



Department of Computer Science and Information Systems,

CS222: Data Structures and Algorithms

Final Project - Event Scheduler and Calendar

Group 20

Dr. Govindha R Yeluripati

Group Members

Delice Ishimwe 42402026

Steve Nsabimana 36422026

Maxwell Antwi 80082025

August 3, 2024

The Event Calendar Management System

1. Project Overview

The Event Calendar Management System is a user-centric application designed with a focus on enhancing the end-user experience through its intuitive graphical user interface (GUI). The system allows users to efficiently create, modify, delete, view, search, and sort events. A notable feature of our application is its ability to prevent scheduling conflicts by ensuring that no multiple events occur at the same datetime, which helps users manage their schedules more effectively. The system clearly displays upcoming events on the dashboard, while also allowing access to and organization of both current and completed past events. The GUI is crafted to be interactive and visually appealing, with clear, well-structured summaries of events within a specified date range, ensuring a comprehensive and user-friendly scheduling experience.

To achieve optimal performance and efficiency in our event management system, we made several key design decisions:

Features

- **Dashboard:** Displays a table of events with options to refresh, remove, and update events.
- **Add Event:** Form to input details for new events, including title, location, description, date, start time, end time, and priority.
- **Search Events:** Allows users to search for events by attributes such as title, location, priority, description, and date.
- **Sort Events:** Provides options to sort events by title, date, or priority.
- **Event History:** Displays past events.
- **Generate Summary:** Generates a summary of events within a specified date range.
- **Update button :** To modify past events
- **Remove button:** to remove selected event

2. Objectives

The primary objectives of the project are:

- To enable users to create and manage events with various attributes such as title, date, time, location, description, and priority.
- To prevent scheduling conflicts by ensuring no two events are set for the same datetime.
- To provide an intuitive interface for performing Create, Read, Update, and Delete (CRUD) operations on events.
- To implement sorting and searching functionalities for easy event management.
- To generate summaries of events based on user-defined date ranges.
- To offer an interactive and visually appealing interface for seamless navigation and scheduling.

- **Data Structures :**

- We utilized a *HashMap* as the main data structures to store events in the dashboard and the history. The decision to use *HashMap* was driven by its efficient average-time complexity for insertions, deletions, and lookups, which are crucial for managing and retrieving events quickly. Unlike other data structures, the *HashMap* provides constant-time performance for basic operations, ensuring that our system can handle large datasets with minimal delay. This choice enhances the overall responsiveness and performance of the application, allowing users to efficiently manage their schedules and access event information promptly.
 - During sorting and filtering, we utilized Lists, particularly *ArrayList* to hold the events temporary. The choice for using *arraylist* was its flexibility to allow for dynamic sizing, providing efficient access and modification.
- **Sorting with QuickSort:** For sorting events, we selected the *QuickSort* algorithm due to its efficient average-time complexity of $O(n \log n)$. QuickSort's divide-and-conquer approach allows it to handle large datasets efficiently, providing a fast and responsive

user experience. Although other sorting algorithms exist, *QuickSort* strikes a balance between speed and simplicity, making it well-suited for our sorting needs.

- **Search Functionality with Hash Table Search:** The `searchEvent(LocalDate date, LocalTime time)` method uses *hash table search* to locate events based on their date and time. This method is chosen because *hash table search* offers constant-time complexity on average for lookups. By using *hash table search*, we ensure quick and efficient retrieval of events, even with a large number of entries, contributing to the overall performance of the application.

➤

3. Design and Implementation Details (code structure, description of classes)

A. Main Class: GUI

The GUI class is a graphical user interface for managing calendar events. This class is part of a larger event management system and provides functionalities to add, search, sort, view history, and generate summaries of calendar events. It extends `JFrame` from the Swing library to create a windowed application.

he application, and managing the graphical user interface (GUI).

