Faculty of Engineering

Ain shams University

Computer Engineering and Software Systems

Project Proposal Software Engineering CSE231

Team Members: -

Martin Magued Wadie	22P0193
Mohamed Ashraf Mohamed	22P0210
Omar Ahmed Abdelmohsen	22P0218
Abdelrahman Mostafa Ali	22P0150
Seif Aly Othman Fahmy	22P0182
Mark Edward Azziz	22P0181

Presented to: -

Prof. Dr. Mahmoud Khalil

T.A. Mahmoud Sohail

Introduction: -

Social media platforms have become an important part of our lives, allowing us to connect, share, and communicate with others. Our project is to create a social media platform using Java. We will use Object-Oriented Programming (OOP) and JavaFX to make our platform powerful and user-friendly.

Object-Oriented Programming (OOP) is a way of writing software by using "objects." Objects are like building blocks that contain data and actions. OOP helps us organize our code into simple, reusable pieces. For our social media platform, we will create objects for users, posts, comments, and other features. This will make our code easier to manage and extend in the future. We will be using java which is a popular programming language known for being reliable and working on many different devices. It provides many tools and libraries that help us write the core logic, handle data, and manage communication between users. Java's strengths make it a great choice for building a social media platform.

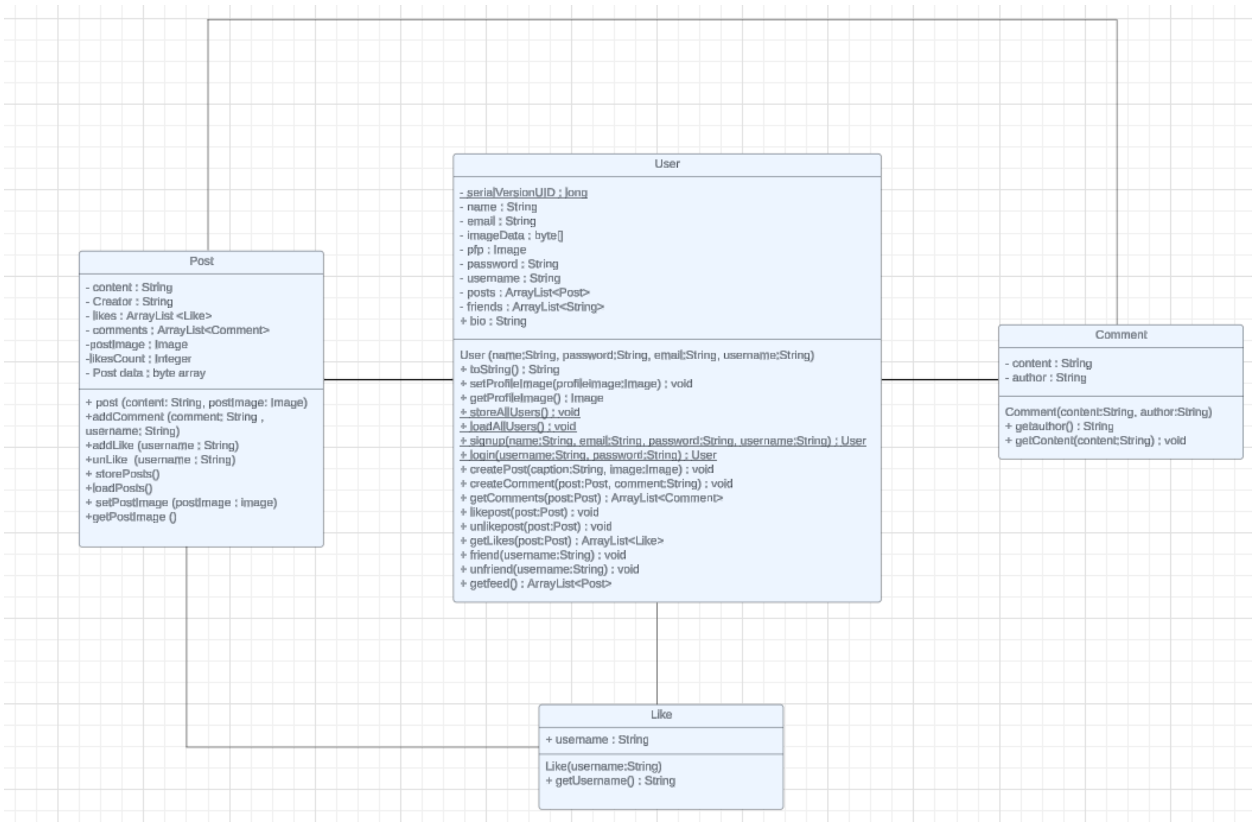
We will also use JavaFX that is a framework used to create rich and interactive user interfaces. It allows us to design and build the visual part of our application, like buttons, menus, and other elements users interact with. With JavaFX, we can create a beautiful and easy-to-use interface for our social media platform, making it more enjoyable for users.

In our project, we will develop key features of a social media platform, such as signing up and logging in, managing user profiles, posting and sharing content and commenting. By combining OOP, Java, and JavaFX, we will build a dynamic and interactive social media application.

Our goal is to create a fully functional social media platform that is not only powerful but also easy to use. We want to showcase the capabilities of Java and JavaFX while following best practices in software development and design.

Table of Contents

Class Diagram: -


User
- serialVersionUID : long
- name : String
- email : String
- imageData : byte[]
- pfp : Image
- password : String
- username : String
- posts : ArrayList<Post>
- friends : ArrayList<String>
+ bio : String
User (name:String, password:String, email:String, username:String)
+ toString() : String
+ setProfileImage(profileimage:Image) : void
+ getProfileImage() : Image
+ storeAllUsers() : void
+ loadAllUsers() : void
+ signup(name:String, email:String, password:String, username:String) : User
+ login(username:String, password:String) : User
+ createPost(caption:String, image:Image) : void
+ createComment(post:Post, comment:String) : void
+ getComments(post:Post) : ArrayList<Comment>
+ likepost(post:Post) : void
+ unlikepost(post:Post) : void
+ getLikes(post:Post) : ArrayList<Like>
+ friend(username:String) : void
+ unfriend(username:String) : void
+ getfeed() : ArrayList<Post>
Post
- content : String
- Creator : String
- likes : ArrayList <Like>
- comments : ArrayList<Comment>
-postImage : Image
-likesCount : Integer
- Post data : byte array
+ post (content: String, postImage: Image)
+addComment (comment: String , username: String)
+addLike (username : String)
+unLike (username : String)
+ storePosts()
+loadPosts()
+ setPostImage (postImage : image)
+getPostImage ()
Comment
- content : String
- author : String
Comment(content:String, author:String)
+ getauthor() : String
+ getContent(content:String) : void
Like
+ username : String
Like(username:String)
+ getUsername() : String

Classes: -

User: -

This class represents a user in the social media application. It stores various details about the user such as their name, email, username, password, profile picture, bio, and lists of posts and friends.

Serializable Interface: By implementing the Serializable interface, the User objects can be converted into a stream of bytes. This allows them to be easily saved to a file (users.ser) and later restored, effectively persisting user data between sessions.

- **name**, **email**, **username**, **password**: Basic information about the user.
- **imageData**: Stores the profile picture of the user as a byte array.
- **Pfp**: Represents the profile picture as an Image object. It is marked as transient, meaning it won't be directly serialized with the object but can be reconstructed from imageData.
- **bio**: User's biography or description.
- **posts**: An ArrayList to store the posts created by the user.
- **friends**: An ArrayList to store the usernames of the user's friends.

