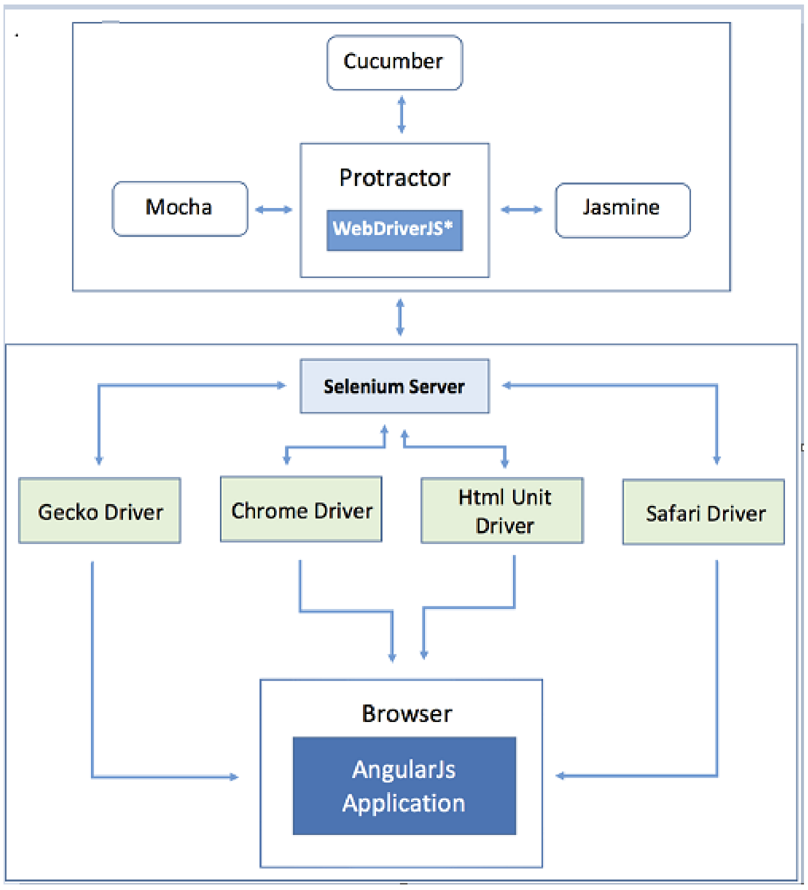
Protractor Implementation

Protractor:

Protractor is an end-to-end testing framework for AngularJS applications and works as a solution integrator combining powerful tools and technologies such as NodeJS, Selenium WebDriver, Jasmine, Cucumber and Mocha.

Protractor supports Angular-specific locator strategies, which allows you to test Angular-specific elements without any setup effort on your part.



Why Protractor over Selenium:

Protractor is built on top of WebDriverJS. Protractor uses WebDriverJS which is based on Selenium. So Protractor is not instead of Selenium, but it is an extra layer on top of Selenium to make testing AngularJS applications easier.

Angular JS applications have some extra HTML attributes like ng-repeater, ng-controller, ng-model, etc. which are not included in Selenium locators. Selenium is not able to identify those web elements using Selenium code. So, Protractor on the top of Selenium can handle and controls those attributes in Web Applications.

Scripting Language: typescript

TypeScript is a superset of JavaScript. One of the big benefits is to enable IDEs to provide a richer environment for spotting common errors as you type the code.

FRAMEWORK:

We are going to use Hybrid Framework that driven by BDD approach

Why HYBRID:

Our Web Automation Framework is going to be the HYBRID Framework that is an amalgamation of Modular, Data Driven, POM Design pattern and BDD approach.

