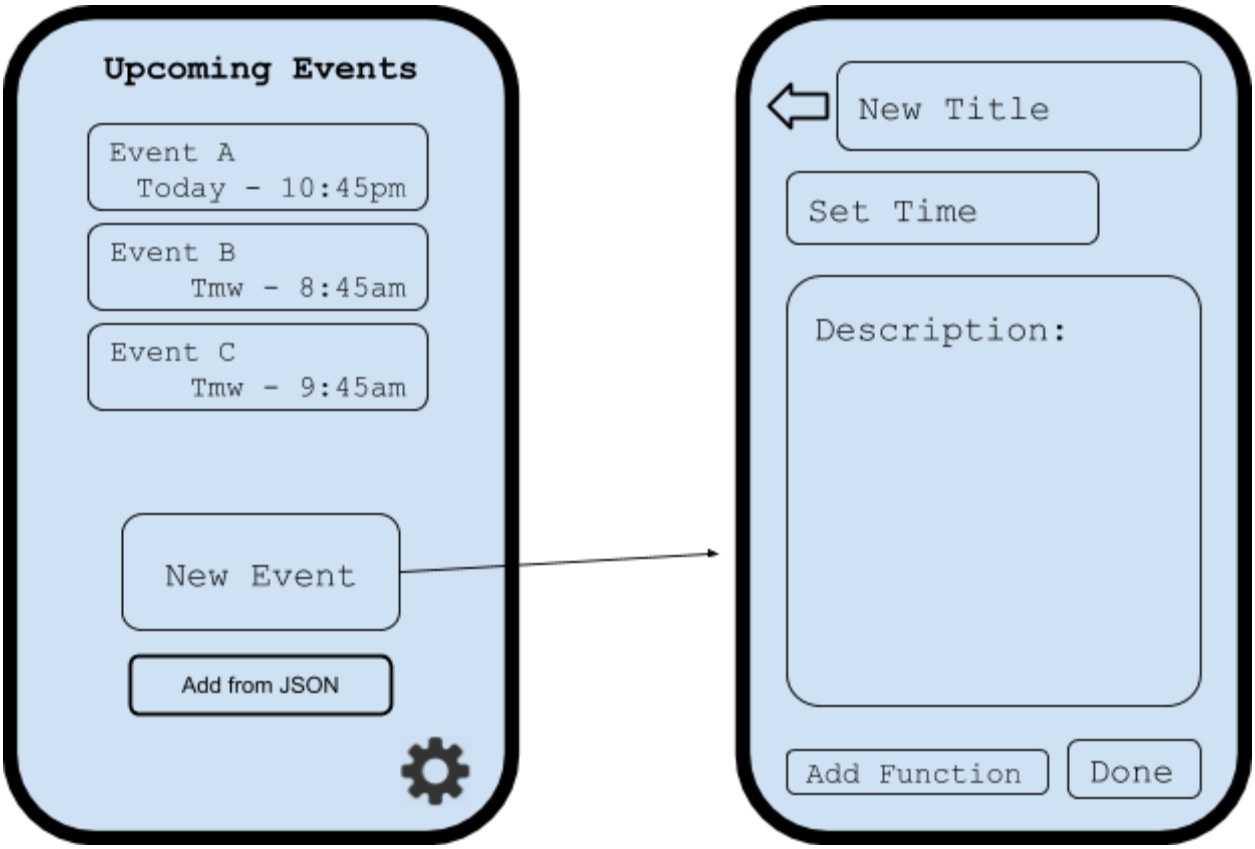
xTeam 158
Nick Jacobs, Jasmine Prianta, Yueyu Wang, Alex Grabowski

Always on Time

In the complex world that we live in, people are always busy with one thing or another, and problems can easily get forgotten in place of more pressing issues. In order to help people organize their lives better and ensure that everything that needs to get done is given time to do so, people need an organized time structure of when to do things.

