Software Design Description

CSE 361 - Spring 2014

**Regression Testing System**

|  |  |
| --- | --- |
| Authors:  Alex Tobias  James Duin | Supervisor:  Dr. Greg Rothermel |

February 28, 2014

TABLE OF CONTENTS

1. INTRODUCTION 3

1.1 Purpose 3

1.2 Scope 3

1.3 Overview 3

1.4 Reference Material 3

1.5 Definitions and Acronyms 3

2. SYSTEM OVERVIEW 4

3. SYSTEM ARCHITECTURE 4

3.1 Architectural Design 4

3.2 Decomposition Description 6

3.3 Design Rationale 6

4. DATABASE DESIGN 6

4.1 Data Description 6

4.2 Data Dictionary 7

5. HUMAN INTERFACE DESIGN 7

5.1 Overview of User Interface 7

5.2 Screen Images 7

5.3 Screen Objects and Actions 12

6. REQUIREMENTS TRACEABILITY 13

**1. INTRODUCTION**

**1.1 Purpose**

The purpose of this software development document is to describe the architecture and system design of our Regression Testing System so that it can be easily implemented during the development stage.

**1.2 Scope**

For our project, we will create a regression testing software platform for Quality Assurance (QA) testers. The user base would be QA testers that have groups of browser automation test cases but do not have a way of running those test cases on a schedule or on multiple browser/os combinations. The idea is the qa tester would access our regression testing web app and would upload the code for their browser automation test cases. They would then group the uploaded test cases into test suites via configuration pages on our website. On another configuration page they would specify the testing schedules they desire. These testing schedules consist of the following: the test suite, the time schedule they want the tests run, the browser/os combination, and the environment (live or dev url) they want to test on. The tests then will be run automatically for them and they will be able to see the test result logs on the website. They also will be able to see summary report pages that the website will generate from the test results. The reports will provide QA testers with the ability to track percent failure and test time for a testing suite over successive releases. It will not allow the user to run the tests on demand, or support more than one teams configuration. However, the system could be expanded to offer these capabilities in the future.

**1.3 Overview**

This document will describe, in detail, the system as a whole first. Then, it will go into the details of the architectural design of the system, followed by the data design of the system. The next part of the document will go over the different components of the system, and finally it will go over the different user-interfaces of the system.

**1.4 Reference Material**

The following web addresses are references for the component open source software systems that will be integrated into our Regression Testing System:

1) Jenkins. (n.d.). *About Jenkins CI*. Retrieved Feb 9, 2014, from Jenkins: <http://jenkins-ci.org/content/about-jenkins-ci>

2) Sahi. (n.d.). *About Us | Sahi*. Retrieved Feb 9, 2014, from Sahi: <http://sahi.co.in/about-us/>

3) Selenium. (n.d.). *About Selenium*. Retrieved Feb 9, 2014, from Selenium: <http://docs.seleniumhq.org/about/>

4) VirtualBox. (n.d.). *Screenshots – Oracle VM VirtualBox*. Retrieved Feb 9, 2014, from VirtualBox: <https://www.virtualbox.org/wiki/Screenshots>

**1.5 Definitions and Acronyms**

The definitions, acronyms, and abbreviations used in this document are as follows:

**QA** - Quality Assurance

**Test Case** - A browser automation script possibly written to be run by Selenium or Sahi.

**Test Suite** - A grouping of test cases, test case groupings are not mutually exclusive.

**Web app** - Web based application.

**IP address** - Internet Protocol address.

**Environment** - The url that the tests are to be executed against. This can be either point to a live site, or a development site that contains features are currently being worked on.

**CSE** - Computer Science and Engineering department at the University of Nebraska at Lincoln.

**VM** - Virtual Machine.

**Front end** - This references the web application portion of our software system.

**Back end** - This references the part of our software system responsible for executing the configuration completed by the user in the web app.

**XML** - Extensible Markup Language

**Jenkins** - Jenkins is an open source continuous integration tool written in Java. The project was forked from Hudson after a dispute with Oracle. Jenkins provides continuous integration services for software development.

**Sahi** - An open source browser automation tool.

**Selenium** - An open source browser automation tool.

**VirtualBox**- A virtualization software package for x86 and AMD64/Intel64-based computers from Oracle Corporation as part of its family of virtualization products.

**JSP page -** JavaServer Pages, a technology that helps [software developers](http://en.wikipedia.org/wiki/Software_developer) create [dynamically generated web pages](http://en.wikipedia.org/wiki/Dynamic_web_page) based on [HTML](http://en.wikipedia.org/wiki/HTML), [XML](http://en.wikipedia.org/wiki/XML), or other document types.

**JDBC -** Java Database Connectivity, a java library that allows a java app to connect to a MySQL database and run queries on the database.

