LBYCPEI

**Object-Oriented Programming Laboratory**



*Art by @Criisalys: https://twitter.com/Criisalys/status/1679603409261936643?s=20*

**Laboratory Module 7**

Graphic User Interface (GUI)

By

Jay Michael T. Carlos, LBYCPEI-EQ8

# INTRODUCTION

Previously, you have used the ACM library to create a graphical user interface to allow users to interact with your program easily and intuitively. You may have noted that it can be quite a hassle to make complex interfaces in ACM because of the steps needed to display objects on the screen. In this module, you will be learning about JavaFX and SceneBuilder, which is an easier way of doing so. JavaFX is a library designed to develop various types of applications for multiple operating systems and platforms (@nishkarshgandhi, 2021).

1. Objectives
2. To understand how to make a user interface in JavaFX.
3. To develop a graphical program in JavaFX.
4. To implement styling in a JavaFX program using CSS.
5. To create a social media app in JavaFX.
6. Materials
7. IntelliJ IDEA (The Java IDE)
8. Java SE 14
9. JavaFX SDK
10. JavaFX Gluon SceneBuilder

# PROCEDURES (Individual) / EXPERIMENTAL PLAN

1. In Laboratory Activity 1: "SocialNet”: The process of creating a JavaFX project is somewhat different from a regular Java project. Go to File > New > Project and select JavaFX in the left sidebar. Then, rename your project appropriately and change the Artifact if needed (it does not play well with spaces and dashes). Click on Next, then click on Create without adding any of the libraries, as we are not going to be using those for now.

In your project directory, you will see two Java files in src > main > java > com.example.projectName: a main file and a controller file. The main file initializes the app and will generally not be modified as much as the controller file. The controller file is where all the functionality of your program will go. By clicking the run button in the top right, you will see a sample program with a button, that when clicked, shows a “Hello world” message.

You can first start by modifying the SocialNet.java class. You will only make two changes. First, in the line that creates the Scene, after fxmlLoader.load(), add two arguments 1280 and 720. A scene is a single app screen; so, if you have, say, a messaging app, the list of your contacts is its own scene, and when you open a person’s messages, that is also its own scene. The arguments 1280 and 720 define the dimensions of the window. Next, in the line stage.setTitle(), put a string argument containing whatever you want to name your new app. In the code at the bottom of the document, though, the app is called “SocialNet”.

Next, open the resources > com.example.activityName folder and open the hello-view.fxml file. You can rename this if you like; however, you must make sure to change the fxmlloader line in SocialNet.java (your main file) to load the new file name. Right-click the hello-view.fxml tab at the top of your screen and click on “Open in SceneBuilder”. This option should be close to or at the bottom of the context menu. A new window should then open.

A screenshot of a computer

Description automatically generated

Your goal now is to create a layout that looks somewhat like this. Note that it does not have to be styled yet; that can be taken care of later. However, do try to recreate the layout as closely as possible to avoid any errors in following this guide. On the left side of your screen, you will see a “Library” and a “Document” pane. The “Library” pane contains various elements available for you to place in your app. The ”Document” pane contains the elements already in your app, and how they relate to one another in the hierarchy. To add an element from the library to your app, simply drag the item to the canvas. You will notice that by default, anything you place down gets put in the very center of the canvas; this is because it is inside a VBox, which evenly distributes items inside it. We can work with this. Drag a ToolBar and a SplitPane (horizontal) to the canvas. The ToolBar should be at the top. You may have noticed that the “Inspector” pane on the right side of your screen now has various options when you click on elements. This is one of the ways you can modify properties of the onscreen elements. Drag the following elements into your ToolBar:

* A Label (or you could put an ImageView to act as the logo)
* A TextField (as the search bar)
* Three Buttons (to add, remove, and lookup people in the database)

In the SplitPane, add the following elements to the left pane:

* A Label (with the text “Update your profile”)
* Five TextFields (one to update a different part of the current user’s profile)
* Five Buttons (one below each TextField to update the appropriate information based on the TextField input)

In the SplitPane, add the following elements to the right pane:

* An ImageView (user’s profile picture)
* Three Labels (user’s name, status, and quote)
* One Label (with the text “Friends”)
* One ListView (display the user’s friends list)

To change the text in a Button or Label, simply double-click it. To change the prompt text in a TextField, click on it and edit the “Prompt Text” field in the “Inspector” pane. Note that you should edit “Prompt Text”, not “Text”, as “Prompt Text” disappears when your user clicks into the field, while “Text” must be deleted manually. Next, click on any item, and at the bottom of the “Inspector” pane, click on “Code”. The first field you see, labeled “fx:id” is how you will access each element from your code. Give each of the items on your canvas a descriptive ID, like statusField for the TextField where the user can type a new status.

After this, switch back to IntelliJ (do not close SceneBuilder, though, as you will need it later). When you run SocialNet.java, you will see your layout in a new window, although none of the buttons do anything yet, and the information displayed is incorrect. In the next parts, you will fix this.

Create a new file UserInterface.java by going to File > New > Java Class and selecting Interface. Inside the interface, first create getter and setter functions for the name, status, quote, and profile picture path. The getter functions should return a String, while the setter functions return void and accept a String parameter. Then, create addFriend(), removeFriend(), getFriend(), and getFriendCount() methods. addFriend() and removeFriend() both return void and accept a String parameter friend. getFriend() returns a String and accepts an integer parameter index. getFriendCount() returns an integer. With this interface, you have the building blocks for the next class you are going to create: User.java.

Create a new class “User” by going to File > New > Java Class. Make the User class implement UserInterface. Inside the class, create protected Strings for the user’s name, quote, status, and profile picture URL. Then, create an empty, protected ObservableList of Strings called friends that will serve as the user’s friends list. Then, create a constructor that accepts four String parameters: name, quote, status, and profileURL. In the constructor, set the following:

this.name = name;

this.quote = quote;

this.status = status;

this.profileURL = profileURL;

The four pieces of info will be added to any new instance of the User class based on what info is provided in the method call. Then, create the getter functions by returning the value that should be searched for from the User object. Then, create the setter functions by setting this.fieldName to the parameter value. The getter functions should return a String and should have no parameters.

