Table of Contents

1. Introduction

Welcome to **WalletCLi**!

WalletCLi is a text-based (Command Line Interface) expenses/loan application that caters to NUS students and staff who prefer to use a desktop application for managing their expenses and loans.

WalletCLi allows its users to create their daily expenses and loans, and enables easy creation, editing and deletion. Users can also pre-set their budget, and **WalletCLi** will automatically track your current expenses to ensure that its users expenses stay within their stated budget. Expenses and loans can be efficiently managed via our intuitive category system.

WalletCLi is optimized for those who prefer to work with a Command Line Interface (CLi) and/or are learning to work more efficiently with CLi tools. Additionally, unlike traditional expenses/wallet applications, **WalletCli** utilizes minimal resources on the user's machine while still allowing users to manage their expenses and keep track of their loans swiftly and efficiently.

2. About this Developer Guide

This developer guide provides a detailed documentation on the implementation of all the various features **WalletCLi** offers. To navigate between the different sections, you could use the table of contents above.

For ease of communication, this document will refer to expenses/loans/contacts that you might add into the planner as *data*.

Additionally, throughout this developer guide, there will be various icons used as described below.

💡	This is a tip. Follow these suggested tips to make your life much simpler when using **WalletCLi**!
ℹ	This is a note. These are things for you to take note of when using **WalletCLi**.
!	This is a sign-post dictating important information. These are information that you will surely need to know to use **WalletCLi** efficiently.
🔥	This is a sign-post informing caution. Please take note of these items and exercise some care.
⚠	This is a rule. Ensure that you follow these rules to ensure proper usage of **WalletCLi**.

3. Setting up

This section describes the procedures for setting up **WalletCLi**.

3.1 **Prerequisites**

1. JDK `11` or later
2. IntelliJ IDE

ℹ	IntelliJ by default has Gradle installed. Do not disable them. If you have disabled them, go to `File` > `Settings` > `Plugins` to re-enable them.

3.2 **Setting up the project in your computer**

1. Fork this repo, and clone the fork to your computer.

2. Open IntelliJ (if you are not in the welcome screen, click `File` > `Close Project` to close the existing project dialog first).

3. Set up the correct JDK version for Gradle.

i. Click `Configure` > `Project Defaults` > `Project Structure`

ii. Click `New…` and find the directory of the JDK.

4. Click `Import Project.`

5. Locate the `build.gradle` file and select it. Click `OK.`

6. Click `Open as Project.`

7. Click `OK` to accept the default settings.

8. Run the `wallet.Main` class (right-click the `Main` class and click `Run Main.main()`) and try executing a few commands.

9. Run all the tests (right-click the `test` folder, and click `Run 'All Tests'`) and ensure that they pass.

10. Open the `StorageFile` file and check for any code errors.

11. Open a console and run the command `gradlew processResources` (Mac/Linux: `./gradlew processResources`). It should finish with `BUILD SUCCESSFUL` message.
    This will generate all resources required by the application and tests.

12. Open `Main.java` and check for any code errors.

. Due to an ongoing issue with some of the newer versions of IntelliJ, code errors may be detected even if the project can be built and run successfully.

i. To resolve this, place your cursor over any of the code section highlighted in red. Press `ALT`+`ENTER`, and select `Add '--add-modules=…' to module compiler options` for each error.

3.3 **Verifying Setup**

1. Run the `Wallet.Main` and try a few commands.
2. Run the tests to ensure they all pass.

4. Design

5. Implementation

6. Documentation

7. Testing

8. Dev ops

Appendix A - Product Scope

Target user profile:

- NUS students
- has a need to organise and manage a significant number of expenses
- prefers desktop apps over other types
- prefers having a completely offline wallet application
- can type fast
- prefers typing over mouse input
- is reasonably comfortable using CLI apps

Value proposition: manage expenses, budgets and loans faster than a typical mouse/GUI driven app and caters to users who prefer an offline solution due to the current technology climate where information privacy/data privacy/data protection has become an uncertainty.

Appendix B - User Stories

This section describes the user stories that the **WalletCLi** developer team ideated. These user stories were used to decide on the desired features for **WalletCLi**. The user stories are categorized into different priorities for implementation:

- High (must have) - ***
- Medium (nice to have) - **
- Low (unlikely to have) - *

Priority	**As a …**	**I can …**	**So that I …**
***	user	Add loans by specifying the amount and contact	Can know who owes me money
***	Forgetful user	Update status of loans by indicating it is being settled	Know which loans are settled
***	user	Delete entries from the lost of loans	Can remove unwanted or old entries from the list
***	user	Record my expenses	Can track my daily expenses
***	user	Delete my expenses record	Can remove records that were added by mistake
***	Organised user	See my expenses of each category	Can plan my budget for each expense category
***	user	Set category expenses	Know which category have I spent most of my money on
***	user	View my expenses according to category	Can know how much I spend on each category per month
***	Power user	Export my expenses record	Can keep a record of my expenses
***	user	Update my expenses record	Can consistently keep track of my money
***	user	Add contacts	Can keep track of their personal information
***	user	Edit my contacts	Can update any changes to their information
***	user	Delete my contacts	Can remove contacts that were added by mistake or no longer in use
***	New user	View the help section	Know how to use the expenses app in command line interface