**2. SYSTEM OVERVIEW**

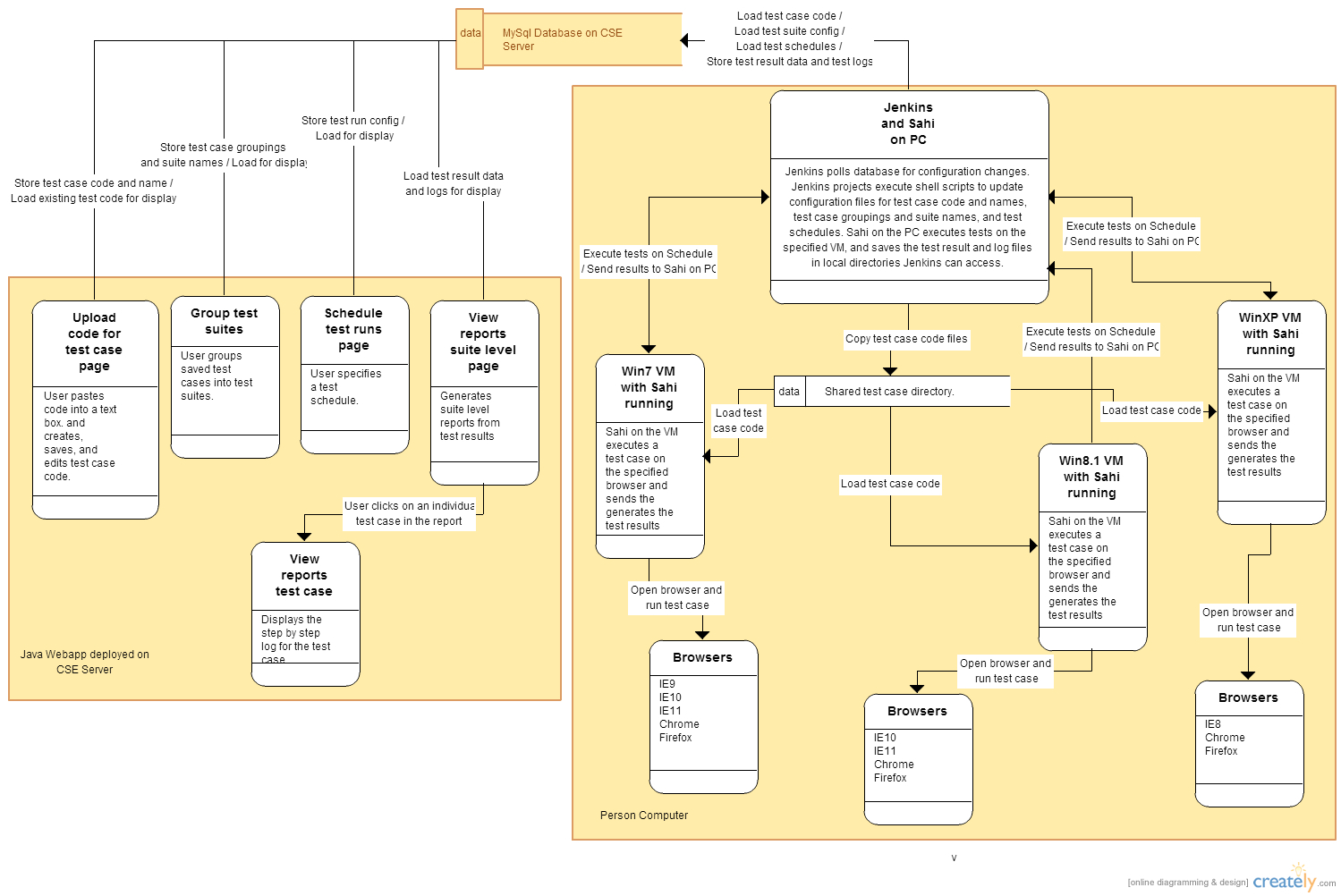
This system will be made up of several different components. The first is several outward facing jsp pages that allows the user to create, save, configure, and schedule test cases and test suites. The second component is the MySQL server that stores all of the data for the system, including test cases, test suites, test schedules, and test case results. This communicates with the jsp pages using jdbc, a java database connection library. The third component of this system is the Jenkins application that grabs test cases and test suites from the database, runs the tests using the correct configurations, and returns the test results to the database. Finally, the results jsp page will retrieve the test results from the database, and populate a table with the information for the user to view.

**3. SYSTEM ARCHITECTURE**

**3.1 Architectural Design**

The code behind the web app will be object oriented. The four main objects will be TestResult, TestSuite, TestCase, and TestSchedule. Each of these objects will have their own functions for data access, which will be done using the jdbc.