Methods

private Component TabComponent(String title)

Creates a custom tab component with a specified title.

private JPanel createDashboard()

Sets up the dashboard panel which includes:

- A table (`JTable`) to display events.
- Buttons for refreshing, removing, and updating events with their respective action listeners.

private JPanel createAddEventPanel()

Creates the panel for adding new events, consisting of:

- Form fields for event details (title, location, description, date, start time, end time, priority).
- An "Add Event" button with an action listener to handle event addition.

private JPanel createSearchPanel()

Sets up the search panel which allows users to:

- Select a search attribute (title, location, priority, description, date).
- Enter a search value.
- Display search results in a text area.

private JPanel createSortPanel()

Creates the sort panel to:

- Select a sorting criterion (title, date, priority).
- Display sorted events in a table.

private JPanel createSummaryPanel()

Sets up the panel for generating event summaries:

- Input fields for specifying the date range.
- A text area to display the generated summary.

private JPanel createHistoryPanel()

Creates the panel to view past events:

- A text area to display event history.
- A button to refresh the history.

Event Handling Methods

private void addEvent()

Handles the addition of a new event:

- Retrieves input from form fields.
- Validates and converts date and time.
- Creates a new Event object and adds it to the calendar.
- Refreshes the event table and clears form fields upon successful addition.

private void removeSelectedEvent()

Removes the selected event from the table:

- Retrieves the selected event's start time.
- Removes the event from the calendar and refreshes the table.

private void updateSelectedEvent()

Updates the selected event:

- Retrieves the selected event's start time.
- Opens a dialog with pre-filled event details.
- Allows the user to modify and save changes.
- Updates the event in the calendar and refreshes the table.

private void refreshEventTable()

Refreshes the event table to reflect the current state of events in the calendar:

- Retrieves sorted events.
- Updates the table model with the latest event data.

B. Event Class: Event

The Event class represents an individual event in the calendar.

Attributes:

- **title:** The title of the event.
- **startTime:** The start time of the event, represented by a LocalDateTime object.
- **endTime:** The end time of the event, represented by a LocalDateTime object.
- **location:** The location where the event takes place.
- **id:** A unique identifier for the event.
- **description:** A description providing details about the event.
- **priority:** The priority level of the event.

Key Methods:

- **Event(*String title, LocalDateTime startTime, LocalDateTime endTime, String location, String id, String description, String priority*)**: Initializes an event with the specified attributes. Throws `IllegalArgumentException` if `endTime` is before `startTime`
- **String toString()**: Provides a string representation of the event, formatted with date and time using `DateTimeFormatter`.
- **boolean equals(Object o)**: Checks for equality with another object based on the event's id. Returns true if they are the same, otherwise false.
- **int hashCode()**: Returns a hash code value for the event, based on its id.

C. Calendar Class: Calendar

The **Calendar** class manages a collection of Event objects, providing functionality to add, remove, update, view, sort, and refresh events. It also maintains a history of past events and allows for generating summaries within specified date ranges.

Attributes:

- **events**: A `Map<String, Event>` that stores the current events, with the event's start time (converted to a string) as the key.
- **history**: A `Map<String, Event>` that stores past events that have already occurred, also using the event's start time as the key.
-

Key Methods:

- **Calendar()**: Constructor that initializes the events and history maps as empty.

- **boolean addEvent(Event newEvent):** Adds a new event to the events map if there is no conflict with existing events. Returns true if the event was added successfully, and false if there is a time conflict.
- **private boolean eventsOverlap(Event first, Event second):** Checks if two events overlap in time. Returns true if they overlap, false otherwise.
- **void removeEvent(String key):** Removes an event from the events map based on the specified key (start time). Prints a message indicating success or failure.
- **void updateEvent(String key, Scanner scanner):** Updates an event based on user input. Allows changes to title, description, location, priority, and start/end times. Validates new times and updates the event if valid.
- **void displayAllEvents():** Displays all current events sorted by their start time.
- **List<Event> viewEvents(String attribute, String filterValue):** Filters and returns a list of events based on a specified attribute (title, location, priority, description, or date) and filter value. Sorts the filtered events by start time.
- **List<Event> sortEvents(String attribute):** Sorts events based on a specified attribute (date, title, or priority) using the quicksort algorithm. Returns a list of sorted events.
- **private <T> void quicksort(List<T> list, Comparator<? super T> comparator, int low, int high):** Performs the quicksort algorithm on a list based on the provided comparator.
- **private <T> int partition(List<T> list, Comparator<? super T> comparator, int low, int high):** Partitions the list for the quicksort algorithm. Returns the partition index.
- **Event searchEventByDatetime(LocalDateTime dateTime):** Searches for an event by its start date and time. Returns the event if found, or null otherwise.
- **void refreshEvents():** Moves past events from the events map to the history map based on the current date and time.
- **List<Event> getHistoryEvents():** Retrieves a list of all past events from the history map.
- **String generateSummary(LocalDate startDate, LocalDate endDate):** Generates a summary of events within a specified date range, including both current and past events. Returns the summary as a string.

