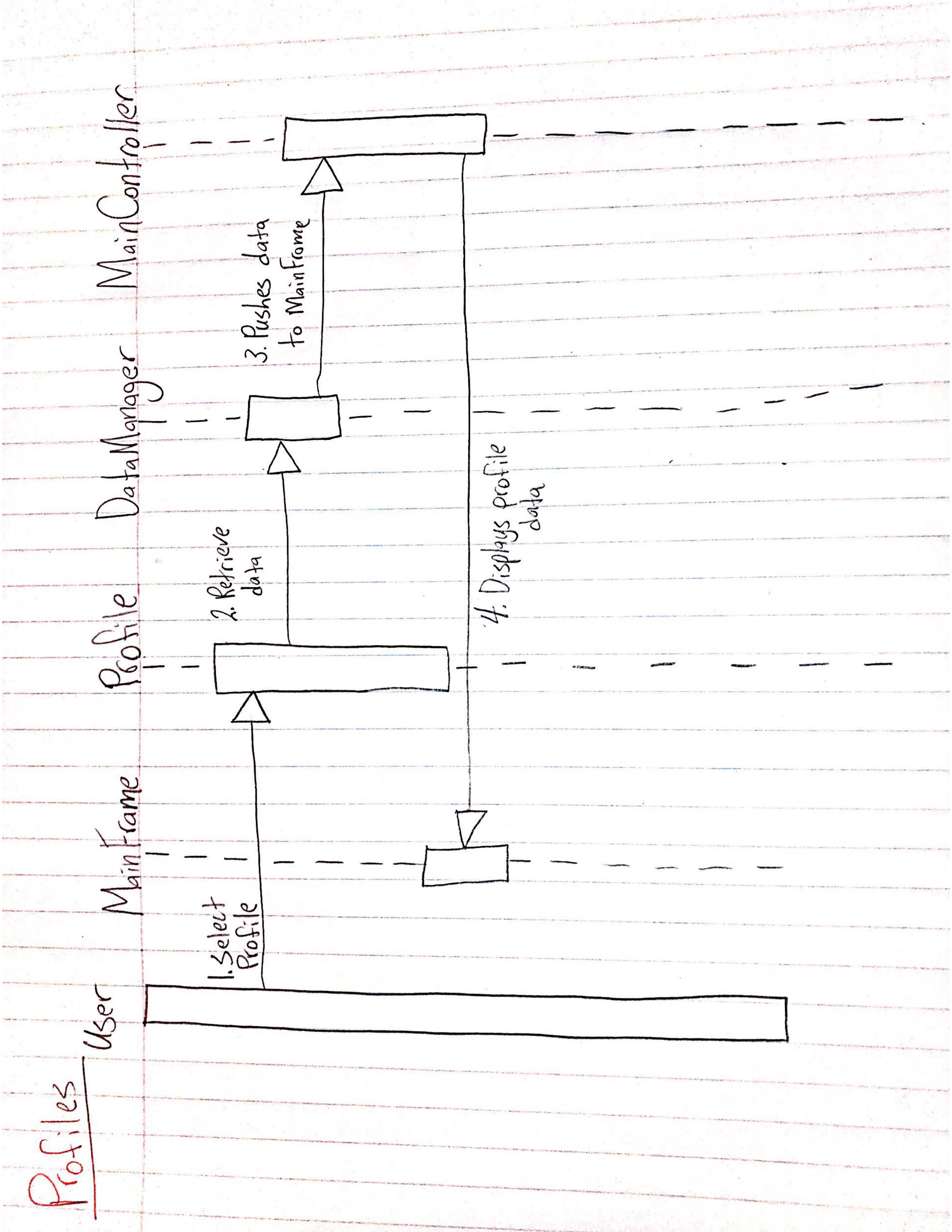