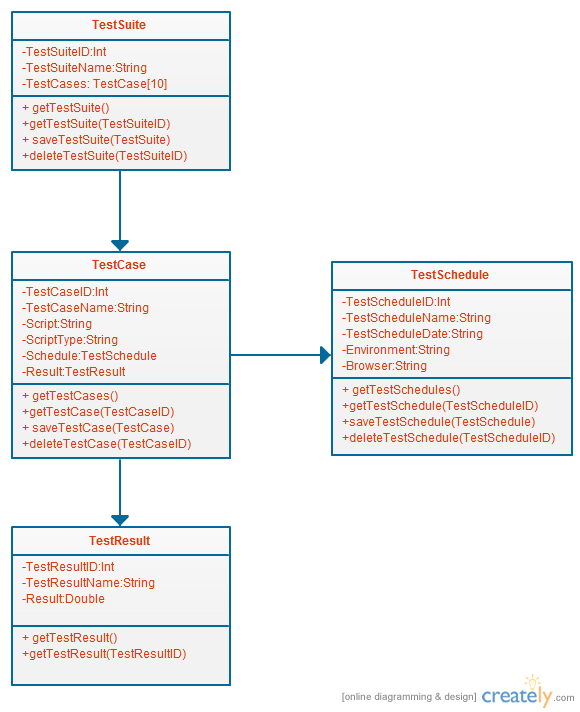
The following Data-flow diagram details how the system is to be implemented.

****

The system consists of the following three main components, the Java webapp deployed on the CSE Server, the MySql database on the CSE server, a personal computer that executes the tests. The webapp contains five pages that display test results to the user, allow the user to upload test case code, group test cases into suites, specify the test schedule, and allow the user to see suite reports and test case logs. The webapp stores all of the configuration and test case code in the MySql database. The webapp accesses the test results and log files stored in the database and generates suite reports and displays the test case logs. The PC loads the configuration from the database, executes the test suites, and updates the database with the test results and test case logs. The PC has Jenkins and Sahi running and three testing VMs with Sahi running and several browsers installed on each of the VMs. Jenkins on the VM polls the database for configuration changes and then executes Jenkins tasks and shell script to update the necessary Jenkins project config files. Jenkins takes test case code stored in the database and loads it into test case files and saves those in a test case directory that is shared between the VMs and the PC. When it is time for a test suite to get executed, Jenkins executes the test case on the specified VM with a shell script. The Sahi instance on the VM opens the specified testing browser and executes the test. When the test is finished, Sahi on the PC will have generated test result and log files in a directory within the Sahi installation on the PC. At this point the Jenkins project that executed the shell script to start the Sahi test, executes a subsequent task to store the test case logs and test results into the database.

**3.2 Decomposition Description**

The UML diagram below shows the variables and methods for the main objects in the webapp.

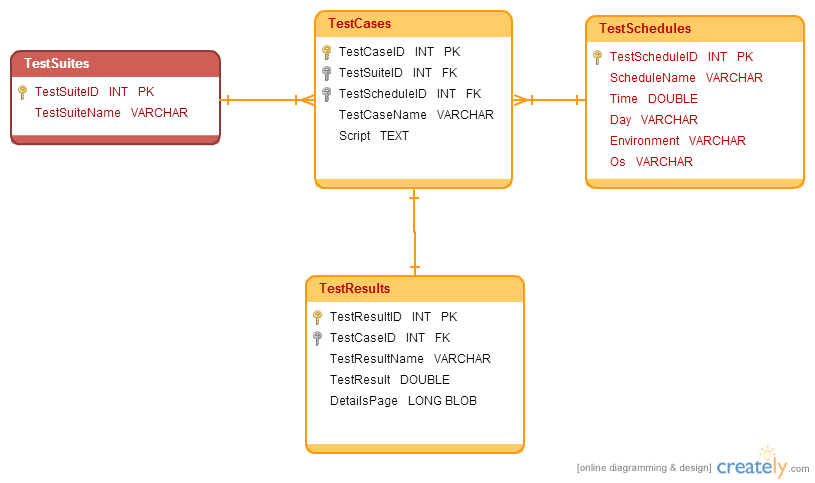
****

**3.3 Design Rationale**

We chose to write our system in an object oriented style because it will make data access and parsing of data much easier for the webapp. Also, since we were limited to using java jsp pages for the webapp, it makes sense to write the webapp in an object oriented style, because that is one of java’s strongest components.

**4. DATABASE DESIGN**

**4.1 Data Description**

****

**4.2 Data Dictionary**

* TestSuites - has a one to many relationship with TestCases table
  + TestSuiteID - autogenerated primary key
  + TestSuiteName - the name of the specific test suite
* TestCases - has a many to one relationship with TestSchedules table, has a one to one relationship with the TestResults table
  + TeCaseID - autogenerated primary key
  + TestSuiteID - foreign key that identifies the test suite that the test case is in
  + TestScheduleID - foreign key that identifies the test schedule that the test case uses
  + TestCaseName - the name of the specific test case
  + Script - the script that the test case uses, is stored as a text object to avoid script length restrictions
* TestSchedules
  + TestScheduleID - autogenerated primary key
  + TestScheduleName - the name of the specific test schedule
  + Time - the time that the test script should start running
  + Day - the day that the test script should run on
  + Environment - the environment that the test script will run on
  + Os - the virtual machine that the test script will run on
* TestResults
  + TestResultID - autogenerate primary key
  + TestCaseID - foreign key that identifies the test case that the individual result is for
  + TestResultName - the name of the specific test result
  + Result - The numerical result for the test script
  + DetailsPage - The html page that is opened for more details on a test result, automatically generated by Jenkins, is stored as a BLOB (binary large object)

**5 HUMAN INTERFACE DESIGN**

**5.1 Overview of User Interface**

The user-interface for this system will be 5 jsp pages running on the cse glassfish server. There is a page to create, edit, and save test cases, a page to create, edit, and save test suites, a page to to create, edit, and save test schedules, a page to view all test results, and a page to view detailed information for a specific test result.

