Joshua Seet - Project Portfolio

Introduction

About Me.

[github]

Hi! My name is Joshua.

I'm currently a second year SoC (School of Computing) student in NUS majoring in Computer Science.

This write-up documents the contributions I made towards the development of FinSec, a project which my team and I undertook for the module CS2103T. Individually, I have benefited in no small measure, as working on the project has significantly enhanced my knowledge and ability to become a more competent software engineer.

PROJECT: FinSec

The description of the project.

FinSec is a useful application that was created as an aid to financial secretaries of any organisation. It is a Command Line Interface (CLI) based tool to cater to computing students who are highly adept at typing but also provides a Graphical User Interface (GUI) interface for users to easily view interact with FinSec.

FinSec comprises carefully evaluated and selected features that are tailored to be relevant to the specific needs of our target audience. All of FinSec's features and implementations are clearly and exhaustively documented in written guides for users and developers.

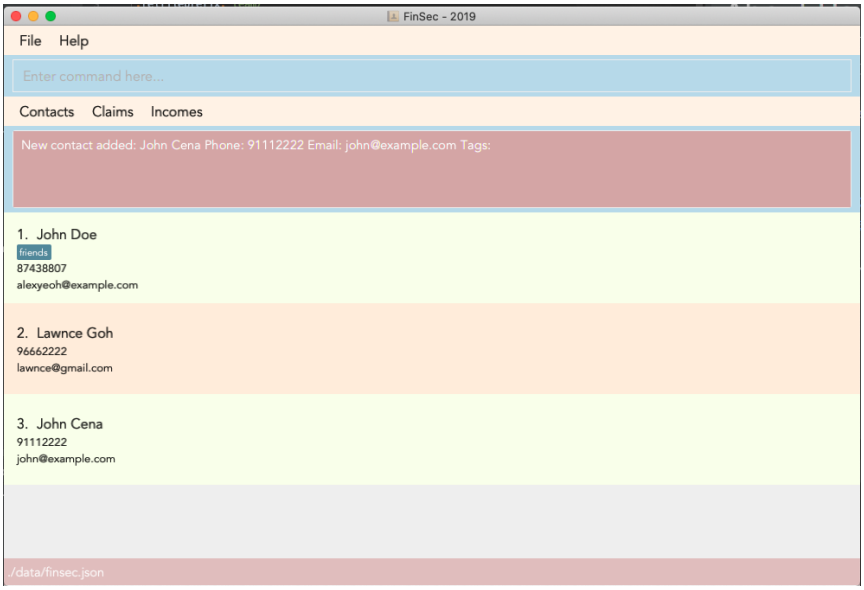

Figure 1. A view of FinSec

Role

The role I played in the team.

My primary role in the project was to create the prototype for the Shortcut feature and the Income feature, and develop them respectively.

I was also in charge of Testing and UI, ensuring that the coverall percentage does not fall too low and the UI/UX of FinSec is up to standards.

Summary of contributions

This section contains a summary of the coding, documentation, and other contributions I made towards the project.

- **Major enhancement**: Added the **Shortcut Feature**
  - What it does: The shortcut feature enables FinSec to convert an unknown command into a shortcut, which becomes a valid command for future use.
  - Justification: This feature significantly enhances user experience of FinSec as it allows users to create their own manual shortcuts for commands. Our target audience, Financial Secretaries, would find their tasks (typically data entry) much aided by FinSec which removes the need to type in the whole command, such as "add_claim" for every entry. Instead, they can, using FinSec, create an alias such as "ac" for the command "add_claim". This would avoid unnecessary duplication of commands, thereby raising efficiency.

- Implementation: For this to work, the flow of command execution in FinSec has to be altered. Executing an unknown command that FinSec does not recognize changes the command execution pathway and creates a new command that FinSec will recognise. This is facilitated by the use of a TreeMap in the parser which stores a list of all available commands and their actions as key-value pairs.
- Highlights: This feature enhances the entire software model to be more Object Oriented since through this implementation, commands are created as objects. This is different from the previous implementation of commands in the Address Book 3 (AB3). With a higher level of abstraction, future features would be easier for implementation. However, the implementation of this feature is challenging because it requires key changes to the flow of command execution of FinSec. After implementing this feature, without proper communication, my team members would have difficulty with the sudden change in the flow of the program. This was easily overcome with proper communication with my team members and more specific documentation in the java docs.

