Faculty Of Engineering

Ain Shams University ICHEP

Computer Engineering and Software Systems Program,

Report about Social Media Platform

Submitted to **Dr. Mahmoud Khalil** , **Eng. Mahmoud sohiel**

Submitted by:

- **Ahmed abbady 22p0308**
- George Mourad 22P0114
- **Seif elhusseiny 22p0215**
- **Anas tamer 22u0005**
- **Fatema saleh 22p0264**
- **Omar Saher 22P0161**

Table of contents:

Introduction:

In the realm of modern communication, social media platforms have revolutionized the way we connect, share, and interact with each other. With the surge in digital connectivity, the creation of a comprehensive Social Media Platform in Java emerges as a formidable endeavor, combining the robustness of Object-Oriented Programming (OOP) principles with the dynamism of Graphical User Interface (GUI), Multithreading, and basic networking.

The journey of developing such a platform unfolds in two pivotal milestones, each building upon the foundation of the previous one to realize a sophisticated and user-centric experience.

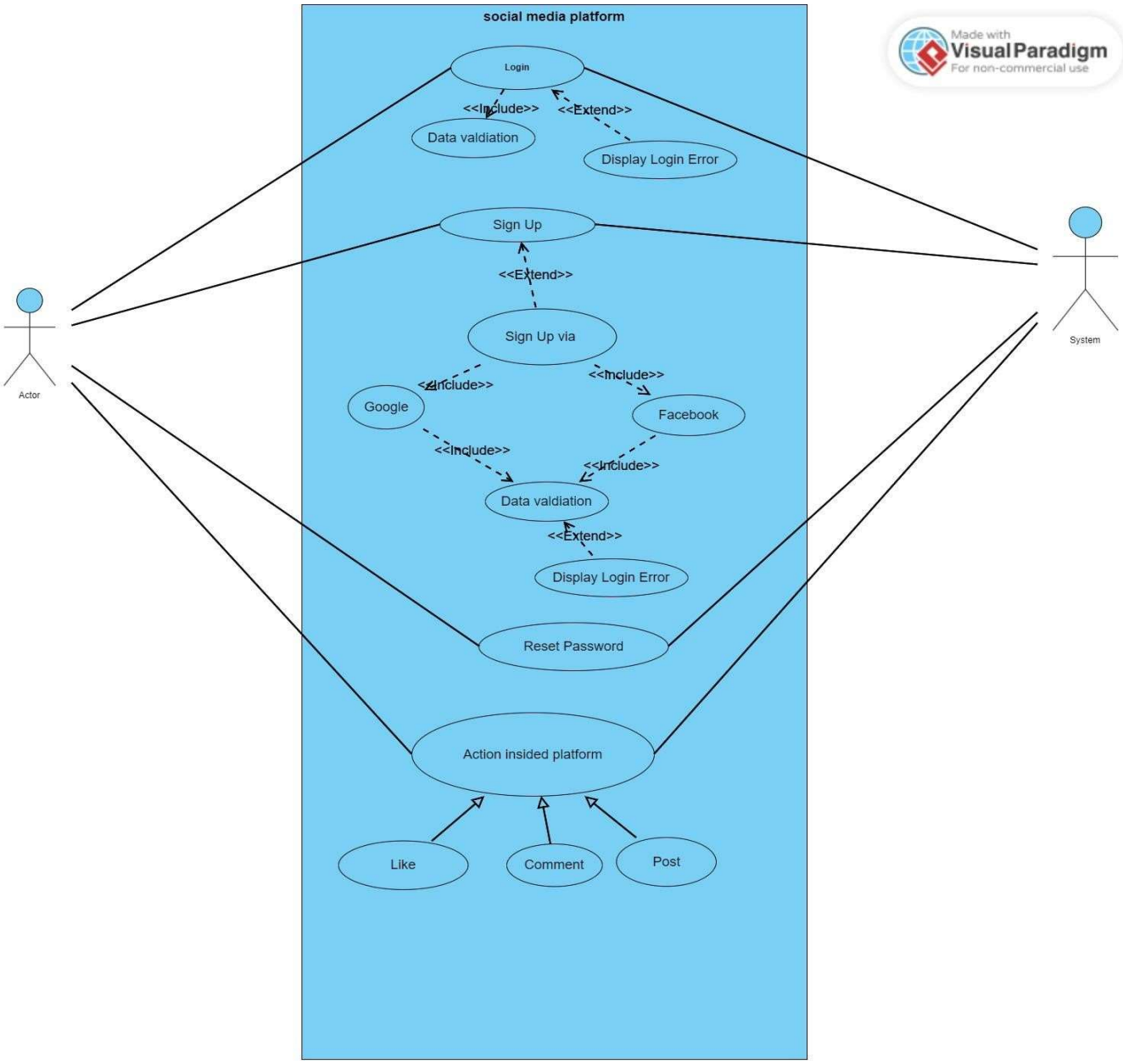
In the first milestone, the focus lies on leveraging OOP principles to architect a scalable and maintainable platform. Through encapsulation, inheritance, and polymorphism, the platform's structure is crafted to encapsulate user profiles, posts, comments, and interactions within a cohesive object-oriented framework. This milestone lays the groundwork for a flexible and extensible platform capable of accommodating future enhancements and features.

Transitioning to the second milestone, the integration of GUI, Multithreading, and basic networking amplifies the platform's capabilities. The introduction of a user-friendly graphical interface enhances user engagement, providing intuitive navigation and seamless interaction with the platform's functionalities. Concurrent processing enabled by Multithreading ensures efficient handling of concurrent user activities, optimizing performance and responsiveness.

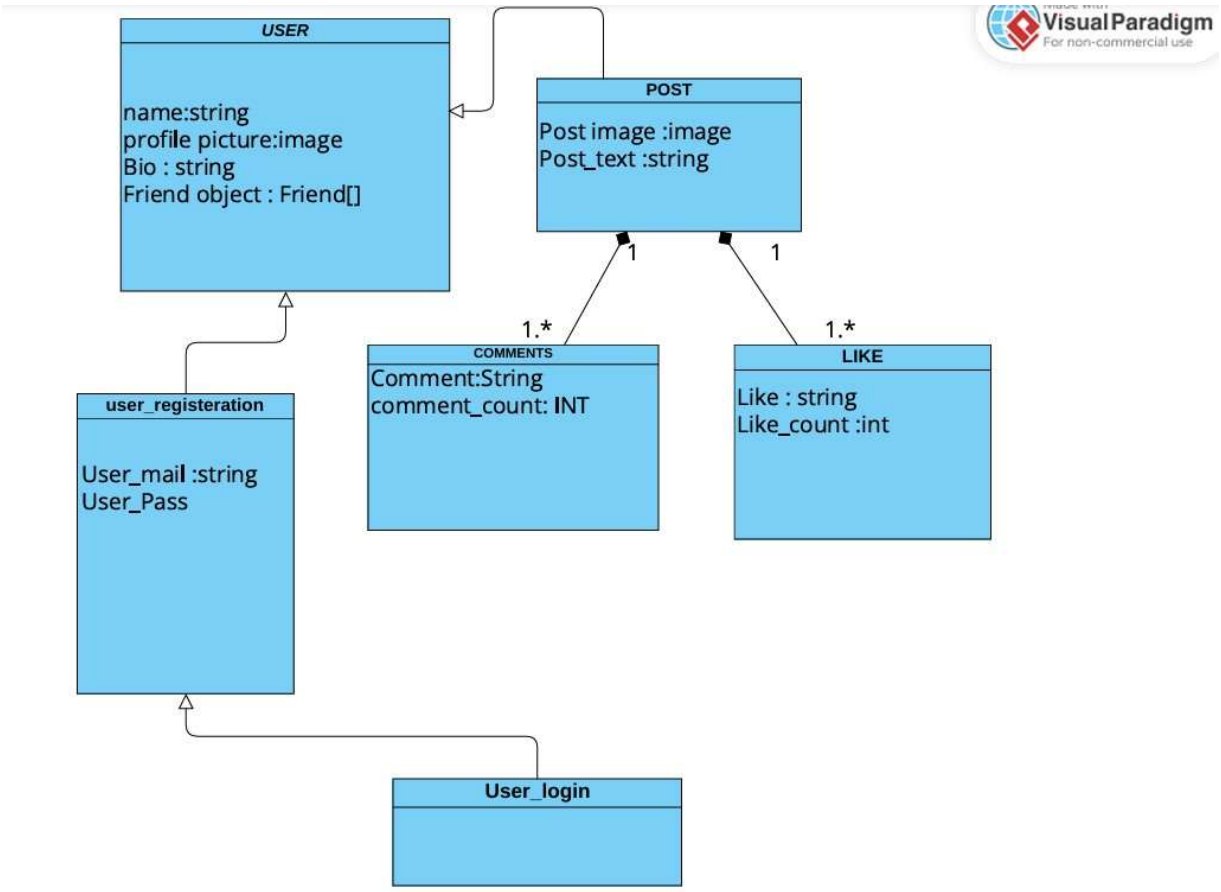
Moreover, basic networking functionality empowers users to connect and communicate across distributed environments, facilitating the exchange of messages, media, and content seamlessly. By embracing these advanced technologies, the Social Media Platform transcends conventional boundaries, fostering a vibrant and interconnected digital community.