**5.2 Screen Images**

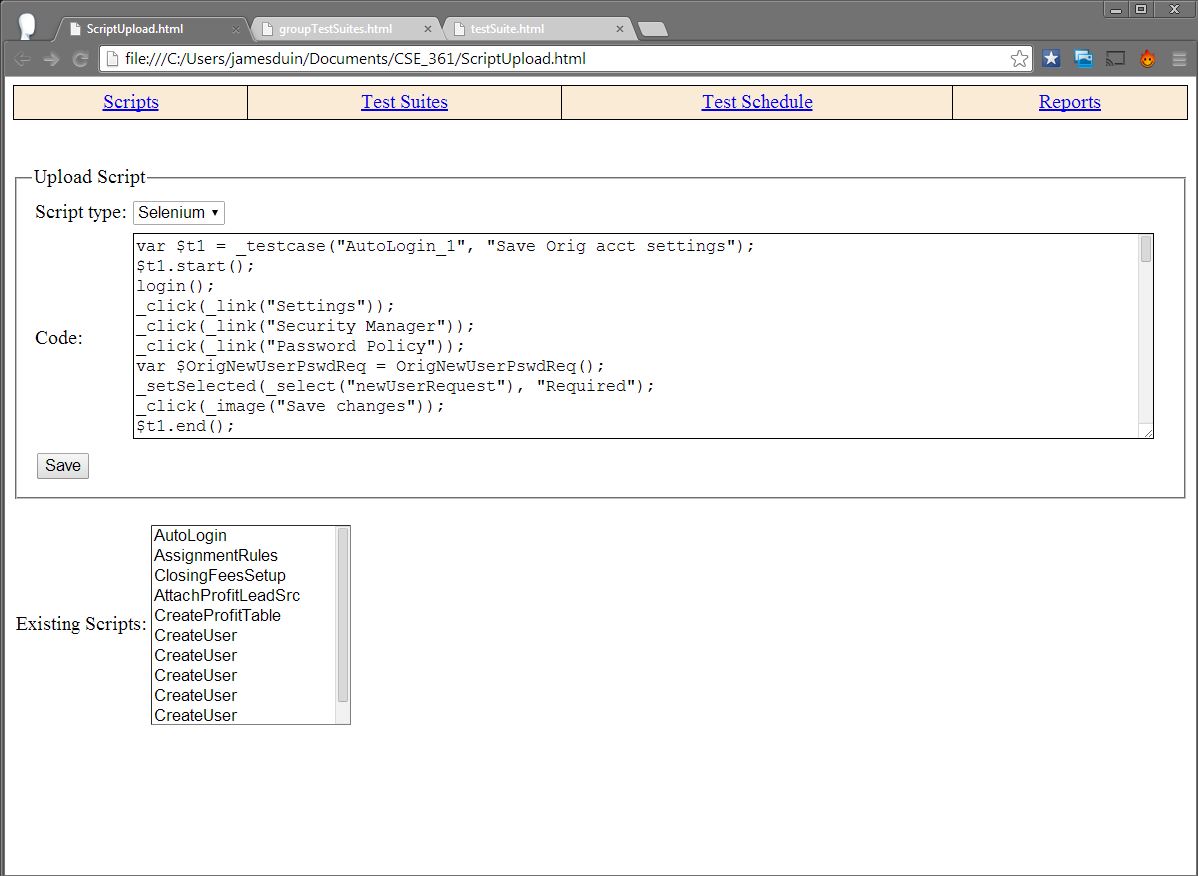


Figure 1: Page to upload a script.