This program would allow the user to set various tasks to automatically happen at a time that they select. Whether it be a reminder to do laundry when they get home or a script set to go off at a certain time that cleans junk from their computer, the user would be able to create an organized time scheme of when to get things done. This would be used by anybody, but in particular people with a lot of different tasks to do in their lives, such as students.

The frontend GUI of the program would primarily consist of the main screen, from which the user may select actions such as scheduling a new event, changing in-app settings, as well as a visual diagram showing a shortlist of upcoming events. From the main menu, the user would be able to enter a secondary menu, where they could create a new event. In this menu, there would be options for setting the time and date of the event, a title for the event, as well as space, to add anything else needed to complete the event (descriptive text, images, code to run, etc).



On the backend, there would be one main class that would serve as the main screen of the program. Inside of which there would be a field for a queue containing the events, sorted in order by the time until they execute. The main program would constantly check only the most recent event, removing it and sending a message to the user, as well as checking its search field (which for all intensive purposes serves as user input) to re-order the events according to the tags listed. The Event_Queue would always handle keeping events in order.

The classes would consist of the main class that runs the app, creates new events, stores the events in the queuw, and pushes events to the Event_Pusher when their time arrives. Event_Queue stores the events of type Event, and Event_Pusher activates a function as described in an input Event object

Input and Output of our Program:

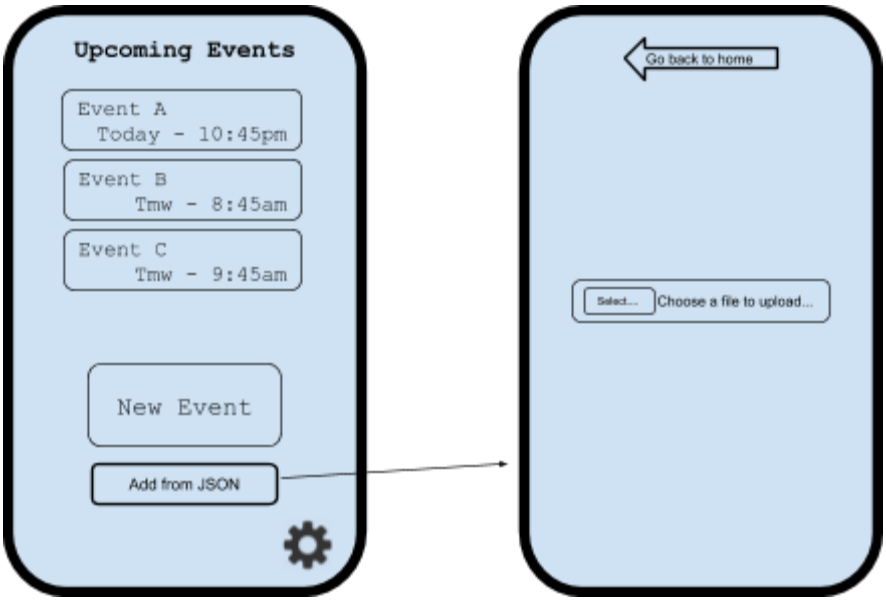
The input diagram is as follows:

According to the program, a title, time and description of your activity is needed to create an event. The data type of the “Title” and “Description” should be formatted as a string. For “Time”, it will provide a choice plate for you to choose both the date and time. After all the information is needed, the time for the event is displayed in this format: “Sep 30, 3:00 PM”. We then can utilize a priority queue to sort the time to find the event’s correct place. A button is also provided for you to go back to the home page if you change your mind.

Inputting from a File

choose “Add from JSON”. The format of the json file is as follows:

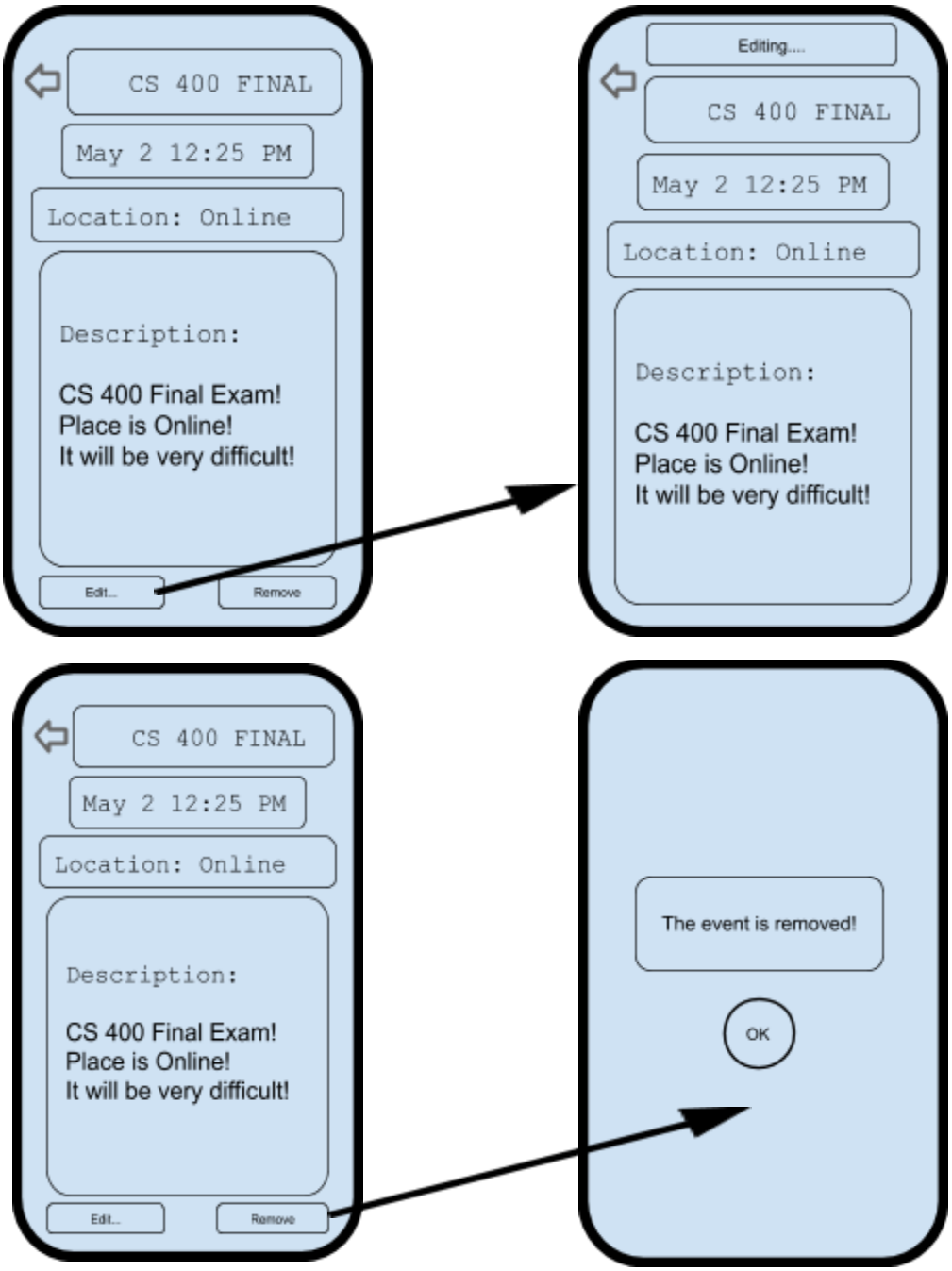
```
{
"Events": [
     {
        "Title": "CS 400 Exam",
        "Time:": "May 2 12:45 PM",
        "Location:": "Online",
        "Description:": "The final exam of CS400"
     },
    {
       "Title": "CS 300 Exam",
        "Time:": "May 5 6:45 PM",
        "Location:": "Sical Science Building",
        "Description:": "The final exam of CS300"

     },
     {
        "Title": "CS 354 Exam",
        "Time:": "May 9 7:45 PM",
        "Location:": "Van Vleck Hall",
        "Description:": "The final exam of CS354"
    },
    {
        "Title": "PHY 109 Exam",
        "Time:": "April 23 7:30 AM",
        "Location:": "Online",
        "Description:": "The final exam of PHY109"
    }
  ]
}
```



If you choose to add events from the json file, you can click the “Add from JSON” button. That way, you can add multiple events at once through the JSON file.