UML Diagrams:

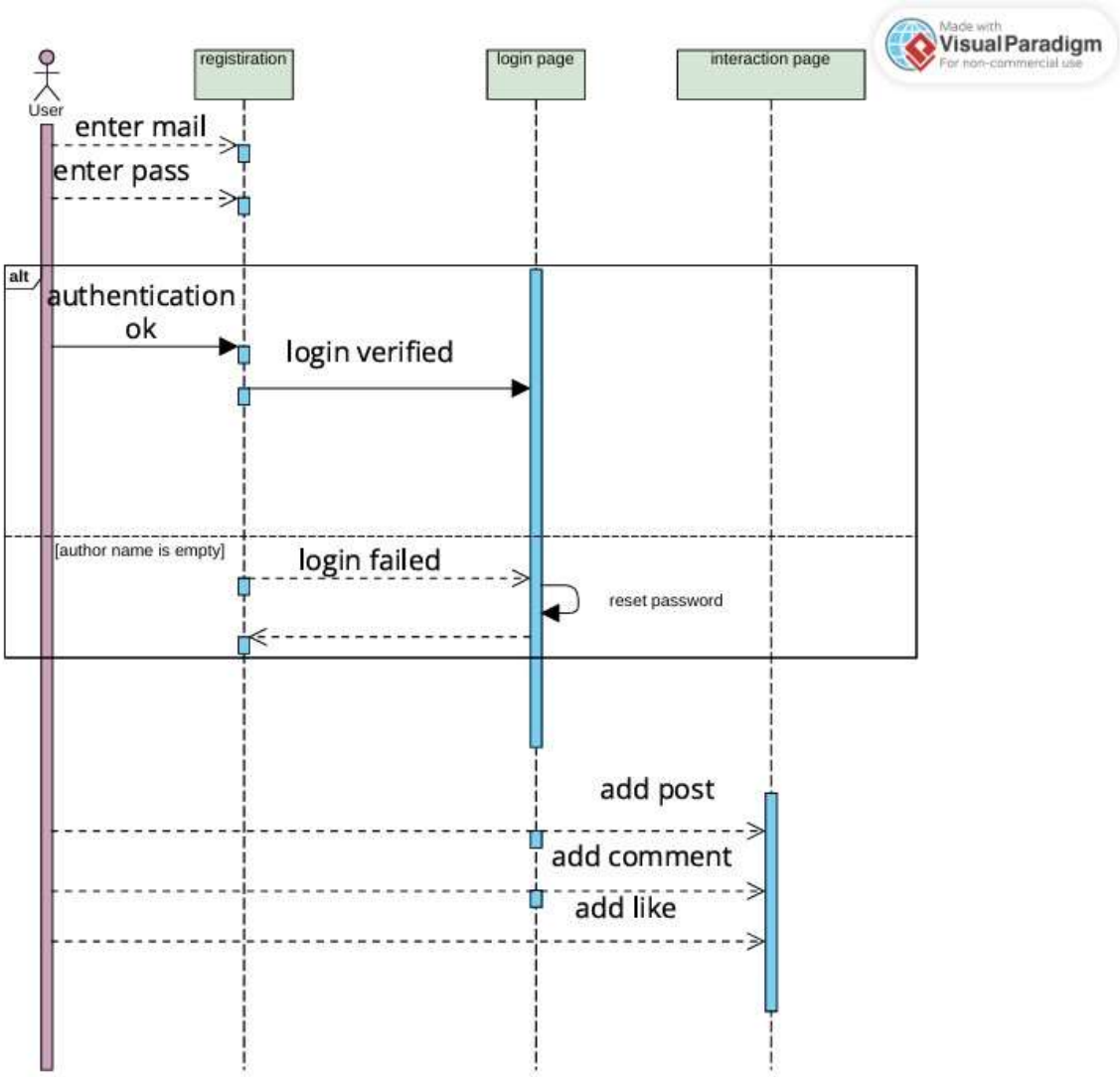
1) use case diagram



2) class diagram



3) sequence diagram


User
registration
login page
interaction page
enter mail
enter pass
alt
authentication ok
login verified
[author name is empty]
login failed
reset password
add post
add comment
add like

Milestone 1:

In this milestone the object oriented programming of the classes was needed . the classes implemented in this phase were user_registiration , user_login , comment, like ,user and post. The main concepts used in this phase are

- Methods: in this phase are the setters and getters and some implemented methods for each class shown in Fig.3 and Fig.4.
- Inheritance:that is used in some classes like the inheritance relation between user_registiration and user shown in Fig.1.
- Constructors: that are used as the way to create objects from a class in other classes(Main class in our case) shown in Fig.2.

```
public class user_registeration extends user {  5 usages  1 inheritor

    protected String user_mail;  7 usages
    protected String user_pass;  5 usages
    private boolean validatepass=false;  2 usages
    private boolean validatemail=false;  2 usages

    public  user_registeration(){  1 usage
        //default
    }
    public  user_registeration(String user_mail ,String user_pass){  3 usages
        this.user_mail = user_mail;
        this.user_pass = user_pass;
        //default
    }
```

Fig.1:(inheritance)

```
public  user_registeration(String user_mail ,String user_pass,  boolean validatepass,  boolean validatemail){
    this.user_mail=user_mail;
    this.user_pass=user_pass;

    this.validatemail=validatemail;
    this.validatepass=validatepass;
    //default
}

//setters
public void set_mail(String maillll ){  no usages
    user_mail=maillll;
}
```

Fig.2:(Constructor)

```
//setters
public void set_mail(String maillll ){  no usages
    user_mail=maillll;
}
public void set_pass(String passs ) { user_pass=passs; }
public void setvalidatemail(boolean validatemail) { this.validatemail=validatemail; }
public void setvalidatepass(boolean validatepass) { this.validatepass=validatepass; }
//getters
public String get_mail() { return user_mail; }
public String get_pass() { return user_pass; }
public String getvalidatemail() { return user_mail; }
```

Fig.3:(Setters and getters methods)

```
public int usermailcheck() {  1 usage
    String mailcheck = user_mail.substring( beginIndex: user_mail.length() - 10);
    int chekerrr1;
    if (!mailcheck.equals("@gmail.com")) {
        chekerrr1=0;

    }
    else{
        chekerrr1=1;//true

    }
    return chekerrr1;
}

public int passwordcheck() {  1 usage
    int checkkerr2;
    if (!Character.isUpperCase(user_pass.charAt(0))) {

        checkkerr2=0 ;
    }
    else{
        checkkerr2=1;
    }
    return checkkerr2;

}
public int validAll() {  2 usages
    int a;
    if (passwordcheck() == 1 && usermailcheck() == 1) {
        a = 1;
    } else {
        a = 0;
    }
    return a;
}



}
```

Fig.4:(Some methods for the class)

The rest of the codes will be listed below:

User class code:

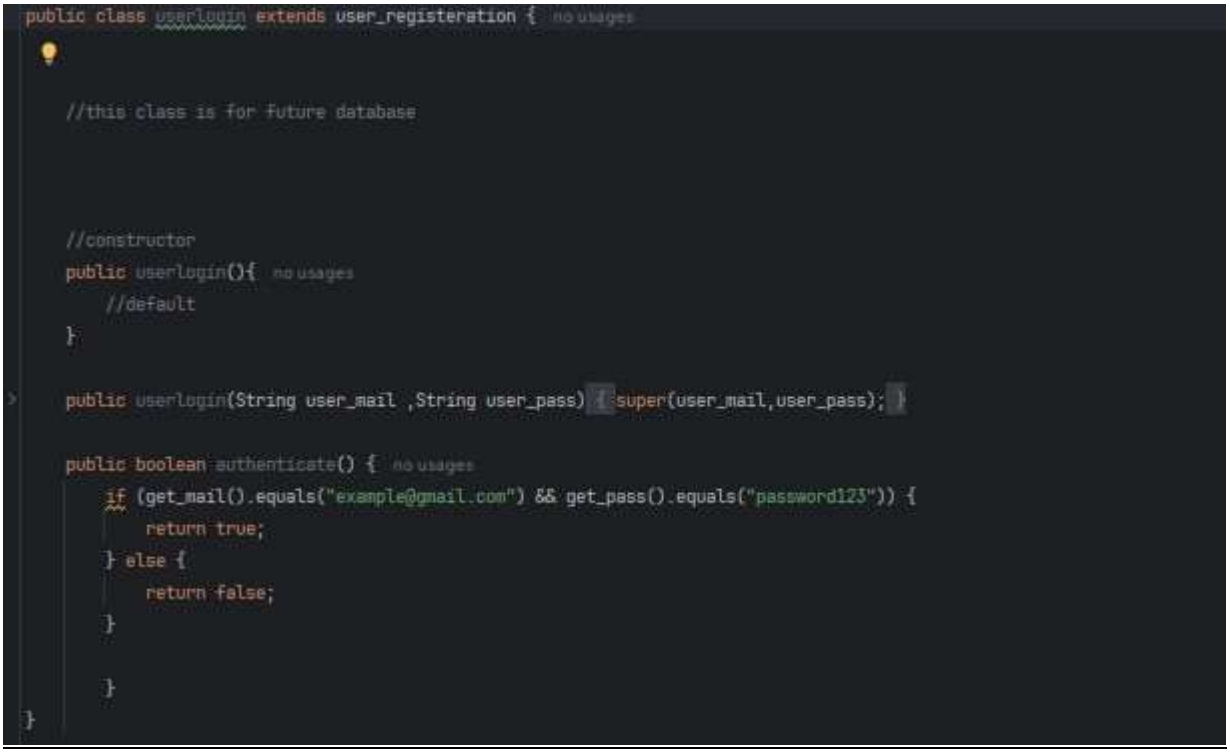
```java
public class user{  1 usage  2 inheritors
    private String name;  4 usages
    private BufferedImage picture;  3 usages
    private String bio;  3 usages
    private String[] friendObjects;  3 usages

    // Default constructor
    public user() {  3 usages
    }

    // Parameterized constructor
    public user(String name, BufferedImage picture, String bio, String[] friendObjects) {  no usages
        this.name = name;
        this.picture = picture;
        this.bio = bio;
        this.friendObjects = friendObjects;
    }

    // Getter methods
    public String getName() { return name; }

    public BufferedImage getPicture() { return picture; }

    public String getBio() { return bio; }

    public String[] getFriendObjects() { return friendObjects; }

    // Setter methods
    public void setName(String name) { this.name = name; }

    public void setPicture(BufferedImage picture) { this.picture = picture; }

    public void setBio(String bio) { this.bio = bio; }

    public void setFriendObjects(String[] friendObjects) { this.friendObjects = friendObjects; }
```

Fig.6:(user classcode)

```java
    public void userException() throws Exception {  no usages
        if (name.isEmpty()) {
            throw new Exception("Username can't be empty");
        }
    }

}
```

Fig.7:(User classmethod)

User login Class



```
public class userlogin extends user_registeration {

    //this class is for future database



    //constructor
    public userlogin(){
        //default
    }

    public userlogin(String user_mail ,String user_pass) { super(user_mail,user_pass); }

    public boolean authenticate() {
        if (get_mail().equals("example@gmail.com") && get_pass().equals("password123")) {
            return true;
        } else {
            return false;
        }

    }
}
```

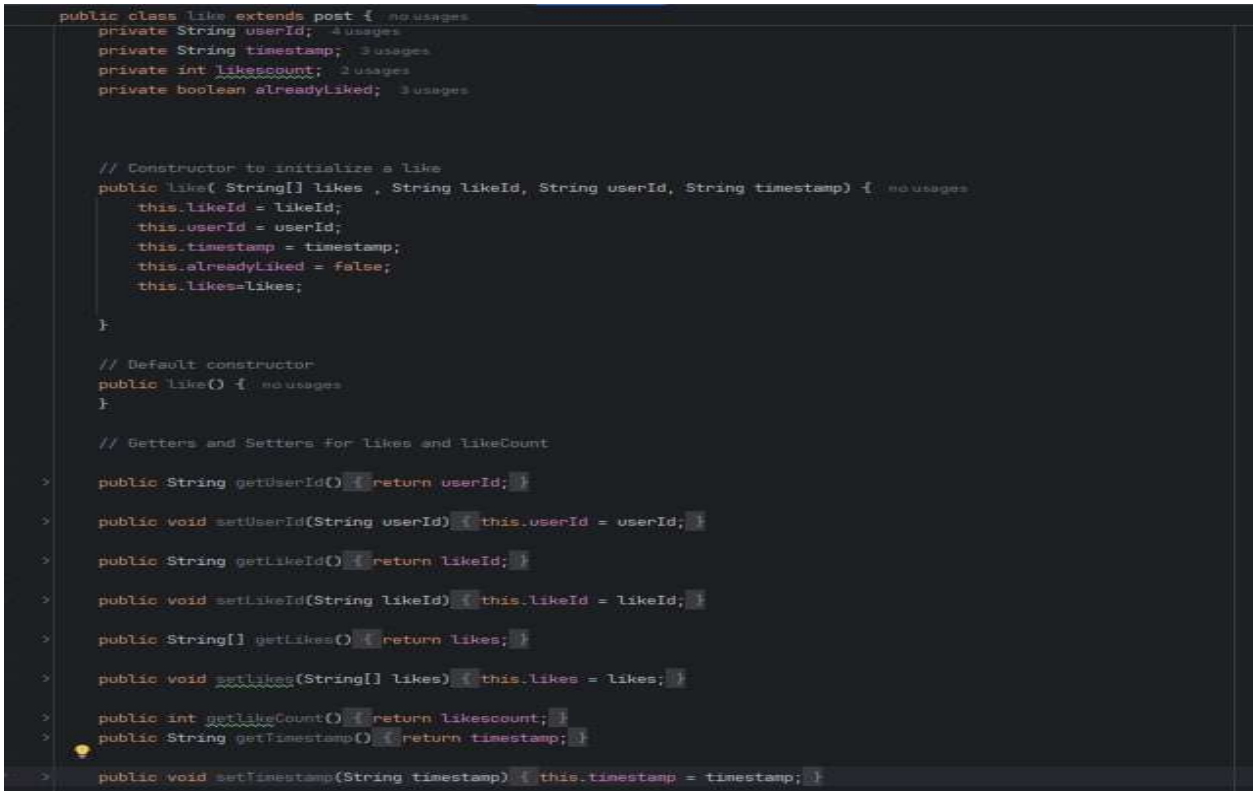
Fig.8:(User login class code)

Post Class

```
public class post {
    protected BufferedImage post_image;
    protected String post_text;

    public post(){
        //default
    }
    public post(BufferedImage IMA ,String ext ){
        //default
    }

    //setters
    public void set_image(BufferedImage post_ima) { post_image=post_ima; }
    public void set_tex(String post_te) { post_text=post_te; }
      //getters
    public BufferedImage get_image() { return post_image; }
    public String get_text() { return post_text; }

    //methods
    public void clearImage() { post_image = null; }
    public boolean hasText() { return post_text != null && !post_text.isEmpty(); }

    public boolean isEmpty() { return post_image == null && (post_text == null || post_text.isEmpty()); }

    public void displayDetails() {
        System.out.println("Post Image: " + (post_image != null ? "Available" : "Not Available"));
        System.out.println("Post Text: " + (post_text != null ? post_text : "No text"));
    }


}
```

Fig.9: (Post Class Code)

