Calvin Chen Xingzhu - Project Portfolio

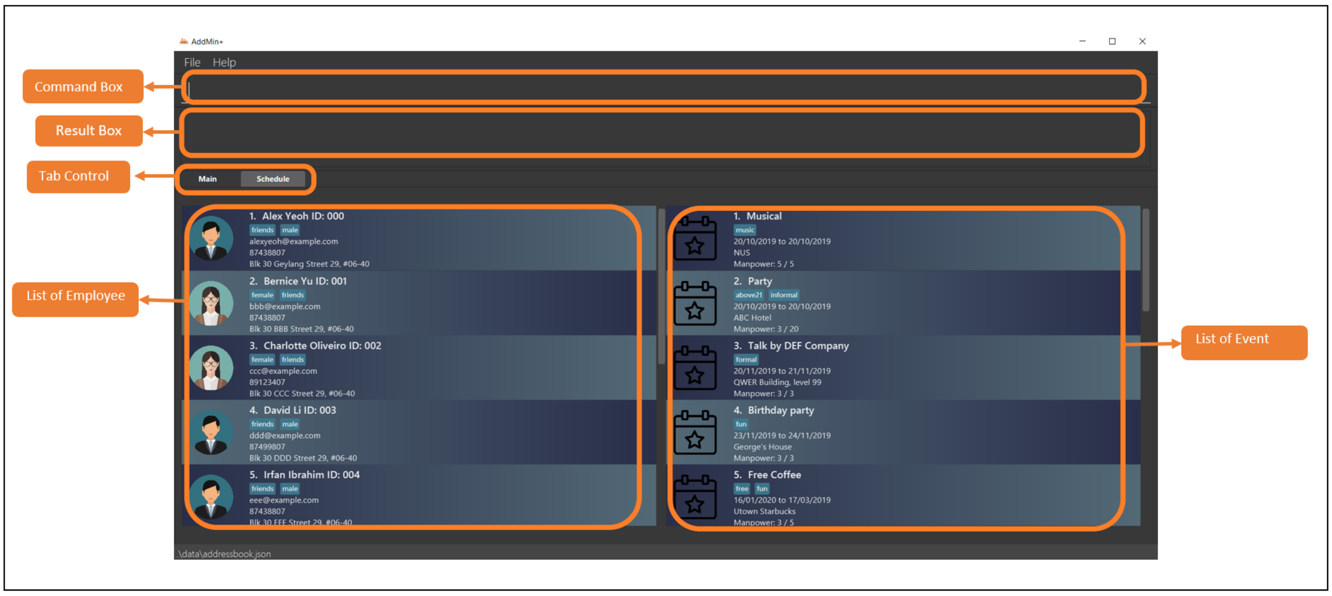
PROJECT: AddMin+

Overview

My team of 4 software engineering students and I were tasked with enhancing a basic command line interface desktop addressbook application for our Software Engineering project. We chose to morph it into a all-in-one administration desktop application called AddMin+.

AddMin+ is specially designed to ease the workload of the admin staffs in these companies and allow him/her to effectively handle the administrative tasks of the company.

This is what our project looks like:



1. Alex Yeoh ID: 000
friends male
alexyeoh@example.com
87438807
Blk 30 Geylang Street 29, #06-40

2. Bernice Yu ID: 001
female friends
bbb@example.com
87438807
Blk 30 BBB Street 29, #06-40

3. Charlotte Oliveiro ID: 002
female friends
ccc@example.com
89123407
Blk 30 CCC Street 29, #06-40

4. David Li ID: 003
friends male
ddd@example.com
87499807
Blk 30 DDD Street 29, #06-40

5. Irfan Ibrahim ID: 004
friends male
eee@example.com
87438807
Blk 30 EEE Street 29, #06-40

1. Musical
music
20/10/2019 to 20/10/2019
NUS
Manpower: 5 / 5

2. Party
above21 informal
20/10/2019 to 20/10/2019
ABC Hotel
Manpower: 3 / 20

3. Talk by DEF Company
formal
20/11/2019 to 21/11/2019
QWER Building, level 99
Manpower: 3 / 3

4. Birthday party
fun
23/11/2019 to 24/11/2019
George's House
Manpower: 3 / 3

5. Free Coffee
free fun
16/01/2020 to 17/03/2019
Utown Starbucks
Manpower: 3 / 5

.\data\addressbook.json

Figure 1. The graphical user interface(GUI) for AddMin+.

My role was to design and write the code for the association of employees with event, which includes event display and the allocation of employee to event. The following sections illustrate these features in more detail, as well as the relevant documentation I have added to the user and developer guides in relation to these features.

