Discussion Network

Group 10

Group Members: Hieu Truong, Selin Topac, Tabbie Brantley, & Lewis Cox

Glossary

I. Functional Specification (author all)

- Platform: Java Swing UI
- Run on: Window, MacOS, Linux

A. Features

1. There is a log-in page for users to log into their accounts or create accounts
2. There is a homepage that has a feed of all recent public posts.
3. There is a category page for filtering posts by topic.
4. There are posts from users that users write. The posts are on any social topic.
5. There is a point system for all user accounts.
6. Users can read other users' posts, agree or disagree with their posts, and repost the post by writing their own opinion on it.
7. Users get points for agrees they get on their posts, and lose points for disagrees they get on their posts.
8. There is a leaderboard that shows the top 5 user accounts based on their points.
9. There is a Profile page for users to view their profile.

B. Functions

1. A user logs in the system using username and password on log in page.
2. A user creates an account by clicking the create account button.
3. A user enters a username and password on the create account page to create an account.
4. A user switches between homepage, profile page, categories page, and leaderboard page by clicking the corresponding buttons on a menu tab.

5. A user edits the profile or password by clicking the edit profile button in the profile page.
6. A user creates a new topic by clicking the write new post button from the profile page.
7. A user scrolls down the home page to read all the recent posts.
8. A user scrolls down the category page to read all the recent posts in that category.
9. A user scrolls down the profile page to read their own page.
10. A user reposts a topic by clicking the repost button when reading a post.
11. A user can click the agree or disagree button when reading a post.
12. A user deletes a post by clicking the delete button on their post.
13. A user can filter public posts by topic on the the category page by selecting one of the displayed topics.
14. Each user accounts' score updates when they receive an agree on their post (plus one point) or a disagree on their post (minus one point).

II. Use Cases

Create Account (author Selin)

1. The user opens the application
2. The system displays the login screen
3. The user chooses to create an account
4. The system displays a new page with boxes for the user to enter their username, password, and password confirmation
5. User enters account name
6. User enters password
7. The user re-enters password in password confirmation box
8. The user presses create account
9. The system creates the account
10. The system brings the user back to the login page.

Variation #1

1.1 In step 7, the user enters an incorrect password.

1.2 After step 8, system then displays error message

Passwords do not match, please re-enter password

1.3 The system continues to step 7

Variation #2

2.1 In step 5 and/or 6, user does not follows username and password length constraints.

2.2 After step 8, the system displays an error message

The username and password must be between 8 and 20 characters.

2.4 User continues to 5 and/or 6.

Variation #3

3.1 In step 5, the user enters an account name in use

3.2 The system displays an error message

The username has already been used,

please use a new username.

3.3 The system continues to step 5

Variation #4

4.1 At any step, the user selects the “Return to Login” button.

4.2 System returns to login screen.

Log In (author Selin)

1. If the user has an account, they open the application
2. User enters their username and password
3. User clicks “Log in”
4. The system logs into their account
5. The home page of the application is shown

Variation #1

1.1 In step 2, if the username and password are incorrect

1.2 The system displays a message

Incorrect Password or Username

1.3 System does not sign into the account

1.4 Continue to step 2

Write New Post (author Hieu)

1. After the user is logged in, the application will display the menu with buttons (Home, Category, Leaderboard, Profile)
2. User clicks on the “Profile” button
3. System displays/opens the profile page
4. User clicks on the “Add Post” button

5. A blank text box appears
6. User writes their post in the text box
7. User selects category for the post
8. User clicks on “Post/Publish”
9. System creates the post

Variation #1

1.1 In step 8, the user clicks on the “Cancel” button

1.2 The post will not be published

1.3 Continue to step 3

Delete a Post/repost (author Hieu)

1. User carries out Log in
2. System displays home page and the menu bar in the top
3. User chooses Profile in the menu bar
4. System display Profile page with the feed of user’s posts
5. Every post displays a “Delete post” button
6. User chooses “Delete post”
7. System confirms deletion
8. User chooses confirm
9. System deletes that post from user’s posts.

Variation #1

1.1 Start at step 7

1.2 User choose cancel

1.3 System doesn’t delete the post

1.4 System continues at step 4

Read Recent Post (author Hieu)

1. User carries out Log in
2. The system retrieves the posts from all categories
3. The system displays homepage with the feed of posts in the order of time
4. The user scrolls up and down to read the posts

Read User Own Posts (author Hieu)

1. User clicks “Profile” button on the menu bar
2. The system retrieves the posts of user’s account
3. The system displays profile page with the feed of posts of user in the order of time
4. The user scrolls up and down to read the posts

Repost (author Hieu)

1. User reads the post
2. User selects repost button
3. The system displays the blank text box
4. User enters the content
5. User select Post/PublishThe system creates new post that linked to the original post
6. The system makes the re-post public.

Variation #1

1.1 Start at step 3

1.2 User selects cancel button

1.3 Continue to step 1

Agree/Disagree Post (author Lewis)

1. User reads the post
2. In every posts, the system also displays the agree/disagree button
3. The user selects agree or disagree
4. The system saves the agree/disagree and add/subtract points to/from the post.
5. The system highlights the agree/disagree button

Variation #1

1.1. Start at step 2

1.2 The user selects the agree/disagree button that has been Highlighted

1.3 The system subtracts/add points from/to the post.

1.4 The system de-highlights the agree/disagree button

1.5 Continue to step 1

Check Leaderboard (author Lewis)

1. System displays the menu bar
2. User selects the Leaderboard button on the menu
3. The system displays a new page with a Leaderboard image
4. System displays name and score of the top five user accounts with the most points

Find Topic Using Categories (author Lewis)

1. System displays the menu bar
2. User selects the categories button on the menu bar
3. The system shows a list of categories
4. The user is able to select one category at a time
5. The system brings the user to a new category page

6. The system displays the most recent post under that category
7. The user can interact with post from the system

Variation #1: Cancel the category list page

1.1 The system starts at step 1

1.2 The user selects the categories button

1.3 system bring user to the Category page

1.4 user selects a category form the list

1.5 System brings the user to a new page with post of the category

1.6 user selects the back button

1.7 system brings user back to category page

1.5 Continue to step 1.4

Edit Profile Name (author Tabbie)

1. System displays profile page
2. User selects the Edit Profile button
3. User selects the name to change it
4. User enters a new name
5. User selects save profile
6. System saves changes
7. User is brought back to Profile page

Variation #1

1.1 In step 4 user enters a name that is too short or too long.

1.2 User selects save button

1.3 An error message displays

Name has to be between 8 and 20 characters.

1.4 Continue to step 1

Variation #2

2.1 In any of the steps 3 through 5, user selects the cancel button instead

2.2 Name information is not changed

2.3 Continue to step 1

Edit Profile Password (author Tabbie)

1. System displays Profile page
2. User selects the Edit Profile button on Profile page
3. User selects the password to change it
4. User enters a new password
5. User confirms password by entering it in a separate text box
1. User selects the save button
2. System saves the changes
3. User is brought back to Profile page

Variation #1

1.1 In step 4 user enters password of wrong length.

1.2 User selects save button

1.3 An error message displays

Password must be between 8 and 20 characters.

1.4 Continue to step 1

Variation #2

3.1 In step 5, user enters a password that does not match the one in step 3

3.2 An error message displays

Your password does not match. Please re-enter your password.

3.3 Continue to step 3

Variation #3

4.1 In any of the steps 3 through 7, user selects the cancel button instead

4.2 Password information is not changed.

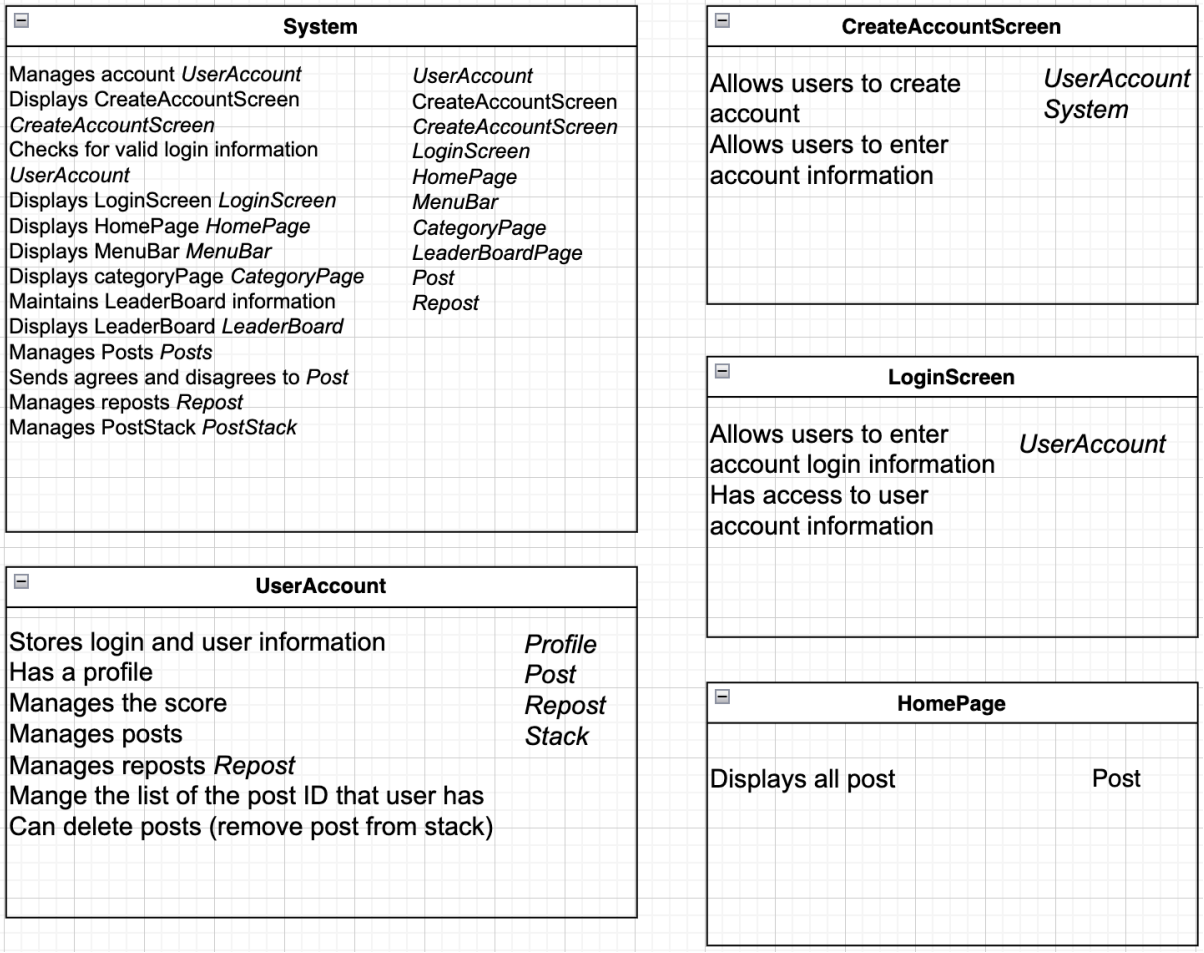
4.3 Continue to step 1

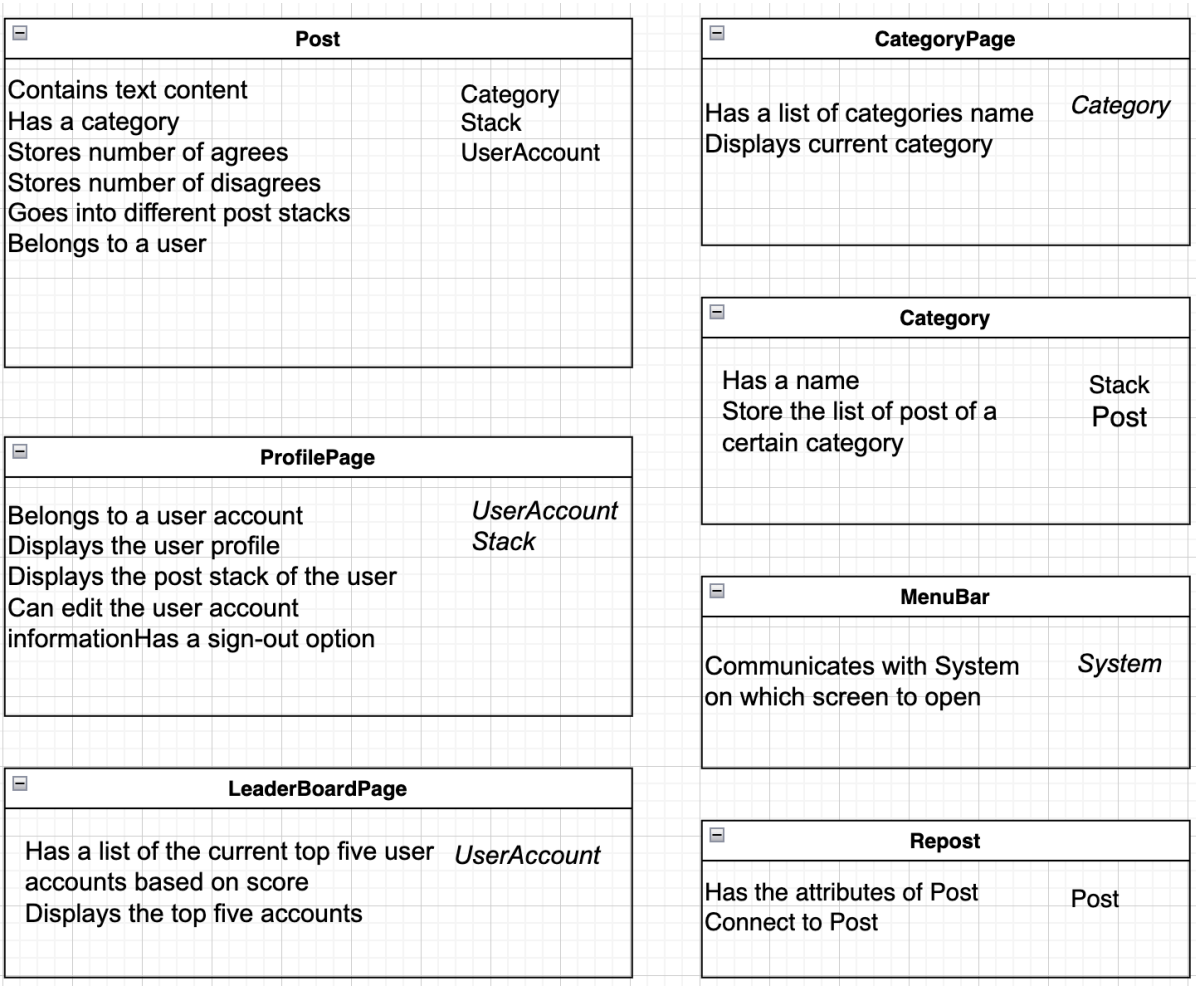
Logout (author Selin)