Now for the attributes:

- **setProfileImage** (Image profileimage): Sets the profile picture of the user. It converts the Image object into a byte array and stores it in imageData.
- **getProfileImage** (): Retrieves the profile picture of the user. It reconstructs the Image object from imageData.
- **storeAllUsers** (): Saves all users to a file (users.ser) using serialization.
- **loadAllUsers** (): Loads all users from the file (users.ser) into the application.

- **Signup** (): Allows a new user to sign up by providing their name, email, password, and username. Checks if the username is already taken before creating a new user.
- **Login** (): Verifies the credentials (username and password) of a user during the login process.
- **createPost** (String caption, Image image): Creates a new post for the user with a caption and an optional image. Adds the post to the user's list of posts and a global list of posts managed by a singleton class.
- **createComment** (Post post, String comment): Allows the user to create a comment on a specific post.
- **Likepost** (Post post) and **Unlikepost** (Post post): Allows the user to like or unlike a post.
- **Getfeed** (): Retrieves the posts from the user's friends to display in their feed. It iterates through all posts and checks if the creator of each post is a friend of the user.

The next part will be talking about data handling and how we stored any type of data related to the user class.

- User information, including posts and comments, is stored in ArrayLists.
- The storeAllUsers () method saves all users to a file (users.ser) using Java object serialization. This file serves as a persistent data store for user information, allowing it to be loaded back into the application when needed.

This code provides a foundation for a social media application, allowing users to manage their profiles, create posts, interact with posts, and connect with friends.

Post:

This class represents a post made by a user in the social media app. Each post contains information such as the creator's username, content (text), likes, comments, and an optional image.

This class has the following attributes used in other codes:

- **creator**: Represents the username of the user who created the post.
- **content**: Holds the textual content of the post.
- **likes**: An ArrayList to store the likes (represented by Like objects) received by the post.
- **comments**: An ArrayList to store the comments (represented by Comment objects) made on the post.
- **postImage**: Represents the image associated with the post. It's marked as transient, meaning it won't be directly serialized with the object.
- **postData**: Stores the image data as a byte array, allowing it to be serialized along with the post.
- **likesCount**: Tracks the number of likes received by the post.

There are two constructors used in this class. Firstly, Post with arguments String creator, String content and javafx.scene.image.Image postImage. This constructor Initializes a new post with the given creator's username, content, and optionally an image. Also, it initializes the likes and comments lists as empty.

Moreover, some methods were used in the codes of this and other classes:

- **addComment** (String comment, String username): Adds a new comment to the post, including the comment text and the username of the commenter.

- **viewComments** (): Retrieves all comments made on the post.

- **addLike** (String username): Adds a like to the post. It checks if the user has already liked the post before adding the like.

- **unlike** (String username): Removes a like from the post. It checks if the user has previously liked the post before removing the like.

- **storePosts** (): Saves all posts to a file (posts.ser) using object serialization.

- **loadPosts** (): Loads all posts from the file (posts.ser) back into the application.

- **setPostImage** (Image postImage): Sets the image associated with the post. It converts the image into a byte array for serialization.

- **getPostImage** (): Retrieves the image associated with the post by reconstructing it from the serialized byte array.

In order to Store the data of this class we made the following:

- Post information, including likes, comments, and images, is stored in ArrayLists.
- The **storePosts** () method saves all posts to a file (posts.ser) using Java object serialization, allowing them to be loaded back into the application later.

Likes & Comments:

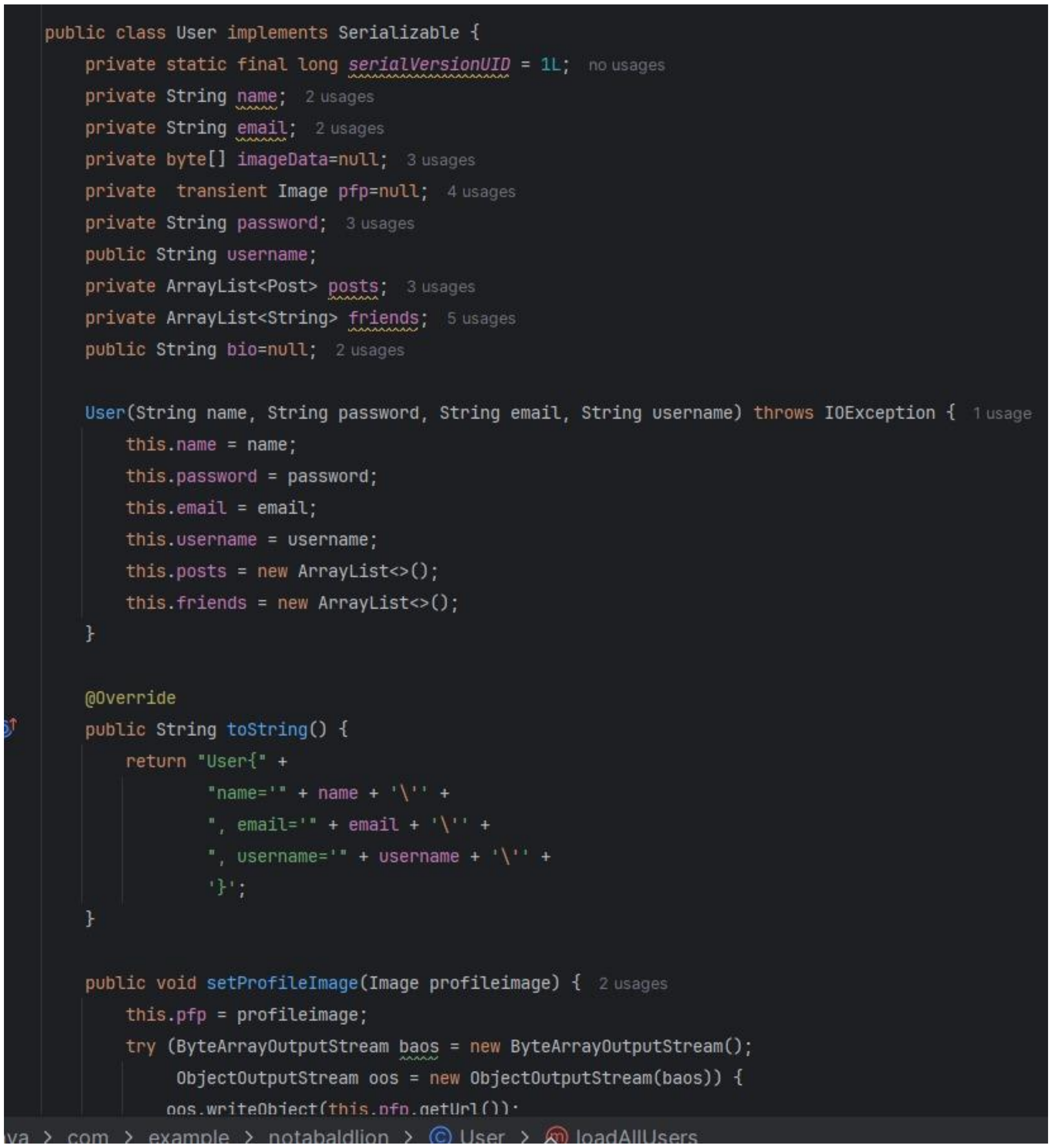
Comment Class

This class represents a comment made by a user on a post. It contains attributes for the content of the comment and the author of the comment. The code itself represents how the system takes in the comments and its author and stores them. Also, it Implements the Serializable interface, allowing instances of the class to be serialized and deserialized.