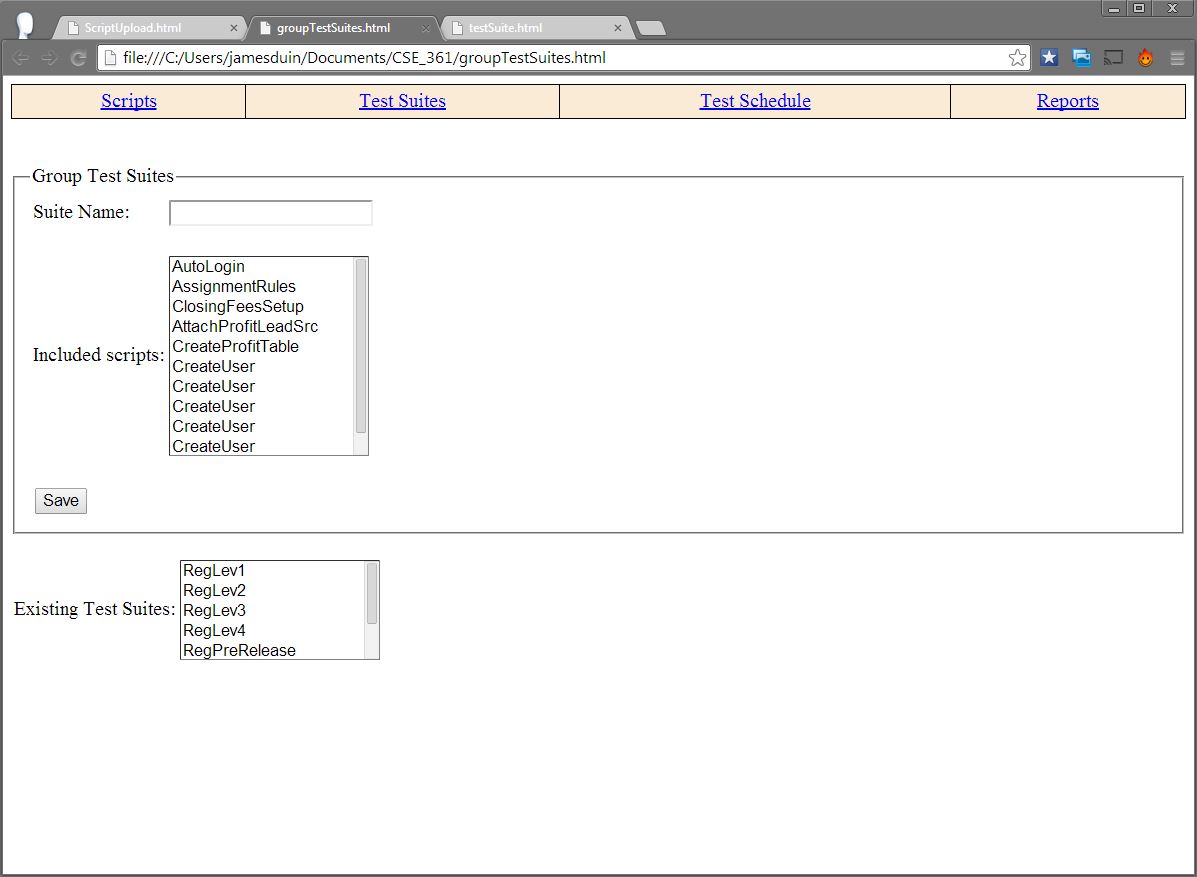


Figure 2: Page to group scripts into test suites.