1. System displays Profile page
2. User selects the “Logout” button
3. System returns to Login screen

II. Design Specification

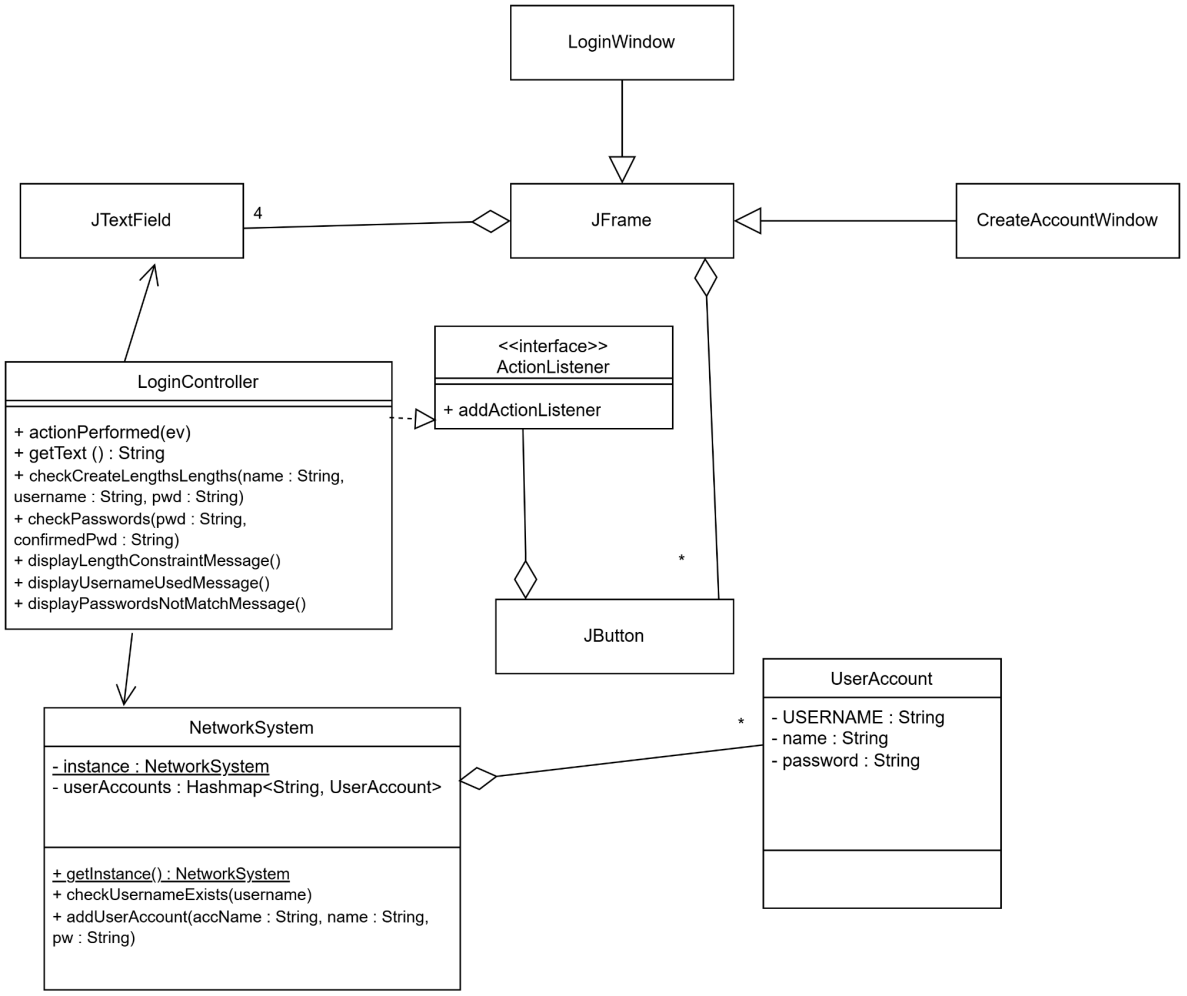
A. CRC cards (author all)




Post
Contains text content
Has a category
Stores number of agrees
Stores number of disagrees
Goes into different post stacks
Belongs to a user
Category
Stack
UserAccount
CategoryPage
Has a list of categories name
Displays current category
Category
Category
Has a name
Store the list of post of a certain category
Stack
Post
ProfilePage
Belongs to a user account
Displays the user profile
Displays the post stack of the user
Can edit the user account informationHas a sign-out option
UserAccount
Stack
MenuBar
Communicates with System on which screen to open
System
LeaderBoardPage
Has a list of the current top five user accounts based on score
Displays the top five accounts
UserAccount
Repost
Has the attributes of Post
Connect to Post
Post

B. UML class diagrams

1. Use case: Create account (author Tabbie)



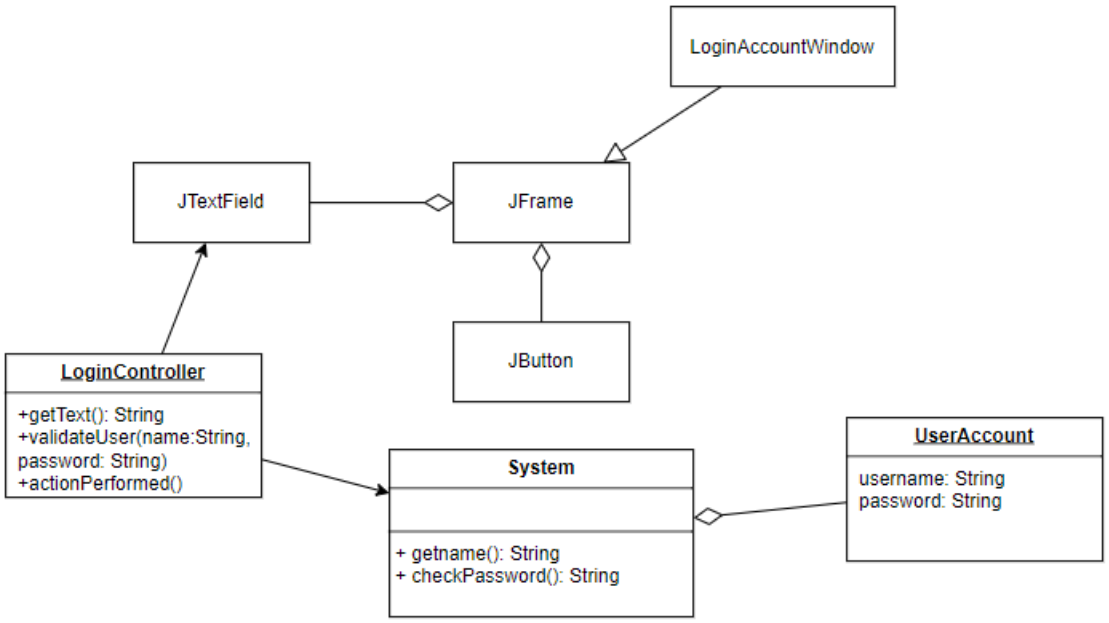
Observer Pattern - Mapping Table:

Name in Design Pattern	**Actual name**
Subject	JButton
Observer	ActionListener
ConcreteObserver	LoginController
attach()	addActionListener
notify()	actionPerformed

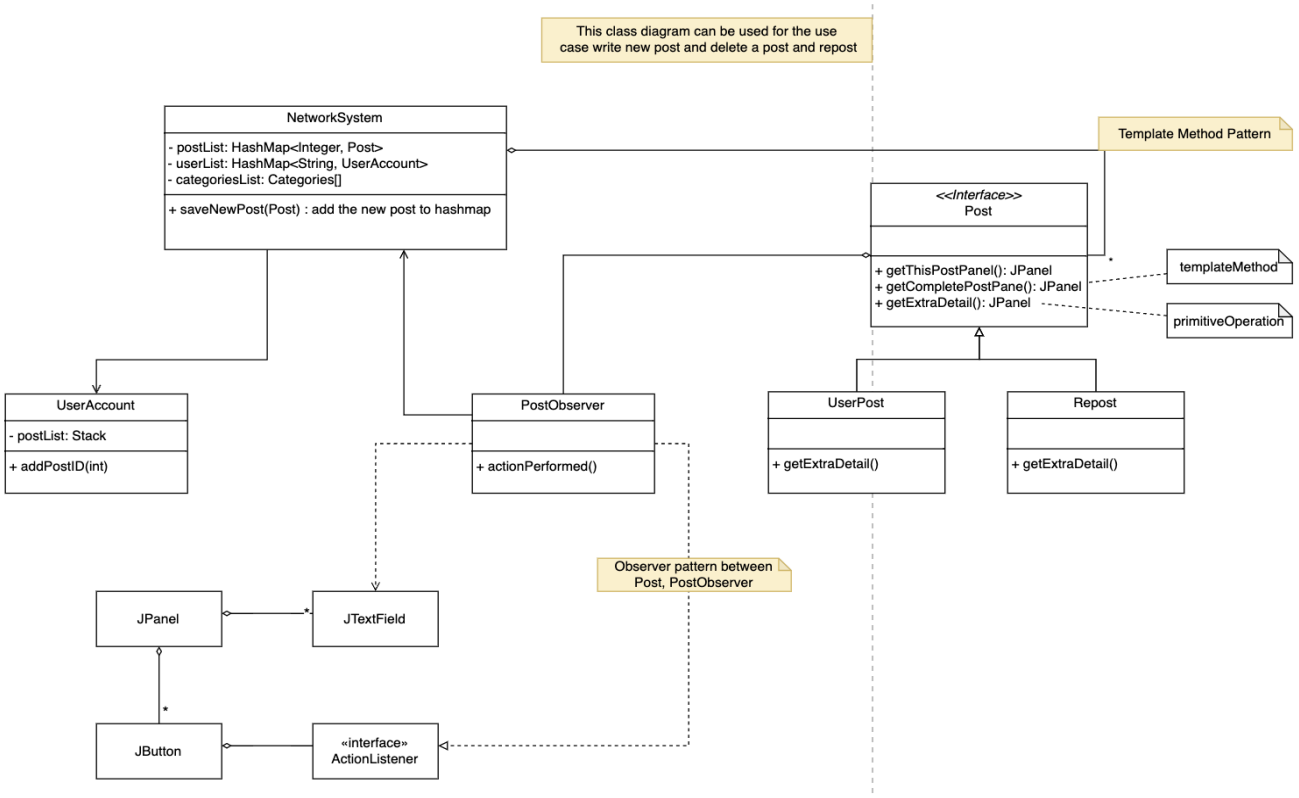
Singleton Pattern - Mapping table:

Name in Design Pattern	**Actual name**
Singleton class	NetworkSystem
Instance variable	instance
Instance method	getInstance()

2. Use case: Log in (author Selin)

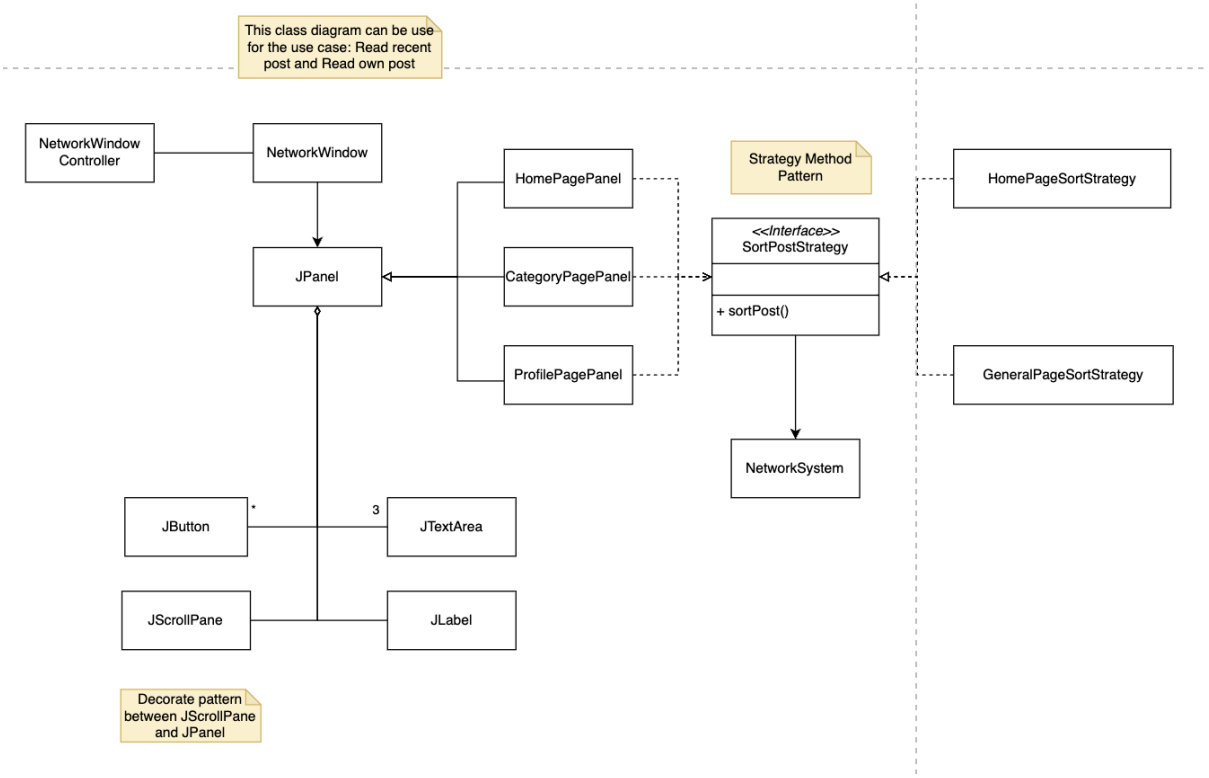


3. Use case: Write new post and delete a post (author Hieu)

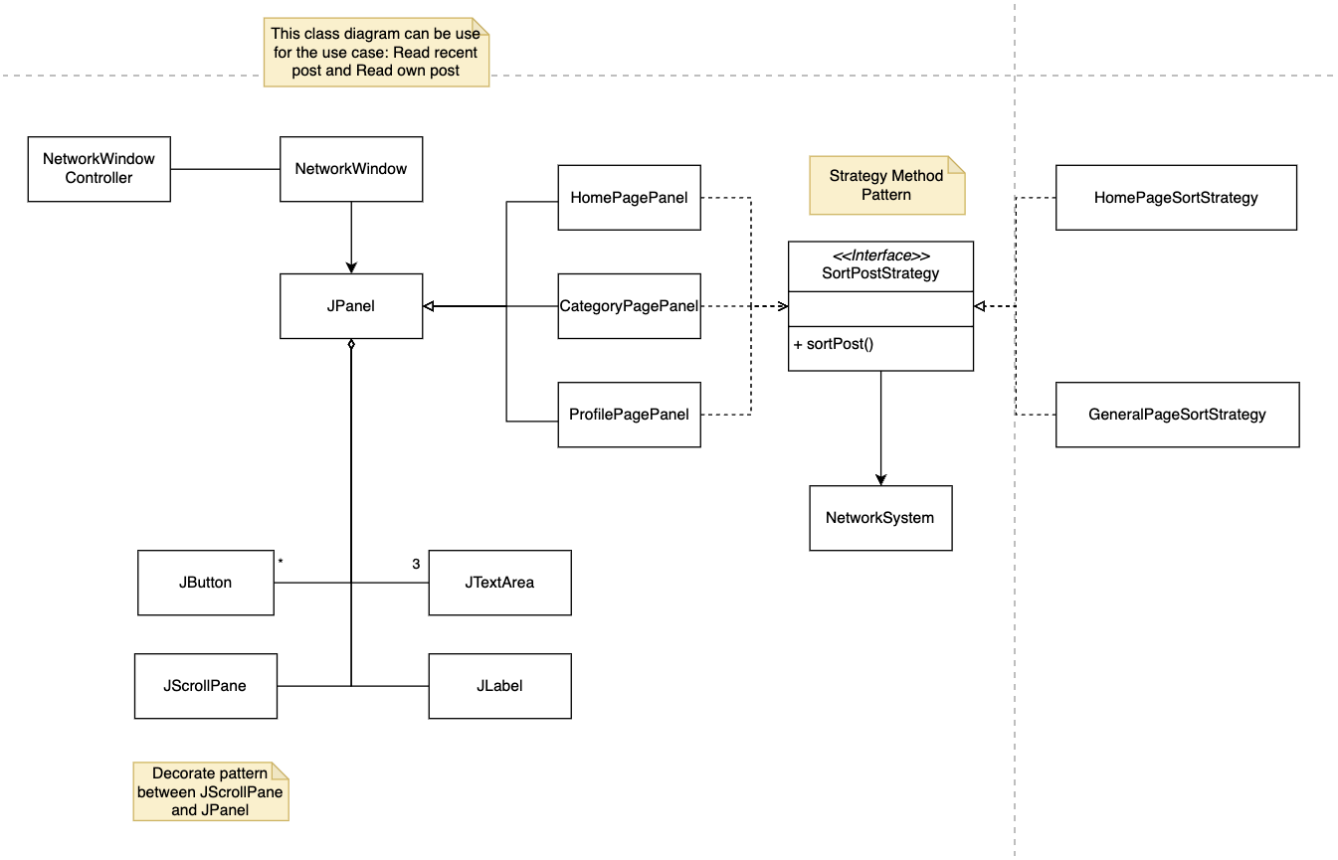


Name in Template	Actual name
AbstractClass	Post
ConcreteClass	UserPost, Repost
templateMethod()	getCompletePostPanel()
primitiveOp1()	getExtraDetail()

