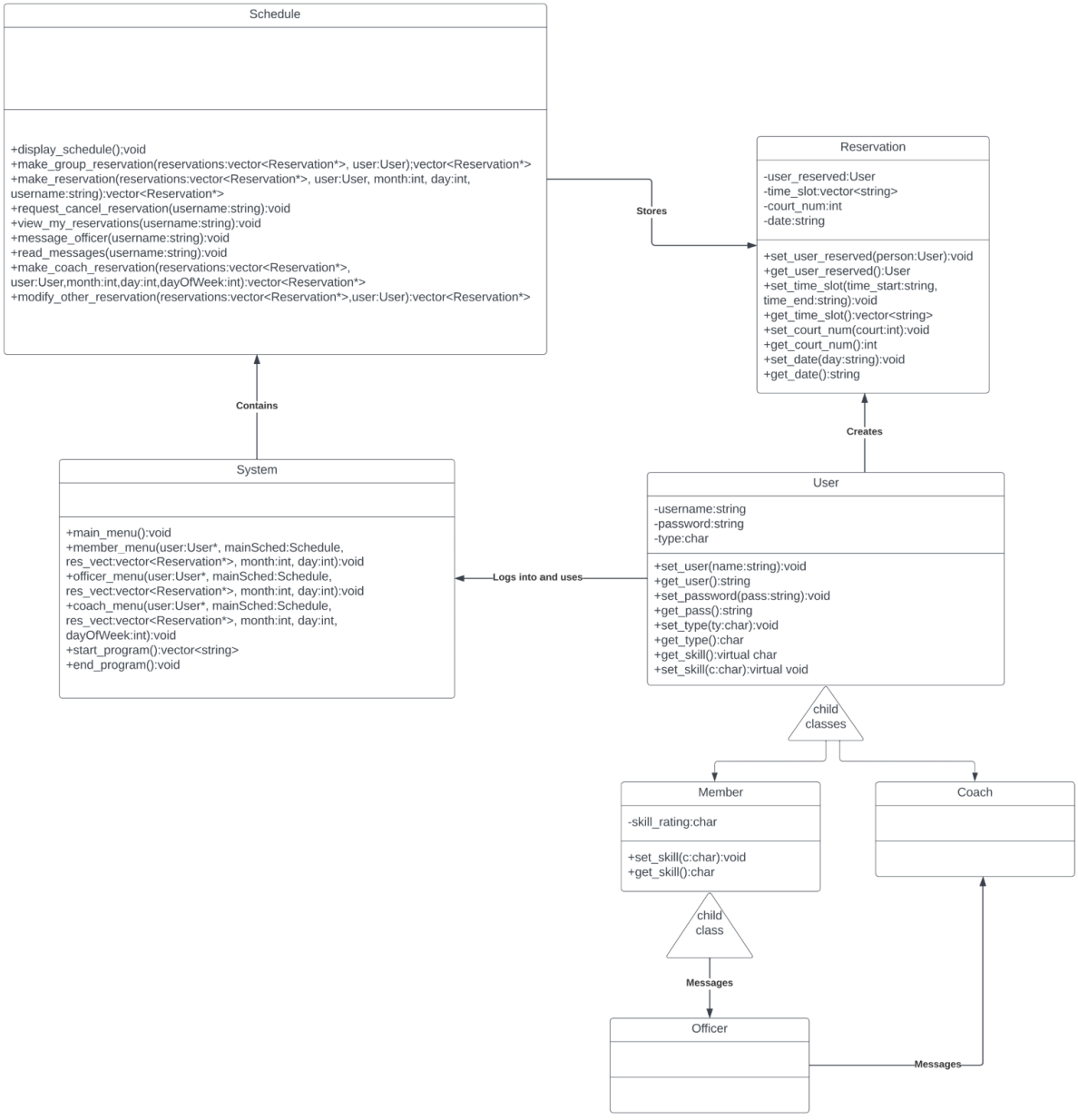
Final Project Report

Group Members:
Brady Hobson, Sean Merkle, Julian Savini

I. UML Diagram

Attached below is our UML diagram. This diagram has changed after writing the code, because we needed more functions than we had anticipated.



II. CRC Cards

After writing our program, our CRC cards have also needed minor changes. There are more responsibilities for each class than we had anticipated. The CRC cards are shown below:

Reservation	
Responsibilities	Contributors
-Store the user logged in -Store a list of time slots -Store the number of a court -Store the date and time -Get and set functions for variables described above	-User -Schedule

Schedule	
Responsibilities	Contributors
-Display schedule in console -Make group or standard reservation/modify reservation -Request cancellation of schedule/Message officer -View reservations -Read messages	-System -Reservation

System	
Responsibilities	Contributors
-Store the menu for a member -Store the menu for an officer -Store the menu for a coach -Start and end program	-User -Schedule

User	
Responsibilities	Contributors
- Stores username - Stores password - Stores type of user - Getter and Setter functions for username, password, type, and skill level	- Reservation -System -Member -Coach

Member	
Responsibilities	Contributors
- Need to get and set skill rating	-User -Officer

Officer	
Responsibilities	Contributors
- All functions stored within user class	-User

Coach	
Responsibilities	Contributors
- All Functions stored within user class	-Member -Coach

III. Discussion of Design

As anticipated, we knew that the responsibilities of a few classes were going to become more complex than our initial design. For instance, reservation does not merely add or remove a reservation. We needed to add functions that allow a user to message an officer, and officers need to modify other users' reservations.

One major change in our design process was instead of having a user have a create reservation function within their class, the function was placed in schedule, and then called for within our cpp file. This change was needed because we already had an add reservation function in our schedule file. We did not want to repeat ourselves, as encouraged by the rule DRY. We also looked towards the rule YAGNI when writing our coach class, because there was less functionality needed than we had anticipated.

Furthermore, whenever we modified our code, we tested our code by creating cases which would break our code. One example, was we tried to create a reservation for a time which has already passed, such as 5/25/2023. When we first tested this case, the create reservation function appended to our txt file a reservation. So, we utilized exception handling so that when a user typed in an old date, the program says that date is invalid, and that requests the user again to enter a valid reservation day. This process of writing code, finding an invalid case, testing the code with said case, and then rewriting our code, was frequently used throughout our process. Especially since there were three people working on the same project, it was important that each of us tested our code immediately after implementation, so no further bugs occurred.

We also had someone who had no coding experience test our code, to see if they found a way to find an error with our code. Although we did not explore too far in this aspect of testing, we found no errors. This is because we always tested our code after implementation, which allowed for a smooth design process.