Yap Shi Hao - Project Portfolio

Welcome to AddMin+

Overview on Project

AddMin+ is a student developed application under the National University of Singapore's *CS2103T Software Engineering Module*. The project required students to enhance a basic command-line interface desktop application which is the AddressBook - Level 3 that was developed by the se-edu team. The AddressBook - Level 3 is a desktop address book application that was designed to be used for teaching Software Engineering principles.

About AddMin+

The AddMin+ Team comprises of 5 Computer science students from AY1920S1-CS2103T-T11-3. The team decided to morph and enhance the capabilities of the original application into an all-in-one administration desktop application. The application is specially designed for the use of any events management start-up company that is faced with limited manpower and resources.

AddMin+ is specially designed to ease the workload of admin staff in these companies and allow them to effectively handle the administrative needs of the company by providing the following functions: event creation and deletion, editing of event details, manual and automatic manpower allocation for events and providing an overview of all the data via statistics while keeping track of the companies schedule to ensure that no event will be missed!

To give you a better picture of AddMin+! This is what the application looks like:

Figure 1. The user interface for AddMin+

My Roles and Responsibilities

My project responsibilities and tasking include being in charge of the group's documentation quality as well as ensure that high standards are maintained and delivered on time and in the right format.

While my coding related tasking was to design and write codes for the entire **Schedule** section of the project. The following sections will illustrate the enhancements in more detail, as well as provide relevant documentation that I have added to the user and developer guides in relation to those enhancements.

Before we start

The following symbols and formatting will be used throughout this document:

TIP	**LightBulbs** indicate a *Tip*, something that is helpful to the reader
NOTE	**'i' Icon** indicate a *Note* that supplements useful information

Text in a `blue font and grey` background indicates a *hyperlink*

Texts with a `grey background` indicates *Code Logic* such as class objects, OOP definitions or user-input.

Summary of contributions

This section will show a summary of all the features, enhancements, documentation, and other useful contributions that I have contributed to the team project.

- **Major feature**: added and implemented the entire **Schedule** section of the project which includes major GUI implementations and 4 schedule related commands.
    - What does it do: The **Schedule** section was implemented to allow users to have an easier way to view and identify events that are happening on a specific date or period. Thus, the `display_schedule_date`, `display_schedule_month` and `display_schedule_between` commands were implemented. The enhancement was also done to improve the user experience, by providing a clean graphical representation of the user's schedule through the use of the `generate_schedule` command, greatly improving the application's user experience.
    - Justification: The **Schedule** feature improves the product significantly because it allows users to have a convenient way to view their upcoming event schedule and keep track of all the important event dates. The feature will greatly improve the user experience of the application by providing a clean user interface to allow the users to have a visual overview of their schedules.
    - Highlights: This enhancement requires handling and dealing with the `Event` objects extensively. The implementation of the **Schedule** feature required an in-depth analysis of the application's storage and modeling architecture. The **Schedule** feature was also

challenging as it required the implementation of an additional `DistinctDate` object for the `generate_schedule` command which was also used in the `Employee` window. During the designing of the **Schedule** component, deep consideration was needed to ensure that the user interface was clean and optimal for the users.

- **Minor features**: Multiple Graphic User Interface improvements were made throughout the application to improve the user experience. Implemented the tab recognition feature, which is used to restrict commands on specific tabs to prevent unusual behavior, which can cause users to be confused. Other minor features includes the implementation of the `add_ev`, `edit_ev`, `find_em_tag` and `find_ev_tag` commands for the AddMin+ App.
- **Code contributed**: [tp Code Dashboard]
  - **Model**: [TagContainsKeywords, EventContainsKeyDate, EventContainsKeyDateRange, EventContainsKeyYearMonth, EventTagContainsKeywords, DistinctDate]
  - **Command**: [FindEventByTag, FindEmployeeByTag, GenerateSchedule, DisplayScheduleBetween, DisplayScheduleForDate, DisplayScheduleForYearMonth]
  - **Parser**: [DisplayScheduleBetween, DisplayScheduleForDate, DisplayScheduleForYearMonth, FindEventByTag, FindEmployeeByTag]
  - **Others**: [DistinctDatesProcessor, DateCard, DateWindow, FetchEmployeeWindow, ScheduleBox]
- **Test Code Contributed**:
  - **Model**: [EventContainsKeyDate, EventContainsKeyDateRange, EventContainsKeyYearMonth, DistinctDateTest]
  - **Command**: [GenerateSchedule, DisplayScheduleBetween, DisplayScheduleForDate, DisplayScheduleForYearMonth]
  - **Parser**: [DisplayScheduleBetween, DisplayScheduleForDate, DisplayScheduleForYearMonth]
  - **TestUtil**: [DistinctDateBuilder, EventBuilder, TypicalDistinctDates, TypicalEvents]
- **Other contributions**:
  - Project management:
    - In-charged of keep track of deadlines, submissions and issue trackers as well as the entire team's documentation quality.
    - In-charged of Add and Edit event commands for AddMin+ (Pull requests #76, #79)
    - In-charged of Find Employee and Event Tags commands for AddMin+ (Pull requests #197)
    - In-charged of the entire **Schedule** feature of AddMin+ which includes 4 schedule related commands (Pull requests #102, #109, #138, #139)
    - In-charged of setting up test cases for all **Schedule** feature - commands, parsers, predicates (Pull requests #233, #249, #267)
    - Assisted in morphing the Events class and storage units into AddMin+ (Pull requests #76, #79)
    - Assisted in the refactoring process for person class to employee class and well as fixed test cases and checkstyle errors. (Pull requests #92)

