*University of Phoenix Material*

**Software Test Plan**

Mojo Banking Solutions, Inc..

The project in and of itself is that of an online based bank which provides various services to its customers such as accounts setup, deposits, credit debit and travel cards. It will provide loans, insurance, investments, and tax services. This service will provide the scalability, availability, security, and manageability one would expect from an online based banking service both through its web-based interface as well as mobile application-based interface.

# Features To Be Tested / Not To Be Tested

|  |  |
| --- | --- |
| **Features to be tested** | **Features to not be tested** |
| View account summary | Interface customization: Not a mission critical component, if there are issues with this it can be addressed through patching. |
| View/Print statements |  |
| Set up online payments. |  |
| Reorder cards (debit, credit, travel) |  |
| Transfer funds between accounts (i.e. checking to savings or visa versa) |  |
| Access other accounts (CD's, IRA's, etc.) |  |
| Set up different accounts (savings, checking, CD's, IRA's, etc.) |  |
| Update personal information (addresses and phone numbers for example |  |
| Set/Change password |  |
| Biometric identification (mobile application) |  |
| View history |  |
| Set/change user id |  |
| Enable/disable account alerts (e-mail/SMS). |  |
| Set up customed alerts (account balance for example) |  |
| Withdrawn and deposit function accuracy (make sure functions are executing properly. |  |
| User ID retrieval |  |

# Testing Pass / Fail Criteria

* Suspension criteria, this would be any situation which impeded the ability to continue testing or value in preforming testing that leads to the suspension of testing activities.
* Resumption criteria, this being a continuation of testing after the issues causing the suspension criteria to have been corrected and patched accordingly.
* Approval criteria, this will be a part of the application being tested that has performed its desired functionality
* Unit testing, each unit should be tested under its designed function, if unit can not live up to its expectations a fail condition is to be issued and further adjustment to its source code and algorithm shall be made.
* Data leakage, test should be run to ensure the source code fallows its constraints as per the programming principal of encapsulation. This would mean one not having the capacity to interact with nor alter the data within the confines of the program. Fail condition would be one being able to access and/or alter the data from the program’s interfaces given.

# Testing Approach

Initial stage will be actual planning of the tests in their entirety. This will bring in the scope of the test itself, the risks associated with each test, and the effect of the development of the entire software package is determined, as well as the implications given the impact analysis of the application being tested.

The second stage is creation of scenarios to test various parts of the application. Primary testing should be done on critical systems especially any parts of the application dealing with finances and any interaction with internal or external systems. Functionality checks should be made on any units within the application that deal with any calculation functions especially those that pertain to financial information (example being account balances). Next should be the behavior of the code, an example being the use of CSS established keywords and how there should be a uniform response with their utilization within the webpage (i.e. H1 tags labeled as “red” in CSS should display as “red” when executed). Functionalities should be tested on page functions as well (input and output, delete functions and so on for example).

Next should be a constructed environment to perform an execute tests in. This should be isolated from any other systems to eliminate any possible outside interference until the units being tested are passed and ready for integration. Generic falsified data should be generated in order for it to be utilized in the testing process. Each unit should be subjected to the appropriate data (mock financial data for financial parts of the application for example). Any failures of a unit or part of the system should be identified and corrected before the integration testing.

Next the scenarios shall be running the scenarios. Adequate time should be scheduled to elevate and time pressure for the testers. The fully planned scenarios should be executed in several iterations until the part of application itself is behaving as expected.

Finally, all results should be recorded and documentation. This report should be simplified in presentation containing screenshots or video documentation of any encountered errors as well as documentation of how the error was addressed and resolved (if possible).

Testing methods utilized shall be unit testing, integration testing, system testing, and acceptance testing. Unit testing itself will be the handling of partitions of the application itself, the application in its entirety shall be broken up into smaller unites to be tested (financial partitions tested together for example). The core idea behind unit testing is to test the smallest piece of code, in this instance it would be the various classes derived from C++. From here automated tools can ensure the code in question is capable of being automated and easily operated repeatedly in succession. Next would be integration testing, this would be the combining of different units into a conglomeration of blocks resembling functions of the overall system. In system testing (or end-to-end testing) would be a full test of the application. After all integrated blocks are integrated themselves into the complete application system testing itself would be applied to the whole of the application to ensure all its functions and devised tools built into the application are functionally working as designed. Finally, acceptance testing is the testing of the application form the end-user perspective. This is to ensure the built-in tools allowing the user to interact with the application are in fact alloying the appropriate conditions and actions.