Summary of contributions

- **Major feature**: enabled the association of employees with events which includes **automated/manual allocation of employees to event commands** and a **deallocate command**, as well as the viewing of changes the users made via the **fetch event** command. All related commands have **GUI features** implemented.
    - What it does: This feature allows the user to check for availability of employees and allocate them to an event with certain filter requirements. The algorithm ensures that employees cannot be allocated to events with conflicting time periods.
    - Justification: This feature improves the product significantly because a user can now allocate employees to events, which is a key component for events management companies that our app is targeting. The GUI features further provide convenience for users.
    - Highlights: This enhancement affects existing commands and commands to be added in future. It required an in-depth analysis of design alternatives. The implementation too was challenging as it required the knowledge of both `Event` and `Employee` classes.
- **Minor feature**: added command history to allow user to view previous commands using up/down arrow keys.
- **Code contributed**: [Functional code] [Test code] *{give links to collated code files}*
- **Other contributions**:
    - Project management:
        - Wrote the skeleton code for our morphed product which required integration of various components (e.g. storage, logic, model). (Pull request #66)
        - Assisted in refactoring process for person class to employee class and well as fixed test cases and checkstyle errors. (Pull request #92)
    - Enhancements to existing features:
        - Updated GUI to show both employee and event list (Pull request #78)
    - Documentation:
        - Did cosmetic tweaks to existing contents of the User Guide: (Pull request #153)
        - Updated class diagram for storage component in the Developer Guide: (Pull request #126)
    - Community:
        - PRs reviewed (with review comments): (Pull requests #76, #79, #89, #104, #109, #114, #129, #132, #134, #138, #138, #145)
        - Reported bugs and suggestions for other teams (examples: 1, 2, 3, 4, 5, 6)

Contributions to the User Guide

Event-Specific Management

NOTE Some event-specific commands have GUI features enabled to improve user experience. However, such features are implemented as a **best-effort service** and should not be heavily relied on.

Fetch Full Details of an Event: `fetch_ev`

Fetches an event by displaying a pop-up window with full details of the event.

Format: `fetch_ev EVENT_INDEX`

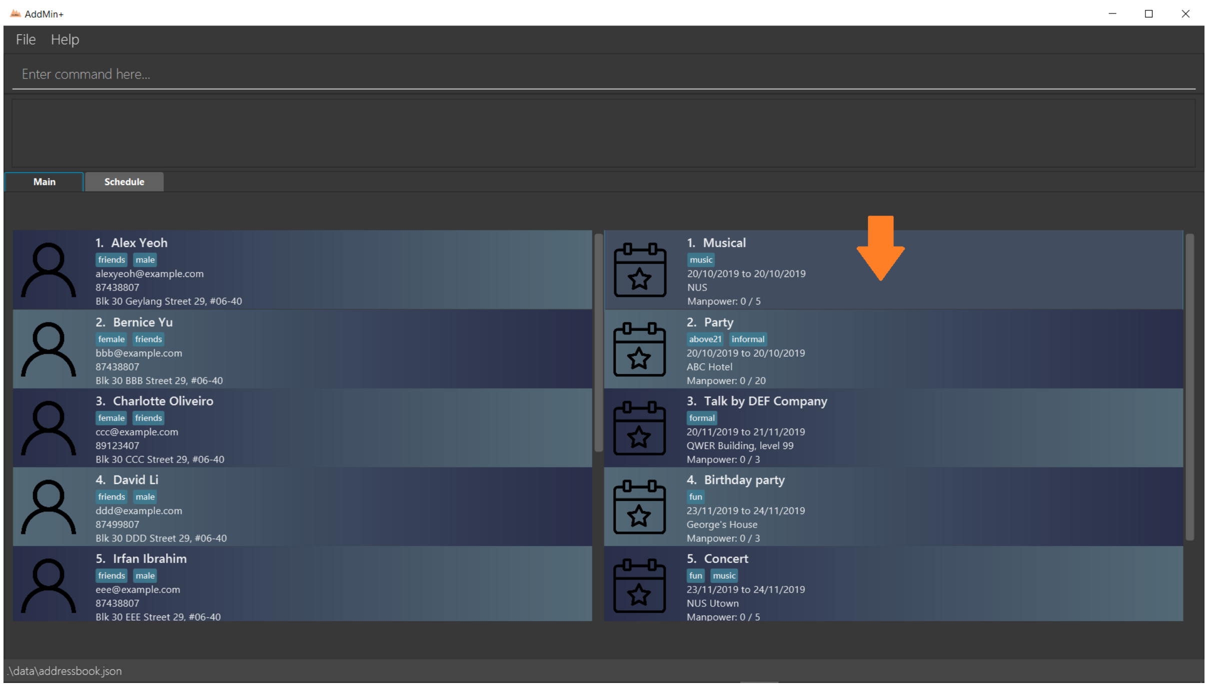
- The `EVENT_INDEX` refers to the index number shown in the displayed event list.
- Note the [Constraints] for **INDEX**.

Examples:

- `fetch_ev 2`
  Returns the 2rd event from the event list.

Alternatively, the `fetch_ev` command can be executed from the GUI in just 2 simple steps.

