SE 3XA3: Test Plan
Spann

Team 8
Christopher Stokes — stokescd
Varun Hooda — hoodav

December 8, 2016

Contents

List of Tables

Table 1: **Revision History**

Date	Version	Notes
Oct. 26, 2016	Christopher, Varun	Initial test plan
Oct. 30, 2016	Varun	Moved to new template
Oct. 31, 2016	Christopher, Varun	Submission for TP-Rev0
Dec. 06, 2016	Christopher, Varun	Final application test cases and revisions

1 General Information

1.1 Purpose

The purpose of this document is to outline the testing methodologies that will be used to test the Spann Web IDE application to ensure the application functions are specified in the software requirements specification document and to reveal possible bugs in the application.

1.2 Scope

The scope of the testing will be the front end JavaScript UI code and the back end C# code and SQL queries, as well as the API responses from the server.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
IDE	Application Development Environment
API	Application Programming Interface
UI	User Interface
C#	C Sharp, A object oriented programming language
SQL	Standard Query Language
CSS	Cascading Style Sheets
LESS	A styling language that transpiles into CSS
NUnit	Unit testing framework for C#
NodeJS	JavaScript runtime based on the V8 engine
Postman	A API testing web application for Chrome
HTML	Hyper Text Markup Language
DM	Domain Models
DTO	Domain Transfer Object
IronPython	A Python runtime for .NET
RequireJS	A NodeJS module for JavaScript dependency handling
AMD	Asynchronous module definition

1.4 Overview of Document

This document will outline the testing methodologies and various test plans that the development team will incorporate and use to test the application.

2 Plan

2.1 Software Description

The software, for which the test plan is being written, is an online, web-based, IDE. The final application will be similar to a desktop IDE that many developers are familiar with. The application will have two parts, a front end that will be written in JavaScript and LESS (transpiled into CSS), and a back end server written in C# (which uses a SQL server for persistent data storage).

2.2 Test Team

The test team will be made up of Christopher Stokes and Varun Hooda.

2.3 Automated Testing Approach

2.4 Testing Tools

For the server NUint will be used for unit testing the C# code. For the front end we will be using jasmine (a NodeJS module) for unit testing the JavaScript UI components. For the server API we will be using postman (a chrome web application).

For code coverage, visual studio's built in functionality will be used.

2.5 Testing Schedule

See Gantt Chart

3 System Test Description

3.1 Tests for Functional Requirements

Front End Tests

1. SHA512

   Type: Functional, Dynamic

   Initial State: Function, has no state.

   Input: UI data objects with a string to be hashed.

   Output: UI response objects with hashed string.

   Description: This test will ensure the SHA512 module return the correct hash for a corresponding string.

   How test will be performed: The test will be written and tested using Jasmine. The expected results will be precomputed using a third party application that hashes a string using SHA512.

2. Encryption utils

   Type: Functional, Dynamic

   Initial State: Function, has no state.

   Input: UI data objects of a string and a salt.

   Output: UI response objects with the correct hash

   Description: This test will ensure the encryption module performs the hashing correctly given a string and a salt.

   How test will be performed: The test will be written and tested using Jasmine. The expected results will be precomputed using a third party application that hashes a string using SHA512 along with a given salt.

3. Login Screen

   Type: Functional, Dynamic

   Initial State: Function, has no state.

Input: Set of correct user credentials and a set of incorrect user credentials.

Output: Successful authentication with correct credentials and no authentication with incorrect credentials.

Description: This test will be used to check if the login/user authentication implementation is functioning correctly and allows user's to access the application when they supply the correct credentials and does not allow users when they fail to supply the correct credentials.

How test will be performed: This test will be written and tested using Jasmine.

3.1.1 Server API Tests

1. Invalid Requests

   Type: Functional, Dynamic

   Initial State: Function, has no state.

   Input: Invalid user authentication request, invalid file/project request.

   Output: Response from server indicating invalid request has been made.

   Description: This test will ensure the system is robust and does not break if an invalid request is made. An example of an invalid request would be a non-existent user, non-existent file or project. Since the web application does not use multiple html pages, there is no need to test invalid urls.

   How test will be performed: This test will be carried out using the Postman application and a test file written for the Postman application.

3.1.2 Back End C# Server

1. SQL Queries

   Type: Functional, Dynamic

   Initial State: Function, has no state.

   Input: Database request object.

Output: SQL query corresponding to the request.

Description: This test will ensure the server generates the correct database queries once it has received a request from the front end via a web socket.

How test will be performed: This test will be performed using NUint and a test case written in C#.

2. Python File Domain Model Tests

   Type: Functional, Dynamic

   Initial State: Function, has no state.

   Input: Server DTO

   Output: Correct formatted DM with correct metadata.

   Desceription: Test the base logic that all Python File DMs extend from.