        - Assisted in debugging AddMin+ based on User Feedback and Mock Practical Exam Feedback (Pull requests #197)
        - Assisted in setting up the test structure for Events to facilitate easier Events related test cases (Pull requests #233)
        - Assisted in implementing AddMin+ tab recognition, to reduce any unusual app behavior. (Pull requests #155, #160, #265)
    - Enhancements to existing features:
        - Upgraded the GUI display and other GUI enhancements, such as the fetch employee window (Pull requests #102, #143)
    - Documentation:
        - Did cosmetic tweaks to existing contents of the User Guide: (Pull requests #114, #143, #149, #150, #197, #231)
    - Community:
        - PRs reviewed (with non-trivial review comments): (Pull requests #66, #76, #83, #85, #92, #137, #273)

Contributions to the User Guide

*The original user guide has been updated to document the new capabilities of the AddMin+ functionality. The following are excerpts from the AddMin+ User Guide, which will showcase the sections that I have contributed to, in particular, the **Schedule** section of the UG. They will showcase my ability to write documentation for end-users.*

Schedule Management

Welcome to Schedule! Looking for a way to display and see what events you have on a specific Date or Month? Want to have an overview of all the dates where you have an event? Then you are at the right place!

To get things started, all Schedule-related commands occur in the Schedule Tab as seen from the figure below. You can get to the Schedule Tab easily by either clicking on the Schedule Tab on the User Interface or just simply type in any Schedule-related commands and AddMin+ will bring you there.

NOTE

- All Event-Related commands that are done in the Schedule Tab, will be referencing the Event List displayed in the Schedule Tab.
- All Employee-Related commands are disabled in the Schedule Tab, as there is no Employee List being displayed in the Schedule Tab.

Figure 2. User Interface (UI) of the Schedule Feature

Display Schedule for a specific date: `display_schedule_date`

Want to check if you have any events on a specific date? Instead of looking through the entire list of events you have, you could use the `display_schedule_date` command to do it!

Format: `display_schedule_date on/dd/MM/yyyy`

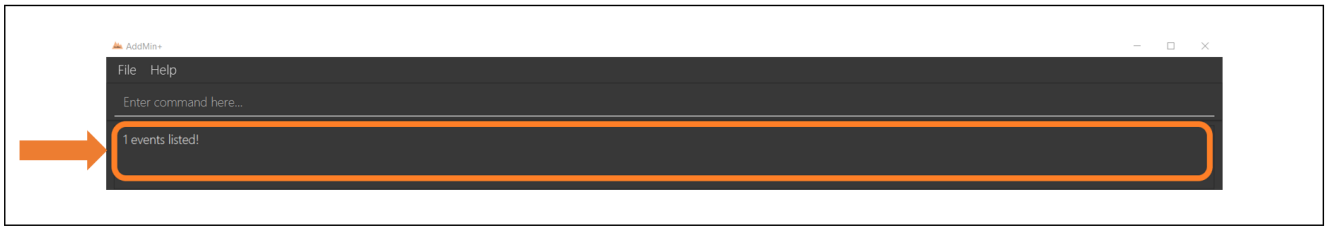
Examples: `display_schedule_date on/02/12/2019`

> - Note the [Constraints] for **DATE**.