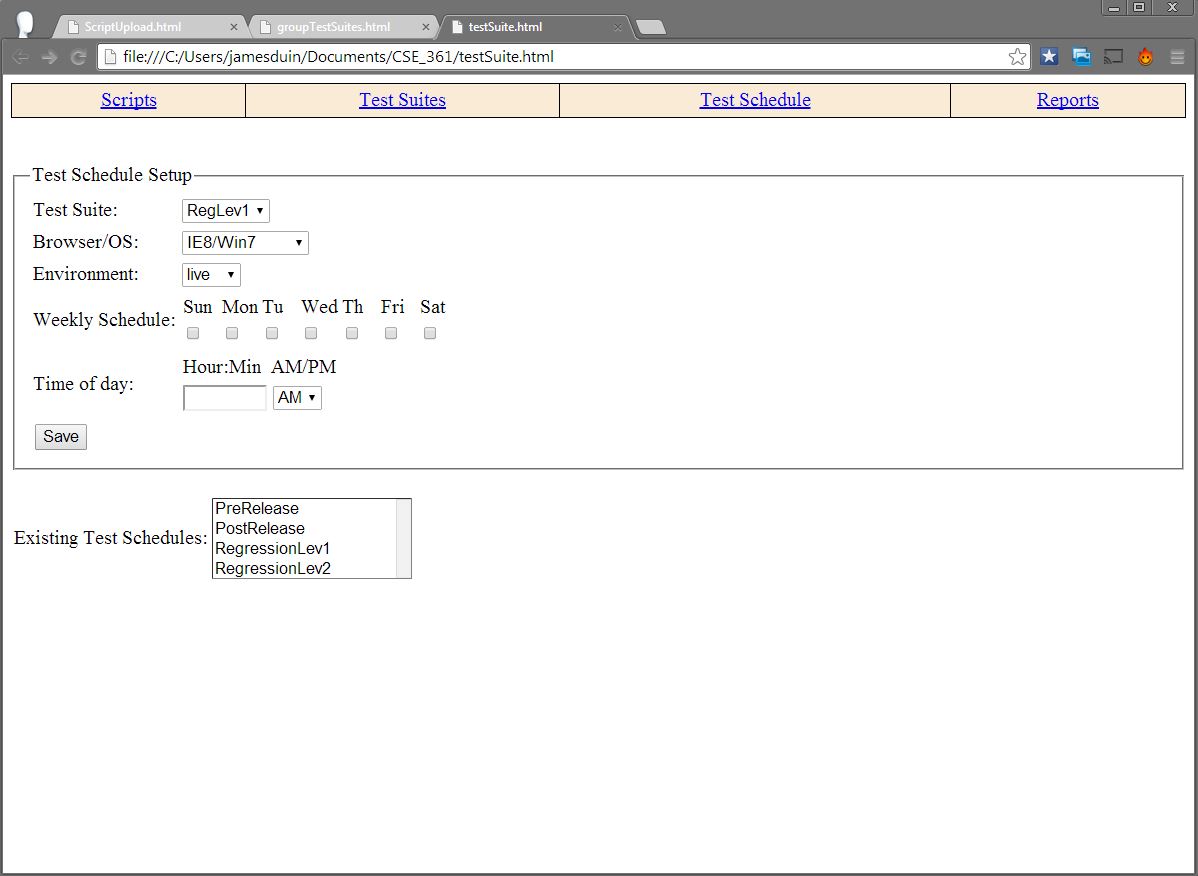


Figure 3: Page to specify a testing schedule.

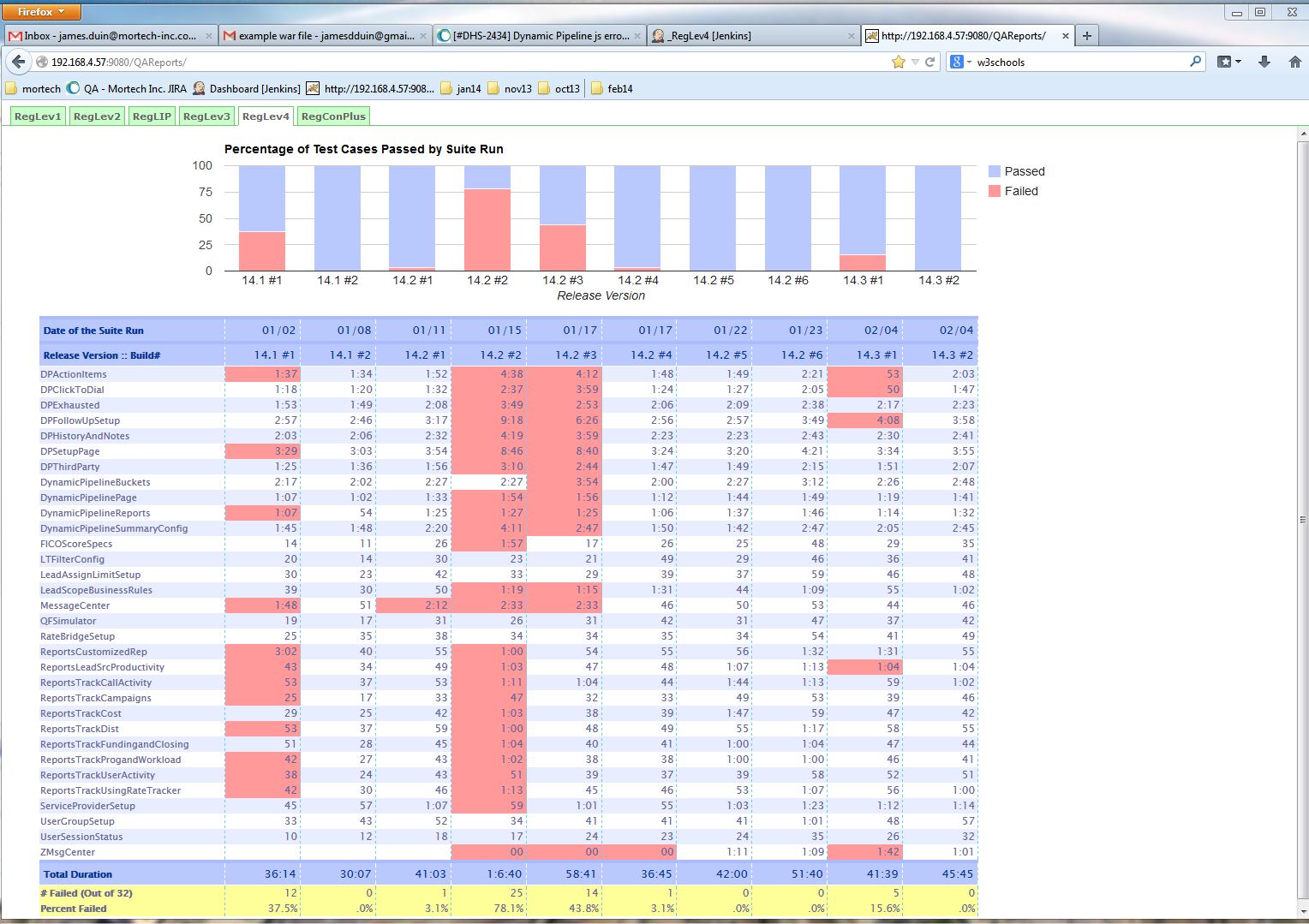


Figure 4: Report pages that show the results of regression testing from successive releases.

