THE UNIVERSITY OF SYDNEY

School of Information Technologies
Faculty of Engineering & IT

ASSIGNMENT/PROJECT COVERSHEET - GROUP ASSESSMENT

Unit of Study: SOFT2412

Assignment name: Tools for Agile Software Development

Tutorial time: CC23_Wed_2pm **Tutor name: Huaicheng Liu**

DECLARATION
We the undersigned declare that we have read and understood the *University of Sydney Student Plagiarism: Coursework Policy and Procedure*, and except where specifically acknowledged, the work contained in this assignment/project is our own work, and has not been copied from other sources or been previously submitted for award or assessment.

We understand that understand that failure to comply with the *Student Plagiarism: Coursework Policy and Procedure* can lead to severe penalties as outlined under Chapter 8 of the *University of Sydney By-Law 1999* (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

We realise that we may be asked to identify those portions of the work contributed by each of us and required to demonstrate our individual knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

Project team members				
Student name	**Student ID**	**Participated**	**Agree to share**	**Signature**
1. Vivian Ha	510075642	Yes / No	Yes/No	Vivian Ha
2. Faye Chen	500111862	Yes / No	Yes / No	FAYE CHEN
3. Anh Dao	510512228	Yes / No	Yes / No	A.D
4.Noah Vass	510357478	Yes / No	Yes / No	n
5.		Yes / No	Yes / No	
6.		Yes / No	Yes / No	
7.		Yes / No	Yes / No	
8.		Yes / No	Yes / No	
9.		Yes / No	Yes / No	
10.		Yes / No	Yes / No	

Vending Machine Development – Final report (sprint 3)

[Oct 26 – Nov 2]

Team structure

Name	Scrum role	Main contribution
Vivian Ha	Scrum master	Scrum management; Logic development and testing
Faye Chen	Core developer	GUI development
Anh Dao	Core developer	Database development and testing; Jenkins management
Noah Vass	Product owner	Project leader; Logic development

Daily-stand-up meeting

Date: Oct 27	**What did you do yesterday?**	**What are you doing today?**	**Any news or challenges?**
Vivian Ha		List all cart items to checkout page	
Faye Chen		Make report functions for SQL	I did not work on any of the SQL functions prior to this so took me a while to figure it out
Anh Dao		Working on Card Checkout functionality	
Noah Vass	Worked on integrating the logic supporting paying with cash with the actual UI and transaction database.	Finishing the integration including making sure that unsuccessful transactions do not alter the amount of money in the database.	Difficult to integrate the logic and transactions as the logic actively modifies the currency database.

Date: Oct 28	**What did you do yesterday?**	**What are you doing today?**	**Any news or challenges?**
Vivian Ha	Cart items UI on payment page	Role access, also link to home page different admin buttons; anonymous user	
Faye Chen		Make report functions for SQL (continued)	Fixing the format of the transaction table took a lot of time
Anh Dao	Worked on card check out functionality	Saving Card per user into the database	Anonymous user
Noah Vass	Finished integration of transactions, cash and UI. Cash payment now works bug-free.	Implement testing of cash function. Make sure that change given is accurately	Testing coverage has been difficult because of poor modularization by

		reflected in the SQL database.	me in terms of the cash logic and UI.

Date: Oct 29	**What did you do yesterday?**	**What are you doing today?**	**Any news or challenges?**
Vivian Ha	Role access logic	Role access UI	
Faye Chen		Make GUI for reports	
Anh Dao	Finished saving CC info into database	Generating report for different Roles	
Noah Vass	Did developer run-throughs of purchases from each role. Minor bug-fixes for both purchase types.	Developer run-throughs of entire system. Bug-fixing.	As the system is tested more thoroughly, the number of issues increase exponentially.

Date: Oct 31	**What did you do yesterday?**	**What are you doing today?**	**Any news or challenges?**
Vivian Ha	Role access UI	Junit testing; add defensive programming from testcases; fix bugs	JUnit tests are unordered; database tests are very difficult to maintain when database is modified when tests run
Faye Chen		Fix idle timer; add different cancelling methods for the cancel transactions report	
Anh Dao	Report generation for different roles		
Noah Vass	Continued to work on integration testing and run-throughs. Recorded many bugs for Jira.	Continuing to run-through the system and ensuring it all works as intended.	Fixing bugs just creates new ones in other places. Worried about reaching the deadline.

Date: Nov 1	**What did you do yesterday?**	**What are you doing today?**	**Any news or challenges?**
Vivian Ha	Extensive Junit testing; achieve 70% coverage	Final JUnit; fix minor bugs; prepare final release	
Faye Chen		Timer bug fixes; report GUI bug fixes	It was hard to use javafx with java Timer object; someone changed the format of

			different admin windows and I did not know so it was confusing for a while
Anh Dao		Fixing any bugs in final release	
Noah Vass	Worked on integration testing and run-throughs.	Finalizing release.	Application is pretty much finished. Just some minor bugs to complete.

Scrum events and artifacts

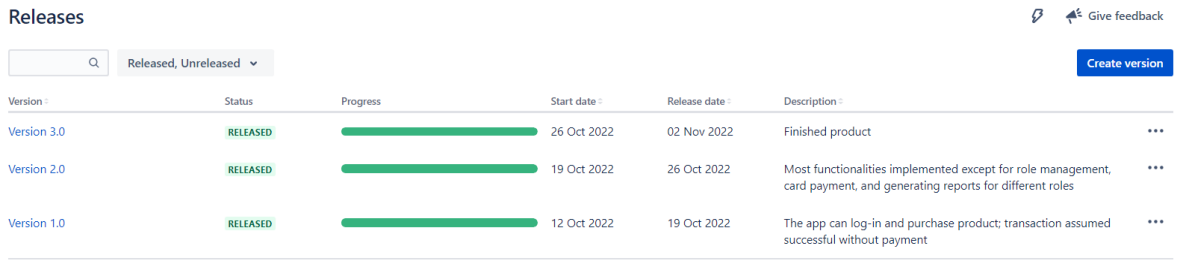
User stories

Refer to Sprint 1 report for the list of all user stories. Below only shows relevant user stories for Sprint 3.

