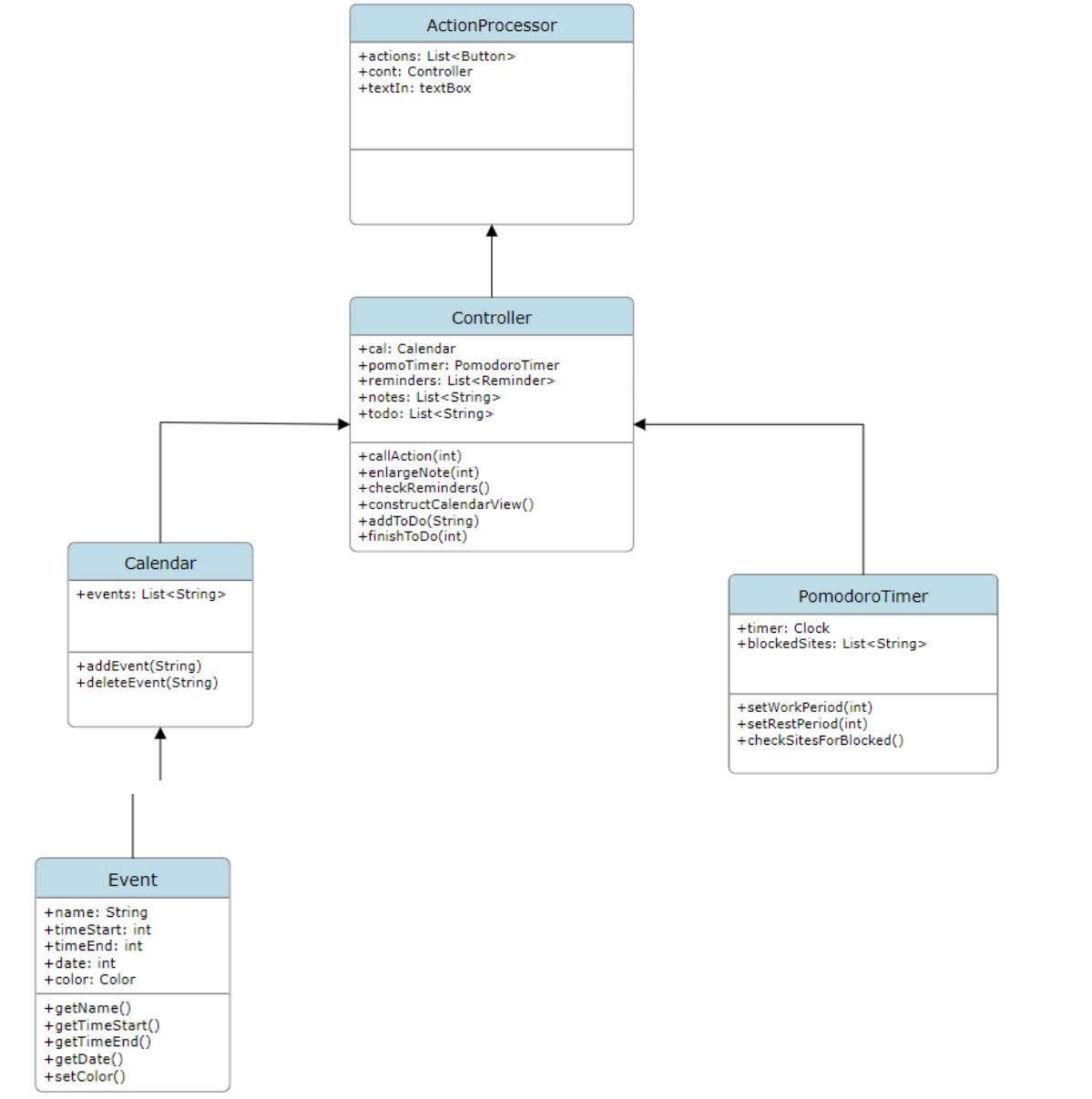
Aaryaman Akhouri, Taylor Mannion, Uday Talasila, Thomas Powderly

Project Milestone 3

Process Deliverable II

Our team has elected to use the Incremental model for this project. To match this process, below is a UML class diagram for our program, as well as documentation explaining what functions of our UML map to our prioritized requirements.



