Ng Zhao Ming - Project Portfolio

1. Introduction

This project portfolio serves to note down my contributions to a team-based software engineering project in the NUS Computing module, CS2103T. During this module, my team of 5 were given six weeks to either *morph* or *enhance* an existing Java project, Address Book - Level 3.

1.1. Project Motivation

There are certain module requirements that our team have to adhere to when developing our application.

Constrains:

1. Our application has to cater to individuals who can type fast and have a preference for typing over other forms of input.
2. The product has to be *Brownfield*.

Our Decision:

In order to address these requirements, we have collectively decided to *morph* Address Book - Level 3 into a Traveller's Diary, *TravEzy*.

Here is how *Travezy* will look when you first start it:

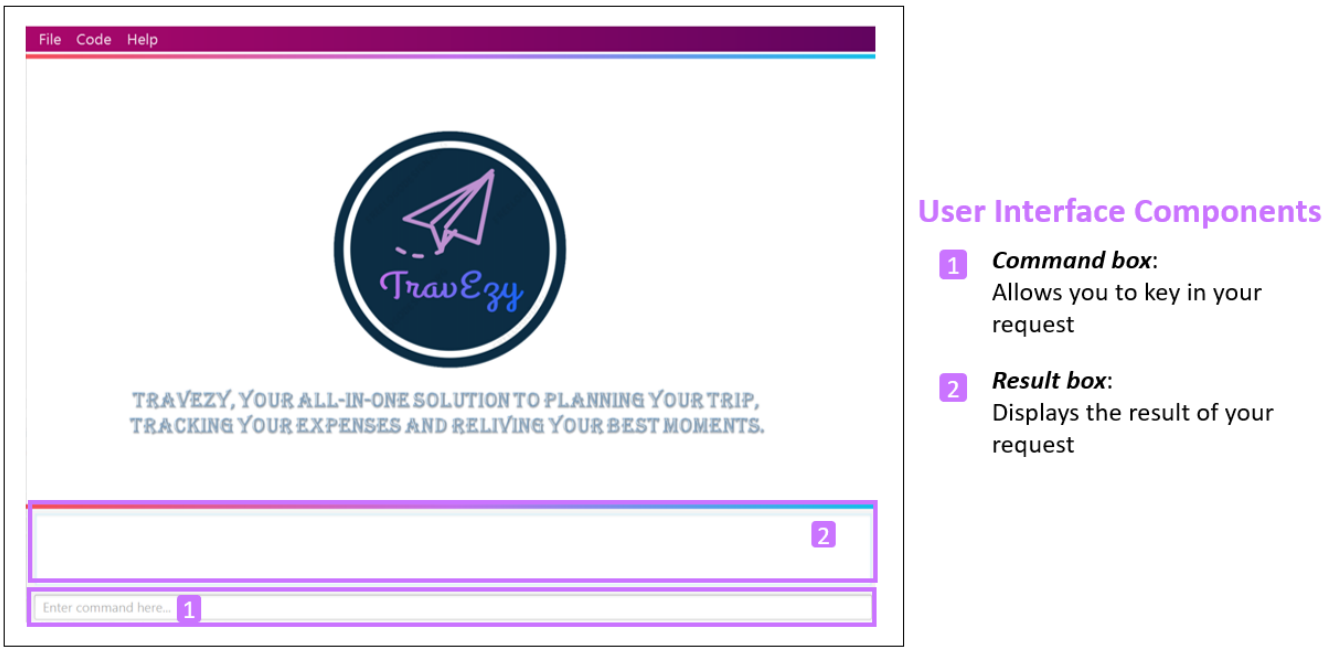


Figure 1. TravEzy Interface

Highlights of *TravEzy*:

1. As seen in *Figure 1.*, *TravEzy* utilizes the Command Line Interface (CLI) through the *command box* and in turn, most user interactions are mediated through commands keyed into *TravEzy*.
2. Is a desktop application and can be accessed on the computer without the need of a web browser.

1.2. About the project - *TravEzy*

TravEzy combines all the essential travelling application and present it as a single package. It is the ideal all-in-one application to assist globetrotters in planning their schedule, tracking their finances and relieving their best memories.

TravEzy targets NUS students who loves to backpack across several countries. As a team of 5 NUS student who have a passion for travel, we believe that we are better to understand and cater to the needs of our target audience.