Step 1. **double-click the event** in the list as shown in the figure below:



1. Alex Yeoh
friends male
alexyeoh@example.com
87438807
Blk 30 Geylang Street 29, #06-40

2. Bernice Yu
female friends
bbb@example.com
87438807
Blk 30 BBB Street 29, #06-40

3. Charlotte Oliveiro
female friends
ccc@example.com
89123407
Blk 30 CCC Street 29, #06-40

4. David Li
friends male
ddd@example.com
87499807
Blk 30 DDD Street 29, #06-40

5. Irfan Ibrahim
friends male
eee@example.com
87438807
Blk 30 EEE Street 29, #06-40

1. Musical
music
20/10/2019 to 20/10/2019
NUS
Manpower: 0 / 5

2. Party
above21 informal
20/10/2019 to 20/10/2019
ABC Hotel
Manpower: 0 / 20

3. Talk by DEF Company
formal
20/11/2019 to 21/11/2019
QWER Building, level 99
Manpower: 0 / 3

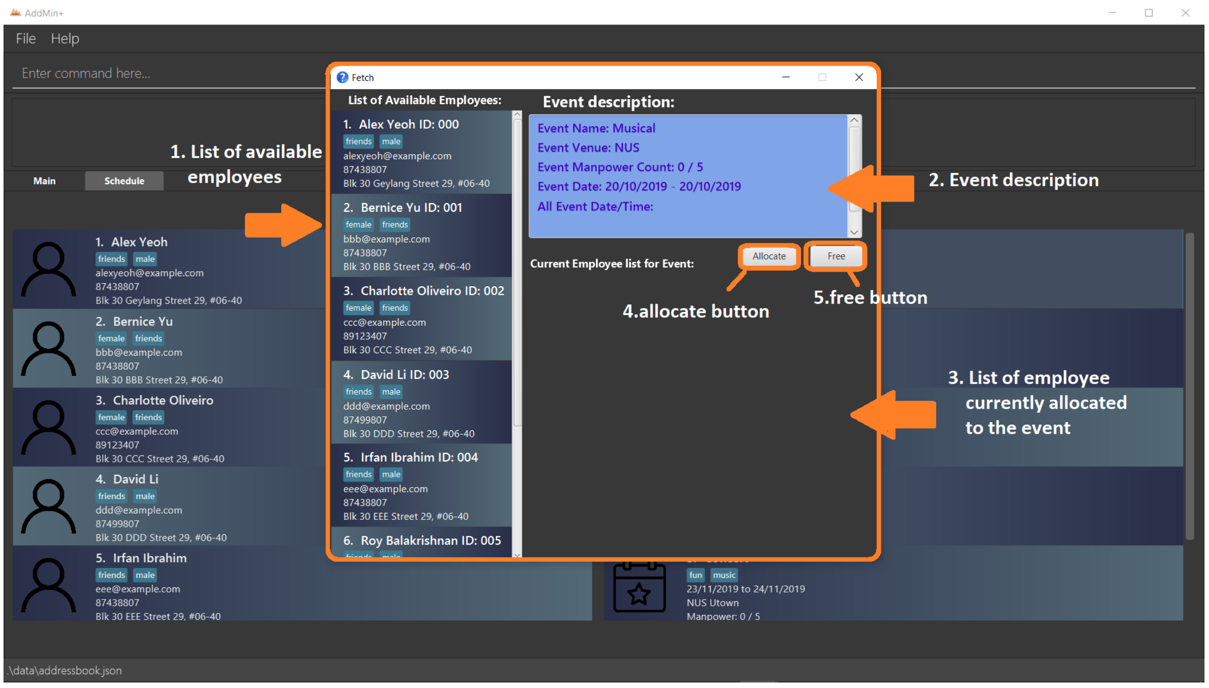
4. Birthday party
fun
23/11/2019 to 24/11/2019
George's House
Manpower: 0 / 3

5. Concert
fun music
23/11/2019 to 24/11/2019
NUS Utown
Manpower: 0 / 5

.\data\addressbook.json

Figure 2. Instruction for user to execute fetch event command

Step 2: After successfully fetching the event, the following **Fetch Event Window** should show:


1. List of available employees
2. Event description
3. List of employee currently allocated to the event
4.allocate button
5.free button
List of Available Employees:
Event description:
Current Employee list for Event:
Allocate
Free

Event Name: Musical
Event Venue: NUS
Event Manpower Count: 0 / 5
Event Date: 20/10/2019 - 20/10/2019
All Event Date/Time:

Figure 3. Fetch Event Window

Automated allocation of Employees to Events: `allocate`

Automatically filters and allocates a specified number of employees from the complete employee list to an event. **AddMin+ prevents conflicting schedules by ensuring that employees cannot be allocated to events with overlapping periods.**

Format: `allocate EVENT_INDEX [n/NUMBER_OF_EMPLOYEES] [t/TAG]…`

NOTE Random selection of employees to allocate if supply exceeds demand of event.

