# Design

## High-level Design

In the context of our FocusBot system's objective, an Event-Based Architecture stands out as the most suitable approach for the design. The main reason behind this choice is the flexibility and responsiveness this architecture pattern provides as it perfectly aligns with our system's core features. Our system is inherently event-driven, involving a sequence of user interactions: planning tasks, initiating work sessions, and receiving productivity recommendations based on work patterns. When users input task details – such as descriptions, deadlines, priorities, and difficulty estimates – the system captures these as events. Then, when users initiate work sessions, another event is dispatched to facilitate real-time monitoring inorder to detect productivity bottlenecks and provide timely aid recommendations if necessary by analyzing the user's productivity patterns. So, an event-driven design allows us to instantly capture and process core user interactions, while offering real-time insights and recommendations to enhance user productivity and task management.

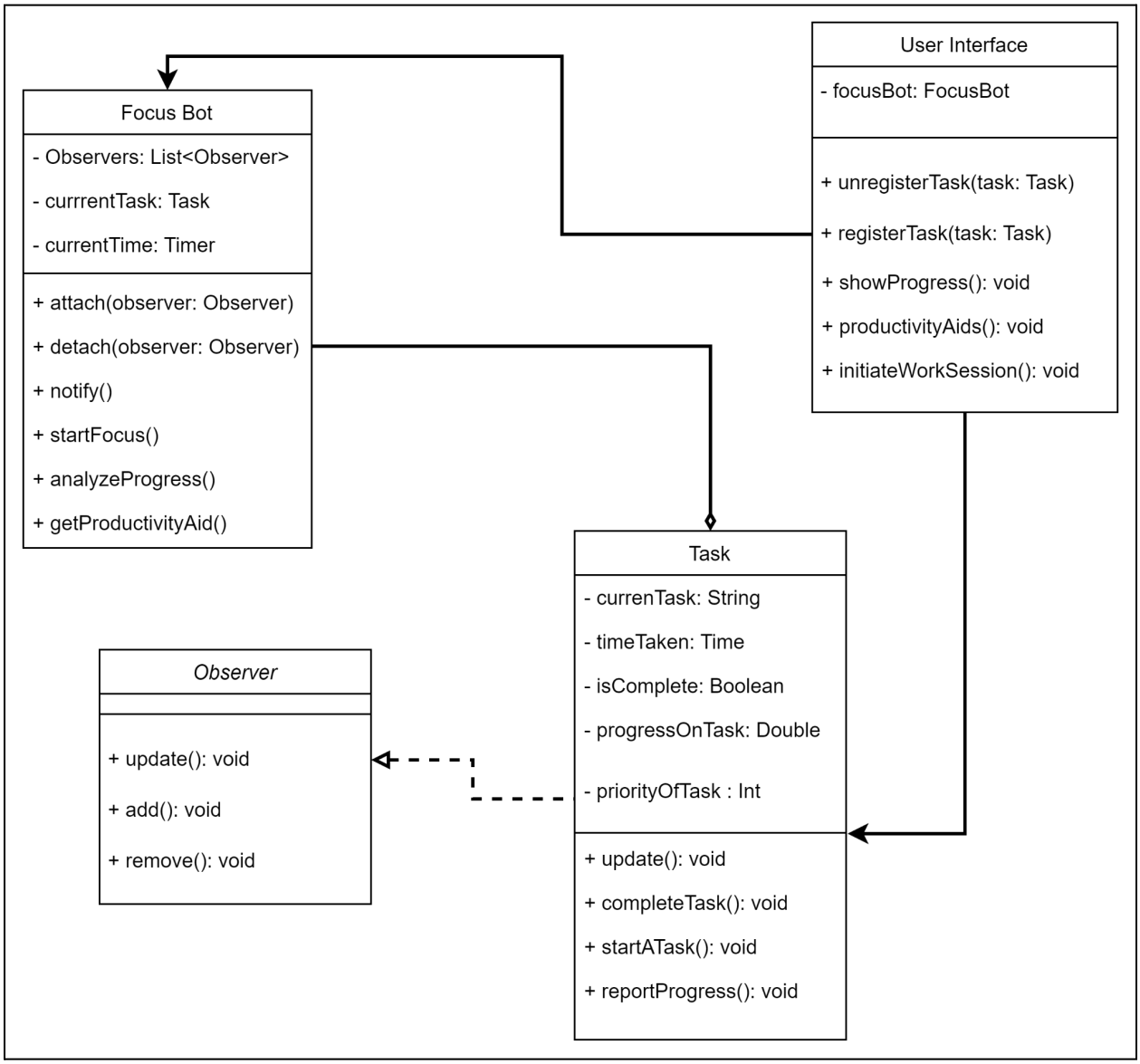
## Low-level Design

For the implementation of our FocusBot system, the design pattern family that would be most helpful is: The Behavioral Family. In particular, the Observer pattern within this family stands out as the most suited for our system's needs. It facilitates the subscription and receipt of updates by objects (i.e., the observers) from a subject when its state changes, while preserving a good level of decoupling between different classes. In the context of our project, the system itself acts as the subject, and the planned tasks as the observers. When users plan their tasks, the FocusBot system does not need to know the specifics of each individual task; its responsibility is to alert the observers when specific events occur, such as the start of a working session or the need for productivity-aids. Hence, as users start working, the system solely monitors and analyzes the time they're dedicating to each task and the progress they're making. Then, when