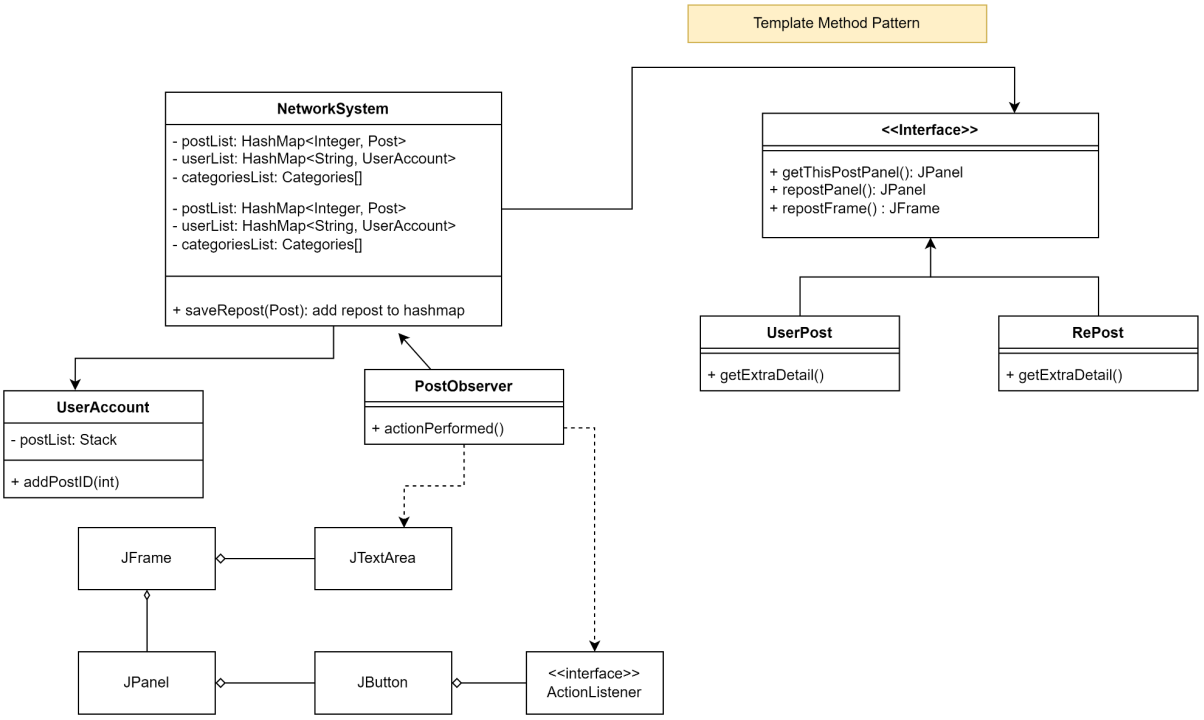
4. Use case: Read recent post (author Hieu)



5. Use case: Read user own post (author Hieu)



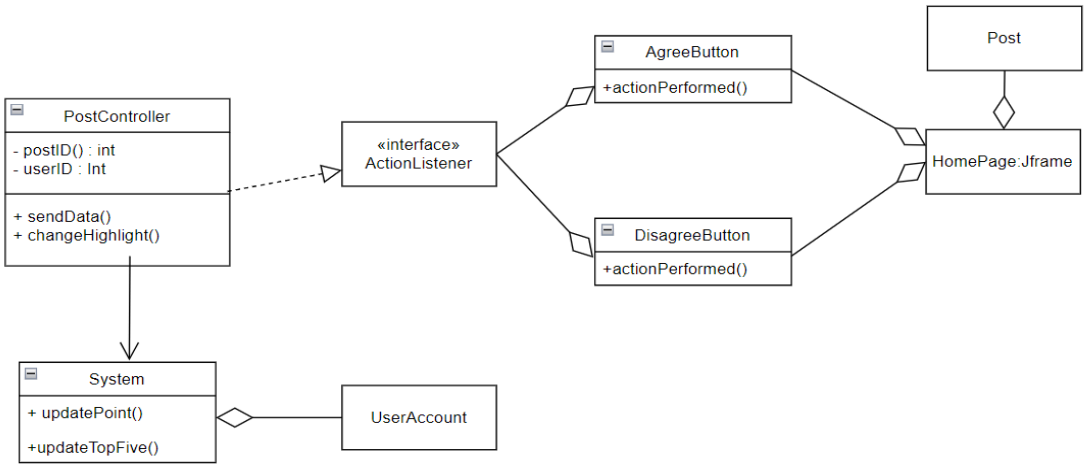
6. Use Case: Repost (author Selin)



Template Pattern

Name in Design Pattern	**Actual name**
AbstractClass	Post
ConcreteClass	UserPost, Repost
templateMethod()	getCompletePostPanel()
primitiveOp1()	getExtraDetail()

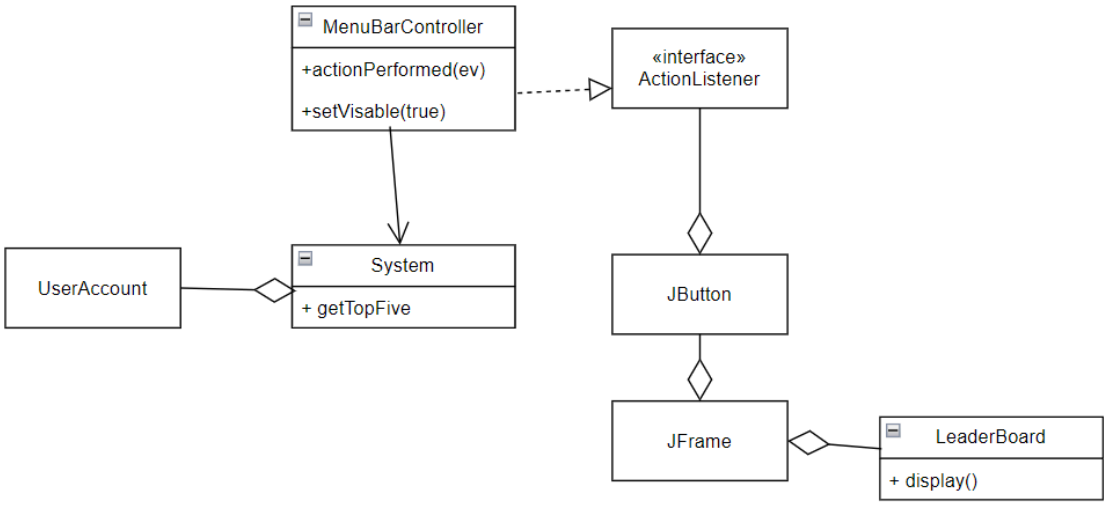
6. Use case: Agree/disagree post (author Lewis)



template pattern

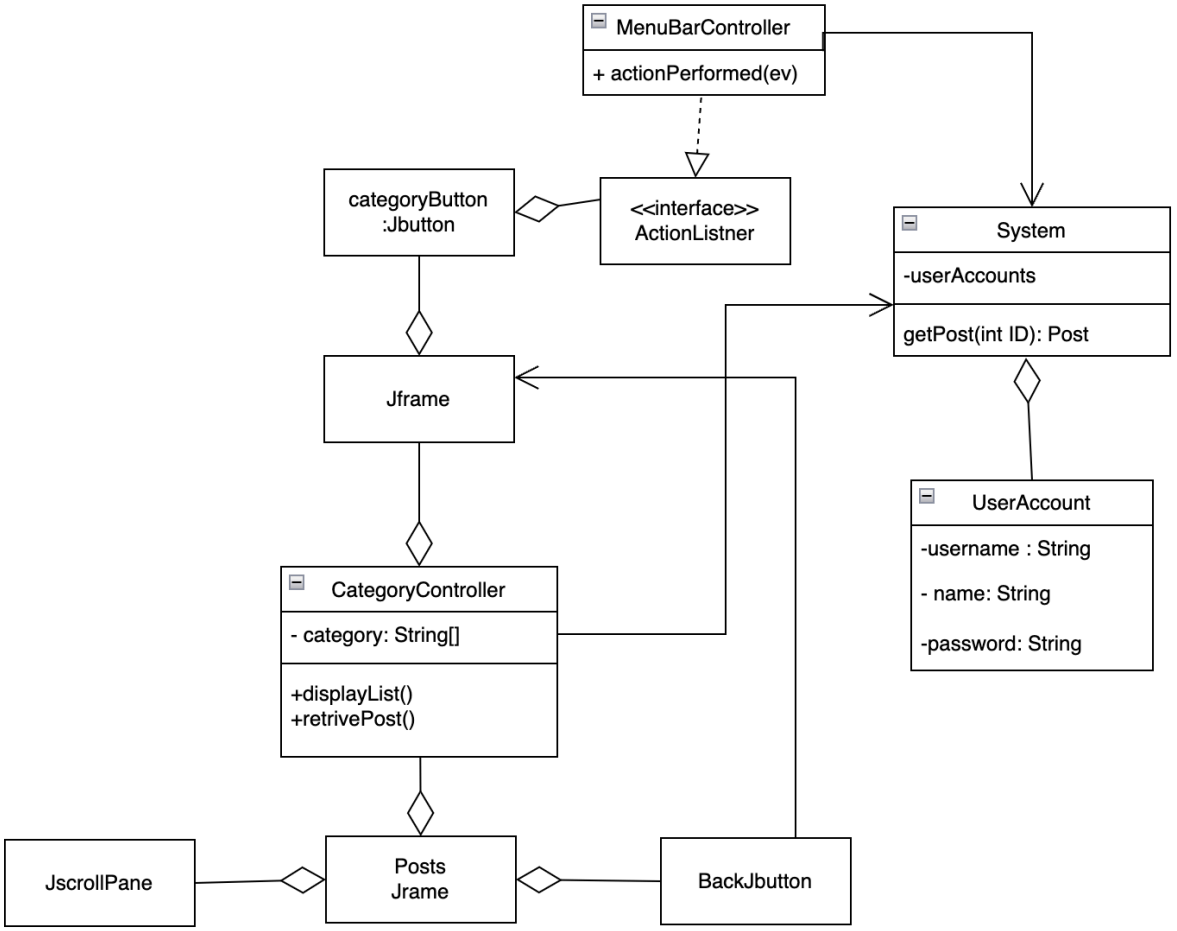
Name in Design pattern	Actual Name
AbstractClass	Post
Concreteclass	Agree, Disagree,
templatemethod	updatepoints

7. Use case: Check leaderboard (author Lewis)



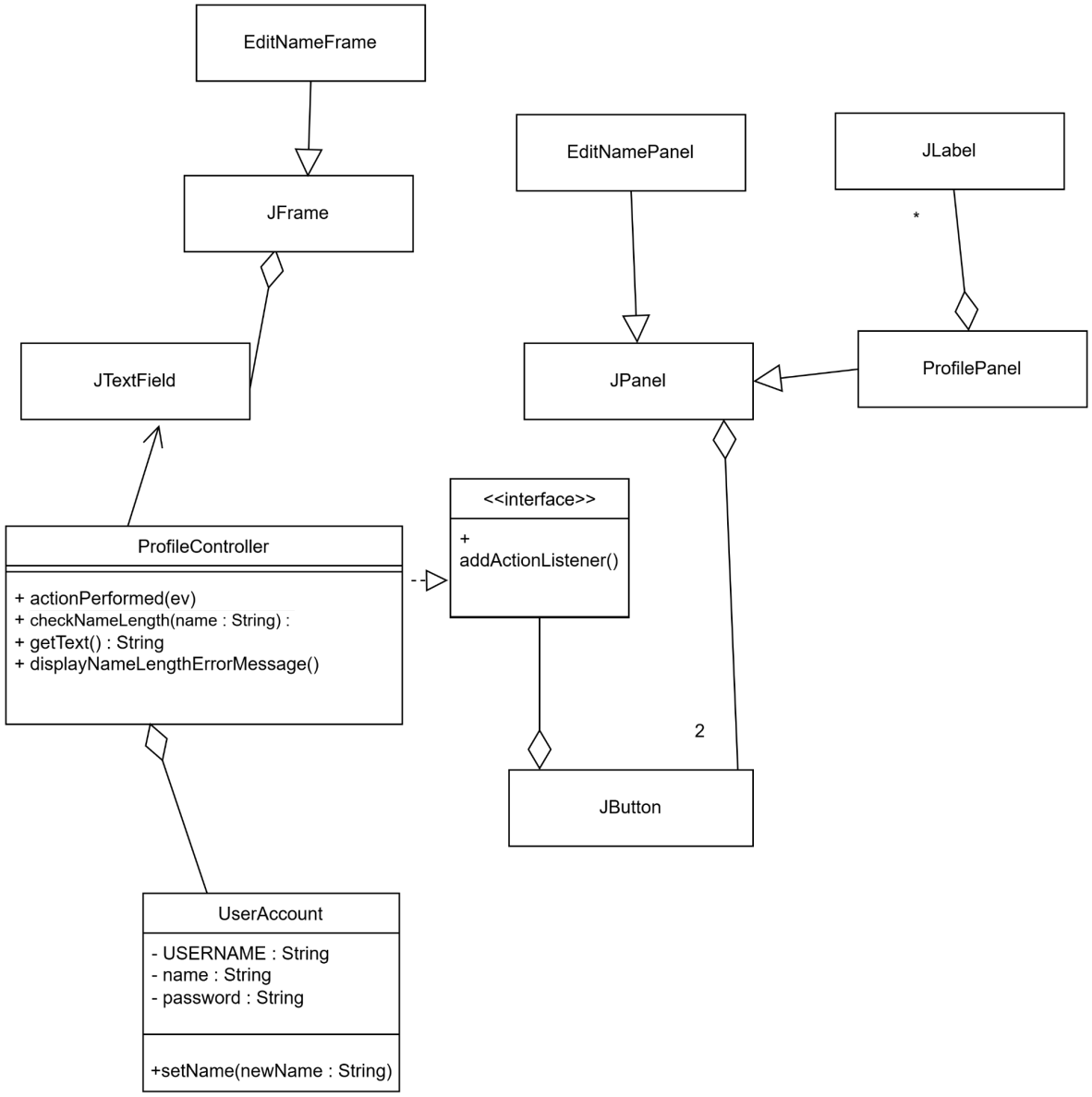
Name in Design Pattern	Actual Name
subject	LeaderBoard
observer	ActionListener
concrete observer	Menu Bar Controller

8. Use case: Find topic using category (author Lewis)



Name in Design Pattern	**Actual name**
Component	CategoryController
Composite	CategoryPage
Leaf	Post
Method	display list(), get post()

9. Use case: Edit profile name (author Tabbie)

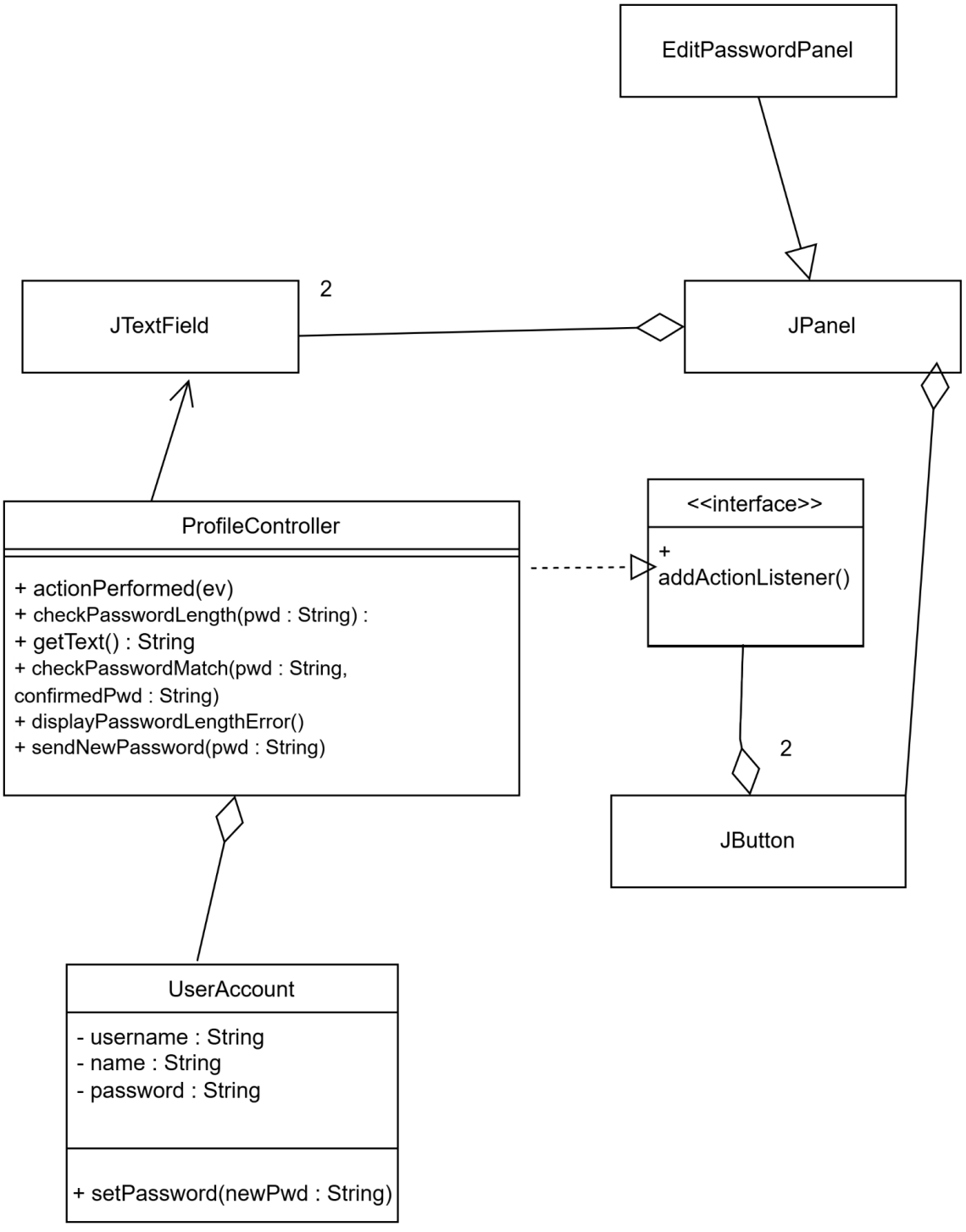


Observer Pattern - Mapping table:

Name in Design Pattern	**Actual name**
Subject	JButton
Observer	ActionListener
ConcreteObserver	LoginController
attach()	addActionListener

notify()	actionPerformed

10. Use case: Edit profile password(author Tabbie)

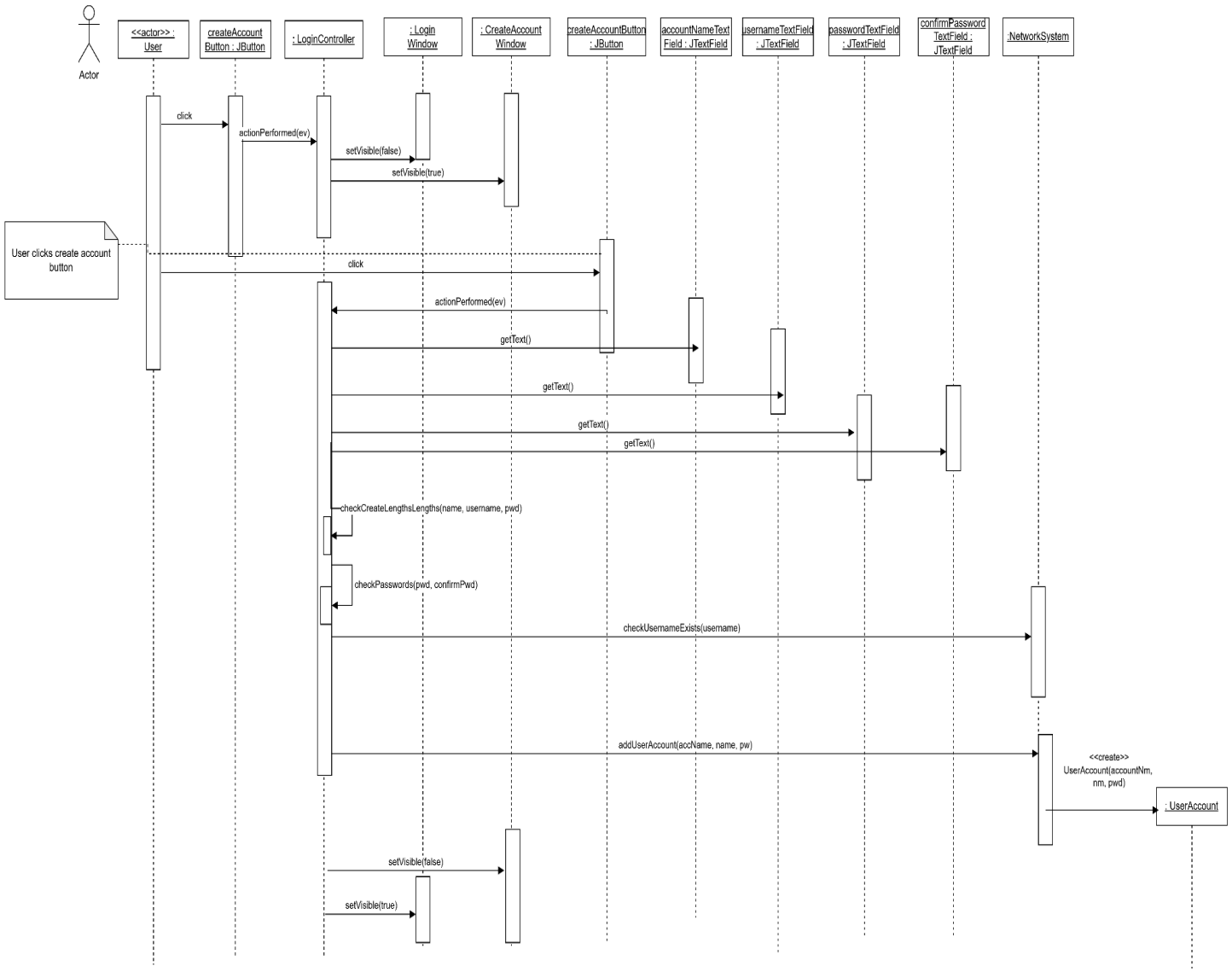


Observer Pattern - Mapping table:

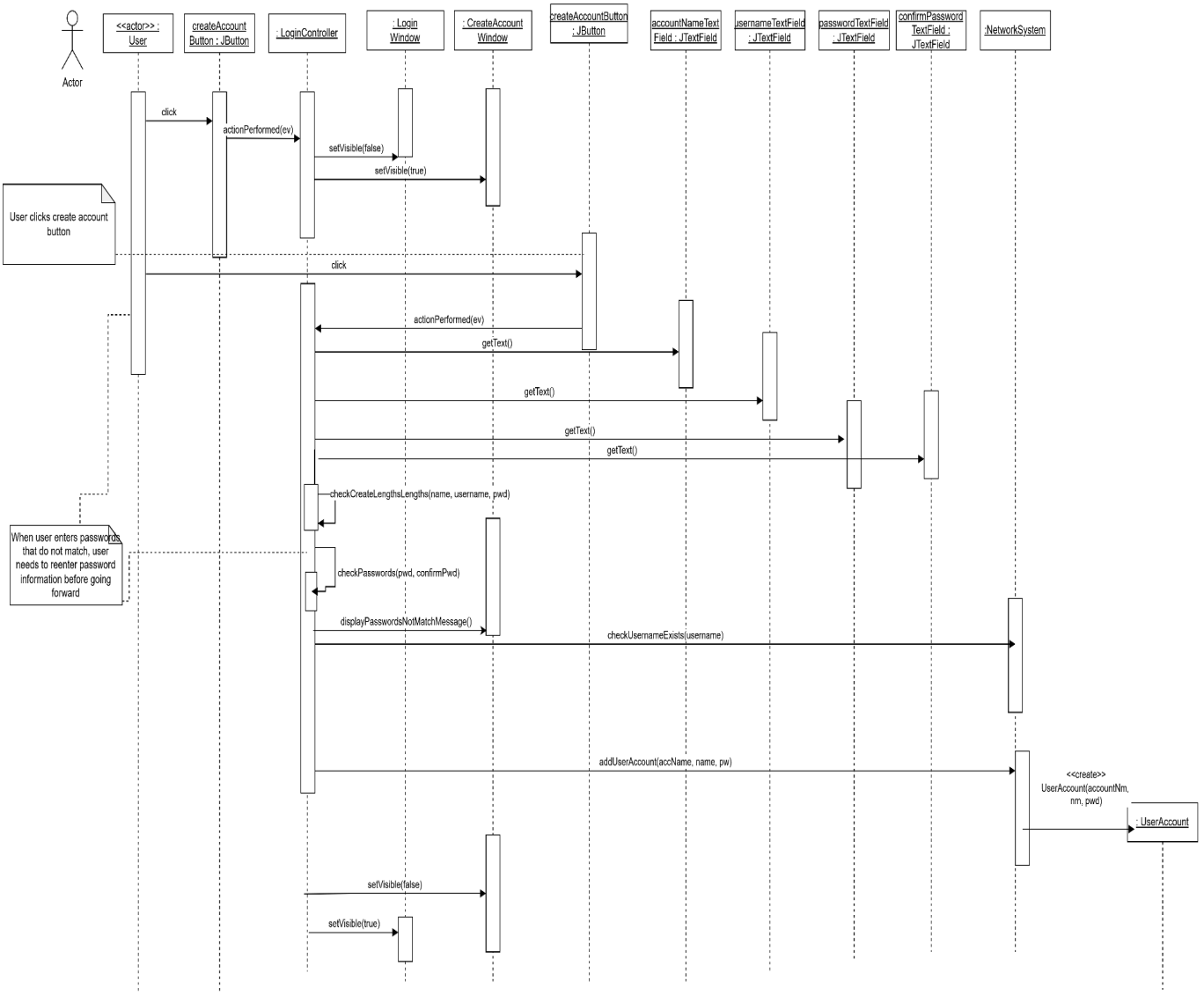
Name in Design Pattern	**Actual name**
Subject	JButton
Observer	ActionListener
ConcreteObserver	LoginController
attach()	addActionListener
notify()	actionPerformed

C. Sequence diagrams

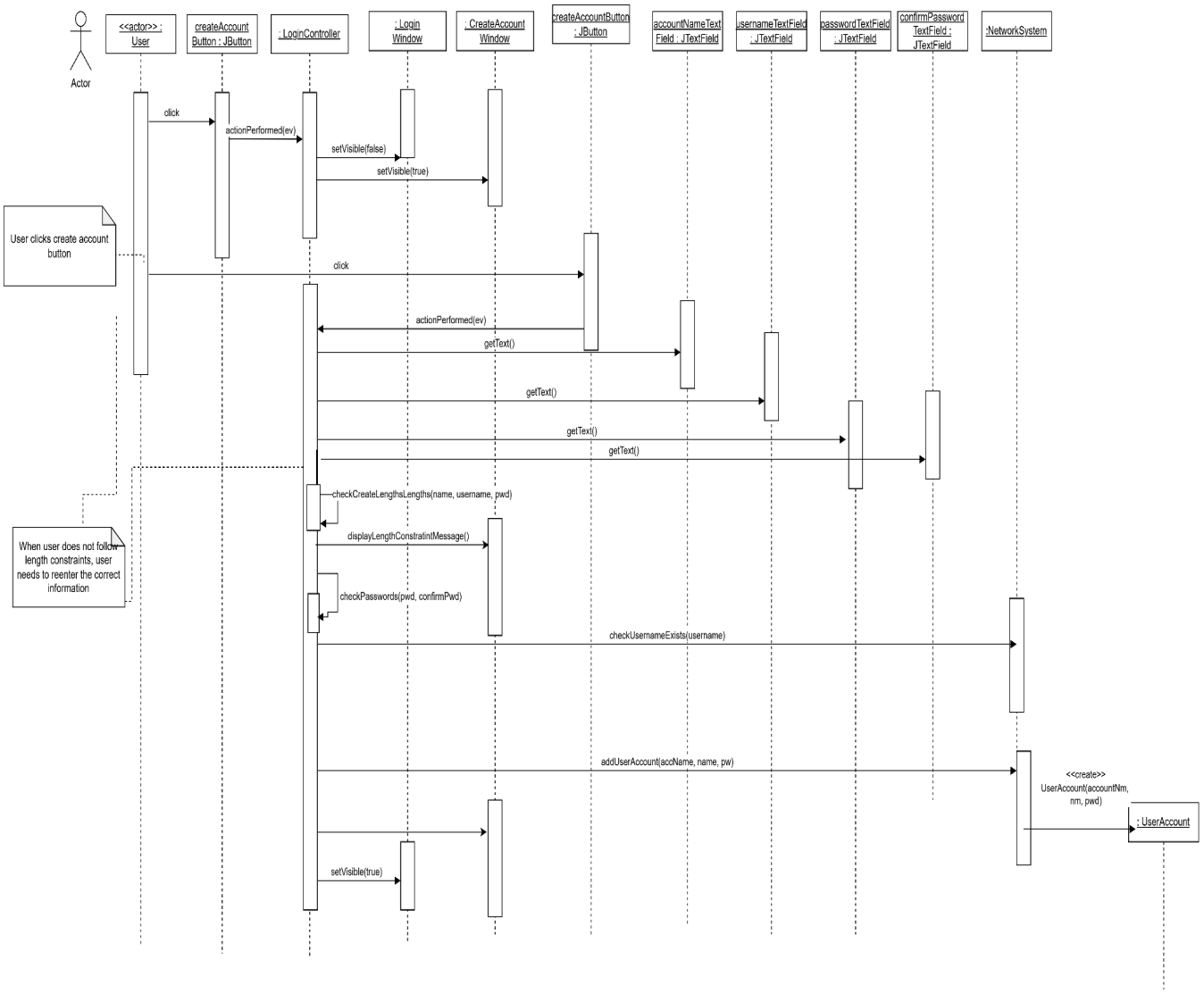
1. Use case: Create account (author Tabbie)