ID	User story
7	As a user, I want to pay by cash in coins or notes. I want to receive my products and any changes with higher notes/coins prioritized.
Acceptance criteria: - User can input notes and coins of valid quantity (and reject wrong quantity). User is shown the remaining quantity and must keep providing inputs until enough money. - If available money in the machine cannot provide changes with given amount, user cannot proceed until they change to valid notes/coins. - User can cancel transaction at any moment before completing the transaction. - Upon successfully completing transaction, user receives products and changes prioritized with higher notes/coins.	
8	As a user, I want to pay by card. I want to be able to save my card details for next transaction.
Acceptance criteria: - User can input “cardholder name” and “credit card number” (card number always hidden with asterisks (*)). - System authenticates against given credit card list and notify errors if not matching. - User has the option to save the card details **after** successful transaction. The details are automatically filled in during next transaction. Same card is not prompted to be saved twice.	
12	As a seller or owner, I want to generate 2 reports: 1 on product details; 1 on selling status of the products.
Acceptance criteria: Seller or owner requests reports through the UI and is given a list of items that is current and accurate as well as a summary of the sales made so far.	
13	As a cashier or owner, I want to be able to update the quantity of available notes or coins in the machine.
Acceptance criteria: Cashier or owner can update quantity of available notes or coins.	
14	As a cashier or owner, I want to generate 2 reports: 1 on money availability; 1 on

	transaction history.
Acceptance criteria: Cashier or owner requests reports through the UI and is given money information and up-to-date transaction history.	
15	As an owner, I want to be able to add or remove Seller or Cashier or Owner user(s).
Acceptance criteria: Owner can add and remove roles to accounts. The system rejects the giving of a role if it leads to multiple roles for a person (remove first).	
16	As an owner, I want to generate 2 reports: 1 on user accounts; 1 on cancelled transactions history
Acceptance criteria: Owner requests reports through UI and gets up-to-date accounts with roles report, and cancelled transaction report (date, user, reason of cancellation).	

Releases


Releases

Give feedback

Released, Unreleased

Create version

Version	Status	Progress	Start date	Release date	Description
Version 3.0	RELEASED		26 Oct 2022	02 Nov 2022	Finished product
Version 2.0	RELEASED		19 Oct 2022	26 Oct 2022	Most functionalities implemented except for role management, card payment, and generating reports for different roles
Version 1.0	RELEASED		12 Oct 2022	19 Oct 2022	The app can log-in and purchase product; transaction assumed successful without payment

Sprint Backlog Item

Sprint	3
Start Date	Oct 26
End Date	Nov 2
Story points committed	38
Story points completed	38

No.	User Story	Priority	Story points	Assignee	Date	Status
SA2-29	7	Must	9	Noah	Oct.31	Complete
SA2-30	8	Must	8	Anh	Oct.30	Complete
SA2-44	12	Must	4	Noah	Nov.2	Complete
SA2-48	13	Must	1	Noah	Oct.31	Complete
SA2-49	14	Must	5	Faye	Oct.31	Complete
SA2-50	15	Must	5	Vivian	Oct.28	Complete
SA2-51	16	Must	6	Faye	Oct.31	Complete
SA2-63	Task derived from sprint 2			Faye	Oct.26	Complete
SA2-64	Task derived from sprint 2			Vivian	Oct.28	Complete
SA2-66	Bug reported on Oct.31			Vivian	Oct.31	Complete
SA2-70	Bug reported on Oct.31			Vivian	Oct.31	Complete
SA2-71	Bug reported on Oct.31			Vivian	Oct.31	Complete
SA2-65	Bug reported on Oct.31			Noah	Oct.31	Complete
SA2-68	Bug reported on Oct.31			Noah	Oct.31	Complete
SA2-69	Bug reported on Oct.31			Noah	Oct.31	Complete
SA2-67	Bug reported on Oct.31			Faye	Nov.1	Complete
SA2-72	Bug reported on Nov.1			Vivian	Nov.1	Complete
SA2-73	Bug reported on Nov.1			Faye	Nov.1	Complete
SA2-74	Bug reported on Nov.1			Faye	Nov.1	Complete
SA2-75	Bug reported on Nov.1			Vivian	Nov.1	Complete

SA2-76	Bug reported on Nov.1	Noah	Nov.2	Complete
SA2-77	Bug reported on Nov.1	Vivian	Nov.1	Complete