Data Structures

HashMap:

- We used a *HashMap* to store events with unique keys generated from the event's date and time. These keys serve as unique identifiers that ensure each event can be quickly and efficiently accessed. The values in the *HashMap* are Event objects that encapsulate all relevant details such as title, location, description, and priority. This design choice leverages the average constant-time complexity $O(1)$ for insertions, lookups, and deletions provided by *HashMap*, optimizing performance for frequent data retrieval and management. The constant-time complexity on average makes *HashMap* ideal for scenarios where quick access to events is critical, while the worst-case scenario of $O(n)$ due to hash collisions is mitigated by good hash function design and load factor management.

ArrayList:

- We used *ArrayList* to handle sorting and searching operations on events. Although the primary data storage is managed with *HashMap* for its efficient $O(1)$ average-time complexity for insertions and lookups, *ArrayList* is utilized for operations that benefit from ordered data. The *ArrayList* holds references to Event objects stored in the *HashMap*, allowing efficient iteration sorting, and searching based on various attributes such as date, title, or priority. Sorting operations in the *ArrayList* utilize efficient algorithms such as QuickSort, with a time complexity of $O(n \log n)$, while searching in an unsorted *ArrayList* has a linear-time complexity of $O(n)$. This arrangement allows us to leverage the strengths of both data structures: the constant-time complexity of *HashMap* for quick access and *ArrayList*'s flexibility for ordering and sorting, ensuring effective event organization and retrieval.

Algorithms

QuickSort:

- We used *quicksort* to sort events by date and time, leveraging its efficient average time complexity of $O(n \log n)$. The divide-and-conquer approach of QuickSort allows our system to handle and sort a large number of events quickly, thus providing an organized and responsive user experience.

Hash table Search:

- In our project, we employed a hash table search algorithm to efficiently search for events by date and time. Hash table search offers an average time complexity of $O(1)$ for lookups, making it highly effective for managing a large number of events. By leveraging a hash table, we can map each event's date and time to a unique key, where the key is the datetime and the value is the event. This allows for constant-time retrieval of events. This approach ensures that our event management system remains exceptionally responsive and well-organized, providing quick access to events based on their scheduled date and time.

5. Performance (time complexity) analysis and optimization techniques

The application efficiently manages events using a HashMap for constant-time complexity on average for insertions and deletions. The QuickSort algorithm ensures efficient sorting with an average time complexity of $O(n \log n)$, and the hash table search operation provides $O(1)$ complexity for searching events. These choices ensure that our system remains responsive and efficient, effectively managing and organizing events while providing a seamless user experience.

Time Complexity Calculations

- **Adding an Event:**
 - Checking for conflicts: $O(n)$ for iterating over existing events.
 - Insertion into HashMap: $O(1)$ on average.
 - **Overall Complexity:** $O(n)$ due to conflict checking.
- **Removing an Event:**
 - Deletion from HashMap: $O(1)$ on average.

- **Overall Complexity:** $O(1)$.
- **Updating an Event:**
 - Removing and re-adding an event: $O(1)$ for each operation.
 - **Overall Complexity:** $O(1)$.
- **Displaying All Events:**
 - Sorting events: $O(n \log n)$ due to sorting complexity.
 - **Overall Complexity:** $O(n \log n)$.
- **Filtering Events:**
 - Iterating over all events: $O(n)$.
 - Sorting filtered results: $O(m \log m)$ where m is the number of filtered events.
 - **Overall Complexity:** $O(n + m \log m)$.
- **Sorting Events:**
 - Quicksort: $O(n \log n)$ on average.
 - **Overall Complexity:** $O(n \log n)$.
- **Refreshing Events:**
 - Iterating over events: $O(n)$.
 - **Overall Complexity:** $O(n)$.

