**Design Document**

**Adelbert Choi (s3618753)**

**Miracle John Jr Octavio (s3638336)**

**Both members equally contributed to this project**

**1. Explain the changes if you use a different design compared to your assignment 1**

Our network design is very similar to what we did for our assignment 1. We have an abstract user class, and we have three subclasses extending this abstract class (Adult, Child, YoungChild classes). These classes have instance variables storing the profile information of each user. Also, each user created will have their own list of friends.

We also have a relationship class that stores each user’s relationships. This relationships class keeps track of all connections between users and their relationship types. This is required since the list of friends instance variable of each user does not specify relationship types. It is good note that two relationship objects are always created for each relationship between people in the social network. In particular, if user A and B are friends, then relationship objects pertaining to A 🡪 B (A is friend of B), and B 🡪 (B is a friend of A) will be created.

We also have a People class that controls all users. Similarly, we have a Connections class that manages all relationships instantiated between users. Both people and connections objects are specified as public static instance variables in the main startup class MiniNet. These instance variables are made public static to allow GUI classes to mutate them. This avoids having to pass these objects around the entire application.

(see figure 1 for class diagram of social network design)

**2. Explain how the new classes are organized**

We did not make a lot of changes in our social network design. We did however add classes to implement GUI and data (/file handling).

For GUIs, each scene in the application has its own class. Moreover, each scene class created has a single purpose. Also, the methods that are within each GUI class assist to achieve its purpose or a certain need.

**The following enumerates each GUI class created and a brief description of its purpose.**

1. MainMenu: main menu of the social network application

2. SearchUser: search a user in the social network

3. AddUser: add a new user to the new social network. The type of user to be created depends on the age input.

4. ListAllUsers: this is an alternative to search user. Instead of searching for a user throughout typing their username, a username can be selected. The profile of the user with the selected username is then selected.

5. AddChildUser: continuation of AddUser if a child/young child is desired to be created.

6. ShowUser: Displays the profile information of a certain user

7. AddFriend: Adds a relationship to certain user. The possible users to add to a certain user’s network depends of their type (e.g., Adults are only allowed to add Adults as a friend/classmate/colleague).

8. DeleteFriend: Deletes one of the relationship of a user.

9. EditUser: Edit the profile information of a user

10. FindFriend: Additional implemented to find the shortest path to a specific friend in the social network.

**How our classes are organised:**

There is a hierarchy of scene calls amongst GUI classes. In particular, starting at the main MiniNet class the MainMenu scene is instantiated. The MainMenu can instantiate SearchUser, ListAllUsers and AddUser classes. To provide another illustration we have the ShowUser class. This class can instantiate AddFriend, DeleteFriend, FindFriend and EditProfile classes. In summary, only certain GUI classes can instantiate other GUI classes. A visual representation of this hierarchy is shown as a class diagram in figure 2. Overall, everything starts in the MiniNet class, and most GUI scenes can go back to the MainMenu class.

On the other hand, when handling data (i.e., user information from text file/database). File handling classes are initially instantiated by the MiniNet main startup class. This is to initialise the profiles in the social network. Corresponding errors are thrown and handled when no data/file are seen by the program.

(see figure 3 for class diagram of social network design)

