Team name: Turtle and Rabbit Inc.
Names and PIDs: Chung Chan (cchan02), Angel Perez (angelpg), Pranesh Ambokar (pambokar), John Lynch(jwlynch4)

High-level Design (4%)
Describe which architectural pattern you would use to structure the system. Justify your answer.

We would use a three-layer architecture to ensure the protection of data and an easily understandable structured hierarchy. We would have a user interface layer for interacting with users, a business logic layer for handling tasks, priorities, and scheduling, and a data storage layer for storing task data. Using this layered pattern, we could separate these concerns, making the application more maintainable and modular.

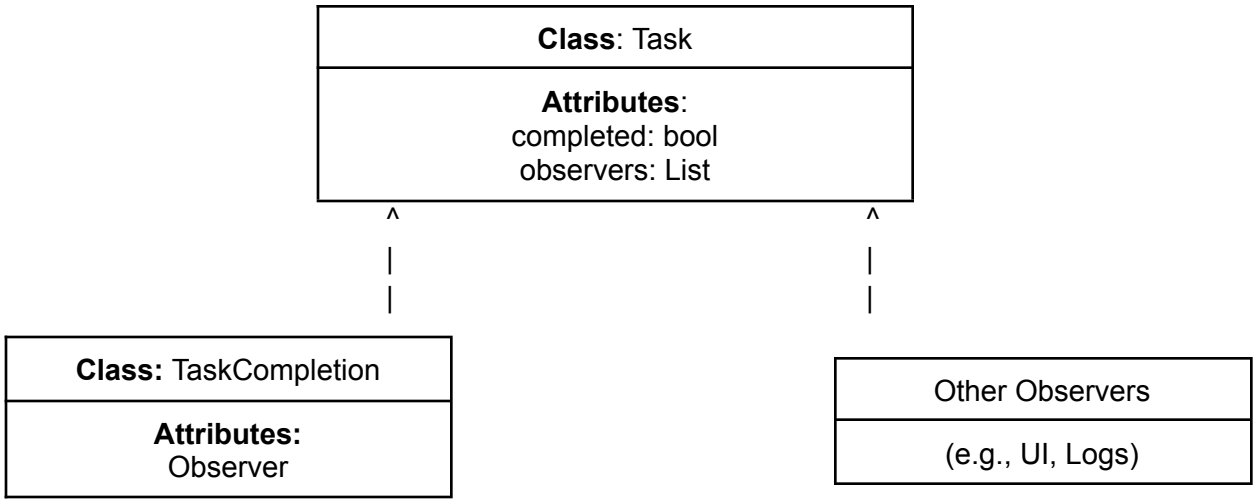
Low-level Design (4%)
Discuss which design pattern family might be helpful for implementing this project. Justify your answer, providing a code or pseudocode representation and an informal class diagram.

The TODO application involves various interactions and responsibilities between classes or objects. It deals with user interactions, task management, and the distribution of responsibilities between different components of the system, which is particularly suited to the Behavioral Design Pattern Family. Because of the need to handle user interactions (e.g., adding tasks, editing tasks, marking tasks as completed) and managing the behavior of tasks and their priorities, behavioral design patterns are well-suited for modeling and managing these interactions and responsibilities.

One example of a pattern within the Behavioral Family that could potentially be implemented within this project would be the Observer class, which would provide a way to notify a number of classes about changes. This is showcased through the following pseudocode snippet:

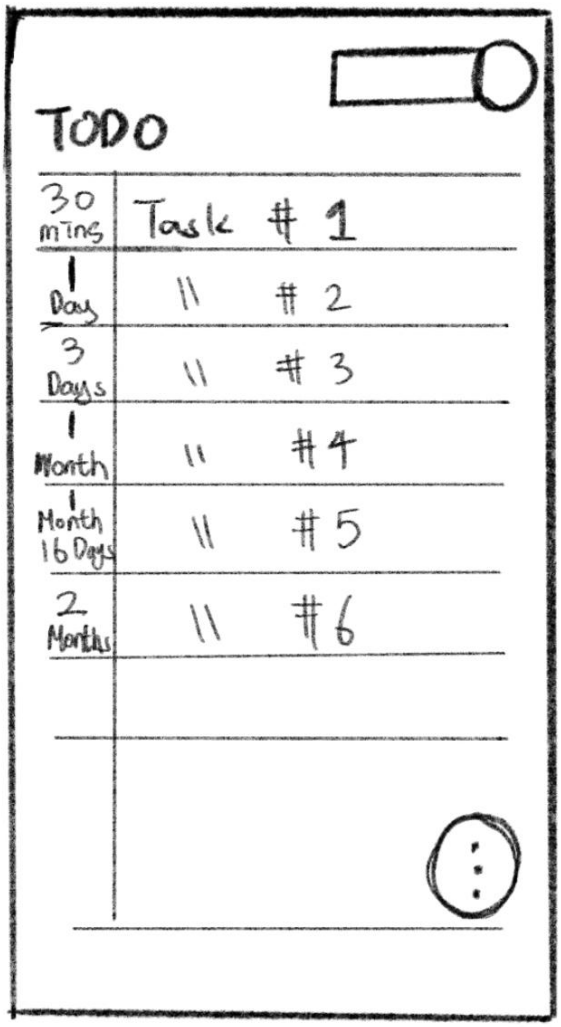
test

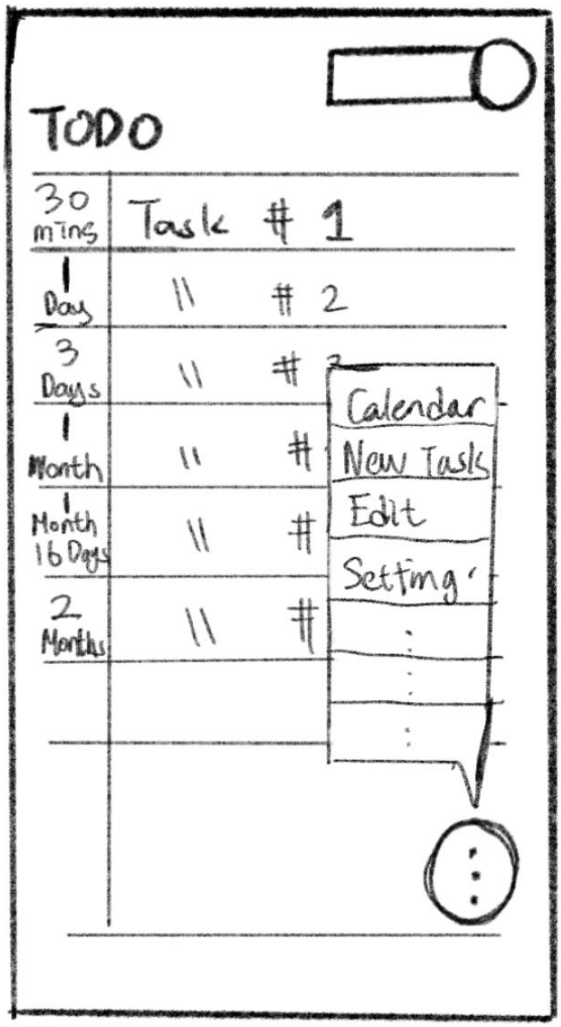
```
interface Observer {
    method update(task: Task)
}

# Concrete Observer
class TaskCompletionObserver implements Observer {
    method update(task: Task) {
        if task.isCompleted() {
            // Notify the user or perform some action
            displayTaskCompletionNotification(task)
        }
    }
}

class Task {
    private boolean completed
    private List<Observer> observers = new ArrayList()

    method addObserver(observer: Observer) {
        observers.add(observer)
    }

    method markAsCompleted() {
        this.completed = true
        notifyObservers()
    }

    method notifyObservers() {
        for each observer in observers {
            observer.update(this)
        }
    }
}

# Client code
task = new Task()
task.addObserver(new TaskCompletionObserver())

# When the task is marked as completed, the observer will be notified
task.markAsCompleted()
```



In this pseudocode representation and informal class diagram, the Observer pattern is used to notify various components (observers) when a task is marked as completed. This demonstrates how a Behavioral design pattern can help manage interactions and responsibilities within the TODO application.

Design Sketch (4%)
Creating a wireframe mockup of our project user interface.