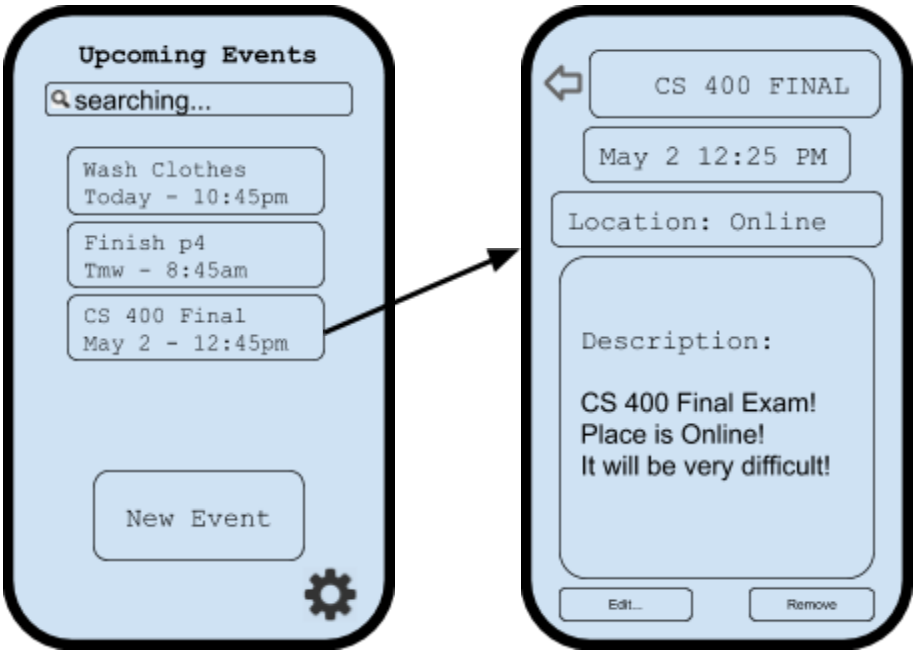
Editing or Removing an Existing Event

click the event frame and enter information into the event detail diagram. There are two buttons at the bottom of each event; Click the edit button so that you can edit an event, and click the remove button so that you can remove the event. It will jump to a new diagram that shows that the event is successfully deleted. When you click the OK button, it will take you back to the Home page.



Output Diagram

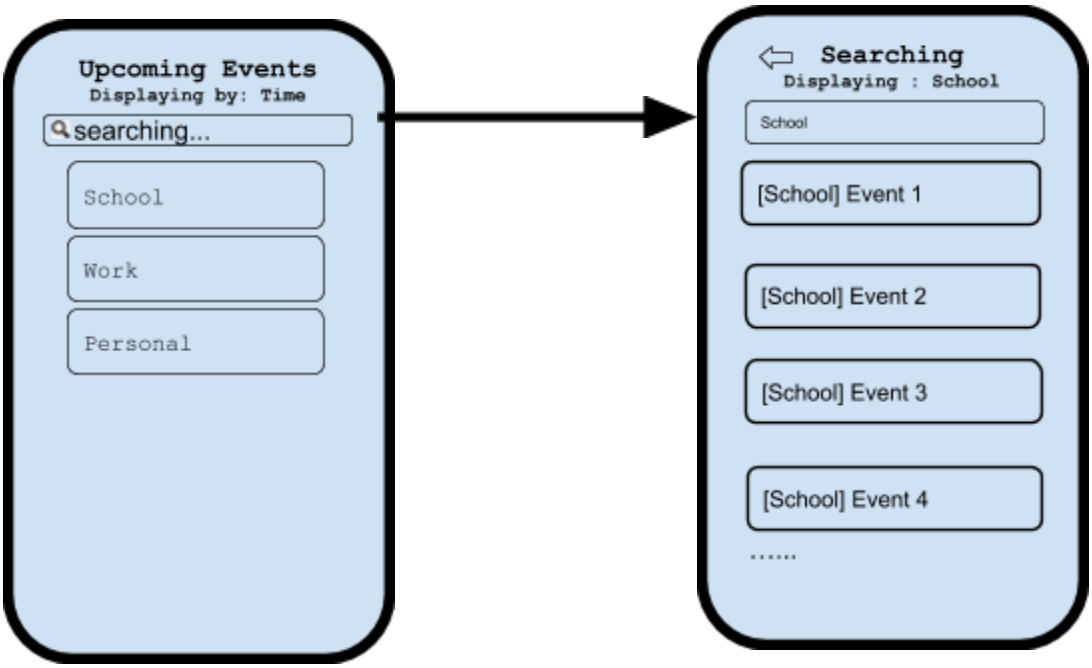
To check an event's fields, click the event frame, then the details of the event will show. There is also a back arrow, which takes you back to the main screen.