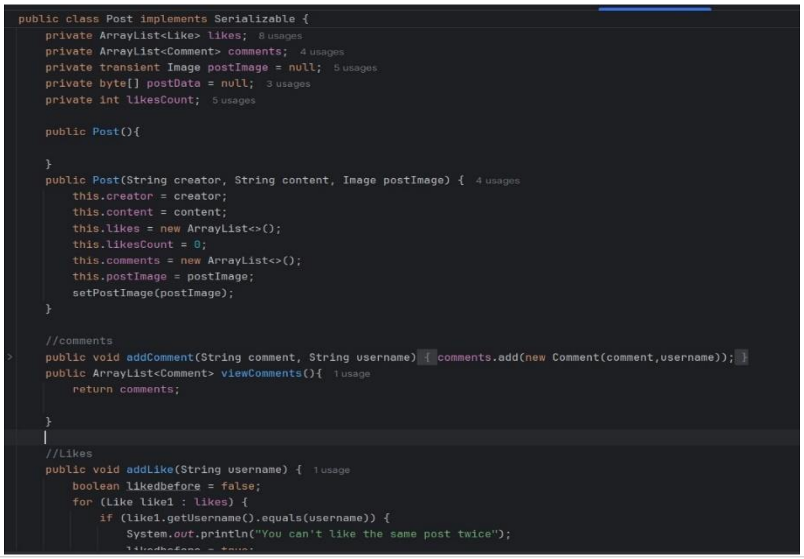
Like Class

This class represents a user's like on a post. It contains an attribute username to store the username of the user who liked the post. Similar to class comment it stores the like on the posts and the performer of the like and Implements the Serializable interface, indicating that instances of the class can be serialized and deserialized.

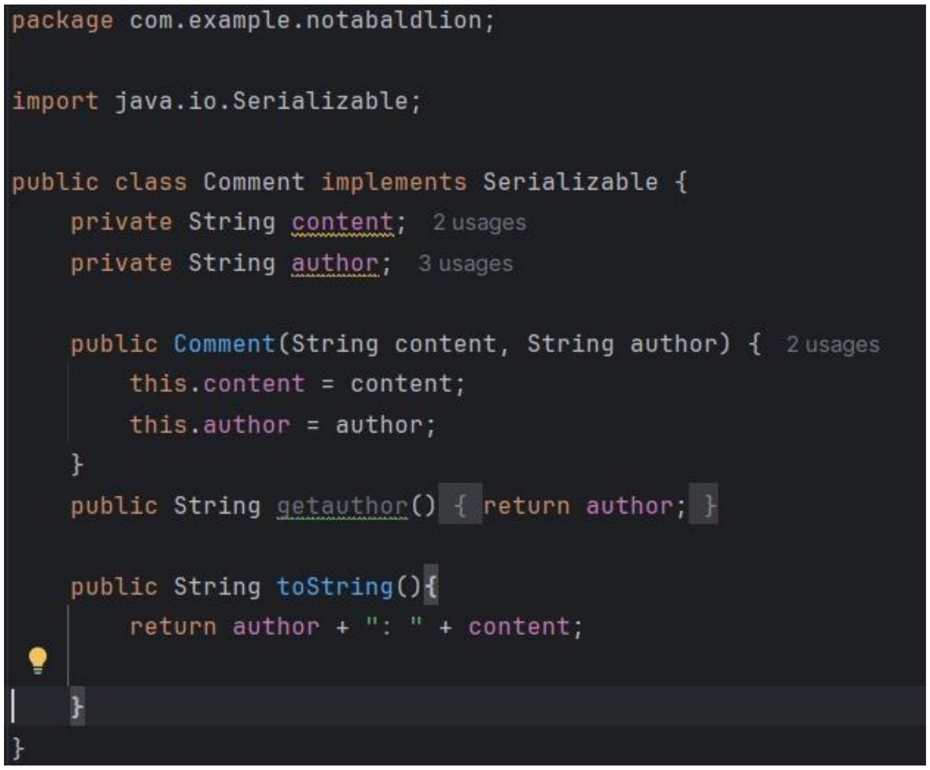
Both classes serve as essential components in a social media application where they both interact with the posts. As, users can engage with posts by either liking them or commenting on them. Finally, each comment or like is associated with the username of the user who performed the action, allowing for attribution and interaction tracking.



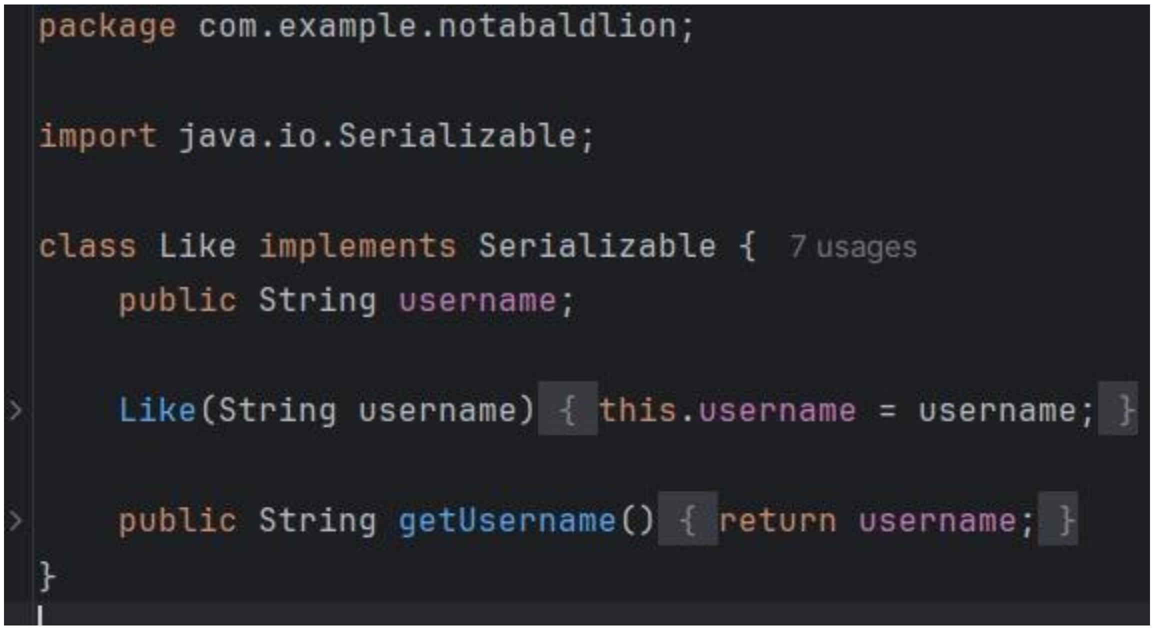
```java
public class User implements Serializable {
    private static final long serialVersionUID = 1L;  no usages
    private String name;  2 usages
    private String email;  2 usages
    private byte[] imageData=null;  3 usages
    private  transient Image pfp=null;  4 usages
    private String password;  3 usages
    public String username;
    private ArrayList<Post> posts;  3 usages
    private ArrayList<String> friends;  5 usages
    public String bio=null;  2 usages

    User(String name, String password, String email, String username) throws IOException {  1 usage
        this.name = name;
        this.password = password;
        this.email = email;
        this.username = username;
        this.posts = new ArrayList<>();
        this.friends = new ArrayList<>();
    }

    @Override
    public String toString() {
        return "User{" +
                "name='" + name + '\'' +
                ", email='" + email + '\'' +
                ", username='" + username + '\'' +
                '}';
    }

    public void setProfileImage(Image profileimage) {  2 usages
        this.pfp = profileimage;
        try (ByteArrayOutputStream baos = new ByteArrayOutputStream();
             ObjectOutputStream oos = new ObjectOutputStream(baos)) {
            oos.writeObject(this.pfp.getUrl());
```



```java
public class Post implements Serializable {
    private ArrayList<Like> likes;  8 usages
    private ArrayList<Comment> comments;  4 usages
    private transient Image postImage = null;  5 usages
    private byte[] postData = null;  3 usages
    private int likesCount;  5 usages

    public Post(){

    }
    public Post(String creator, String content, Image postImage) {  4 usages
        this.creator = creator;
        this.content = content;
        this.likes = new ArrayList<>();
        this.likesCount = 0;
        this.comments = new ArrayList<>();
        this.postImage = postImage;
        setPostImage(postImage);
    }

    //comments
    public void addComment(String comment, String username) { comments.add(new Comment(comment,username)); }
    public ArrayList<Comment> viewComments(){  1 usage
        return comments;

    }

    //Likes
    public void addLike(String username) {  1 usage
        boolean likedbefore = false;
        for (Like like1 : likes) {
            if (like1.getUsername().equals(username)) {
                System.out.println("You can't like the same post twice");
```



```
package com.example.notabaldlion;

import java.io.Serializable;

public class Comment implements Serializable {
    private String content;  2 usages
    private String author;  3 usages

    public Comment(String content, String author) {  2 usages
        this.content = content;
        this.author = author;
    }
    public String getauthor() { return author; }

    public String toString(){
        return author + ": " + content;

    }
}
```



```
package com.example.notabaldlion;

import java.io.Serializable;

class Like implements Serializable {  7 usages
    public String username;

    Like(String username) { this.username = username; }

    public String getUsername() { return username; }
}
```

