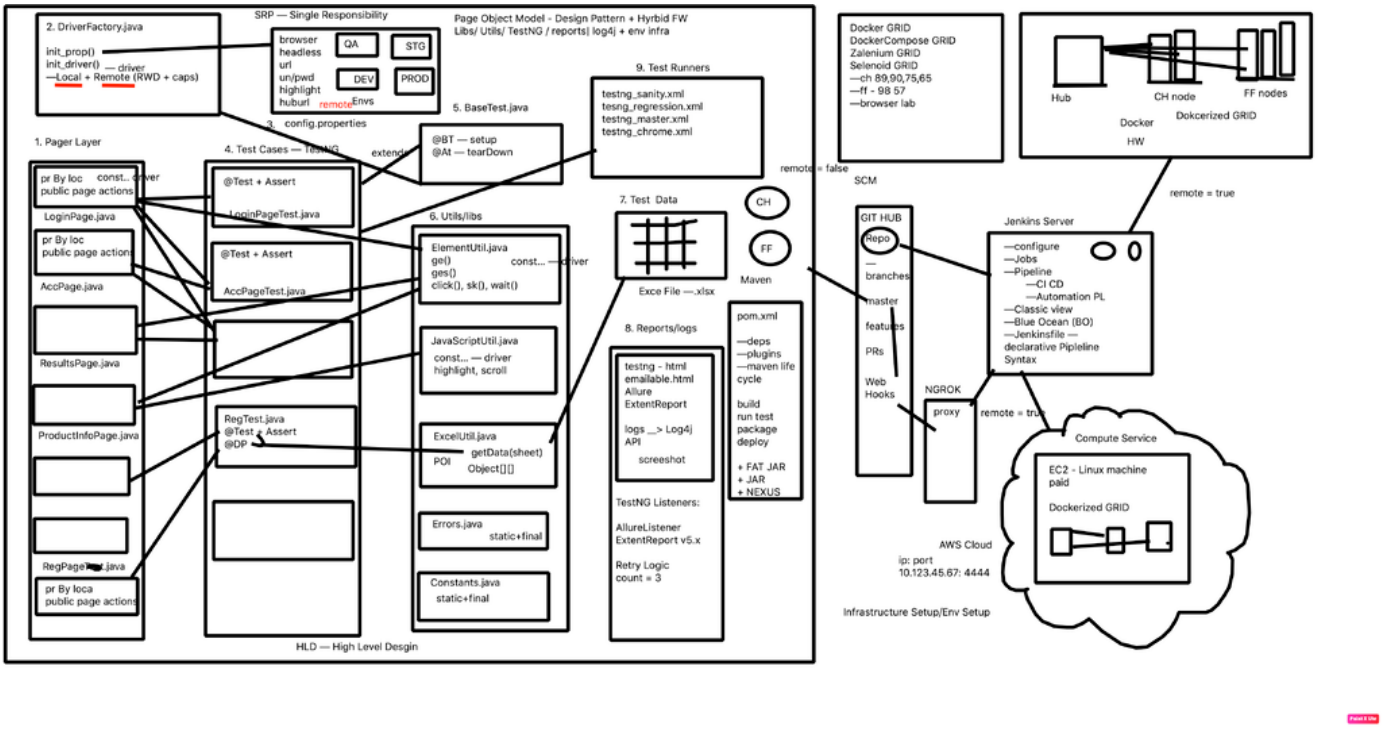
Page Object Model (POM) design pattern with Hybrid Framework.



This diagram represents a detailed design of a test automation framework that leverages the Page Object Model (POM) design pattern combined with some components of a Hybrid Framework.

I'll break down each section and its components:

1. **Page Layer**:
   a. Represents different web pages, which are abstracted using the POM design pattern.
   b. Examples given are **LoginPage.java**, **AccPage.java**, **ResultsPage.java**, **ProductInfoPage.java**, **RegPage.java** where each .java file likely contains page locators and methods to interact with specific elements on the respective page.
2. **DriverFactory.java**:
   a. This is responsible for instantiating the web driver (could be local or remote depending on requirements).
   b. It reads from configurations (**config.properties**) to determine settings like browser type, environment (QA, STG, DEV, PROD), etc.
3. **Test Cases**:
   a. These are the actual test scripts which will make use of page objects and carry out the testing.
   b. They might include setup (**@BT**) and teardown (**@AT**) methods, and tests may use assertions to validate behavior.
   c. Test data can be parameterized using the **@DP** (data provider) annotation.
4. **Utilities/Libraries**:
   a. **ElementUtil.java**: Contains common methods and utilities to interact with web elements (like click, sendKeys, wait).
   b. **JavaScriptUtil.java**: Used to execute JavaScript commands on the web page, useful for operations like scrolling.
   c. **ExcelUtil.java**: Utility to read data from Excel files, which can be used for data-driven testing.
   d. **Errors.java & Constants.java**: Might contain error handling logic and static constant values respectively.
5. **BaseTest.java**:
   a. This could be a base class from which all test classes inherit.
   b. Provides setup and teardown logic, common functionalities, and initializes other common properties.
6. **Test Runners**:
   a. These are configurations or XML files used by test frameworks (like TestNG) to execute specific test suites/cases. Example runners provided: **testsng.sanity.xml**, **testsng.regression.xml**, etc.
7. **Test Data**:
   a. Data, possibly in an Excel file, used for testing.
8. **Reports/Logs**:
   a. Outputs of test runs, possibly in HTML format.Tools like Allure and ExtentReport can be used to generate rich, graphical reports.Logs can be generated using utilities like Log4j.
9. **Infrastructure**:

a. **Docker GRID**: An infrastructure where Selenium Grid runs inside Docker containers. It has nodes for different browsers like Chrome (CH) and Firefox (FF).
b. **Jenkins Server**: Used for Continuous Integration and Continuous Deployment (CI/CD). It can run jobs, pipelines, and has various views.
c. **Compute Service**: Indicates cloud infrastructure (like AWS EC2) where tests might run on Dockerized grids.
d. **GIT HUB Repo**: Source code repository where the test scripts, page objects, utilities, etc., are stored. It's integrated with Jenkins using webhooks.
e. **Maven**: A build tool used to manage dependencies, build projects, and run tests.

10. **Infrastructure Setup/Env Setup**:
    a. Consists of setup on cloud platforms like AWS, the use of proxies like NGROK, etc.

This framework is designed to be modular, scalable, and maintainable. It leverages various tools and technologies to achieve automation, from writing test cases to executing them on different environments and generating detailed reports.