- **Minor enhancement**: Added the **Income feature**
  - What it does: This feature allows users (typically Financial Secretaries) to enter Income transactions into the FinSec application and track their income entries.
  - Justification: The `income feature` is a vital feature to the FinSec application since that the majority of organisations and clubs have incomes to keep track of. It allows Financial Secretaries to keep proper records of the organisation and to keep track of its financial status.
  - Implementation: The Person class in Address Book 3 (AB3) is remodelled to resemble an income entry (with additional attributes of Description, Amount, Date).
  - Highlights: The implementation of this feature is fairly straightforward since that it is a direct adaptation of the Person class.
- **Minor enhancement**: Added the **Autocorrect Suggestions feature**
  - What it does: This feature gives users suggestions in a drop-down list as they are typing their commands into FinSec. The suggestions given corresponds with what the user is currently typing. Currently, the possible suggestions are the default commands available for the user to use. Whenever the user adds a shortcut (from the `shortcut feature`) or a contact, it will add on (in real-time) to the suggestions list that could be displayed to the user (depending on what the user types).
  - Justification: The drop-down list that this feature provides greatly enhances the user experience when using the application. With this feature, the user does not have to perpetually refer to the User Guide for the list of available commands. This is especially useful for FinSec, an application with 20 commands.
  - Implementation: An additional class AutocorrectTextField is created to cater for the add-ons to the TextField class. By loading the updated Set of Strings in that class, the new dictionary of suggestions is available to be displayed to the user when he types.
  - Highlights: The implementation of this feature is slightly difficult since that it deals with majority of UI which I am not very familiar with. However, tutorials online help me to learn more about the different components such as Listeners. I also made minor enhancements to this feature by adding attributes (such as Name) into the claim suggestions for better

usability for the user.

- **Code contributed**: [All commits] [Project Code Dashboard]
- **Other contributions**
  - Project management
    - I managed the release of version 1.4 of FinSec on GitHub.
  - Enhancements to existing features
    - I refactored and optimised some reusable code in AB3 to increase code quality and make the program more Object Oriented : #150, #345, #340
  - Documentation
    - I made improvements to the Developer's Guide and did the Use Cases section: #350, #181, #142
  - Community
    - I conducted reviews on other team's PR to give suggestions and constructive comments: #31
    - I also reported bugs and potential flaws in other teams' project to help prepare them for the demo: #1, #10, #8
  - Tools
    Created Google Docs for team planning

Contributions to the User Guide

> *Given below are sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users.*

Creating a shortcut

Accidentally typed a command that is not in FinSec? Don't worry! FinSec will recognise that it is an unknown command and is smart enough to make that unknown entry into a shortcut!

Upon entering an unknown command, FinSec will prompt you as to whether you would like that as a new shortcut or if it was just a mistake.

Choice 1 : You can enter the keyword `n` to continue as per usual.

Choice 2 : You can enter any `existing commands` to map your previous entry to it!

If you have made a mistake and entered the wrong command, **Choice 1** would allow you to continue.

Example of Choice 1 :

User : add_conagtact

FinSec: Create shortcut? To which command? If no, type "n"

User : n

Result:

No shortcut is created. Continue using FinSec as per normal!

Figure 3.11.1.1, Figure 3.11.1.2 and Figure 3.11.1.3 shows what you can expect to see after typing in the example: `add_conagtact`.