Given that NUS students are constantly hustling and still love to find time to unwind and travel, *TravEzy* is ideal application for them to do so. In addition, *TravEzy* will also be useful for students who are embarking on their Semester Exchange Programmes during their junior and senior years in NUS.

The list below highlights the main features *TravEzy* has to offer:

1. Calendar
2. Itinerary
3. Financial Tracker
4. Address Book
5. Diary

1.3. Overview of my role in *TravEzy*

I was tasked to implement the itinerary feature for TravEzy. The itinerary helps users to organize their busy schedule through inputting of timed events.

The following sections illustrate these implementations in more detail. Besides that, I will also delve into the contributions I have made in the documentations of *TravEzy's* User and Developer guides.

Before we move on, I would like to highlight the symbols and text formatting that will be used in this document, which aims to improve readability of this document.

IMPORTANT	Crucial information that you should take note off while reading the document.

`command`: Text which is contained in a grey box (as can be seen on the left) refers to *commands* that the user can type into the *command box* in order to interact with *TravEzy*

Components: Italicised text indicates that the text has a definition that is specific to the application.

Highlight Bold text is mainly used for headings

1.4. Summary of contributions

Here is a summary in order of importance on the contributions that I have made during the team project.

Major enhancement:

Itinerary Feature

- What it does:
  - The Itinerary helps users to plan their schedule by organizing all their *Event Entries* in a single convenient to access *Event List*.
- Justification:
  - According to a survey that we have conducted on the general NUS student population, an overwhelming 90 percent of the responders indicated their need to organize their trip while travelling overseas. Therefore, the Itinerary provides a platform for them to do so, together with other nifty features to improve the user's experience. This makes *TravEzy's* Itinerary the frontier of convenience.
- Highlights:
  Other than being able to add, delete, check and display event entries, *TravEzy's* Itinerary also offers several complex features to bring convenience to you when organizing your schedule. This includes:
  1. Searching and sorting of events in the *event list*
  2. Ability to input your commands quicker through autocomplete

Code contributed

1. Itinerary (v1.1). Implemented the basic commands for *TravEzy's* Itinerary.
2. Itinerary (v1.2). Design the Graphic User Interface (GUI) for the Itinerary page.
3. Itinerary (v1.3). Implemented the complex commands for *TravEzy's* Itinerary.
4. Itinerary (v1.4). Refine Itinerary, fixed bugs and work on *TravEzy* documentation.
5. RepoSense. Keeps track of my code contribution to the team.

Minor enhancement:

***TravEzy* Design**

- Designed the *Event Card* for the Itinerary
- Designed *TravEzy* application logo

Other contributions:

- Project management:
  - Created and managed milestones (V1.1 - V1.3)
  - Assigned labels to teammates to remind them of key deadlines through Github Issues
  - Tested out *TravEzy* software and fixed bug: #98 #145 #154
- Documentation:
  - Draft a skeletal of the User and Developer guide for the team to start working on: 1, 2, 3
  - Draw the Architecture Diagram for *TravEzy* in the Developer Guide
  - Proof-read and refine the *QnA* section of the User Guide: #101 #161 #198
- Community:
  - Reported bugs and suggestions for other teams
  - Proof-read other teams User and Developer guide: #61

2. Contributions to the User Guide

Now, I will be sharing the documentation that I have contributed to the User Guide. It manifests my ability to provide clear and concise instructions for end-users to use *TravEzy*. I will be focusing on 2 relevant commands which are `sort` and `search`.

2.1. Searching for an event: `search`

Does your *event list* too many events to browse through? Let *TravEzy* Itinerary help you by searching for events that matches specific keywords.

TIP	If your *event list* is too cluttered up, try using the `clear` command instead to reset the **whole** *event list* instead.