SA2 Sprint 3 26 Oct – 2 Nov (22 issues) 0 0 38 Complete sprint •••

~~SA2-30~~ Card payment	8	DONE	AD
~~SA2-51~~ Print 2 reports for owners (accounts & cancelled transactions)	6	DONE	FC
~~SA2-49~~ Print 2 reports for cashiers and owners (money availability & transaction history)	5	DONE	FC
~~SA2-29~~ Cash payment	9	DONE	NV
~~SA2-44~~ Print 2 reports for sellers and owners (product details & sell status)	4	DONE	NV
~~SA2-63~~ Fix timer to refresh all user activities		DONE	FC
~~SA2-50~~ Add/remove role access to user accounts	5	DONE	VH
~~SA2-64~~ List all cart items to checkout page		DONE	VH
~~SA2-48~~ Update available notes/coins in the machine	1	DONE	NV
~~SA2-74~~ Idle timer doesn't log user out		DONE	FC
~~SA2-73~~ Idle time warning happens even thought user is not idle		DONE	FC
~~SA2-67~~ Logging out adds 2 transactions to transaction database		DONE	FC
~~SA2-70~~ Buttons from previous role carry over		DONE	VH
~~SA2-66~~ User can use admin functionality without perms		DONE	VH
~~SA2-71~~ Text carries over between payment methods		DONE	VH
~~SA2-65~~ Users show multiple times in user report		DONE	NV
~~SA2-68~~ 100 denomination shows twice in report		DONE	NV
~~SA2-69~~ Cashier can print admin reports		DONE	NV
~~SA2-75~~ Changing item details doesn't update screen without logging out and back in		DONE	VH
~~SA2-72~~ Transactions added even though cart is empty		DONE	VH
~~SA2-77~~ Card number field stays when switching from card payment to cash payment		DONE	VH
~~SA2-76~~ Change doesn't update in SQL after purchase is made		DONE	NV

Scope change during sprint 3

2022-10-28	SA2-50	Add/remove role access to user accounts	Story	Estimate changed from 2 to 5	2 → 5

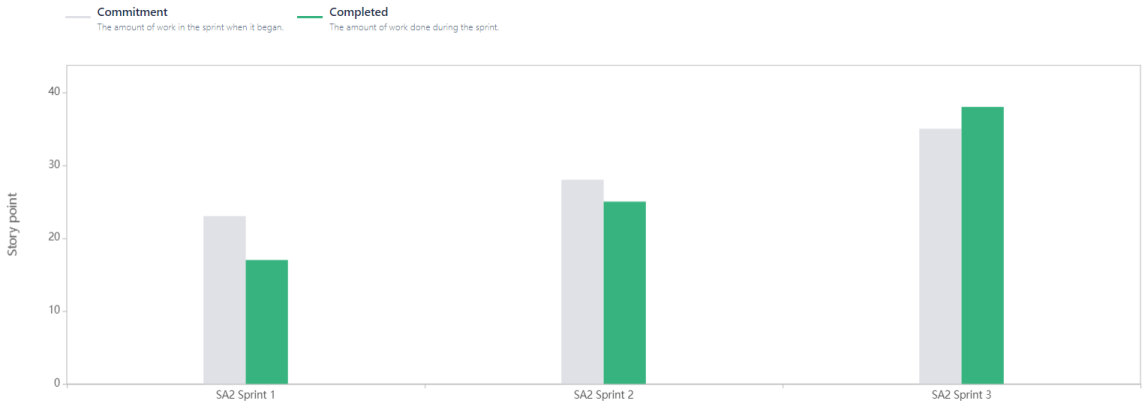
Burnup report

Burndown chart

Velocity report

Velocity report

> How to read this report



Sprint	Commitment	Completed
SA2 Sprint 1	23	17
SA2 Sprint 2	28	25
SA2 Sprint 3	35	38

Key Development events

GitHub

In addition to what we used for Sprint 2, we have only added a “how to” guide markdown file on GitHub.

Code release	
“How to” guide	A readme.md with user guide is included. Typical use cases for different user roles are outlined.

README.md

Vending Machine

To run the application, please use:

gradle run

Typical use cases

User

1. Login with username and password. If no account, redirect to sign-up page and create an account.
2. You can show products by category. Click on any product or manually enter its code, then enter the quantity you want. Press enter to add to cart (initial stock = 7).
3. Checkout. You can go back to add more items. Choose payment method.
4. For cash, input valid note/coin. Machine will feed you changes if necessary.
5. For card, enter valid details.
6. Confirm transaction should auto logout. If pay by card, you can choose to save card info or quit without saving.

Seller

Gradle

Build.gradle did not change from sprint 2. New Gradle commands or notable commands used are listed below.

gradle jacocoTestReport	Manually generates code coverage report for the test task. We don't usually run this command, since we configured this command to auto run after gradle test or gradle build (see sprint 2 report for more details).
	`PS C:\Users\Vivian Ha\Downloads\USYD\2022_s2\SOFT2412\assignment_2\cc23_g1_A2\A2> gradle jacocotestreport`
	
	`> Configure project :`
	`Project : => 'com.example.a2' Java module`
	
	`BUILD SUCCESSFUL in 6s`
	`5 actionable tasks: 3 executed, 2 up-to-date`

gradle test --info	This command is extensively used in this sprint for unit test debugging. It will print the error stack trace, and which line the assertion errors occurred. AssertEqual is particularly useful because it will show us the actual output.
	`databaseTesting STANDARD_OUT`
	`DONE!`
	
	`databaseTesting > getLastFiveTransactions() FAILED`
	`org.opentest4j.AssertionFailedError: expected: <1> but was: <2>`
	`at app/org.junit.jupiter.api@5.8.2/org.junit.jupiter.api.AssertionUtils.fail(AssertionUtils.java:55)`
	`at app/org.junit.jupiter.api@5.8.2/org.junit.jupiter.api.AssertionUtils.failNotEqual(AssertionUtils.java:62)`
	`at app/org.junit.jupiter.api@5.8.2/org.junit.jupiter.api.AssertEquals.assertEquals(AssertEquals.java:150)`
	`at app/org.junit.jupiter.api@5.8.2/org.junit.jupiter.api.AssertEquals.assertEquals(AssertEquals.java:145)`
	`at app/org.junit.jupiter.api@5.8.2/org.junit.jupiter.api.Assertions.assertEquals(Assertions.java:527)`
	`at app/com.example.a2@1.0-SNAPSHOT/com.example.a2.databaseTesting.getLastFiveTransactions(databaseTesting.java:280)`
	Fig. Sample Failed tests