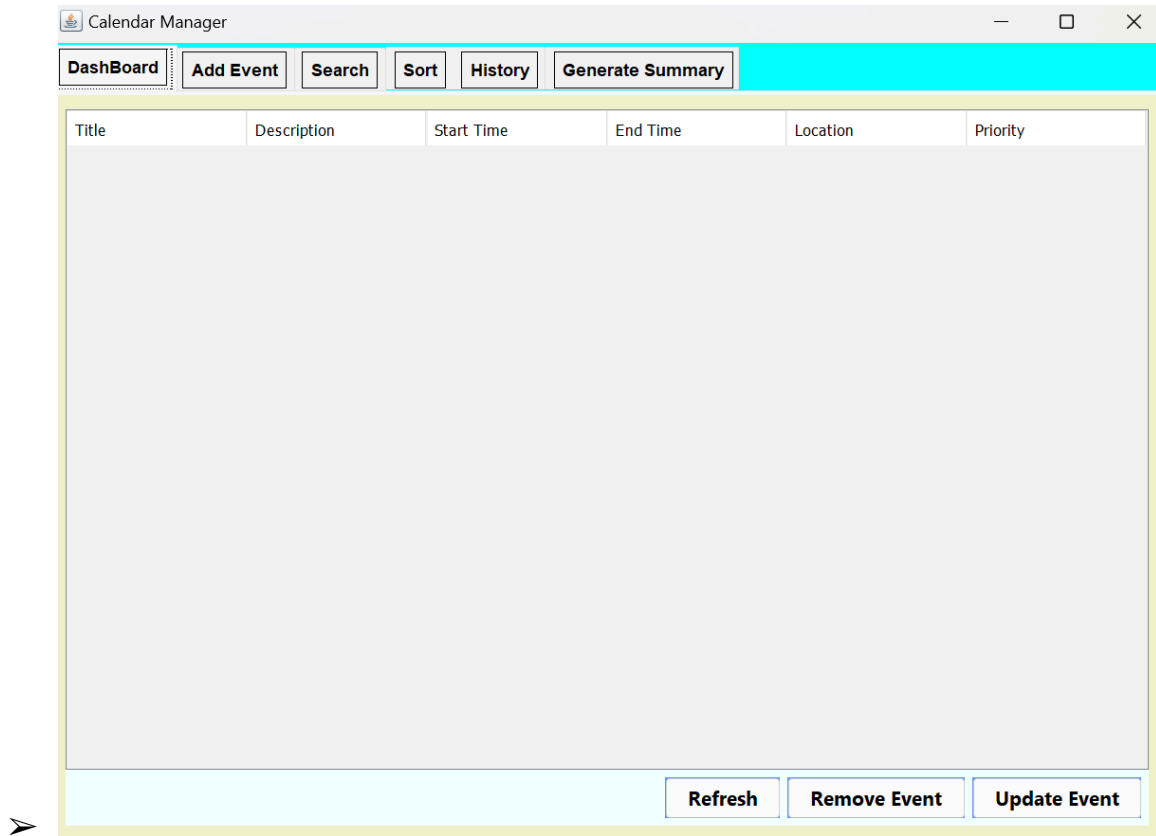
6. Challenges Faced and Solution Implemented

- **Challenge 1: Algorithm Selection:** Choosing the right sorting and searching algorithms required careful consideration of their time complexity, space usage, and how well they fit with our system's requirements. This process involved multiple rounds of testing, discussions, and refinements to ensure the algorithms provided optimal performance for our event management system.
- **Solution:** We selected QuickSort for its efficiency in sorting large datasets, ensuring rapid organization of events. For searching, we utilize hash table search due to its average time complexity of $O(1)$, which allows for swift retrieval of events by their scheduled date and time. This combination of QuickSort for sorting and hash table search for retrieval ensures that our event management system remains both fast and responsive.

- **Challenge 2: Date and Time Parsing:** Ensuring that date and time inputs are correctly formatted and accurately parsed was crucial. We had to handle various formats and exceptions to prevent incorrect event scheduling or data entry errors.
- **Solution:** We implemented robust parsing to handle various date and time formats and exceptions, ensuring accurate event scheduling.
- **Challenge 3: User Input Validation:** Managing user inputs posed a challenge, as it was essential to validate that the data entered was accurate and met the system's requirements. This included preventing invalid entries that could disrupt the scheduling process or lead to errors in event management.
- **Solution:** We incorporated exception handling to manage and validate user inputs, ensuring data accuracy and preventing input-related errors.
- **Challenge 4: Event Modification:** Allowing users to modify event attributes while maintaining data integrity was challenging. We needed to ensure that changes were accurately reflected and did not result in inconsistencies or conflicts within the event calendar.
- **Solution:** We developed mechanisms to update event details while maintaining data integrity and preventing scheduling conflicts.