Format:

`search SEARCH CONDITION`

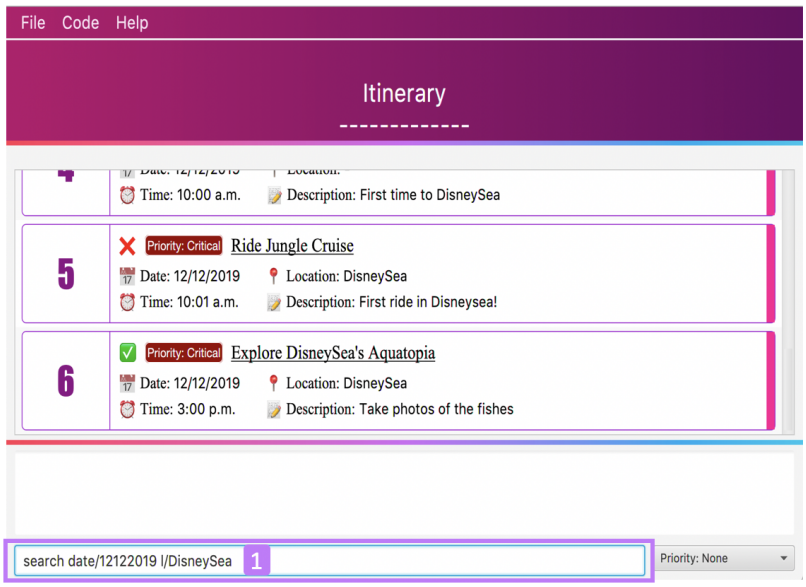
where `SEARCH CONDITION` can be any of the following: `title/TITLE`, `date/DATE`, `time/TIME`, `l/LOCATION`, `d/DESCRIPTION` or `tag/`

Example:

```
search date/12122019 l/DisneySea
```

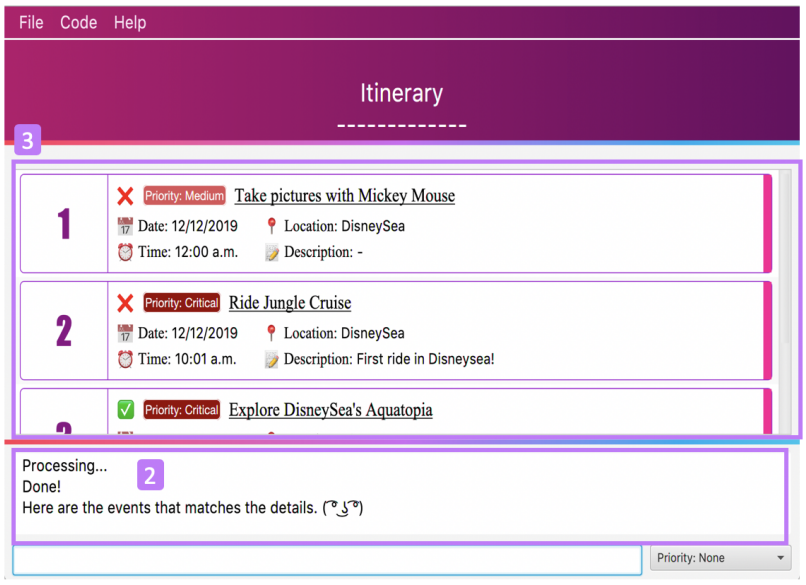
Step by step:

Step 1. Type `search` in the *command box* and press *Enter*.



Step 2. The *result box* will display the message "Processing... Done! Here are the events that matches the details. (□° □□ □°)"

Step 3. The filtered *event list* will be shown containing events that matches the keywords given.



2.2. Sorting the events: `sort`

Organizing your events in the *event list* has never been easier the Itinerary's awesome sorting capabilities.

Format:

`sort by/SORT CONDITION`

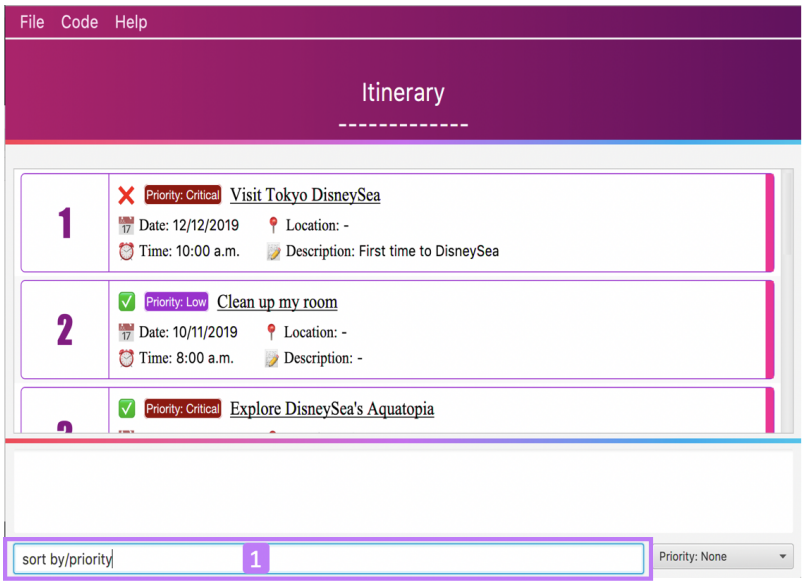
where `SORT CONDITION` can be any of the following: `title`, `location`, `chronological`, `completion` or `priority`

