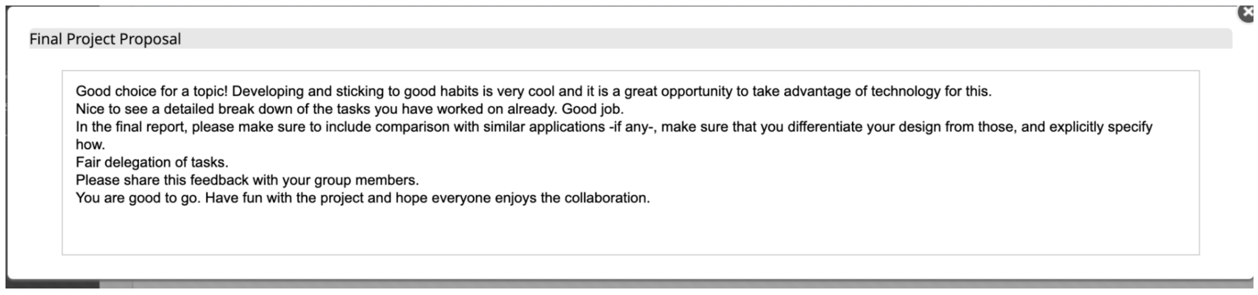
CS3354 Software Engineering
Final Project Deliverable 1

60 Days

Sharmin Gaziani, Abdurrehman Zulfiqar, Jaggan Jestine, Sam Williford,
John Bui, Peter Shen, Winifred Ojo.

1. **[5 POINTS] Please attach here the Final Project draft description (that contains the instructor's feedback)**.



Final Project Proposal

Good choice for a topic! Developing and sticking to good habits is very cool and it is a great opportunity to take advantage of technology for this.
Nice to see a detailed break down of the tasks you have worked on already. Good job.
In the final report, please make sure to include comparison with similar applications -if any-, make sure that you differentiate your design from those, and explicitly specify how.
Fair delegation of tasks.
Please share this feedback with your group members.
You are good to go. Have fun with the project and hope everyone enjoys the collaboration.

Above we have attached a copy of the feedback we received from the Final Project Proposal. The feedback we received was largely positive. The ideas and task delegation for our project passed the professor's approval and we look forward to implementing it in a way that emulates professional enterprise software engineering. We'll be sure to get familiar with any and all similar products in the market and seek to make our own project unique.

2. **[10 POINTS] Setting up a GitHub repository.**

   Group Repository URL: https://github.com/sharmingaz/3354-60-Days

3. **[5 POINTS] Delegation of tasks:**

   Sharmin: For the project I'm tasked with gathering all the materials from team members to submit them, creating the GitHub/adding all the members to it, and building the project schedule.

   John: I am tasked with getting information to list the software requirements for the project clearly. I will also be responsible for evaluating the personnel estimation.

   Jaggan: For the project, I am tasked with creating the UML class diagram, as well as creating a software test plan that may be used when the project is ready for testing.

   Sam: I will look at similar projects to compare designs and create a sequence class diagram.

   Abdurrehman: I'll be in charge of creating our first README - aside from that, my main roles will be coming up with some pricing estimation figures as well as working on architectural design for our project.

   Winifred: I am tasked with making a deliverable conclusion on the project as well as creating a use case diagram.

   Peter: I will be creating our project scope commit and doing research on the hardware/software estimation for our product.

4. **[5 POINTS] Which software process model is employed in the project and why.**

   The waterfall model is the software process model employed in this project. This is because of its linear sequence lifecycle model that adopts a simple structure of phases in which the result of each phase affects and falls into the next phase of development. The waterfall model helps keep everyone involved in the project up to speed on the latest development involving the project. This is because technical documentation is necessary for the initial phase of development. That way everyone knows what they are doing and their goals and objectives. By defining all requirements needed for a project, the costs can be estimated with a high level of accuracy preventing extra costs from being imposed later during development. The waterfall model makes testing easier and transparent as the test scenarios would have been explained in the functional specification of the requirement phase.

