Web UI Testing Discussion

# Types of automated tests

## Automated Unit Tests

Testing an individual unit of code, where bugs can be identified in functions and methods.

Some of the most populate testing frameworks are NUnit, JUnit, and Microsoft’s MSUnit.

## Automated Web Service / API Tests

When testing the API layer, the GUI is generally not involved. At this level, we can for functionality, compliance, and security issues.

We can also test the Request and Response to see whether they are secure and encrypted.

Some populate API testing tools are SoapUI, Postman, and Swagger (used closely with Identity Server).

## Automated GUI Tests

This is the most challenging form of automation because it involves the UI. Since the UI is highly subject to change, UI automation testing is critical to catch issues that may be hard to test manually. For example, long and boring UI tests (such as filling out long HTML forms) can be quickly run with UI automation tests.

# Testing Misconceptions

## Automation replaces human testers

Human testers can never be replaced; however, it assists in making the tests faster and more reliable.

You can use the automations to quickly test the long and boring tests, allowing you to focus on new and important functionality.

## Everything can be automated

We cannot automate 100% of our test cases. If that were the case, we wouldn’t need manual testers. In fact, there are cases where manual testing is easier than automation (i.e a workflow situation where multiple users are involved; one enters data, another approves data, etc.).

So in reality we cannot automate 100% of our test cases; and certain bugs will be discovered once they are rolled out to the client.

# Automation only involves recording and playback

Some opinions state that pure automation engineers do not use playback at all. And if used, they would only be used to determine how a tool is generating scripts.

Those opinions state that all automation tests are scripted.

# Automation Tools

## Screenster

* Record and playback via Screenster projects
* Playback your previous tests, comparing old and new screen shots

## Selenium

Supports Java, C#, Python, Ruby, Perl, PHP, and JavaScript.

See “Introducing the Selenium-WebDriver API by Example” at <http://www.seleniumhq.org/docs/03_webdriver.jsp#introducing-webdriver>

## Protractor

E2E testing for Angular-based apps. Based on Selenium WebDriver.

## SpecFlow - Business Readable UI Automation

* Document the features of the system we’re building in a non-technical manner
* Executable as tests
* The buns readable tests are source controller, alongside the test code
* The documentation stays up-to-date; tests will fail if the code does not match the docs

## Backstop JS

Based on PhantomJS, CasperJS, and ResembleJS, Backstop JS is a Visual Regression Testing framework.

PhantomJS does the screen capturing, CasperJS does the UI Navigation Scenarios, and ResembleJS takes care of the image analysis and comparison.

# Demo Apps on Azure VM

## MEAN.JS Sample App

### Installed in C:\dev\meanjs\mean (see ..\meanjs\ for app’s user/pswd ))

1. Run the grunt and mongo bat files on deskstop
2. Launch <http://localhost:3000/>
3. Run tests: > ***grunt test –force***

### Run custom tests (..\volunteer\volunteer\modules\tests-bob\)

1. > webdriver-manager start
2. > protractor conf.js

## Selenium Sample App

C:\dev\selenium-demo\SeleniumSupport\, ***LoanApplicationSite*** solution file.

Tests are created in the ***DemoWebApp.Tests*** project.

**First run the LoanApplication** website in non-debug mode – BEFORE running tests.

## Protractor Tutorial

Tutorial Url: <http://www.protractortest.org/#/tutorial>

On VM, see c:\dev\protractor-test\

* **Run tests:**

1. > webdriver-manager start
2. > protractor conf.js

## Screenster install

Launch Chrome browser at <http://localhost:7070/>