Controllers: -

Feed Page

Upon initialization, the controller retrieves the current user from the singleton using the singleton.getuser () method. It also sets up the main application reference.

FXML injection is utilized to link various UI elements from the FXML file to the controller. These include buttons for exiting, posting, and navigating to the user's profile, as well as a label for displaying the user's name and an image view for the user's profile picture.

Button actions within the controller serve distinct purposes:

For example, exitButtonAction (ActionEvent event): Triggered upon clicking the "Exit" button, this action stores all user data and posts using the User.storeAllUsers() and Post.storePosts() methods, ensuring that changes are persisted before exiting the application.

Secondly, initialize (): This method is invoked upon initialization of the feed page. It retrieves the user's feed using the controller.getfeed() method, which contains posts from friends. For each post, a corresponding UI element is dynamically generated using FXMLLoader. The post data is set using the setdata () method of the PostController class, and the generated UI element is added to the feed pane.

Third, postbuttonAction (): Executed when the "Post" button is clicked, this action navigates the user to the post page using the main.changeScene() method, enabling users to create new posts.

Lastly, profileButtonAction (ActionEvent event): Launched upon clicking the "Profile" button, this action navigates the user to their profile page using the main.changeScenewithcontroller () method, facilitating seamless access to profile management functionalities.

Comment page:

The CommentsPageController class is designed to handle user interactions with the comments page in a JavaFX application. Upon initialization, it retrieves the current user and creates an instance of the Post class. This allows it to access and manage comments related to a specific post. The controller injects various UI elements from the FXML file, including a ListView to display comments, a "Back" button to navigate back to the feed page, and an "Add Comment" button to allow users to add new comments.

The initialize (Post post) method serves as the entry point for setting up the comments page. It takes a Post object as a parameter to initialize the page with comments related to that specific post. Within this method, it first clears the ListView of any previous comments. Then, it retrieves the comments associated with the provided Post object, converts them to strings, and populates ListView with these comments, allowing users to view existing comments.

For user interactions, the controller defines two button actions: BackToFeed (ActionEvent event) and addCommentAction (ActionEvent event). The BackToFeed method is invoked when the "Back" button is clicked. It uses the changeScenewithcontroller2() method of the main class to navigate back to the feed page, providing users with a way to return to the main content feed.

On the other hand, the addCommentAction method handles the addition of new comments. When the "Add Comment" button is clicked, it retrieves the text entered by the user in the comment_text TextField. It then calls the createComment () method of the User class to add the new comment to the current post. After adding the comment, it refreshes the ListView by clearing it and repopulating it with the updated list of comments, ensuring that the newly added comment is displayed to the user.

Post:

Upon initialization, the controller retrieves the current user using the singleton.getuser() method, enabling access to user data for post-related actions. Through FXML injection, various UI elements such as buttons, labels, and image views are linked to the controller, facilitating seamless interaction with the post UI.

One of the key methods in the controller is setdata(Post post), which populates the post UI with data retrieved from the provided Post object. This method dynamically sets the post creator's username, caption, number of likes, post image, and user image based on the information stored in the Post object.

The controller also defines actions for liking and unliking posts:

The Like (MouseEvent event) action is triggered when the user clicks on the "Like" button (represented by an image view). It hides the "Like" button and shows the "Liked" button, indicating that the user has liked the post. Additionally, it increments the post's like count by calling the likepost () method of the User class.

Conversely, the unlike(MouseEvent event) action is executed when the user clicks on the "Liked" button (also an image view). This action hides the "Liked" button and shows the "Like" button, indicating that the user has unliked the post. It also decreases the posts like count by calling the unlikepost () method of the User class.

Furthermore, the commentbuttonAction (ActionEvent event) method handles user interaction with the "Comment" button. When clicked, this action navigates the user to the comments page (Comments_Page.fxml) while passing the associated Post object. This allows users to view existing comments and add their own comments to the post.

***Post page*:**

The postPagecontroller class is responsible for managing user interactions during the creation of a new post within the JavaFX application. It begins by retrieving the current user instance from the singleton using singleton.getuser(), ensuring that the post is associated with the correct user.

There are three main buttons in the post page, which are:

Back Button Action: The backtofeed (ActionEvent event) method is executed when the user clicks on the "Back" button. This action navigates the user back to the feed page (feed-page.fxml), abandoning the post creation process.

Final Post Button Action: Triggered by clicking the "Post" button, the finalpostButtonAction (ActionEvent event) method handles the finalization of the post creation process. It retrieves the caption entered by the user in the captionlabel text field and ensures that both an image and a caption are provided before creating the post using the createPost () method of the User class. If the conditions are met, the user is directed back to the feed page.

Choose Image Button Action: The chooseimagebuttonaction (ActionEvent event) method is activated when the user clicks on the "Choose Image" button. It opens a file chooser dialog, allowing the user to select an image file (in JPG, JPEG, or PNG format) from their device. Upon selection, the chosen image is loaded into the previewpost image view for the user to preview before finalizing the post.

Profile page:

The profilepageController class manages the profile page functionality within the JavaFX application, facilitating user interactions with their profile information and posts.

Key functionalities of the controller include:

- **Back Button Action**: The backbuttonAction (ActionEvent event) method is triggered when the user clicks on the "Back" button. This action navigates the user back to the feed page (feed-page.fxml), allowing them to return to the main application flow.
- **Edit Button Action**: Activated by clicking the "Edit" button, the editbuttonaction (ActionEvent event) method redirects the user to the editing profile page (editing-profile.fxml). Here, users can modify their profile information such as their bio and profile picture.
- **Friends Button Action**: Upon clicking the "Friends" button, the friendsbuttonaction (ActionEvent event) method loads the friends page (friends-page.fxml). This page displays the user's list of friends, allowing them to manage their social connections within the application.

During initialization, the initialize() method populates the profile page with the user's information, including their username, bio, and profile picture. It retrieves this data from the current user instance obtained from the singleton. Additionally, the method fetches the user's posts and dynamically generates UI elements to display each post in the profile's post container (vboxpostcontainer).

Each post is represented by a separate anchor pane containing the necessary post UI elements, facilitating easy navigation and interaction with the user's posts.

Friends page:

The FriendsPageController class plays a crucial role in managing user interactions with the friends page within the JavaFX application. Upon initialization, the controller retrieves the current user from the singleton using singleton.getuser(), allowing seamless access to user data for friend management.