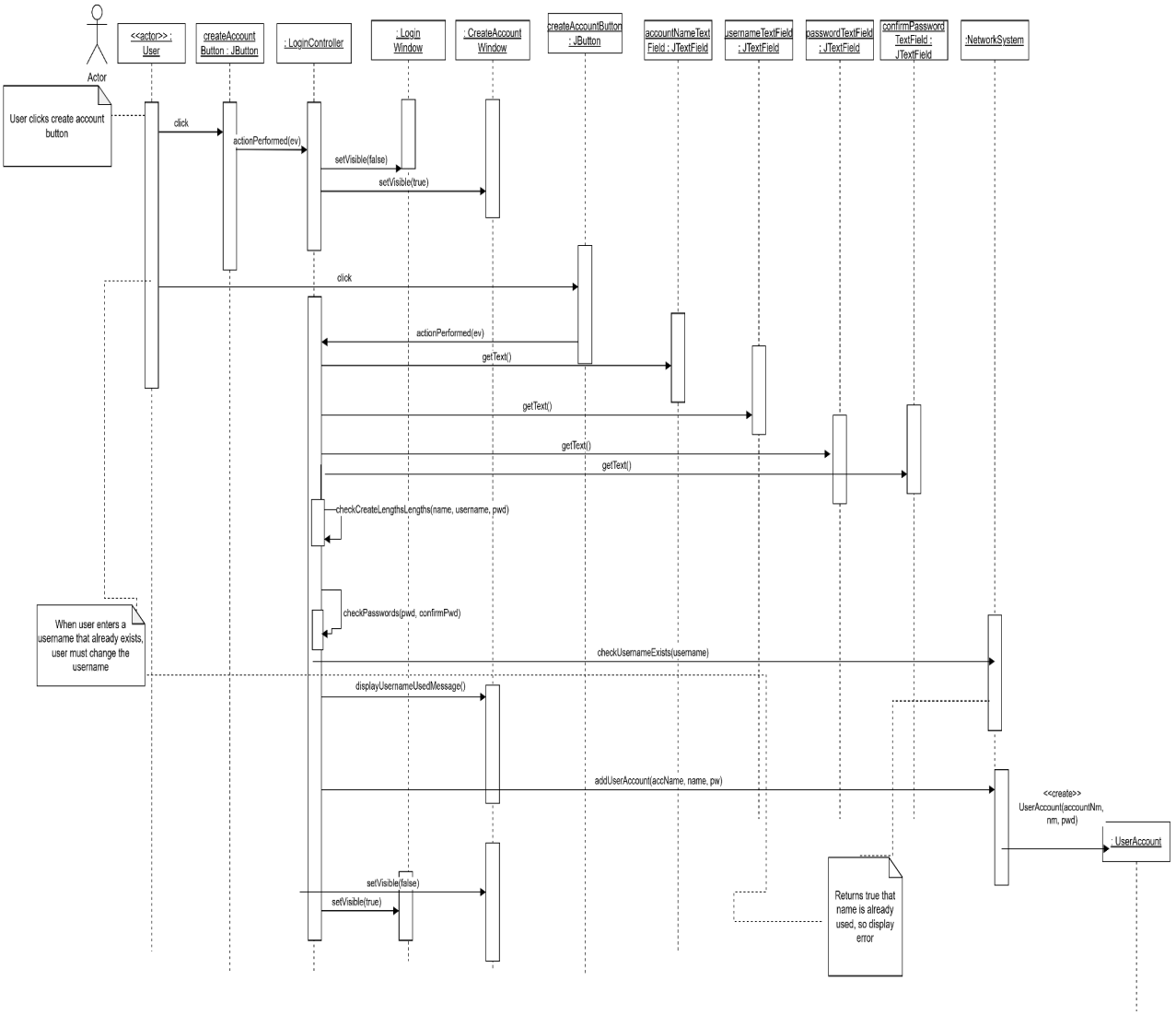
- Variation #1: Passwords do not match



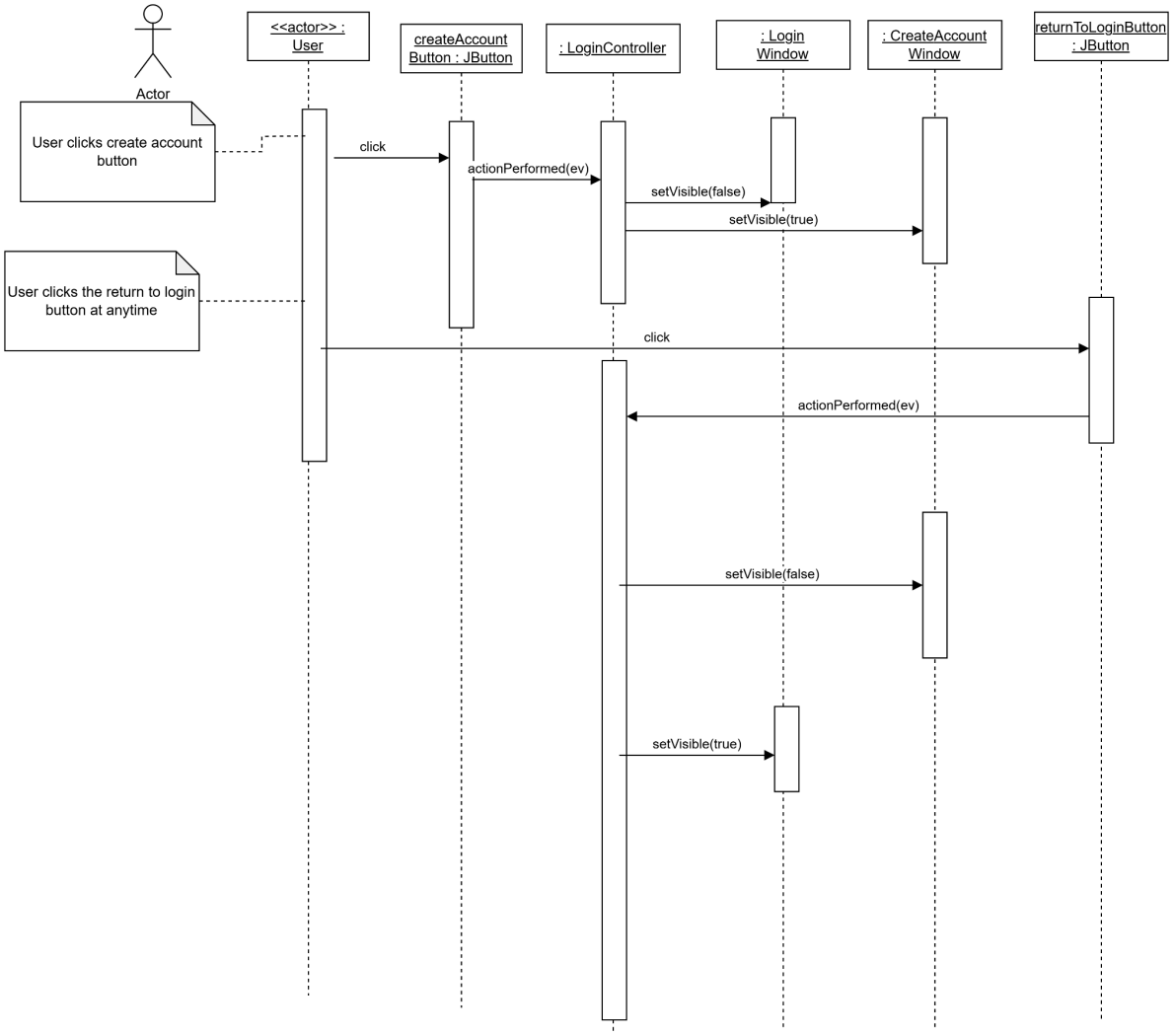
- Variation #2: Length constraint on name and/or password



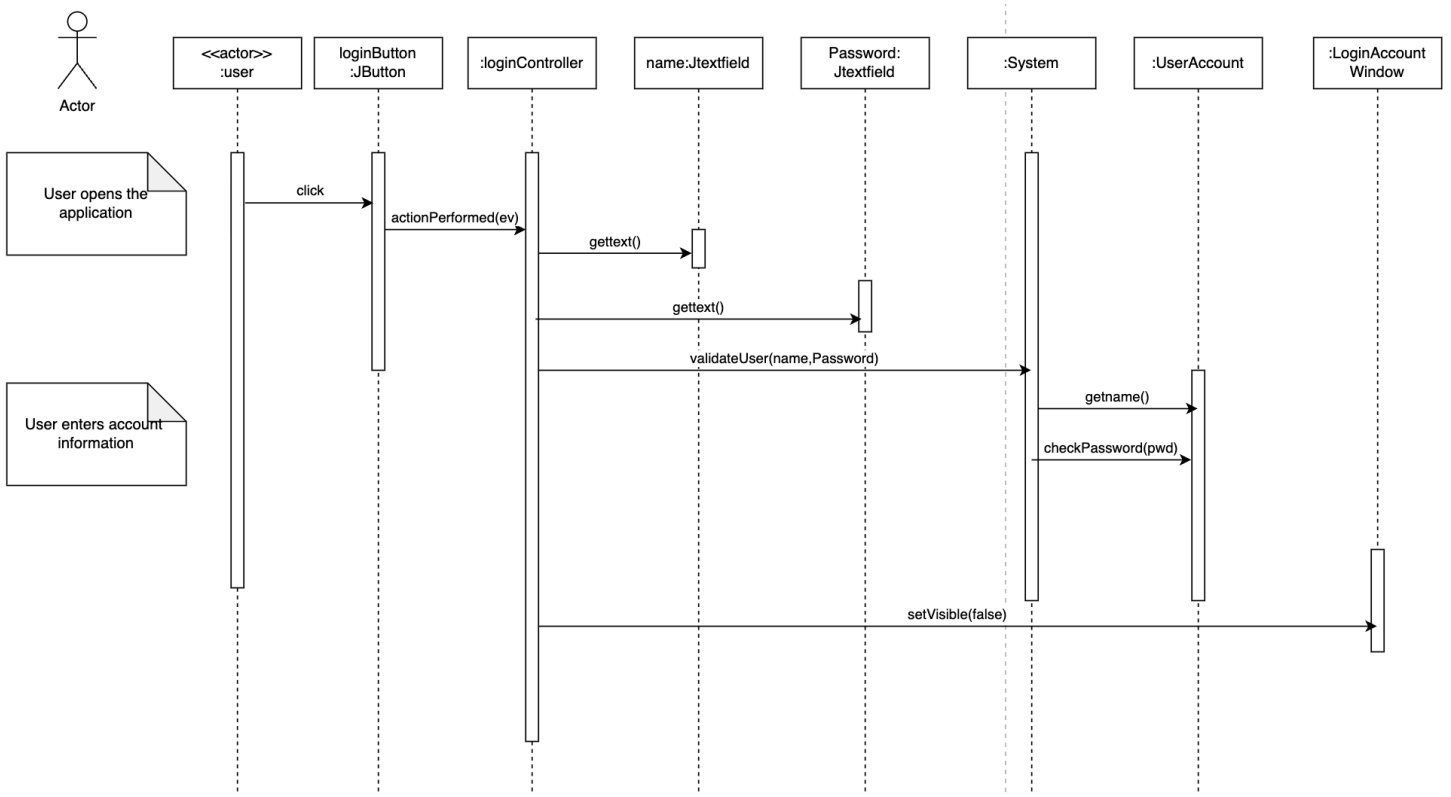
- Variation #3: Username already used



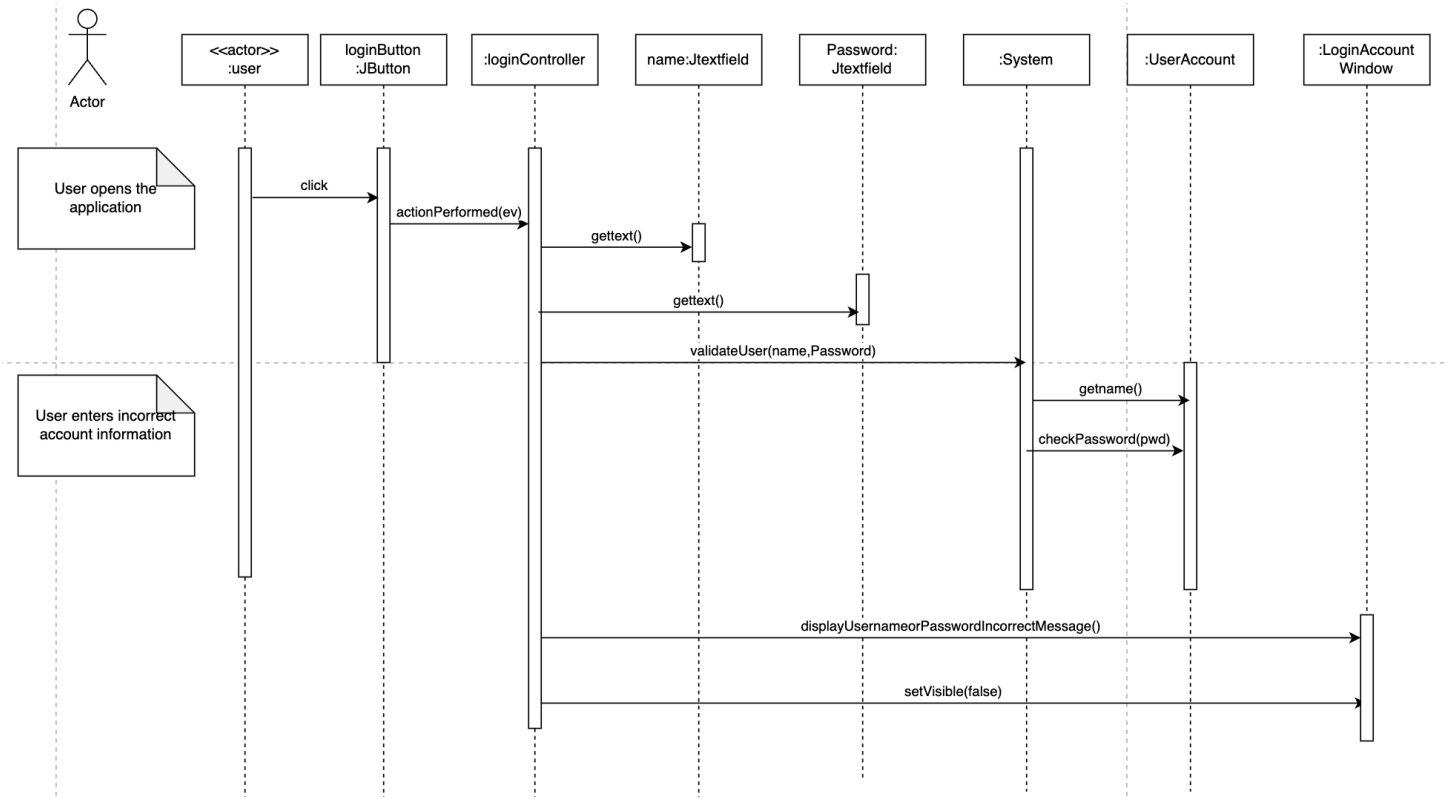
- Variation #4: User selects return to login button



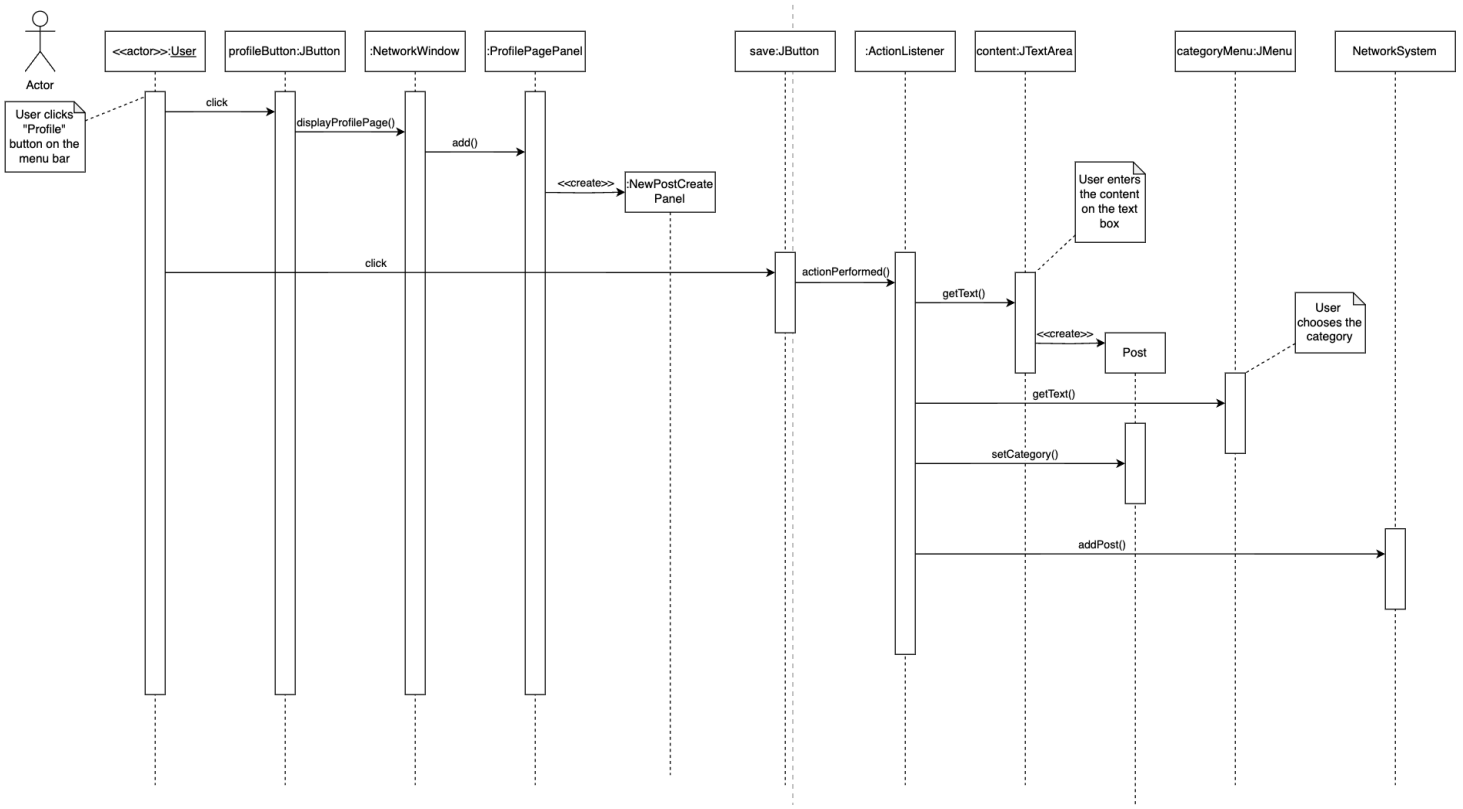
2. Use case: Log in (author Selin)



