General Implementation and Discussion of Major Design Patterns

So, in our system, we had many several changes as we worked through it and this was expected to happen as it does for almost everyone. While making the final project, we made some important decision on how to link the system together and how to make it as efficient as possible.

The system has 6 different java files which extend JFrame and is part of our GUI for the system. Each page has a .java file associated with it and the reason behind it is specific. If we had all the GUI code in one .java file (which still is possible), the Jframe would look messy as the option we would have to go for is to use pop-up windows to display the different frames. This would clutter up the screen and be messy. We instead set different files for different frames and hide/show the Frame depending on what needs to be displayed.

We have 4 main classes in our back-end. We have Manager, User, Statistics and Database. The heart of the system in our case is the Manager class. We will get back to the manager class after discussing the implementation of the Database. Database also plays a huge role in our System because that is where all the data we have is stored. To aid us in this process we need a file to store the data when the system is not running so that we can refer to it while we run the system. We chose to use Apache POI library to save all the user info and the saved statistics in a excel workbook. Hence, we created a workbook file and whenever our system turns on or off the data get saved and when the system turns on, it takes the data again from the excel. The database.java file has many methods, but we commented on them to explain to a fellow programmer on how the database is working

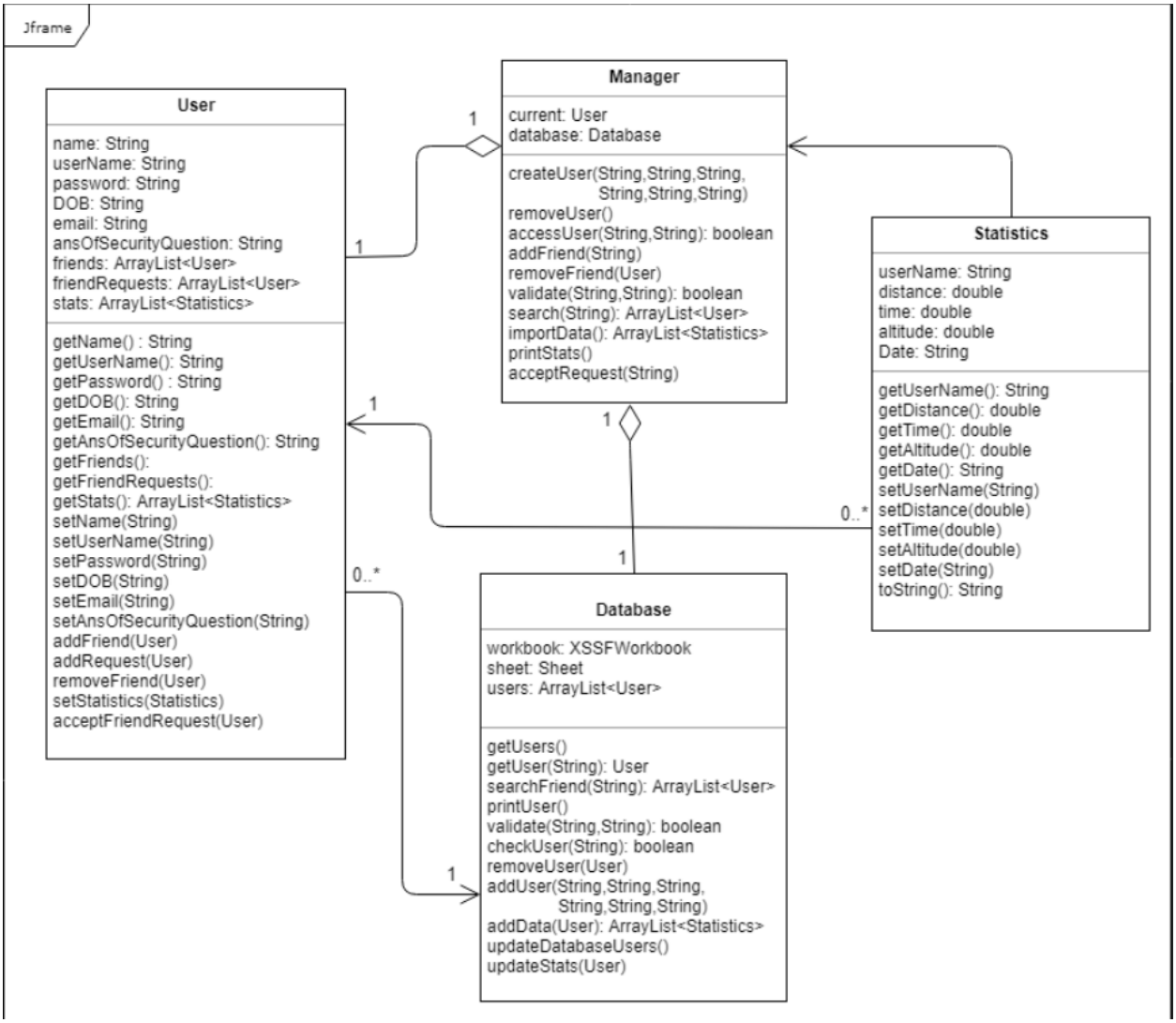
Our Database stores the ArrayList containing all the users and the workbook that we save all the file. Our User class has all the basic information regarding the User, his friends and statistics. Since we have a whole different subject, we chose to make a different class Statistics and chose to aggregate it inside the User to help save the repeated code and dupplication for future updates. User can have a list of Statistics Objects and can refer to them.