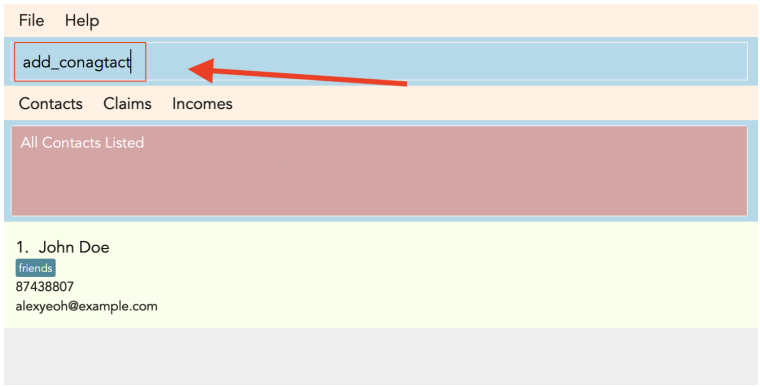

Figure 3.11.1.1: When you enter an accidental typo

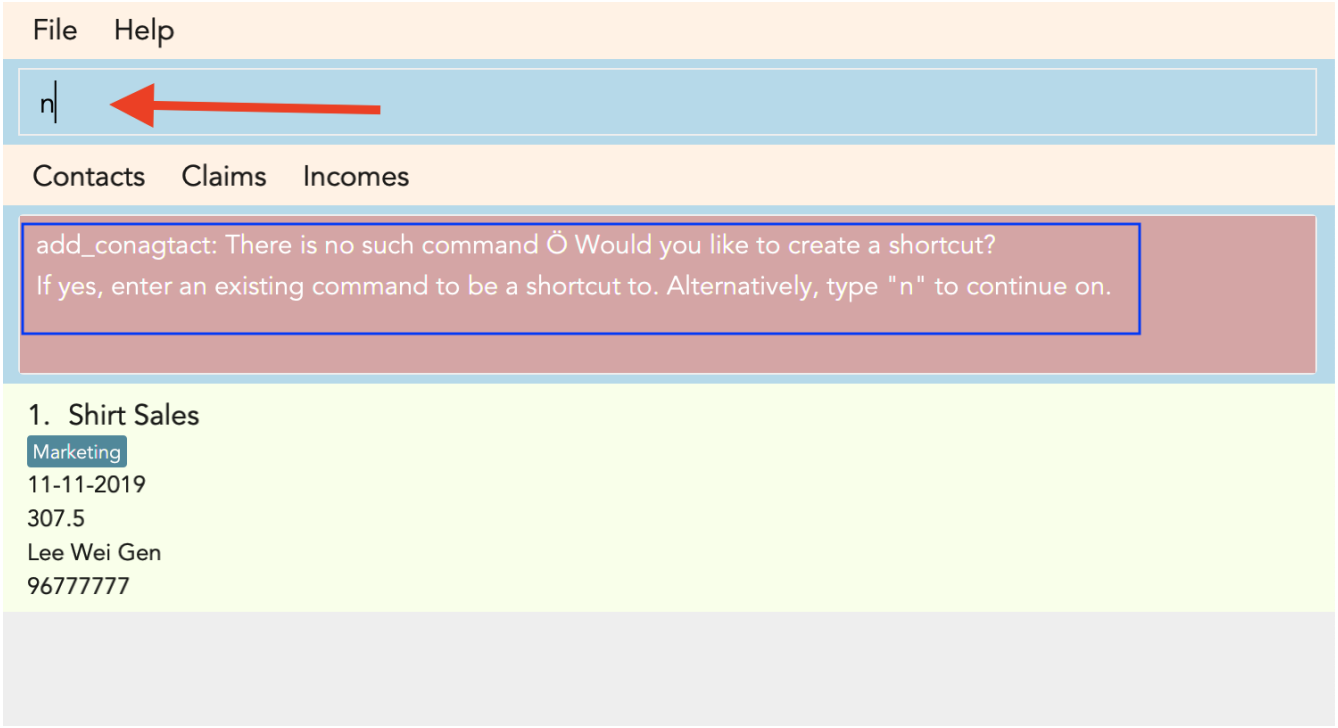

Figure 3.11.1.2: FinSec will ask you if you want to create a shortcut (in blue). If you do not wish to create one, enter the command "n" as shown above.

