Code Miners

*“Never dig straight down!”*

**-SCRUM LOG-**

26 OCT 2019:

In attendance:

Moe Soliman

Po Jen Su

Adam Hurd (Scrum Master)

George Good

Not in attendance:

Sami (Product Owner)

Topics: Team rules, coding standards, initial Agile stories

Team rules:

1) Be on-time for meetings

2) Finish assignments before sprint checkpoints

3) Give advance notice if you’re going to miss a meeting

4) Give team updates on assignment progress. Avoid overlapping assignments.

5) Members are responsible for writing and submitting their agile stories to the product owner.

Coding standards:

1) Frameworks: SQLite for database, Qt (C++) for UI, Github (Version control; individual branches for each member), Trello (Agile management), Doxygen (documentation)

2) Provide sufficient commenting that the code function is evident.

3) Use descriptive identifier names (“i” for index is fine)

Agile Stories (Writing): DUE MONDAY & ONE UML ITEM PER MEMBER. Planning poker will be played during team meeting after class.

Mo – 4, 5, 6

Po Jen – 13, 14

Adam – 8, 9 ,10

George – 1, 2, 3

Sami – 7, 11, 12

**Agile story template:**

#) Description: As a \_\_\_\_\_\_,

a. Assumptions:

b. Assignee:

c. Story Point Estimation:

d. Priority:

e. List of Tasks and Tests:

i. Tasks:

1.

ii. Tests:

1.

f. Definition of Done

**EXAMPLE:**

1. Description: As a customer, I want to be able to click on a help option that will explain how to operate the program.

a. Assumptions: Main window is working and space on the window is allocated for the help button.

b. Assignee:

c. Story Point Estimation: 1

d. Priority: 1

e. List of Tasks and Tests:

i. Tasks:

1. Create a button named “HELP”.

2. Make the help button go to another window.

3. Come up with some text that explains how to operate the program that will be displayed on the help window.

ii. Tests:

1. Clicking on the button will take you to a new window.

2. The new window displays the correct text.

f. Definition of Done: This story is done when there exists a help button on the main window that takes the customer to a separate window with information regarding how to operate the program.

General tasks:

~~- Establish Github~~

~~- Establish database~~

~~- Plan out UI~~

~~- Plan out program structure (UML)~~

~~- Team rules~~

~~- Coding standards~~

~~- Agile stories- Trello board?~~

- Continuous Integration? (Extra credit) -ASK PROF

- Planning poker (need agile stories)

- UML (Class, State, Use case, Activity)

Adam: in-progress

8) Add/Delete items

Description: As an administrator, I want to manage the store’s inventory list by adding and deleting items from the list.

a. Assumptions: The inventory list (database) is created , the menu/UI is created, and the admin account is created.

b. Assignee: Adam

c. Story Point Estimation: TBD

d. Priority: (Product Owner: TBD)

e. List of Tasks and Tests:

i. Tasks:

1. Create buttons for “Add item” and “Delete item”.

2. Connect add button to “insert” new item with price (price should be float/double)

3. Display record to be added and create “Confirm addition” button

4. Add record (INSERT)

5. Connect delete button to “delete” existing item (price not included)

6. Display record to be deleted and create “Confirm deletion” button. Create “Confirm deletion

7. Delete record (DELETE)

8. Error-check input: check for missing fields (“add”) or invalid item names (“delete”).

ii. Tests:

1. Click “Add item”. A window should pop up with text prompts for item name and price.

2. Enter information. Click “Accept”.

3. Confirmation window should appear with data to be entered. Click “Confirm”.

4. Navigate back to table view. Table should be updated to reflect addition.

5. Click “Delete item”. A window should pop up with a text prompt for the name of the item to be deleted.

6. Enter the name of the item added in step 1. Click “Accept”

7. Confirmation window should appear with data to be entered. Click “Confirm”.

8. Navigate back to table view. Table should be updated to reflect deletion.

f. Definition of Done: When there are “Add” and “Delete” item buttons that allow the user (administrator) to add or delete items from the item table in the database.