The controller defines several button actions to handle specific tasks:

- **addfriendbuttonAction** (ActionEvent event): Executed when the "Add Friend" button is clicked, this action retrieves the username entered by the user in the friendusername text field and adds that user as a friend using the friend () method of the User class.
- **removefriendAction** (ActionEvent event): Triggered upon clicking the "Remove Friend" button, this action removes the user entered in the friendusername text field from the friend list using the unfriend () method of the User class.
- **searchbuttonAction** (ActionEvent event): Launched when the "Search" button is clicked, this action retrieves the search query entered by the user in the username text field. It then searches for users whose usernames match the query, displaying the search results in the list view after clearing any previous entries.
- **removefriendbuttonAction** (ActionEvent event): A placeholder method for handling actions related to removing friends. Currently empty and not implemented.

The initialize () method is responsible for populating the list view with usernames of all users available in the application upon initialization of the friends page. This allows users to see all potential friends and manage their connections conveniently.

For search functionality, the searchList (String searchWords, List<String> listOfStrings) method is implemented. This method takes a search query and a list of strings to search within, then returns a list of strings that contain all the search terms entered by the user. It is utilized in the searchbuttonAction () to filter and display search results, enhancing user experience by facilitating efficient user discovery and connection management.

Edit profile:

The editprofileController class is essential for managing user interactions with the profile editing interface in a JavaFX application. Upon initialization, the controller retrieves the current user from the singleton using the singleton.getuser () method, enabling seamless access to the user's profile data for editing.

Button actions within the controller govern specific tasks:

- **backbuttonAction** (ActionEvent event): Triggered upon clicking the "Back" button, this action navigates the user back to the profile scene, ensuring smooth navigation within the application.
- **changingbiobuttonAction** (ActionEvent event): Executed when the "Change Bio" button is pressed, this action retrieves the new bio text entered by the user in the changebio text field. It then updates the user's bio using the setbio () method of the User class, ensuring that profile information is accurately reflected.
- **changingpasswordAction** (ActionEvent event): Activated upon clicking the "Change Password" button, this action retrieves the new password entered by the user in the passwordtextfield text field. Subsequently, it updates the user's password using the setpassword () method of the User class, ensuring secure management of user credentials.

- **changingphoto** (ActionEvent event): Launched when the "Change Photo" button is clicked, this action opens a file chooser dialog using FileChooser, allowing the user to select a new profile picture from their system. Upon selection, the chosen image is loaded into the newpfp ImageView and set as the user's profile picture using the setProfileImage () method of the User class, facilitating seamless updating of profile visuals.

The utilization of a file chooser dialog ensures user-friendly selection of profile images, enhancing the overall user experience. Additionally, the integration of the User class methods ensures that changes made to the user's profile data are accurately reflected and persisted within the application.

Data Storage: -

To store the information inserted into the system we need to store the objects in a file. To serialize the objects.

Serialization is the process of converting objects into a format that can be easily read and written into files.

This format is converting them to a byte array when storing data in memory or a file.ser.Thats why , throughout the code that handles images , an byte array i/o stream is used to make a memory location for an array of bytes and the object i/o stream serializes the objects (make them a byte array) and stores them inside the byte array stream created.To read from the files we need to deserialize them.

However, we use the Javafx.scene.image.Image data type consistently throughout code whether for the users profile image or for the images loaded in posts and that represents an obstacle as these type cannot be serialized.

To fix this issue, the image is declared as "transient". This basically means prevents the attempt of the serialization of the image when storing the object and therefore ease storing the objects with that attribute.

Setting an Image:

Nonetheless, the image wasn't stored anywhere yet, so how is it going to be stored?

To do so we declare 2 more attributes in the class, a byte array and a string of the image URL.

The process of storing images all

happens in the setter for it. This setter takes the image and checks if it's not null. If it is not null, we obtain the URL of the image using getURL () function (from javafx.scene.image.Image). This URL is then serialized and written in the object in the file using a byte array output stream and an object output stream that's linked to it. The byte array representation of the serialized data is then obtained.

Getting an image:

Similarly, this process happens in the getter of the getter of the image. It checks if the image is null first. If its not null, a new byte stream is made set to the byte array representation of the image. An object input stream is then created and linked to 5$ 4-byte array input stream created before.

Then the serialized URL is read using .read object() and casts it to string. A new image is then declared using the URL obtained (the image type in javafx.scene.image.Image provides constructors that accept URL as a parameter)and the desired image is then set to that. It then returns this image.

Singleton class and Main class:

In the singleton class, a new array list of users and posts are declared to be used globally inside the code. These array lists are used as a place to load all the objects from the files as soon as the program starts to be used inside the program. At the end of the program, when the exit button is pressed, the objects in these array lists are then stored inside their respective files.

Loading and Storing objects:

The process of loading and storing happens with serialization as mentioned before. These objects are stored and loaded from .ser files, "Users.ser" and "Posts.ser" for User and Post objects respectively.

Storing:

To store an object , firstly a new array list of the type of object declared and is initialized to the singleton.getters methods which returns the object array list Similarly a new object output stream is made to convert the objects to a byte array, however this time , the objects are stored in a file not memory so a new file output stream is used to write the byte array to the files.

Loading:

A new file input stream is made to read from the file that has the objects stored in. A new object input stream and initialized with the file input stream previously declared to read the serialized objects from the file. The. read object () function then read the serialized objects and deserialize it to return the object. The singleton object array list is then initialized to the. read object function to fill the array list with the objects in the file and then the singleton.setters methods are used to return the array list of objects

Main: -

The main class serves as the entry point for the JavaFX application. It extends the Application class, providing the start method to initialize and display the primary stage of the application.

In the start method, various scenes and their associated controllers are defined and initialized. The scenes include the login page, registration page, feed page, post page, and profile page. These scenes are loaded from their respective FXML files using FXMLLoader.

The changeScene method is a utility function used to switch between scenes within the application. It takes the path to the FXML file of the desired scene as a parameter, loads the scene using FXMLLoader, sets it as the scene of the primary stage, and displays it.

Similarly, the changeScenewithcontroller and The changeScenewithcontroller2 are utility methods for switching scenes while passing a controller object to the new scene. These methods load the scene, retrieve the controller instance using loader.getController (), initialize the controller, and set the scene of the primary stage.

The main class also contains event handlers for various UI components, such as buttons, text fields, and image views. These event handlers are responsible for handling user interactions, such as logging in, registering, navigating between pages, and posting content.

Overall, the main class orchestrates the initialization, navigation, and event handling logic of the JavaFX application, providing a structured and organized approach to building the user interface and managing user interactions.

Bonus Tasks: -

Search bar:

In the friends scene of the application, users are provided with a search functionality to find and add new friends. The search section typically consists of a text field where users can input the username of the person they want to search for. Upon entering a username and clicking the search button, the application retrieves a list of all users from the database. Then, it filters this list based on the entered username, displaying matching results in a list view below the search field. Users can browse through the search results and select the person they want to add as a friend. This search functionality enhances the user experience by enabling efficient discovery and connection with other users within the application's network.