necessary, it steps in with productivity boosting suggestions to enhance the work session. So, the Observer pattern from the Behavioral design pattern family aligns perfectly with the FocusBot system's features by providing seamless real-time notifications and interactions between the system and the planned tasks – to help the user improve their performance.

# Informal Class Diagram

Below is an informal representation of our system in the form of a class diagram. It embodies the Observer design pattern, with the:

- FocusBot class managing and notifying Task observers about state changes.
- Task class maintaining individual task states and responding to notifications with tailored updates.
- UserInterface class providing users the ability to interact with the system through various operations (e.g., adding/removing tasks, initiating work sessions…).



## Pseudocode Representation

Below are the pseudocode representations of our system's main tasks:

- Performing the necessary attachments, notifications, and detachments from the observers (i.e., tasks)

```
public void notify()
{
    for observer in observers
    {
        observer.update(task, timeOnTask);
    }


}

public void attach(Observer observer)
{
    observers.add(observer);

}

public void detatch(Observer observer)
{
    observers.remove(observer);

}
```

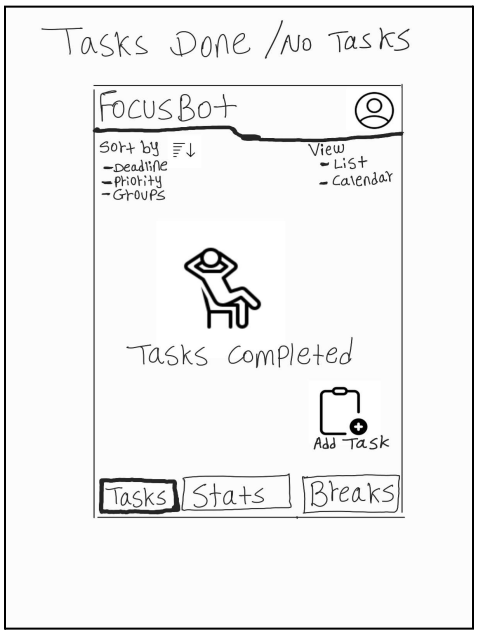
- Performing the necessary updates to changes in the observers (i.e., tasks)

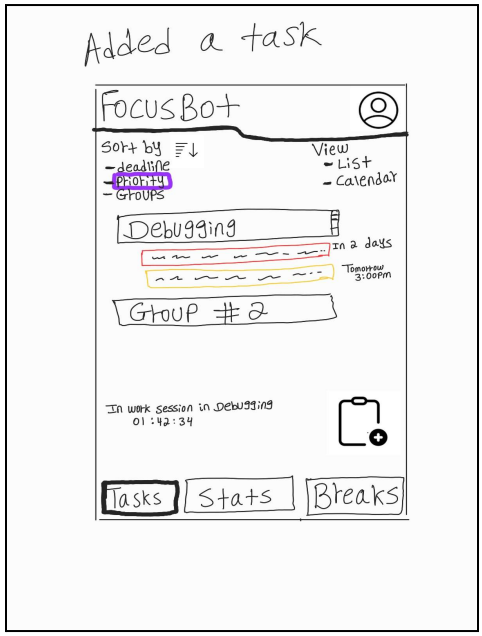
```
public void update(task, timeOnTask)
{
    if (task == this.task)
    {
        if (timeOnTask > thresholdTotalTime)
        {
            //reccomned the user changes the task
        }
        else if (timeOnTask - this.timeOfBreak > thresholdBreakTime)
        {
            //tell user to take break or offer somekind of productivity advice
        }
    }
    else
    {
        if (this.priority - task.priority > 0 &&  priorityTime/(this.priority - task.priority)  > timeOnTask)
        {
            //inform the user that there is a more important they should consider handling
        }
    }
}
```