Like Class



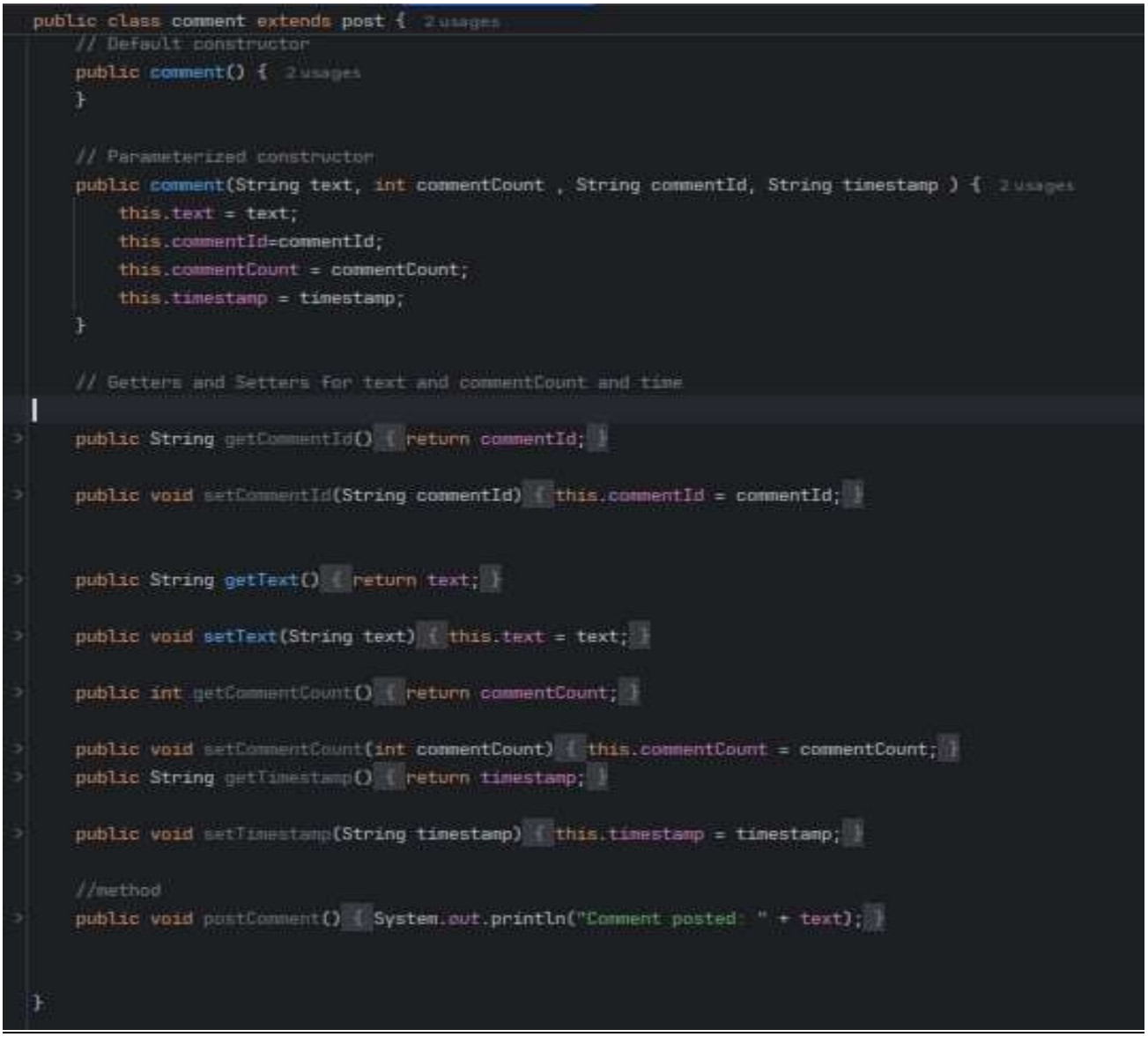
```
public class like extends post {
    private String userId;
    private String timestamp;
    private int likescount;
    private boolean alreadyLiked;



    // Constructor to initialize a like
    public like( String[] likes , String likeId, String userId, String timestamp) {
        this.likeId = likeId;
        this.userId = userId;
        this.timestamp = timestamp;
        this.alreadyLiked = false;
        this.likes=likes;

    }

    // Default constructor
    public like() {
    }

    // Getters and Setters for likes and likeCount

    public String getUserId() { return userId; }

    public void setUserId(String userId) { this.userId = userId; }

    public String getLikeId() { return likeId; }

    public void setLikeId(String likeId) { this.likeId = likeId; }

    public String[] getLikes() { return likes; }

    public void setlikes(String[] likes) { this.likes = likes; }

    public int getlikeCount() { return likescount; }
    public String getTimestamp() { return timestamp; }

    public void setTimestamp(String timestamp) { this.timestamp = timestamp; }
```

Fig.10: (Like Class Code)

```
// Method to like the content
public void likeContent() {
if (alreadyLiked) {
        throw new IllegalStateException("User has already liked this content.");
    }
    // Logic to save the like to the database or update the UI
    System.out.println("Content liked by user " + userId);
    alreadyLiked = true;
    likescount++; // Increment likesCount when a user likes the content
}
}
```

Fig.11: (Like Class method)

Comment Class



```
public class comment extends post {  2 usages
    // Default constructor
    public comment() {  2 usages
    }

    // Parameterized constructor
    public comment(String text, int commentCount , String commentId, String timestamp ) {  2 usages
        this.text = text;
        this.commentId=commentId;
        this.commentCount = commentCount;
        this.timestamp = timestamp;
    }

    // Getters and Setters for text and commentCount and time

    public String getCommentId() { return commentId; }

    public void setCommentId(String commentId) { this.commentId = commentId; }


    public String getText() { return text; }

    public void setText(String text) { this.text = text; }

    public int getCommentCount() { return commentCount; }

    public void setCommentCount(int commentCount) { this.commentCount = commentCount; }
    public String getTimestamp() { return timestamp; }

    public void setTimestamp(String timestamp) { this.timestamp = timestamp; }

    //method
    public void postComment() { System.out.println("Comment posted: " + text); }


}
```

Fig.12: (Comment Class Code)

Milestone 2

Description:

In this phase we used the graphical user interface to design a social media platform stages and scenes .

the designed Scenes in our program are :

1. login scene : shown in figures (13 to 15)
2. news feed Scene :shown in figures (16 to 19)
3. add post scene : shown in figures (20 to 22)

4. add comment scene: shown in figures (23 to 25)
5. user profile scene: shown in figures (26 to 28) in this phase the concepts used are Threading which is used in figures (29 to 30) also the fxml files are used to load the scenes which are mainly designed using Scenebuilder by controlling the

Classes using controller classes

Milestone 2 images.

Login scene

Description: the user in this page is asked to validate the account by entering his mail (must ends with gmail) and his password(must start with capital letter) and then the user will be switched to the newsfeed page. The used methods in this page is signup

```
import java.io.IOException;

public class loginScene {  2 usages  AhmmedAbbady

    public PasswordField password_field;  no usages
    Parent root;  4 usages
    Stage stage;  6 usages
    Scene scene;  4 usages

    String txt_email;  3 usages
    String txt_pass;  3 usages

    @FXML
    private Text Email_text;

    @FXML
    private Text Pass_text;

    @FXML
    private Button btn_login;

    @FXML
    private Button btn_signup;

    @FXML
    private Label checktoLogin;

    @FXML
    private Text dont_have_acc;

    @FXML
    private PasswordField passworHiddd;

    @FXML
    private TextField tfEmail;

    @FXML
    private Label welcome;
```

Fig.13: (Login scene code)

```
    @FXML
    private TextField tfEmail;

    @FXML
    private Label welcome;

    @FXML  AhmmedAbbady
    void login(ActionEvent event) throws IOException {...}
@FXML  AhmmedAbbady
void Signup(ActionEvent event) throws IOException {
    root = FXMLLoader.load(getClass().getResource( name: "registeration.fxml"));
    stage = (Stage)((Node)event.getSource()).getScene().getWindow();
    scene = new Scene(root);
    stage.setScene(scene);
    stage.show();
}
```

Fig.14: (Login scene methods)

Fig.15: (Login scene after running)

