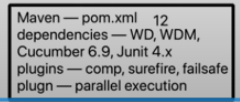
extent.properties

extent-config.xml

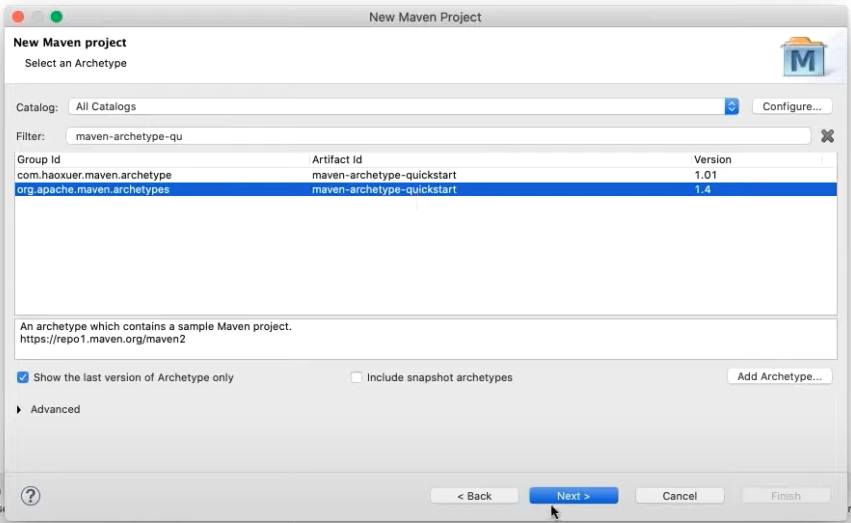
1. **Section 11 – Reporting**



1. **Section 12 – Maven pom.xml file**



**Github Cucumber6 Project -** [**https://github.com/naveenanimation20/LatestCucumber6WithPOM**](https://github.com/naveenanimation20/LatestCucumber6WithPOM)

1. While creating a new Maven project it is a best practice to select **maven-archetype-quickstart** any version as shown in the below image rather than starting as a blank project.
2. Setup the pom.xml file. Set the version numbers in the properties and assign the same in the dependencies and plugins section.









1. Add the plugins in the pom.xml section