Then, create a new class UserDatabase.java.

Now, open SocialNetController.java. Create an int currentUserIndex with a value of 0. Then, create a new instance of the UserDatabase class in SocialNetController. The @FXML line is required before each method declaration and each item’s fx:id. You need to declare each with the appropriate object type and fx:id. For example, if you have a TextField called statusField, you must type the two following lines in your controller file:

@FXML

private TextField statusField;

Do this for all the fields, buttons, and labels that can be changed or updated. One given method is the initialize() method, which looks like this:

@FXML

private void initialize() {

}

You will add code here later. This code will be run as soon as the app launches.

Create a .txt file in your resources folder and put all the user info in that file, with each value separated by commas. Each user is on their own line. Here is an example:

Jay Carlos,i can fix her,🍞 <- mmm baguette,jay.png,butter dog,Scott Wozniak,LowTierGod,Jebediah Schlatticus,Tom Anderson  
Bill Gates,Success is a lousy teacher. It seduces smart people into thinking they can't lose.,Bill Gates is speaking...,bill.jpg,Steve Jobs,Tom Anderson  
Steve Jobs,Beating you is Apple II easy.,Launching the iPhone 4,steve.jpg,Bill Gates,Tom Anderson  
Scott Wozniak,hey all,scott here,scott the woz.jpg,butter dog,Jebediah Schlatticus,Tom Anderson  
butter dog,dog with the butter on him,dog with the butter,butter dog.jpg,Tom Anderson  
Jebediah Schlatticus,big man,big men,jschlatt.jpg,Scott Wozniak,butter dog,Tom Anderson  
LowTierGod,Helping others emotionally,Love yourself. You serve all purpose. I mean that 100% 1000%.,lowtiergod.jpg,Jebediah Schlatticus,Tom Anderson  
Tom Anderson,30 male,:-),tom.jpg,Jay Carlos,Bill Gates,Steve Jobs,Scott Wozniak,butter dog,Jebediah Schlatticus,LowTierGod

The first value is the user’s name, the second value is their quote, the third value is their status, and the rest of the values are their friends list.

Then, create a new class UserDatabase. Create an observable list of strings like so:

ObservableList<String> friendList = FXCollections.observableArrayList();

This is where the user’s friends list will be stored when loaded from the file. You will need to be able to access, add, and remove users, as well as get the number of users. First, create two methods addUser() that both return nothing to add users with one accepting only a String parameter name, and the other accepting four String parameters name, quote, status, and profileURL. The former will be used to create a blank user, while the other will be used to add the users from the .txt file into the database. Inside of both, create a new instance of the User class user with name, quote, status, and profileURL as arguments. Then, use the add() method in lists to add user to the end of userList. However, in the addUser() method that only accepts the name, you can hardcode a quote, status, and path to a placeholder profile picture for blank users. Next, create a removeUser() method that returns nothing and accepts a User parameter user. In it, use remove() method in lists to remove user from the list. Then, create a getUser() method that returns User and accepts an integer parameter index. Use the get() method in lists to return the User object at the given index in userList. Then, create a getSizeOfDatabase() method that returns an integer. You can use the size() method in lists to return the size of userList.

Loading the file is the last method that you will create in your UserDatabase class, and you will be ready to interact with your database. Create a method loadDatabase() that returns nothing and accepts the String parameter filename. Inside this method, create a try-catch block. Inside this block, create a new instance of a FileReader fileReader, a new instance of a Scanner text, a String array userInfo with null as its value, and an integer lineCount with a value of 0. Then, create a while loop that loops through the entire file by setting text has characters to scan, like so:

while (text.hasNext()) {

}

Inside the loop, create a String line that stores the value of the next unread line. Remember that users are stored in the file line-by-line, with their info separated by commas. Then, split the line by commas and save the tokens to userInfo[]. You can use Java’s built-in split() method for strings and pass the comma as an argument to split the line. Then, use the addUser() method you created, passing userInfo[0], userInfo[1], userInfo[2], and userInfo[3] as arguments. Remember that addUser() can accept a name, status, quote, and profile picture path as arguments, which is what you are doing here. Then, create an if statement that checks if the length of userInfo is greater than 4. You can do this by putting a dot operator and the keyword length after userInfo. Inside, create a for loop that uses the loop variable *i*, which starts at 4 and continues while *I* is less than the length of userInfo. It should increment *i* by 1 each iteration. Remember that the first four values of userInfo are always: name, status, quote, and profile picture path. Whatever values come next compose the user’s friends list, which is why a loop with a check is necessary; a user could have 0 friends, 100, or any number in between. Inside the loop, create a User currentUser whose value is the user at the current position lineCount in the database. You can use the getUser() method you made earlier, with lineCount as an argument to do so. Then, create a String friend whose value is the *i*th element in userInfo. Then, check if friend is not null and use the User’s addFriend() method, with friend passed as an argument, to add friend to their friends list. The null check is done to ensure that the user can only add a friend if that friend is also in the database. At the end of the while loop, be sure to increment lineCount by 1. The catch part of the block should handle a FileNotFoundException e and print an error message like “File does not exist.”. This part of the code will run if the database file cannot be read or found.

Returning to the SocialNetController class, you can now create the updateView() method, that returns nothing and accepts an integer parameter index. This method refreshes the onscreen labels with new information from the user at the position index in the database. It can therefore be used to display a different profile or update the currently viewed one. Inside the method, create a String observable array initialFriendList like so:

ObservableList<String> initialFriendList = FXCollections.observableArrayList();

Then, create a for loop that runs from 0 to the friend count that adds the user at the position index’s *i*th friend in their file’s friend list. The friend list in SocialNetController.java makes the user’s friends list accessible to the user interface code, and is different from the list in the file, as that is predefined by your .txt file. You can use the database’s getUser() method, with index as an argument, chained with the User’s getFriend() method, with i as an argument. After the loop, you use the setText() method to update the name, status, and quote text. This method accepts a String argument. Remember that you added the fx:ids of the items you wanted to interact with in your code earlier. ImageViews have a similar setImage() method that accepts an Image as an argument (not a path to an image). You will have to create a new instance of an Image with the path as an argument inside the setImage() method. Then, use the ListView’s setItems() method, given initialFriendList as an argument, to update the friends list. Lastly, set the value of currentUserIndex to index. This makes it so that if you choose to now display, say, user 1’s info on the screen, the current user’s index is now correctly reported as being 1. Moving back to the initialize() method, load the database by using your loadDatabase() method, given the relative path to your .txt file as an argument. Then, call the updateView() method, giving it 0 as an argument to point to the first user.