Newsfeed Scene

Description: in this page the user will interact with friends post by adding a comment (this will switch him to comment scene) and the user has multiple options to choose from (add post) or (go to his profile) or (see notifications appearing on the right pane of this page) the methods used here changetext , gotocomment , NewsfeGoprofile and logoutaction.

```
17      public class control_newsFeeeed {  1 usage   AhmmedAbbac
35          @FXML
36          private ImageView addpostimg;
37
38          @FXML
39          private HBox cr7img;
40
41          @FXML
42          private SplitMenuButton friendddd;
43
44          @FXML
45          private ImageView icon_accNews;
46
47          @FXML
48          private ImageView img1;
49
50          @FXML
51          private ImageView img22;
52
53          @FXML
54          private ImageView imgview1;
55
56          @FXML
57          private ImageView imgview2;
58
59          @FXML
60          private Label lausername;
61
62          @FXML
63          private ImageView like_post1;
64
65          @FXML
66          private ImageView like_post2;
67
68          @FXML
69          private Button logouttttt;
70
71          @FXML
72          private Text newstext;
```

Fig.16: (newsfeed code)

```
@FXML
private Button profileBtn;
@FXML
void changetext(MouseEvent event) {
    new Thread(() -> {
        // Assuming 'text' is the variable for the text component you want to change
        newstext.setStyle("-fx-text-fill: red;");
    }).start();
}

@FXML
void gotocomment(ActionEvent event) throws IOException {

 root = FXMLLoader.load(getClass().getResource( name: "commentview.fxml"));
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    scene = new Scene(root);
    stage.setScene(scene);
    stage.show();


}
```

Fig.17:(Newsfeed method)

```
    @FXML
    void newsfeGoprofile(ActionEvent event) throws IOException {
        root = FXMLLoader.load(getClass().getResource( name: "userprofile.fxml"));
        stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        scene = new Scene(root);
        stage.setScene(scene);
        stage.show();

    }
    @FXML
    void logoutAction(ActionEvent event) throws IOException {
        root = FXMLLoader.load(getClass().getResource( name: "loginScene.fxml"));
        stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        scene = new Scene(root);
        stage.setScene(scene);
        stage.show();

    }
    @FXML
    void newsfeeGoaddPost(ActionEvent event) throws IOException {
        root = FXMLLoader.load(getClass().getResource( name: "createpost.fxml"));
        stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        scene = new Scene(root);
        stage.setScene(scene);
        stage.show();

    }

}
```

Fig.18: (Newsfeed methods)

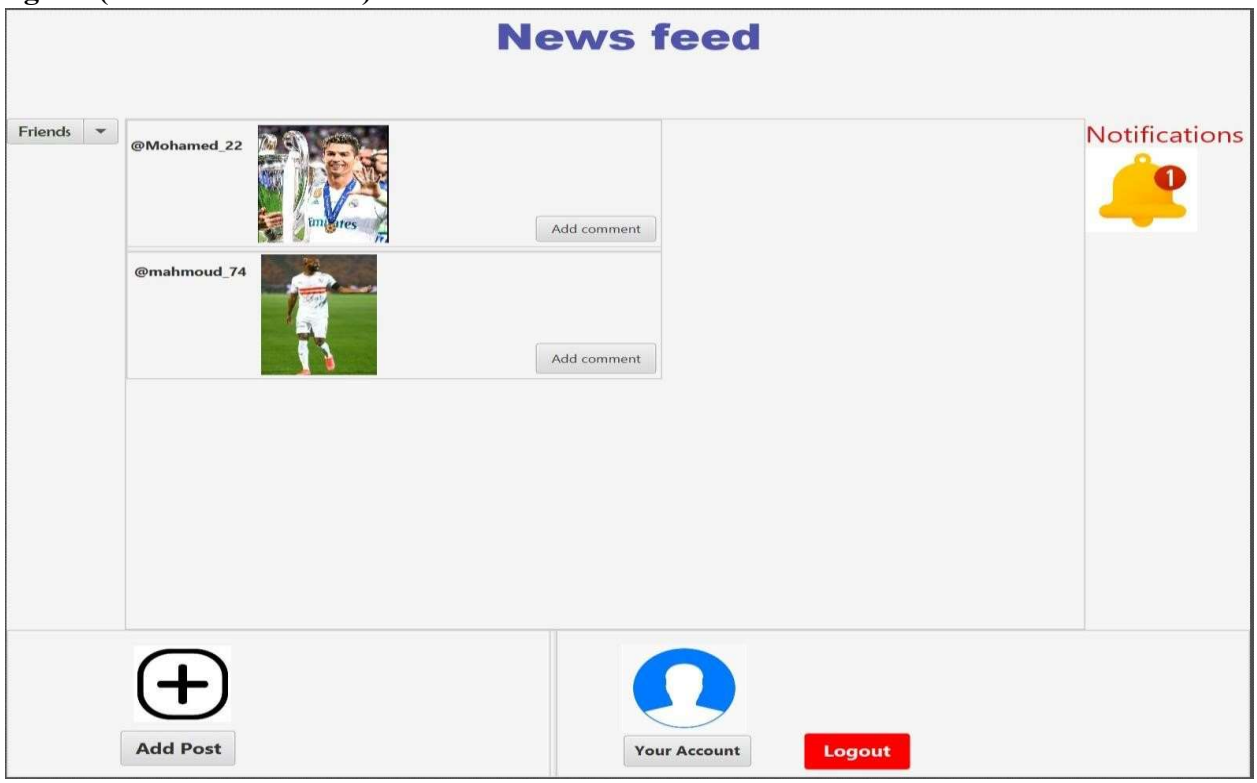
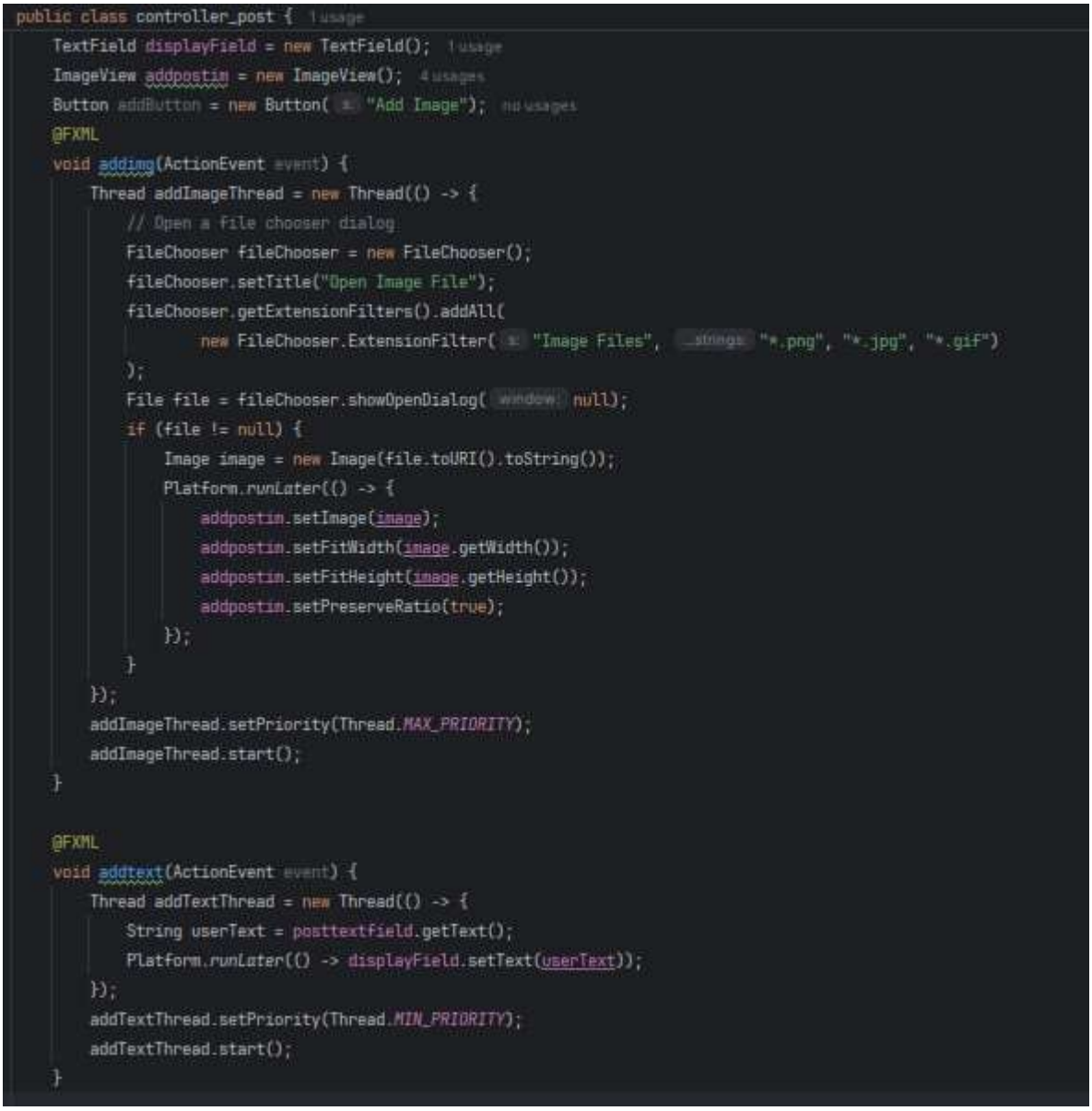