When you click each event's frame, it will jump to another screen that shows the details of the event. Also, when the time arrives, the specified event's detailed screen will automatically appear to notify you.

Searching

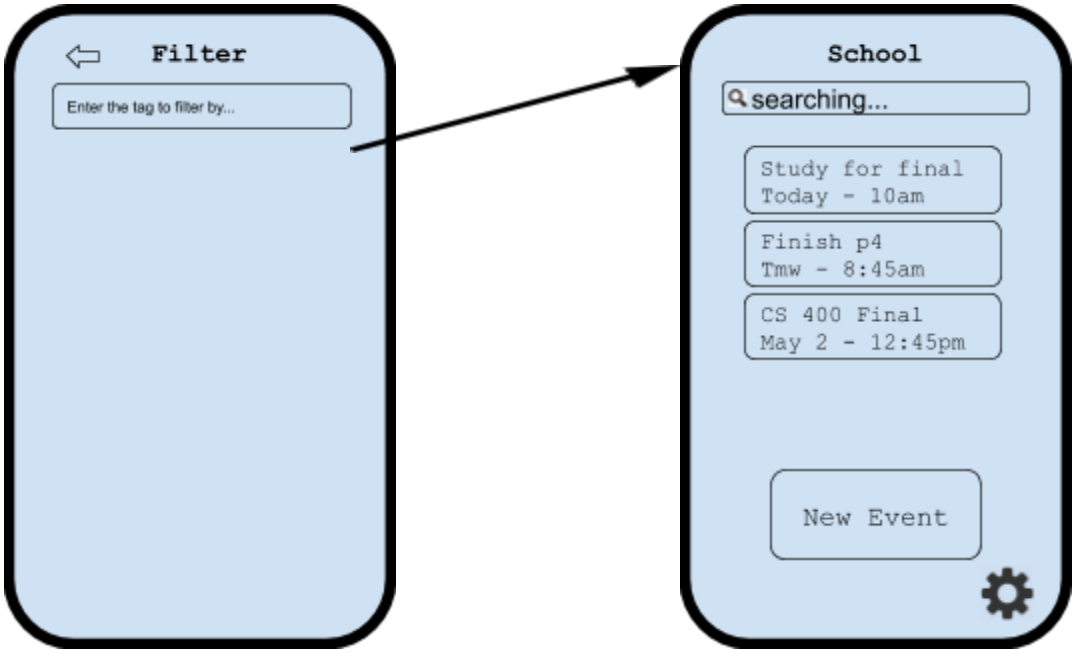
Tap the search bar, and it will jump to another screen that displays some searching options, which can help you search your event by different details of the event. The diagrams are shown below.