JUnit

We have written unit tests for all backend logic with most branches covered in this sprint. The final overall coverage (excluding GUI view) is 77%.

databaseTesting

all > com.example.a2 > databaseTesting

18	0	0	0.810s
tests	failures	ignored	duration

100% successful

Tests | Standard output

Test	Duration	Result
addRemoveProductTest()	0.013s	passed
addRemoveUserTest()	0.036s	passed
addgetTransactionTest()	0.032s	passed
getCostTest()	0.023s	passed
getCurrencyTest()	0.004s	passed
getItemDetailsTest()	0.113s	passed
getItemSummaryTest()	0.118s	passed
getLastFiveTransactions()	0.041s	passed
getPasswordTest()	0.048s	passed
getUserIDTest()	0.024s	passed
getUsers()	0.025s	passed
getUsersReportTest()	0.008s	passed
productInitTest()	0.011s	passed
savegetCCInfoTest()	0.008s	passed
testValidCredit()	0.160s	passed
transactionHistoryTest()	0.121s	passed
updateCurrencyAndReportTest()	0.014s	passed
updateSoldTest()	0.011s	passed

Fig. Test cases duration and result for database (DBManage class)

vendingMachineTest

all > com.example.a2 > vendingMachineTest

8	0	0	0.493s
tests	failures	ignored	duration

100% successful

Tests | Standard output | Standard error

Test	Duration	Result
testAddToCart()	0.001s	passed
testCheckInput()	0.001s	passed
testGetTotalCost()	0.026s	passed
testInitialProducts()	0.002s	passed
testMakeCashPurcase()	0.379s	passed
testShowProductCategorized()	0s	passed
testUpdateProduct()	0.063s	passed
testUpdateUserRole()	0.021s	passed

Fig. Test cases duration and result for logic (VendingMachine class)

A2 > com.example.a2

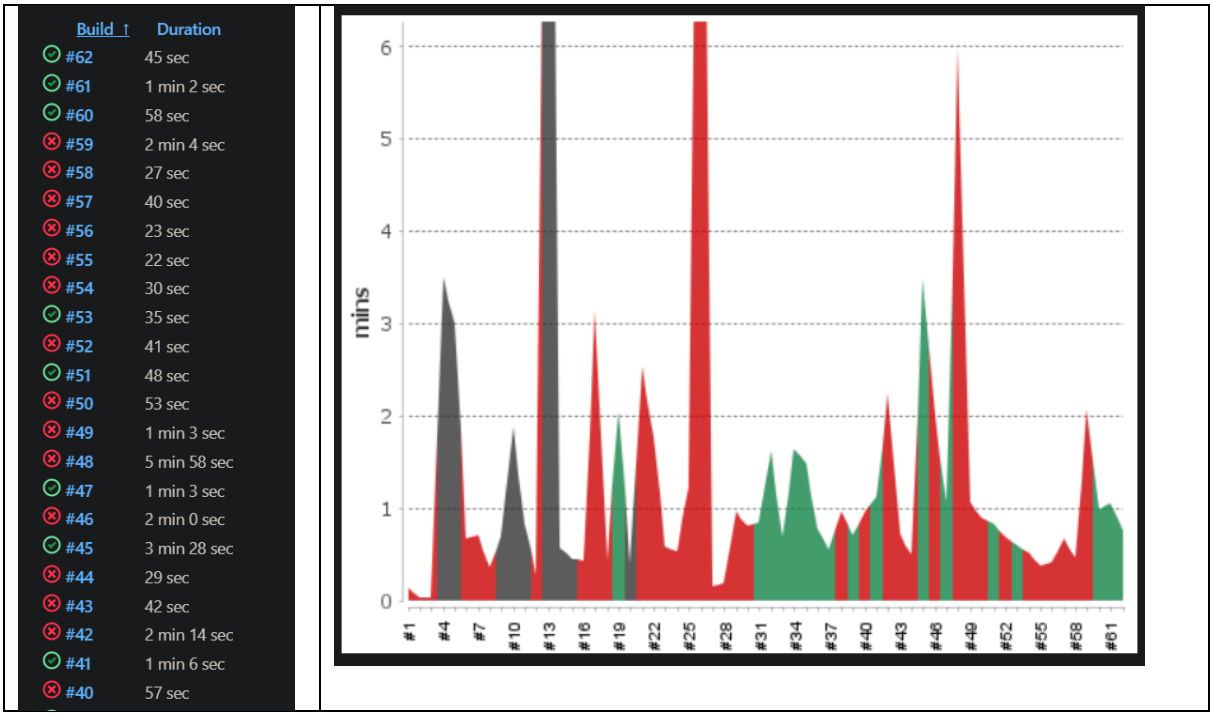
com.example.a2

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
DBManage		77%		61%	50	102	187	707	0	31	0	1
HelloApplication		1%		n/a	12	13	38	39	12	13	0	1
VendingMachine		94%		89%	18	94	31	261	7	27	0	1
Sys		46%		n/a	17	23	20	37	17	23	0	1
Userbase		45%		0%	6	9	12	23	2	5	0	1
VendingMachine.new TimerTask().{...}		0%		n/a	2	2	8	8	2	2	1	1
Transaction		42%		n/a	6	8	12	21	6	8	0	1
Currency		0%		n/a	1	1	4	4	1	1	1	1
Owner		92%		77%	3	9	3	18	1	3	0	1
User		89%		n/a	1	5	1	11	1	5	0	1
Seller		75%		n/a	1	2	1	2	1	2	0	1
Cashier		75%		n/a	1	2	1	2	1	2	0	1
Total	1,000 of 4,502	77%	77 of 283	72%	118	270	317	1,132	51	122	2	12