When the GUI is running, we need something that can refer to every other class in some sort of way, from the front-end we would need to access information from all the other classes because we would have to eventually display it to the running Jframe. This is where our Manager class helps. This, as mentioned earlier, is our heart to the system and it connect every component. The manager has two variables. It has a User variable type called current and a Database type variable called database. Since we are having a multi-user application, we need to keep track of who's currently working with the system and current has that. Hence, from current we can refer to the user object instance and the statistics of the user since it's a part of the user object as explained before. The only remaining reference we would need is the

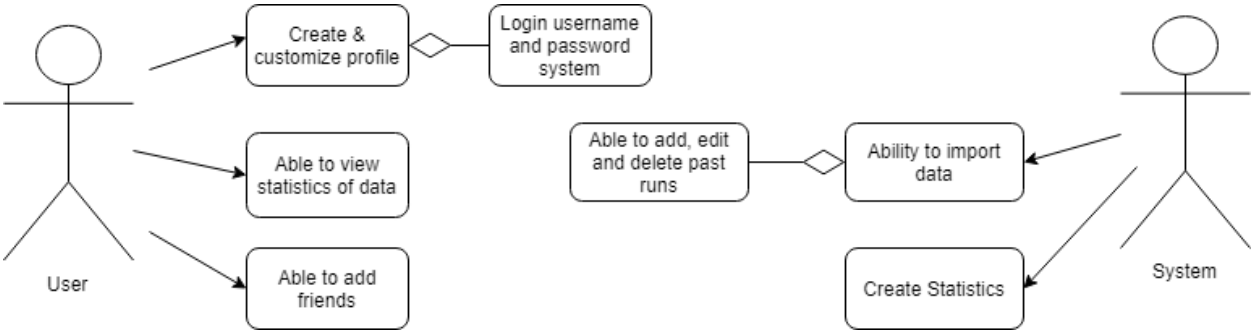
database and we make a Database type to keep track of it. Hence the main principle we use is delegation. This helps us also maintain the SRP principle. We will come back to that in a bit. The Manager class has to get access to the database to get information about user or information about the friends and the Manager class has a User object variable “current” which refers to the user object. But we use delegation to use the same methods that are in User class and implement them in Manager class according to what is required in front end.

Generally , one would just copy/paste the code in both the classes and it would work but that would be very messy and duplicate code is a hassle so instead we delegate the manager class methods to the User class methods. This can be seen from the UML diagram but for example there is a method in the Manager class called “removeFriend(User friend)” but we also have “removeFriend(User friend)” in the User class so instead to copying the code we refer to it by using the “current” reference. {current.removeFriend(friend)} So hence we call the same method in the User object and it works. This helps solving our problem and makes the system more efficient