Example:

```
sort by/priority
```

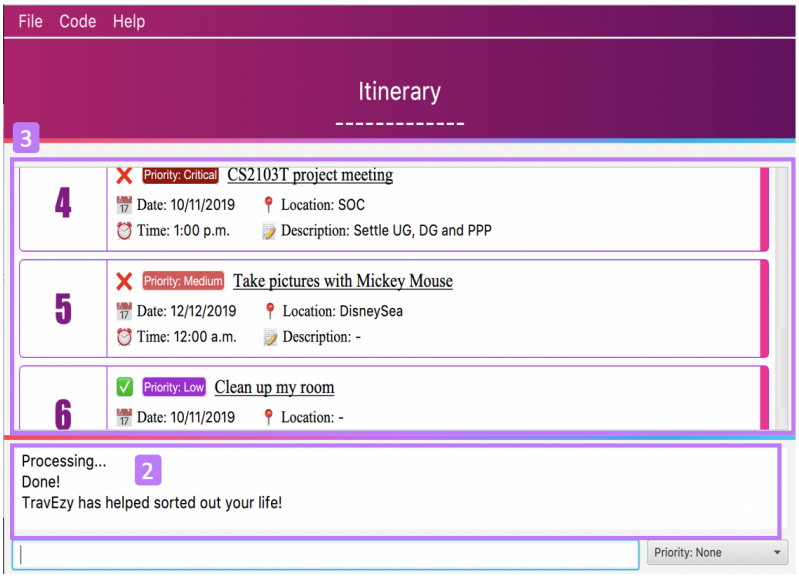
Step by step:

Step 1. Type `sort by/priority` in the *command box* and press *Enter*.



Step 2. The *result box* will display the message "Processing... Done! TravEzy has helped sorted out your life!"

Step 3. The Itinerary will present the sorted *event list* based on the sort condition given.



3. Contributions to the Developer Guide

Finally, I will be sharing the documentation that I have contributed to the Developer Guide. The Developer Guide showcase my ability to write technical documentation and the technical depth of my contributions to the project.

3.1. Itinerary

The itinerary feature in TravEzy allows users to organize their events and view these events in one convenient *Event List*.

Current, the itinerary feature supports the basic commands of add, delete, edit and marking your events as done. In addition, it also includes other more advanced commands such as search and sort to better organize your events. With these implementations, *TravEzy* aims to be at the frontier of convenience.

The itinerary feature implements the aforementioned feature based on the use cases below:

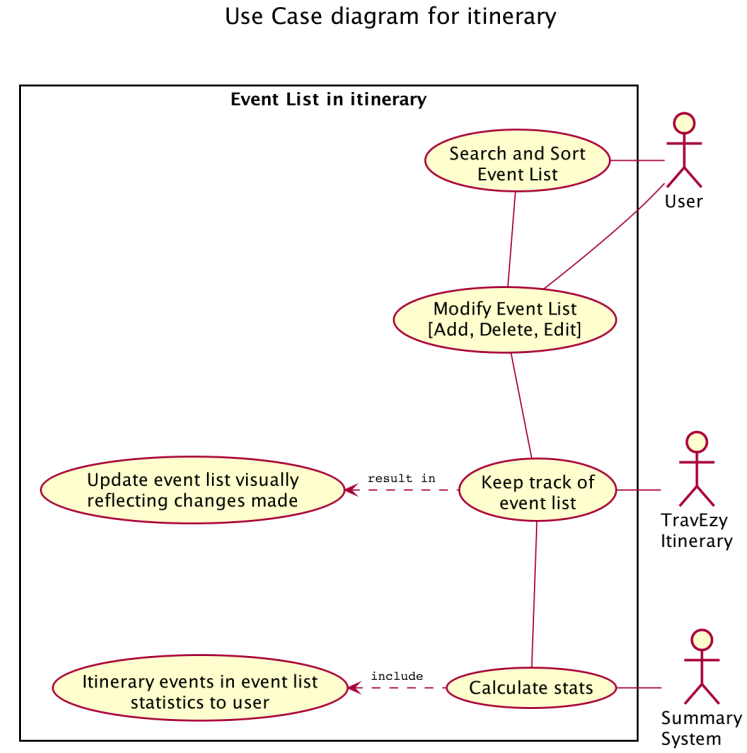


Figure 2. Use Case Diagram of the Itinerary feature

Due to the numerous features supported by the itinerary, it requires a complex structure to ensure that each input by the user are cautiously parsed before giving the appropriate command result.

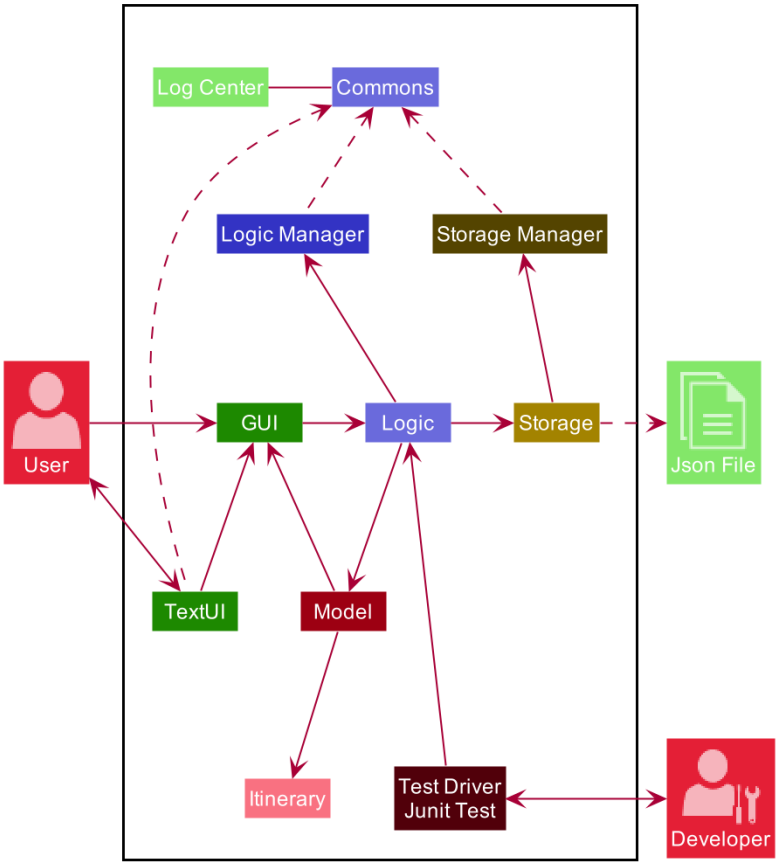


Figure 3. Architecture Diagram of the Itinerary feature

The ***Architecture Diagram*** given above explains the high-level design of the itinerary feature.

Inputs given by the user are channeled from the text UI and parsed in the logic package before different commands are formed which generates the model and updates the itinerary object which contains the event list.

The `text UI`, `logic manager` and `storage manager` all stem from the common package of the main TravEzy application. However, in the `parser` package of itinerary, it contains various parser objects for the different command. This is to ensure that each command in the itinerary have only one parser validating the command.

3.1.1. Model Component

The implementation of the model class in TravEzy is to be a generic. Hence, the model object being instantiated could be any of the following 5 features, `Calendar`, `Itinerary`, `Financial Tracker`, `Travel Diary` and `Achievements`. Below is the model class diagram for the itinerary feature:

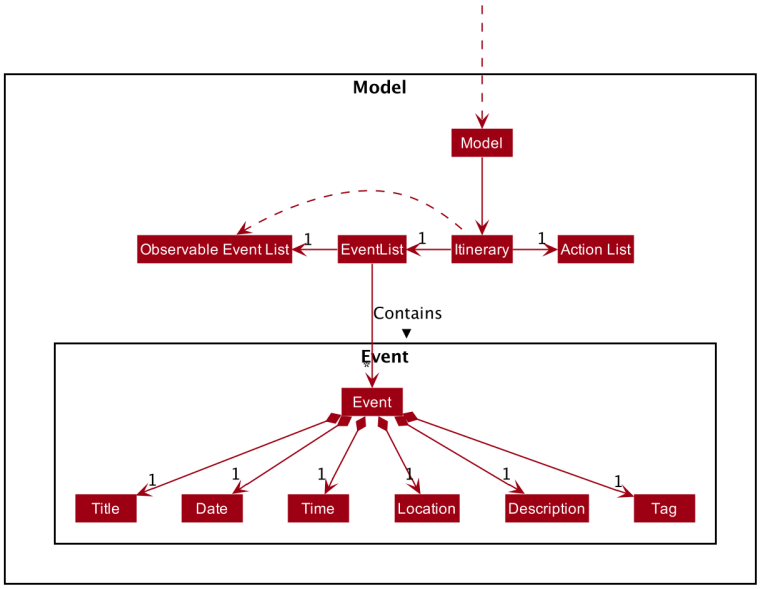


Figure 4. Model Diagram of the Itinerary feature

The Model, is the crux of the itinerary feature which serves several functions this include:

- stores the Itinerary data which includes the event list which keeps track of all the events that are included by the user and stores it into the storage in a JSON file.
- exposes an unmodifiable ObservableList<Event> that can be 'observed' e.g. the UI can be bound to this list so that the UI automatically updates when the data in the list change.
- does not depend on any of the other three components, UI, logic and storage which are common through all the features throughout TravEzy.

3.1.2. Itinerary search command

The search command for specified events in the event list is facilitated by the `Itinerary` class which contains an event list and keep track of the events that the user has inputted into TravEzy. There are several search conditions available for the users to search from based on the different class attributes that form the `Event` class:

- search title/[title]
- search date/[date]
- search time/[time]
- search l/[location]
- search d/[description]

Given below is the *sequence diagram* of how the Itinerary feature interacts with TravEzy when the search command is being called.

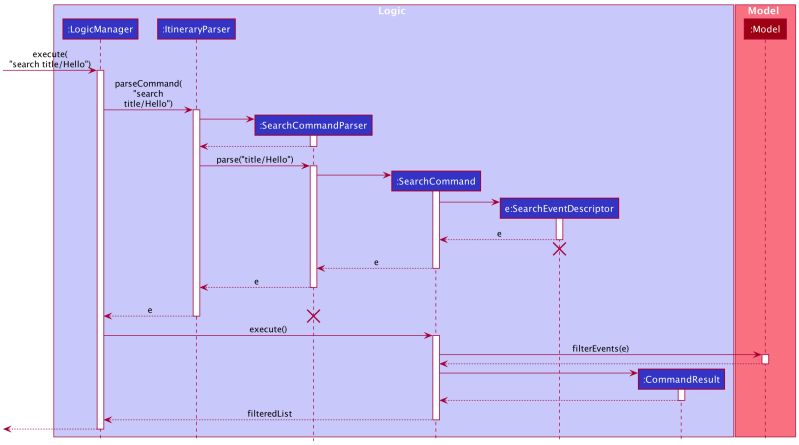


Figure 5. Sequence Diagram of the Search Command

IMPORTANT	The lifeline for SearchCommandParser and SearchEventDescriptor should end at the destroy marker (X) but due to a limitation of PlantUML, the lifeline reaches the end of diagram.

The search command is implement as follows, upon giving the command by the user in the text UI, the command will be channeled to the `Logic` class where it identifies it as a itinerary command and passes it to the `ItineraryParser` class.

The `Itinerary Parser` passes through the command into a switch case block and identifies this as a `SearchCommand`. This will create a new `SearchCommandParser` which will then accept the arguments from the user's input and parse the arguments of the command.

Once the arguments are parsed and considered as valid, the `SearchCommandParser` will generate a new `SearchCommand`. The `SearchCommand` will in turn create a "pseudo-event" known as the `SearchEventDescriptor` which is an event which only contains attributes with the search condition while the rest of it's attribute will be placed as null.

This `SearchEventDescriptor` will in turn be returned and used in the `Predicate` field as the event in comparison. The `filterEvents(e)` method will be called with `e` being the `SearchEventDescriptor` that is being generated. Events that are currently in the event list will be filtered accordingly based on whether it matches the attributes in the `SearchEventDescriptor`. Finally, the `filteredList` will be generated and returned.