Personal Financial Manager User’s Guide and Test Plan

D. Weissenberger, J. Doan, M. Maibach, Z. Smith

University of Maryland Global College – CMSC 495 – 24416

Sunday, April 5, 2020

**PREFACE**

This User’s Guide and Test Plan outlines the developmental process of the Personal Financial Manager application throughout the length of CMSC 495, class number 24416. PMP development and planning will be carried out by members of Group C.

**Personal Financial Manager User’s Guide and Test Plan**

**TABLE OF CONTENTS**

[SECTION 1. OVERVIEW 5](#_Toc37013629)

[1.1 PURPOSE 5](#_Toc37013630)

[1.2 AUDIENCE 5](#_Toc37013631)

[SECTION 2. TEST STRATEGY 6](#_Toc37013632)

[2.1 SCOPE 6](#_Toc37013633)

[*2.1.1* *Out of Scope* 6](#_Toc37013634)

[2.2 ASSUMPTIONS 7](#_Toc37013635)

[2.3 SCHEDULES 7](#_Toc37013636)

[*2.3.1* *Test Scenario Prep* 7](#_Toc37013637)

[*2.3.2* *Test Documentation* 7](#_Toc37013638)

[*2.3.3* *Test Execution* 8](#_Toc37013639)

[*2.3.4* *Test Cycles* 8](#_Toc37013640)

[2.4 ROLES AND RESPONSIBILITIES 9](#_Toc37013641)

[*2.4.1* *Team Members* 9](#_Toc37013642)

[*2.4.2* *Tasks Delegated and Acknowledged* 9](#_Toc37013643)

[*2.4.3* *Module Leads and Contact Info* 10](#_Toc37013644)

[2.5 DELIVERABLES 10](#_Toc37013645)

[2.5.1 Test Artifacts by Time Frame 10](#_Toc37013646)

[2.6 ENVIRONMENT 11](#_Toc37013647)

[*2.6.1* *Environment Requirements* 11](#_Toc37013648)

[*2.6.2* *Environment Testing Responsibilities* 11](#_Toc37013649)

[*2.6.3* *Contingency Plan* 11](#_Toc37013650)

[2.7 TOOLS 11](#_Toc37013651)

[*2.7.1* *Team Access* 11](#_Toc37013652)

[*2.7.2* *Usage and Responsibilities* 12](#_Toc37013653)

[2.8 DEFECT MANAGEMENT 12](#_Toc37013654)

[*2.8.1* *Defect Report Access* 12](#_Toc37013655)

[*2.8.2* *Usage and Responsibilities* 12](#_Toc37013656)

[2.9 RISK 13](#_Toc37013657)

[*2.7.1* *Risk Analysis* 13](#_Toc37013658)

[*2.7.2* *Risk Mitigation* 14](#_Toc37013659)

[2.10 EXIT CONDITIONS 15](#_Toc37013660)

[SECTION 3. REFERENCES 16](#_Toc37013661)

# SECTION 1. OVERVIEW

## 1.1 PURPOSE

This personal financial management tool will provide users with the ability to manage their accounts, track and record basic financial transactions, and produce reports to view aspects of their finances.

## 1.2 AUDIENCE

This tool is targeted towards individuals seeking a program to manage basic, personal financial transactions.

# SECTION 2. TEST STRATEGY

## 2.1 SCOPE

There are several items that we wish to ensure are properly tested as we are developing our finance application.

* Testing of all input/output functions relating to the accessing and modification of data through our database. This means unit testing for all data access functions, as well as manual testing, as the application is developed.
* Manual testing of all GUI related features, since testing of GUI related items can be difficult. Extensive manual testing can be completed as the interface is developed.
* Integration testing of functions that either process a significant amount of data, or perform a “critical” function, such as saving data to the database, or updating/modifying values in response to a GUI interaction.

### *2.1.1 Out of Scope*

Items that are considered OOS (Out of Scope)

* Testing of application on non-supported platforms, including Linux and other flavors of distros.
* Items that were not originally considered in the initial scope of the application, this includes features that were touted as possible “TO-DOs” in the course of the development of the application.
* Testing of any deployment of database-related items, including any kind of auto-deployment process, or the initial set up process around the DBO area of the application.

## 2.2 ASSUMPTIONS

We must assume several items to be true before we can proceed with effective testing of the application at large.

* The tester is running the application on a supported platform
* The database has been correctly configured and is accessible by the application itself.
* All unit tests have been written and proven to be accurate, manually testing of what the unit test aims to test should be completed before committing the said test.

## 2.3 SCHEDULES

### *2.3.1 Test Scenario Prep*

To prepare for testing of the application, a development environment must be available to the tester. This includes a development environment, including either local or remote access to the database. The user must also be on a branch of the application denoted as “QA”, ensuring that we are not testing features that are still being developed.