***	user	View the total sum of money that I have	Can track how much money I have left in my wallet
***	user	view color coded categories	Can easily view the different types of categories
***	user	Choose to view my entire expenses and loans or view specific loans and expenses by date	Can view my expenses and loans history
**	Forgetful user	Set reminders to pay my bills	Will not forget to pay the bills
**	Lazy user	Add recurring expenses	Can have expenses added monthly/daily/yearly automatically on the application
**	user	Use the app to take photos of receipts	Can input my expenses later
**	user	Start new sessions of ‘WallerCLi’ that have their own saved states	Can keep track of each one individually and differently
**	user	Convert currency	Do not need to calculate it manually
**	user	Set expenses goals by specifying how much i want to spend per week or month	Will not overspend
**	user	Redo command	Can add multiple of the same entry
**	user	Undo command	Can go back to the previous state if there are mistakes.
**	user	View command history	Can know the previously executed commands.
*	user	Receive cashbacks when i save money	Can lower my expenses
*	user	Show off to my friends how much I saved per month on social media	Can encourage people to do so too
*	user	Get notifications from the application	Can be notified of important expenses or if i don't have much money left
*	user	Sync my record to my bank account	Can have more convenience
*	user	Make direct transactions when shopping online	Do not need to actively use a card for online transaction

Appendix C - Use Cases

This section describes the Use Cases for some of our implemented features. (For all use cases below, the System is `WalletCLi` and the Actor is the `user`, unless specified otherwise)

Use Case 1: Adding an expense

- **MSS:**

  1. User inputs `add expense` command with all required parameters.

  2. System adds the expense into the expense list.

  Use case ends.

- **Extensions:**

  1a. System detects an error in the given input.

    1a1. System outputs an error message.

    Use case ends.

  1b. System detects missing required parameters in the given input.

    1b1. System outputs an error message.

    Use case ends.

Use Case 2: Editing an expense

- **MSS:**

  1. User inputs `edit expense` command with ID of expense and new values.

  2. System modifies and updates the expense in the expense list.

  3. System outputs the edited expense with the updated values.

  Use case ends.

- **Extensions:**

  1a. System detects ID of expense is invalid.

    1a1. System outputs an error message.

    Use case ends.

  1b. System detects an error in the given input parameters.

    1b1. System outputs an error message.

    Use case ends.

  1c. System detects no parameters in given input

1c1. System outputs an error message.

Use case ends.

Use Case 3: Deleting an expense

- **MSS:**

  1. User inputs `delete expense` command with the ID of expense to delete.
  2. System deletes the expense and updates the expense list.
  3. System outputs the deleted expense with its values.

  Use case ends.

- **Extensions:**

  1a. System detects ID of expense is invalid.

  1a1. System outputs an error message.

  Use case ends.

  1b. System detects no parameters in given input.

  1b1. System outputs an error message.

  Use case ends.

Use Case 4: Listing all expenses

- **MSS:**

  1. User inputs `list expense` command.
  2. System outputs all expenses along with their values.

  Use case ends.

- **Extensions:**

  1a. System detects an error in the given input parameters.

  1a1. System outputs an error message.

  Use case ends.

  1b. System detects no parameters in given input.

  1b1. System outputs an error message.

  Use case ends.

Use Case 5: Setting a budget for the month

- **MSS:**

  1. User inputs `set budget` command with the required parameters.
  2. System sets the budget for the given month and outputs the new budget for the month.

Use case ends.

- **Extensions:**

  1a. System detects an error in the given input parameters.

  1a1. System outputs an error message.

  Use case ends.

  1b. System detects no parameters in given input.

  1b1. System outputs an error message.

  Use case ends.

Use Case 6: Adding a loan

- **MSS:**

  1. User inputs `add loan` command with all required parameters.

  2. System adds the loan into the loan list.

  Use case ends.

- **Extensions:**

  1a. System detects an error in the given input.

  1a1. System outputs an error message.

  Use case ends.

  1b. System detects missing required parameters in the given input.

  1b1. System outputs an error message.

  Use case ends.

Use Case 7: Editing a loan

- **MSS:**

  1. User inputs `edit loan` command with ID of loan and new values as parameters.

  2. System modifies and updates the loan in the loan list.

  3. System outputs the edited loan with the updated values.

  Use case ends.

- **Extensions:**

  1a. System detects ID of loan is invalid.