5. **[15 POINTS] Software Requirements including 5.a.) [5 POINTS] Functional requirements.**

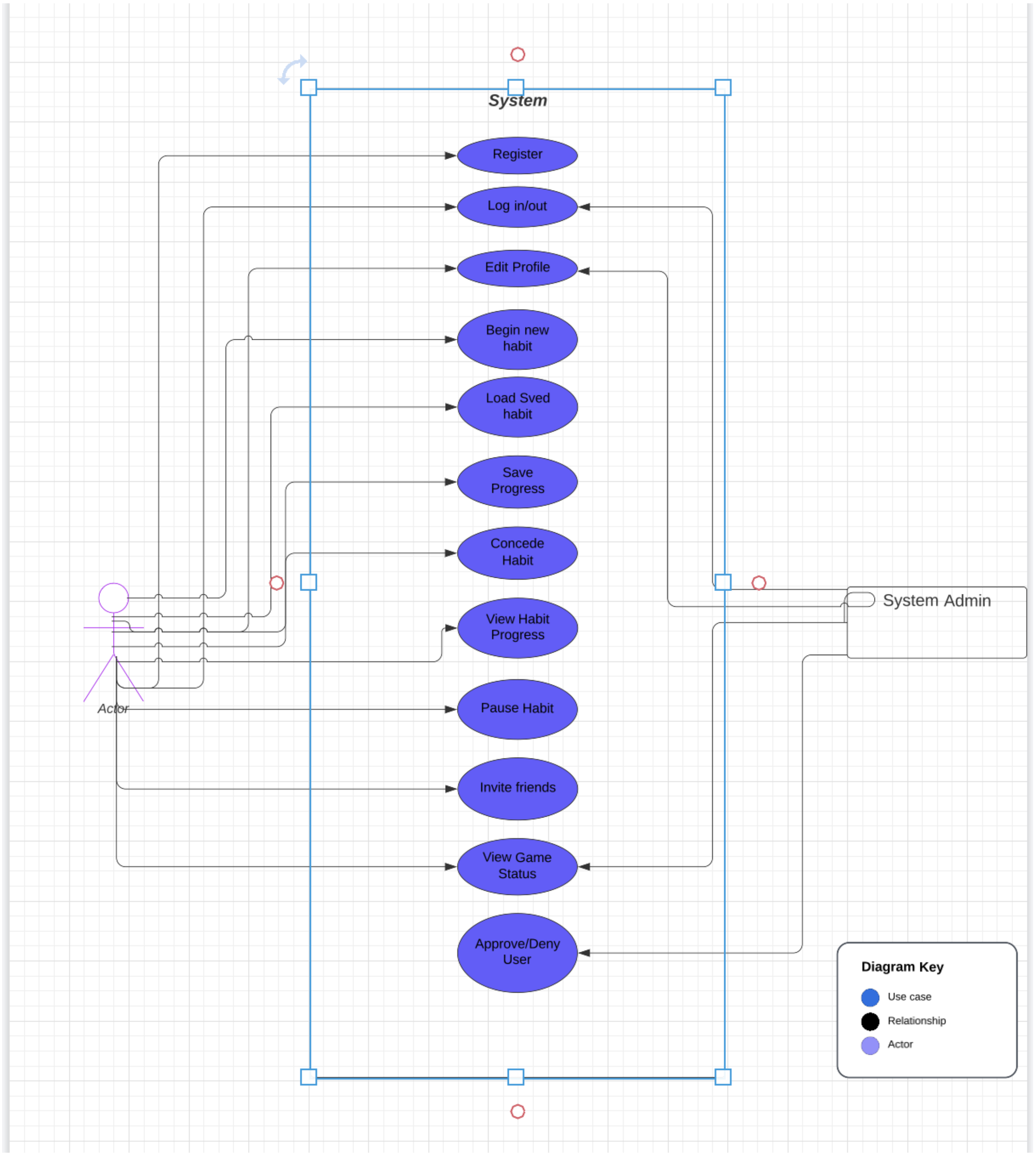
- The system must be able to authenticate the user's login credentials.
- The system must send a lock screen notification to remind the user at a specific set time every single day for 60 days of that certain habit.
- The system must be able to let the user set multiple habits at the same time.
- The system must be able to reward the user for maintaining progress shown on a progress bar.
- The system must be able to deplete your progress bar for missing days and a missed day will add a day to build towards the habit
- The system must be able to let you add your friends and view their profiles to see their completed habits