Fig. Native Jacoco report output (excluding GUI [com.example.a2.view])

Jenkins

Build report history	Sprint 3 started from build #33



Build ↑	Duration
#62	45 sec
#61	1 min 2 sec
#60	58 sec
#59	2 min 4 sec
#58	27 sec
#57	40 sec
#56	23 sec
#55	22 sec
#54	30 sec
#53	35 sec
#52	41 sec
#51	48 sec
#50	53 sec
#49	1 min 3 sec
#48	5 min 58 sec
#47	1 min 3 sec
#46	2 min 0 sec
#45	3 min 28 sec
#44	29 sec
#43	42 sec
#42	2 min 14 sec
#41	1 min 6 sec
#40	57 sec

Overview of Application

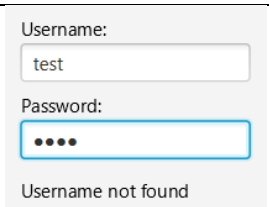
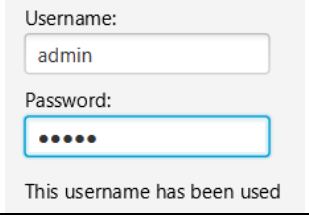
The group mainly communicates via messages. Stand-up meetings are held 2~3 times weekly on average, and on other days each member would communicate and records their progress by answering the 3 questions. The group uses Jira for project planning. User stories are set by Scrum master and Product owner and assigned to individuals at the start of the sprint. All developers report extra tasks and bugs when necessary. GitHub is used for version control, where each developer has its own branch to develop features separately. The developer who is responsible for writing a function would also test that function in JUnit. Minor bugs that could be fixed promptly are tracked using GitHub Issues. A tagged working product is released at the end of each sprint and is tracked with GitHub Release. Jenkins ensures the deployed app builds successfully.

<u>GUI Testing with various inputs</u>

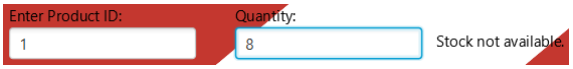
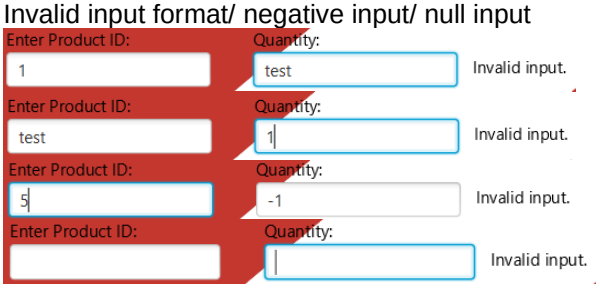
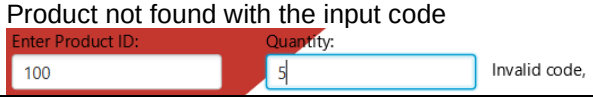
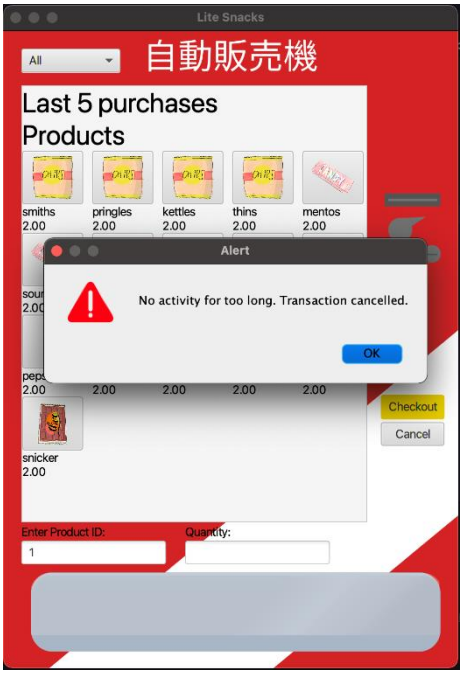
Backend logic is tested via JUnit. Other functional requirements that cannot be tested via unit tests (e.g., GUI events in the ControlHandler class) are tested manually by developers. Invalid inputs are then handled individually, predominantly in ControlHandler class. Appropriate messages are then shown on UI. All test cases shown in the logs below.

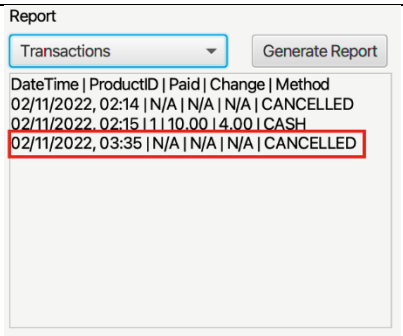
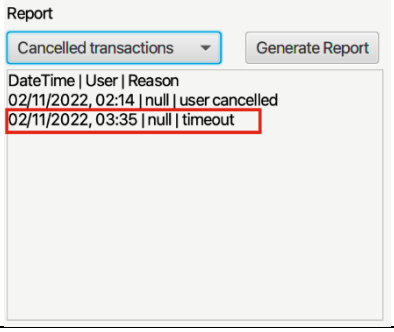
Login page