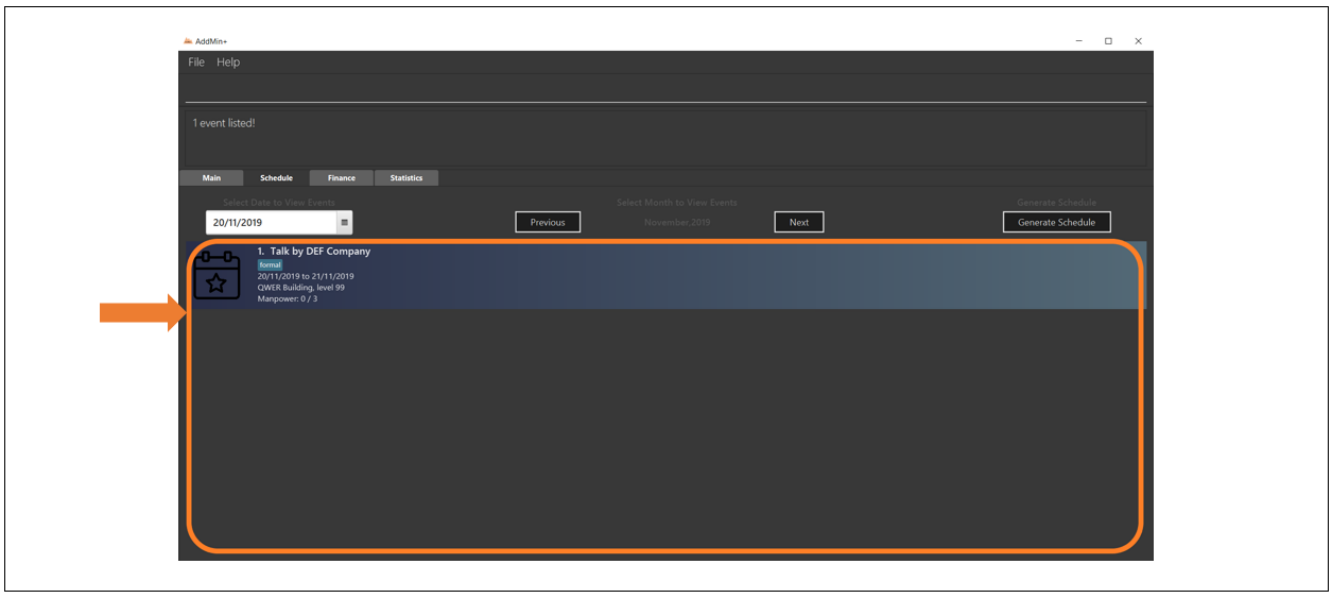
Example: To display schedule for a specific date (20/11/2019):

Step 1. Either type `display_schedule_date on/20/11/2019` into the command box or click on the specific date from the date picker as seen from the figure below.

Step 2. The result box will display a message informing you how many events are being listed. In this case, the message displayed is "1 event listed!"



Step 3. In the event list, you will be able to see all the events that are on the specified date. In this case, the only event that is on 20/11/2019 is "Talk by DEF Company".



NOTE

- The event list will be empty and not display anything if there is no event on the specified date.
- The command is designed to give an overview of the user's schedule and is based on the event's Start and End date. It does not take into account of the event's set date and time. (To have a more detailed view of the event, simply double click on that event).
- As long as the user's specified input date falls between the event's start and end date, the event will be displayed.

Contributions to the Developer Guide

Given below are sections I contributed to the Developer Guide. The section will cover the design considerations and design code structure of the `generate_schedule` *command. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.*

Generate Schedule Feature

Proposed Implementation

The Generate Schedule Feature is implemented to allow users to have an overview of the event schedule. It will display all dates that have an event and the specific events that are happening on those dates listed. Do note that the `generate_schedule` command will only display dates and events that have a set date and time allocated to it. The feature is facilitated by a `DistinctDatesProcessor` and requires the use of a new Object - `DistinctDate`, as well as an internal ObservableList - `distinctDatesList` found in the `ModelManager`.

The `DistinctDateProcessor` processes the entire list of Events in the `EventList` when the command is called. The `DistinctDateProcessor` will then process through these events to create specific `DistinctDate` Objects which stores a list of events that occurs on the date they are representing. These DistinctDate Objects are then used, to create `DateCard` which will be displayed on the GUI. This feature can be seen in the generate schedule window as well as the employee fetch window.