9) Display sold items

Description: As a store manager, I want to view enter the name of an item and see how sales of that item as well as total revenue.

a. Assumptions: The inventory list (database) is created and the store manager account is created, and the menu/UI has been created.

b. Assignee: Adam

c. Story Point Estimation: TBD

d. Priority: (Product Owner: TBD)

e. List of Tasks and Tests:

i. Tasks:

1. Create a button named “View item sales”.

2. Connect the button to a dialog box that will read in an item name.

3. Search the item table for that name and display the item name, price, quantity sold, and total revenue (before tax).

4. Error-check input: check for missing field or invalid item name.

ii. Tests:

1. Clicking on “View item sales”. A window should pop up with a text prompt for item name.

2. Enter the item name and click “accept”.

3. Navigate to the table view. The item should be displayed with its name, price, quantity sold, and total revenue (pre-tax).

4. Attempt a search with a missing or invalid item name. A prompt alerting the user to the error should pop up and give the user another chance to enter a correct name.

f. Definition of Done: When there is a “View item sales’ button that allows a store manager to search for an item by name and view sales of that item.

10) View membership purchases

Description: As a store manager, I want to enter the name of a customer or their membership number and view the purchases of that member.

a. Assumptions: The customer list (database) is created and the store manager account is created, and the menu/UI has been created.

b. Assignee: Adam

c. Story Point Estimation: TBD

d. Priority: (Product Owner: TBD)

e. List of Tasks and Tests:

i. Tasks:

1. Create a button named “View membership purchases”.

2. Connect the button to a dialog box that will read in an member name or their membership number.

3. Search the member table for that name or membership number and display the customer name, membership number, and total purchases [pre-tax?].

4. Error-check input: check for missing fields or invalid member name/number. Note that the search should execute if one field is left blank, so long as the either field has valid input.

ii. Tests:

1. Clicking on “View membership purchases”.”. A window should pop up with a text prompt for a member name or membership number.

2. Enter a member name and click “Accept”.

3. Navigate to the table view. The member’s info (name, membership number, and total purchases) should be displayed.

4. Repeat steps 1-3 but use the member’s number instead.

5. Attempt a search with both fields empty. A prompt should pop up to instruct the user to enter either a membership number or a member name.

6. Attempt two searches: one with an incorrect member name, and the other with an incorrect membership number. Both searches should have a prompt pop up to alert the user to the error and allow them to try again.

f. Definition of Done: When there is a “View membership purchases” button that allows a store manager to search for a member by name or by membership number and view their total purchases.

13) User accounts

Description: As a developer, I want to ensure that only store managers and system administrators have access to the program; therefore, each class of user will have an account name and password that must be used to log in and access features.

a. Assumptions: The store manager and admin accounts are created, and the menu/UI has been created. All manager/admins features have been identified.

b. Assignee: Po Jen

c. Story Point Estimation: TBD

d. Priority: (Product Owner: TBD)

e. List of Tasks and Tests:

i. Tasks:

1. Create a button named “Login”.

2. Connect the button to a dialog box that will read in a username and password.

3. Based on the username, log in the user as either a manager, an admin, or none (incorrect input).

4. Error-check input: check for missing/incorrect username or password.

5. Create menus for each account types to reflect the features that they have access to.

6. Create a “Logout” button that returns the user to the main menu with a cleared login state.

ii. Tests

1. Clicking on “Login”. A window should pop up with a text prompt for a username and password.

2. Enter the a manager’s username and password. Click “Accept”.

3. A menu should pop up that displays all of the manager’s options.

4. Click “Logout”. The user should be returned to the main menu.

5. Repeat steps 1-4 but with an admin account.

6. After logging out a second time, attempt to login with a missing name, missing password, and incorrect name and password. All attempts should receive an error message and allow the user to try again.

f. Definition of Done: When there is a “Login” button that allows a user to log in as a manager or administrator. They should have access to all appropriate features.