Our prioritized requirements are that all of the productivity tools such as Pomodoro timers, task management, calendars, and reminders are encapsulated within our application. Our designs outlined above and below do a good job of providing all of this functionality while still retaining great usability. The Pomodoro timer will be handled by a separate class with dedicated parameters for a timer and a blocked website list and functions for setting the work length, the rest length, and for checking whether the user is trying to access any blocked websites during their work period. The task management functionality will be handled by a list of strings within the Controller that will be displayed within the program, with functions for adding and removing items from the to-do list. The calendar will be handled via a separate class with a list of Event objects with names, dates, and start and end times as well as a color to display with on the Calendar. Additionally, the Controller class will have a function for constructing the view of the calendar based on the list of events within the Calendar function. The reminders functionality will be handled via a simple list of strings that will act as events on the calendar that don't display when constructing the calendar view in the main program.

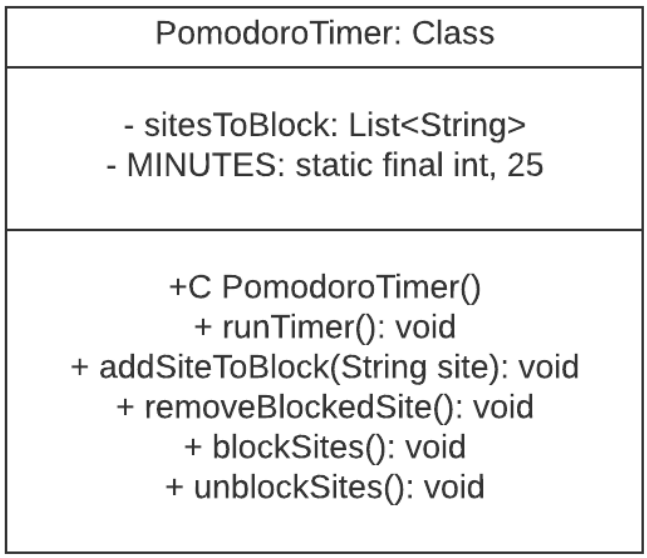
As we move forward we will continue to increment through the design processes to ensure that our end product looks and feels good to the user while still abiding to our prioritized requirements. Our project is still in early build stages, but as we progress we will most likely find better ways of doing things or discovering features that would be very useful as a part of this application which is why we choose incremental. This design model allows us to go back through the design process multiple times until we are happy with the end product.

High-Level Design

We would use the Model-View-Controller architecture to structure the system of FocusExtension. This architectural pattern, which is used in most modern web applications, would allow users a UI to interact with. In this UI, they could see their tasks on their to-do list, reminders, events on their calendar, Pomodoro timer, and notes. Since this application would basically involve doing the same things over and over again, such as setting up tasks and events, and starting Pomodoro timers, we need an architectural pattern that supports reuse and enhancement, which MVC does. Furthermore, as the user interacts with the controller, the model gets updated, with tasks and events being added, deadlines being set, and pomodoro timers starting. The model would then update the view for the user when a task or event is approaching, a deadline has passed, or certain websites need to be blocked during a Pomodoro session. This makes the MVC architecture seem intuitive for our application.

Low-Level Design

One sub-task of this project that needs to be implemented is a Pomodoro timer that blocks certain websites when the user is focusing on work. For implementing this sub-task, the behavioral design pattern family is suitable. This is because the interaction of the objects in this case, the Pomodoro timer and the websites to block, are important. When the Pomodoro timer is running, it is assigned the responsibility to ensure that the websites are blocked for at least 25 minutes. After the Pomodoro timer is done running, the blocked websites need to be unblocked. In other words, the state has changed, so the behavior of the Pomodoro timer must be altered. This makes the behavior design pattern family an appropriate choice for this sub-task. An informal class diagram with pseudocode written below further demonstrates this.