Then, create a method updateAppStatus() that returns nothing and accepts a String parameter status. All it does is it updates the label in the bottom right of the window. It uses the setText() method in appStatus with status as an argument. Then, create a method searchForMatch() that accepts a String parameter query and returns an integer. Inside is an if statement that checks if query is not null and iterates through each user in the database to check if it matches the query and returns the index of that user. The null check ensures that the search field is not blank. If the query is null, return -1 (this is not a valid index). To implement this, create a for loop with the loop variable *i* that starts at 0, continues while *i* is less than the size of the database, and *i* is incremented by 1 each iteration. Inside is an if statement that checks for the following condition:

database.getUser(i).getName().toUperCase().equals(query.toUpperCase())

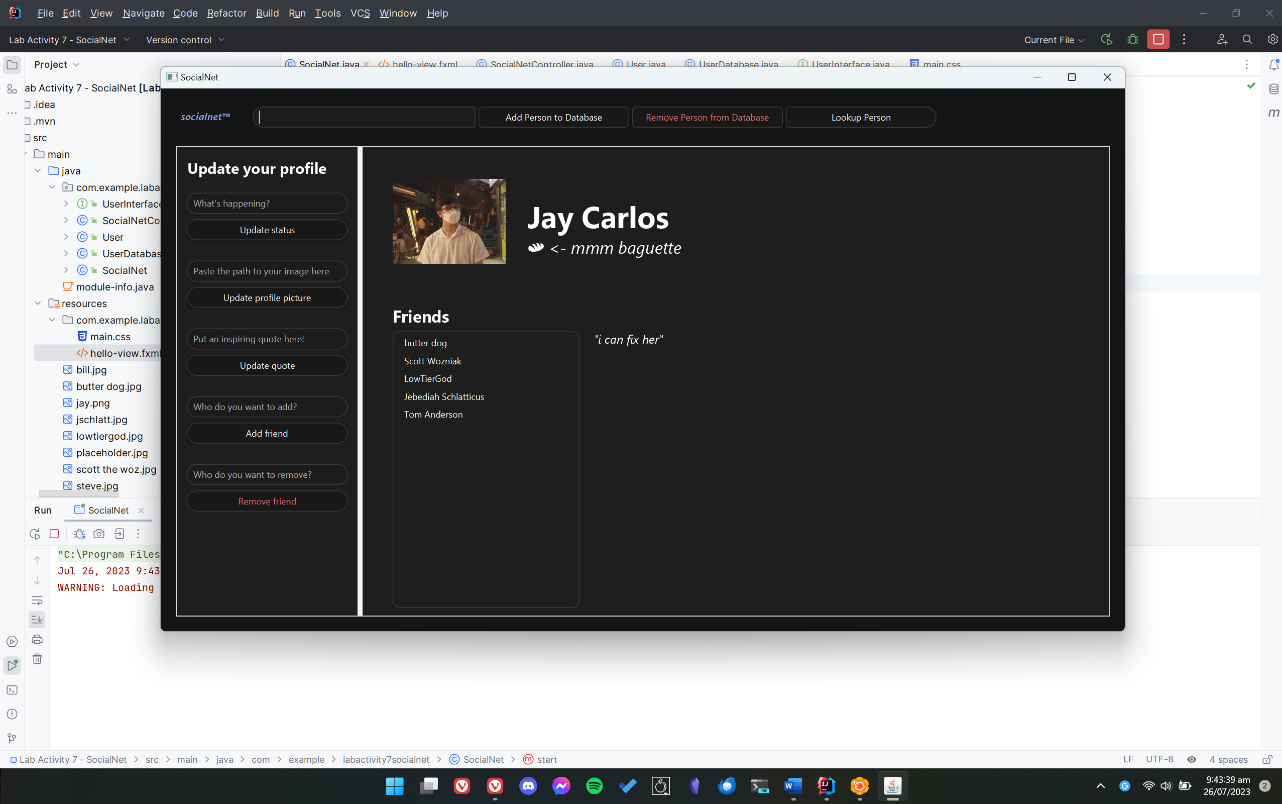
toUpperCase() is a String method that changes all the letters in the string to capital letters. equals() is a method that checks if two objects are of the same type. It then returns *i* if the condition is met. Then, create a method getCurrentUserIndex() that returns the currentUserIndex as an integer.

The last eight methods in SocialNetController are ones that respond to each of the buttons: add person to database, remove person from database, lookup person from database, update status, update profile picture, update quote, add friend, and remove friend. The database buttons follow a similar format, so start with those. The remove from database button should have an integer indexToRemove that uses the searchForMatch() method with the field’s text (which you can get with the getText() method) as an argument. Then, create an if-else block that checks if indexToRemove is not -1 (remember that searchForMatch() returns -1 if no match is found) and use updateAppStatus() to display a message showing the app’s status. Then, use the database’s removeUser() method to remove the user from the database. Next, for the add to database method, create a String that stores the value of the field and checks if the value does not exist in the database. You can use searchForMatch() and check if it returns a -1. Use addUser() given the field’s text as an argument. Then, use updateAppStatus(). Next, for the lookup in database method, create an integer indexToRemove that stores the value of the field’s text. Then, create an if-else block that checks if indexToRemove is not -1, calls updateAppStatus() and updateView() with the index of the user in the query. In all of these, if the condition is not met, call the updateAppStatus() method with an error message as an argument. Next, for the name, status, and profile picture update buttons, create a String that stores the value of the field. Then, check if the field text is not null (the user has entered something in the field) and it calls the database’s getUser() method and the appropriate User setter method with the field’s text as the argument. Then, call updateView() with the current user’s index as the argument. If the field text is null, call updateAppStatus() with an error message string as an argument. Next, for the add friend button, create a string that stores the value of the field text, then check if the text is not null, that the person does not exist in the database, and that the friends list does not already contain that person. The friends list has a contains() method that you can use. If all these conditions are met, call the user’s addFriend() method, then call the updateAppStatus() and updateView() methods. Next, for the remove friend button, create a string that stores the value of the field, then check if the field is not null, that the user exists in the database, and that the user’s friends list contains the text in the field. Then, use the user’s removeFriend() method with the value of the field as an argument. Then call the updateAppStatus() and updateView() methods. Lastly, when these methods do not meet the conditions, display an error message through updateAppStatus().