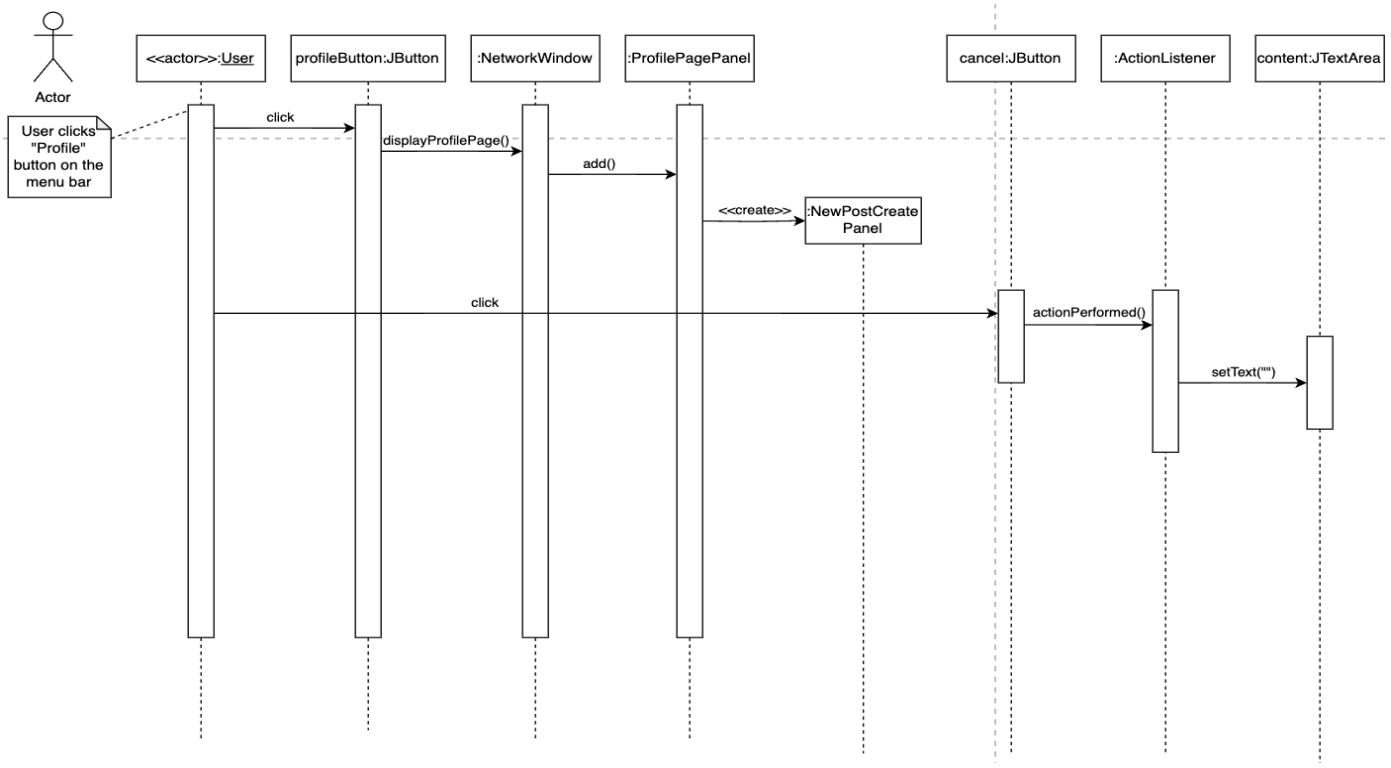
- Variation #1: Incorrect password or username



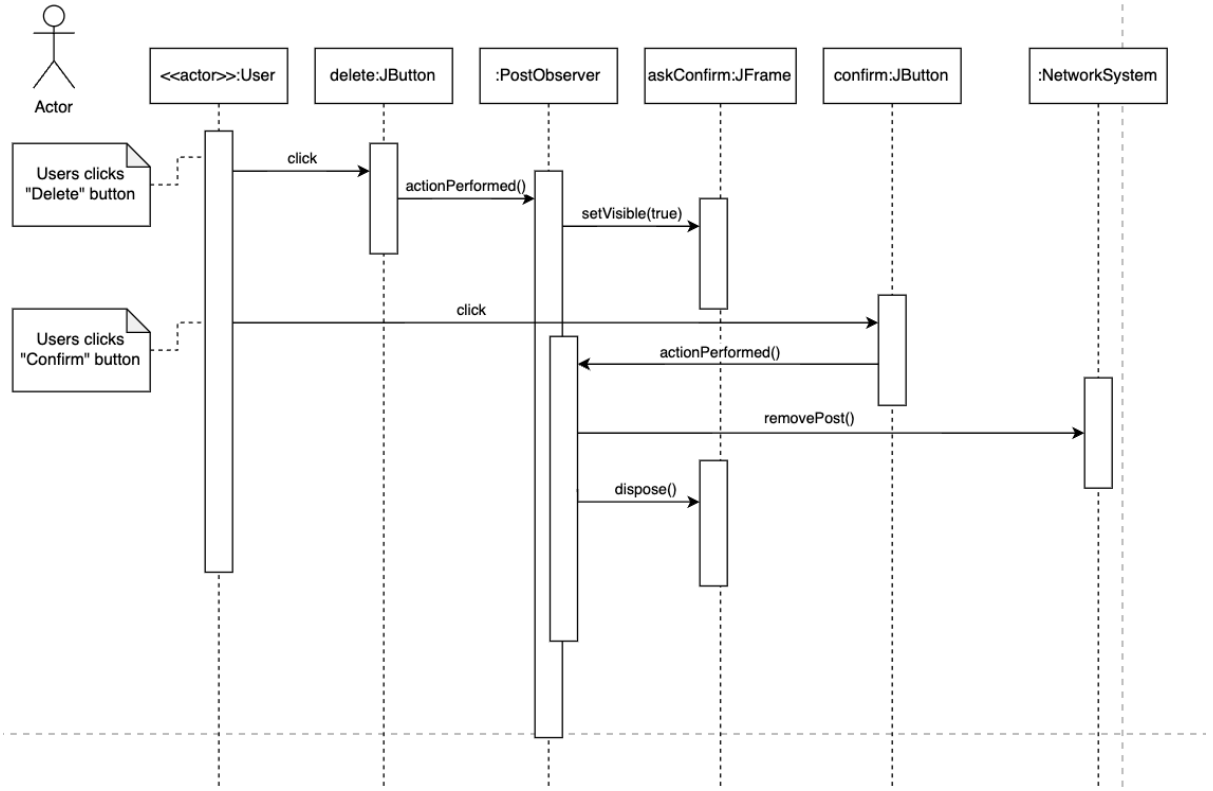
3. Use case: Write new post (author Hieu)



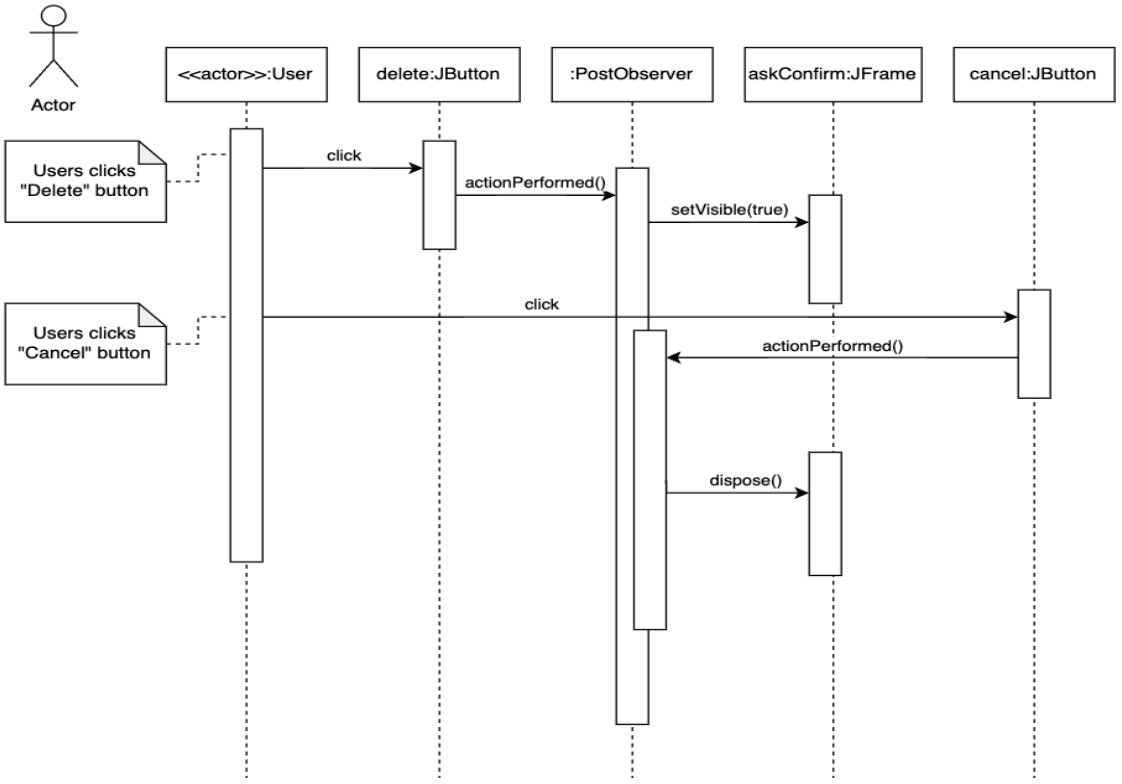
- Variation #1: Cancel the creating



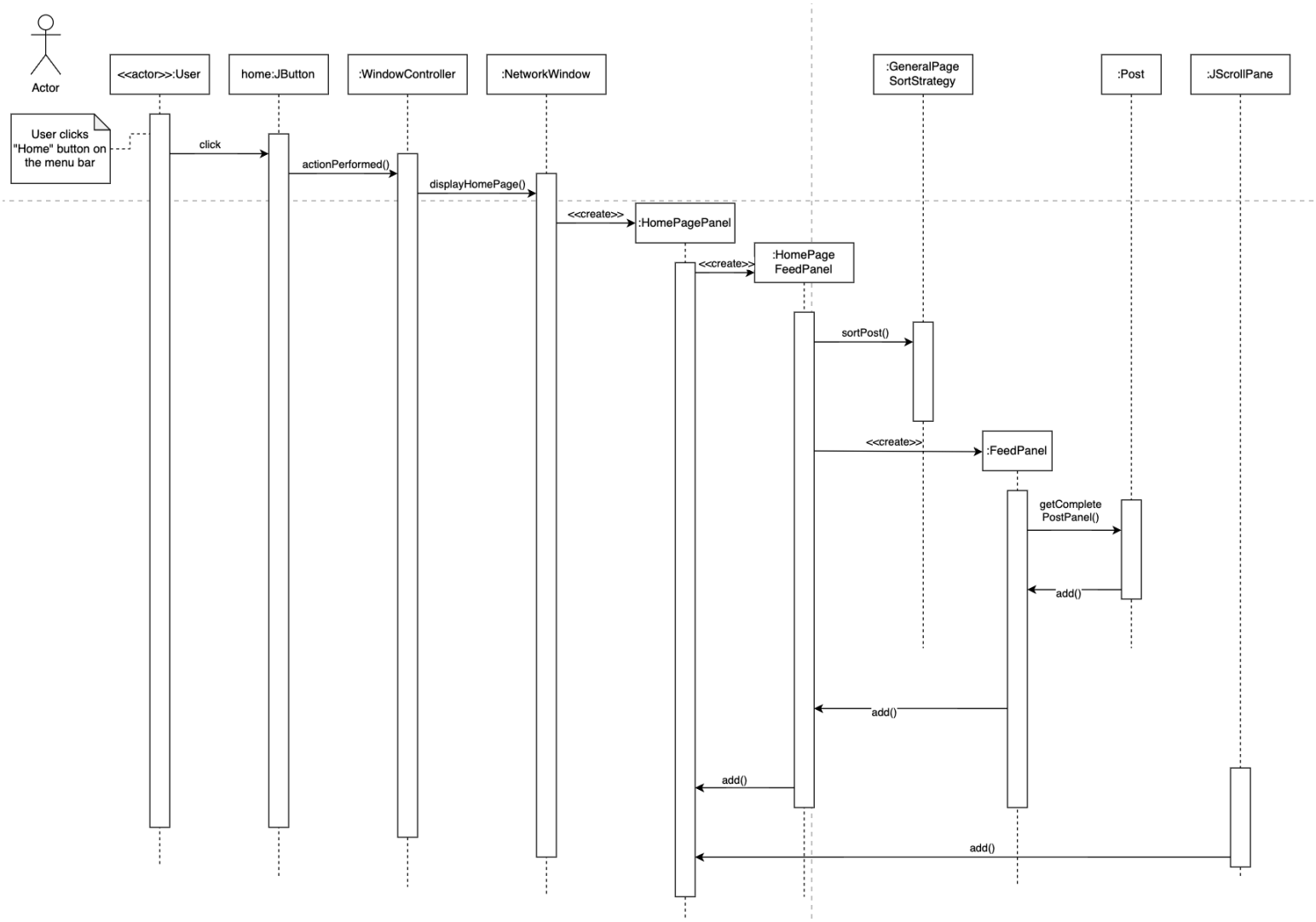
4. Use case: Delete a post (author Hieu)



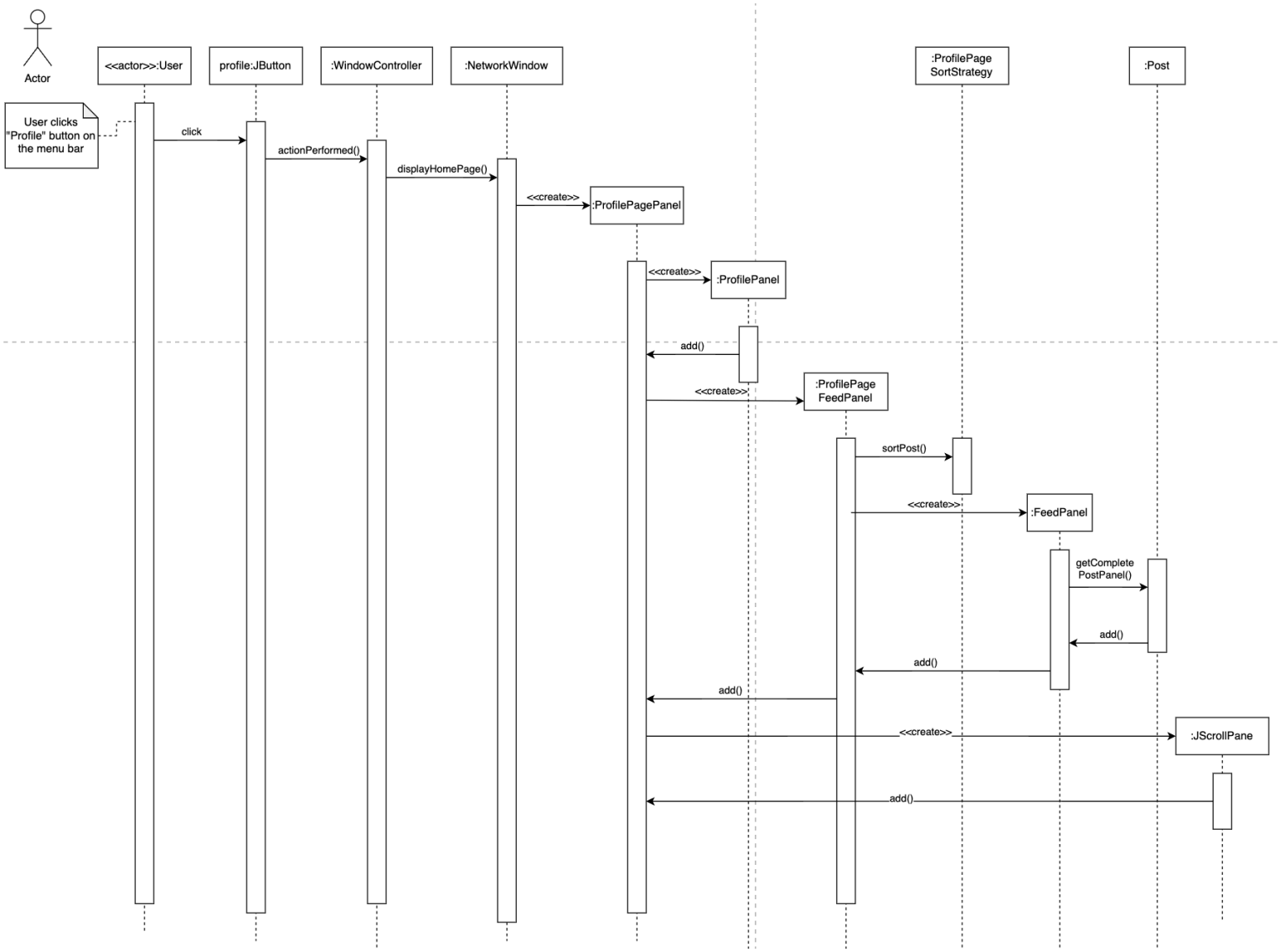
- Variation #1: Cancel deletion



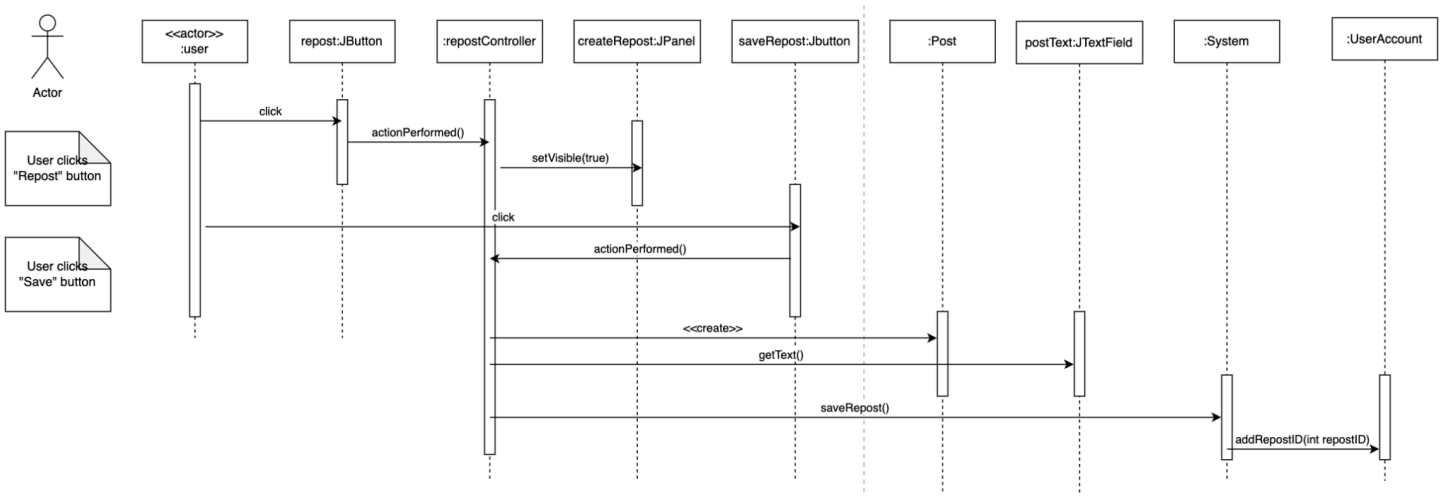
5. Use case: Read recent post (author Hieu)



