# **What everyone worked on:**

## **Bora:**

For Phase 0 of our project, I worked on the Window interface, some of the Window classes and their implementations. First of all, me and one of my group members implemented the Window interface with its methods according to CRC cards. Then, I started to implement some of the skeleton code. The ones that I worked on were LoginWindow, CreateAccountWindow and TimetableWindow. For LoginWindow, I overrode the getUserInput method in Window interface by returning a list of usernames and passwords for every user that logs in. For CreateAccountWindow, I overrode the getUserInput method in Window interface by returning a list of names, usernames and passwords for every user that creates an account. For the TimetableWindow class, I overrode the getUserInput method in the Window interface for going back to View Account. If the user wants to go back to View Account, the user must type VA and it will return a list with index 1. Also I added a display method which displays (prints) a visual representation of the schedule.(by using Schedule.toString()). I plan to work on the same classes and maybe work on some different Window classes as well for Phase 1.

## **Benjamin:**

I worked on ManagementSystem, AddMedicineWindow and MedicineManager. In addition to that, I partially implemented and debugged the interface adapters and framework and drivers layers classes. I wrote unit tests for the management system. I also helped write the CRC cards while providing input on what the Entities should and whether we needed interfaces. In the future I plan on helping wherever necessary, in addition to working on some of the CRC cards we left out, such as UserDatabase and MedicineDatabase. I also plan on making sure that the times are actually used and can be used to set reminders. I also plan on adding additional tests for the functions as we left a lot out due to time constraints.

## **Mouaid:**

For phase 0 of our project, I implemented the User Class as well as the Medicine class in the business rules layer. Moreover, I played a big role in coming up with and perfecting classes that we would need for phase 0 and the upcoming phases as well as implementing many CRC cards. Furthermore, I largely contributed to the implementation of AppManger. I also wrote documentation for most of the Classes as well as their individual methods in all of the Business rules layer, Application rules layer as well as the interface adaptors layer. I reviewed the structure of the application and identified issues with code implantation as well as redundant code. And finally, I wrote down unit tests for the classes and their respective methods accordingly. For the next phase I plan on expanding on some of the extra features that we have planned like an alert system as well as implement SleepSchedule and MealSchedule.

## **Sujoy:**

For phase 0 of the project, helped design and write out the specification of our project idea. I created a document for everyone to come and edit the CRC cards and contributed greatly to the development and design of the CRC model. I also played a major part in designing and writing out our scenario walkthrough and aided my group in creating the skeletal program. I implemented the classes MedicineSchedule, Schedule, Event, ScheduleManager, ScheduleCompiler and did a majority of the implementation of AppManager. I also made miscellaneous changes to other classes, such as adding some documentation here and there and changing data types to better fit the project. I also wrote out the summaries for the CRC cards and skeletal program. For the future, I plan on working on the prescription aspect of our project, such as implementing the interface and managing the user’s prescriptions.

## **Eren:**

For the Phase 0 of our Project I created the initial versions of the Window interface (which is now a class), AddMedicineWindow, StartScreenWindow, and ViewAccountWindow. This version of the program uses the console. For the upcoming phases, I plan on making a visual displayer for the Window classes which can handle user inputs.

## **Mohamed:**

For Phase 0 of our project, I implemented UserManager and MedicineManager classes in the application and business rules layer. Also, I largely contributed to implementing ManagementSystem. In addition to that, I partially implemented and debugged the interface adapters and framework and drivers layers classes. Finally, I wrote unit tests for the classes and their respective methods accordingly. Regarding the CRC cards, I’ve made several CRC cards in the application business rules and the business enterprise rules. I plan on expanding the program’s capabilities such as adding an alert system and visualizing the program using XML in android.

# **Brief Summaries**

## A brief summary of our specification:

Our program organizes the times a user takes his medications by creating a schedule for them. In our program, the user can create an account, add a new medication or prescription and the timings for it, and our program will generate a schedule for them based on what they enter. The user can also enter the specific times they want to take their medication and currently, the schedule works on a weekly basis.

## A brief summary of our CRC cards

Our CRC cards were made based on the specification. In order to make our code as modular as possible, and to follow clean architecture, we created a lot of classes and managers for these classes. We decided our entities (enterprise business rules) to be the medicine, prescriptions, schedules, and user. We made use case interactors for each of the entities, such as a user manager, schedule manager, and medicine manager. We decided to have an input/output boundary called ManagementSystem and a controller AppManager. AppManager also acts as the presenter for our program. We decided to have a bunch of windows for each page, but they all are children of a class called Window, which AppManager depends so, thus ensuring we don’t violate clean architecture. In essence, we have entities, controlled by manager classes, which are called upon by the management system, which is called upon by the app manager. App manager decides what is to be shown on the screen and manages user input.

## A brief summary of our Scenario-walkthrough:

In our scenario walkthrough, the user can sign up, add some medicine to their account, and see the medicine added. When the user runs the main file they should select S for sign up as login has not been implemented. They can then proceed to add their name, username, and password. They are then allowed to decide what to do, with add, view, and logout being the only options implemented. Selecting anything else would cause the program to stop. When the user selects add, they can then fill in the information as necessary. Selecting view allows the user to see what they added while logout simply ends the program.

## A brief summary of our skeletal program:

For our skeletal program, we tried to follow our scenario walkthrough as closely as possible. We didn’t get to implement all the responsibilities of every class we implemented since a lot of them were not necessary for the scenario walkthrough. We just tried to get the scenario walkthrough done for phase 0. We use the command line as our UI for now, and allow the user to sign up, add medicine, and view the created timetable. We take user input from the console. For now, please follow the scenario walkthrough while running the program, as we did not have time to implement the other features and didn’t get to completely verify proper user input. We also added a couple of unit tests to run, but we decided not to add unit tests for getter and setters, which is a majority of the code for some classes.

# **CRC Cards:**

This is included in the repository under the folder CRC cards. The folder contains screenshots of the CRC cards. Each image is of cards grouped together based on similarity, and the images are separated into different folders based on which layer of clean architecture they are a part of.

# **What worked well with our design so far:**

Initially, our design was meant to display the schedule that will be compiled onto a screen. However, for phase 0, our goal is to display the schedule onto the console instead. As it can be seen, this has gone well for us so far and we are able to display a clean schedule on the console without any error.

# We managed to create well-structured code that adhered to clean architecture and had distinct classes in all 4 layers.

I believe we did well in designing the manager classes and the boundary classes between the layers. Currently, we are able to add medicine and view the schedule, but the way the code is designed so far will make it easier for us to add the ability to edit and remove medicine, as well as work with prescriptions. The way the code is designed right now will make it easier for us to implement the features we have planned in the future.

# **Open Question to the TA:**

We are interested in utilizing databases and want to know how we could go about storing and retrieving information from online or local databases.

We are also struggling with interdependency and coupling. It feels a bit difficult to understand what low coupling indicates, and how do we go about decreasing coupling and increasing modularity without overcomplicating the code?

.