# Testing Cases

|  |  |
| --- | --- |
| **Test Case** | **Test Scenario** |
| 1 | Check system response on entering valid username/password |
| 2 | Check system response on entry of invalid username/password. |
| 3 | Check system response on entry of no entry of username or password. |
| 4 | Check system response on the update of user data. |
| 5 | Check system response of account information access handling tools such as viewing and adding to one's balance for example. |
| 6 | Check system response for computational functions. |
| 7 | Check system response for database update. |
| 8 | Check system response for instanciation creation from functions (accounts). |
| 9 | Check system response for switching to different user functions (Taxes, insurance, etc.) |
| 10 | Check system response for notifications (e-mail & SMS) |
| 11 | Check system response for custom notifications (Account balance for example). |
| 12 | Check system response for duplicate account creation (duplicate username creation for example) |

# Testing Materials (Hardware/Software Requirements)

Materials that are needed for the testing of an application being the actual environment the application. This would be the physical servers being used as well as an array of generated virtual servers the application would depend on. A functioning Datawarehouse would be best to test the retrieval and placement of data being entered into the system itself through its constructed interfaces. An Interactive Development Environment would have to be loaded onto a host machine and configured to work build C++ code structures into for the beginning of said project. Microsoft’s Visual Studio Code is easy to set up and easy to use for such a venture as an example. While as the front end application of the system is ran using JavaScript, programs such as MochaJS can be used in building that respective code, just as with Visual Studio it would simply need a host machine to operate the code on and trouble shoot it. Terminals would be needed to connect to the cloud infrastructure as laid out to access systems. While as the system itself would exist in a cloud environment, one would still need an office to operate out of at least while the application is being tested, this would allow easier collaboration for the build team. One would also need an entire testing tool such as Clicqa Bank Testing Services in order to test the entirety of the banking application.

# Testing Schedule

| Testing Activity | Duration | Resource | Comments |
| --- | --- | --- | --- |
| Test Plan Creation | 5 days | Test Manager | Establishment of test plan protocols. |
| Test Specification Creation | 10 days | Test Leads | Establishment of testing specifications and the criteria the testing team will have to follow. |
| Test Specification Team Review | 5 days | Project Team | Collaboration of the testing team in order to review testing guidelines, policies, and factors. |
| Component Testing | 20 days | Component Testers | Testing of each constructed segment of the application. |
| Integration Testing | 20 days | Component and System Testers | Testing of each constructed after it is integrated with other segments. |
| System Testing | 15 days | System Testers | Testing of the entire system application after all parts have been integrated. |
| Performance Testing | 5 days | System Testers | Putting the system under different constraints to ensure it can handle workload and live up to established expectations. |
| Use Case Validation | 10 days | System Testers | Testing the system from the customers perspective to ensure all desired functionalities are met. |
| Alpha Testing | 5 days | Product Managers / Analysts | Testing the system in house to ensure each system behaves as desired for the companies side of the application as well as the end-user or customers side of the application. |
| Beta Testing / Pilot Program | 20 days | Pilot Program End-Users | Testing the system just before launch in its cloud environment before allowing outside user (customer) access. |

# Risks and Contingencies Matrix

| Risk | Probability | Risk Type | Owner | Contingencies / Mitigation Approach |
| --- | --- | --- | --- | --- |
| Do not have enough skilled workers to test components as they are ready for testing. | 25% | Project Resources | Testing Manager | Testing schedule will be adjusted based upon available resources. |
| Testing team member turnover | 10% | Project Resources | Testing Manager | Adjust testing schedules. Make sure testing team members are cross-trained on testing techniques in case a team member leaves the organization. |
| Having enough infrastructure to support rollout demands | 10% | Project Resources | Project Supervisor | Adequately building needed infrastructure to successfully test the application under stressed conditions. |
| Scope Creep | 25% | Project Resources | Testing Manager | Ensuring application is tested within the established parameters needed. |
| Scheduling deviation | 15% | Project Resources | Testing Manager | Ensuring proper testing protocols are called out in their scheduled allotted time. |