Messaging:

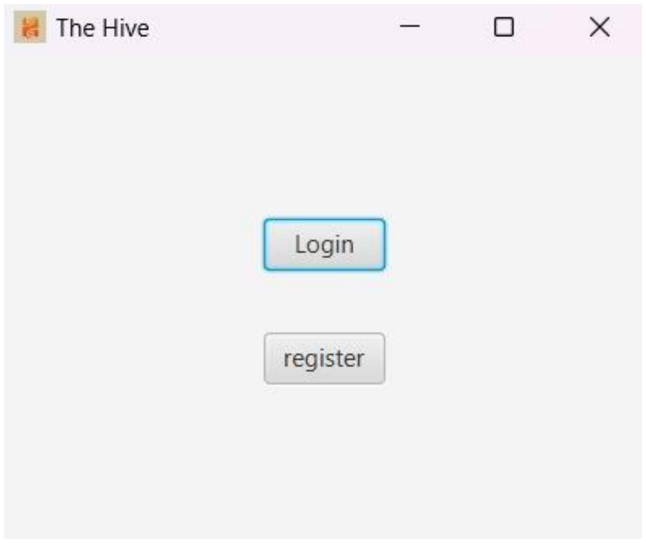
It serves as a simple chat platform with features like sending messages, selecting emojis, and saving chat history. The main class extends to Application, a class provided by JavaFX for creating GUI applications. Within the start method, the UI components are initialized, including a list of users on the left side, a chat history display area in the center, and input fields for typing messages and selecting emojis at the bottom. Each user listed has a button associated with them, and clicking on a user's button opens a chat window specific to that user. The application supports the sending of messages, emoji selection, and saving of chat history to a file. It utilizes event handling to trigger actions such as sending messages or selecting emojis. The code also includes utility methods for managing user data and chat history, such as retrieving usernames and saving/loading chat history from a file. Overall, the code demonstrates the use of JavaFX for building a basic chat application with a user-friendly interface and essential functionality.

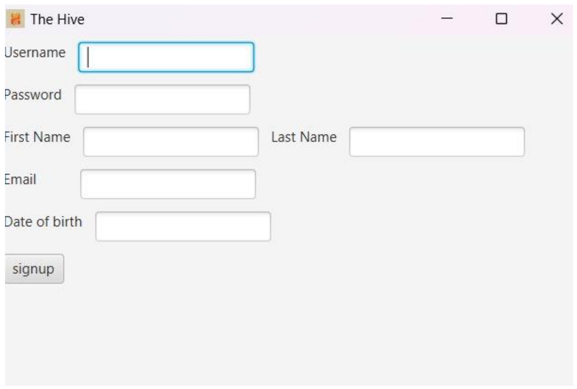
End User guide: -

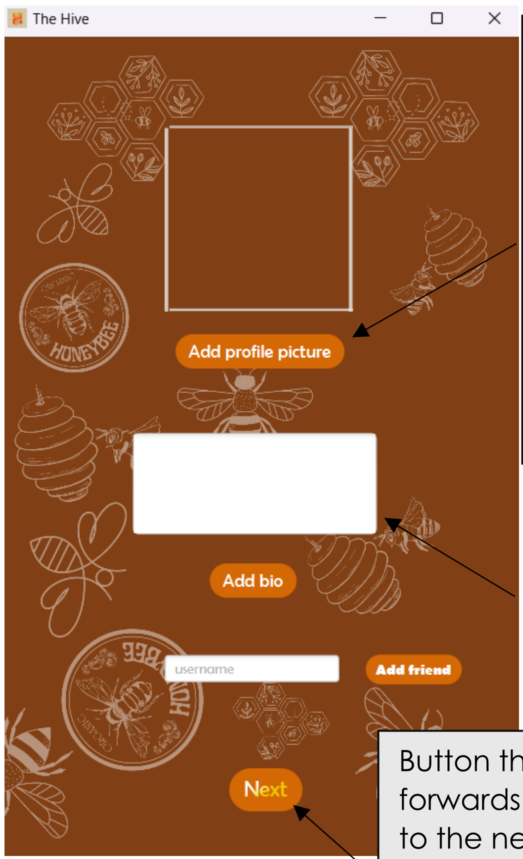
Entry page

This is the entry page that lets you choose between registering if you are new to the application or to login if you already have an account.

This page has only 2 buttons, one for the registration and one for the login.







Button that allows you to choose a photo from your computer to be inserted.

Text field to write your bio

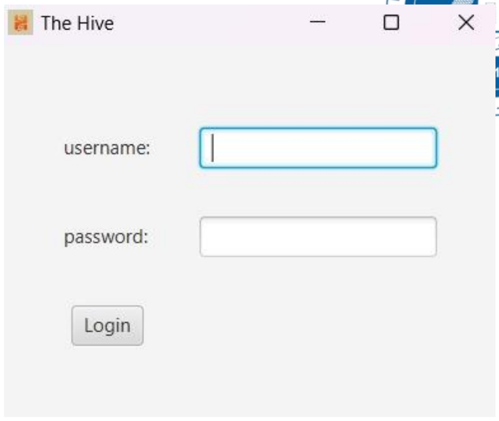
Button that forwards you to the next page.

If you chose to register, then you will be forwarded to another page that will ask you to input some personal information (username, password, first name, and so on)

After providing this information, you will be forwarded to another page that will allow you to input your profile image, your bio and add some friends you might know.

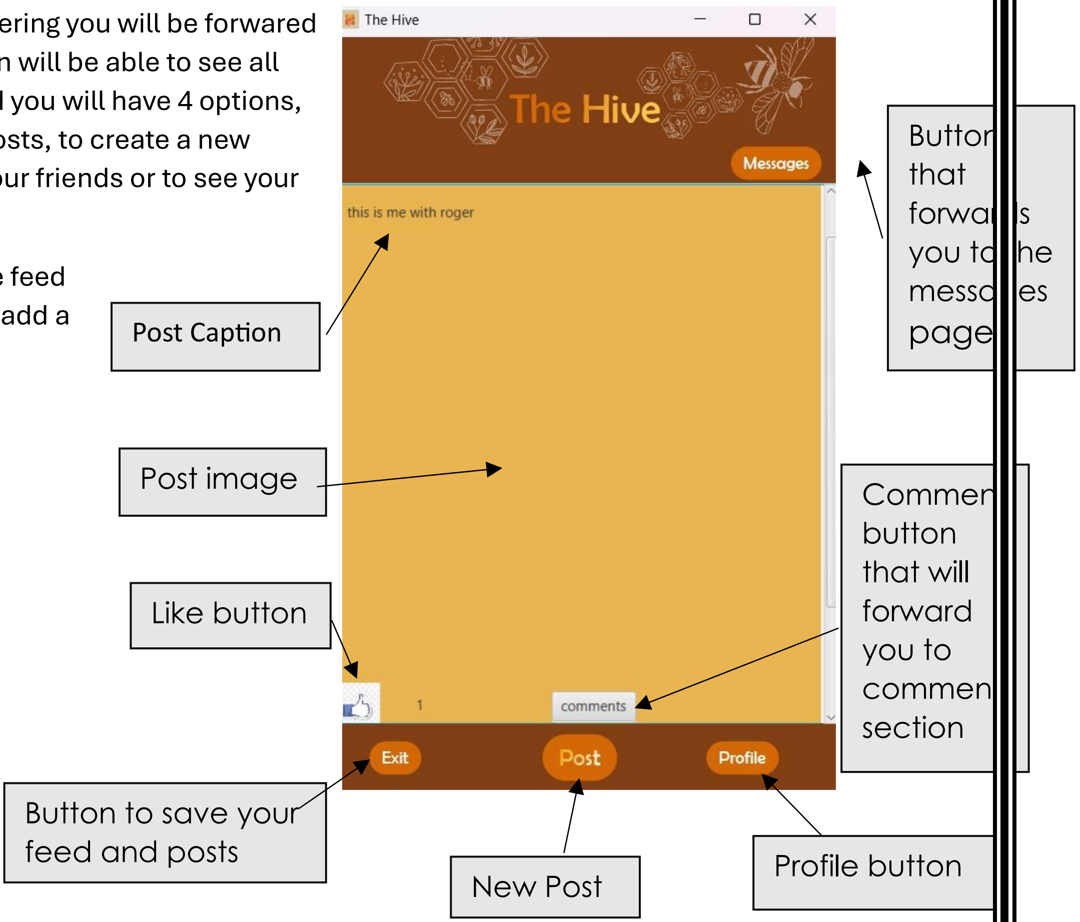
But if chose to login then you will be required to enter your username and password to be checked and have access to your account.