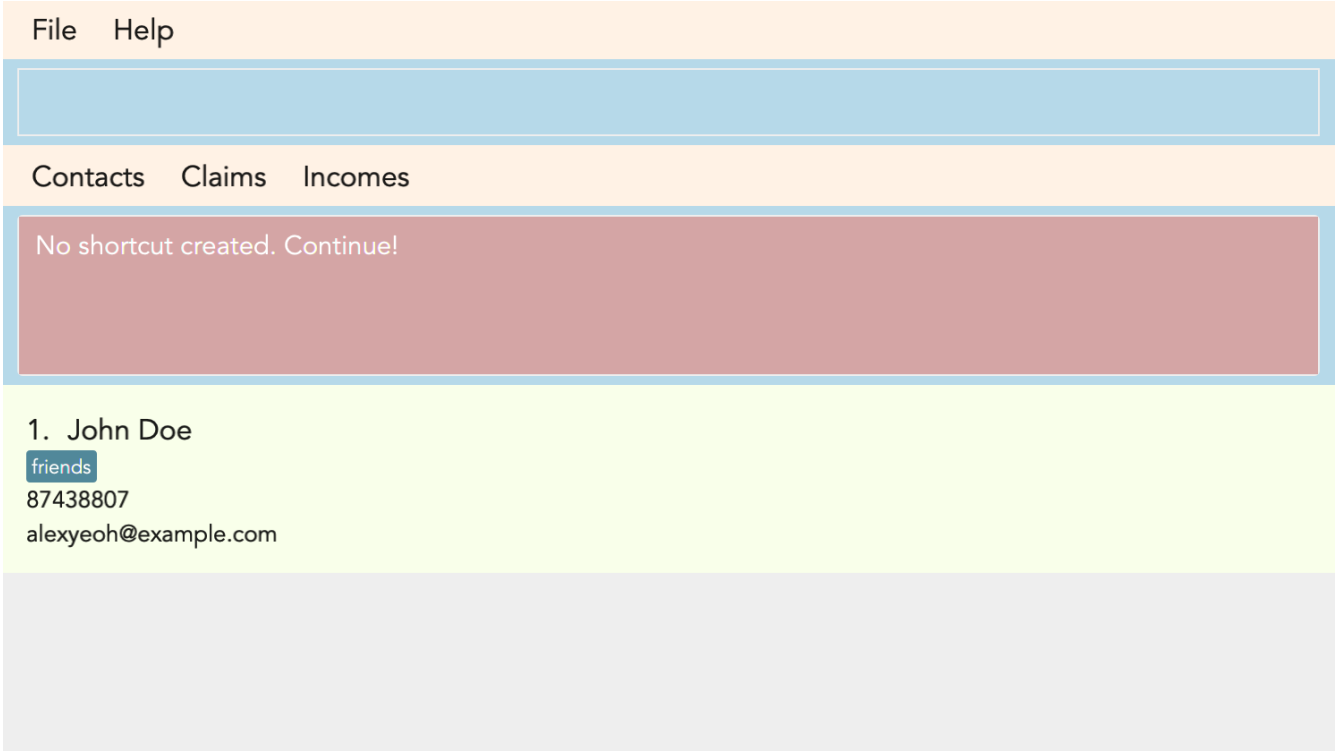

Figure 3.11.1.3: Entering 'n' will allow you to continue.

If you have wish to create a shortcut, **Choice 2** would allow you to create a shortcut to a command.

Example of Choice 2 :

User : ai

FinSec: Create shortcut? To which command? If no, type "n"

User: add_income

FinSec: New shortcut created! ai to add_income

Result:

A new shortcut is created for you! From now on, `ai` can be used as a substitute for `add_income`!

Figure 3.10.2.1, Figure 3.10.2.2 and Figure 3.10.2.3 shows what you can expect to see when you want to create a shortcut `ai` as shown in the example.