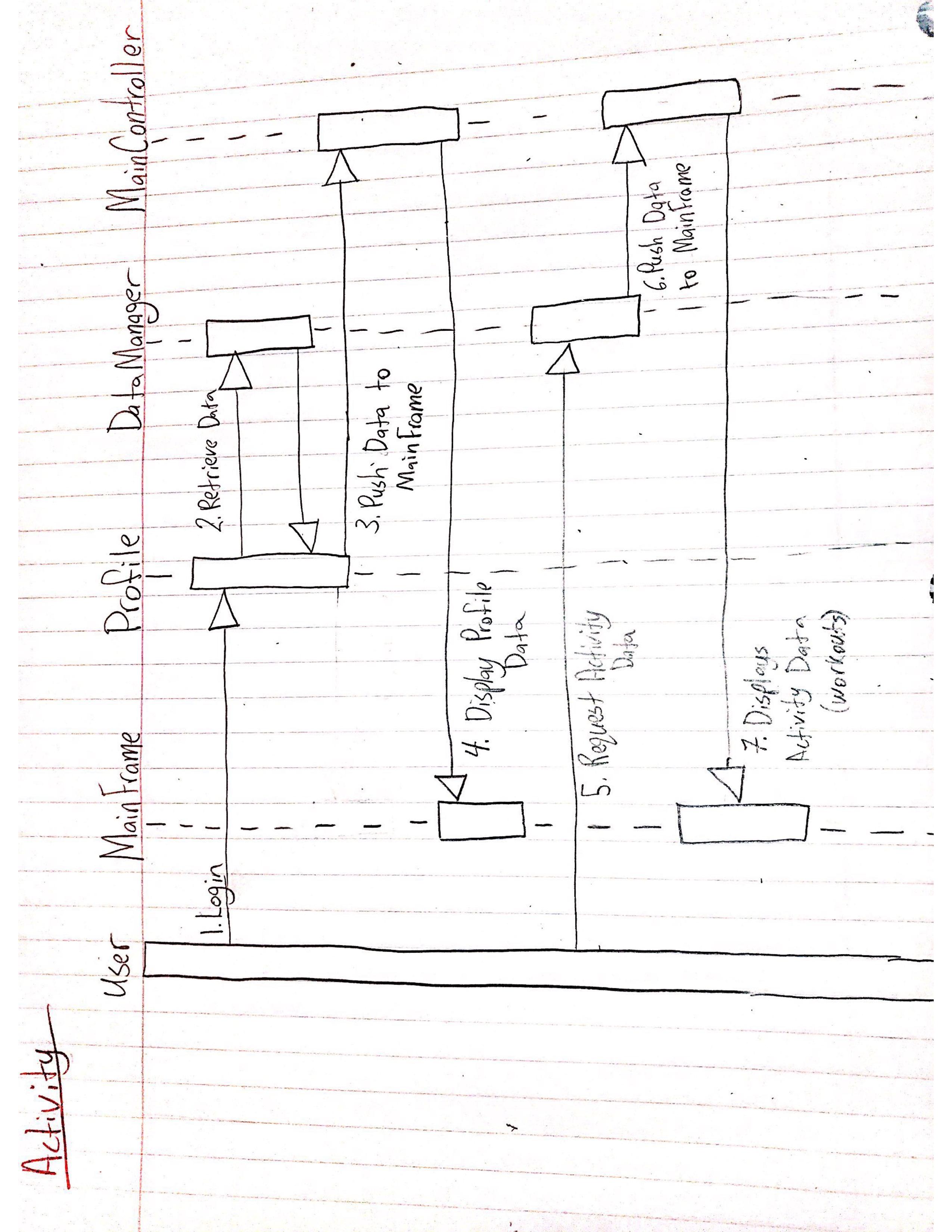
Group 6 Designing Architecture