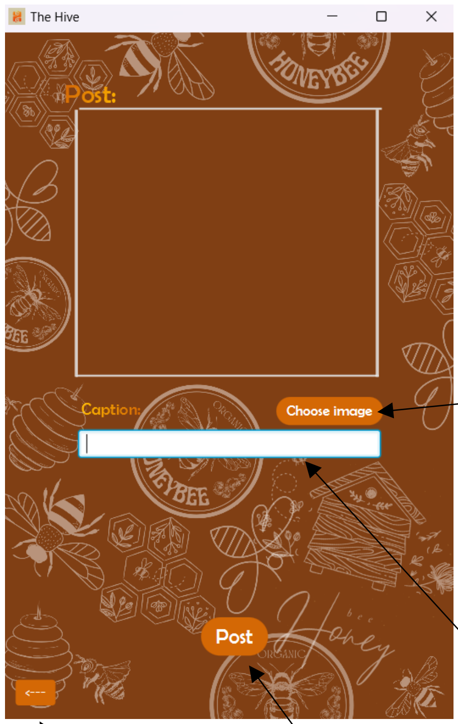
After logging in, the username and password that you entered either right will be forwarded to the feed page, or wrong so you will be required to enter the username and password again.



After either loging in or registering you will be forwared to the feed page that you then will be able to see all the posts of your friends, and you will have 4 options, to scroll to see the newest posts, to create a new post, to send messages to your friends or to see your profile page.

The posts you will have in the feed page can be liked or you can add a comment.


The Hive
The Hive
Messages
this is me with roger
1
comments
Exit
Post
Profile
Post Caption
Post image
Like button
Button to save your feed and posts
New Post
Profile button
Buttor that forwar s you to he messa es page
Commen button that will forward you to commen section



Button that allows you to choose a photo from your computer to be inserted.

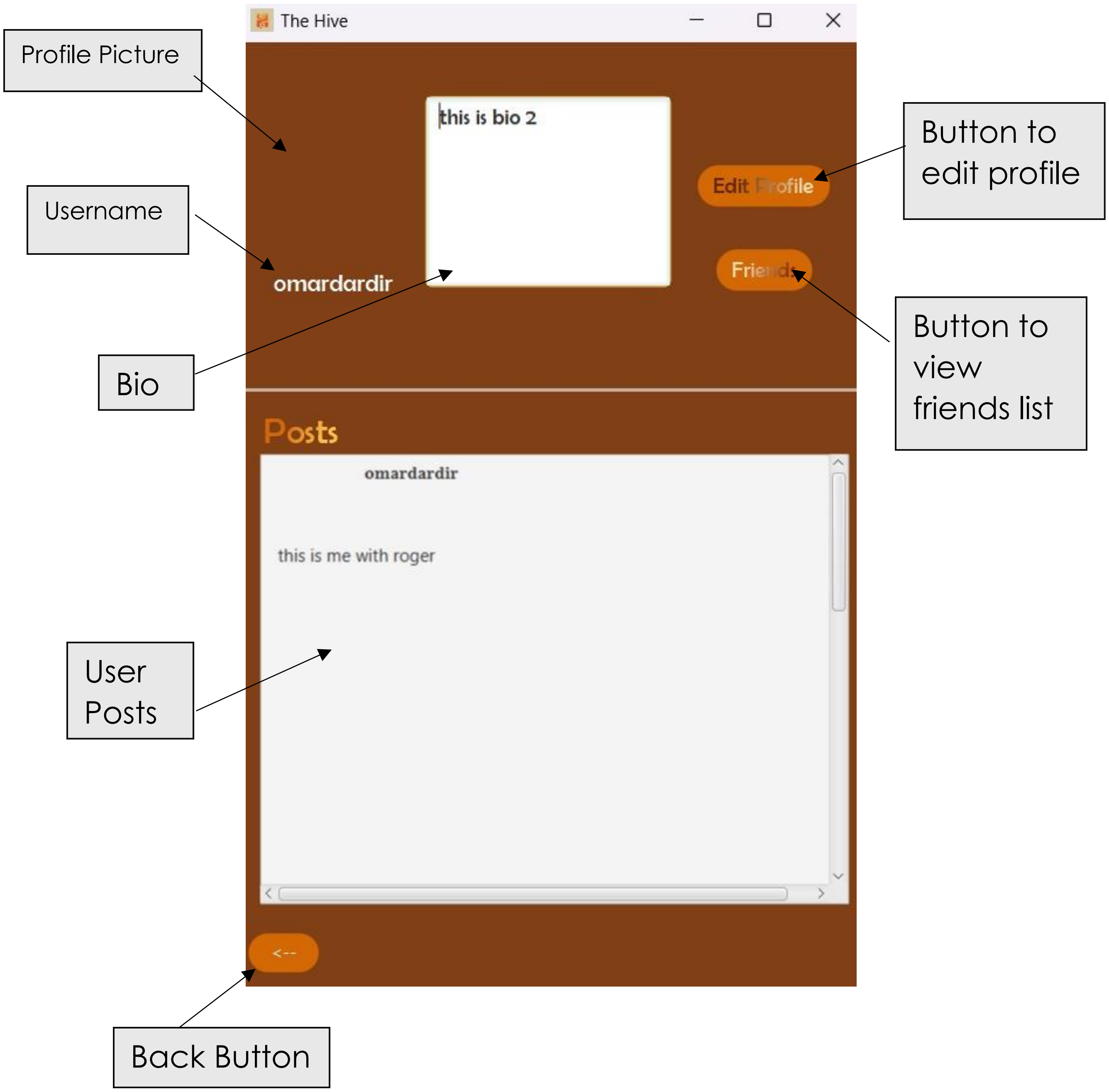
Add Caption

New Post

Back button

To add a new post, you will be forwarded to a new page that will allow you to add image and add caption

If the profile button was selected, then the user will be forwarded to his profile page to see his profile picture, his bio and his posts. Also he will be able to view his friends and edit his profile.