14) Database persistence

Description: As an administrator, I want to ensure that all changes made to the database are persistent between executions. This includes modifications (add, delete, or update) to item and member tables.

a. Assumptions: The database is created with item and member tables.

b. Assignee: Po Jen

c. Story Point Estimation: TBD

d. Priority: (Product Owner: TBD)

e. List of Tasks and Tests:

i. Tasks:

1. Utilize SQL to create a database.

2. Use INSERT, UPDATE, and DELETE commands to modify the database.

ii. Tests:

1. Use features developed in [agile stories] that involve adding, modifying, or deleting content from the database.

2. Note the changes and close the program.

3. Run the program again and go to the database table view. All changes should still be present.

f. Definition of Done: When there is a database that is persistent across executions.

**UML**

Case number 8

System: Qt

Use Case: Add items

Description: This allows the user (an administrator) to add or delete items from the store inventory. The store inventory is a table within the main database that holds item names and price.

**Actors:**

* User (must be admin)

**Triggers:**

* The user clicks on the “Add item” button from the admin menu.

**Preconditions:**

* There is a table for the item inventory in the database
* The user is logged in as an administrator.
* The user has the item name and price to be added.

**Postconditions:**

* The user will receive a confirmation window that the item has been added.
* The table will updated with the item information.

**Normal Flow:**

1. The user clicks on the button to add an item.

2. The system will display a dialog box with a field for the user to enter the item name and price.

3. The system will display a confirmation window with item information to be added.

4. The user clicks “Accept”.

5. The system reads in the item name and price, create a record, and inserts it into the database table.

6. The user is returned to the admin menu.

**Alternate Flows:**

4A1. The user notices an error and clicks “Cancel”.

4A2. The system returns to the original dialog box for the user to enter a different item name and/or price.

5A1. The system fails to properly insert the new record. An error message is displayed.

5A2. The user is returned to the admin menu.

Sami: in-progress

7) Description: As a team member, I want the ability to create *“dummy”* purchases for new customers that do not affect the stores, to validate the accuracy of the software.

1. Assumptions: The database is established as well as the GUI to interact with the checkout method.
   1. Assignee: TBD
   2. Story Point Estimation: TBD
   3. Priority: TBD
   4. List of Tasks and Tests:
      1. Tasks:
         1. Create the Database to hold our products information
         2. Create the check out/ purchase method
         3. Create another method that calls upon the checkout method, however it does not affect the numbers of the database and is only used to validate the software.
         4. This should not be accessed by the customer, however **may** be useful to administrators of the program. But should be allowed to team members working on the project:
      2. Tests:
         1. Click “Trial Purchase Method” which should call upon our check-out method.
         2. Team member selects the products they desire and once they click “Confirm Purchase” they are prompted with a receipt.
         3. Team member should check the database to make sure the query has executed properly with the database.
2. Definition of Done:
   1. The team member should have a method which should only show when logged in as a team member, that allows them to test the validity of the checkout method. This method allows the team member to make a “*dummy”* purchase which does not affect the database values but rather tests functionality.

11) As a team member, I want to the application to determine if any Regular Customer should be prompted to promote to an Executive Status when it is beneficial to them based on their spending.

1. Assumptions: A hierarchy is set up that denotes which customers are regular customers and which customers are executive customers. In addition to the customer check-out method is also established.
   1. Assignee: TBD
   2. Story Point Estimation: TBD
   3. Priority: TBD
   4. List of Tasks and Tests:
      1. Tasks:
         1. Establish the threshold when it is considered beneficial to upgrade to executive status.
         2. Establish the algorithm used to determine when it the promotion should be prompted.
         3. Set up a pop up to ask the user if they wish to upgrade, if they accept their rank in the database should reflect their decision.
      2. Tests:
         1. Attempt checking out below our set threshold and proceed without being prompted.
         2. Attempt checking out above our set threshold, and be prompted to upgrade status.
         3. Accept the status upgrade and check if it reflects the database.
2. Definition of Done: When checking out as a customer if a certain threshold is met you will be prompted asking if you wish to be upgraded to executive status. Otherwise if the threshold is not met you will not be prompted to upgrade your status.