Next, go back to SceneBuilder. Click each of the interactable elements, go to the “Code” section on the right, and attach the appropriate method to each of the labels and buttons. Lastly, you could create a .css file in your resources folder, next to your .fxml file, and style each of the elements in your scene. You can use any class names you want; just make sure the appropriate stylesheet is applied and the element is given the appropriate class/es in the “Inspector” section.

# RESULTS AND DISCUSSION

1. Laboratory Activity 1: "SocialNet” Result:



**Explanation**:

There are seven main files that you need to interact with in making your project:

* User.java
* UserInterface.java
* UserDatabase.java
* userDatabase.txt
* SocialNet.java
* SocialNetController.java
* main.css

hello-view.fxml is also a crucial file, but you mainly interact with it through SceneBuilder. User.java implements UserInterface and contains the user’s info, a constructor that sets the user’s info upon the creation of the object, and getter and setter methods. UserInterface defines the getter and setter methods required for the User class. UserDatabase defines classes that allow you to interact with the database. loadDatabase() gets the data from the userDatabase.txt file and creates User objects. addUser() creates the User object given parameters. removeUser() removes a specified User object. getUser() gets the user at the *i*th index in the database. getSizeOfDatabase() gets the number of users in the database. userDatabase.txt contains the users on each line, with their info separated by commas. SocialNet initializes the app. SocialNetController contains the interactive elements in the app. initialize() loads the database and displays the first user onscreen. updateAppStatus() changes the text in the label that shows the user the most recent action done. searchForMatch() searches the database for a username that matches the text the user entered in the search field. updateView() updates the on-screen graphics. onRemoveDatabaseButtonPressed(), onLookupDatabaseButtonPressed(), onAddPersonDatabaseButtonPressed(), onUpdatestatusButtonPressed(), onUpdateProfilePictureButtonPressed(), onUpdateQuoteButtonPressed(), onAddFriendButtonPressed(), and onRemoveFriendButtonPressed() respond to the button presses and do the appropriate action.

# CONCLUSION:

I achieved all my objectives for the module. Making an interface in JavaFX was achieved through the SceneBuilder drag-and-drop UI. A graphical program was developed in JavaFX to recreate a social media site. Lastly, CSS styling was implemented at the end of the development process. I have learned how to make a GUI app in Java, and how more complex apps can be structured. Attitude-wise, I learned to be more patient when it comes to modeling large, complex ideas; this is necessary for apps such as these that have more moving parts. The most common pitfalls I encountered were returning wrong types and being unable to imagine the interactions and connections between objects in the app. I overcame both by visualizing the connections with PlantUML, as well as taking time off coding to really understand how I want the app to work. My main recommendation is to go in with a detailed plan for how your app should work, how objects should interact, what data will be passed and where, and what methods you will need to achieve this. Take note of the types required to avoid confusion. After you model the app, implementing it becomes much easier.

# REFERENCES

1. Gosling, J., Joy, B., Steele, G., Bracha, G., & Buckley, A. (2015). The Java language specification. Oracle America, Inc.
2. @nishkarshgandhi. (2021). JavaFX Tutorial - GeeksforGeeks. https://www.geeksforgeeks.org/javafx-tutorial/
3. Roberts, E. (2005). Karel the robot learns java. Department of Computer Science Stanford University.
4. Roberts, E. (2008). The Art & Science of Java. Pearson.
5. Oracle. Java Documentation. ( https://docs.oracle.com/en/java )
6. Schildt, H. (2017). Java: A Beginner's Guide, 7th Edition. McGraw-Hill Education.
7. Troccoli, N. (2017). CS 106A: Assignment #1: Karel the Robot. Stanford University
8. UML basics: The component diagram. http://www.ibm.com/developerworks/rational/library/dec04/bell/

# APPENDIX

1. SocialNet.java

package com.example.labactivity7socialnet;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
  
public class SocialNet extends Application {  
 @Override  
 public void start(Stage stage) throws IOException {  
 FXMLLoader fxmlLoader = new FXMLLoader(SocialNet.class.getResource("hello-view.fxml"));  
 Scene scene = new Scene(fxmlLoader.load(), 1280, 720);  
 stage.setTitle("SocialNet");  
 stage.setScene(scene);  
 stage.show();  
 }  
  
 public static void main(String[] args) {  
 launch();  
 }  
}

2. SocialNetController.java

package com.example.labactivity7socialnet;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.Label;  
import javafx.scene.control.ListView;  
import javafx.scene.control.TextField;  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
import javafx.scene.text.Text;  
  
public class SocialNetController {  
 *// tracks the index of the user being viewed onscreen at any given moment.* int currentUserIndex = 0;  
 @FXML  
 UserDatabase database = new UserDatabase();  
 @FXML  
 private Text userName;  
 @FXML  
 private Text userStatus;  
 @FXML  
 private Text userQuote;  
 @FXML  
 private ImageView userProfilePicture;  
 @FXML  
 private ListView<String> friendList;  
 @FXML  
 private TextField personLookupField;  
 @FXML  
 private Label appStatus;  
 @FXML  
 private TextField statusField;  
 @FXML  
 private TextField profilePictureField;  
 @FXML  
 private TextField quoteField;  
 @FXML  
 private TextField addFriendField;  
 @FXML  
 private TextField removeFriendField;  
  