**3. Explain the process by which your program will interact with user and external data source to the social network.**

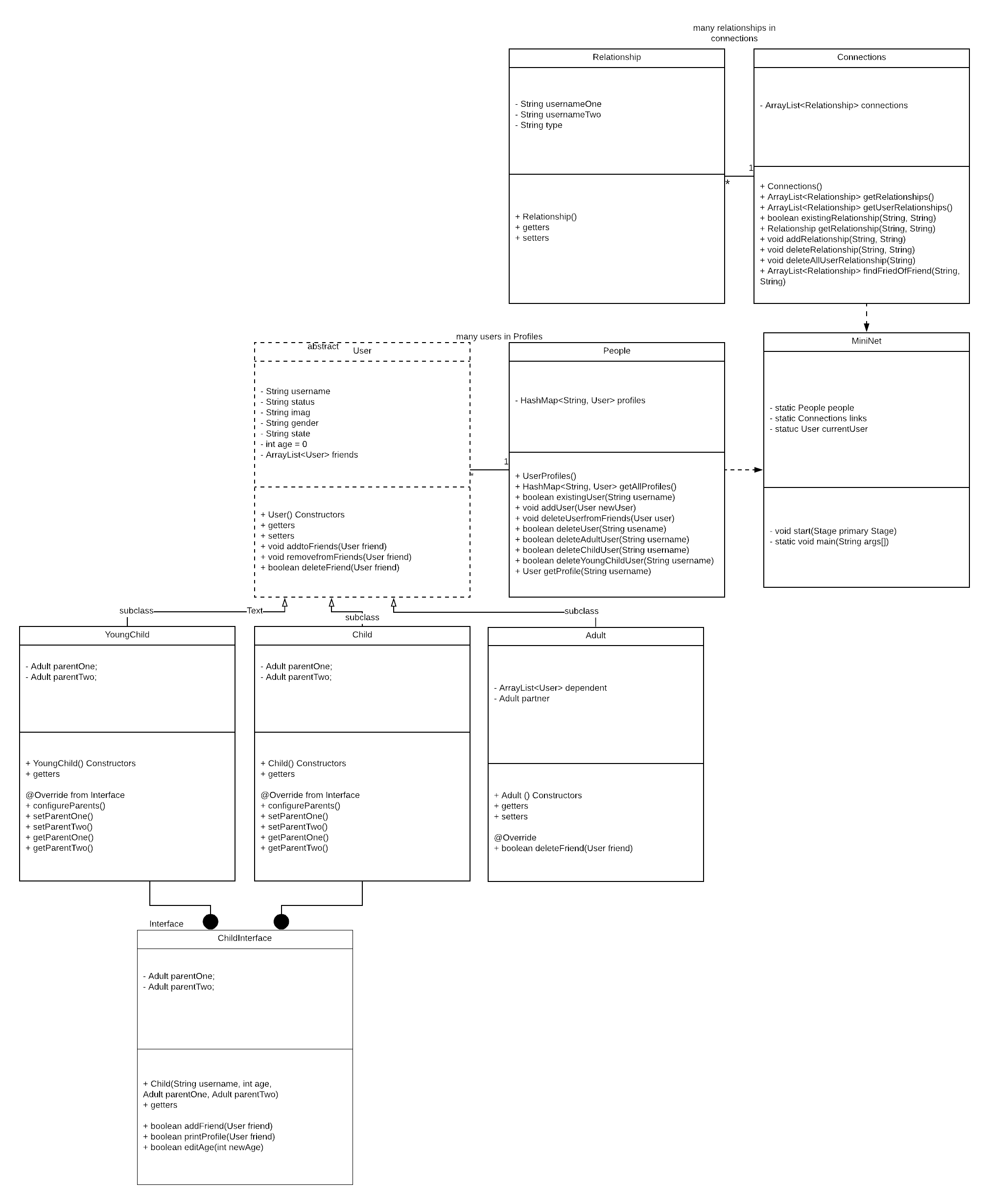
Each GUI class designed are aimed to make a user’s life easier by preventing the occurrence of errors. Using AddFriend() scene as an example, this scene initialises a list view showing the users in the current social network that can be added as a friend to a current user of interest.

To illustrate the class AddFriend() shows the scene when a user desires to add a friend to a user in the current social network. The class has its own instance variables that are needed to be initialised for the entire scene. All methods in this class support the objective of the class which is to add a friend to a current user.