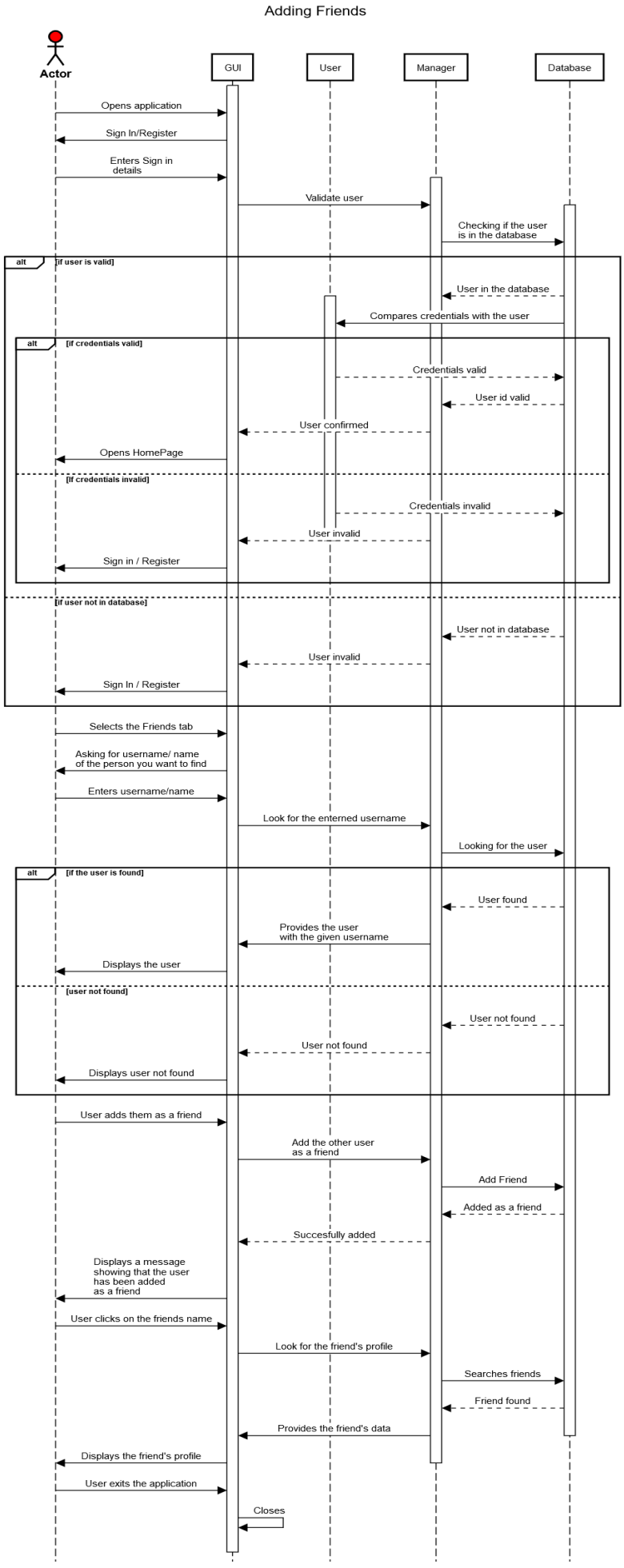
UML



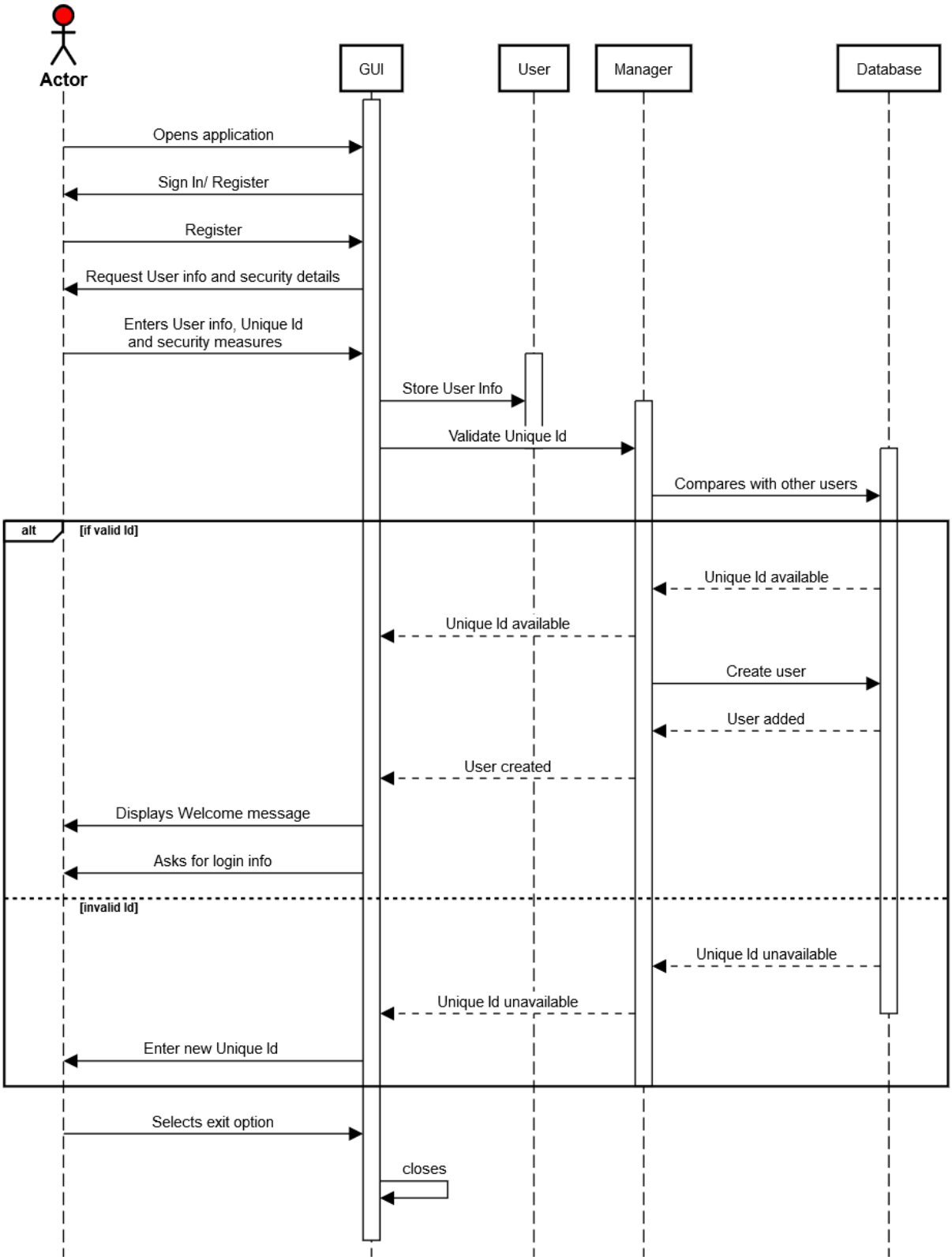
UseCase Diagram



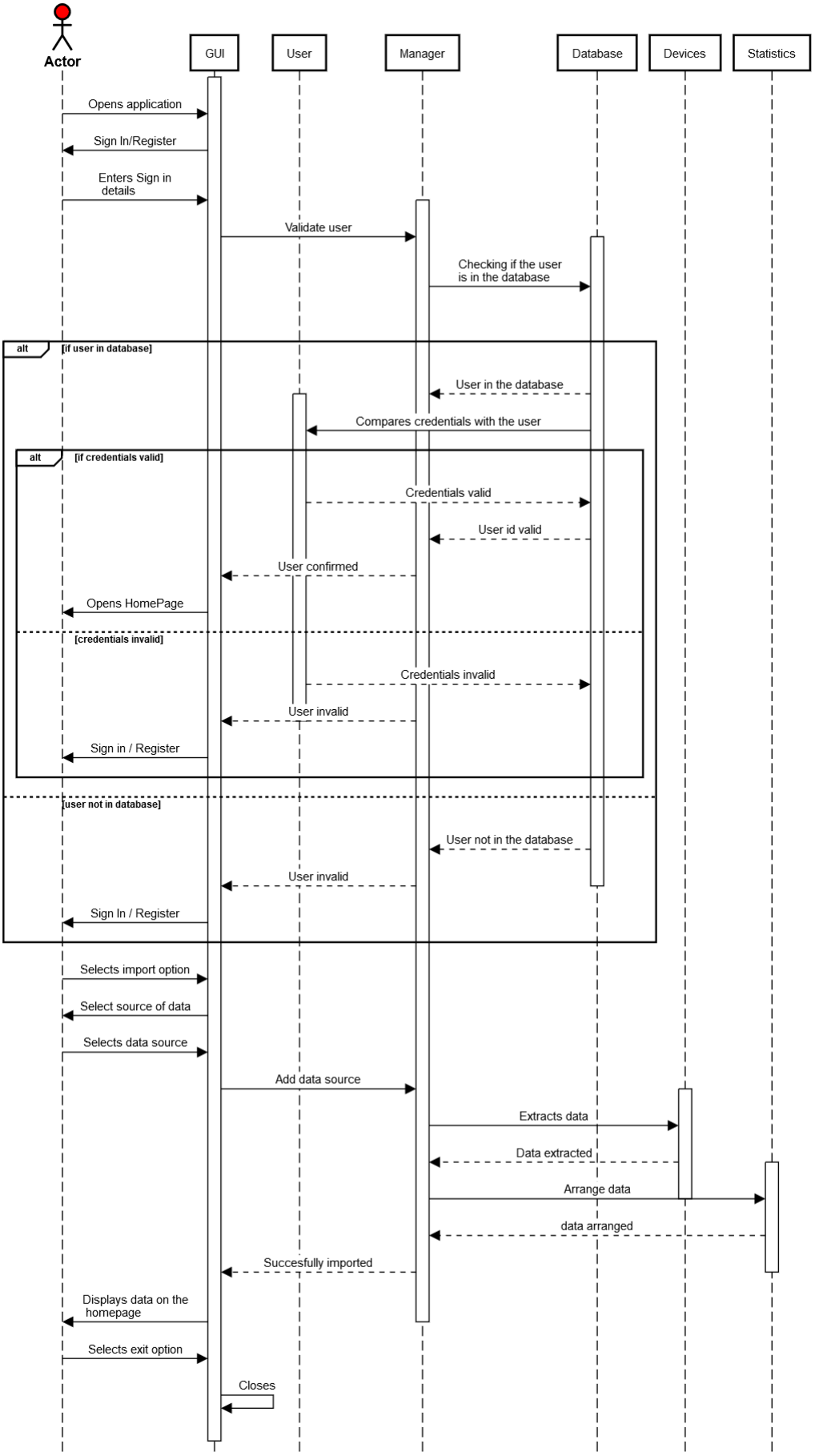
SEQUENCE DIAGRAMS


Adding Friends
Actor
GUI
User
Manager
Database
Opens application
Sign In/Register
Enters Sign in details
Validate user
Checking if the user is in the database
alt
[if user is valid]
User in the database
Compares credentials with the user
alt
[if credentials valid]
Credentials valid
User id valid
User confirmed
Opens HomePage
[if credentials invalid]
Credentials invalid
User invalid
Sign in / Register
[if user not in database]
User not in database
User invalid
Sign In / Register
Selects the Friends tab
Asking for username/ name of the person you want to find
Enters username/name
Look for the enterned username
Looking for the user
alt
[if the user is found]
User found
Provides the user with the given username
Displays the user
[user not found]
User not found
User not found
Displays user not found
User adds them as a friend
Add the other user as a friend
Add Friend
Added as a friend
Succesfully added
Displays a message showing that the user has been added as a friend
User clicks on the friends name
Look for the friend's profile
Searches friends
Friend found
Provides the friend's data
Displays the friend's profile
User exits the application
Closes