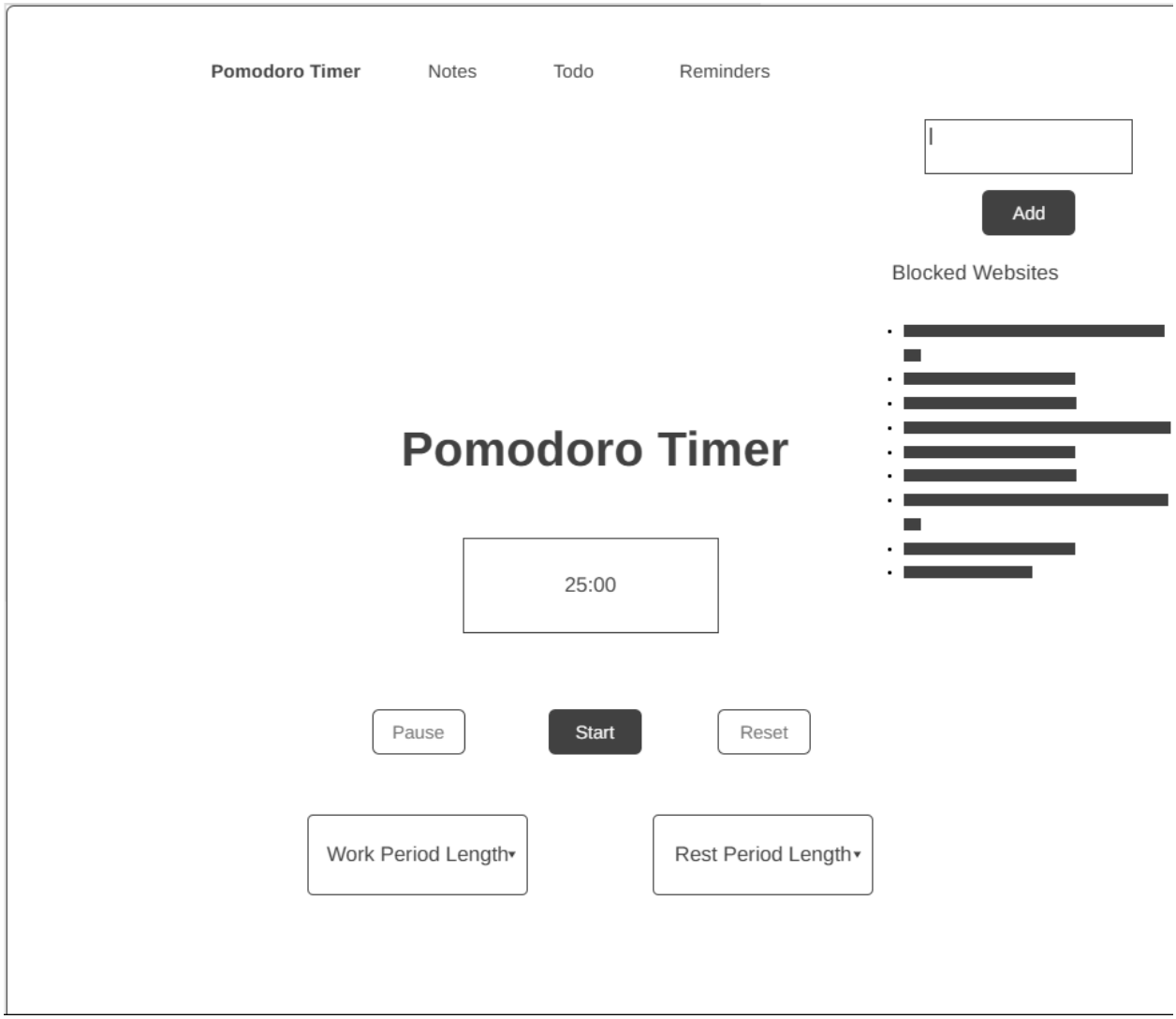

```
runTimer()
{
 int minutesPassed = 0;
 blockSites();
   while (minutesPassed < MINUTES)
   {
     // Count minutes here
     minutesPassed++;
   }

   unblockSites();
}
```

In this pseudocode and class diagram for running the Pomodoro timer, the number of minutes passed starts at 0. The sites are blocked with the `blockSites()` function. While the number of minutes passed is less than the final integer of minutes, 25, the minutes would be counted

(would need to find out a way to do this in actual code). With each minute passed, the number of minutes passed is incremented with `minutesPassed++`. After the timer has finished, the sites are unblocked with the `unblockSites()` function.

Design Sketch



The two screens were designed with the idea of keeping things as simple as possible, as well as attempting to meet usability heuristics. One example of this is that the name of the screen that is being used currently in the navbar is bolded. This meets the first heuristic which says that the design should keep the user informed. It also provides an easy way to switch between the different screens, namely just clicking what is in the navbar which meets the third principle of user control and freedom. It meets principle 8 by keeping the amount of information and elements in the screen to a minimum. Finally, it meets principle 4 by using common standards such as the + button to create a note and using a navbar of sorts to navigate to different areas of the application.