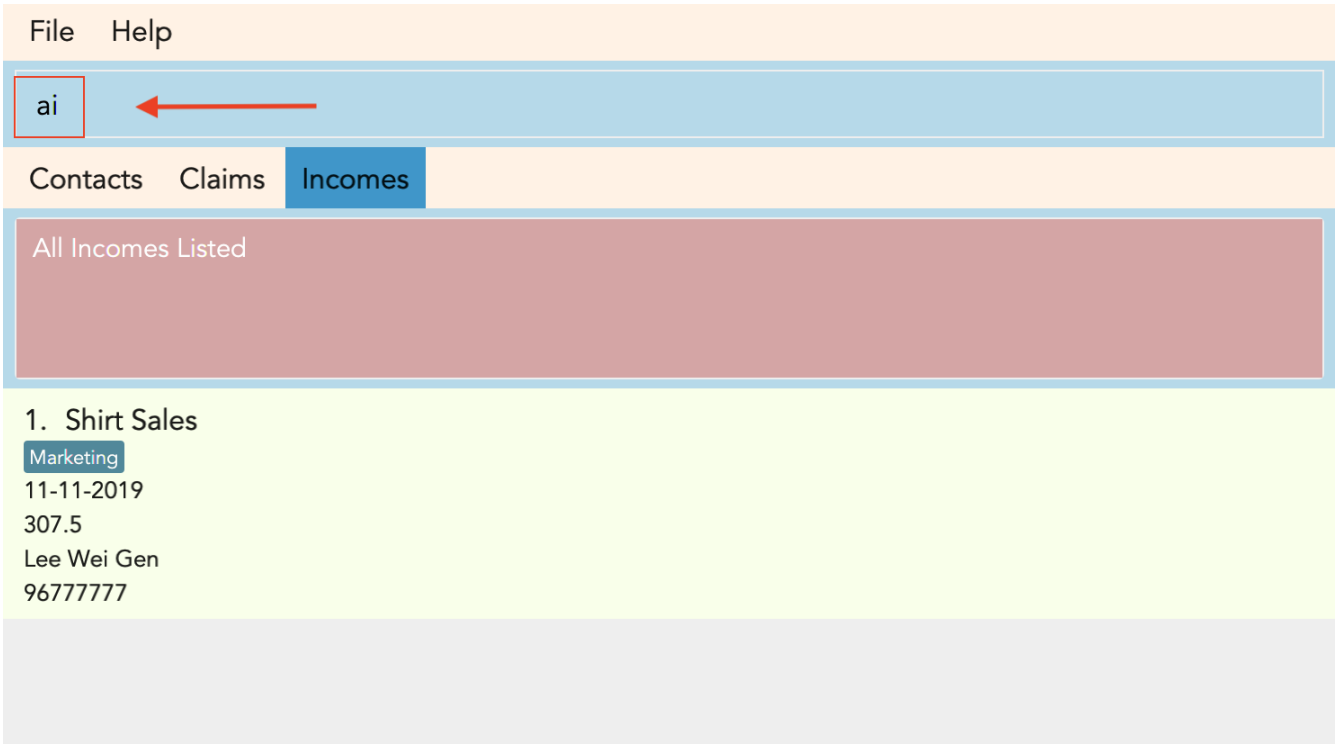

Figure 3.10.2.1: Enter the shortcut you wish to create.

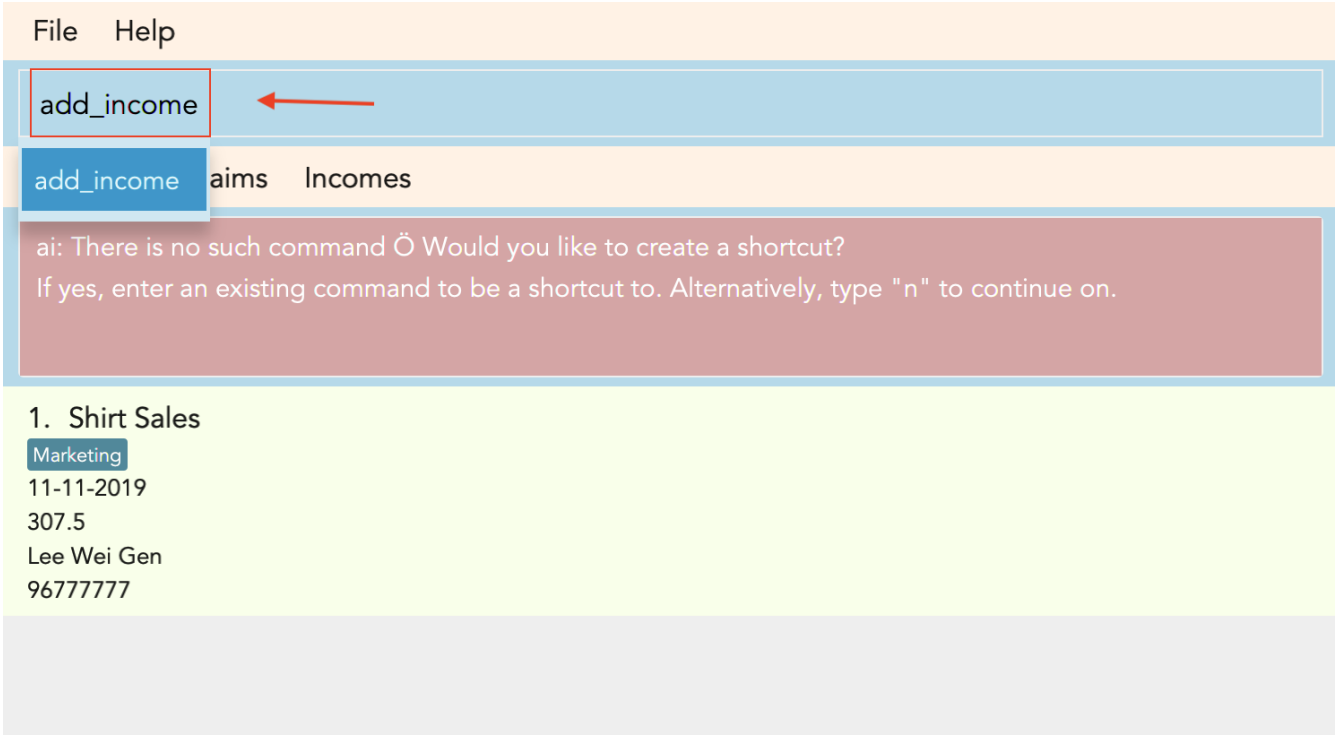

Figure 3.10.2.2: Enter the command you wish to have an alias to.