No. 1 **No. 2**

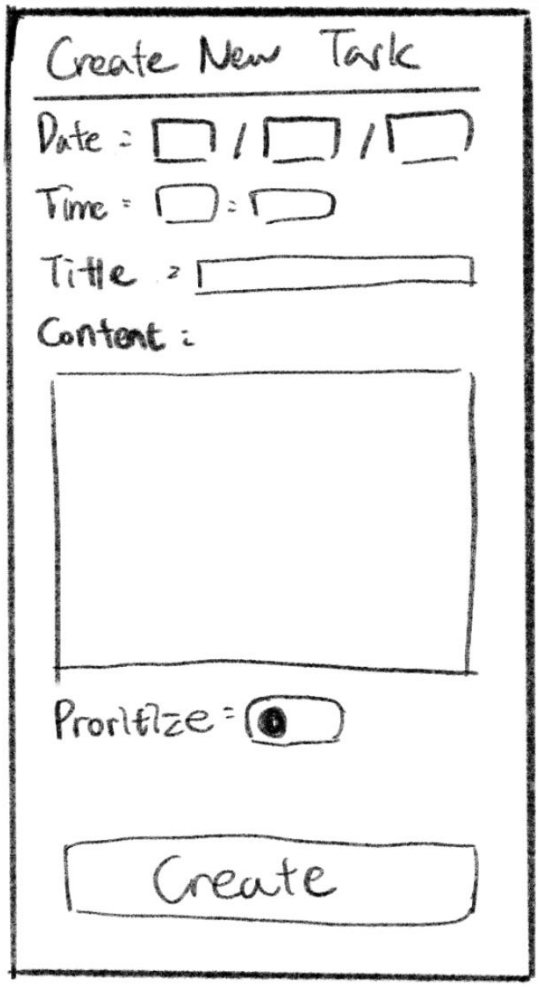
App section 1&2:

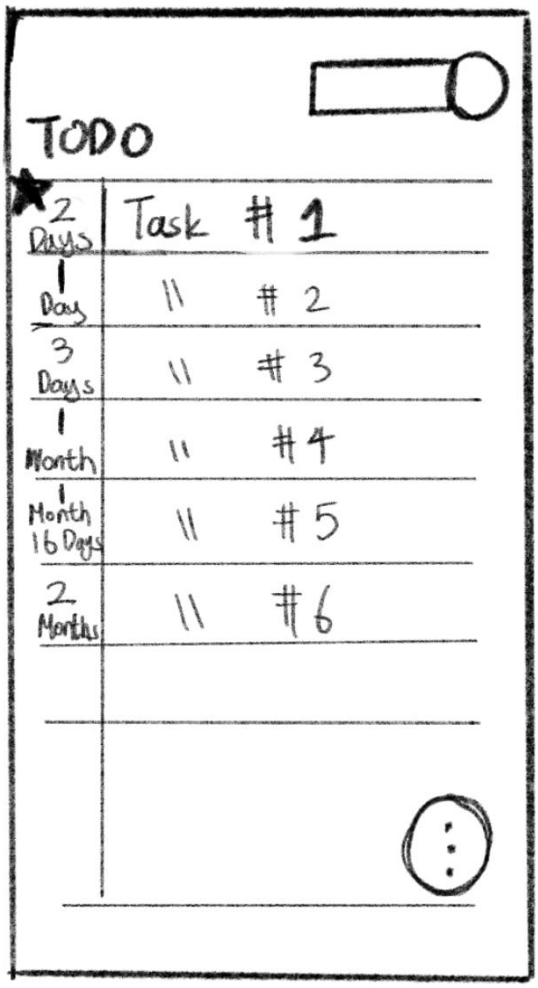
No.1:

This is the home page of our TODO application. It shows the most recent tasks by date. (If the task has a prioritize tag, it will show up at the top with other multiple prioritize tasks). The first column shows the date, the second column shows the task title, and the user can click in and check the details. The upper right corner is user information, and the bottom right corner is the menu button. We keep our design simple and to-the-point, because we believe a TODO application should focus more on reminding the user what's left to be done.

No.2:

This shows our menu function. "Calendar" shows the entire todo list monthly, "New Task" allows the user to create new tasks, "Edit" allows the user to edit existing tasks, and "Settings" allows the user to modify the app, for example, font size, color-blind, application language, etc.





No. 3 **No. 4**

App section 3&4:

No.3:

This is the page for creating a new task. Users can input the format by YYYY/MM/DD, and input time by HH: MM (Hours: Minutes). Users can also give the task a title, and type in the details in the “Content” text box. To make it a prioritize task, the user just needs to turn on the prioritize button. Finally, the “Create” button will tell the server to create a task and add it to the user’s TODO list.