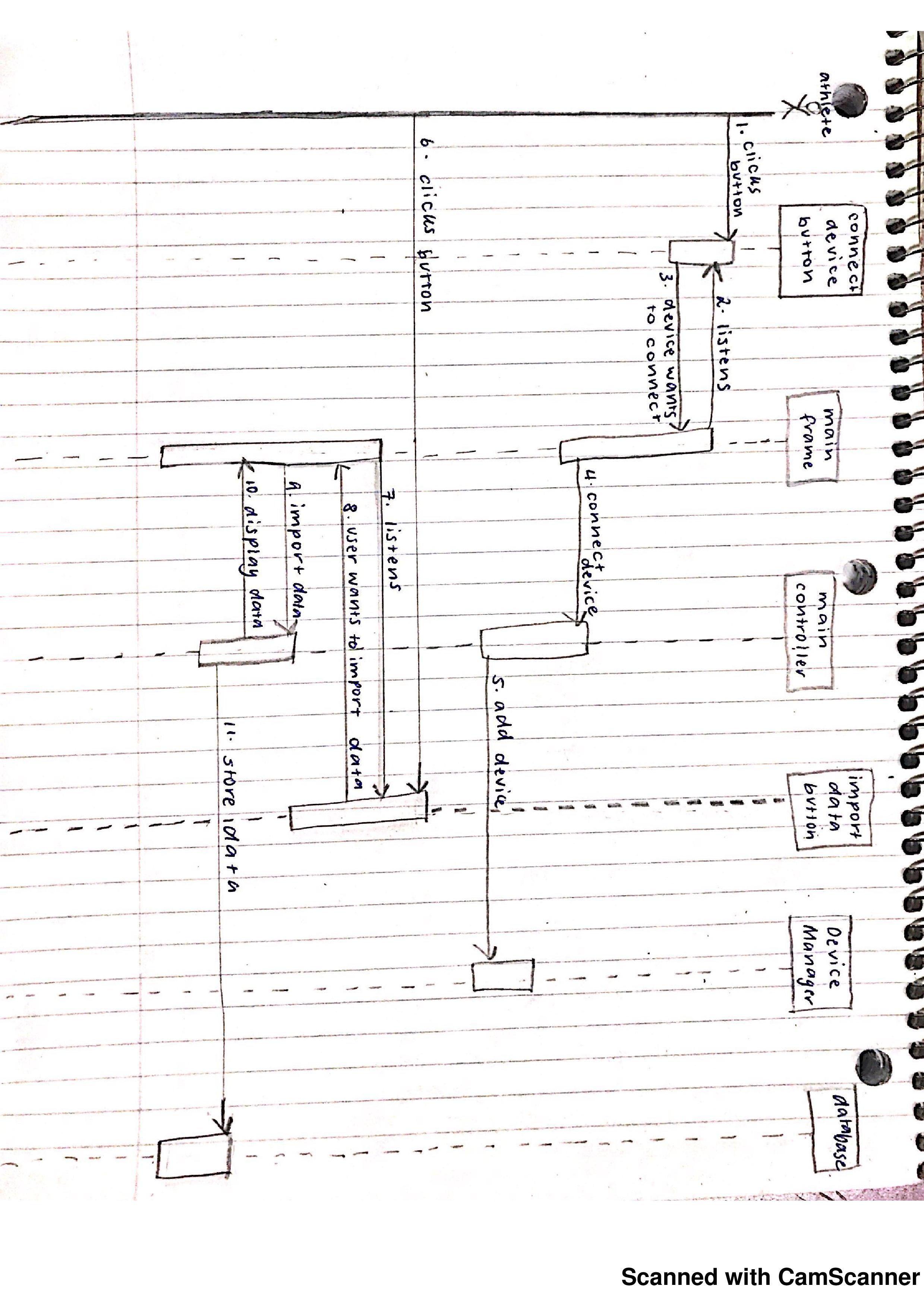
Prioritizing Features of System

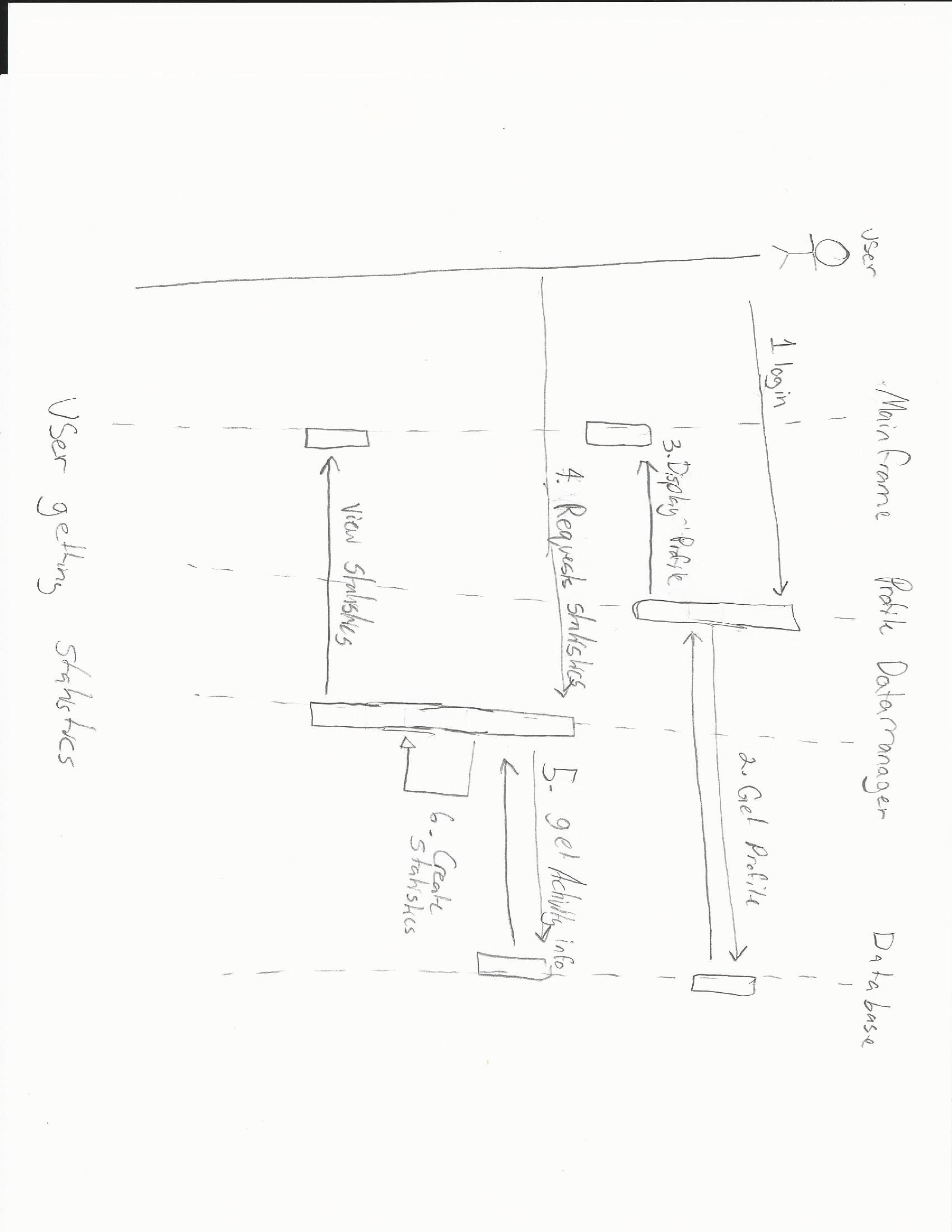
|  |  |  |
| --- | --- | --- |
| Feature | What’s significant | Why |
| F1: Allows for the creation of an account for each new user | Profile of the user, creation of the Account class | It is part of the essence of the system |
| F4: Ability to import activities from smart watches/devices | Importing the data recorded from the smart watch | We can implement this feature by importing data in from a text file and displaying data in a table format |
| F2: Can track a range of different activities, including runs, walks, biking etc | Creating a class for Activities | The system is an ActivityTracker, thus it is integral that the system is able to track different kinds of activities. We also know how we are going to implement this feature- by creating different instances of the class Activity. |
| F6: Provides the user with diverse statistics | Attributes of the Activity class | We can easily implement this feature by including different attributes under the class Activity. |
| F7: Includes analytics of the data (which was the fastest/longest run) | Identifying the top activities by pace and length | This is a feature we understand and know how to implement. |
| F10: Ability to sort runs based on pace, distance and time | Ordering the activities based on different attributes | This is a feature we understand and know how to implement. |
| F3: Ability to connect multiple devices | Creating a DeviceManager class | This isn’t a top priority, but can be easily implemented through use of a ‘DeviceManager’ which will essentially store a list of devices that have been connected. |
| F8: Stores data for more than one user | Implementing a database to store data | We want to make sure the system works for one user before we add in the ability to have more than one account |
| F5: Save imported data to the user’s profile | Saving the data to a profile | Although this is a pretty important feature of the system, it is not our top priority right now. We want the system to function properly before we start adding in the ability to store data. |
| F9: Ability to edit and delete data | Modification of data | The system can function without the inclusion of this feature. It is not a top priority. |
| F11: Allows for social interaction between users | Having the ability to add friends and potentially connect to social media | The system can function without the inclusion of this feature. It is not a top priority. |
| F12: Has a goal reward system | Creation of a goal and training program class | The system can function without the inclusion of this feature. It is not a top priority. |
| F13: Allows for personalisation of goals of each user | Creation of a goal and training program class | The system can function without the inclusion of this feature. It is not a top priority. |