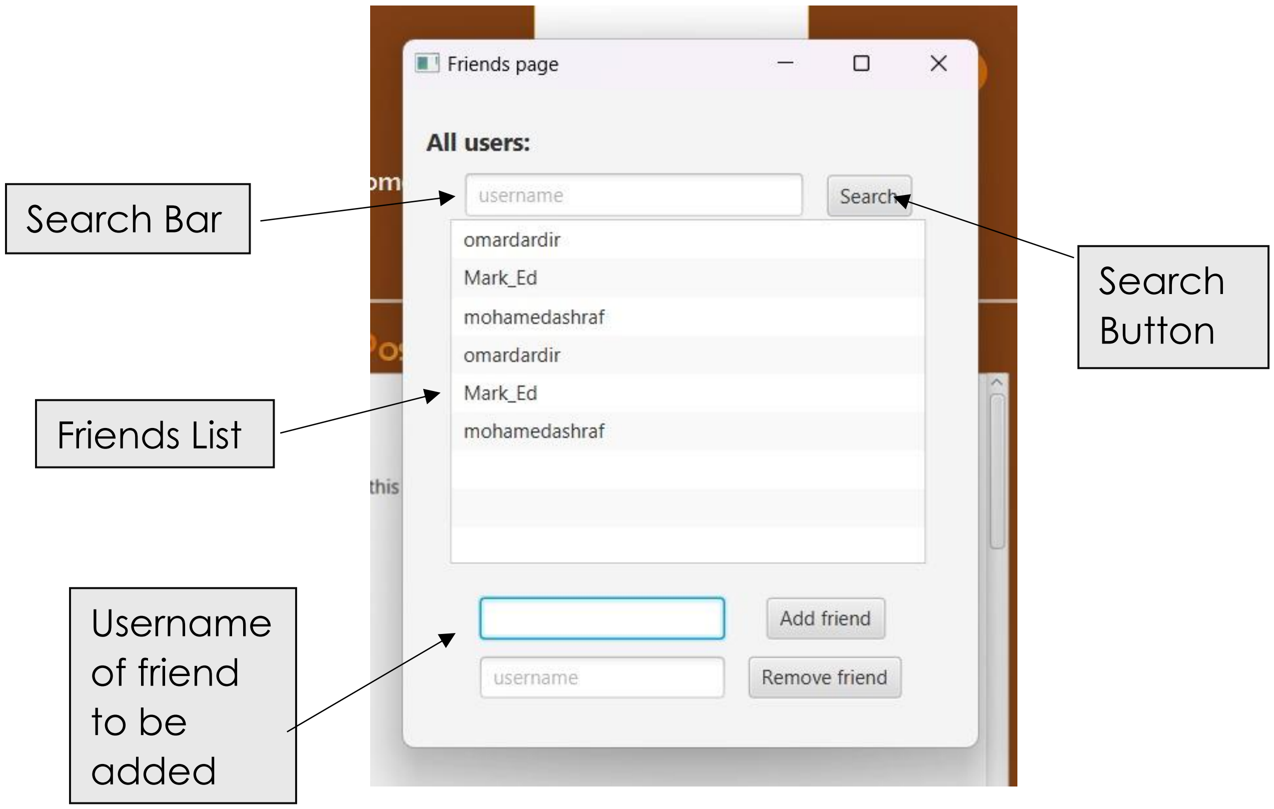


this is bio 2

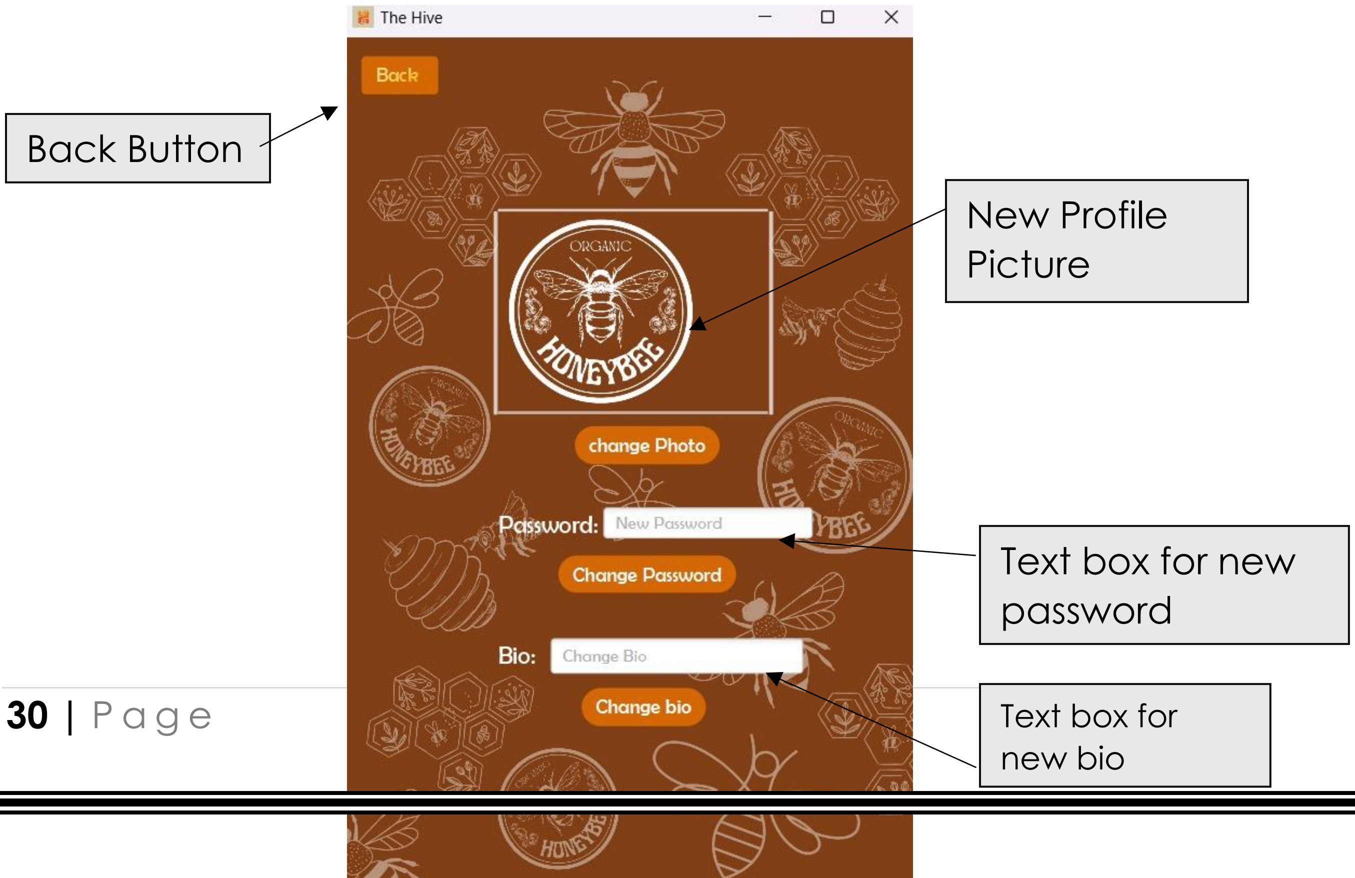
omardardir

this is me with roger

If you chose to view friends, then a list with all the users will be displayed to aloow you to search among them to certain friend you want to add.



And if the edit profile was selected, you will be forwarded to a page that will allow you to cahnge your profile picture or your bio or your password.



Conclusion: -

In conclusion, our project to develop a social media platform using Java, JavaFX, and Scene Builder has been a valuable learning experience in software development. By leveraging Object-Oriented Programming (OOP), we were able to design a well-structured and modular application. Each component, from users to posts and comments, was implemented as distinct objects, making the codebase easier to manage and extend.

Java provided a solid foundation for building the core functionality of the platform, allowing us to handle user interactions, data processing, and communication effectively. JavaFX, combined with Scene Builder, enabled us to create an engaging and intuitive user interface. Scene Builder's drag-and-drop interface simplified the design process, allowing us to focus on creating a seamless user experience.

For data storage, we utilized text files to save and retrieve information. While not as robust as databases, text files offered a straightforward solution for managing data in this project. This approach helped us understand the basics of data persistence and file handling in Java.

Overall, this project has successfully demonstrated how Java and its related technologies can be used to create a functional and interactive social media platform. It has provided us with practical experience in OOP, GUI development, and data storage, laying a strong foundation for future projects and more advanced applications.

Through this journey, we have gained valuable insights into software design and implementation, and we are well-prepared to tackle more complex challenges in the world of software development.

References: -

[1] Dr. Mahmoud Khalil slides

[2] Introduction to java programming 10th edition Daniel Liang

[3] brocade YouTube channel.

Contribution List: -

Martin Magued Wadie	(22P0193)
Messaging, Phase one classes.	
Mohamed Ashraf Mohamed	(22P0210)
Serialization and image handling.	
Omar Ahmed Abdelmohsen	(22P0218)
Even Handling, Phase one classes and testing.	
Abdelrahman Mostafa Ali	(22P0150)
Searching, phase one classes and report.	
Seif Aly Othman Fahmy	(22P0182)
Gui, Report and class diagram.	
Mark Edward Azziz	(22P0181)
Feed Page and event handling.	