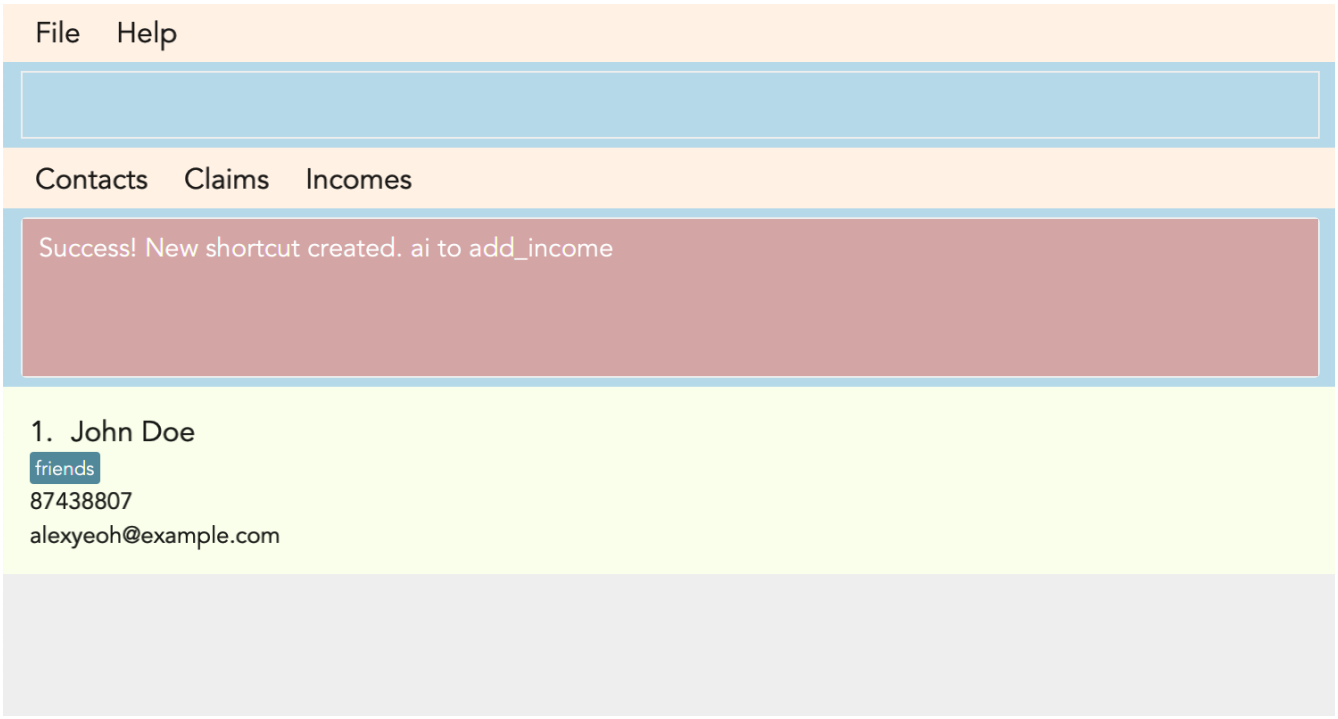

Figure 3.10.2.3: Congratulations! You have just created a shortcut! Now the shortcut is able for use.

Warning

- The shortcut you want to add has to be to a default FinSec command.

Contributions to the Developer Guide

Given below are sections I contributed to the Developer Guide. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.

Shortcut feature

This feature gives the user an option to create a shortcut when an unknown command is entered into FinSec.

Overview

To enable an easy implementation of this feature, we have created a TreeMap to store the default commands and shortcuts in the `FinSecParser`. Each command and shortcut have their Command Word stored as they key value (so as to ensure there are no duplicates) and Command task as the values.

All default commands and shortcuts are also initialised as `CommandItem` objects which would be handled by the `Model Manager` class whenever we add or delete a shortcut. Command classes such as `ShortCutRequestCommand`, `CreateShortcutCommand` and `NoShortCutCommand` are created to facilitate the implementation of the shortcut feature.

When a user enters a command, `FinSecParser` would parse the input and check if the first word of the input is in the TreeMap key set. If the command word is not in the key set, FinSec would create a `ShortCutRequestCommand` and return the `ShortCutRequestCommand` object with `CommandWord` as the parameter.