Fig.19:(newsfeed Scene after running)

Add post scene

Description: in this scene the user is given the ability to add a img or a text to post to appear in the newsfeed (note: adding image has higher priority than adding a text) the methods used here are addimg, addtext , btngohome



```
public class controller_post {
    TextField displayField = new TextField();
    ImageView addpostim = new ImageView();
    Button addButton = new Button("Add Image");
    @FXML
    void addimg(ActionEvent event) {
        Thread addImageThread = new Thread(() -> {
            // Open a file chooser dialog
            FileChooser fileChooser = new FileChooser();
            fileChooser.setTitle("Open Image File");
            fileChooser.getExtensionFilters().addAll(
                    new FileChooser.ExtensionFilter("Image Files", "*.png", "*.jpg", "*.gif")
            );
            File file = fileChooser.showOpenDialog(null);
            if (file != null) {
                Image image = new Image(file.toURI().toString());
                Platform.runLater(() -> {
                    addpostim.setImage(image);
                    addpostim.setFitWidth(image.getWidth());
                    addpostim.setFitHeight(image.getHeight());
                    addpostim.setPreserveRatio(true);
                });
            }
        });
        addImageThread.setPriority(Thread.MAX_PRIORITY);
        addImageThread.start();
    }

    @FXML
    void addtext(ActionEvent event) {
        Thread addTextThread = new Thread(() -> {
            String userText = posttextfield.getText();
            Platform.runLater(() -> displayField.setText(userText));
        });
        addTextThread.setPriority(Thread.MIN_PRIORITY);
        addTextThread.start();
    }
```

Fig.20: (Add post scene code)

```
@FXML
void addtext(ActionEvent event) {
    Thread addTextThread = new Thread(() -> {
        String userText = posttextfield.getText();
        Platform.runLater(() -> displayField.setText(userText));
    });
    addTextThread.setPriority(Thread.MIN_PRIORITY);
    addTextThread.start();
}

@FXML
private ImageView addpostimg;
@FXML
private Button btngohome;

@FXML
private Text addposttext;

@FXML
private Button postimg;

@FXML
private TextField posttxt;

@FXML
private VBox postveb;

@FXML
void btngohome(ActionEvent event) throws IOException {
    root = FXMLLoader.load(getClass().getResource( name: "newsFeeeed.fxml"));
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    scene = new Scene(root);
    stage.setScene(scene);
    stage.show();

}
```

Fig.21: (Add post scene method)

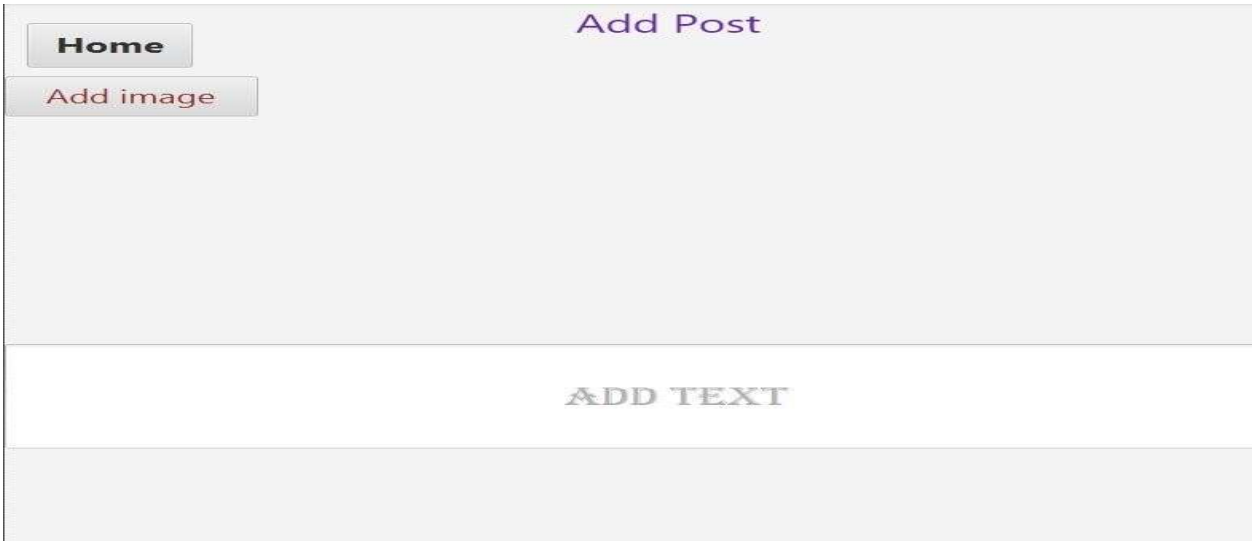
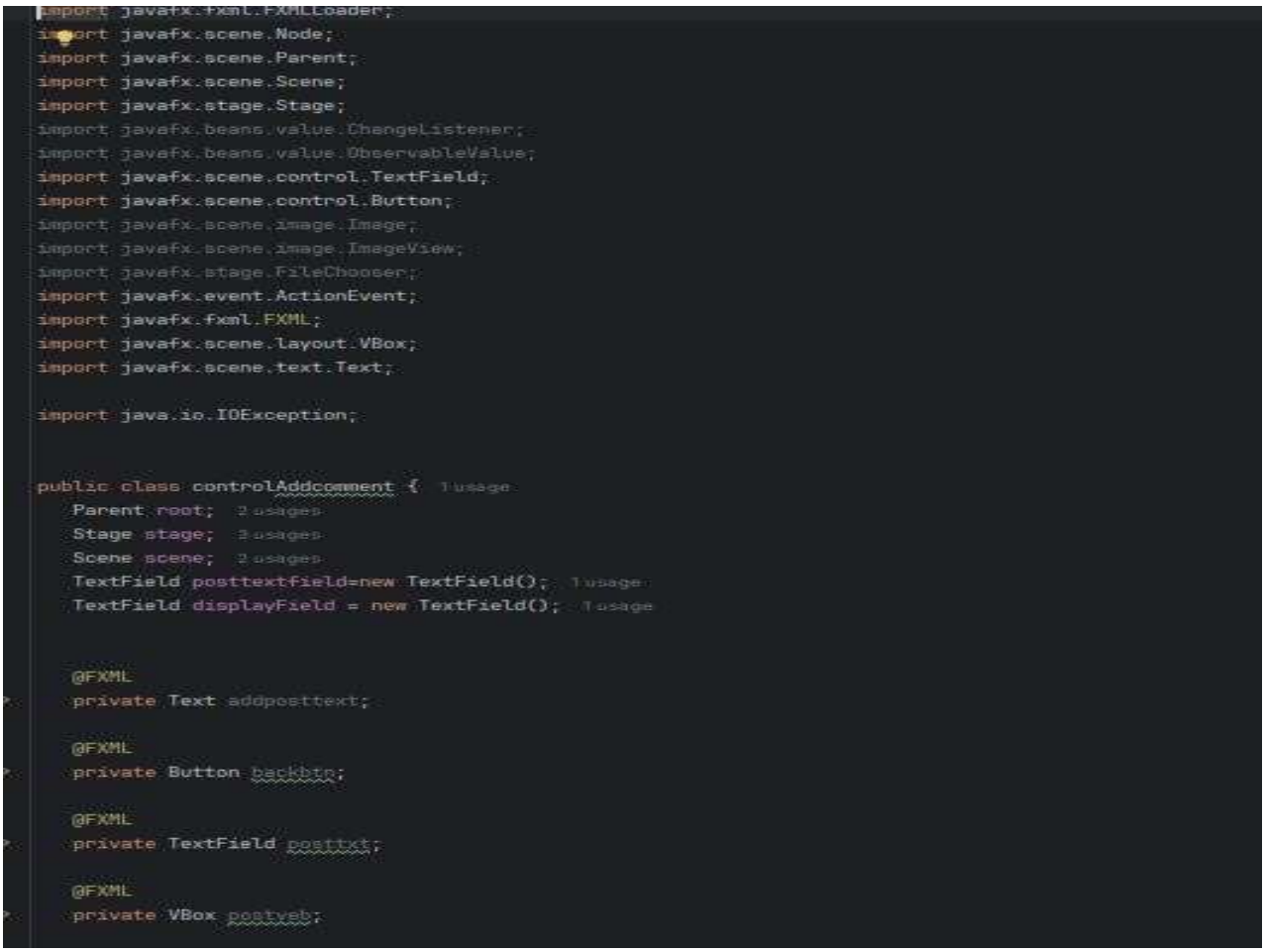