Framework = MODULAR+ DATA DRIVEN+ POM + BDD

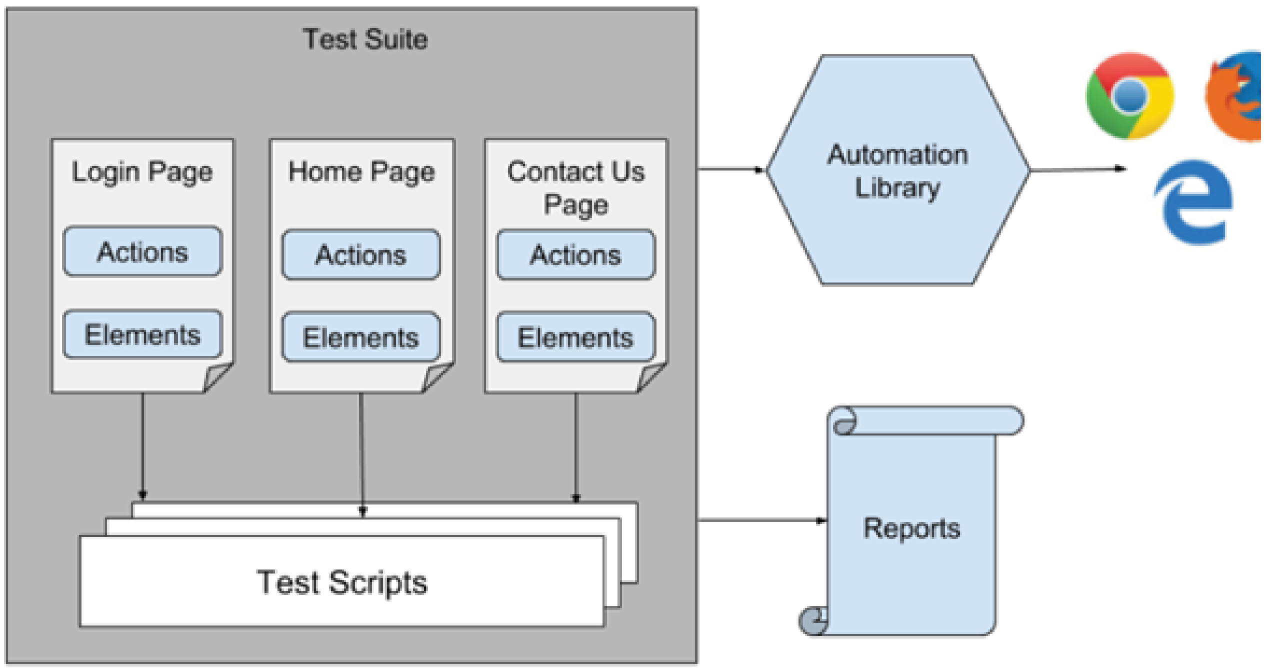
Framework is mainly designed into three parts:

1)Reusable Components:

- *Consists frequently usable functions (e.g. Enter, Wait, Scroll Up or Read Data Utility)*
- *These functions are the utilities that are going to be used in Pages for Performing the actions on the objects*
- *We can achieve **REUSABLITY** Concept by using this*
- *It will help to maintain the script easily and Following DRY **(Don't Repeat Yourself)** principle aimed at reducing repetition.*

2)POM:

- *Page Object Model is a design pattern to create Object Repository for web UI elements.*
- *For each web page in the application, there should be corresponding page class.*
- *This Page class will find the Web Elements of that web page and also contains Page methods which perform operations on those Web Elements.*
- *We can divide the pages into some modules that consists objects and functions. This way we can achieve the **Modularity** of the framework.*
- *This approach is also going the help us to achieve "**Separation of Concerns", separating** an application into distinct pages such that each page addresses a separate concern.*
- *This also provides our framework "**Encapsulation**", where web elements are tightly coupled with the actions and from Test. js we can access only functions, not the objects or web elements*
- *Maintenance of the script will be easier if there is any change in the elements.*



3)Tests:

- *Test Scripts will be written in the Test.js File.*
- *We going to perform each assertion in this file*
- *As a part of **DATA DRIVEN Approach**, data will be taken from external sources like Excel or JSON*
- *It will also give the input for Report based of Results (PASS Or FAILED).*

4)Protractor and BDD:

- *Out of the box, Protractor supports Jasmine. Jasmine allows you to write your specs based on the behavior of the application. This is great for unit tests but may not be the preferred format for business-facing users. What if your business team wants the ability to see a higher-level view of what your suite is testing against? This is where Cucumber comes in.*
- *Cucumber is another BDD framework that focuses more on features or stories.*
- *It mimics the format of user stories and utilizes Gherkin. Cucumber provides your team with living documentation, built right into your tests, so it is a great option for incorporating with our Protractor tests.*
- *It also allows us to better organize suites of tests together with tags and hooks.*
- *js is the well-documented JavaScript implementation of the framework and can be easily incorporated in your Protractor tests.*

5)How is Our Framework going to support BDD:

We are going to utilize Cucumber to incorporate with our framework and support BDD approach. Cucumber is comprising of three important blocks.

