Koh Hui Hui Elizabeth - Project Portfolio

1. Introduction

This project portfolio documents my contributions as well as key features that I have implemented in my team project , +Work. The project was for my Software Engineering module (CS2103T) read at the National University of Singapore (NUS).

1.1. About The Team

Including myself, my team comprises of 5 computer science undergraduates from the National University of Singapore.

1.2. About The Project

+Work is a project that is created for the our Software Engineering Module. In this module, we had to morph or enhance the existing AddressBook application to meet the needs of a particular user group.

+Work is a desktop project management application targeted at NUS project leaders of small groups. Users can use +Work for task allocation, inventory management and scheduling purposes. Features include generating a pdf report of inventory items and finding a common time slot among team members.

+Work uses a Command Line Inteferface for users to type into, while incorporating a simple Graphical User Interface (GUI) for users to visualise project details easily.

1.3. Formatting of Document

Shown below are symbols that will appear in this document, and their significance.

IMPORTANT	Information listed here are essential information that users need to take note of in order for +Work to function as expected.
NOTE	Information here are things that the user can learn, so as to more efficiently use +Work.
TIP	Information here are things the user can learn so that they have a faster experience with +Work

2. Summary of Contributions

This section details all of my contributions to the project. It showcases how I worked on different aspects of the project throughout the course.

2.1. Enhancements

- Major enhancement: I implemented the autocomplete feature for the command line.
  - What it does: This feature suggests the possible +Work commands based on what the user has typed in the command line in a dropdown list. If an autocomplete suggestion has been chosen, it will paste the relevant prefixes for that specific command.
  - Justification: This feature increases the usability of +Work, especially for new users, as users do not have to rely on error messages to understand the correct input format. Furthermore, much of the user's interaction is spent on using the command-line and would arguably improve the overall experience of using +Work greatly.
  - Highlights: As of the latest release, the `right arrow` key will toggle the caret to make user interactions with the command line much quicker.
- Minor enhancement:
  - Created UI for the help page for new users to interact and understand +Work easily. (Pull request #88)
- Code contributed: [Functional code] [Test code]

2.2. Other Significant Contribution to Project

- Project management
  - Set up a project management tool, Trello, at the start of the project to facilitate ease of communication and clear division of roles.
  - Refactored code to remove addressbook (Pull request #162)
- Enhancements to existing features
  - Enhanced the styling for the dropdown menu of autocomplete (Pull request #168)
- Documentation
  - Wrote parts of the +Work User Guide relating to Quick Start, Introduction, Autocomplete and Command Summary.
  - Wrote parts of the +Work Developer Guide relating to AutoComplete and Architecture.
- Community
  - Reported bugs and suggestions for other teams in the class (report)

3. Contributions to the User Guide

The following section is an excerpt from my additions to the +Work User Guide for the Quick Start,

Introduction and autocorrect feature.

3.1. Quick Start

1. Ensure you have Java 11 or above installed in your Computer.
2. Download the latest pluswork.jar here.
3. Copy the file to the folder you want to use as the home folder for your +Work.
4. Double-click the file to start the app. The GUI should appear in a few seconds.

5. Type the command in the command box and press Enter to execute it.
   e.g. typing help and pressing Enter will open the help window.
6. Here are some commands you can try to get you started:
   - ! add-member mn/Adam Smith mi/AS : Adds a member with name "Adam Smith" to the project.
   - ! add-task tn/Finish up milestone setting : Adds a task with name "Finish up milestone" to the project.
   - ! assign ti/1 mi/JD : Assigns the project member "John Doe" (with member id "JD"), to the task "Finish up milestone setting".
   - ! doing-task ti/1 : Changes status of task "Finish up milestone setting" to doing:

3.2. Command Line Autocomplete

+Work will automatically prompt you on the various possible commands based on input and help you paste the correct command format into the command line if chosen.
For example, when you type add-mem into the command-line, +Work will prompt you to select add-member from the drop down menu.

After selecting `add-member`, +Work will paste `add-member mn/ mi/ mt/` into your command-line automatically!

TIP | You can use `control` to move to the next prefix.

4. Contribution to the Developer Guide

The following section displays my additions to the +Work Developer Guide for the `AutoComplete` feature and Architecture Section. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.

4.1. AutoComplete feature

4.1.1. Proposed Implementation

- Alternative 1 (current choice): Logic handles the autocomplete logic. While Commandbox will receive the result and populate the context menu based on output from logic.
  - Pros: Easy to change autocomplete logic in the future if need be, such as integrating prefix suggestions, which require the dashboard's data;
  - Cons: Need to access the autocomplete component. Hard to pass props to logic with regards to textbox like caret position.
- Alternative 2: Handle all autocomplete logic within command box
  - Pros: This would be easier to implement and maintain UI components. Can access TextField directly.
  - Cons: Difficult to access other logic components. Handles both UI and logic in the same component.]

4.1.2. AutoComplete control flow

Given below is an example usage scenario and how AutoComplete behaves at each step:

Step 1. The user launches the application. The command box is initialized together in the main window. Logic is initialised and passed into command box.

Step 2. The user attempts to type a command `input` in Command Box.

Step 3. Command Box calls `Logic#getAutoCompleteResults()` which calls `AutoComplete#completeText(String input)`, which calls `Keywords #commandList(String input)`. This returns a filtered Linked List of possible commands.

Step 4. The Linked List of commands will be passed back into Command Box who will call `populatePopup(LinkedList<String> searchResult)` to make its own ContextMenu to be displayed.

Step 5. The UI now reflects a list of available commands filtered based on text.