The `DistinctDateProcessor` utilises the following operations in the `generate_schedule` command:

- `generateAllDistinctDateList(Model model)` — Returns a list of `DistinctDate` Objects. This operation utilises the generateDistinctDateList() operations.
- `generateDistinctDateList(List<Event> eventList)` — Returns a list of `DistinctDate` Objects. This operation utilises the generateDateList() and generateListOfEventForDate() operations.
- `generateDateList(List<Event> eventList)` — Takes in the entire list of events, identify all the dates that have been mapped to which that has at least one event and returns it as a list.
- `generateListOfEventForDate(EventDate date, Model model)` — Takes in an `EventDate` object, and processes through the entire list of events, to find all events on that specific date, and return them as a list.

Below is an example usage scenario and how the `display_schedule` command behaves at each step.

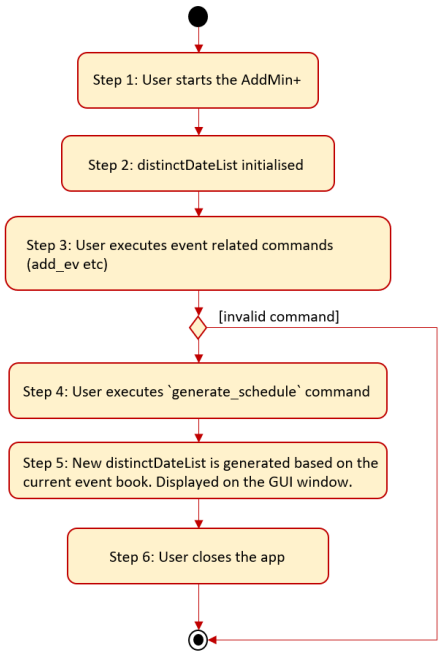


Figure 3. Program flow of the Generate Schedule Feature

Step 1. The user launches the application for the first time.

Step 2. The `distinctDatesList` will be initialised based on the initial event book state.

Step 3. The user executes `add_ev n/Free Coffee ⋯` to add a new event into the Eventlist. The distinctDatesList will not be updated, and will not contain the new event that is added.

NOTE	Any command that alters the eventBook will not change the distinctDatesList. Only when the `generate_schedule` command is called, a new distinctDateList will be generated again using the latest EventList.

Step 4. The user executes `generate_schedule` to see all the dates that have a set time mapping and the respective events on those dates.

Step 5. The distinctDateList will be generated again based on the current list of events in the EventList and will be displayed on a separate window.

Step 6. The user now decides to close the app, the current state of the EventBook and EmployeeBook will be stored, however, the DistinctDateList would not.

NOTE	Note that the Display Schedule Feature does not load and store the DistinctDate Objects. It processes and generates the list when it is called upon or when the application starts.

Step 7. Done.

The following sequence diagram shows how the `generate_schedule` operation works:

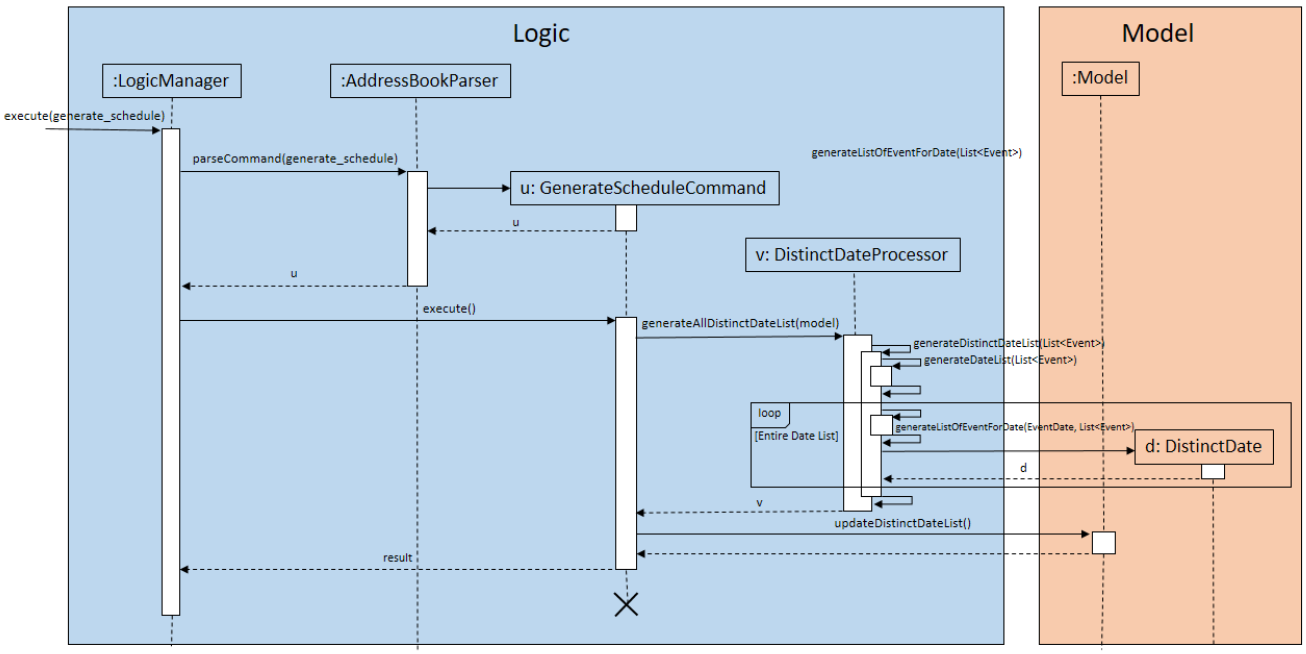


Figure 4. Sequence Diagram for generate_schedule Command

NOTE	The lifeline for `GenerateScheduleCommand` ends at the destroy marker (X).

Design Considerations

	Alternative 1	Alternative 2
Consideration 1: Data Structure to support Generate Schedule Command.	**Generates and Processes the DistinctDate Object upon `generate_schedule` command. (Current choice)**: *Pros:* Easy to implement and requires less storage capacity and storage infrastructure to support the entire feature. *Cons:* The program will have to iterate through the entire list of events and create the corresponding `DistinctDate` objects, whenever `generate_schedule` command is called, can cause time complexity issue of database gets big.	**Creates and Stores the DistinctDate object whenever a new event is added.** *Pros:* Do not have to create a new list of DistinctDate object every time it is called. *Cons:* Requires new storage unit to store a new entity which is not as important and frequently used. This implementation can cause speed and time complexity issues as well, as the program is required to process through all DistinctDate Object whenever there are any changes to the event list.
Why We chose Alternative 1: Alternative 1 makes more logical sense and will be more efficient as compared to Alternative 2. Alternative 1 requires less intermediate processing and storage units to support the feature. Processing is only done when it is needed. Looking at the use case of the `generate_schedule` command, it is likely to be used when the users have finalised all the events and details before generating the schedule.		

	Alternative 1	Alternative 2

Consideration 2: UI Decisions for Generate Schedule Command	**Display Directly on the Schedule Tab, update the list when `generate_schedule` command is called**: *Pros:* Users are able to view the generated schedule directly from the application's schedule tab, without the need of another window. *Cons:* May cause confusion, as the list being displayed might be outdated if the user forgets to call the `generate_schedule` command after altering the events.	**Display on a separate window generates and display the list on the new window when `generate_schedule` command is called. (Current choice)** *Pros:* Allows for better user experience since the generated list is only displayed when the user needs it. Ensures that the list being displayed is always updated as of when the user needs it. *Cons:* Harder to implement, requires additional JavaFx windows and implementations. Will require additional windows being opened.
Why we chose Alternative 2: Alternative 2 is a cleaner and more user-friendly approach compared to Alternative 1. Alternative 2 helps to prevent the Schedule Tab from being filled with too many lists and information. Alternative 2 also helps prevent user confusion, as the list that is displayed is always updated as of when it is called upon.		

Conclusion:

The AddMin+ project was created by the AddMin+ Team as part of a university module requirement. Through AddMin+, I was able to experience what it was like to be involved in a software development project, where we were able to experience both the technical and non-technical aspects of a software development project. We were able to directly apply the key software development concepts taught in CS2103T in our project and also experience what it was like dealing with a brown-field project.

Another aspect of software development which I felt is often overlooked is actually the non-technical aspect, such as documentation writing. However, through CS2101, we were able to learn the steps of writing quality documentation of our project that caters to our target audiences. Finally, even though the AddMin+ project was only 6 weeks long, I was able to pick up many valuable knowledge that will be essential in becoming a software developer in the future. If you would like to know more about me or AddMin+, feel free to drop by my personal website or contact me via LinkedIn.