### *2.3.2 Test Documentation*

Documentation of the testing should take place each time the application is ready to be tested, this usually means after a merge from development to QA. The tester should have a list of features that they need to complete testing based on the features that were added during the application within the development branch. The tester should also have a regressive list of all features that were added previously to the application to support their efforts in testing for regressive bugs (Bugs that are new but appear in older features of the application that were previously tested). The tester should lastly have access to the full list of JUnit tests that were written for the application in their IDE.

### *2.3.3 Test Execution*

The tester should begin their testing by first executing all of the unit tests that are currently available and working in the application. They should document the result of all of the unit tests either by writing down if they were successful or not, or take a screenshot of all unit test results. All unit tests that have failed should be written down for future reference in debugging why those tests failed in the future. After the completion of all of the Unit tests, the tester then can move onto manually testing all of the features that currently do not have a unit test or integration test written for it. The user should follow a strict format for recording what happens during their testing of a particular feature. The tester should include the following items into their written results.

* What was the expected outcome?
* What was the actual outcome?
* If the outcome was negative, can it be replicated easily?
* What were the exact steps to replicate the issue?
* What are your platform specifications?
* What is the current version of the application you are testing on?

All of the results, both written and automated, should coalesce into a single document, available to all of the developers who originally created the features for debugging and fixing.

### *2.3.4 Test Cycles*

For automated items, such as unit tests and integration tests, these should be run in a set of 3 times, to ensure that the test is correct each time.

For manually tested features, each feature should be tested 3 times as well, ensuring that even with potentially improper usage of the application, it will still succeed at what it was designed to do. (Either through successfully completing what it needed to do or by gracefully handling errors and informing the user.)

## 2.4 ROLES AND RESPONSIBILITIES

All the conditions that need to hold true for us to be able to proceed successfully

### *2.4.1 Team Members*

* Justin Doan
* Marcus Maibach
* Zachary Smith
* David Weissenberger

### *2.4.2 Tasks Delegated and Acknowledged*

Tasks for a given week may not be acknowledged before the start of that week, 12am Monday EST. The file **Timeline.xlsx** has been made available to all team members using versioning tools.

(TBA = To Be Assigned)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Week#** | **Justin** | | **Marcus** | | **Zach** | | **David** | |
|  | **Task and Acknowledgement** | | **..** | | **..** | | **..** | |
| 2 | *Task:* Access Teams & Github | **X** | *Task:* Access Teams & Github | **X** | *Task:* Access Teams & Github | **X** | *Task:* Access Teams & Github | **X** |
| 3 | *Task:* Tools (Github) | **X** | *Task:* Desktop GUI research | **X** | *Task:* Database research | **X** | *Task:* Desktop GUI research | **X** |
| *Task:* Test Plan | **X** | *Task:* Test Plan | **X** | *Task:* Test Plan | **X** | *Task:* Test Plan | **X** |
| 4 | *Task:* TBA |  | *Task:* TBA |  | *Task:* Set up SQL DB |  | *Task:* UML Diagram |  |
| 5 |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |

### *2.4.3 Module Leads and Contact Info*

The Module Lead role is responsible for the design, testing plan (not implementing), addressing other modules’ integration requirements, and tasks will be created to accomplish this.

* Desktop UI — David
* Database — Zach [zsmith19@student.umuc.edu](mailto:zsmith19@student.umuc.edu)
* Authentication — TBA

## 2.5 DELIVERABLES

Quality Assurance testing documentation should be created at the end of each major feature development. When a feature is successfully moved from development into ready for QA, we will begin our testing, and after testing, we should have a completed QA document ready for review by all members of the team for further analysis.

### 2.5.1 Test Artifacts by Time Frame

Each completed document should contain the following information.

* A list of all new features that have been tested.
* A list of all features that were regressively tested.
* A screenshot or list of all unit tests that were new and their respective results.
* A screenshot or list of all unit tests that were not new and their respective results compared to their previous results from the last testing document that was created.
* A list of all errors/issues/concerns that were discovered by the tester.

## 2.6 ENVIRONMENT

All the conditions that need to hold true for us to be able to proceed successfully

### *2.6.1 Environment Requirements*

* A computer capable of running Java version \_\_\_\_\_\_\_ .
* An SQL server that is accessible from where the user is located, either via standard HTTP/HTTPS connections, TCP/UDP connections, or via access over a VPN.

### *2.6.2 Environment Testing Responsibilities*

A developer should work with the tester to ensure that their local development set up meets the standards that are required of the application, and that the database is remotely accessible from their workstation. The testers’ environment should be as close to the expected actual working environment as possible.

### *2.6.3 Contingency Plan*

Investigate backup database solutions that meet accessibility and security requirements (PCI Security Standards Council, 2020).