   How test will be performed: This test will be performed using NUint and a test case written in C#.

3.2 Tests for Nonfunctional Requirements

3.2.1 Front End JavaScript UI

Look and Feel

1. UI visual inspection

   Type: Manual

   Initial State: Application running inside browser.

   Input: Mouse inputs for navigating the application and keyboard for any textual input for the purposes of seeing text/code.

   Output: Visual output of the application rendered in the browser.

   Description: The purpose of this test will be to manually to inspect the application to discover visual bug, artifacts, and any other UI imperfections.

How test will be performed: This type of testing is only possible manually, since it is simply not possible to automatically inspect the UI using a computer.

2. UI performance

   Type: Manual

   Initial State: Application running inside browser.

   Input: User input using mouse clicks and keyboard input.

   Output: Qualitative observation of the performance of the UI components of the system.

   Description: The purpose of this test is to ensure the application's UI is fast and responsive. This will help identify any UI performance issues.

   How test will be performed: This will also be done manually since it would be very difficult to programmatically test.

3.2.2 API /Back End Tests

Stress Test

1. API/Server Stress

   Type: Functional, Dynamic

   Initial State: Function, has no state.

   Input: String of python code from multiple concurrent connections.

   Output: Server's response to each requests. The expected response time should be less than RESPONSE_TIME seconds.

   Description: To test whether the sever is able to process a large number of concurrent requests.

   How test will be performed: This will be written and run using Jasmine, which would make requests to the back end server.

4 Tests for Proof of Concept

4.1 Front end JavaScript UI

1. UI performance

   Type: Manual

   Initial State: Application running inside browser.

   Input: User input using mouse clicks and keyboard input.

   Output: Qualitative observation of the performance of the UI components of the system.

   Description: This test will help the developers see if the current design of the proof of concept provides the performance required in the final application.

   How test will be performed: This testing will be manually since it is difficult to automate it.

4.2 Back end C# server

1. Python Execution

   Type: Functional, Dynamic

   Initial State: Function, has no state.

   Input: String of valid python code and a string of invalid python code.

   Output: Expected result of the valid code and expected exceptions of the invalid code.

   Description: The purpose of this test is to ensure the server is able to receive python code and execute the code using IronPython on the server system.

   How test will be performed: This test will be performed using NUint and C# test cases.

5 Comparison to Existing Implementation

The original project, repl.it, on which this project is based supports multiple languages and a mature and well maintained project. Our application, on the other hand, focuses on the python programming language specifically. So the scope of our project is much narrower than the original.

A narrower scope means we are also able to test our application more thoroughly. The original project seems to focus more on test higher level features, specifically, whether the front and back end have a connection, data is being sent back and forth. Our project will go more into the specific features and perform more extensive tests to verify the functionality and the non-functional aspects of the application.

6 Unit Testing Plan

6.1 Unit testing of internal functions

6.2 C#

In order to test the internal functions and logic of the C# server, the reflective capabilities of C# will be utilised. NUnit utilises reflection to provide access to internal classes not normally accessible, this is necessary as by default C# classes are private to all but the namespace. To extend this capability reflection will be used to access the private member properties, and functions allowing direct calls to these elements and direct access to their return values. This will greatly reduce the amount of mocking required to execute the tests, and overall providing simpler tests and more rugged and reliable test which are not easily broken.

6.3 JavaScript

In order to test the JavaScript modules, multiple tests patterns will need to be used due to the nature of the code.

The first type applies to all the custom UI components of the Spann UI engine, where the private members are located directly in the object returned from the module. The internal parts of the components are located in an object named _private on the module objects. This is a consequence

of patterns used to allow inheritance in JavaScript and lack of a dependency manager for the UI components, increasing performance.

In comparison, the remaining code in the Spann client uses the revealing module pattern as well as AMD with RequireJS. This combination means a simple management of dependencies but as a consequence, there is no way to access the components not revealed from the module. Therefore, all modules will reveal a method named _getInternals which will return all the internal methods allowing them to be tested.

6.4 Unit testing of output files

There are no files output by the application.

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

RESPONSE_TIME 10 seconds

7.2 Usability Survey Questions

1. How would you rate the usability of the application's navigation on a scale of 1-5?
2. How would you rate the usability of the application's editor on a scale of 1-5?
3. How would you rate the usability of the application's console on a scale of 1-5?
4. How would you rate the responsiveness of the application on a scale of 1-5, with 5 begin the most responsive?
5. How would you rate the usability of the application's project management features on a scale of 1-5?
6. How would you rate the ease of use of the application on a scale of 1-5?
7. How would you rate the usability of the application for small pieces of code or small project one a scale of 1-5?
8. How would you rate the usability of the application for medium sized pieces of code or medium sized project one a scale of 1-5?
9. How would you rate the usability of the application for large sized pieces of code or large sized project one a scale of 1-5?