WEEK 7 TEAM NAME HONEY BADGER NAME MICHAEL GRIFFIN, DAVID CHAN

Role - TESTER

The Testers are the most important link between the technical vision of the TL and the market-requirement of the BA. They work closely with BAs to develop User Acceptance criteria and user-stories based test plan which ensure that quality is build into the process and thus the product. They enforce software quality and ensure the product is as bug-free and market-friendly as possible and document the test results

Revision History

| Week Number | Author | Description of changes |
|---|---|---|
| 1 - 4 | Eden & Matt | Added possible testing criteria and layout to this file. Testing credit/debit, login and account creation. |
| 5 - 7 | Michael & David | Added more testing criteria, found new issues with new features. Developed JUnit tests for existing features, and features that haven't been implemented. |
| | | |

*Comments:*

- *This page should be updated each week and the TAs will use it to determine the week's work done by the team-member in the particular role.*
- *For each of the items mentioned in the following pages. Be as brief as possible in your responses. A good rule of thumb is to keep each response within two paragraphs.*

*- This document represents your log of decisions. You are not bound to follow a decision blindly. You may change the decision if new aspects come to light which make your decision inappropriate. However, this may include repercussions like code rewrite etc. So choose wisely.*

*- You may delete the commented regions for your weekly turn-in submissions.*

1. Testing Strategy
    1. Test Objectives

        All tests will be well conducted in each module before performing a test with a combination of modules to ensure that errors are taken care of as early as possible.

    2. Scope and Level of Testing

        JUnit test will be implemented soon to streamline account interactions. Everything is being manually tested until sufficient tester cases have passed to implement automated mass test case testing.

    1. Known Issues

| Known Issues | Causes | Solutions | Results |
|---|---|---|---|
| Incorrect Login | Page does not exist | Create the page | Page created and users are allowed to login. |
| Incorrect Transactions | There is no transactions | Verify transactions with database | |
| Missing Backend | No server | Create server | Utilize Parse, server up and running. |
| Unable To Create Account | Missing infrastructure | Add “add account” feature and create account object in the database | Feature implemented create account now working |
| Debit and Credit not functioning | Debit feature no implemented | Implement feature and verify algorithm with JUnit testing. | Feature implemented debit and credit now working |
| Adding Account type failing | Missing infrastructure to handle account type | verify account types with database | |
| Login Function unstable | unknown | | |

| | | | |
|---|---|---|---|
| No scrolling on pages, have to close keyboard | Pages are static | Make them dynamic by considering keyboard as part of the page | |
| Enter causes newline at the end of the string | Values are terminated with a newline | Remove the newline or figure out a new way to terminate string | |
| UI functional but not appealing | Layouts are not arranged with great design in mind | Have a developer begin developing with UI in mind | |
| Negative number showing for account number | The way random number is generated sometimes results in a negative number | Don't allow the number for the account number to be negative | |
| Interest and penalty not implemented | Have not gotten that functionality implemented yet | Write the tests for that because we are doing test driven development, and also implement the functionality | |
| Negative balance occurs | Currently only shows an error message but still performs the debit | Do not allow debit to occur if more than balance | |

1.

1.

## 2. User Acceptance Scenarios

| BA Crteria | PM Criteria | Overall Criteria |
|---|---|---|
| User should be able to securely log in as a bank employee or account holder | Connection should be secure from login to logout | Create login page |

| | | |
|---|---|---|
| User should be able to log out and return to the log in screen | User must be able to login and logout | Create logout button |
| New customers should be able to open their first bank/user account with name, address, date of birth, email and account type. | Must be able to create accounts | Create server to handle accounts |
| Teller should be able to look up a customer's account to perform transactions for them. | Transaction lookup shuld exist | Server handles lookups |
| Account holder/teller should be able to add a new bank account. | New account should appear in database | Create adding feature for new account |
| User should be able to retrieve password by submitting their email and account number | Account details should exist in database for retrieval | Create backend with password retrieval |
| Account holder should be able to see all of their accounts | Account should appear in users list | Create listing feature |
| Account holder/teller should be able to disable and delete one of the customer's bank accounts. | Delete account but still exist on server for history purposes | Handle deletion feature |
| Notify user that account must have zero balance to be closed. | Create criteria to be checked before deletion | Handle zero balance to close case |
| Account holder/teller should be able to withdraw money from a customer's bank account IF withdrawal amount does not exceed the account balance. | Create criteria to be checked before withdrawals | Grant administration power to teller account |
| Check is emailed to account holder once they debit their account. | Backend to handle check creation | Email check to user with a generator |
| Notify user that they can't perform this action due to insufficient funds in source account | Create criteria to check funds | Criteria to check funds |
| Account holder/teller should be able to deposit money into one of customer's bank account. Account holder can do this through credit and check. Teller can do this through cash as well. | Teller account mode has administrative power to do more actions on user account. | Create account with more administrative functions |
| Account holder should be able to take a picture of a check to credit their account. | Camera is used to take the picture and Teller account mode handles the deposit. | Camera handles check |