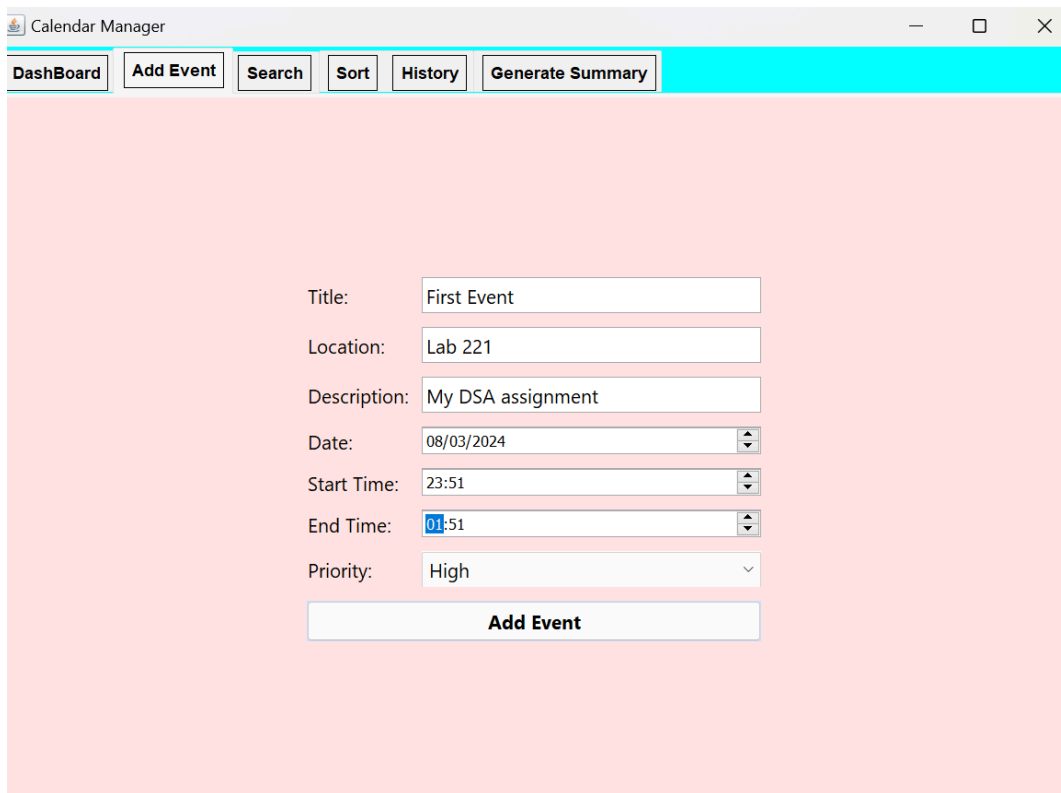
8. Instructions for Using the Application (with Sample Screenshots)

- **Run the Application:** Execute the GUI class to start the application. When started, it will show the dashboard where all events and different tabs and buttons can be seen. Initially, no events can be found there because nothing has been added.



- **To add an event:** Click on the add event tab on the top left to open the pop up allowing you to add an event. By default, the priority is High. click the drop down to change the

priority as well.