The following sequence diagram shows how the autocomplete works.

Figure 1. Operational flow of `getAutoCompleteResults()`

4.2. Architecture

Figure 2. Architecture Diagram

The *Architecture Diagram* given above explains the high-level design of the App. Given below is a quick overview of each component.

TIP

The `.puml` files used to create diagrams in this document can be found in the diagrams folder. Refer to the Using PlantUML guide to learn how to create and edit diagrams.

`Main` has two classes called `Main` and `MainApp`. It is responsible for,

- At app launch: Initializes the components in the correct sequence, and connects them up with each other.
- At shut down: Shuts down the components and invokes cleanup method where necessary.

`Commons` represents a collection of classes used by multiple other components. The following class plays an important role at the architecture level:

* `LogsCenter` : Used by many classes to write log messages to the App's log file.

The rest of the App consists of four components.

* `UI`: The UI of the App.
* `Logic`: The command executor, parses user input.
* `Model`: Holds the data of the App in-memory.
* `Storage`: Reads data from, and writes data to, the hard disk.

Each of the four components

* Defines its *API* in an `interface` with the same name as the Component.
* Exposes its functionality using a `{Component Name}Manager` class.

For example, the `Logic` component (see the class diagram given below) defines it's API in the `Logic.java` interface and exposes its functionality using the `LogicManager.java` class.

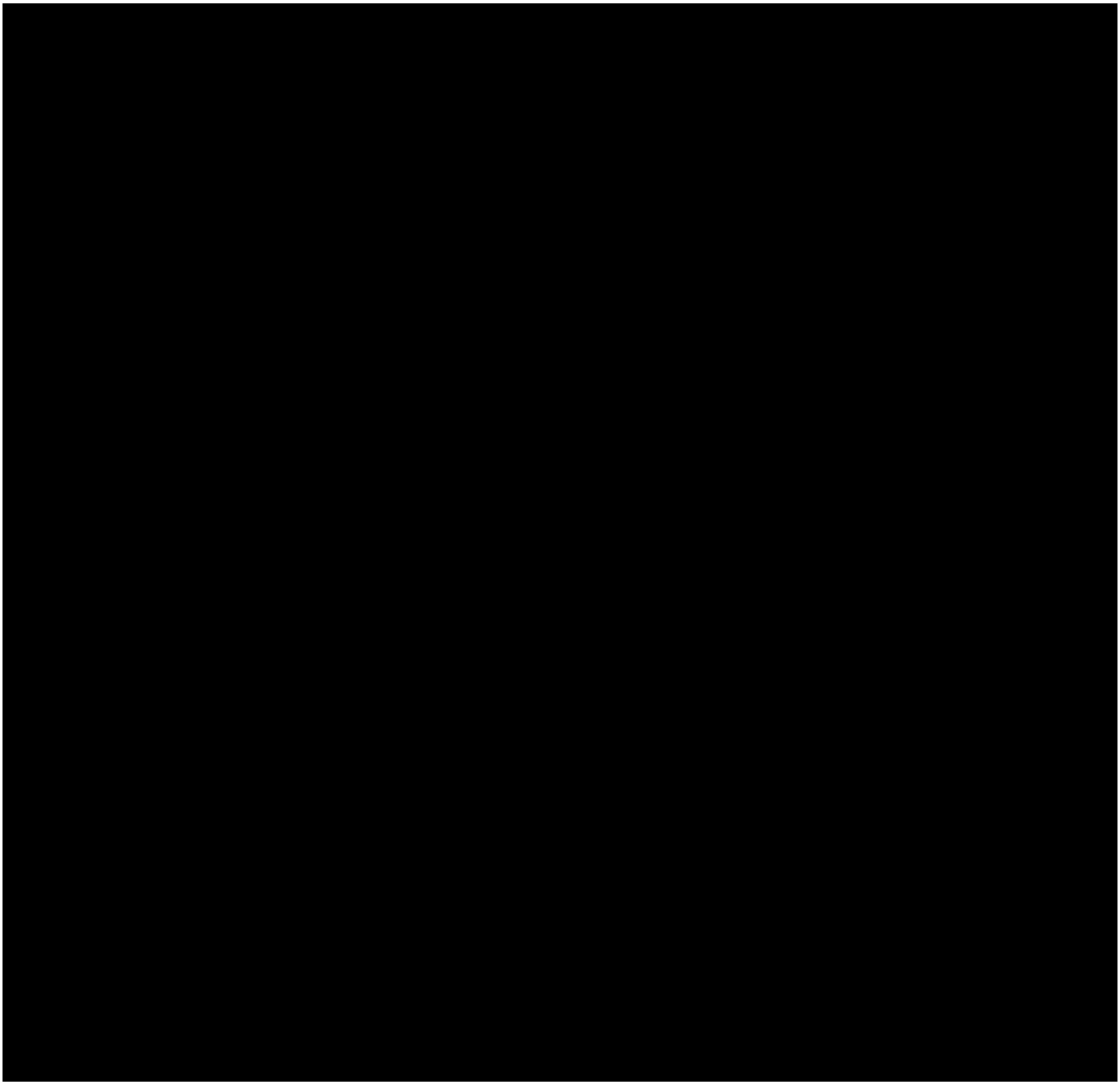

Figure 3. Class Diagram of the Logic Component

How the architecture components interact with each other

The *Sequence Diagram* below shows how the components interact with each other for the scenario where the user issues the command `add-member mn/Abhi`.

Figure 4. Component interactions for `add-member mn/Abhi` *command*