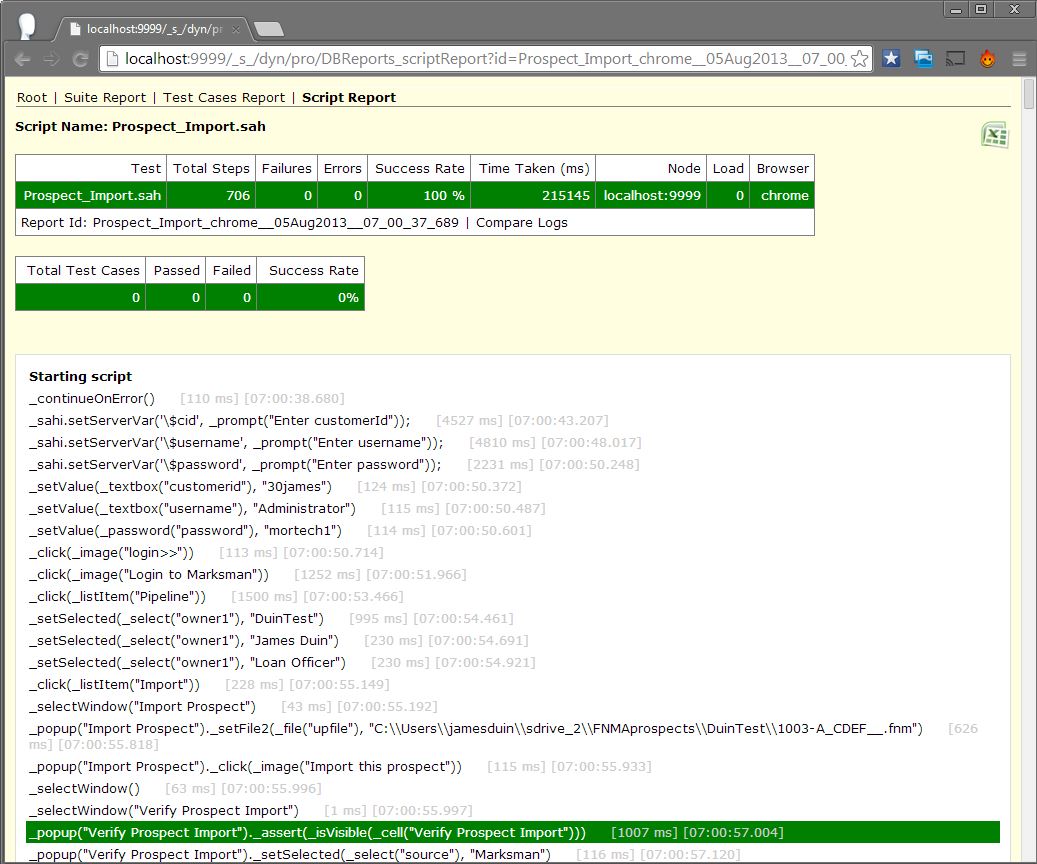


Figure 5: Sahi generated html log for an individual test case showing the step by step execution of the test.

**5.3 Screen Objects and Actions**

* **Figure 1**
  + Drop down list at the top of the page will contain all of the scripting languages available to the user
  + Save button will call the saveTestCase() function which will grab the script and the script name and add it to the database
  + The list at the bottom of the page will auto fill with all of the test scripts in the database when the page loads, using the getTestCases() function
  + When an existing test case is selected, the information for the test case will be retrieved from the database using the getTestCase(TestCaseID) function and it will fill the information in at the top accordingly
* **Figure 2**
  + Save button will call saveTestSuite() function which will grab all of the test cases from the table and add the test suite to the database.
  + The list at the bottom of the page will auto fill with all of the test scripts in the database when the page loads, using the getTestCases() function
  + When an existing test suite is selected, the information for the test suite will be retrieved from the database using the getTestSuite(TestSuiteID) function and it will fill the information in at the top accordingly
* **Figure 3**
  + Test Suite drop down list will populate on page load with all of the test cases from the database, retrieved using the getTestCases() function
  + Save button will retrieve all of the information from the page, verify that it is all there, and then save it as a test schedule using the saveTestSchedule() function
  + The list at the bottom of the page will auto fill with all of the test schedulesin the database when the page loads, using the getTestSchedules() function
  + When an existing test schedule is selected, the information for the schedule will be retrieved from the database using the getTestSchedule(TestScheduleID) function and it will fill the information in at the top accordingly
* **Figure 4**
  + Data table will automatically populate on page load with test case results from the database using the getTestCaseResults() function
  + Each individual test result will be a button that opens up the html page with more detailed analysis in figure 5

**6. REQUIREMENTS TRACEABILITY**

REQ-1.1: The system shall allow a user to upload a single script in Sahi code via a Scripts page, this script will be saved and should be accessible whenever navigating to the Scripts page.