No.4:

This page shows what the app will look like if there’s a prioritized task on the TODO list. It has a similar function as the no.1 wireframe. All columns, user information, and menu buttons remain the same. But in this wireframe, task #1 changed to a prioritize task. The app will order the tasks by this algorithm: Priortize tag > Priortize tag task’s date > date.

Project Check-In

Complete this survey to provide an update on your team progress on the project for this semester. Only one team member needs to complete this for the group.

Process Deliverable (3%)
Our prototype system.

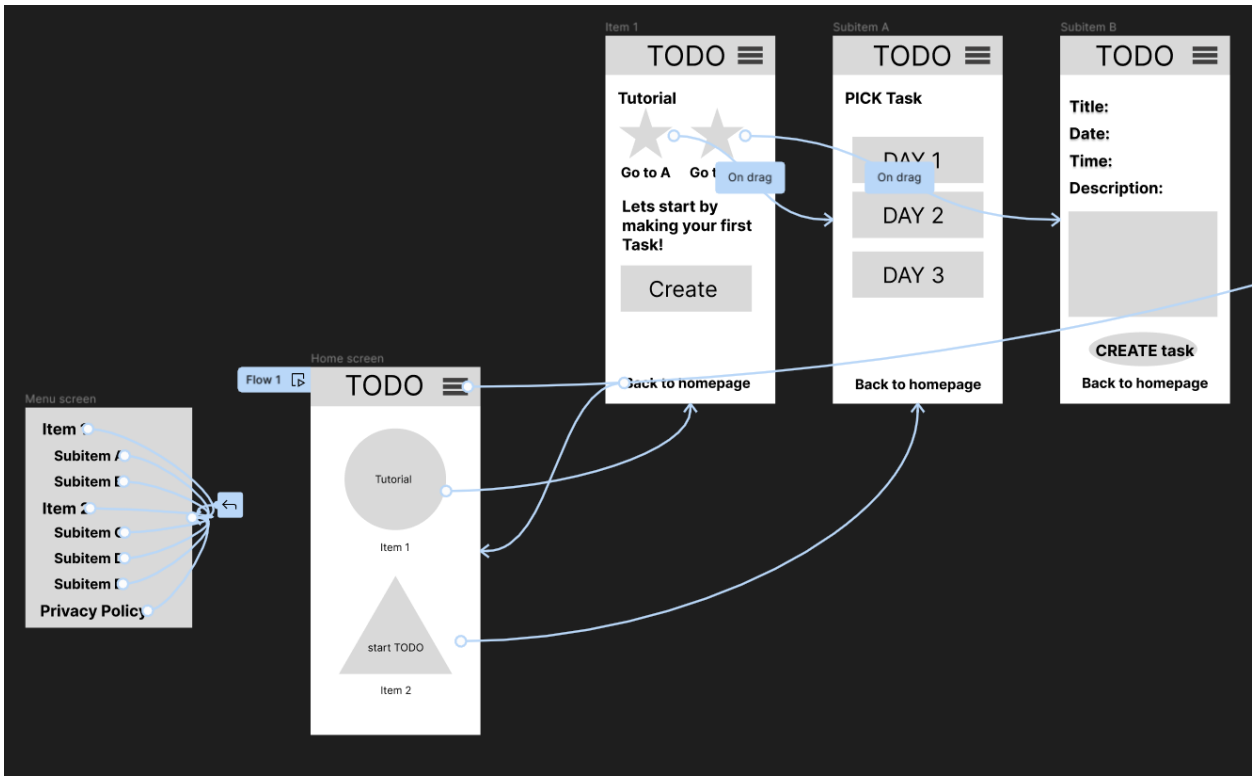


Figure 1

Our prototype shows the steps and UI interface of our design. TODO applications will have capabilities to be seen in mobile apps and websites. The customer will have the ability to see our prototype, in this case, a mobile version system is seen in **Figure 1**. This means if any mistakes are made during our prototype, we as designers can revise and fix the problems. Our TODO application will lead users to tutorials and start making tasks. Compared to our wireframe, here we want to know the actual steps taken by the user. We plan to revise when user feedback is given. This system is short of what our application will do.