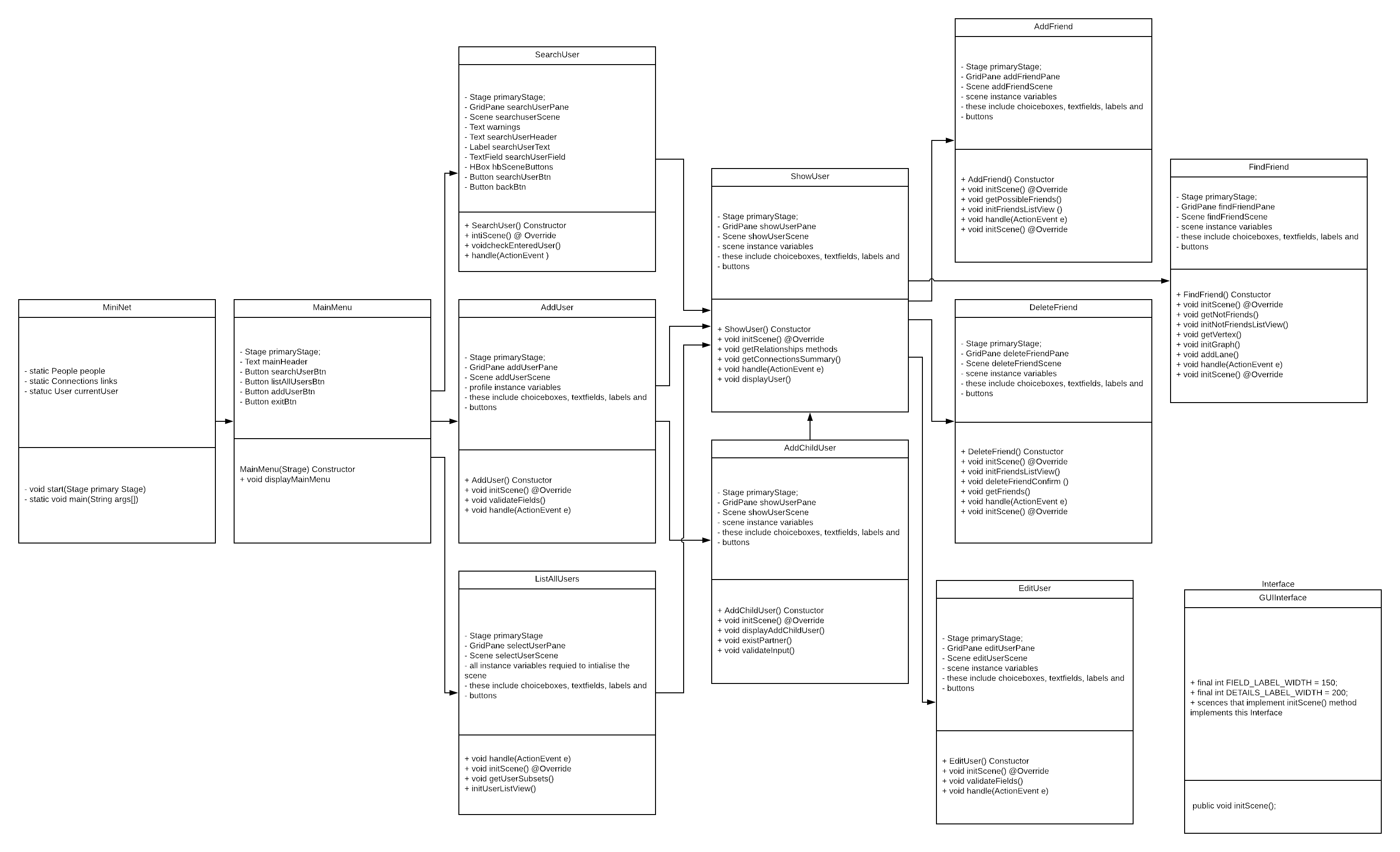
If the current user is an Adult type user, then the list view of users will only contain Adult type user usernames that can be added as a friend (plus other specifications specified in the requirements are met). This is similar for child and young child users. We think that this design/user friendly interface is a good and thoughtful design. All our scene designs created are keeping this in mind.

On the other hand, text file and SQL database handling are initially implemented to add data to the social network. In situations where no existing data files are detected, the program will throw appropriate error messages, and exceptions are handled accordingly.

One scenario is the inability of the program to find a text file to retrieve data. If no text file is present the program will try to find a database connection. If a connection is found, data from the database will be read and used in the application. However, in a situation where no text file or database connection is found, then the exception is handled and an error message is shown.

**Figure 1. Social Network components**

**Figure 2. GUI Class Diagram and GUI hierarchy**



**Figure 3. File Handling Classes Class Diagram**