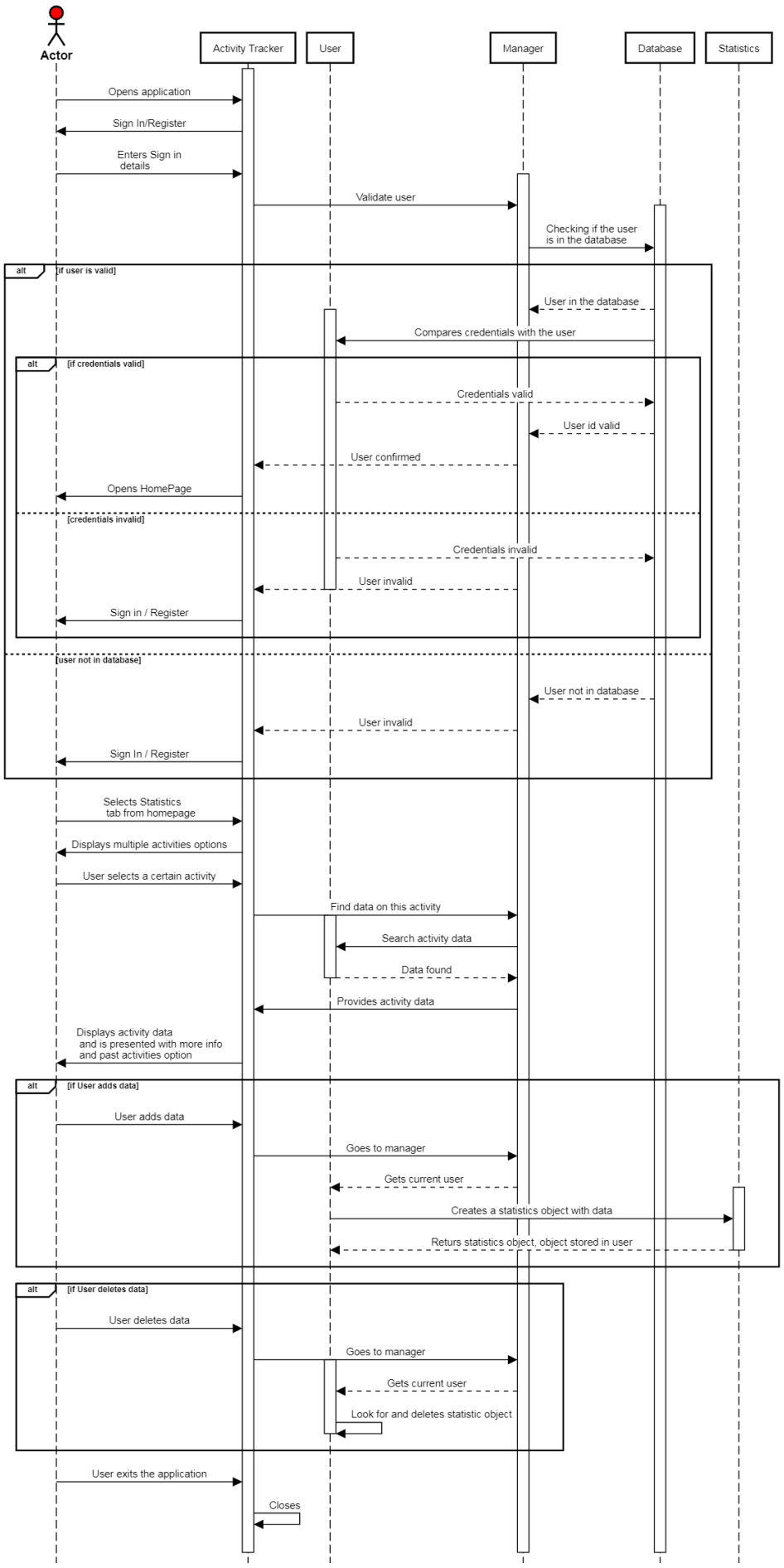
Creating profile

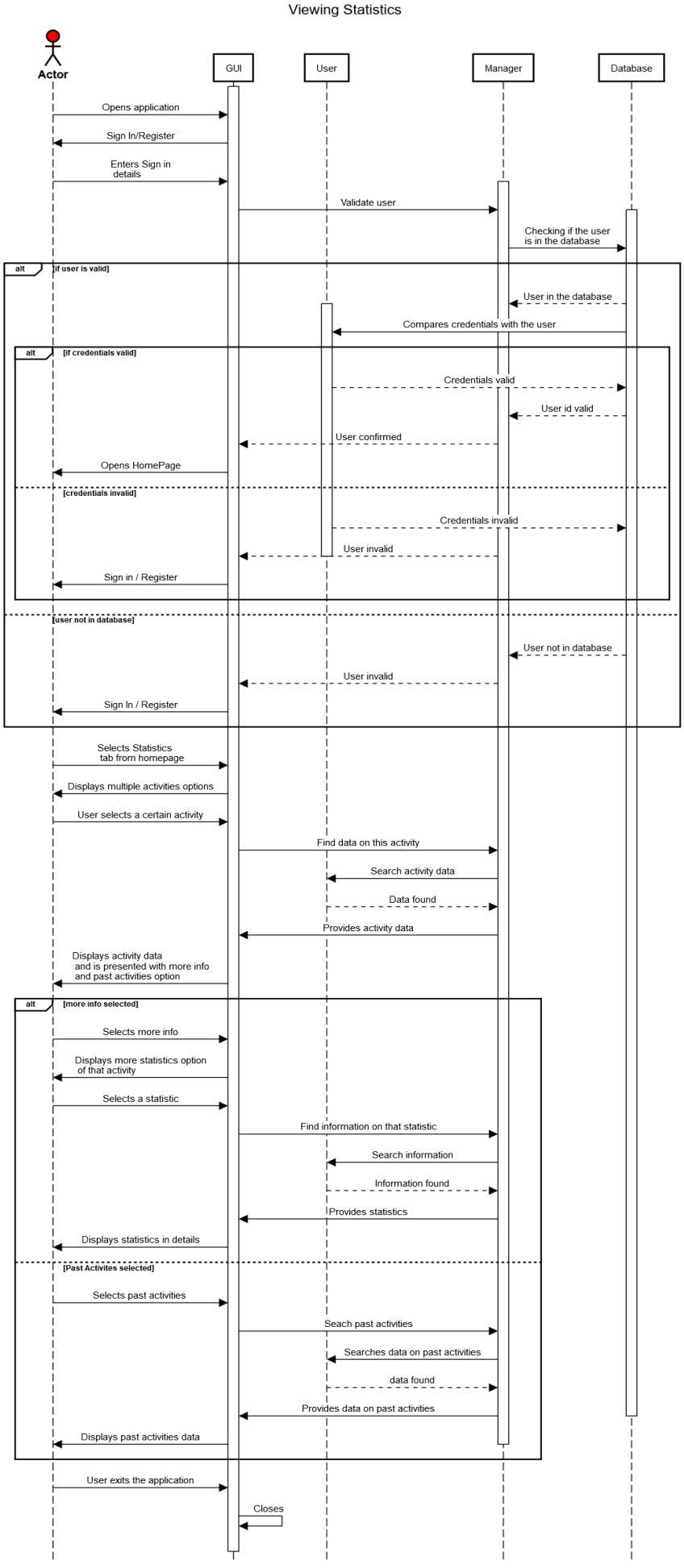


Importing data



Manage Statistics




Viewing Statistics
Actor
GUI
User
Manager
Database
Opens application
Sign In/Register
Enters Sign in details
Validate user
Checking if the user is in the database
alt
[if user is valid]
User in the database
Compares credentials with the user
alt
[if credentials valid]
Credentials valid
User id valid
User confirmed
Opens HomePage
[credentials invalid]
Credentials invalid
User invalid
Sign in / Register
[user not in database]
User not in database
User invalid
Sign In / Register
Selects Statistics tab from homepage
Displays multiple activities options
User selects a certain activity
Find data on this activity
Search activity data
Data found
Provides activity data
Displays activity data and is presented with more info and past activities option
alt
[more info selected]
Selects more info
Displays more statistics option of that activity
Selects a statistic
Find information on that statistic
Search information
Information found
Provides statistics
Displays statistics in details
[Past Activites selected]
Selects past activities
Seach past activities
Searches data on past activities
data found
Provides data on past activities
Displays past activities data
User exits the application
Closes

Design specific implementation (From Assignment 3)

The following document contains design specific implementation for all the priority features, followed by the reason for choosing the specific method of implementation

1) **Creating a profile and Sign in**
   a) The user must have a unique userID to create new profile
   b) User must fill all the required details to create the profile
   c) The system should validate uniqueness of the userID and strength of password
   d) System provides an ability to reset password
   e) System should store the all the details once an account is created.
   f) System should be able to access the userID whenever necessary

   Implementation: In order to implement the requirements above the Sign Up page is connected to the manager class, when the user clicks sign up the information added by the user is validated (including userID and password). If the validation goes through then the user profile is created by creating a user class object, else user is given a specific message with the error. To create and store the user profile the manager class adds the data to database class which intern saves the data in a connected file/server and also saves a copy of user object. While signing in the userID and password are validated by the system and the user is taken to the first page if they match and are valid, else user is shown an error message. To do this, when the user clicks the sign in button, the manager class is accessed, and the manager class accesses the database to validate the information.

   Reason: The database class stores user class objects which contains user details, this is done to gather all user objects in one place from where they can be accessed. The manager class creates an instance of database and uses it to store and access the data (user objects), this is done because there needs to be only one database which can be accessed by manager and creating one instance does the job. Finally, the GUI will create and use a manager instance to do the tasks. This is done because only one manager class instance is required for all the necessary system tasks.