#4

**Story**: As an administrator, I want to produce a list of customers that will be sorted and shows the amount of rebate to making it easy to manage the customer’s list. And to recommend upgrades.

**Assumptions**: After creating the table in the database page, will create a button that will sort customers by name and display the amount of rebate. Table will contain all the customers names and their type and rebates amount.

**Tasks:**

* Using QT create a database page and add the table view widget.
* Administrator must first verify credentials to access database page.
* Once credentials get verified QT will change to the database section where it will show two tables one for customers and one for items.
* A sort button will be created with the name of rebate once the button is clicked the list will reflect the change only customers with executive type will be displayed and their rebate amount by their names.

**Definition of Done:**

* Make sure code adheres to coding standards.
* Perform multiple tests once finished with the code. First will test with the existing table without modifying any customers on the list. Then will add and remove customers and perform more tests to debug the code and make sure it works as needed.
* Get feedback from customer.
* If customer feedback was negative work on changing the code to reflect the changes needed by the customer.
* Confirm with product owner that code adheres to the team and the project standards.
* Document the code to reflect implementations and meaning.
* The database should not show regular type customers on the sorted list.
* The database should not show any other columns other than the rebate amount and the customer's name and executive type column.

#5

**Story**: As an administrator, I want to have a section to input the month and be able to see all the customer’s memberships that will expire that current month in order to track them and notify them to renew the membership. Which makes managing the customer’s database easier.

**Assumptions**: After creating the administrator page, their is going to be a section called memberships. This section will be mainly created to manage memberships only. In the page there is a box called expiration month once a month is inputted the list will reflect changes and show customers with expiring memberships.

**Tasks:**

* Using QT create a memberships page and add the table view widget.
* Administrator must first verify credentials to access database page.
* Once credentials get verified QT will change to the membership section where it will the whole customers database.
* A box with the name expiration month will be created. Then methods will have to be written that takes input and send it to the sort entries method in the customer controller class and sort customers accordingly.

**Definition of Done:**

* Make sure code adheres to coding standards.
* Perform multiple tests once finished with the code. First will test with the existing table without modifying any customers on the list. Then will add and remove customers and perform more tests to debug the code and make sure it works as needed.
* Get feedback from customer.
* If customer feedback was negative work on changing the code to reflect the changes needed by the customer.
* Confirm with product owner that code adheres to the team and the project standards.
* Document the code to reflect implementations and meaning.
* The database should not show any customer with good standing membership.
* The database should only show customer’s names, Type and their expiration date.

#6

**Story**: As an administrator, I want to have the ability to be able to add and remove customers off the list just in case manual overrides is needed.

**Assumptions**: After creating the table in the database page, will create two buttons one to remove and one to add customers to the customer’s database.

**Tasks:**

* Using QT create a database page and add the table view widget.
* Administrator must first verify credentials to access database page.
* Once credentials get verified QT will change to the database section where it will show two tables one for customers and one for items.
* In the database page will create two buttons one to add customers and one to remove them.
* Methods will be created to take the signal input and update the database accordingly.

**Definition of Done:**

* Make sure code adheres to coding standards.
* Perform multiple tests once finished with the code. First will make sure that the add button is working, by adding customers and checking the database being updated or not using the existing database table and another 3rd party database access. Same for the remove option.
* Get feedback from customer.
* If customer feedback was negative work on changing the code to reflect the changes needed by the customer.
* Confirm with product owner that code adheres to the team and the project standards.
* Document the code to reflect implementations and meaning.
* The database should reflect added customers and removed customers.

Moe’s Use case name: Add customer to the database

**Summary**: Administrator uses valid credentials to enter the database table and either remove or add customers to the list.

**Actor**: Administrator

**Precondition**: Admin open the program login window shows on the screen.

**Description**:

* Administrator uses login credentials to login to database.
* Once system recognize credentials administrator will have full access to customer’s database.
* Database page will have a variety of buttons which represent different actions that can be done on the list.
* Administrator then clicks the add button to add a new customer to the database.
* The system will prompt the administrator to add customer information to store in the database.
* Once Administrator input the information, the information will be stored in the database.