  1a1. System outputs an error message.

  Use case ends.

  1b. System detects an error in the given input parameters.

1b1. System outputs an error message.

Use case ends.

1c. System detects no parameters in given input.

1c1. System outputs an error message.

Use case ends.

Use Case 8: Deleting a loan

- **MSS:**

1. User inputs `delete loan` command with the ID of loan to delete.

2. System deletes the loan and updates the loan list.

3. System outputs the deleted loan with its values.

Use case ends.

- **Extensions:**

1a. System detects ID of loan is invalid.

1a1. System outputs an error message.

Use case ends.

1b. System detects no parameters in given input.

1b1. System outputs an error message.

Use case ends.

Use Case 9: Listing all loans

- **MSS:**

1. User inputs `list loan` command with the required parameters.

2. System outputs all loans along with their values.

Use case ends.

- **Extensions:**

1a. System detects an error in the given input parameters.

1a1. System outputs an error message.

Use case ends.

1b. System detects no parameters in given input.

1b1. System outputs an error message.

Use case ends.

Use Case 10: Setting time for auto reminder

- **MSS:**

1. User inputs `reminder set` command with the required parameters (eg. `Time-in-seconds`)

Example command: `reminder set 3600`

2. System outputs a string, indicating auto reminder is properly set with the appended value.

Use case ends.

- **Extensions:**

1a. System detects an error in the given input parameters.

1a1. System outputs an error message.

Use case ends.

1b. System detects no parameters in given input.

1b1. System outputs an error message.

Use case ends.

Use Case 11: Setting time for auto reminder

- **MSS:**

1. User inputs `reminder set` command with the required parameters(eg. `Time-in-seconds`)

Example command: `reminder set 3600`

2. System outputs a string, indicating auto reminder is properly set with the appended value.

Use case ends.

- **Extensions:**

1a. System detects an error in the given input parameters.

1a1. System outputs an error message.

Use case ends.

1b. System detects no parameters in given input.

1b1. System outputs an error message.

Use case ends.

Use Case 12: Turning off auto reminder

- **MSS:**

1. User inputs `reminder off` command.

2. System outputs a string, indicating auto reminder is turned off.

Use case ends.

- **Extensions:**

    1a. System detects an error if auto reminder is already turned off.

        1a1. System outputs an error message.

        Use case ends.

Use Case 13: Turning on auto reminder

- **MSS:**

    1. User inputs `reminder on` command.

    2. System outputs a string, indicating auto reminder is turned on..

    Use case ends.

- **Extensions:**

    1a. System detects an error if auto reminder is already turned on.

        1a1. System outputs an error message.

        Use case ends.

Use Case 14: Undo commands

- **MSS:**

    1. User inputs `undo` command.

    2. System outputs a string, indicating command has been undone.

    Use case ends.

- **Extensions:**

    1a. System detects an error if there are no previous commands.

        1a1. System outputs an error message.

        Use case ends.

Use Case 15: Redo commands

- **MSS:**

    1. User inputs `redo` command.

    2. System outputs a string, indicating command has been redone.

    Use case ends.

- **Extensions:**

    1a. System detects an error if there are no commands after the current state.

        1a1. System outputs an error message.

        Use case ends.

Use Case 16: View command history

- **MSS:**

  1. User inputs `history` command.

  2. System outputs a history of commands (buffer size: 10)

  Use case ends.

Appendix D - Non-functional Requirements

This section describes the non functional requirements of **WalletCLi**.

1. The application should work on any mainstream OS as long as it has Java 11 or higher installed.
2. The application should work on both 32-bit and 64-bit environments.
3. The application should be able to hold up to over a hundred entries of expenses, loans and contacts without a noticeable sluggishness in performance for typical usage.
4. A user with above average typing speed for regular English text (i.e. not code, not system admin commands) should be able to accomplish most of the tasks faster using commands than using the mouse.
5. The system should respond relatively quickly to user commands so as to not make the user wait around; this is an advantage of using **WalletCLi**.
6. The system should take up relatively little space on the local machine so as to cater to all users and OS.
7. The system should be easy to use, intuitive and simple, such that any student regardless of past experience with wallet/expenses application is able to use it.
8. The system should be flexible to allow all kinds of expenses that target users might have.
9. The data should be encrypted to prevent private data such as contact information from being accessed.
10. Application should work even without any Internet connection

Appendix E - Glossary

This section further explains some terms/words used in **WalletCLi.**

1. Data: expenses/loans/contacts that you might add into the application.
2. MSS: Main Success Scenario (MSS) describes the most straightforward interaction for a given use case, which assumes that nothing goes wrong. This is also called the Basic Course of Action or the Main Flow of Events of a use case.
3. Use case: a specific situation in which a product or service could potentially be used.

Appendix F - Instructions for manual testing