 @FXML  
 private void initialize() {  
 database.loadDatabase("src/main/resources/userDatabase.txt");  
  
 updateView(0);  
 }  
  
 *// update the status label at the bottom right.* @FXML  
 private void updateAppStatus(String status) {  
 appStatus.setText(status);  
 }  
  
 *// search for a match in the database based on the text entered in lookupField. returns the index of the matched user, and -1 if no match is found.* @FXML  
 private int searchForMatch(String query) {  
 if (query != null) {  
 for (int i = 0; i < database.getSizeOfDatabase(); i++) {  
 if (database.getUser(i).getName().toUpperCase().equals(query.toUpperCase())) {  
 return i;  
 }  
 }  
 }  
  
 return -1;  
 }  
  
 *// update the on-screen graphics.* @FXML  
 private void updateView(int index) {  
 ObservableList<String> initialFriendList = FXCollections.observableArrayList();  
 *// loop through each of the initial user's friends and save all of their friend's names to the observable list.* for (int i = 0; i < database.getUser(index).getFriendCount(); i++) {  
 initialFriendList.add(database.getUser(index).getFriend(i));  
 }  
  
 userName.setText(database.getUser(index).getName());  
 userStatus.setText(database.getUser(index).getStatus());  
 userQuote.setText("\"" + database.getUser(index).getQuote() + "\"");  
 userProfilePicture.setImage(new Image(database.getUser(index).getProfileURL()));  
 friendList.setItems(initialFriendList);  
  
 currentUserIndex = index;  
 }  
  
 *// remove a person from the database when the button is pressed.* @FXML  
 private void onRemoveDatabaseButtonPressed() {  
 int indexToRemove = searchForMatch(personLookupField.getText());  
 if (indexToRemove != -1) {  
 updateAppStatus("Removed user " + database.getUser(indexToRemove).getName() + " from the database.");  
 database.removeUser(database.getUser(searchForMatch(personLookupField.getText())));  
 updateView(0);  
 } else {  
 updateAppStatus("No profiles found with that name. Check for any spelling mistakes.");  
 }  
 }  
  
 *// search for a person in the database when the button is pressed.* @FXML  
 private void onLookupDatabaseButtonPressed() {  
 int indexToRemove = searchForMatch(personLookupField.getText());  
 if (indexToRemove != -1) {  
 updateAppStatus("Viewing " + database.getUser(searchForMatch(personLookupField.getText())).getName() + "'s profile.");  
 updateView(searchForMatch(personLookupField.getText()));  
 } else {  
 updateAppStatus("No profiles found with that name. Check for any spelling mistakes.");  
 }  
 }  
  
 *// return the current user's index in the database.* @FXML  
 private int getCurrentUserIndex() {  
 return currentUserIndex;  
 }  
  
 *// add a person to the database when the button is pressed.* @FXML  
 private void onAddPersonDatabaseButtonPressed() {  
 String lookupField = personLookupField.getText();  
 if (searchForMatch(lookupField) == -1) {  
 database.addUser(lookupField);  
 updateAppStatus("User " + lookupField + " was added to the database.");  
 } else {  
 updateAppStatus("The user was not added. A user already exists with that name.");  
 }  
 }  
  
 *// update status* @FXML  
 private void onUpdateStatusButtonPressed() {  
 String statusFieldText = statusField.getText();  
 if (statusFieldText != null) {  
 database.getUser(getCurrentUserIndex()).setStatus(statusFieldText);  
 updateAppStatus("User " + database.getUser(getCurrentUserIndex()).getName() + "'s status was updated.");  
 updateView(getCurrentUserIndex());  
 } else {  
 updateAppStatus("Type something to update your status.");  
 }  
 }  
  
 *// update profile picture* @FXML  
 private void onUpdateProfilePictureButtonPressed() {  
 String profilePictureFieldText = profilePictureField.getText();  
 if (profilePictureFieldText != null) {  
 database.getUser(getCurrentUserIndex()).setProfileURL(profilePictureFieldText);  
 updateAppStatus("User " + database.getUser(getCurrentUserIndex()).getName() + "'s profile picture was updated.");  
 updateView(getCurrentUserIndex());  
 } else {  
 updateAppStatus("Type a valid path to an image you want to set as your profile picture.");  
 }  
 }  
  
 *// update the current user's quote.* @FXML  
 private void onUpdateQuoteButtonPressed() {  
 String quoteFieldText = quoteField.getText();  
 if (quoteFieldText != null) {  
 database.getUser(getCurrentUserIndex()).setQuote(quoteFieldText);  
 updateAppStatus("User " + database.getUser(getCurrentUserIndex()).getName() + "'s quote was updated.");  
 updateView(getCurrentUserIndex());  
 } else {  
 updateAppStatus("Type something to change the quote displayed on your profile.");  
 }  
 }  
  
 *// add a person to the current user's friends list.* @FXML  
 private void onAddFriendButtonPressed() {  
 String addFriendFieldText = addFriendField.getText();  
 *// if the field isn't empty, the person exists in the database, and isn't already in the current user's friends list, add them.* if (addFriendFieldText != null && searchForMatch(addFriendFieldText) != -1 && !database.getUser(getCurrentUserIndex()).friends.contains(addFriendFieldText)) {  
 database.getUser(getCurrentUserIndex()).addFriend(addFriendFieldText);  
 updateAppStatus("User " + addFriendFieldText + " has been added to " + database.getUser(getCurrentUserIndex()).getName() + "'s friend list.");  
 updateView(getCurrentUserIndex());  
 } else {  
 updateAppStatus("Could not add friend. This user is not in the database.");  
 }  
 }  
  