Sequence Diagrams









Identifying Design Patterns and Key Decisions Made

Our first major decision that we made in the design process was to prioritize the creation of an Account class, as it is an essential part of the Activity Tracker Application. The Account class works with a DataManager class to save and retrieve profiles. It is used for creating a profile and retrieving profile information that has already been saved to the database. Having the profiles saved separately from the profile class makes it more straight forward to debug any issues with profile retrieval.

Another design decision we had to make was how to structure the classes for different activities. All the activities have the same statistics; the only difference is the type of activity that was performed. Due to the commonality between the activities, we have chosen to design one class for Activity that contains all attributes related to a workout (calories burned, distance, speed, etc.). The addition of activities other than running can be added by making an instance of the Activity class. This adds flexibility to the system by making it less complex to add more activities in the future. It is not necessary to make a new class for each activity that may be added in the future. We have applied the Don’t Repeat Yourself Principle here, so as to not have multiple classes for different activities that will have virtually the same code. The activity data is saved to the DataManager class where it is linked to a profile. Statistics are calculated in the DataManager and then sent back to the activity class.

A key decision has been to prioritize the other main components of the code before creating a database to store the information. Since the database is not going to be implemented until last, we have decided that creating a DataManager class that will serve a similar purpose will be used first. In the final product the DataManager will work with the database that contains all the stored data. Rather than storing the created statistics in the database, the DataManager will create the statistics with the data. Everything accessing the data will go through the DataManager initially; this task will then be given to the database which will be programmed later.

Another key design decision that we made was to create every object separate and so that it works on its own. Instead of having all the objects interact with each other, there will only be one controller for the GUI- the main frame, and another controller that works with the different classes- the main controller. The main frame will listen to all the objects connected with the GUI and when a button is clicked, the main frame will send a message to the main controller, that will then communicate with the necessary objects and classes needed to complete the relevant action. An advantage to this modular approach is that if a particular event does not work, the rest of the system will still function, and the problem will be easier to identify and resolve. This approach will also be more convenient further along in the design process, as this allows for flexible changes and easier additions to the software.

In addition to this, we have decided that the goals and training program will be a low priority for us to implement into the system, as it is not an essential part of the application, and once all of the other features are finished it should not be too hard to incorporate into the system. When thinking about the flexibility of this system, we thought it was best to implement the display table for data with unfixed table dimensions. This way, if a new statistic is added in the future, the table will expand automatically without needing to create a new column in the code.

The Single Responsibility Principle is one of the design patterns that we used to make the system more extensible. By applying this principle, the system has become more modular. Making changes and detecting issues with the code is less difficult to isolate with this approach as code is not being duplicated.

The main design pattern we ended up applying to our system is the Model-View-Controller pattern. Our GUI will take care of the view of the system, as we will add graphics to the system using Java Swing so that the user will be able to see something and interact with the system. The Profile, Synchronisation and Store Data modules will make up the model of the system, as they model what’s happening in the application. We will then write a controller class to manipulate the model of the system. The controller will make changes to the view of the application based on what the model is doing and vice-versa.