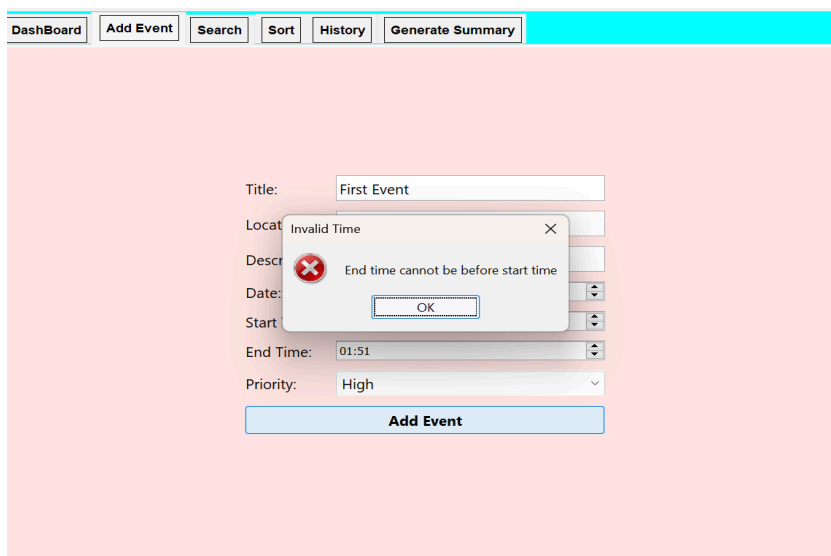


The screenshot shows a window titled "Calendar Manager" with a cyan header bar containing buttons: "DashBoard", "Add Event", "Search", "Sort", "History", and "Generate Summary". The "Add Event" button is active. The main area has a light pink background and contains a form with the following fields:

- Title: First Event
- Location: Lab 221
- Description: My DSA assignment
- Date: 08/03/2024
- Start Time: 23:51
- End Time: 01:51
- Priority: High

Below the form is a button labeled "Add Event".

- The program will automatically choose current time and date. Change these times and dates. If the End time is before the start time, the program will prompt a warning to choose the right time.



The screenshot shows the same "Calendar Manager" window as before, but with an error dialog box displayed over the form. The dialog box has a red "X" icon and the text "Invalid Time" and "End time cannot be before start time". It has an "OK" button. The form fields are partially visible behind the dialog box.

- Once an event is added, a success message will appear and it will be added to the Dashboard. The table can be adjusted in case the texts do not appear in full. It can be dragged to expand or contract.
- When a new event is added, it will be sorted based on date with the earlier date being on top.

Title	Description	Start Time	End Time	Location	Priority
Second Event	Meet team	08/03/2024 23:05	08/03/2024 23:06	Conference room	Low
First Event	My DSA assignment	08/03/2024 23:30	08/03/2024 23:59	Lab 221	High

Dashboard

Add Event

Search

Sort

History

Generate Summary

Refresh


Remove Event

Update Event

- When adding new event that conflicts with the old ones, a warning message will display showing that no two events will occupy same time.

Title:

Time Slot Occupied

 The selected date and time (08/03/2024 23:30) is already used by another event. Please choose a different time.

OK

End Time:

Priority:


Add Event

- When the date time of a particular event passes, it will be moved to the history. Once, you click on the refresh button on the dashboard, the dashboard will be updated. Only current events will remain and the past events will move to history.
- To see these events from history, click on Refresh History button at the bottom of the panel. And this will display all the past events
- To remove or update an event, select the event on the Dashboard before clicking on the Remove or Update Event. If not, you will get a warning message.

DashBoard **Add Event** **Search** **Sort** **History** **Generate Summary**

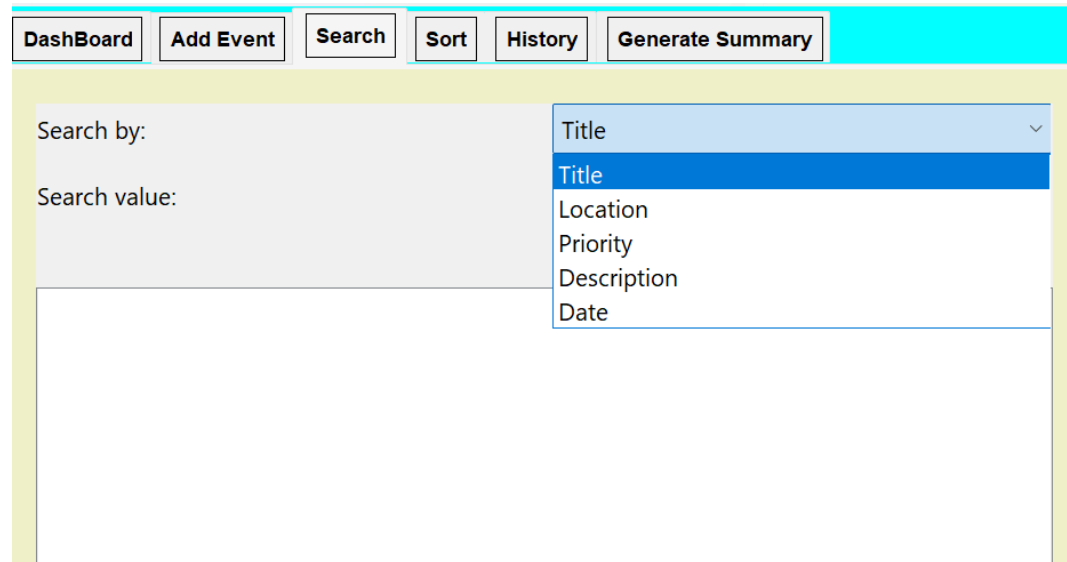
Title	Description	Start Time	End Time	Location	Priority
Event	ssss	09/03/2024 02:12	09/03/2024 02:12	sss	High