# Design Sketches

The following sketches provide an overview of our project's primary functionality. When designing the interface, we wanted to maintain a clean and uncluttered appearance to ensure a

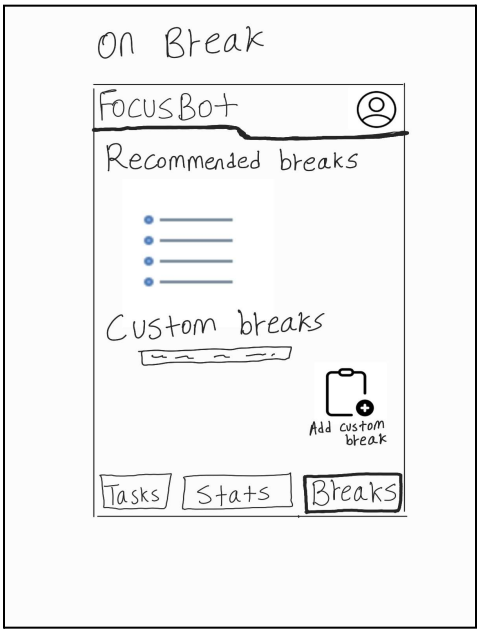
user-friendly experience. To achieve this, we limited the interface to three tabs, each representing a core feature of our project.



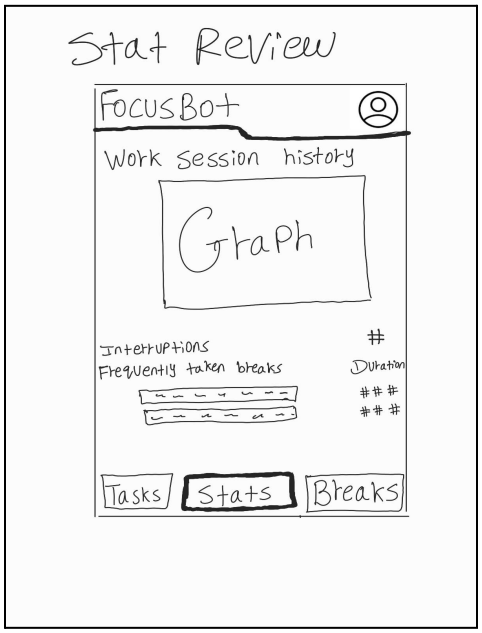

The initial tab within our system is ‘Tasks’. This section allows users to add, edit or delete tasks and view them either in a list or calendar format. Users can also sort their tasks by deadline, priority or groups (i.e., testing, debugging…). Upon clicking the ‘Add Tasks’ button in the first sketch, users are prompted to input the task details and then dynamically populate the screen with the added task. The second sketch shows the tab when it’s populated with tasks and the user is in an ongoing working session. Here, the working session is logged at the bottom, featuring the current task’s name and duration, while the center of the screen displays ALL the tasks to work on during the session, arranged by priority and deadline.

The next tab within our system is ‘Break’. When users decide to take a break or receive a system suggestion to do so, they are directed to the screen presented in the sketch to the right. This screen features a curated list of recommended activities for the break and the option to add custom break activities. After completing one or more break activities, users can transition back to their work session in the ‘Tasks’ tab.




The last tab within our system is ‘Stats’. It provides users with an overview of their work patterns. It includes a graph illustrating their overall productivity trends based on various factors, such as the duration for

each task and work session, as well as the number of interruptions (i.e., breaks taken). This feature, showcased in the following sketch, allows users to gain a deeper understanding of their work habits.




# Process Deliverable – Scrum Meeting Notes

- Andrew:
    - I have made good progress on our project's UML diagram so that Brandon can start working on the pseudocode representation.
    - But I still need to incorporate the necessary class relationships into the diagram.
    - And currently, I'm not facing any blockers – except for managing other homework assignments.
- Brandon:
    - I have started to work on the pseudocode representation for our project.
    - However, more work is still needed to complete the pseudocode representation.
    - And currently, I'm waiting on the completion of the diagram before finalizing the pseudocode representation.
- Daniel
    - I have designed a couple of sketches that show our project's primary functions.
    - Moving forward I need to finish the write-up that goes with the sketches.
    - And I'm not facing any blockers.
- Mali
    - I have completed the write-up for the high-level design section of our project.
    - Moving forward I need to complete the write-up for the low-level design section.

- And I’m not facing any blockers.