 *// remove a person from the current user's friends list.* @FXML  
 private void onRemoveFriendButtonPressed() {  
 String removeFriendFieldText = removeFriendField.getText();  
 *// similar deal to the method where you add a friend, but it checks if the name is in the user's friends list before removing it.* if (removeFriendFieldText != null && searchForMatch(removeFriendFieldText) != -1 && database.getUser(getCurrentUserIndex()).friends.contains(removeFriendFieldText)) {  
 database.getUser(getCurrentUserIndex()).removeFriend(removeFriendFieldText);  
 updateAppStatus("User " + removeFriendFieldText + " has been removed from " + database.getUser(getCurrentUserIndex()).getName() + "'s friend list.");  
 updateView(getCurrentUserIndex());  
 } else {  
 updateAppStatus("Could not remove friend. They are not in your friends list.");  
 }  
 }  
}

3. User.java

package com.example.labactivity7socialnet;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
  
import java.util.ArrayList;  
import java.util.Objects;  
  
public class User implements UserInterface {  
 UserDatabase database = new UserDatabase();  
 protected String name;  
 protected String quote;  
 protected String status;  
 protected String profileURL;  
 protected ObservableList<String> friends = FXCollections.*observableArrayList*();  
  
 public User(String name) {  
 this.name = name;  
 this.quote = "";  
 this.status = "";  
 this.profileURL = "placeholder.png";  
 this.friends = FXCollections.*emptyObservableList*();  
 }  
  
 public User(String name, String quote, String status, String profileURL) {  
 this.name = name;  
 this.quote = quote;  
 this.status = status;  
 this.profileURL = profileURL;  
 }  
  
 public String getName() { return name; }  
 public void setName(String name) { this.name = name; }  
 public String getQuote() { return quote; }  
 public void setQuote(String quote) { this.quote = quote; }  
 public String getStatus() { return status; }  
 public void setStatus(String status) { this.status = status; }  
 public String getProfileURL() { return profileURL; }  
 public void setProfileURL(String profileURL) { this.profileURL = profileURL; }  
 public ObservableList<String> getFriends() { return friends; }  
 public void setFriends(ObservableList<String> friends) { this.friends = friends; }  
 public void addFriend(String friend) { this.friends.add(friend); }  
 public void removeFriend(String friend) { this.friends.remove(friend); }  
 public String getFriend(int index) { return this.friends.get(index); }  
 public int getFriendCount() { return this.friends.size(); }  
}

4. resources/UserDatabase.txt

Jay Carlos,i can fix her,🍞 <- mmm baguette,jay.png,butter dog,Scott Wozniak,LowTierGod,Jebediah Schlatticus,Tom Anderson  
Bill Gates,Success is a lousy teacher. It seduces smart people into thinking they can't lose.,Bill Gates is speaking...,bill.jpg,Steve Jobs,Tom Anderson  
Steve Jobs,Beating you is Apple II easy.,Launching the iPhone 4,steve.jpg,Bill Gates,Tom Anderson  
Scott Wozniak,hey all,scott here,scott the woz.jpg,butter dog,Jebediah Schlatticus,Tom Anderson  
butter dog,dog with the butter on him,dog with the butter,butter dog.jpg,Tom Anderson  
Jebediah Schlatticus,big man,big men,jschlatt.jpg,Scott Wozniak,butter dog,Tom Anderson  
LowTierGod,Helping others emotionally,Love yourself. You serve all purpose. I mean that 100% 1000%.,lowtiergod.jpg,Jebediah Schlatticus,Tom Anderson  
Tom Anderson,30 male,:-),tom.jpg,Jay Carlos,Bill Gates,Steve Jobs,Scott Wozniak,butter dog,Jebediah Schlatticus,LowTierGod

5. UserDatabase.java

package com.example.labactivity7socialnet;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import java.io.FileNotFoundException;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.Objects;  
import java.util.Scanner;  
  
public class UserDatabase {  
 ObservableList<User> userList = FXCollections.*observableArrayList*();  
  
 *// load the csv.* public void loadDatabase(String filename) {  
 try {  
 FileReader fileReader = new FileReader(filename);  
 Scanner text = new Scanner(fileReader);  
 String[] userInfo = null;  
 int lineCount = 0;  
  
 *// loop through csv file.* while (text.hasNext()) {  
 String line = text.nextLine();  
  
 userInfo = line.split(",");  
  
 *// create user object and add to list with name, status, quote, and profile picture path.* addUser(userInfo[0], userInfo[1], userInfo[2], userInfo[3]);  
  
 if (userInfo.length > 4) {  
 for (int i = 4; i < userInfo.length; i++) {  
 *// set current user to user object at index lineCount.* User currentUser = getUser(lineCount);  
  
 *// search for user object with matching name and add it to the current user's friend list.* String friend = userInfo[i];  
 if (friend != null) {  
 currentUser.addFriend(friend);  
 }  
 }  
 lineCount++;  
 }  
 }  
 } catch (FileNotFoundException e) {  
 System.*out*.println("File does not exist");  
 }  
 }  
  
 public void addUser(String name, String quote, String status, String profileURL) {  
 User user = new User(name, quote, status, profileURL);  
 userList.add(user);  
 }  
  
 public void addUser(String name) {  
 User user = new User(name, "Insert a quote here.", "Type your status here.", "placeholder.jpg");  
 userList.add(user);  
 }  
  
 *// remove a user from the database.* public void removeUser(User user) { userList.remove(user); }  
 *// get the user at a specified index.* public User getUser(int index) { return userList.get(index); }  
 *// get the user with the specified name.* public User getUser(String userName) {  
 for (User user : userList) {  
 if (user.getName().equals(userName)) {  
 return user;  
 }  
 }  
 return null;  
 }  
  
 *// get the index of a specified user.* public int getIndex(User user) { return userList.indexOf(user); }  
 *// get the size of the user database.* public int getSizeOfDatabase() { return userList.size(); }  
}