No Selection

 Please select an event to remove.

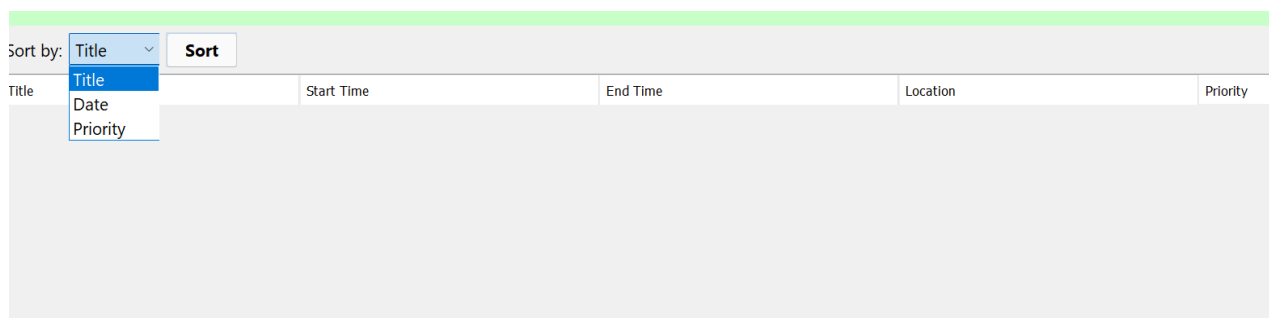
OK

- To search for event, click on the Search tab. This will ask you to select the dropdown to show which attribute you want to search by and enter what you want to search based on what you selected, then click on the search button. If not found, it will prompt you. If found it will print it out on the display.



The screenshot shows the 'Search' tab selected in the top navigation bar. Below the navigation bar, there is a search form with two input fields: 'Search by:' and 'Search value:'. A dropdown menu is open next to the 'Search by:' field, showing a list of attributes: Title, Location, Priority, Description, and Date. The 'Title' option is currently selected and highlighted in blue.

- To sort for an event, you will be provided 3 main attributes, either to sort by date, title or priority. If date, it will sort the events based on the earlier date, and if title, it will print it alphabetical order. As well with priority, it's either High or Low. Once you click on sort, it will provide the sorted events.



The screenshot shows the 'Sort' tab selected in the top navigation bar. Below the navigation bar, there is a sort form with a 'Sort by:' dropdown menu and a 'Sort' button. The dropdown menu is open, showing a list of attributes: Title, Date, and Priority. The 'Title' option is currently selected and highlighted in blue. Below the sort form, there is a table with columns: Title, Start Time, End Time, Location, and Priority. The table is currently empty.

- To generate a Summary based on time range, you click on the generate summary. This allow you to enter the start and end date for the summary and it will Generate the summary within this time range.

DashBoard	Add Event	Search	Sort	History	Generate Summary
Start Date (MM/dd/yyyy): 04/05/2024		End Date (MM/dd/yyyy): 09/01/2024		Generate Summary	

Event Title: Second
Start Time: 08/04/2024 4:26 PM
End Time: 08/04/2024 6:26 PM
Location: fAB
ID: 9dc59dcc-67f6-4362-a5a6-2d863e6e378b
Description: CAPSTONE
Priority: High

Event Title: committee meeting
Start Time: 08/05/2024 5:27 PM
End Time: 08/05/2024 6:27 PM
Location: conference room
ID: d4173f43-1272-4922-b225-73f96f7335f3
Description: Meet for ASC
Priority: Low

APPENDIX

UML

UML: Event Scheduler and Calendar

