Bookkeeping Project User Guide & Test Plan

Jennifer Brady

Matthew Dobson

Andrew Eissen

Kevin Ramirez

Christian Rondon

Steven Wu

University of Maryland University College

CMSC 495

Dr. Robinson

November 6, 2018

Contents

User Guide

Usage Requirements

The bookkeeping project application can be used by the client in several ways. Depending on the client's specific needs, the application may be used on a personal, offline basis by making use of a private server used to store the codebase on the client's computer. Otherwise, assuming the application has been deployed to an external hosting site, the client may simply make use of a modern browser on a computer or mobile device to navigate to the web URL in question.

Making use of the offline development build of the program will require a computer with enough free space to store a copy of the November 13, 2015 build 5.6.14 of the XAMPP Apache/MySQL bundle. This private server is used to store the files pertaining the display of the interface and all those back-end database-oriented SQL files used to store user account info. In addition, the user will need access to a supported browser with the "JavaScript enabled" permission set to true, as the program does make use of JavaScript to both dynamically build the document object model (DOM) and handle the configuration and display of user data as appropriate.

The deployed online build of the program will simply require a computer or mobile device, a decent Internet connection, and access to a supported browser with the "JavaScript enabled" permission set to true, for the reasons discussed above. The program should be expected to behave in the same way regardless of the means by which it is employed, either online via Internet access or offline via a private server.

All elements on the page that permit the addition or inclusion of user-input data will be able to be manipulated solely via a standard BS 4822 United States layout

QWERTY keyboard and modern computer mouse of the mechanical, optical, trackball, or wireless varieties. For mobile devices, support will extend to the use of styli and the conventional onscreen United States layout QWERTY keyboard.

Data Viewing

As far as program usage is concerned, the interface will be designed to be user-friendly and readily interactive. It will consist of a configuration toolbar containing all the various sorting filters and display options available to the user and a main checkbook-style ledger displaying entries related to the five main document categories. The user will be able to manipulate stored data via the aforementioned filters to view only certain document types, outstanding balances, or active invoices, allowing the client to see at a glance who is owed what at what time. This ease of use and straightforward data display customization will enable the client to bypass the learning curve required by many other feature-heavy bookkeeping software alternatives.

Data Entry

The user will be able to add documents or entries to the main ledger by making use of a JavaScript-powered popup modal that will permit the addition of new documents and the saving of incomplete documents for completion at a later date. This modal will be equipped with its own configuration buttons for the submission of data, saving of progress on an incomplete document, and deletion of an unwanted document. Users will also be able to add new individual transaction entries directly to currently viewed documents displayed in the ledger by making use of the inbuilt entry adder without having to use the modal. This will require that the user input data for fields related to debit, credit, description, and account type, among others.

Introduction

The Bookkeeping Project is a full-featured bookkeeping program that allows the user to track accounts receivable, accounts payable, assets, liabilities, and equity. Using the simple register-style layout, users will be able to enter invoices and payments. These entries will be sortable and a balance sheet style report will be available.

Accessing the Program

Users can access the program by navigating to the provided web domain. From there, the user will create a logon account during their initial access. After initial access has been made, the chosen username and password will need to be entered for all future accesses of the program.

Entering Bookkeeping Project for the First Time

Initially, the user will need to establish an account and what company database they are associated with. This will be accomplished by selecting *Register a new user*.

[insert picture after development]

The user will establish what company database they are associated with and create a username and password for this connection.

[insert picture]

Future Entries

After connecting to the correct web domain, the user will find themselves at the logon screen. The user will enter their selected username and password before pressing enter. Successful logon will load the register screen. Unsuccessful logon will prompt the user to try again or reset their password.

[insert picture]

Learning the Basics

Bookkeeping Project is designed to work like an electronic check registry that provides simple business accounting functions. The top line of the registry is the form from which a user can select the actions they would like to perform. These actions include creating vendors and customers, adding documents for the ledger and associating the vendors, customers and documents with accounts.

[insert picture]

Adding Customers and Vendors

Before creating accounts receivables or payable, customers and vendors must be established. This actions will be completed by…

- **Creating Vendor:** In order to create a vendor, the **Company** name must be entered and an **Address** must be provided. This will establish the vendorID number.