1. *Features,*
2. *Config*
3. *Step Definition*

FEATURES:

- *Each and every User story is going be treated as Feature*
- *Feature may have a summary.*
- *Each Features can have one or multiple scenarios.*
- *Scenario should have a Scenario Name*
- *Scenario Description must be written in GIVEN, WHEN, THEN format*
- *It needs to be saved in. feature file.*

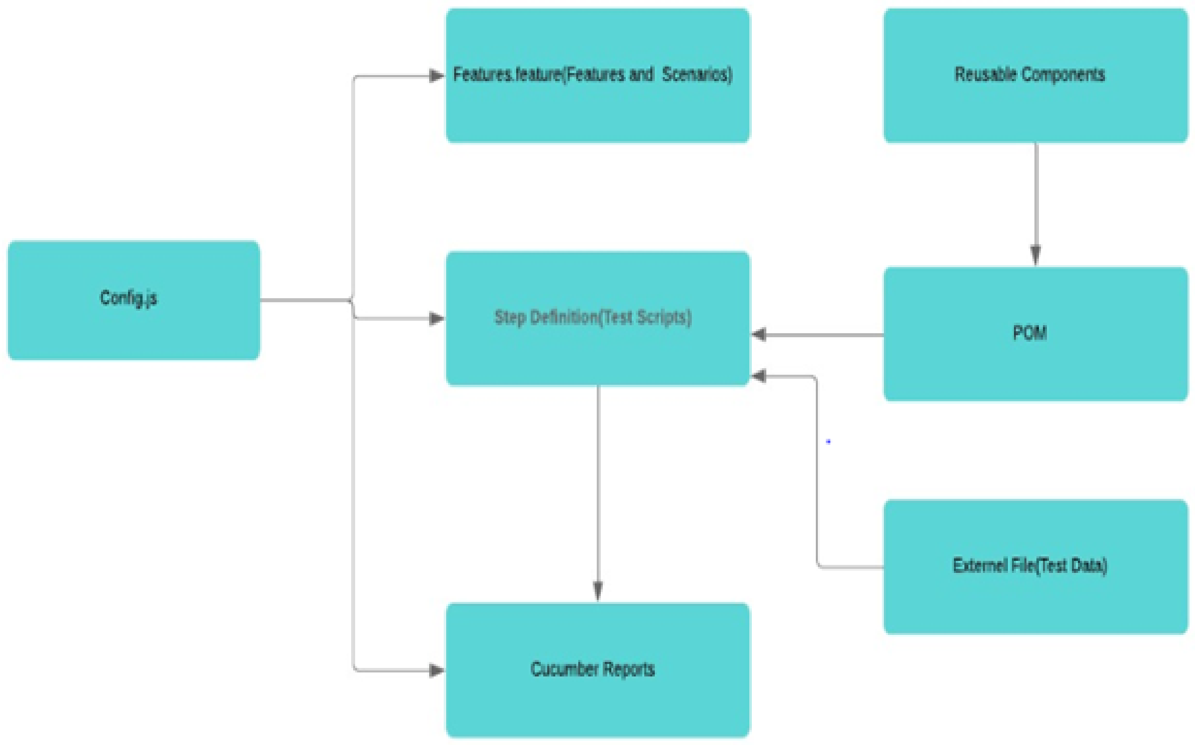
Note: Scenarios of feature are written in simple English, Not in any technical language

STEP DEFINITION:

- *Each Features is being converted into a blueprint, this is called Step Definition*
- *Step Definition is like Test.js where logic or validation, code is going to feed into that template.*
- *One Feature file should consist one or more step definitions.*
- *Step Definition will be going to call the report based on pass or fail of each step*

CONFIG FILE:

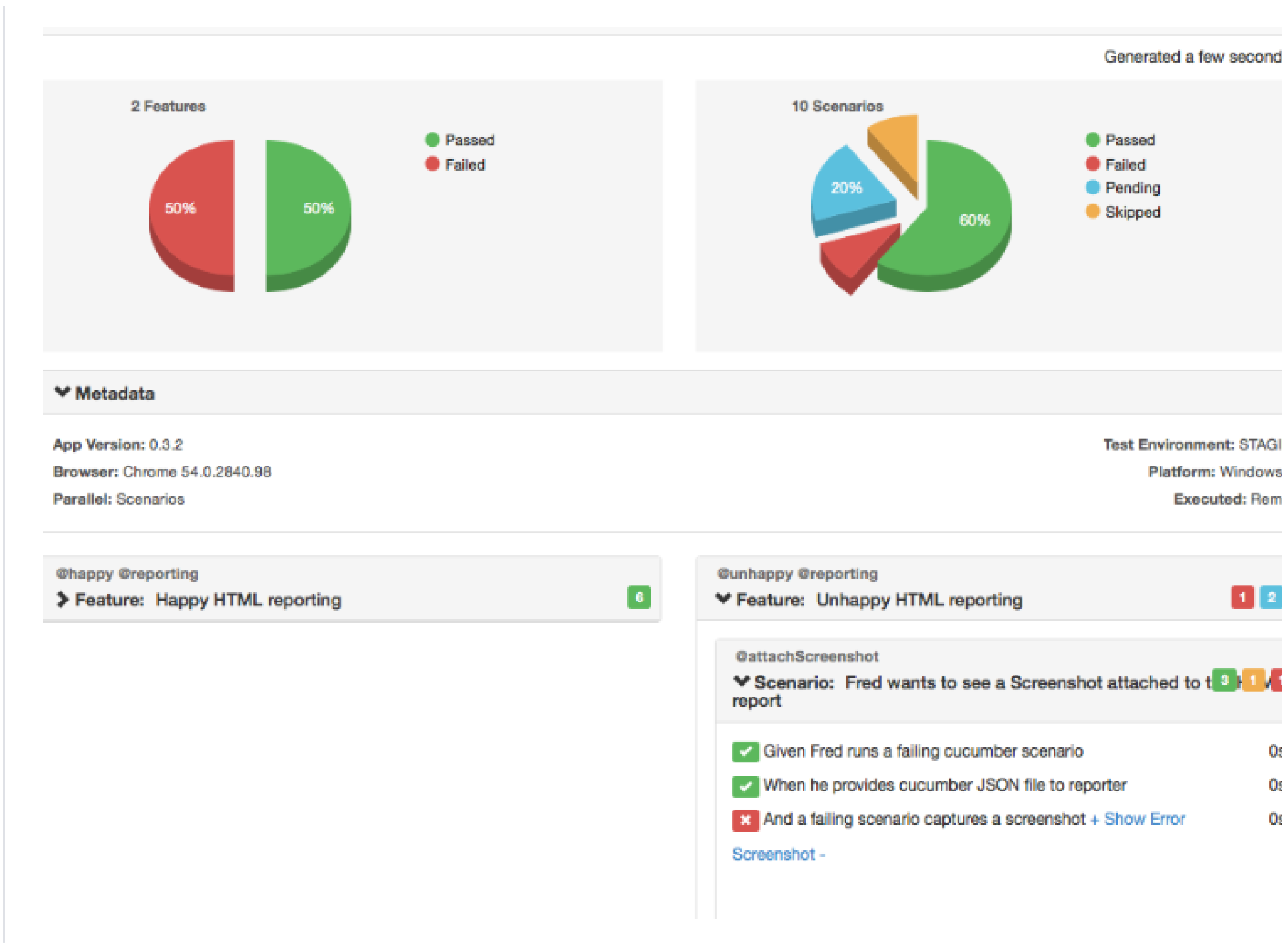
- *Config file is the runner of our framework*
- *It will configure which features and which step definition needs to be executed*
- *Report plugin and other cucumber related details are being given in cucumber Opts of Config file.*
- *Config file will drive every scenario (Features) and Tests (Step definition) and reflects on Reports.*



Basic Block diagram of Framework

6)Reports

We will be using protractor-multiple-cucumber-html-reporter



Generated a few second

2 Features

● Passed
● Failed

50% 50%

10 Scenarios

● Passed
● Failed
● Pending
● Skipped

20% 60%

Metadata

App Version: 0.3.2
Browser: Chrome 54.0.2840.98
Parallel: Scenarios

Test Environment: STAGI
Platform: Windows
Executed: Rem

@happy @reporting
Feature: Happy HTML reporting 6

@unhappy @reporting
Feature: Unhappy HTML reporting 1 2

@attachScreenshot
Scenario: Fred wants to see a Screenshot attached to t 3 1
report

✓ Given Fred runs a failing cucumber scenario 0
✓ When he provides cucumber JSON file to reporter 0
✗ And a failing scenario captures a screenshot + Show Error 0

Screenshot -