## 2.7 TOOLS

All the conditions that need to hold true for us to be able to proceed successfully

### *2.7.1 Team Access*

All team members will be able to access testing plans, documentation, and results. These items will be stored in the Teams repository. No tools beyond the application itself and the test tracking will be used.

### *2.7.2 Usage and Responsibilities*

Users can access the teams application through logging in with their school email address into the Teams website. All documents can then be accessed within the application.

## 2.8 DEFECT MANAGEMENT

All the conditions that need to hold true for us to be able to proceed successfully

### *2.8.1 Defect Report Access*

Access to all defects that are reported or discovered by either a user/tester should be available either through the issues tracker/report available within Github or from the testing document that is created by the tester during routine testing of new application features.

### *2.8.2 Usage and Responsibilities*

All defects that have been reported should be assigned to a developer for review, they should attempt to replicate the defect themselves, and begin debugging the application to figure out what/how the defect occurred. An additional document should be created by the developer if the defect was discovered by a tester, or if the defect was reported via the issues tracker on Github, their documentation of the defect should be continued to be reported within the original issue that was created, barring the inclusion of any information that is deemed not allowed to be reported outside of the development team. Once the defect has been repaired or mitigated, the developer in charge of the defect should report back to QA with their results and have their fix moved into ready for QA. QA can then retest the defect. Once the defect has been repaired and has successfully passed QA, it can be released to the public. The developer in charge of the defect should also report back to the user (if the issue was reported by a user), to ask them to see if the new fix has fixed the defect that was discovered. The developer then can close the issue if the defect was correctly fixed, or reopen another investigation if the defect was not fixed, or resulted in the creation of another defect.

## 2.9 RISK

### *2.7.1 Risk Analysis*

***2.7.1.1 Risk Analysis of discovered issues.***

If a defect/bug/issue is discovered either by QA or is reported by a user, a risk analysis should take place in order to prioritize the severity of the bug. Several, or in our case, all of the developers should plan a meeting after QA has gone through a new set of features. All developers can then go one by one through each defect/issue that was discovered by the tester, and delve into if the defect is severe or not. All developers will follow the OWASP Risk Rating Methodology, an open-source risk rating methodology that has been proven to be effective at rating the risk of potential bugs/defects. By pointing to the risk of all the defects, priority can then take precedence over which defects should be fixed first, from severe to non-severe.

***2.7.1.2 Risk Analysis ahead of time***

Similar to how we plan and prepare our reaction to discovered defects and bugs, we can also plan for known risks that come with software development in general, such as access to databases. When a weak point is noticed by a developer or any member of the team, a process should start immediately similar to how we handle discovered issues. If we know ahead of time that access to the database is a critical issue that we need to ensure we protect, we can begin a Risk analysis of what could potentially happen if unrestricted access was somehow gained by an attacker. Similar to how we rate the defects with the OWASP risk rating method, we can do the same for issues we know of ahead of time, allowing us to go deep into analyzing the potential issues and begin planning around how to ensure that a particular part of our application is protected.

### *2.7.2 Risk Mitigation*

Similar to how we take Risk Analysis to potential issues we know ahead of time, we can plan and ensure to preemptively make those features as secure as possible. Starting at the planning stage of any new features, a Risk Mitigation meeting should take place, wherein every member of said meeting can discuss and talk about parts of the feature that are more susceptible to attacks and issues than others. We can then silo these potential issues ahead of time, and go through our risk analysis of them to determine if the potential issues that could arise are high enough to prompt preemptive action to secure that part of the feature. For example, we know our application will have access to a database, we can plan ahead of time to build in safeguards to the transmission and storage of sensitive customer information, and meet regulatory requirements (PCI Security Standards Council, 2020). We can also restrict database access to only verified client applications, and ensure the usage of encryption keys to encrypt traffic between the client application and the database.

## 2.10 EXIT CONDITIONS

Testing should continue as long as we release new versions of the application, however, once a version of an application has been released, no further testing of that application should take place for that particular version. Testing should, however, continue on new versions of the application that are currently being developed/tested.

Testing can also continue once we move the application to a point where only bug fixes are done instead of new feature development. Testing will only be done around the development of fixes to those bugs, but similar to new feature development, after the release of a bugfix version, no more testing will take place on that version.

Once the application has reached a point to where no further development is needed/required/or can be supported, we should issue an announcement about an EOL date to our application, in which after that date, there will be no further updates or versions, and that testing will officially end after that date since there is no new development taking place. Users should be warned that the application is no longer being supported and that no further bugfixes will take place, even for severe issues.

# SECTION 3. REFERENCES

1. PCI Security Standards Council. 2020. *Maintaining Payment Security.* Retrieved from: <https://www.pcisecuritystandards.org/pci_security/maintaining_payment_security>