  [insert picture]

- **Creating Customers:** In order to create a customer, the **Customer** name must be entered and an **Address** must be provided. This will establish the customerID number.

  [insert picture]

Entering Income and Expenses

Income and expenses can be entered from the top form line on the page. Here, a user will select the vendor or customer, enter a documentType, date, debit or credit and a description.

[insert picture]

Viewing Reports

Bookkeeping Project has the ability to display several reports. These reports will be accessed by selecting the preset link.

[insert picture]

Test Plan

Overview

In order to illustrate the types of accepted input and the manner in which this legitimate data may be collated and eventually displayed to the user, a number of example test cases illustrating certain use-case scenarios have been included. Making use of sample data that represents what the user or client could be reasonably expected of attempting to enter into the database, these scenarios describe in detail every step of the process from the input data evaluation to the output in the interface display.

UI Functionality Testing

UI Test 1(Compatibility)

Input 1: UI is tested in Chrome 70 and Firefox 62

Expected Output: UI loads and passes each functionality test

UI Test 2 (Navigational)

Input: Each site link is tested individually

Expected Output: All links are directed to the correct location or output

UI Test 3 (Forms)

Input 1: User mandatory fields are not completed

Expected Output: Error message prompting user for correct input

Input 2: Verify default values are populated

Expected Output: N/A

Input 3: Submitted form connects to database

Expected Output: Tables are updated with new information

Input 4: Incorrect data types are entered into forms

Expected Output: Error message prompting user for correct format or error message showing submission was not accepted

UI Test 4 (Optimization)

Input 1: UI is tested on a minimized browser

Expected Output: UI resizes appropriately

Input 2: UI is tested on a mobile device

Expected Output: UI is visible and functional on device

UI Test 5 (Content)

Check	Completion
Site contains no spelling errors	
All content is visible	
All colors are standardized and easily viewed	

General User-Interface Testing

UI I/O Scenario 1.1 (Successful Authentication and Access to the Database Server)

Input: User types both the username and password correctly.

Expected Output: A customized message should be generated on the webpage indicating the user has successfully logged in to the system. Any other error messages generated by the PHP/SQL compiler are suppressed.

UI I/O Scenario 1 .2 (Failed Authentication and Access to the Database Server)

Input: User types wrong username and/or password.

Expected Output: An error message displayed on the website/app stating, "Wrong username and/or password." Another message can be display encouraging users to register a user account or offering options on how to recover his username/password. Any other error messages generated by the PHP/SQL compiler are suppressed.

UI I/O Scenario 2.1 (Successful User Registration to the Database System)

Input: User types his or her own chosen unique and strict username and a secure password.

Note: The syntax for a username must only contain alphanumeric characters. Password minimum length should be specified while allowing special characters.

Expected Output: The system gives the newly registered user with regular privileges and with a message indicating a successful registration.

UI I/O Scenario 2.2 (Failed User Registration to the Database System)

Input: User types his or her own chosen unique and strict username and a secure password, but fails to meet the requirements.

Expected Output: The system generates a customized error message to the user's website client, notifying him or her that he or she failed to meet the requirement specification for a username and/or password. The user is then referred back to the registration webpage until he or she successfully registered (UI I/O Scenario 2.1). Any other error messages generated by the PHP/SQL compiler are suppressed.

UI I/O Scenario 3.1 (Regular User Having Read rights to his/her Stored Information Only)

Input: User views his or her accounts, documents, and general ledgers.

Expected Output: Information related to his or her stored accounts, documents, and general ledgers must only be viewable to him or her exclusively.

UI I/O Scenario 3.2 (Regular User Having Edit rights to his/her Stored Information Only)

Input: User wants to change (or edit) some of his or her personal information (name, address, etc.) and makes changes accordingly.

Expected Output: The system must store the newly stored (or edited) information and reflects this change after verification from the database server and display a message that changes were successfully applied.

UI I/O Scenario 3.3 (Regular User Having Edit rights Failed to Apply the Changes)

Input: User wants to change (or edit) some of his or her personal information (name, address, etc.) and makes changes accordingly, but the syntax for the new (or edited) information was invalid.