- Allocates a `NUMBER_OF_EMPLOYEES` of employees to the event at the specified `EVENT_INDEX` filtered based on `TAG`.
- The `EVENT_INDEX` refers to the index number shown in the displayed event list.
- The `NUMBER_OF_EMPLOYEES` refers to the number of employees to be allocated to the event.
- Note the [Constraints] for **INDEX** and **INTEGER** *(for NUMBER_OF_EMPLOYEES)*.
- If no `NUMBER_OF_EMPLOYEES` is specified, it is assumed to be the current manpower count required by the event.

Examples:

- `allocate 1`
  Allocates available employees to the 1st event shown in the event list.
- `allocate 2 n/3 t/female`
  Allocates 3 employees who are tagged as 'female' to the 2nd event shown in the event list.

Alternatively, the `allocate` command can be executed from the GUI in just 1 simple step!

Step 1: To perform a `allocate` command without number/filter specification, click the **allocate** button as shown in the **Fetch Event Window** in [Fetch Full Details of an Event: `fetch_ev`].

Manually allocation of Employees to Events: `allocatem`

Manually chooses and allocates a single employee to an event.

Format: `allocatem EVENT_INDEX n/EMPLOYEE_INDEX`

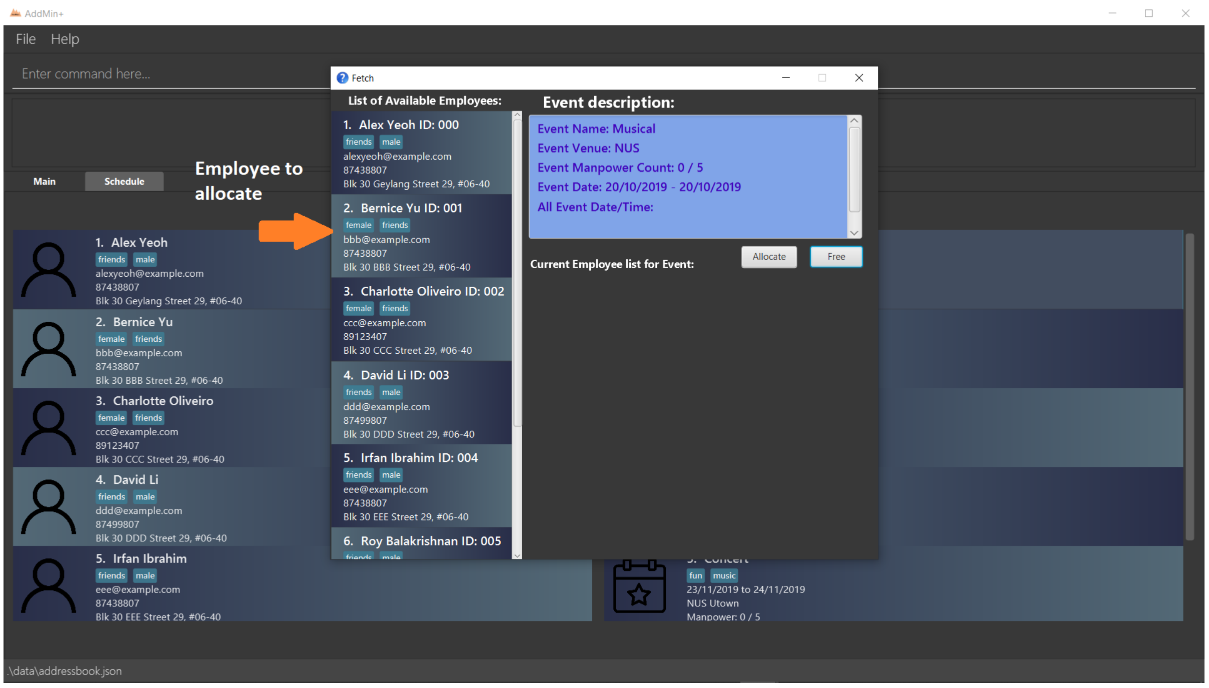
- Allocates an employee with `EMPLOYEE_INDEX` to the event at the specified `EVENT_INDEX`.
- The `EVENT_INDEX` refers to the index number shown in the displayed event list.
- The `EMPLOYEE_INDEX` refers to the index number shown in the displayed employee list.
- Note the [Constraints] for **INDEX**.

Examples:

- `allocatem 1 n/2`
  Allocates the 2nd employee on the employee list to the 1st event on the event list.

Alternatively, the `allocatem` command can be executed from the GUI in just 2 simple step!

Step 1: double-click the employee card on the left list. Notice the employee to allocate as shown in the following figure.