Example

Given below is an example usage scenario of how the shortcut feature mechanism behaves at each step.

Step 1 : The user launches FinSec for the first time. The `FinSecParser` will be initialised. All the default commands will be added to the TreeMap using `FinSecParser#initialiseDefaultCommands()`. Previously created shortcuts would be added to the TreeMap from the constructor of the `FinSecParser`, where the ObservableList<CommandItem> will be the parameter.

Step 2 : The user enters an unknown command that is not recognised by FinSec. FinSec then returns a `ShortCutRequestCommand`.

Step 3 : `ShortCutRequestCommand` is executed in `LogicManager` and LogicManager would save the command word in a Stack. A `CommandResult` with a new Boolean value of "createShortCut" is then returned to `MainWindow` to display the result of the entry. The `CommandResult#isCreateShortCut()` sets the `MainWindow#unknownEntry` to true.

Step 4 : Since now the boolean value of `MainWindow#unknownEntry` is true, the next input would use a different execute command. Instead of the usual method `LogicManager#execute(String command)`, the next input would be executed by `LogicManager#executeUnknownInput(String command)`. Hence, if the subsequent input by the user is `n`, a `NoShortCutCommand` would be executed and it would return a

`CommandResult` with a false value of `createShortCut` which would reinstate back the normal state of FinSec.

Step 5 : If the user enters a value other than `n`, the `LogicManager#executeUnknownInput(String command)` would call `FinSecParser#checkCommand(String currentInput, String prevInput)` to check if the command is existing or not. If the command is existing, it would return another `ShortCutRequestCommand` object with the same input. However, if a valid default command is entered, a new `CreateShortCutCommand` object is returned (go back to **Step 3**).

- The code snippet below shows the `FinSecParser#checkCommand(String currentInput, String prevInput)` method

```
XYSeries public Command checkCommand(String currentInput, String prevInput) {
        if (FinSecParser.commandList.containsValue(currentInput)) {
            FinSecParser.commandList.put(prevInput, FinSecParser.commandList.get
(currentInput));
            return new CreateShortCutCommand(FinSecParser.commandList.get(
currentInput), prevInput);
        } else {
            return new ShortCutRequestCommand(currentInput);
        }
    }
```

Step 6 : When a `CreateShortCutCommand` object is returned, it is executed in `LogicManager` and `LogicManager` would use `CreateShortCutCommand#execute()` to add the `CommandItem` into the `Model` which is handled by the `ModelManager`.

Step 7 : `CreateShortCutCommmand` would then return a `CommandResult` to the `LogicManager` which would then be returned back to the user.

The following diagrams summarises what happens when a user executes an unknown command:

Figure 2.4.1 is the activity diagram when a user inputs an unknown command

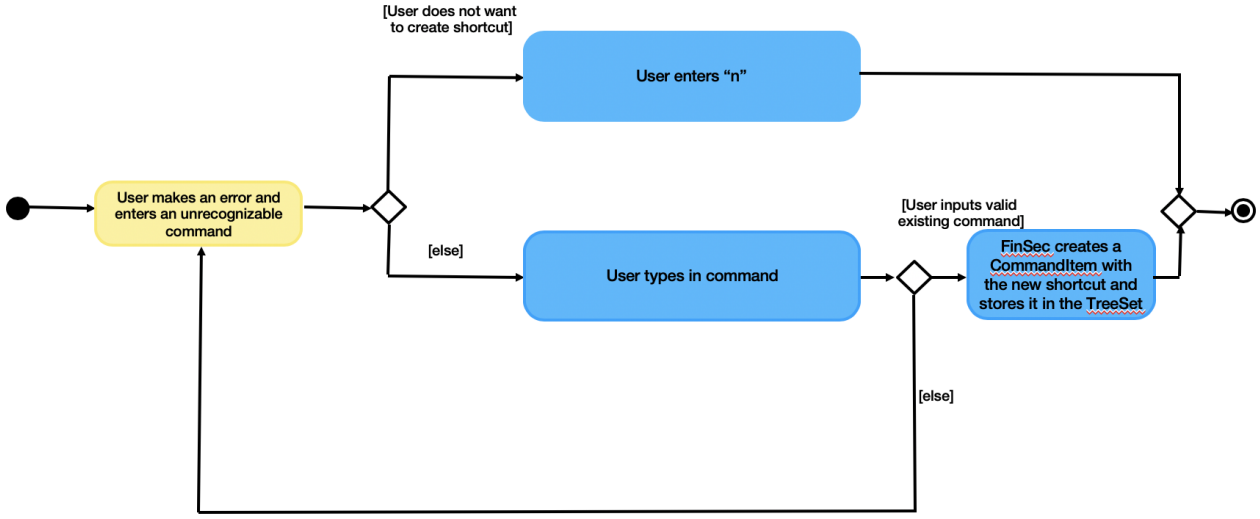

Figure 2.4.1: ActivityDiagram when a user inputs an unknown command

Figure 2.4.2 shows the UML diagram of the flow of logic when a user creates a shortcut to a valid command