SUPPORT: The page to upload code for a test case should be allow the user to upload and save a single script. It does not need to allow the user to save and access multiple scripts at this point it will load the one saved script at this time. The page to Group the test suites does need to exist at this point.

REQ-1.2: The system shall allow a user to save a test schedule for the single Sahi script on IE9/Win7.

SUPPORT: The page to Schedule test runs will be partially complete, it will all the user to specify a schedule to run the one saved script. It does not need to all the user to select between various test suites at this point. It will only allow the user to select one VM and browser combination. On the back end, only on VM (Win7) and only one Browser (IE9) need to be installed on the pc. It will be required however, for Jenkins and Sahi on the PC to execute tests on the VM, and for Jenkins to read and update test case schedule configuration and test case code from the database. Jenkins should have the ability to poll the database and update the necessary Jenkins property configuration files when the user saves a change in the database via the webapp.

REQ-1.3: The system shall execute the test on the desired test schedule and post the test results to the report page.

SUPPORT: Jenkins should be able to store the test results in the database after the test has been executed. The reports page on the suite level will be partially completed, it should show the results of the single test case run on the suite level reports page.

REQ-1.4: The system shall allow the user to click on the a test result from the reports page and access the Sahi generated step by step html log for that test.

SUPPORT: Jenkins should be able to store the test case logs in the database after the test has finished. The test case level reports page should be loaded with these test case logs when the user clicks on the test case from the suite report page.

REQ-2.1: The system shall allow a user to upload and save multiple scripts in Sahi code on the Scripts page.

SUPPORT: The Scripts page will have full functionality including the ability to save multiple scripts.

REQ-2.2: The system shall allow a user to access the saved script code by selecting an existing script from on the scripts page and loading the code for that script onto the page.

SUPPORT: The Scripts page will have full functionality including the ability to save multiple scripts.

REQ-2.3: The system shall allow a user to group the existing saved scripts into test suites, saving the test suites on a test suite page.

SUPPORT: The test suite page will be fully functional allowing the user to group test cases into test suites.

REQ-2.4: The system shall allow a user to save multiple test schedules for a test suite on one of the many supported browser/os combinations. These combinations including: IE8,Chrome,Firefox on XP; IE9,10,11, Chrome, Firefox on Win7; IE10,11, Chrome, Firefox on Win8.1.

SUPPORT: The test schedule page will be fully functional at this point, allowing users to choose between multiple test suites and multiple browser/os combinations for a test schedule. On the back end, multiple browsers and VMs will have to be setup and Sahi on the PC will have to be able to execute the test cases on the VMs. The Sahi instance on the VMs will have to be able to open the correct browser and execute the test cases.

REQ-2.5: The system shall prevent the user from saving a test schedule within 30 minutes (in the future) of the current time.

SUPPORT: Input validation will have to be added to the test schedule page to prevent the user from saving test schedules within 30 minutes of the current time to give processes in the back end time to update so that the system can guarantee the user that the test schedule will always be executed correctly.

REQ-2.6: The system shall shall execute the test suite on the desired browser/os combination and post the result back to the web app.

SUPPORT: Jenkins will have to be able to update the database with suite level results.

REQ-2.7: The system shall allow the user to access a report page showing the last 10 runs for a given test suite.

SUPPORT: The suite report page will be fully functional at this point. It will allow the user to select to show suite level results for any of the test suites that have been executed.

REQ-2.8: The system shall detect when an overlap in test runs has occurred and display a warning detailing the suites in conflict on the web app.

SUPPORT: The webapp will have to check for overlap in test suite run times. This only applys to test suites that have been schedule for the same VM and only when the second test suite is schedule to soon after the first test suite. It will be able to check for this system error by accessing the test suite start times and checking that they match the test schedule configuration for the suite. The back end handles this error by waiting to execute the second test suite until the first test suite has finished. The error is that the second test suite does not get executed on the specified schedule because it was delayed.

REQ-3.1: The system shall execute a test schedule on Safari/OS X.

SUPPORT: The back end should have a VM for OS X and with Safari installed.

REQ-3.2: The system will allow multiple QA teams to login to the web app and only access configurations saved on that account.

SUPPORT: The webapp will be expanded to include account logins, the backend will be expanded to have a Jenkins and Sahi instance on the PC, and duplicate VMs for each user.

REQ-3.3: The system shall allow a user to upload and save multiple scripts in Selenium code on the Scripts page.

SUPPORT: Selenium will have to be installed and configured on the PC and on all of the VMs, Jenkins will have to be configured to start Selenium tests. The webapp will not have to be changed extensively, any input validation to check that the uploaded code was Sahi code will have to be expanded allow for Selenium code.