List of Available Employees:

1. Alex Yeoh ID: 000
friends male
alexyeoh@example.com
87438807
Blk 30 Geylang Street 29, #06-40

2. Bernice Yu ID: 001
female friends
bbb@example.com
87438807
Blk 30 BBB Street 29, #06-40

3. Charlotte Oliveiro ID: 002
female friends
ccc@example.com
89123407
Blk 30 CCC Street 29, #06-40

4. David Li ID: 003
friends male
ddd@example.com
87499807
Blk 30 DDD Street 29, #06-40

5. Irfan Ibrahim ID: 004
friends male
eee@example.com
87438807
Blk 30 EEE Street 29, #06-40

6. Roy Balakrishnan ID: 005

Event description:

Event Name: Musical
Event Venue: NUS
Event Manpower Count: 0 / 5
Event Date: 20/10/2019 - 20/10/2019
All Event Date/Time:

Current Employee list for Event:

1. Alex Yeoh
friends male
alexyeoh@example.com
87438807
Blk 30 Geylang Street 29, #06-40

2. Bernice Yu
female friends
bbb@example.com
87438807
Blk 30 BBB Street 29, #06-40

3. Charlotte Oliveiro
female friends
ccc@example.com
89123407
Blk 30 CCC Street 29, #06-40

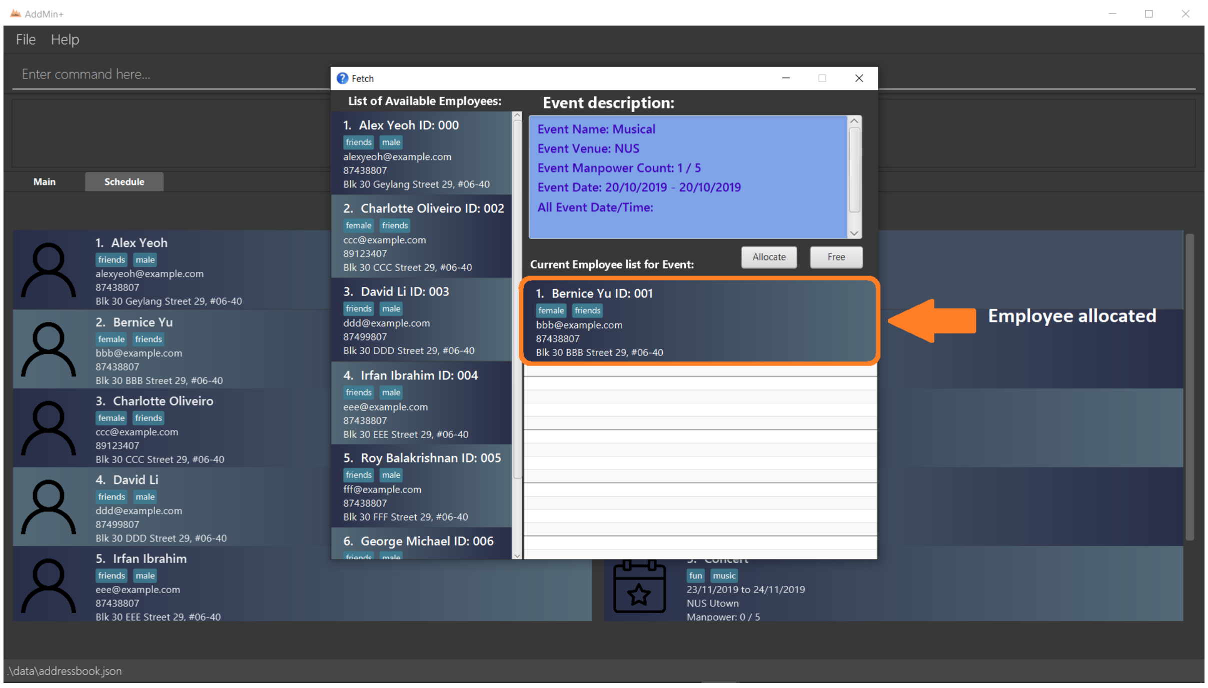
4. David Li
friends male
ddd@example.com
87499807
Blk 30 DDD Street 29, #06-40

5. Irfan Ibrahim
friends male
eee@example.com
87438807
Blk 30 EEE Street 29, #06-40

5. Concert
fun music
23/11/2019 to 24/11/2019
NUS Utown
Manpower: 0 / 5

Figure 4. Instruction for user to execute `allocatem` command

Step 2: result after the `allocatem` command is shown below:



List of Available Employees:

1. Alex Yeoh ID: 000
friends male
alexyeoh@example.com
87438807
Blk 30 Geylang Street 29, #06-40

2. Charlotte Oliveiro ID: 002
female friends
ccc@example.com
89123407
Blk 30 CCC Street 29, #06-40

3. David Li ID: 003
friends male
ddd@example.com
87499807
Blk 30 DDD Street 29, #06-40

4. Irfan Ibrahim ID: 004
friends male
eee@example.com
87438807
Blk 30 EEE Street 29, #06-40

5. Roy Balakrishnan ID: 005
friends male
fff@example.com
87438807
Blk 30 FFF Street 29, #06-40

6. George Michael ID: 006

Event description:

Event Name: Musical
Event Venue: NUS
Event Manpower Count: 1 / 5
Event Date: 20/10/2019 - 20/10/2019
All Event Date/Time:

Current Employee list for Event:

1. Bernice Yu ID: 001
female friends
bbb@example.com
87438807
Blk 30 BBB Street 29, #06-40

1. Alex Yeoh
friends male
alexyeoh@example.com
87438807
Blk 30 Geylang Street 29, #06-40

2. Bernice Yu
female friends
bbb@example.com
87438807
Blk 30 BBB Street 29, #06-40

3. Charlotte Oliveiro
female friends
ccc@example.com
89123407
Blk 30 CCC Street 29, #06-40

4. David Li
friends male
ddd@example.com
87499807
Blk 30 DDD Street 29, #06-40

5. Irfan Ibrahim
friends male
eee@example.com
87438807
Blk 30 EEE Street 29, #06-40

3. Concert
fun music
23/11/2019 to 24/11/2019
NUS Utown
Manpower: 0 / 5

Figure 5. Result after `allocatem` *command*

De-allocation of Employees from Event: `free`

Frees employees allocated to the event.

Format: `free EVENT_INDEX [id/EMPLOYEE_ID]`

- The `EVENT_INDEX` refers to the index number shown in the displayed event list.
- Note the [Constraints] for **INDEX**.
- if `EMPLOYEE_ID` is not specified, all employees allocated to the event will be removed.
- `EMPLOYEE_ID` must match the exact 3-digit ID shown in the displayed employee list.

Examples:

- `free 1`
  Frees all employees allocated to the 1st event on the event list.
- `free 1 id/001`
  Frees an employee with id: "001" allocated to the 1st event on the event list.

Alternatively, the `free` command can be executed from the GUI in just 1 simple step!

Step 1: To perform a `free` command without ID specification, click the **free** button as shown in the **Fetch Event Window** in Fetch Full Details of an Event: `fetch_ev`.

NOTE	To **free** a particular employee to an event, double-click the employee card on the right list as shown in the 2nd figure in Manually allocation of Employees to Events: `allocatem`

Contributions to the Developer Guide

Storage component

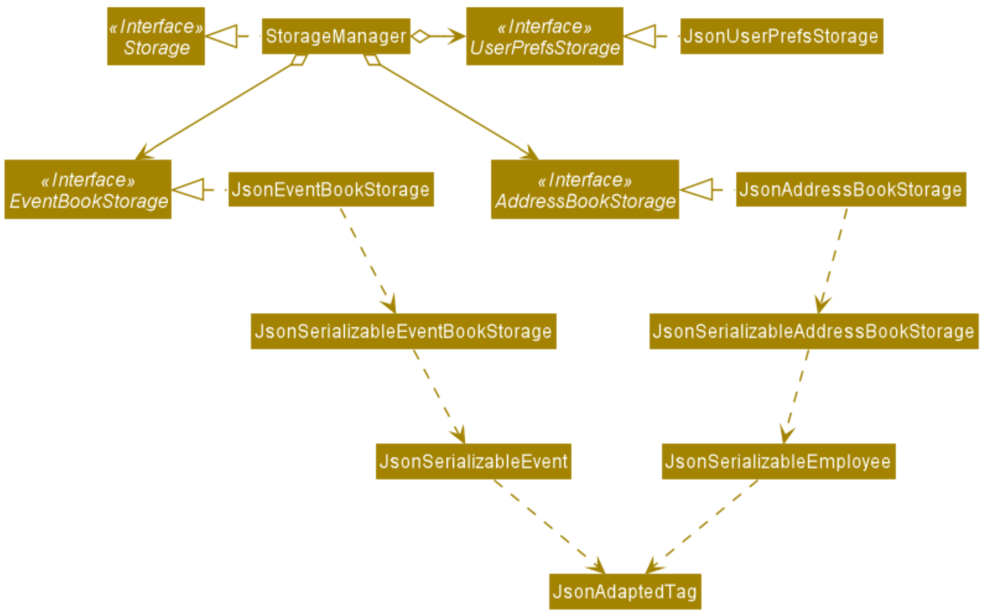


Figure 6. Structure of the Storage Component

API : `Storage.java`

The `Storage` component,

- can save `UserPref` objects in json format and read it back.
- can save the App data in json format and read it back.

Automated allocation of Employees to Events feature

Implementation

The `AutoAllocateCommand` has an auto-allocation mechanism which is facilitated by methods in `Event`. The `AutoAllocateCommand` takes in three arguments:

1. `eventIndex` - index of event in the displayed event list
2. `ManpowerCountToAdd` - number of employees to allocate [optional]
3. `tagList` - a set of tags to filter the employees [optional]