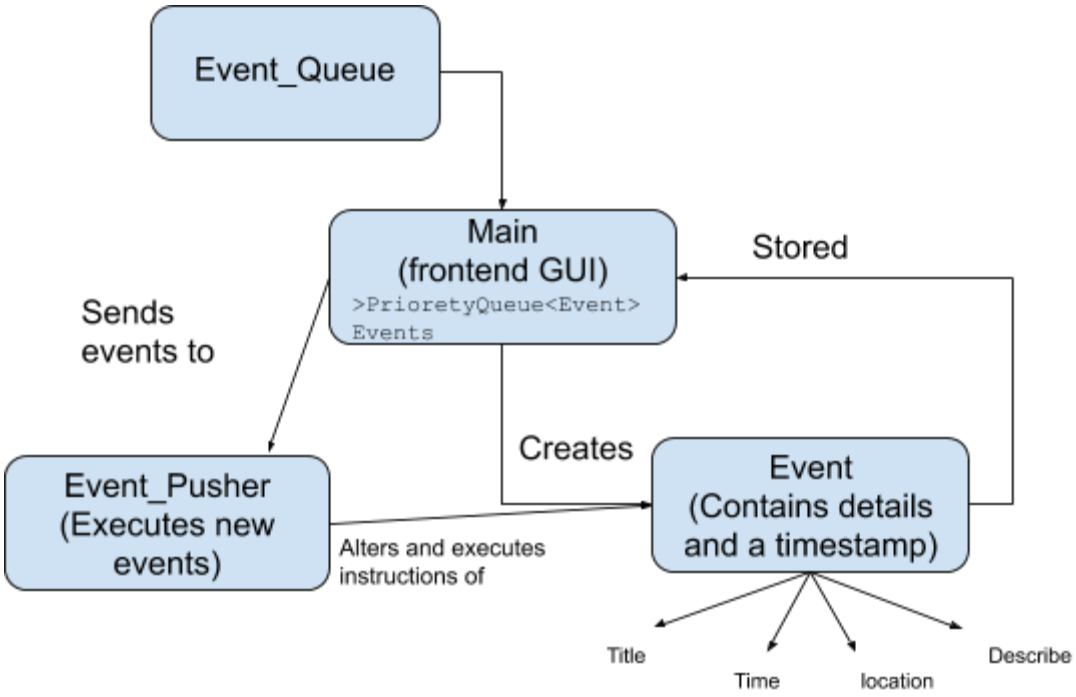
In the search bar, the user will have the ability to filter events by a specific tag, or to show specific events based on keywords that correspond to the the *location* and *title* fields of the Event class.

Filtering

The user is able to only see the events in a certain category. This allows them to easily see what events are more essential to them and quickly access the events they want to view/edit. When you go to settings and choose “Filter by”, it will pop up with a screen similar to the picture below. When you enter in the tag you want to filter by, it will go to the next screen, where it displays only the events within that category.



Milestones:



1. According to our proposal, the Event class can be the most independent one, which can be finished first and tested in advance. The event class also includes Get/Set methods
2. Then, the Events_Queue class, which stores each event can be created, creates a priority Queue. The class will be tested if it sorts these different events by time. Also, all searching methods including search for time, location, and title can be implemented in this queue class.
3. After that, the Event_pusher class will be created and tested if it shows the correct planned event at the correct time.
4. Finally, the main Class (frontend GUI) needed to be created to integrate all of the other classes and build diagrams for user input and output. Then the main class will be tested if it can access each function and execute each function correctly.

```
public class Event{

	String title = “”;
	List<String> tags;
	String description = “”;

	public Event( String title, List<String> tags, String description );
	public Event();

	void setTitle( String in );

	void addTag( String in );

	void removeTag( String in );

	void setTitle( String in );

	void updateGUI();

}
```

Assigned tasks:

Alex: Will create the Event class, add work on creating the methods for the communication between the Event_Queue and Main class.

Yueyu: Will create the Event_Queue class will be utilized by the front end GUI class to store and sort the list of events, then test if it is correctly stored and correctly sorted by time.

Nicholas: Will create the Event_pusher class to execute each event at the correct time, test it.

Jasmine: Will create the Main class and the diagrams for input and output, link every other class to the main class, ensure the main class can be run correctly.

Together: Check the correctness of the program and find other unknown bugs, fix them