Expected Output: The system checks this information and validates it if it meets the required valid syntax. If not, the user is referred back to the webpage where he or she attempted the change and displays a customized error message stating "The <info type> that you wished to change failed due to ...". This process repeats itself until a valid syntax for that info type is met (UI I/O Scenario 3.2). Any other error messages generated by the PHP/SQL compiler is suppressed.

UI I/O Scenario 4.1 (Regular User Having Read rights Uses Built-In Search Engine and Founds a Match With a Corresponding Info Type)

Input: User types something on a search bar with specified info type.

Expected Output: The webpage displays a partial view of table's rows. The webpage can display a suggestion that the same search terms are found in other tables, but I suggest this is optional.

UI I/O Scenario 4.2 (Regular User Having Read rights Uses Built-In Search Engine and Founds no Match With a Corresponding Info Type)

Input 1: User types something on a search bar with specified info type, but his or her search terms returns with no results.

Expected Output: The webpage displays a customized message stating, "No results found for <search terms>." Any other error messages generated by the PHP/SQL compiler is suppressed.

SQL Unit Testing

Assert Unit Testing 1.1 (Asserting Equal Row Count After Multiple (or Single) INSERT Queries)

Input 1: Execute a single INSERT query on an empty table to insert entries from another

table and COMMIT. In PHP assert unit testing, call the testGetRowCount function to compare these two tables.

Expected Output 1: After executing COMMIT, this SELECT query must return true if the same number when executing SELECT query for another table. The testGetRowCount function must pass the test if two tables have the same number of rows.

Sample 1:

	Input 1	Output 1
MySQL (MariaDB)	SQL> INSERT INTO DB.TABLE (SomeField) SELECT SomeField FROM DB.OTHERTABLE; SQL> COMMIT; SQL> SELECT IF((SELECT COUNT(*) FROM DB.TABLE) = (SELECT COUNT(*) FROM DB.OTHERTABLE), 'Equal', 'Not Equal');	Equal
PHP (PHPUnit)	<?php testSameRowCountTwoTables('table', 'othertable'); ?>	Pass

Input 2: Execute *n* number of INSERT queries on an empty table and COMMIT. In PHP unit testing, call the testExactRowCountOneTable function to verify the exact entries added.

Expected Output 2: After executing COMMIT, the SELECT query must return 'Equal' if the row count is the same as number of *n* INSERT queries. The testExactRowCountOneTable function must pass the test if all INSERT queries

successfully added to the table.

Sample 2:

	Input 2	Output 2
MySQL (MariaDB)	SQL> INSERT INTO DB.TABLE (SomeField) VALUES (‘value1’); SQL> …. SQL > INSERT INTO DB.TABLE (SomeField) VALUES (‘value100’); SQL> COMMIT; SQL> SELECT IF((SELECT COUNT(*) FROM DB.TABLE) = 100, ‘Equal’, ‘Not Equal’);	Equal
PHP (PHPUnit)	<?php testExactRowCountOneTable(‘table’, 100); ?>	Pass

Assert Unit Test 1.2 (Asserting Lesser or More Row Count After Multiple (or a Single) INSERT or DELETE Queries After State Changes of the Table)

Input 1: Execute a single INSERT or DELETE queries on a non-empty table and COMMIT. In PHP unit testing, call the testOperatorRowCountTwoTables function to test if the non-empty table is greater or lesser than the other table after a single INSERT or DELETE query.

Expected Output 1 (Insert): After executing COMMIT, the SELECT query must return ‘Greater’ since the non-empty table has added all entries from the other table. The testOperatorRowCountTwoTables function must pass the test since all entries of the other table have been added into the existing non-empty table, verifying the new

existing table has greater row count than the other table.

Expected Output 1 (Delete): After executing COMMIT, this SELECT query must return ‘Lesser’ since all. The testExactRowCountOneTable must the test if a single DELETE query has depopulated all entries of the table.