2) **importing data**
   a) The user should be able to connect multiple device or import data from file
   b) The system should store previously imported data
   c) The user should be able to undo a previous import
   d) The user should have ability to select specific runs from data

   Description: In order to implement the requirements above the import wizard is used in GUI. The user could select the brand and type of device, they system would look for any such connected device. If the system finds such device and if it is not already stored in the user's data, then a device class object will be created and the reference to this object will be saved in the user's user class object. If the user selects to import data from file the user will be asked to select a file, system will check if the file is valid and if it is then the system will use the data in file to create a activity specific class object (ex: run class object) and a reference to this new object will be stored into the user's user object

   Implementation and Reason: As the user will click the import button in the import wizard, the previously created instance of manager class will be called to validate the details of import. If

valid a Device object will be created (if device was selected) and stored in current user's user object which will be accessed through manager class who will always know which user is currently active. An activity specific class object (ex: run class object) will be created with the data from device (or if the user chooses to import from file). Again, this object will be stored in current user's user object (accessed as stated previously). The activity specific classes will be sub classes of statistics class and will contain all the attributes of statistics class. The manager class keeps track of current user object to access it quickly. The user object needs a reference to device and statistics object to keep track of which objects are connected to this particular user, keeping a list of all device and statistics objects will fulfil this requirement. The activity specific classes are sub classes of the statistics class because the statistics class will contain the attributes which are same for all activities and the activity classes will contain the general attributes plus the attributes specific to them. By having a super class to contain general attributes we avoid duplication of code.

3) **Viewing stats and Managing runs (add, delete)**
   a) System should show the user all the statistics which are connected to the user
   b) The user should have the ability to organize the statistics by date/distance/time etc.
   c) The user should be able to view different activity statistics separately
   d) The user should have the ability to select and add/delete specific records of run

   Description: In order to add the above requirements to system. The system will have an activity classes (subclass of statistics class) which will keep track of each activity record. Each record be stored in particular activity's class for segregation. The activity classes' objects (records) will be connected to the user objects (user objects will have reference to activity specific objects ex: run class object). To access the data the GUI will call the previously created instance of manager and the manager will class the instance user (current user) and the GUI will retrieve and display the statistics objects in a readable format, in the GUI page. The GUI will also omit the records which doesn't belong to activity selected by user. The GUI will change the arrangement of records if the user wish to sort them differently, the GUI will also provide an option to import more data and delete specific records. If data is added implementation in import data will be used, if a record is deleted then the GUI will access the user object (as it did previously) and delete the reference to that particular object (record).

   Implementation and Reason: To implement this, the GUI will have an instance of manager class which it created previously. The manager instance will have a reference to current user at all time to access it quickly. Once the current user is accessed the user will have a list of all the statistics objects (the objects will be from specific activity classes) connected to it. The GUI will look into these objects to identify which objects are for the activity requested by the user. It will then display this subset of objects to the screen. When a record is deleted by the user the GUI will access the user object (as it did previously) and delete the reference to that particular object (record). The activity specific classes are sub classes of the statistics class because the statistics class will contain the attributes which are same for all activities and the activity classes will contain the general attributes plus the attributes specific to them. By having a super class to contain general attributes we avoid duplication of code.

4) Adding friends

a) The user should be able to search for people using search box
b) The user should be able to add multiple friends
c) The user should receive a notification when he/she receives a friend request
d) This user should be notified when his/her friend request is accepted
e) The user should be able to remove a friend

Description: To add the above functionality to the application, each user object will have a list called friends and a list called requests. When a user will receive a friend request, a reference to sending user's object will be added to the list and the receiving user will be notified about the request. If the user accepts a friend request, then the user object will be moved from requests list to friends list. The user who sent the request will receive a notification that his request was accepted and the friends list of this user (sender) will be update with new friends' user object. The user could send a request be looking up using search box, which looks through users list in database class and returns all users with matching name (as search query). This list of users will be displayed in GUI. The GUI contains search box, when search button is clicked the GUI will access the previously created instance of manager and manager will class search in database. GUI will receive list of all users through same channel and will display the list on screen. The user could click add friend from here. When "add friend" is click the GUI accesses the user object and adds current user's object reference in requests (current user's object reference retrieved from manager).

Implementation and Reason: To implement the solution a manager instance will be a created and used through GUI. The manager instance will create and use a database instance to keep track of user objects. The manager instance will also keep record of the current user. The database instance will contain a list of all the user objects part of system. The user objects will have two list, friends and requests, which will contain user objects as the name suggests. This implementation approach is necessary because the GUI needs to be connected to one place from where everything can be accessed, and the manager instance is the best answer to that problem. The manager instance needs access to one database which has all the information about users and the database instance does the job. The user objects contain list of other user objects to keep track of which users are connect to each other. This is important to show the users' data about their friends. Having a list of user objects also makes it easier to locate friend data from the current user.