| | | |
|---|---|---|
| Account holder should be able to transfer money from one of their bank accounts to another. | Transfer funds by account number | Verify transations |
| User must confirm this transaction | Second request to verify transaction | Verify transations |
| Account holder should be able to view interests of each of their bank accounts. | Display transaction history | Calculate interest and display it in account balance |
| Balances of accounts should be listed to Account holder. | Create a page that displays accounts | Create account balance in account page |
| Account holder should be able to view their transaction history | A button is created that shows a log with the transaction history | Feature is listed to view transaction history |
| Account holder should be able to retrieve and print out their transaction history | Print function will handle export of transaction history. | Print button function in transaction history |

1.

1.

1.

*Comment: there should be an acceptance criteria for each user story developed by the BA or PM. Remember that you are looking for a balance here, enough to ensure a glitch-free experience for the end-user and relaxed enough to ensure productive and quick code-iteration cycles.*

1. Testing Risks

| S. No | Risk | Probability | Impact | Mitigation Plan |
|---|---|---|---|---|
| 1 | Security | High | New Features, loopholes | Create a secure connection. |

| | | | | |
|---|---|---|---|---|
| 2 | Unstable | High | Slow application, possibly multiple conflicts from other features | Login not fully functioning fully, but still functional. |
| 3 | | | | |
| | | | | |

1.
1.
1.

1. Test Results:
    1. Document the results of the tests you have done, along with severity (high, medium or low)

| User-story/ Acceptance Criteria | Results | Severity |
|---|---|---|
| Create Account | Account is created and user is able to login | Low |
| Scrolling Issue | With keyboard display, user cannot scroll | Medium |
| Navigation | User cannot go back with the android back key | High |
| | | |
| | | |
| | | |

## Code Testing Plan:

- creation of account( )

various strings (null, char limit, +- numbers, all ascii characters, )

- changing checking balance( )

(user) {

transferring double values to same user's accounts (0, negative, > 0, letters, all ascii)

transferring double values to other user's accounts (invalid phone ID number, null in phone ID, all ascii in ID, 0, negative, > 0, letters, all ascii)

}

(teller) {

changing debit value for users( 0, 0.0?, negative numbers, putting letters, more than in account, all ascii characters)

check ID(from: null, too: null; from: ID, too: null; from:null, too: ID;

changing credit value for users( 0, 0.0?, negative numbers, putting letters, more than in account, all ascii characters)

}

else {

check closing account (check attempt at 0, check if (somehow) negative)

check daily limit 10,000 (11:59 pm, 12:00 am)

}

- changing saving balance( )

(user) {

transferring double values to same user's accounts (0, negative, > 0, letters, all ascii)

transferring double values to other user's accounts (invalid phone ID number, null in phone ID, all ascii in ID, 0, negative, > 0, letters, all ascii)

}

(teller) {

various double values debit( 0, 0.0?, negative numbers, putting letters, more than in account, all ascii characters)

various double values credit( 0, 0.0?, negative numbers, putting letters, more than in account, all ascii characters)

}

else {

check closing account(check attempt at 0, check if (somehow) negative)

check daily limit 5,000 (11:59 pm, 12:00 am)

}

- get Number of accounts( )

check on (0 accounts, > 0 accounts)

- interest calculation( )

check monthly increase( jan - feb, dec - jan, oct - nov)

- penalty calculation( )

check monthly penalty(29 days, 30 days, 31 days, hours 5pm - 8pm, if amount 0, amount 99, amount 100, amount 101)

Note it's 25 dollar penalty

## Code Snippet

## Pseudocode Examples of Particular Tests

We first thought to test the creation of the User object. Since there are input fields for various types of information (string, int, ascii, etc.), we test these fields by inputting corner cases (null, ascii characters). After testing and creating the User account, we then verify that the information was stored on the database.