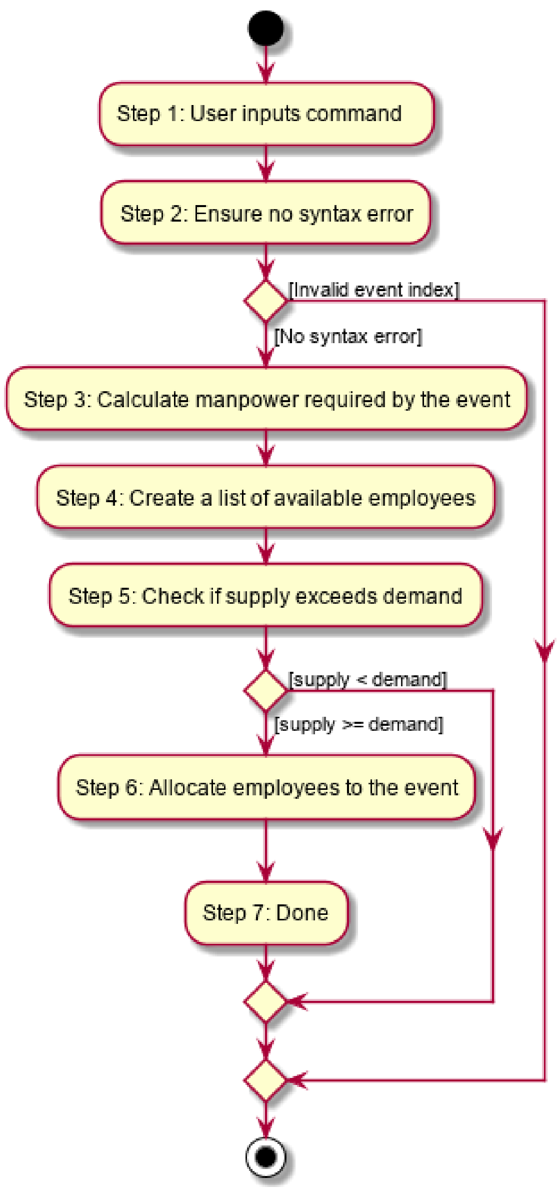
Additionally, the `AutoAllocateCommand` uses the following operations:

- `Event#isAvailableForEvent()` — Checks if an employee is available for the event.
- `AutoAllocateCommand#createAvailableEmployeeListForEvent()` — Creates a list of employees available for the event, filtered by the tags specified by user.
- `AutoAllocateCommand#getManpowerNeededByEvent()` — Calculates the number of employees

currently required by the event.

- `AutoAllocateCommand#createEventAfterManpowerAllocation()` — Creates a new event with a updated manpower list.

Given below is an example usage scenario and how the auto allocation mechanism behaves at each step.



Step 1: User inputs command
Step 2: Ensure no syntax error
[Invalid event index]
[No syntax error]
Step 3: Calculate manpower required by the event
Step 4: Create a list of available employees
Step 5: Check if supply exceeds demand
[supply < demand]
[supply >= demand]
Step 6: Allocate employees to the event
Step 7: Done

Figure 7. Program flow of the Auto Allocate Feature

Step 1. The user executes `allocate 1 n/2 t/female` with the intention to allocate 2 employees with tag [female] to the 1st event displayed in the event list.

Step 2. The command checks if `eventIndex` is valid and if `ManpowerCountToAdd` is specified.

NOTE If `ManpowerCountToAdd` is not specified, it is assumed to be the maximum number possible for the event. Validity of other command arguments e.g. if `ManpowerCountToAdd` is a positive integer is checked by `AutoAllocateCommandParser` and not within the command `AutoAllocateCommand`.

Step 3. The command calls its own method `AutoAllocateCommand#getManpowerNeededByEvent()` to get the number of employees required by the specified event.

Step 4. The command calls its own method `AutoAllocateCommand#createAvailableEmployeeListForEvent()` to create a filtered list of employees based on the `tagList` and if employee satisfies `Event#isAvailableForEvent()`.

Step 5. The command checks if supply (number of employees in filtered list in step 4) exceeds demand (number of employees required by event, generated in step 3).

NOTE	If demand exceeds supply, an exception will be thrown to the user. If the supply exceeds demand, employees will be randomly selected instead.

Step 6. The command calls `Event#createEventAfterManpowerAllocation()` to create a new event with a updated manpower list.

NOTE	For storage purposes, only the `Employee#EmployeeId` is saved in the event's manpower list.

Step 7. Done.

The following sequence diagram shows how the auto allocation works:

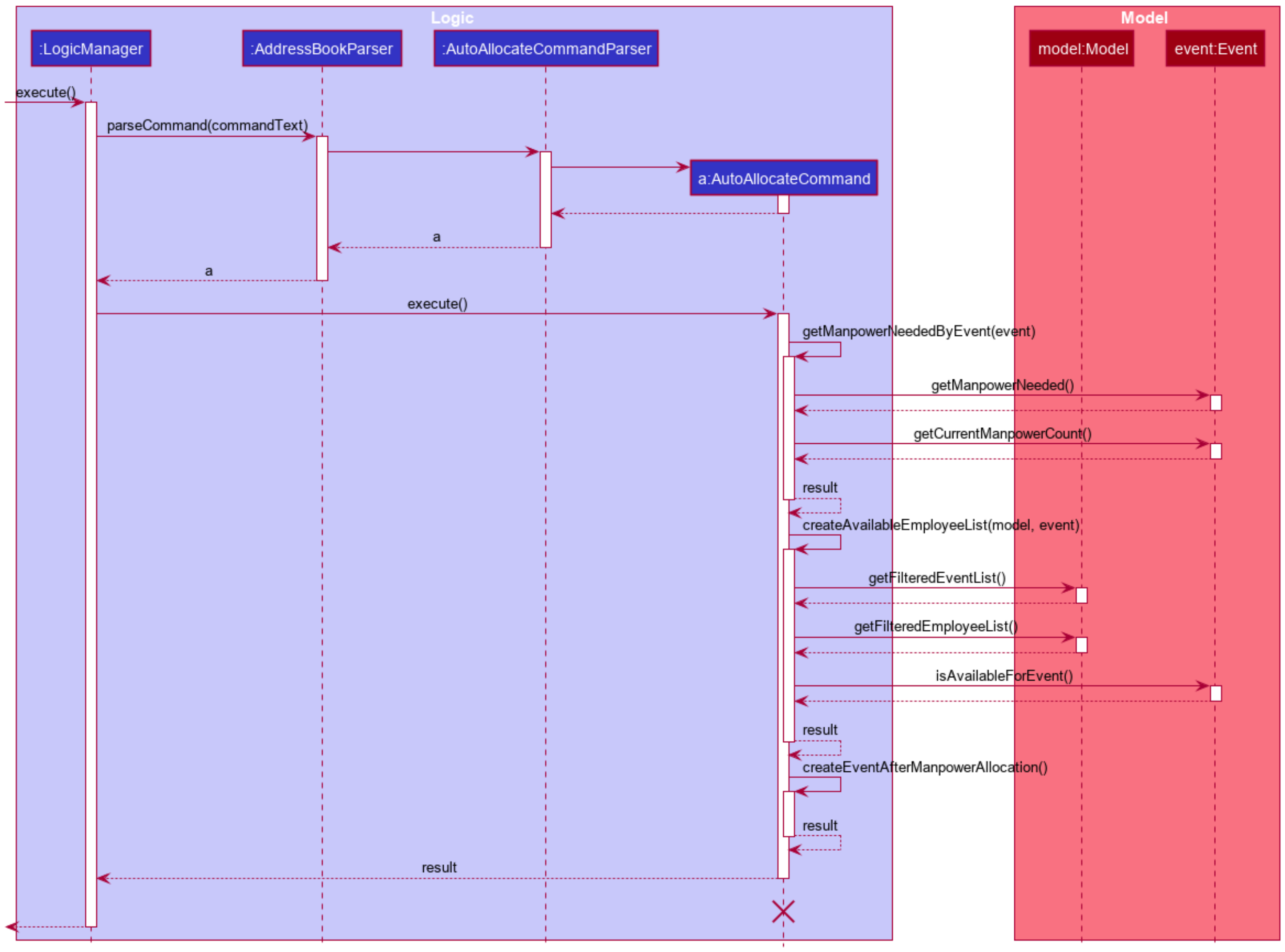

Logic
Model
:LogicManager
:AddressBookParser
:AutoAllocateCommandParser
model:Model
event:Event
execute()
parseCommand(commandText)
a:AutoAllocateCommand
a
a
execute()
getManpowerNeededByEvent(event)
getManpowerNeeded()
getCurrentManpowerCount()
result
createAvailableEmployeeList(model, event)
getFilteredEventList()
getFilteredEmployeeList()
isAvailableForEvent()
result
createEventAfterManpowerAllocation()
result
result

Figure 8. Sequence Diagram of the AutoAllocate Command

NOTE	The lifeline for `AutoAllocateCommand` should end at the destroy marker (X) but due to a limitation of PlantUML, the lifeline reaches the end of diagram.

Design Considerations

Aspect: Storage of employees associated with event after successful command

Feature	Alternative 1	Alternative 2
Storage of employees associated with event after successful command	Saves only the `Employee#EmployeeId` associated with the event. **Pros**: Easy to implement. Will use less memory. **Cons**: Future accesses require more time. **I decided to proceed with this option** because it creates less dependencies on other classes which is is a good programming practice.	Saves all fields of `Employee` associated with the event. **Pros**: Easy retrieval in the future. **Cons**: Changes in `Employee` attributes have to be reflected in the event. This meant that `EditCommand` and `DeleteCommand` for `Employee` have to be heavily modified.
Update of changes made to the manpower list of an event after the allocation of employees.	Directly modifies the `EventManpowerAllocatedList` of the specified event **Pros**: Easy to implement. **Cons**: May cause unwanted behaviours if testing is not done properly.	Create a new event with a newly created and updated manpower list. **Pros**: Defensive programming. **Cons**: Harder to implement. **I decided to proceed with this option** because it complies with the Law of Demeter which states that objects should not navigate internal structures of other objects.

Conclusion

Given the time constraints of only 6 weeks for this project, I feel that sufficient progress has been made to build a great app. However, the team has plans to turn **AddMin+** into a truly revolutionary app that changes the entire administrative process.