5.b.) [10 POINTS] Non-functional requirements

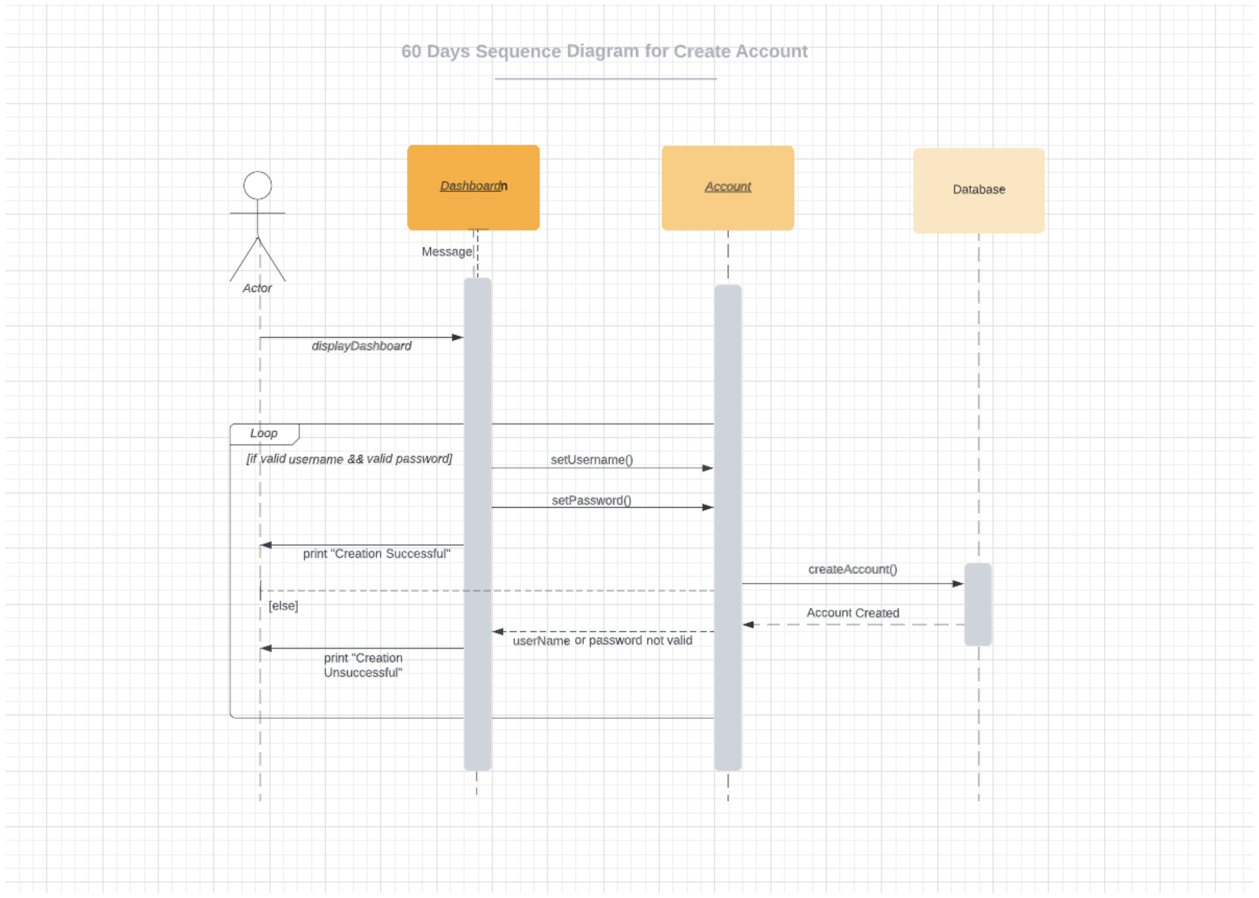
- Usability requirements: The interface must be easy to learn and navigate, and functions are easy to understand and simple to interact with and with few or no errors.
- Performance requirements: The system shall respond in no more than two seconds, and must be sustainable with multi-user concurrent execution where the load on server resources may not be more than 75 percent on average.
- Space requirements: The space needed for the software shall be within 100MB, any update size should be under 30MB.
- Dependability requirements: The system shall be available to the user 24/7 and downtime at in anytime shall not exceed three seconds.
- Security requirements: The system's firewall shall be able to prevent threats from malicious software. The server will cut its internet connection whenever attacks are detected.
- Environmental requirements: The system shall be working under ambient temperature, and the hardware should be kept between 50 to 90 degrees Fahrenheit.
- Operational requirements: A total load of habits shall not exceed 12 hours each day. Users should log in to the system with their own accounts to record the progress of their habits.
- Development requirements: The system must be adjustable and implementable so that it can be maintained and developed for further updated packages.