6. UserInterface.java

package com.example.labactivity7socialnet;  
  
import javafx.collections.ObservableList;  
  
import java.util.ArrayList;  
  
public interface UserInterface {  
 public String getName();  
 public void setName(String name);  
 public String getQuote();  
 public void setQuote(String quote);  
 public String getStatus();  
 public void setStatus(String status);  
 public String getProfileURL();  
 public void setProfileURL(String profileURL);  
 public ObservableList<String> getFriends();  
 public void setFriends(ObservableList<String> friends);  
 public void addFriend(String friend);  
 public void removeFriend(String friend);  
 public String getFriend(int index);  
 public int getFriendCount();  
}

6. resources/com.example.labactivity7socialnet/hello-view.fxml

*<?*xml version="1.0" encoding="UTF-8"*?>  
  
<?*import java.lang.String*?>  
<?*import javafx.geometry.Insets*?>  
<?*import javafx.scene.Cursor*?>  
<?*import javafx.scene.control.Button*?>  
<?*import javafx.scene.control.Label*?>  
<?*import javafx.scene.control.ListView*?>  
<?*import javafx.scene.control.SplitPane*?>  
<?*import javafx.scene.control.TextField*?>  
<?*import javafx.scene.control.ToolBar*?>  
<?*import javafx.scene.image.Image*?>  
<?*import javafx.scene.image.ImageView*?>  
<?*import javafx.scene.layout.AnchorPane*?>  
<?*import javafx.scene.layout.VBox*?>  
<?*import javafx.scene.text.Font*?>  
<?*import javafx.scene.text.Text*?>*<VBox alignment="CENTER" prefHeight="720.0" prefWidth="1280.0" spacing="20.0" style="-fx-background-color: rgb(20, 20, 20);" xmlns="http://javafx.com/javafx/20.0.1" xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.example.labactivity7socialnet.SocialNetController">  
 <padding>  
 <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />  
 </padding>  
 <children>  
 <ToolBar style="-fx-background-color: rgb(20, 20, 20);">  
 <items>  
 <Label prefHeight="17.0" prefWidth="92.0" text="socialnet™" textFill="#7ca2e4">  
 <font>  
 <Font name="Lucida Sans Demibold Italic" size="12.0" />  
 </font>  
 </Label>  
 <TextField fx:id="personLookupField" prefHeight="25.0" prefWidth="296.0" promptText="Search for a person" style="-fx-border-radius: 50px 10px 10px 50px; -fx-background-radius: 50px 10px 10px 50px;" styleClass="text-field" stylesheets="@main.css" />  
 <Button fx:id="addDatabaseButton" mnemonicParsing="false" onAction="#onAddPersonDatabaseButtonPressed" prefHeight="25.0" prefWidth="200.0" styleClass="toolbar-button" stylesheets="@main.css" text="Add Person to Database" textFill="WHITE">  
 <cursor>  
 <Cursor fx:constant="HAND" />  
 </cursor></Button>  
 <Button fx:id="removeDatabaseButton" mnemonicParsing="false" onAction="#onRemoveDatabaseButtonPressed" prefHeight="25.0" prefWidth="200.0" stylesheets="@main.css" text="Remove Person from Database" textFill="#eb7171">  
 <styleClass>  
 <String fx:value="delete-button" />  
 <String fx:value="toolbar-button" />  
 </styleClass></Button>  
 <Button fx:id="lookupDatabaseButton" mnemonicParsing="false" onAction="#onLookupDatabaseButtonPressed" prefHeight="25.0" prefWidth="200.0" style="-fx-border-radius: 10px 50px 50px 10px; -fx-background-radius: 10px 50px 50px 10px;" styleClass="toolbar-button" stylesheets="@main.css" text="Lookup Person" textFill="WHITE" />  
 </items>  
 </ToolBar>  
 <SplitPane dividerPositions="0.29797979797979796" stylesheets="@main.css" VBox.vgrow="ALWAYS">  
 <items>  
 <AnchorPane maxWidth="240.0" minHeight="0.0" minWidth="160.0" prefHeight="623.0" prefWidth="309.0" style="-fx-background-color: rgb(30, 30, 30);" stylesheets="@main.css">  
 <children>  
 <Button fx:id="updateStatusButton" layoutX="13.0" layoutY="96.0" mnemonicParsing="false" onAction="#onUpdateStatusButtonPressed" prefHeight="25.0" prefWidth="214.0" style="-fx-border-radius: 50px;" styleClass="toolbar-button" stylesheets="@main.css" text="Update status" textFill="WHITE" />  
 <TextField fx:id="statusField" layoutX="13.0" layoutY="61.0" prefHeight="25.0" prefWidth="214.0" promptText="What's happening?" styleClass="text-field" stylesheets="@main.css" />  
 <TextField fx:id="profilePictureField" layoutX="13.0" layoutY="151.0" prefHeight="25.0" prefWidth="214.0" promptText="Paste the path to your image here" style="-fx-text-fill: rgb(235, 235, 235);" stylesheets="@main.css" />  
 <Button fx:id="updateProfilePictureButton" layoutX="13.0" layoutY="186.0" mnemonicParsing="false" onAction="#onUpdateProfilePictureButtonPressed" prefHeight="25.0" prefWidth="214.0" style="-fx-border-radius: 50px;" styleClass="toolbar-button" stylesheets="@main.css" text="Update profile picture" textFill="WHITE" />  
 <TextField fx:id="quoteField" layoutX="13.0" layoutY="241.0" prefHeight="25.0" prefWidth="214.0" promptText="Put an inspiring quote here!" style="-fx-text-fill: rgb(235, 235, 235);" styleClass="text-field" stylesheets="@main.css" />  
 <Button fx:id="updateQuoteButton" layoutX="13.0" layoutY="276.0" mnemonicParsing="false" onAction="#onUpdateQuoteButtonPressed" prefHeight="25.0" prefWidth="214.0" style="-fx-border-radius: 50px;" styleClass="toolbar-button" stylesheets="@main.css" text="Update quote" textFill="WHITE" />  
 <TextField fx:id="addFriendField" layoutX="13.0" layoutY="331.0" prefHeight="25.0" prefWidth="214.0" promptText="Who do you want to add?" style="-fx-text-fill: rgb(235, 235, 235);" styleClass="text-field" stylesheets="@main.css" />  
 <Button fx:id="addFriendButton" layoutX="13.0" layoutY="366.0" mnemonicParsing="false" onAction="#onAddFriendButtonPressed" prefHeight="25.0" prefWidth="214.0" style="-fx-border-radius: 50px;" styleClass="toolbar-button" stylesheets="@main.css" text="Add friend" textFill="WHITE" />  
 <TextField fx:id="removeFriendField" layoutX="13.0" layoutY="421.0" prefHeight="25.0" prefWidth="214.0" promptText="Who do you want to remove?" style="-fx-text-fill: rgb(235, 235, 235);" styleClass="text-field" stylesheets="@main.css" />  
 <Button fx:id="removeFriendButton" layoutX="13.0" layoutY="456.0" mnemonicParsing="false" onAction="#onRemoveFriendButtonPressed" prefHeight="25.0" prefWidth="214.0" style="-fx-border-radius: 50px; -fx-background-radius: 50px;" stylesheets="@main.css" text="Remove friend" textFill="WHITE">  
 <styleClass>  
 <String fx:value="toolbar-button" />  
 <String fx:value="delete-button" />  
 </styleClass></Button>  
 <Text fx:id="userName1" fill="WHITE" layoutX="14.0" layoutY="36.0" strokeType="OUTSIDE" strokeWidth="0.0" text="Update your profile">  
 <font>  
 <Font name="System Bold" size="20.0" />  
 </font>  
 </Text>  
 </children>  
 <styleClass>  
 <String fx:value="split-pane-divider" />  
 <String fx:value="split-pane" />  
 </styleClass>  
 </AnchorPane>  
 <AnchorPane prefWidth="355.0" style="-fx-background-color: rgb(30, 30, 30);" stylesheets="@main.css">  
 <children>  
 <ImageView fx:id="userProfilePicture" fitHeight="150.0" fitWidth="150.0" layoutX="40.0" layoutY="43.0" pickOnBounds="true" preserveRatio="true">  
 <image>  
 <Image url="@../../../bill.jpg" />  
 </image>  
 </ImageView>  
 <Text fx:id="userName" fill="WHITE" layoutX="219.0" layoutY="108.0" strokeType="OUTSIDE" strokeWidth="0.0" text="Bill Gates">  
 <font>  
 <Font name="System Bold" size="40.0" />  
 </font>  
 </Text>  
 <Text fx:id="userStatus" fill="WHITE" layoutX="219.0" layoutY="142.0" strokeType="OUTSIDE" strokeWidth="0.0" text="Bill Gates is speaking..." wrappingWidth="453.806640625">  
 <font>  
 <Font name="System Italic" size="22.0" />  
 </font>  
 </Text>  
 <Text fill="WHITE" layoutX="40.0" layoutY="233.0" strokeType="OUTSIDE" strokeWidth="0.0" text="Friends">  
 <font>  
 <Font name="System Bold" size="22.0" />  
 </font>  
 </Text>  
 <Text fx:id="userQuote" fill="WHITE" layoutX="307.0" layoutY="261.0" strokeType="OUTSIDE" strokeWidth="0.0" text="&quot;Success is a lousy teacher. It seduces smart people into thinking they can't lose.&quot;" wrappingWidth="453.806640625">  
 <font>  
 <Font name="System Italic" size="16.0" />  
 </font>  
 </Text>  
 <ListView fx:id="friendList" layoutX="40.0" layoutY="244.0" prefHeight="368.0" prefWidth="248.0" style="-fx-background-color: rgb(30, 30, 30); -fx-border-color: rgb(60, 60, 60); -fx-border-radius: 10px;" stylesheets="@main.css">  
 <styleClass>  
 <String fx:value="list-cell" />  
 <String fx:value="list-view" />  
 </styleClass></ListView>  
 <Label fx:id="appStatus" alignment="CENTER\_RIGHT" layoutX="621.0" layoutY="582.0" prefHeight="27.0" prefWidth="357.0" textAlignment="RIGHT" textFill="#aeaeae" />  
 </children>  
 <styleClass>  
 <String fx:value="split-pane" />  
 <String fx:value="split-pane-divider" />  
 </styleClass>  
 </AnchorPane>  
 </items>  
 <styleClass>  
 <String fx:value="split-pane" />  
 <String fx:value="split-pane-divider" />  
 </styleClass>  
 </SplitPane>  
 </children>  
</VBox>

