***Scrumptious Finance***

***Testing and Refactoring***

Financial Web Application

February 28th, 2018

Version 1.1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[Project Lead Signature Block]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[Project Manager Signature Block]

**Revision Sheet**

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Brief Summary of Changes | Signed |
| Version 1.1 | February 28th 2018 | Created Unit Testing for Transactions | *Jordan E.* |

**Table of Contents**

[1 Unit Testing Transactions 4](#_Toc510040366)

[1.1 Summary 4](#_Toc510040367)

[1.2 Testing 4](#_Toc510040368)

[1.3 Refactoring 6](#_Toc510040369)

[2 Unit Testing Account Related Functions 6](#_Toc510040370)

[2.1 Summary 6](#_Toc510040371)

[2.2 Testing 6](#_Toc510040372)

[2.3 Refactoring 6](#_Toc510040373)

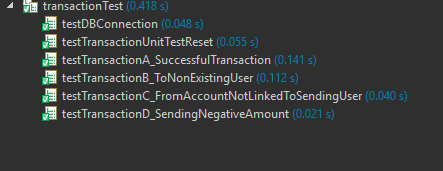
# 1 Unit Testing Transactions

## 1.1 Summary

The goal of testing transactions was to have extremely secure transactions while keeping fast transaction speeds. In order to complete these two main points, we developed a system where the data was received from the client, sanitized of any malicious characters, checked basic security points (1.2 Testing), and then the transaction was either confirmed or denied based on the security points. If the transaction was successful, then we would remove the balance from the sending account and add the balance to the receiving account. However, we changed this concept in the refactoring process.

## 1.2 Testing

The first task we completed when testing the Transaction was to make sure that the database connection was working properly during the tests. Next, we created a test for verifying that the data reset was working because we don’t want to leave test data on the database. After this, we tested a transaction with valid input to see how long it would take (~ 0.15s). Finally, we created a few tests to check that the proper error codes were returned for the matching errors. (Figure 1.1)



*(Figure 1.1)*

PHPUnit 7.0.1 by Sebastian Bergmann and contributors.

...... 6 / 6 (100%)

-----Transaction A-----

Sent Transaction('jordems', 12.12, "Money Owed", 36) in format(tousername,amount,reason,fromaccountid)

\*\*\*Expecting Successful Transaction\*\*\*

Transaction A's Request is Successful

--Verifying with Database's Account Data--

FromAccount Balance Before: $5000

FromAccount Balance After: $4987.88

Amount withdrew: $12.12

ToAccount Balance Before: $416.38

ToAccount Balance After: $428.5

Amount deposited: $12.12

-----Transaction A's Test is SUCCESSFUL!-----

-----Transaction B-----

Sent Transaction('NOtExistentUser', 12.12, "Money Owed", 36) in format(tousername,amount,reason,fromaccountid)

\*\*\*Expecting Failed Transaction as Username Doesn't Exist\*\*\*

Transaction B Failed, Receiving Username Doesn't exist or Doesn't have a main Account

FromAccount Balance Before: $5000

FromAccount Balance After: $5000

Amount withdrew: $0

-----Transaction B's Test is SUCCESSFUL!-----

-----Transaction C-----

Sent Transaction('jordems', 12.12, "Money Owed", -1) in format(tousername,amount,reason,fromaccountid)

\*\*\*Expecting Failed Transaction as withdrawing Financial Account Doesn't link to this user\*\*\*

Transaction C Failed, User Not Linked to Account

FromAccount Balance Before: $0

FromAccount Balance After: $0

Amount withdrew: $0

-----Transaction C's Test is SUCCESSFUL!-----

-----Transaction D-----

Sent Transaction('jordems', -12.12, "Money Owed", 36) in format(tousername,amount,reason,fromaccountid)

\*\*\*Expecting Faild Transaction, Error Message: Insufficient funds\*\*\*

Transaction D Failed, Insufficient Funds

-----Transaction D's Test is SUCCESSFUL!-----

Time: 544 ms, Memory: 8.00MB

OK (6 tests, 6 assertions)

## 1.3 Refactoring

While testing we came across a few small bugs that we have since fixes. But, one larger problem we had in testing was that if someone were to interrupt a transaction or the database crashed in the middle of a transaction then the database would have inconsistent data with incorrect results. So, we researched and found that there is something called SQL Transactions that allows you to disable auto-committing when dealing with databases. What this does is if there are any type of interruptions during a section of code, then it will automatically fall back to the previous version of the database. This completely solved the issue that we were having, and it also increased the speeds of our transactions by about 5%.

# 2 Unit Testing Account Related Functions

## 2.1 Summary

[Insert a summary of what these tests do and what they accomplish

## 2.2 Testing

[Insert what was done for testing, and screen shots of tests]

## 2.3 Refactoring

[Insert what changes resulted in the testing completed]