/*

Instance of checking user Account info field: Account.name

Minimum name length >= 3

Name allows numbers or letters

Name doesn't allow other ascii values

*/

```
//checking null
if( Account.setName(null) == false )
    System.out.println("Account errors on null name");
    return false;

//checking minimum name limit >= 3 with letters
if (Account.setName("a") == true)
    System.out.println("Error on minimum name length");
    return false;
if( Account.setName("ab") == true)
    System.out.println("Error on minimum name length");
    return false;
```

```
if( Account.setName(”abc” == false)
    System.out.println(“Error on minimum name length”);
    return false;

//checking minimum name limit >= 3 with numbers
if (Account.setName(“1”) == true)
    System.out.println(“Error on minimum name length”);
    return false;
if( Account.setName(“12”) == true)
    System.out.println(“Error on minimum name length”);
    return false;
if( Account.setName(”123” == false)
    System.out.println(“Error on minimum name length”);
    return false;

//checking other ascii chars from 33 to 47, 58 to 64, 91 to 96, 123 to 126
char asciiChar;
String template = “abc”;
String nameTemp;
for( int i = 33; i < 48; i++ ) {
  asciiChar = i;
  nameTemp = template + asciiChar;
  if(Account.setName(nameTemp) == true) {
      return false;
  }
}
for( int i = 58; i < 64; i++ ){
  asciiChar = i;
  nameTemp = template + asciiChar;
  if(Account.setName(nameTemp) == true) {
      return false;
  }
}
```

etc.

We wanted to include a pseudo code snippet of another test, so we thought to skip to an example of our test for a User's Account money transfer. The input fields we would need to test are the "from" and "to" phone ID input fields, as well as valid input money values that are within the bounds of the account.

Checking the "from" and "to" phone ID input fields will contain the same ascii character checks (to make sure the program doesn't break) as well as for validity, but the input money values will need to be tested a little differently. We also need to test for money bounds and format. We decided to do these extra checks as follows:

```
//check that the input money field is robust (doesn't break with odd values)
if ( myUser.transfer(tempFrom, null ,tempTo) == true)
    //error, should have rejected input
    return false;
if ( myUser.transfer(tempFrom, 0, tempTo) == true)
    //error, should have rejected input
    return false;

//… then we check ascii character values like before …

//now we check for values at and above given limit, current account has 100
if( myUser.transfer(tempFrom, 100, tempTo) == false)
    //error, should have accepted at max
    return false;
if( myUser.transfer(tempFrom, 101, tempTo) == true)
    //error, should have rejected, request more than in account
    return false;
```

CSE110 Iteration Review Questions

NAME: David Chan, Michael Griffin

TEAM NAME: Team Honeybadgers

ROLE: Testers

• Is everyone happy with the quality of work? Documentation? Testing?

Everyone is happy with the quality of work of the team. Everyone tries to keep everyone else accountable for team meetings and work. As for documentation, github documents and code tests / comments help team communication.

• How did everyone feel about the pace of the iteration? Was it frantic? Reasonable? Boring?

Iterations have gone pretty smoothly so far with the team. When there is a common understanding with deadlines, the team is able to communicate and run at a comfortable place.

• Is everyone comfortable with the area of the system they were working in?

The team is open for everyone to communicate preferences on what types of work they want to do. With this, everyone is comfortably challenged in the areas they are working in.

• Are there any tools particularly helping or hurting productivity? Are there any new tools the team should consider incorporating?

The tools that are helping team productivity are Creately for creating the UML Diagram and Trello for team communication. These particular tools are greatly helping team productivity.

• Was the process effective? Were any reviews conducted? Were they effective? Are there any process changes to consider?

The process was very effective for the team. Although some requirements of the class were confusing, each member adapted and communicated well with the rest of the team. Everyone made sure that each member was pulling enough “weight” for the team.

• Was there any code identified that should be revisited, refactored or rewritten?

Particular sections of code have been written with skeleton/placeholder methods to allow developers to finish coding other designated sections before coming back to revisit. These sections have indeed been identified accordingly.

• Were any performance problems identified?

The team found little to no performance problems during the project. Small issues with code platforms and communication have been easily ironed out.

• Were any bugs identified that must be discussed before prioritization?

Some bugs were indeed identified that caused the program to crash at times.

• Was testing effective? Is our test coverage high enough for everyone to have confidence in the system?

Testing was effective for the sections of code that have been completed for each iteration. As each module of code was written and pieced into the larger program as a whole, the team was able to test and trust the system in development.

• Is deployment of the system under control? Is it repeatable?

The deployment of the system should be easily repeatable. Installation is a simple enough process, plugged into the Parse.com libraries and server.