7. resources/com.example.labactivity7socialnet/main.css

.list-view {  
 -fx-background-color: rgb(30, 30, 30);  
 -fx-background-radius: 10px;  
}  
  
.list-cell {  
 -fx-background-color: rgb(30, 30, 30);  
 -fx-text-fill: white;  
}  
  
.list-cell:pressed {  
 -fx-background-color: rgb(159, 212, 243);  
 -fx-text-fill: rgb(30, 30, 30);  
}  
  
.toolbar-button {  
 -fx-background-color: rgb(24, 24, 24);  
 -fx-border-color: rgb(60, 60, 60);  
 -fx-border-radius: 5px 5px 5px 5px;  
 -fx-background-radius: 5px;  
}  
  
.toolbar-button:hover {  
 -fx-background-color: rgb(28, 28, 28);  
}  
  
.toolbar-button:pressed {  
 -fx-background-color: rgb(36, 36, 36);  
}  
  
.delete-button {  
 -fx-text-fill: #eb7171;  
}  
  
.delete-button:pressed {  
 -fx-background-color: #eb7171;  
 -fx-text-fill: rgb(24, 24, 24);  
}  
  
.text-field {  
 -fx-background-color: rgb(28, 28, 28);  
 -fx-border-radius: 50px;  
 -fx-background-radius: 50px;  
 -fx-border-color: rgb(60, 60, 60);  
 -fx-text-fill: rgb(235, 235, 235);  
}  
  
.text-field:hover {  
 -fx-background-color: rgb(36, 36, 36);  
}  
  
.text-field:focus {  
 -fx-background-color: rgb(44, 44, 44);  
}  
  
.split-pane > .split-pane-divider {  
 -fx-background-color: transparent;  
}