Fig.22: (Add post scene after running)

Add comment scene

Description: in this scene the user can see other users comments and can also add his comment in the text field prompted add comment and then can return back to

the newsfeed from the back button. The used methods are Switchtoprofile and addtext



```
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.scene.control.TextField;
import javafx.scene.control.Button;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.FileChooser;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;

import java.io.IOException;


public class controlAddcomment {  1 usage
    Parent root;  2 usages
    Stage stage;  3 usages
    Scene scene;  2 usages
    TextField posttextfield=new TextField();  1 usage
    TextField displayField = new TextField();  1 usage


    @FXML
    private Text addposttext;

    @FXML
    private Button backbtn;

    @FXML
    private TextField posttxt;

    @FXML
    private VBox postveb;
```

Fig.23: (Add comment scene code)

```
    @FXML
    private Text addposttext;

    @FXML
    private Button backbtn;

    @FXML
    private TextField posttxt;

    @FXML
    private VBox postveb;

    @FXML
    private TextField text1;

    @FXML
    private TextField text2;

    @FXML
    private TextField text3;
    @FXML
    void Switchtoprofile(ActionEvent event) throws IOException {
        root = FXMLLoader.load(getClass().getResource( name: "userprofile.fxml"));
        stage = (Stage)((Node)event.getSource()).getScene().getWindow();
        scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }
    @FXML
    void addtext(ActionEvent event){
        String userText = posttextfield.getText();

        // Set the text in displayField
        displayField.setText(userText);
    }
}
```

Fig.24: (Add comment scene methods)

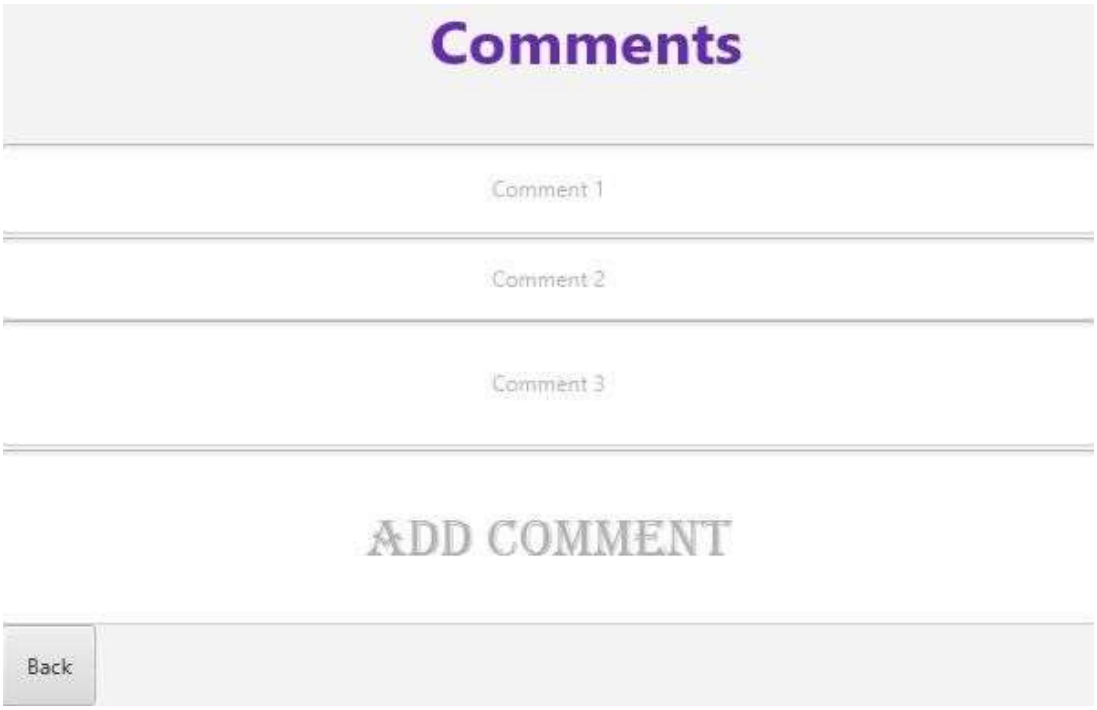
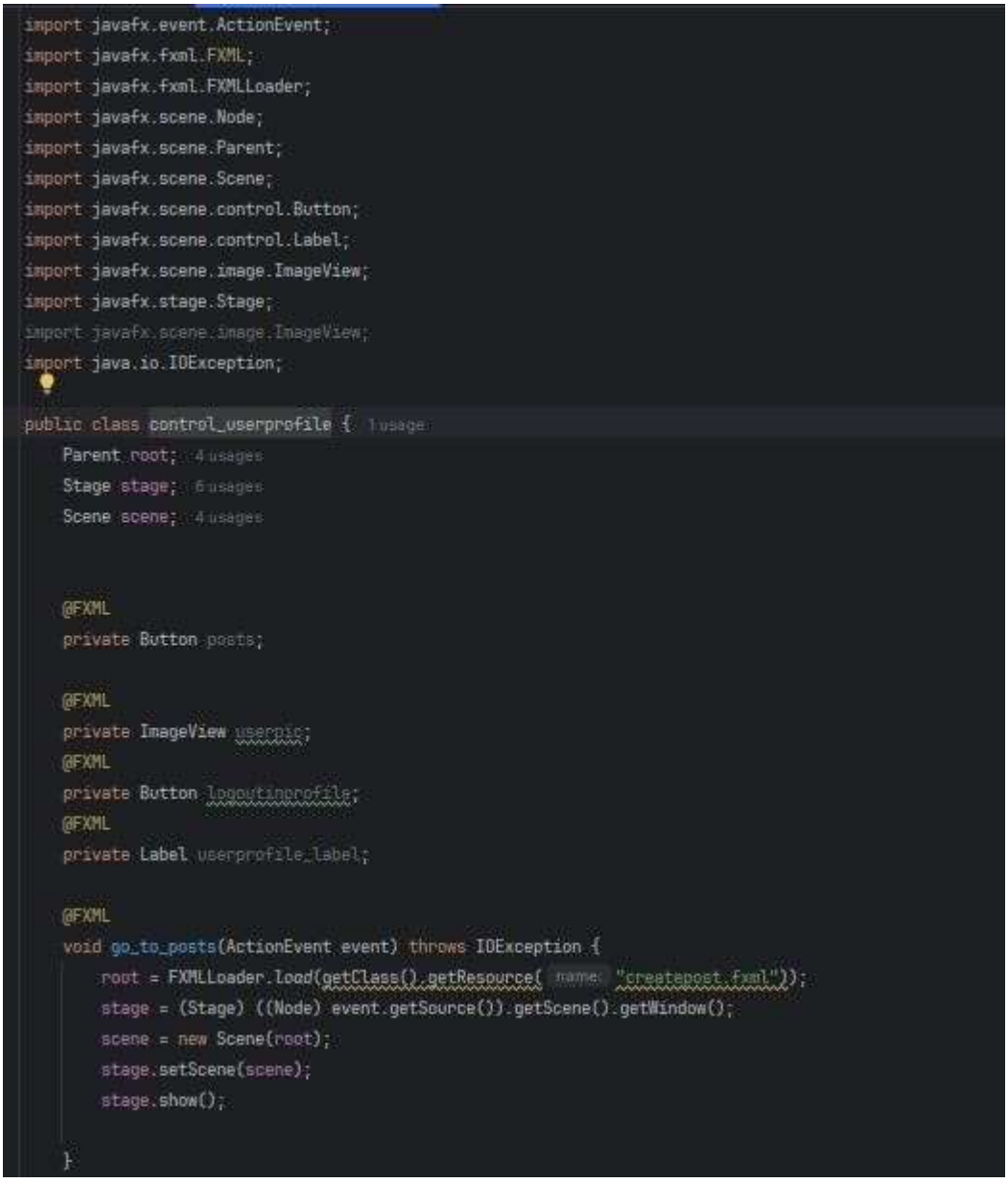