Sample 3.1:

	Input 1	Output (Insert) 1
MySQL (MariaDB)	SQL> INSERT INTO DB.TABLE (SomeField) SELECT SomeField FROM DB.OTHERTABLE; SQL> COMMIT; SQL> SELECT IF((SELECT COUNT(*) FROM DB.TABLE) > (SELECT COUNT(*) FROM DB.OTHERTABLE), ‘Greater’, ‘Lesser’);	Greater
PHP (PHPUnit)	<?php testOperatorRowCountTwoTables(‘table’, ‘othertable’, ‘Greater’); ?>	Pass

Sample 3.2

	Input 1	Output (Delete) 1
MySQL (MariaDB)	SQL> DELETE FROM DB.TABLE; SQL> COMMIT; SQL> SELECT IF((SELECT COUNT(*) FROM DB.TABLE) = (SELECT COUNT(*) FROM DB.OTHERTABLE), ‘Equal, ‘Not Equal’);	Lesser

PHP (PHPUnit)	<?php testOperatorRowCountTwoTables(‘table’, ‘othertable’, ‘Lesser’); ?>	Pass

Assert Unit Test 1.3 (Asserting the Addition/Deletion/Update of a Particular Entry’s one of its Field Values is Added/Deleted/Updated On Table’s Population)

Input 1: Execute a single INSERT query with a field value on a table and COMMIT. In PHP unit testing, call the testIfFoundValue function to test if the about-to-be added value has been inserted to the table.

Expected Output 1: After executing COMMIT, the SELECT query must return ‘Found’ of that particular entry’s specific field value. The testIfFoundValue function must pass if the new INSERT query with a ‘value’ of field has been successfully added to the table.

Sample 4.1

	Input 1	Output 1
MySQL (MariaDB)	SQL> INSERT INTO DB.TABLE (SomeField1) VALUES (‘value’); SQL> COMMIT; SQL> SELECT IF(SELECT SomeField FROM DB.TABLE WHERE SomeField = ‘value’) = ‘value’, ‘Found’, ‘Not Found’);	Found
PHP (PHPUnit)	<?php testIfFoundValue(‘table’, ‘value’, ‘Found’); ?>	Pass

Input 2: Execute a single DELETE query on a non-empty table and COMMIT. In PHP unit testing, call the testIfFoundValue function if the recently DELETE query with a ‘value’ of field has been deleted from the table.

Expected Output 2: After executing COMMIT, the SELECT query must return ‘Not Found’ of that particular entry’s specific field value. The testIfFoundValue function must pass the test if the recently DELETE query with that specific value has been successfully removed from the table.

Sample 4.2

	Input 1	Output 1
MySQL (MariaDB)	SQL> DELETE FROM DB.TABLE WHERE SomeField = ‘value’; SQL> COMMIT; SQL> SELECT IF(SELECT SomeField FROM DB.TABLE WHERE SomeField = ‘value’) = ‘value’, ‘Found’, ‘Not Found’);	Not Found
PHP (PHPUnit)	<?php testIfFoundValue(‘table’, ‘value’, ‘Not Found’); ?>	Pass

Input 3: Execute a single UPDATE query on a non-empty table and COMMIT.

Expected Output 3: After executing COMMIT, the SELECT query must verify the changes applied on that particular entry’s specific field value.

Sample 4.3

	Input 1	Output 1
MySQL (MariaDB)	SQL> UPDATE DB.TABLE SET SomeField = ‘new value’ WHERE SomeField = ‘value’; SQL> COMMIT; SQL> SELECT IF(SELECT SomeField FROM DB.TABLE WHERE SomeField = ‘value’) = ‘value’, ‘Found’, ‘Not Found’);	Not Found
PHP (PHPUnit)	<?php testIfFoundValue(‘table’, ‘value’, ‘Not Found’); ?>	Pass

Assert Unit Test 2 (Asserting Addition/Deletion of a Particular Entry Across the Referenced Tables Are Applied)

Input: Execute a single INSERT or DELETE query of a particular entry to a table and COMMIT. In PHP unit testing, the testForeignKeyIsReferencedAcross function to find a foreign key by all referenced by other tables.

Expected Output: After executing COMMIT, the SELECT queries must return ‘Found’ of that particular foreign key. The testForeignKeyIsReferencedAcross function must pass the test if all referenced tables are found to have the specific foreign key value which is ‘keyvalue’.