- Regulatory requirements: The software must be consistent with the regulation under all circumstances, such as Telephone Consumer Protection Act (TCPA), and General Data Protection Regulation (GDPR).
- Ethical requirements: The software must be ethical with the user’s privacy and data, the user data should only be used in the system, and any type of data leak is prohibited.
- Accounting requirements: The system must be performed by rules under U.S. GAAP basis financial statements.
- Safety/security requirements: The source code of the software will not be open source to prevent damage to the database and protect the user’s privacy

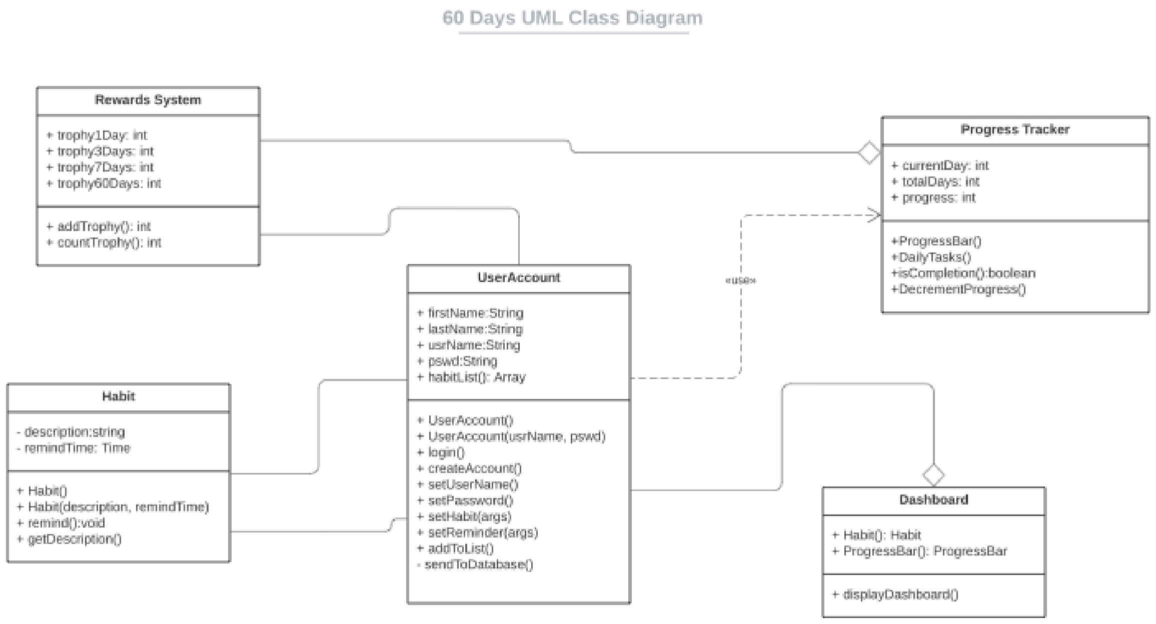
6. **[15 POINTS] Use case diagram –**



7. **[15 POINTS] Sequence diagram –**


60 Days Sequence Diagram for Create Account
Dashboardn
Account
Database
Actor
Message
displayDashboard
Loop
[if valid username && valid password]
setUsername()
setPassword()
print "Creation Successful"
createAccount()
[else]
Account Created
userName or password not valid
print "Creation Unsuccessful"

8. **[15 POINTS] Class diagram –**



9. **[15 POINTS] Architectural design –** Provide an architectural design of your project. Based on the characteristics of your project, choose and apply only one appropriate architectural pattern from the following list: (Ch 6 section 6.3) 9.1. Model-View-Controller (MVC) pattern (similar to Figure 6.6) 9.2. Layered architecture pattern (similar to Figure 6.9) 9.3. Repository architecture pattern (similar to Figure 6.11) 9.4. Client-server architecture pattern (similar to Figure 6.13) 9.5. Pipe and filter architecture pattern (similar to Figure 6.15)

The architectural pattern we'll be using in our project will be the Model-View-Controller pattern (MVC). We chose MVC for the specific use case of our system - in it there exist multiple views as well as multiple ways to interact with data. Additionally, future requirements for data interaction and data presentation are not yet known to us - however, with how mobile applications are built, it is natural to have updated releases in the future.

Moreover, users may create certain potential use cases that may require additional requirements in the future. The MVC model is good for adapting to these new requirements, as it allows the data to change independently of its representation and vice versa. Changes made in one representation will be made in all of them. Additionally, our project setup closely fits the MVC model in its design. For one, the Model component is carried out on the length of a habit that has been set so far and the progress building up to the end of the 60 days. The View component is represented through our app dashboard, and the Controller component handles when the user carries out key actions such as creating or deleting a hobby, and subsequently passes these interactions to the View and the Model. What is done with the interactions depends on the component they are passed to - either the results are client-facing, or they are done on the data and backend.