BMC Software, Inc.

Technical Disclosure Publication Document

Authors

Mahesh Joshi
Rahul Kulkarni
Gaurav Pal

SaaS-Based Tool for Testing JavaScript/HTML-Based UI Using Standard Browser Capability

Posted: October 28, 2015

Overview

This document describes a new user interface (UI) testing framework that addresses the problems faced by quality assurance (QA) teams in automating the testing of HTML/JavaScript-based UIs. The proposed solution resolves short-comings in current commercial automation tools. The proposed solution is to use the browser's native UI-rendering capability for testing. The proposed solution also uses SaaS-based delivery of the tool, which helps avoid the setup and configuration of the tool on test machines.

Background

Currently, UI automation is based on commercial or free tools that are commonly available.
All of these tools require setup and configuration and hence require dedicated/identified test machines on which automation can run. Most of these tools use different technologies to interact with the browser; some use keyboard/mouse drivers to generate action, others use browser-provided hooks. These interaction methods can change across browser version and browser vendors; making it hard to maintain.

Some tools (ex. SilkTest, selenium) do not allow any other human activity on the computer while the tests are being executed; effectively locking out the test machine during test runs. This prevents multiple tests being run simultaneously from the same machine.

Some tools (ex. SilkTest) require knowledge of programming language to write test cases using application programming interfaces (APIs) provided by the vendor. Some vendors do not support all browsers (ex. iMACRO does not support Internet Explorer). In such cases, QA teams are forced to develop tests cases separately for each browser they wish to execute the tests on.

Solution

We are proposing an architecture that resolves all of the above issues:

1. The proposed solution is SaaS based, with all scripts being downloaded to the browser during execution. Because there are no plugins or libraries to be installed, there are no requirements to setup, install, or configure prior to the test's execution.
2. The proposed solution is completely based on standard resources (HTML, javascript) available on all browsers, so it works seamlessly on all browsers (versions and vendors) that support these standards.

3. The proposed solution does not include or use any extra javascript UI libraries (extJS, jQueryUI, etc.) for its execution. However, if included by application, it uses those to interact with the UI elements.

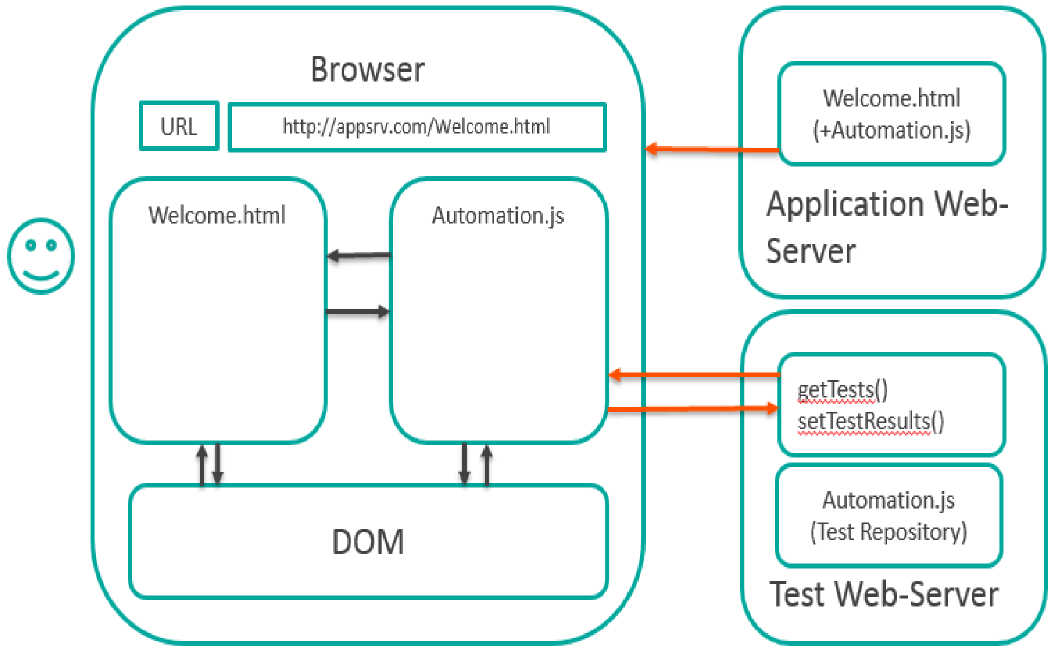
The core of the proposed solution is a generic javascript that simulates user actions. This javascript is based on a version of the European Computer Manufacturers Association (ECMA) standard that every javascript engine implementation follows. This javascript is made available in SaaS mode to any web-application for use. Any existing (or new) web-server can be configured to host this javascript.

Inclusion of javascript in the HTML/JSP pages can be done by any known method (static injection in HTML, dynamic injection at web-server, or dynamic injection at web-server on conditions) and at different places (build time, run-time). The script should appear at the end of the HTML files. The javascript execution, in browser, can also be controlled using Uniform Resource Locator (URL) parameters.

The javascript, once loaded, talks to the web-server (using AJAX) running the web-application that provides requested test-cases and stores results on the disk or in the database. URL parameters can be used to communicate extra information. The JSON format is a preferred way for exchange as the browsers have in-built knowledge of JSON format. The test cases communicated using JSON are not in the ready-to-execute javascript code. The javascript parses JSON structure and decides and triggers instructions accordingly. The web application is responsible for deciding the result of the execution. The web application may optionally compile the test cases to handle pseudo-language or multi-language testing of the UI (by parameterizing parameters and strings in test cases).

The test cases can be reused in multiple scenarios and by sharing with other developers that modify the UI further.

Drawings


Browser
URL
http://appsrv.com/Welcome.html
Welcome.html
Automation.js
DOM
Welcome.html
(+Automation.js)
Application Web-Server
getTests()
setTestResults()
Automation.js
(Test Repository)
Test Web-Server