Sample 5

	Input 1	Output 1
MySQL (MariaDB)	SQL> INSERT INTO DB.TABLE1 (SomeForeignKey) VALUES (‘keyvalue’); SQL> COMMIT; SQL> SELECT IF(SELECT SomeForeignKey FROM DB.TABLE1 WHERE SomeForeignKey = ‘keyvalue’) = ‘keyvalue’, ‘Found’, ‘Not Found’); SQL> … SQL> SELECT IF(SELECT SomeForeignKey FROM DB.TABLEN WHERE SomeForeignKey = ‘keyvalue’) = ‘keyvalue’, ‘Found’, ‘Not Found’);	Found (From multiple SELECT queries from multiple referenced tables)
PHP (PHPUnit)	<?php testForeignKeyIsReferencedAcross(‘SomeForeignKey’, ‘keyvalue’); ?>	Pass

Assert Unit Test 3.1 (Asserting an Entry to the DOCUMENTS Table that It Cannot Have Two Nulls or Present on Columns vendorID and customerID at the Same Time)

Input: Execute two INSERT queries where each have a value of vendorID or customerID. In PHP unit testing, the testValidValuesDocuments function tests if it can retrieve from the table with the specified values of customerID and vendorID.

Expected Output: The compiler should return an error message indicating that only one of them must be null and the other present. The testValidValuesDocuments function must pass since it cannot retrieve an entry with these two existing values at the same time.

Sample 6

	Input 1	Output 1
MySQL (MariaDB)	SQL> INSERT INTO DB.DOCUMENTS (userID, vendorID, customerID, name, type, isPosted) VALUES (1, 1, 1, 'some name', 'JE', 1); SQL> INSERT INTO DB.DOCUMENTS (userID, vendorID, customerID, name, type, isPosted) VALUES (1, NULL, NULL, 'some name', 'JE', 1);	Two INSERT queries will fail and generates error messages with constraint name specifying some improper value.
PHP (PHPUnit)	<?php testValidValuesDocuments('1', '1', 'Not Found'); ?>	Pass

Assert Unit Test 3.2 (Asserting an Entry to the GENERALLEDGER Table that It Cannot Have Two Nulls or Present on Columns Debit and Credit at the Same Time)

Input: Execute an INSERT query where the values for credit and debit are either null or present at the same time.The testValidValuesGL function tests if it can retrieve two specified values at the same time from the table.

Expected Output: The compiler should return an error message indicating that only one of them must be null and the other present. The testValidValuesGL function must pass since it cannot retrieve such entry with two existing values. The sample table for this unit test is similar to Assert Unit Test 3.2.

Assert Unit Test 3.3 (Asserting an Entry to the DOCUMENTS and ACCCOUNTS Tables Having An Existing Enumerated Type Value is Valid)

Input: Execute an INSERT query where the type value is not one in the enumerated list. The testValidEnum function retrieves an entry with specified enum type.

Expected Output: The compiler should return an error message indicating that provided type value does not exist. The testValidEnum function must pass if it cannot find an entry whose enum type that does not exist.

	Input 1	Output 1
MySQL (MariaDB)	SQL> INSERT INTO DB.DOCUMENTS (userID, vendorID, customerID, name, type, isPosted) VALUES (1, 1, 1, 'some name', 'JG', 1); SQL> INSERT INTO DB.ACCOUNTS (userID, code, name, type) VALUES (1, 1, 'some name', 'DEBT');	Two INSERT queries will fail and generates error messages with constraint name specifying some improper value.
PHP (PHPUnit)	<?php testValidEnum (1, 1, 1, 'some name', 'JG', 1, 'documents', 'Not Found'); testValidEnum(1, 1, 'some name', 'DEBT', 'accounts', 'Not Found'); ?>	Pass

General Security Main Objectives

There are no exact input and expected output for these security tests for this test plan. However, a vulnerability scanner is used to identify vulnerabilities on our front-end and back-end codes and patch them up before releasing them as final product as a double-entry bookkeeping system software. Specifically, we will use OpenVAS to address these security tests below and to confirm if we are able to meet them.

Security Test 1.1 (Verify Root Account is Disabled)

Security Test 1.2 (Verify the Access to the Database Server is not in Root)

Security Test 1.3 (Verify all Administrators Have Non-Root Privileges)

Security Test 1.4 (Verify all Regular Users Have Non-Administrative and Non-Root Privileges)

Security Test 2 (Verify all the Inputs Provided by the User Have Been "Sanitized")

Security Test 3 (Verify all Versions of Software Packages Used Are Concealed)