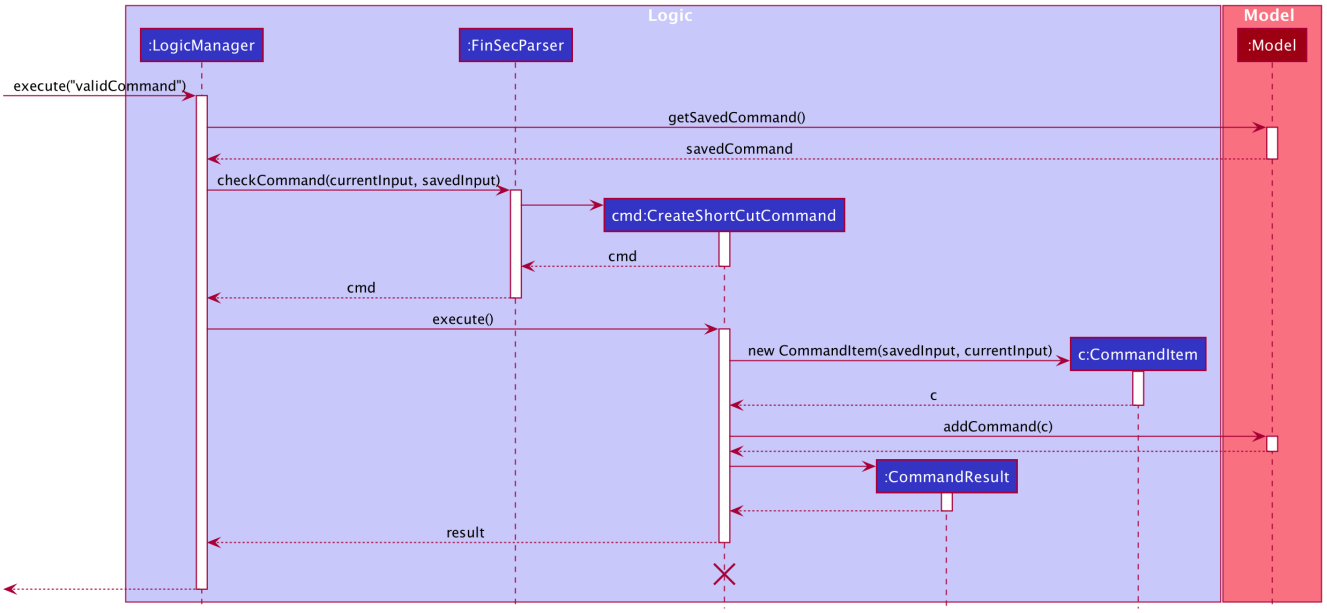

Figure 2.4.2: UML diagram when a user creates a shortcut

Why was it implemented this way?

In order for us to create new Command Words that gives the same functionality of a default command, it was imperative for us to create a CommandItem class that creates an object for every command with the String attributes of `CommandWord` and `CommandTask`. This way, it was possible to create more commands as the user uses the application.

Now with creation of new commands being made possible, we have to decide how we were going to store the list of commands for frequent reference to ensure a bug-free implementation of this feature.

We considered between two alternatives.

Table 1. Data structure to store commandWord and commandTask alternatives

Data Structure Consideration	**Pros and Cons**
TreeMap (Current Choice)	**Pros** : Since each commandWord has 2 attributes, `commandWord` and `commandTask`, this data structure was perfect for storing commands and newly created shortcuts. Furthermore, since we have to look up this TreeMap frequently, a TreeMap would improve the performance of the application since the retrieval speed of a TreeMap is extremely fast. **Cons** : A TreeMap is not as intuitive to implement compared to an ArrayList or List.
ArrayList	**Pros** : It may be more intuitive to implement an ArrayList. **Cons** : Since we have to look up the list of commands frequently, the use of an ArrayList would significantly lower the performance of FinSec. This might negatively affect user experience.

We have decided to opt for the first option primarily because it significantly improves the performance of the application.