Login	User does not exist

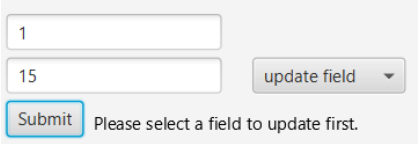
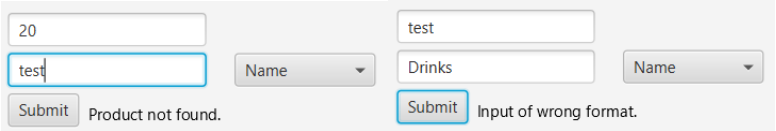
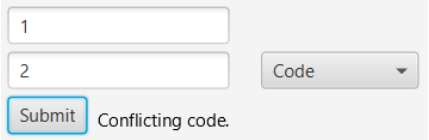
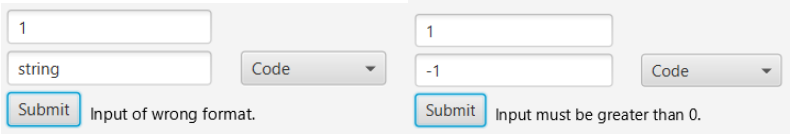
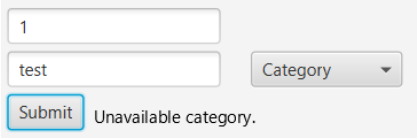
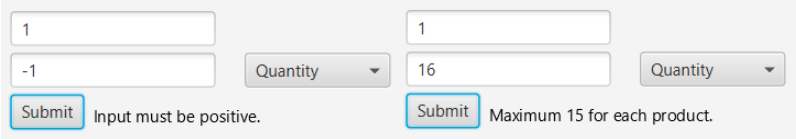
	
Sign up	Username already exists 

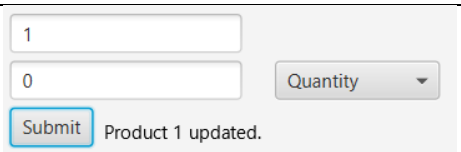
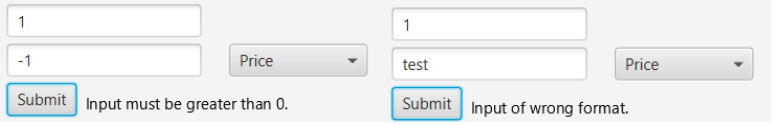
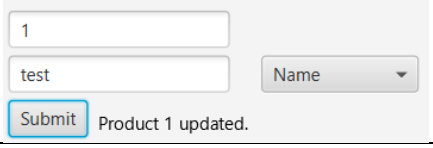
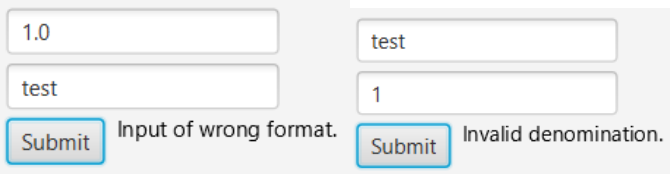
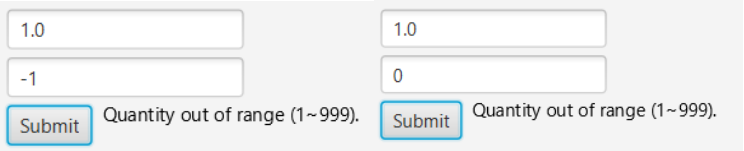
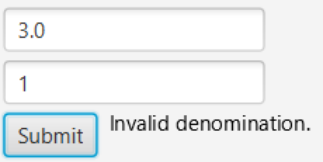
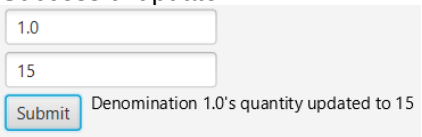
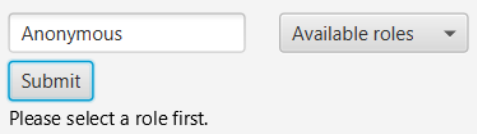
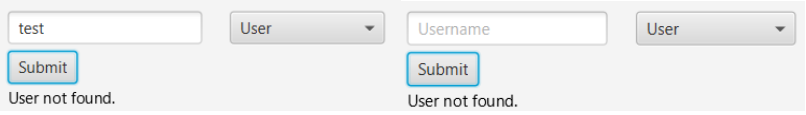
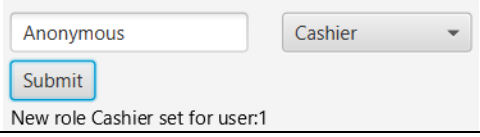
Home page

Add product to cart	Exceed stock  Invalid input format/ negative input/ null input  Product not found with the input code 
After closing idle timer alert window, user is logged out, automatically navigated to home page, and a cancelled transaction is logged into the system.	Timer alert window  Cancelled transaction added to a table of transactions