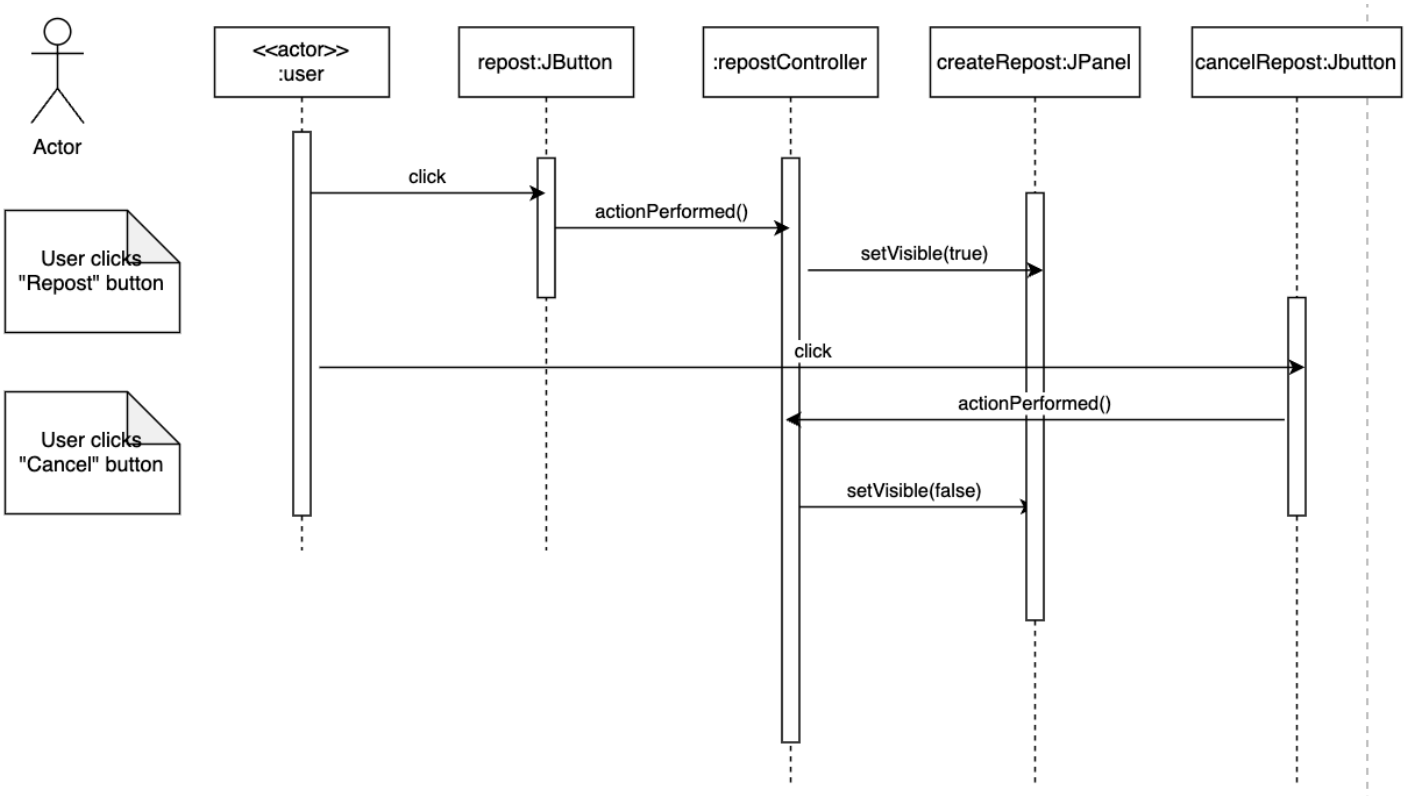
6. Use case: Read user own post (author Hieu)



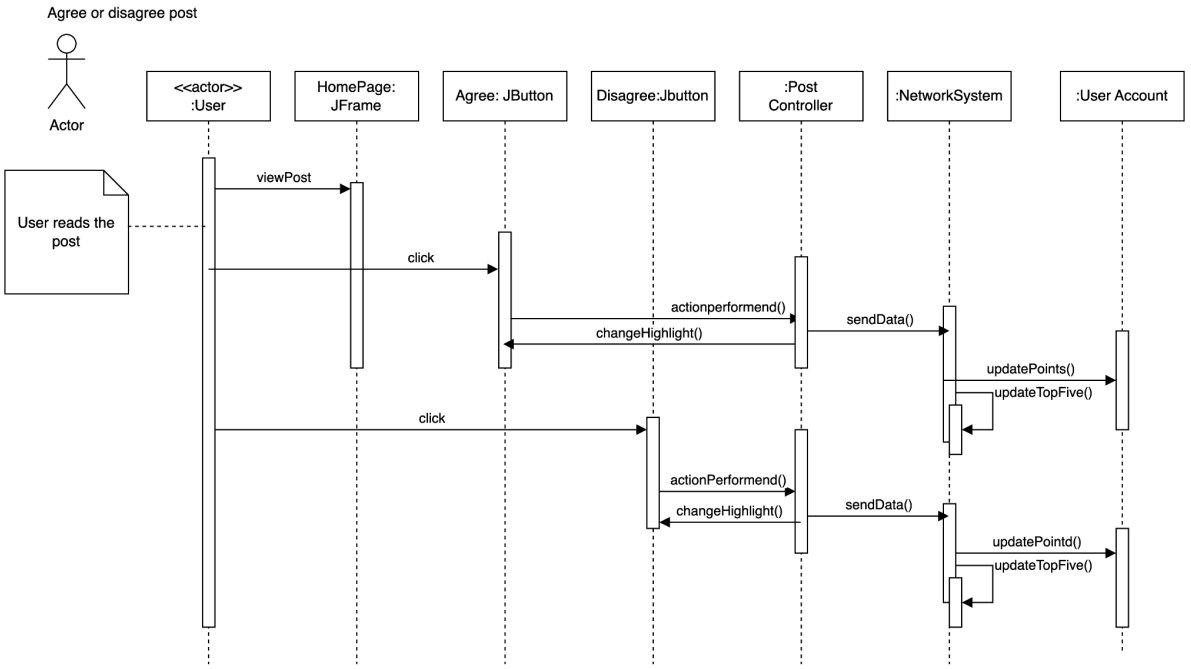
7. Use case: Repost (author Selin)



