# UI Automation Framework

This is a starter guide to help users of the simple selenium-based framework that we created to help our automation activities in Ascend Learning. Please not that this is a ‘growing’ framework and we can keep adding functionality as and when we need it.

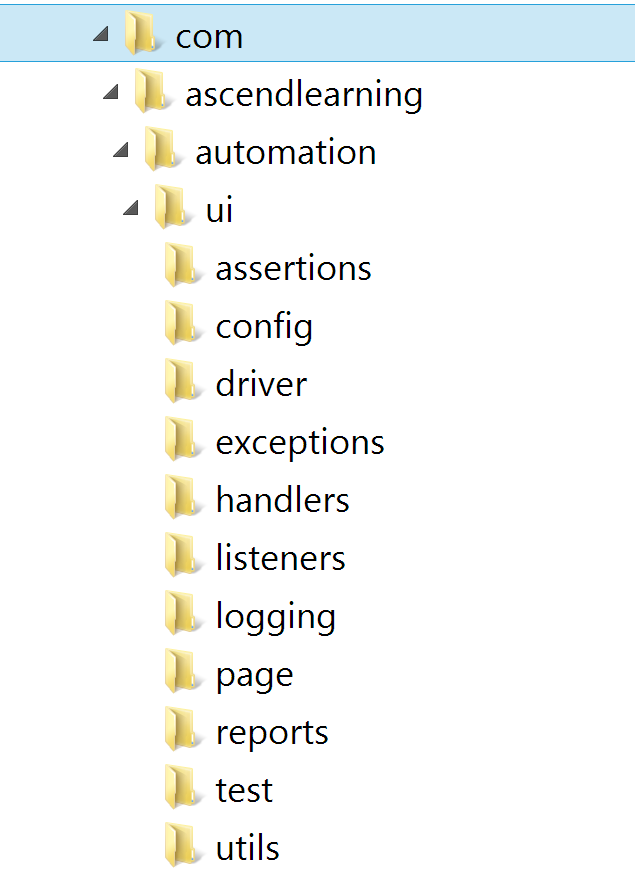
## GitHub Repository

[**https://github.com/gautamsabba/uiframework.git**](https://github.com/gautamsabba/uiframework.git)

## Salient Features

1. We’ve tried to keep things as simple as possible without over-engineering the framework classes
2. We’ve used standard practices that are widely accepted in the community (well, at least the articles and blog posts we’ve come across)

## Package Structure



## Primary Components

### WebDriver

The WebDriver instantiation has been designed to support grid-based execution of the tests. An instance of a WebDriver can be retrieved using **DriverFactory.getInstance().getDriver();** but this is extracted to the base classes (more about these later) and you don’t need to worry about it.

### Base Classes (**BasePage** and **BaseTest**)

All your test classes MUST extend **BaseTest** and similarly your page objects MUST extend **BasePage**. Yes, we follow the page object pattern here but we leave it up to you on how you want to design your page objects. Some have one page object per physical page some use a page object for an entire flow. This is entirely up to you. *Hint: More page objects = more maintenance headach*e

Ensure that your test methods are independent of each other so that they can be run in grid mode without any constraints

**BaseTest** uses the **@BeforeMethod** and **@AfterMethod** annotations for setup and teardown.

#### Test class skeleton

**public class YourTest extends BaseTest {**

**Logger logger = LogManager.getLogger(this.getClass());**

**@Test(groups = { "smoke" })**

**public void testFunctionality() {**

**YourPage yPage = new YourPage(driver);**

**String result = yPage.functionalityMethod();**

**VerificationHandler.verifyEquals(result, PropertiesRepository**

**.getString("property.containing.gold.data"));**

**}**

**}**

#### Page object skeleton

**public class YourPage extends BasePage {**

**private Logger logger = LogManager.*getLogger*(this.getClass());**

**public YourPage (WebDriver webDriver) {**

**super(webDriver);**

**}**

**private void loadMainPageURL() {**

**driver.get(PropertiesRepository.*getString*("mainpage.url"));**

**setDriverWait(PropertiesRepository**

**.*getString*("jblearning.mainpage.url.waitfor"));**

**}**

**public String functionalityMethod () {**

**// Load main page**

**loadMainPageURL();**

**MenuHandler menuHandler = new MenuHandler(driver);**

**try {**

**menuHandler.hoverOverMenuItem( PropertiesRepository.*getString*("mainpage.menu.item1"),**

**PropertiesRepository.*getString*("mainpage.menu.**

**item1.waitfor"));**

**} catch (DriverException e) {**

**logger.error("Error while hovering over menu item", e);**

**}**

**TextHandler textHandler = new TextHandler(driver);**

**String result = null;**

**try {**

**result = textHandler.getText(PropertiesRepository**

**.*getString*("detailspage.desc"));**

**} catch (DriverException e) {**

**logger.error("Unable to get book description", e);**

**}**

**return result;**

**}**

**}**

NOTE: Please DO NOT return page objects in your page object methods, it makes the test code unreadable when you do something like this –

**Obj1.meth1().o2meth1().o3meth3()**

### Configuration and Properties Files

All modifiable content should be contained in properties files. Predominantly selectors live here. There are a few properties files provided in the framework whose values can be overridden in your own properties files.

**global.properties** – contains platform and browser information as well as **DesiredCapability** information for the chosen browser

**log4j.properties** – properties to define logging syntax

Create your own properties files for locators and other name/value content that you require for your test suite. However, ensure that you also create a **prop-files.properties** in your /resources folder and include the names of your properties file like so

**propFiles=global-override.properties,one.properties,two.properties**

### Selectors and Locators

We strongly advice the use of native CSS selectors ONLY for locating web elements. For cases where native CSS locators are inadequate like referencing the content of a tag we have implemented Sizzle a Javascript-based CSS selector library.

So, for an anchor tag like this

<**a href=**[**http://google.com**](http://google.com)**>This is Sparta!</a>**

Your selector for the anchor element will look like this – **a:contains(‘This is Sparta!’)**

Please look at <https://github.com/jquery/sizzle/wiki> for other supported functionality.

### Exceptions

We have a single simple exception – **DriverException** that can be used to add your own custom message to standard exceptions.

### Handlers

Several handlers have been added for convenience when operating on standard web elements and components like, menus, buttons, windows, popups, radio buttons, dropdown, etc. The methods self-explanatory

### Waits

This bit it important. We strongly recommend using explicit waits for better efficiency. Explicit waits work on the principle that an operation waits for completion till a desired web element becomes visible. It is different from implicit waits which always wait for a predetermined time. Please review the sample project provided at the end of this document for examples.

### Logging

Logging is pretty straightforward. We use Log4j as usual.

### Utils

We have some rudimentary utility methods for string parametrization. This will grow based on need.