Fig.25: (Add comment scene after running)

User profile scene

Description: in this page the user has the abbility to see his account profile picture and friends and can logout by entering the logout button. Methods used here logoutbutton and go_to_posts.



```
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;
import javafx.scene.image.ImageView;
import java.io.IOException;

public class control_userprofile {
    Parent root;
    Stage stage;
    Scene scene;


    @FXML
    private Button posts;

    @FXML
    private ImageView userpic;
    @FXML
    private Button logoutinprofile;
    @FXML
    private Label userprofile_label;

    @FXML
    void go_to_posts(ActionEvent event) throws IOException {
        root = FXMLLoader.load(getClass().getResource( name: "createpost.fxml"));
        stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        scene = new Scene(root);
        stage.setScene(scene);
        stage.show();

    }
```

Fig.26: (User profile scene code)

```
    @FXML
    void logoutbtnpressed(ActionEvent event) throws IOException {
        root = FXMLLoader.load(getClass().getResource( name: "loginScene.fxml"));
        stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        scene = new Scene(root);
        stage.setScene(scene);
        stage.show();

    }


}
```

Fig.27: (User profile Scene methods)

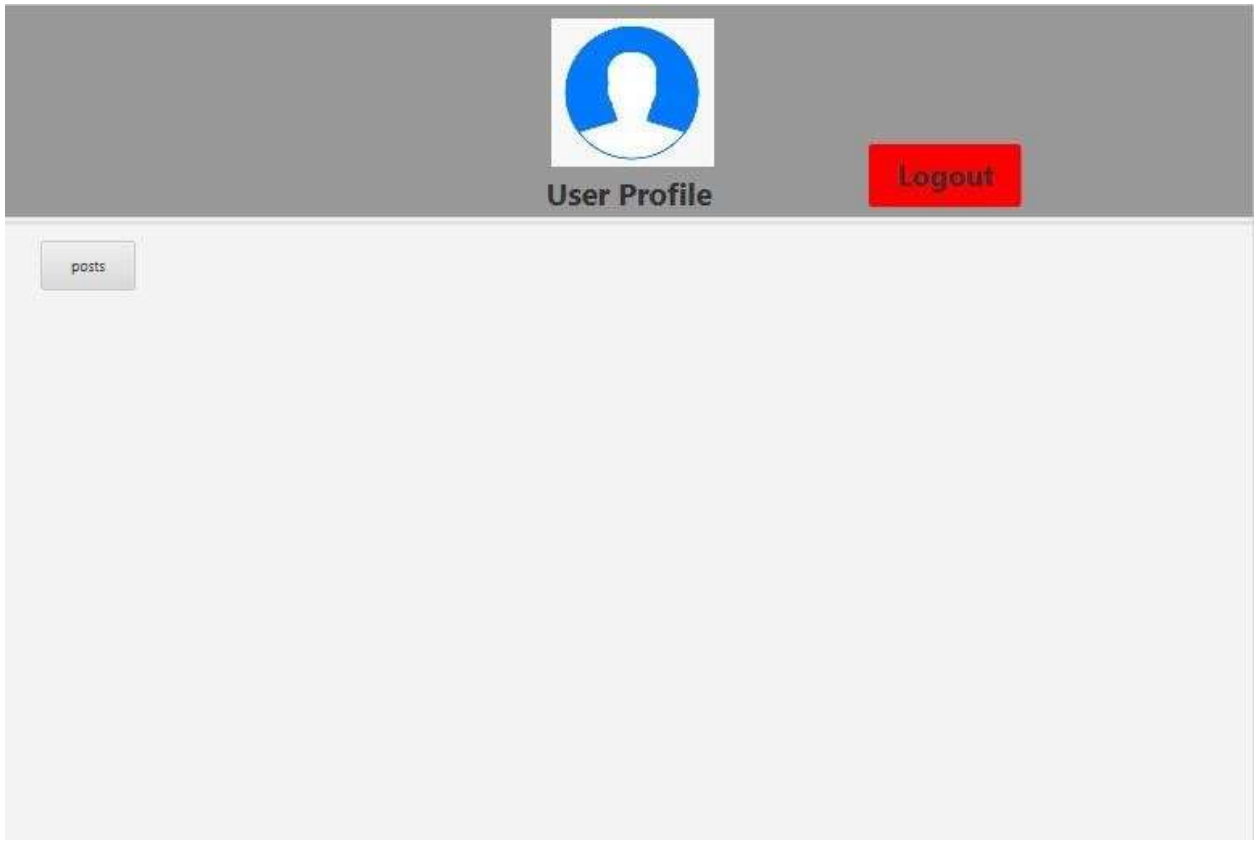


Fig.28:(User profile Scene After running)

Threading

Changetext thread used for changing newsfeed text color to red

```
void changetext(MouseEvent event) {
    new Thread(() -> {
        // Assuming 'text' is the variable for the text component you want to change
        newstext.setStyle("-fx-text-fill: red;");
    }).start();
}
```

Fig.29: (Threading in newsfeed scene code)

Addimg and addtext threads use thread priority to give adding img a higher priority used in addpost scene

```
void addimg(ActionEvent event) {
    Thread addImageThread = new Thread(() -> {
        // Open a file chooser dialog
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Open Image File");
        fileChooser.getExtensionFilters().addAll(
                new FileChooser.ExtensionFilter( s: "Image Files",  ...mmgs: "*.png", "*.jpg", "*.gif")
        );
        File file = fileChooser.showOpenDialog( window: null);
        if (file != null) {
            Image image = new Image(file.toURI().toString());
            Platform.runLater(() -> {
                addpostim.setImage(image);
                addpostim.setFitWidth(image.getWidth());
                addpostim.setFitHeight(image.getHeight());
                addpostim.setPreserveRatio(true);
            });
        }
    });
    addImageThread.setPriority(Thread.MAX_PRIORITY);
    addImageThread.start();
}

@FXML
void addtext(ActionEvent event) {
    Thread addTextThread = new Thread(() -> {
        String userText = posttextfield.getText();
        Platform.runLater(() -> displayField.setText(userText));
    });
    addTextThread.setPriority(Thread.MIN_PRIORITY);
    addTextThread.start();
}
```

Fig.30: (Threading in addpost scene)

4.0 Conclusion:

The development of a social media platform using JavaFX and Scene Builder was a successful and rewarding project. By leveraging the powerful GUI capabilities of JavaFX, combined with the visual design tools provided by Scene Builder, we were able to create an attractive and intuitive social media application.

The use of JavaFX allowed us to build a highly responsive and interactive user interface, with smooth animations, transitions, and controls that provided an engaging experience for users. Scene Builder enabled rapid prototyping and development, allowing us to quickly iterate on the design and layout of the application.

Through the course of this project, we were able to implement key features of a social media platform, including user profiles, newsfeed, posting/sharing content, commenting, liking, and following other users. The modular and extensible architecture of the application also sets the stage for future expansion and addition of new features.

Overall, this project has demonstrated the viability and effectiveness of using JavaFX and Scene Builder for building complex, feature-rich desktop applications like a social media platform. The skills and knowledge gained through this experience will be valuable assets moving forward, both for enhancing this application further and for tackling future software development challenges.

QR_CODE GitHub file :