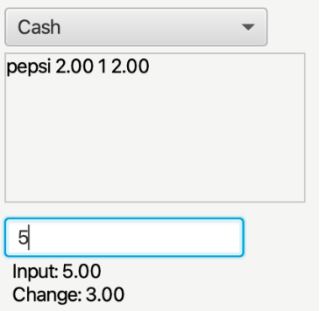
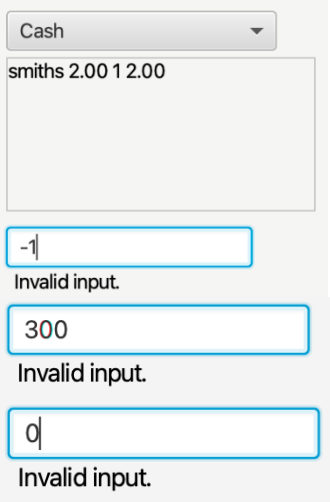
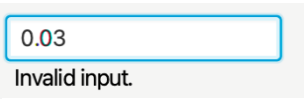
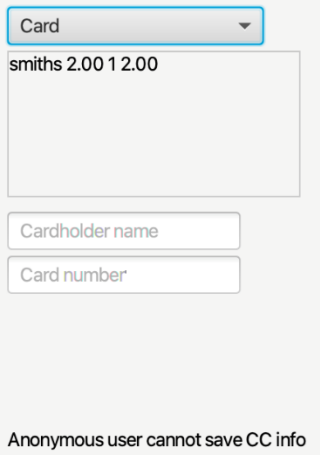
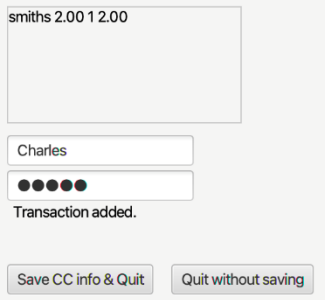
	 Cancelled transaction can be seen in cancelled transactions report with reason as timeout 

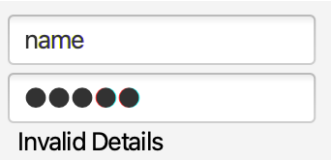
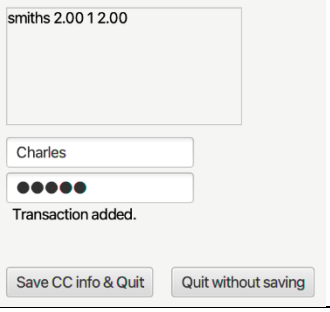
Admin page (including owner, cashier, seller)

Update product	No options chosen  Product not found with the input code/ invalid code format  Change code, but code already used by other products  Invalid input format/ negative input  Category not in list  Negative quantity/ quantity exceeds maximum  Boundary case (0)

	 Negative price/ boundary case (0)/ wrong format  Successful update 
Update notes/coins	Invalid input format  Negative/ boundary case (0)  Denomination not in list  Successful update 
Update user roles	No option chosen  User does not exist/ null input  Successful update 

Payment page

Cash payment method message shows correct information	Shows correct input and change amount when denominator is valid  Invalid denominator inputs  
Does not allow anonymous users to save credit card information	Logged in as anonymous user  Logged in as non-anonymous user 

Machine does not allow cards not in file credit_cards.json	Cardholder name and carf number not in system  Valid cardholder name and card number 

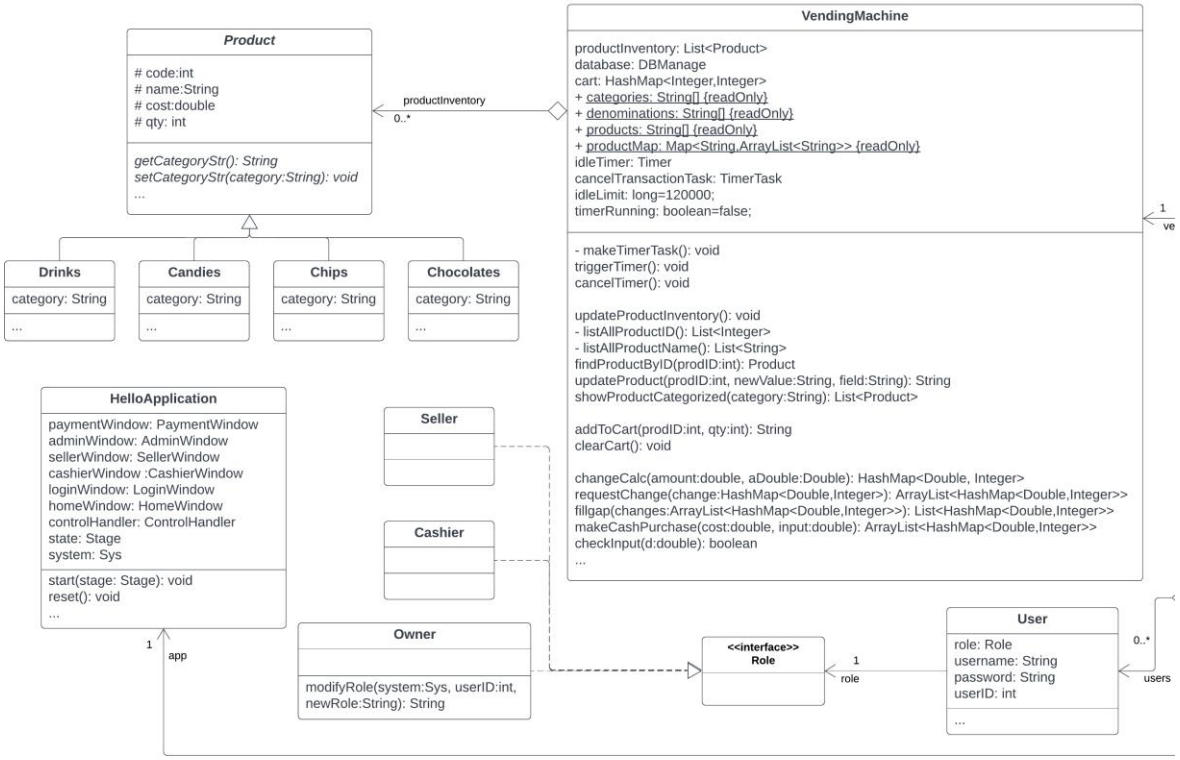
Integrated and system testing was done via developer walk-throughs. It was difficult to do these tests in any other way as both these components relied heavily on the UI. These walk-throughs were done with the goal of covering all common scenarios – including the following:

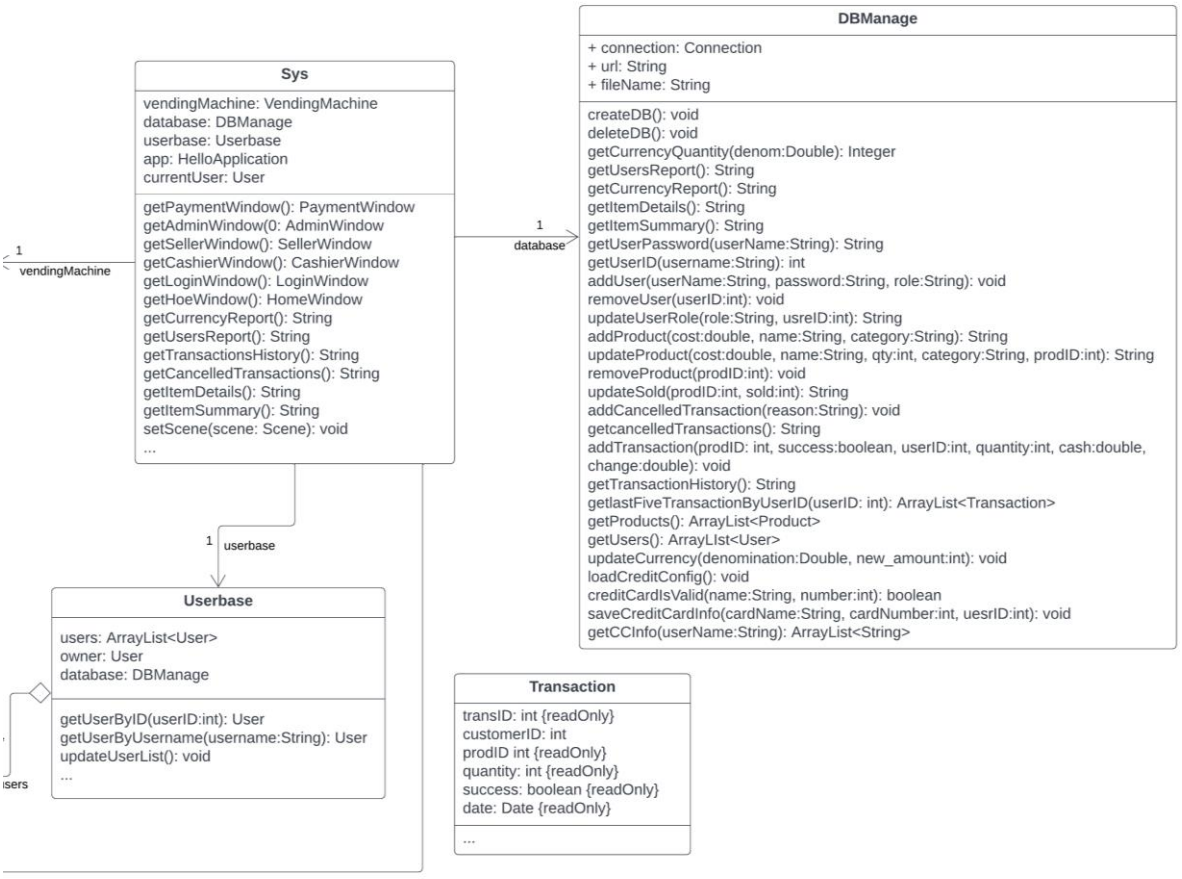
- Admin logging into a new database, updating all attributes of products, checking the generated report, making a purchase with cash and card and checking the generated report again.
- Admin navigating to admin page, updating a product, and navigating back to the home page to see if the home page is showing the correct update.
- New user signing up, admin signing in and making them a cashier, new user signing back in and making a purchase with card, saving the details, user checking the cashier reports and then trying to make another purchase with the saved card details.
- New user signing up, admin signing in and making them a seller, new user signing back in as seller, modifying product details, going back to vending machine UI and checking that the visual representation of products accurately reflect changes.
- Admin revoking a Cashier C their cashier role to a User role, logging in C and checking they do not have access to cashier functions.
- Admin revoking a Seller S their seller role to a User role, logging in S and checking they do not have access to seller functions.

Within each walkthrough, the developer ensured that the application's behavior and output was in line with what was expected from the input. When it was not, new bugs were added to Jira to be addressed. Once a round of bugs was complete, the exact same walkthrough was done again.

Class diagram

The diagram excludes the GUI portion of the application.





Final client showcase

All core features were showcased successfully. Client reports two desirable changes:

- Client wants a user to be able to see the changes in terms of accurate denominations shown on payment page
- Client accepts our assumption, but initially appears to want a cashier/owner to update quantity of notes/coins to 0.

Retrospective

Attendee	All present
Date	Nov.2
What went wrong?	- Poor modularization of classes, ultimately clashing in code changes. - Lack of manual testing, causing a massive workload before end of sprint - Due to lack of defensive programming and unit testing in the first 2 sprints, many bugs are reported in this sprint, especially in the end.
What went right?	- Good communication. - No procrastination. Issues are fixed promptly within 1 day. - Needn't deal with overlapping of work due to everyone keeping track of individual tasks and informing others immediately. - Despite the large amount of work, we have good time management this sprint. Functionalities are finished around 2 days before the last day, leaving enough time for final testing.