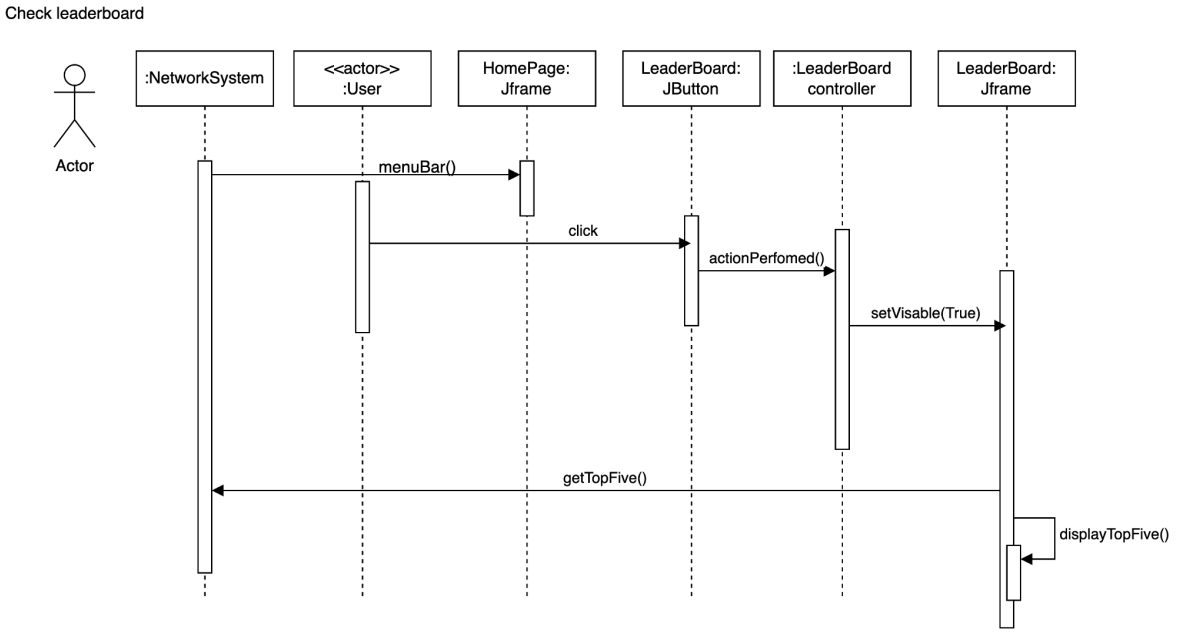
- Variation #1: Cancel repost



8. Use case: Agree/disagree post (author Lewis)

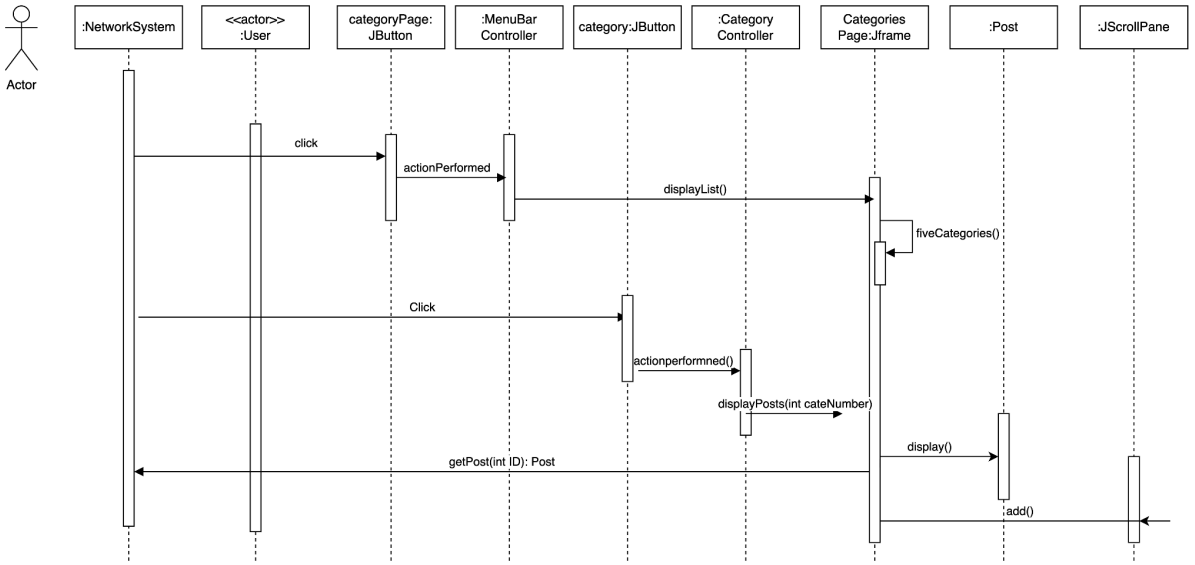


9. Use case: Check leaderboard (author Lewis)

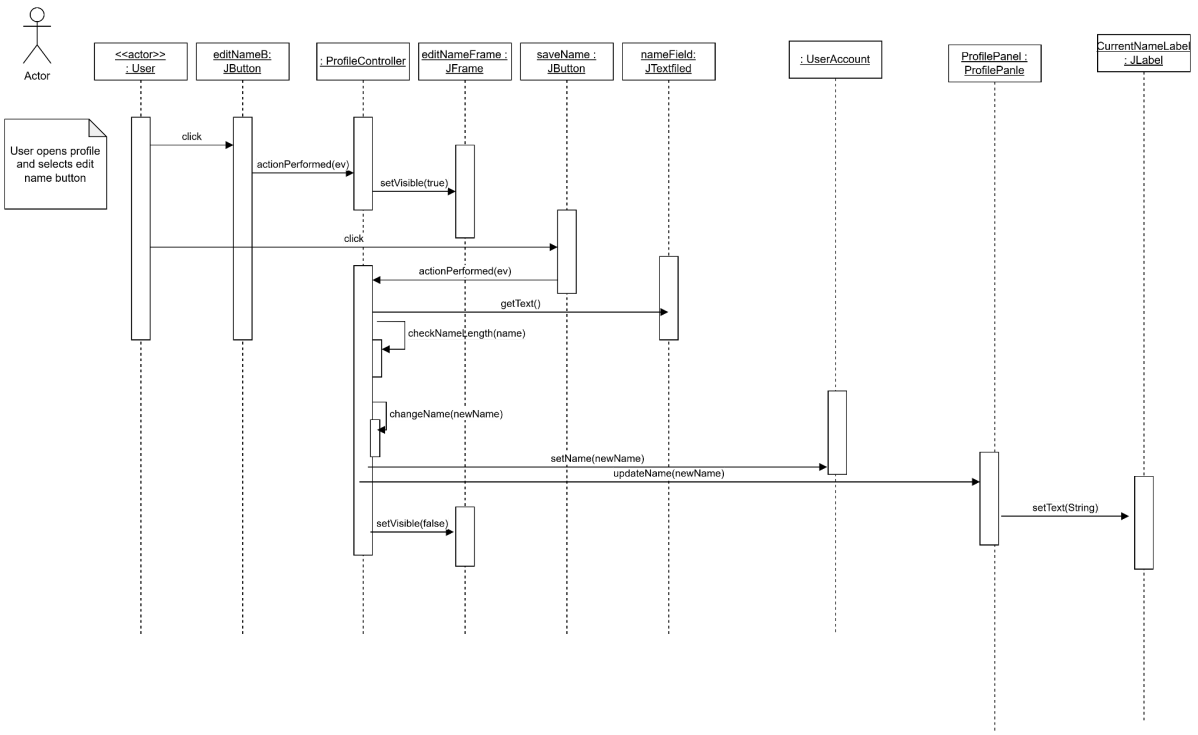


10. Use case: Find topic using category (author Lewis)

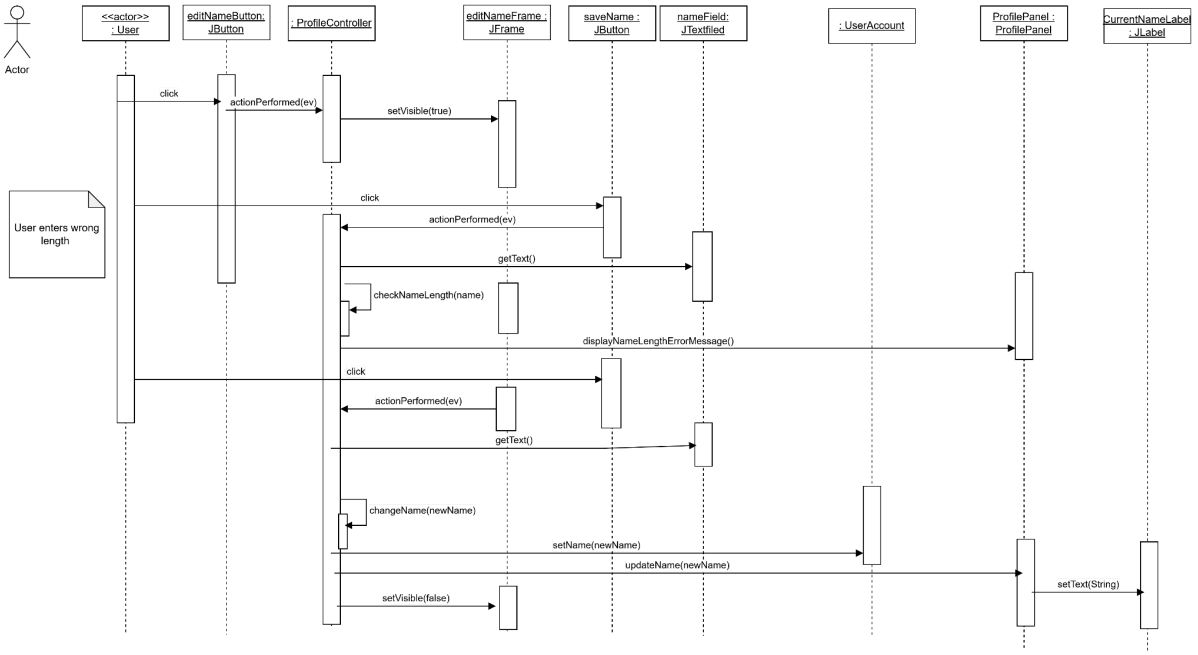
Find topic using categories



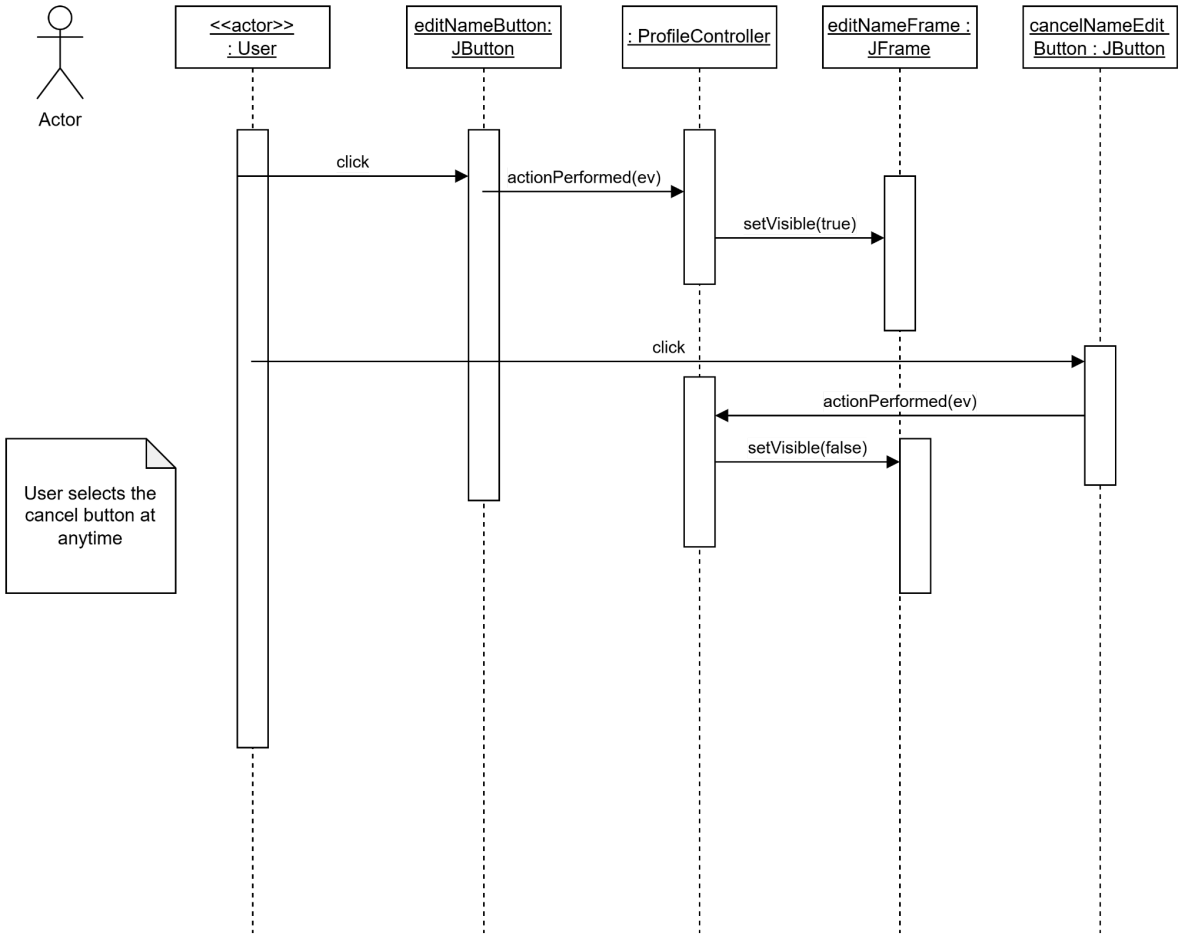
11. Use case: Edit profile name (author Tabbie)



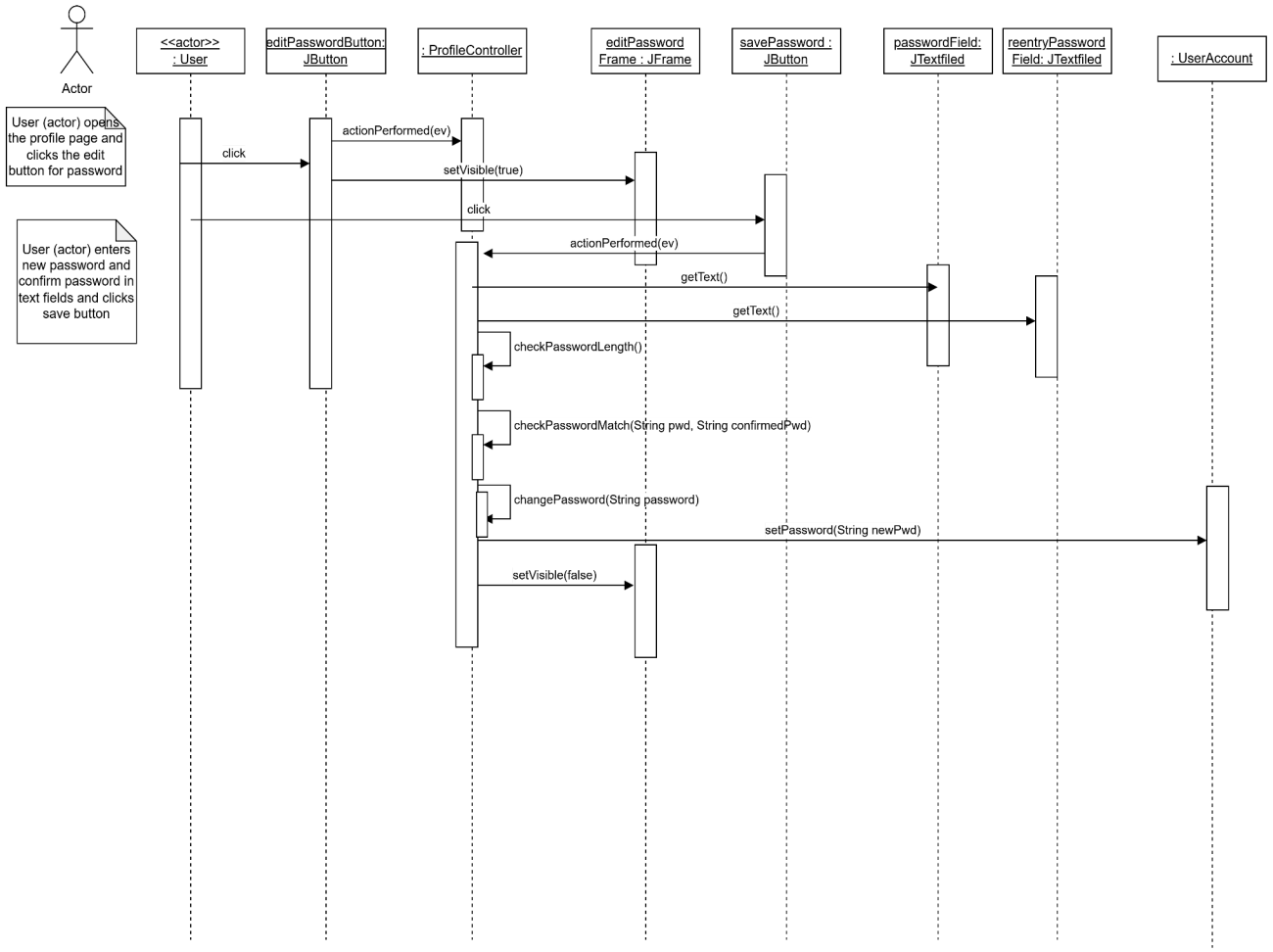
- Variation #1: Length constraint for name met



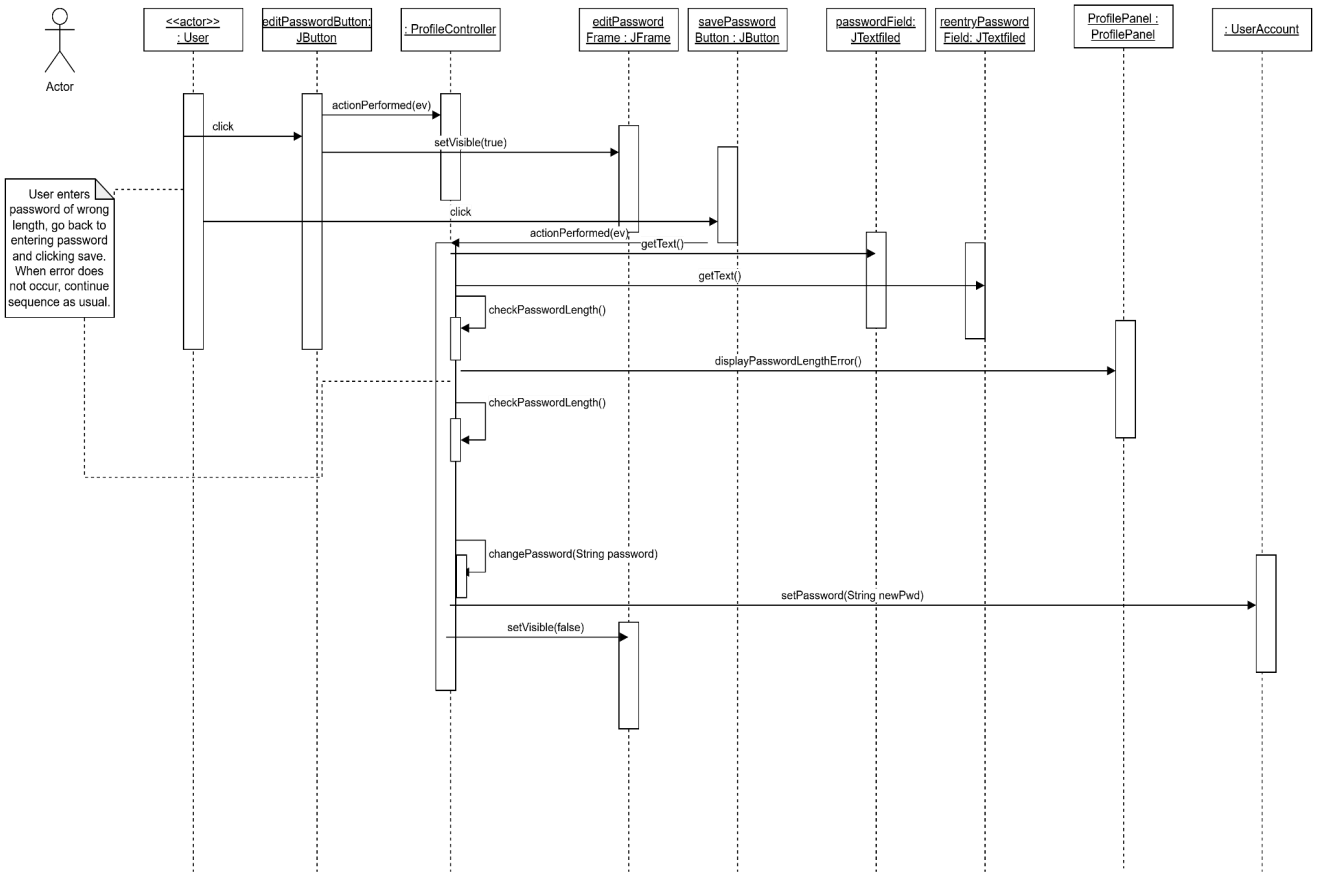
- Variation #2: User selects cancel button


Actor
<<actor>>
: User
editNameButton:
JButton
: ProfileController
editNameFrame :
JFrame
cancelNameEdit
Button : JButton
click
actionPerformed(ev)
setVisible(true)
click
actionPerformed(ev)
setVisible(false)
User selects the
cancel button at
anytime

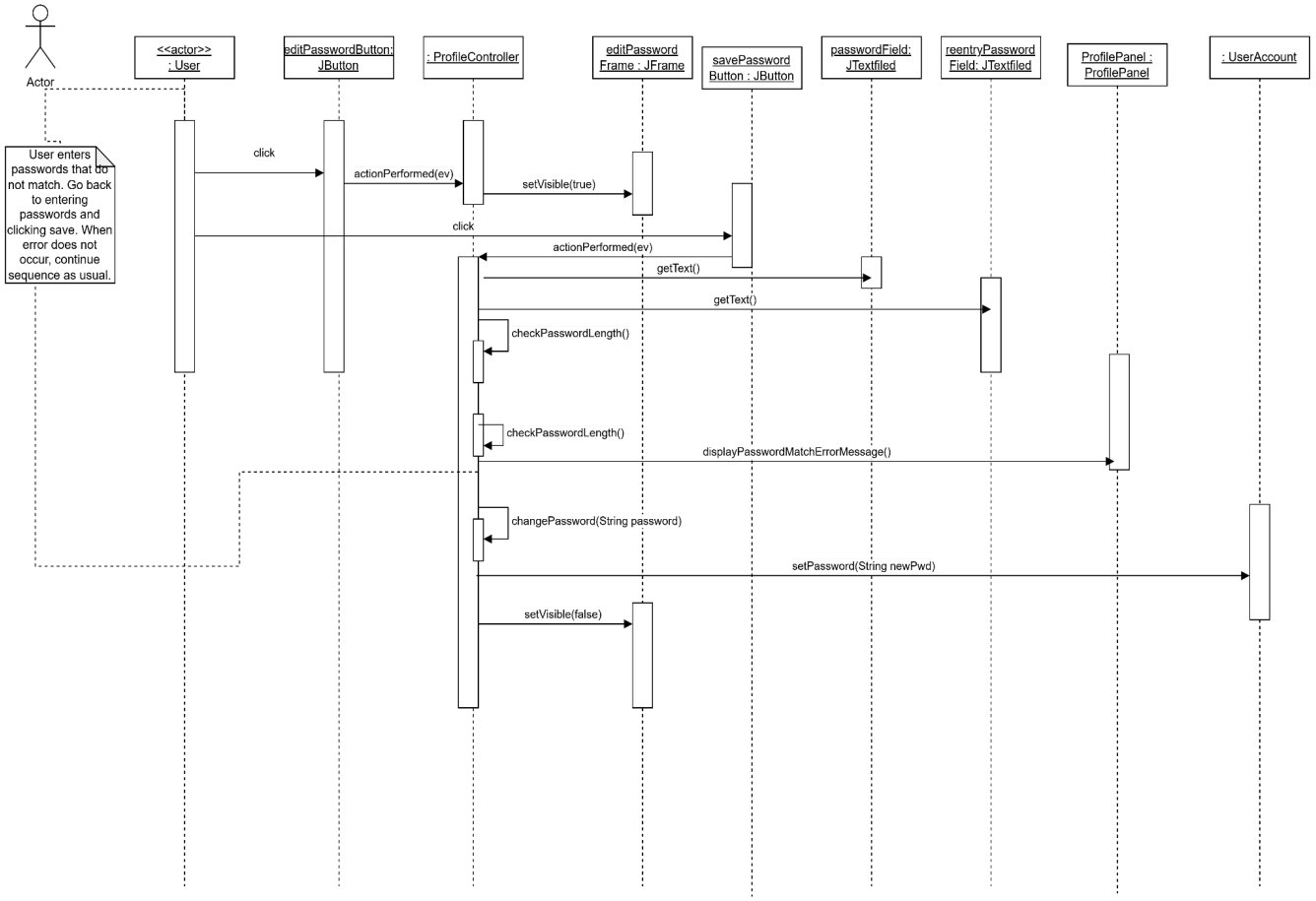
12. Use case: Edit profile password (author Tabbie)



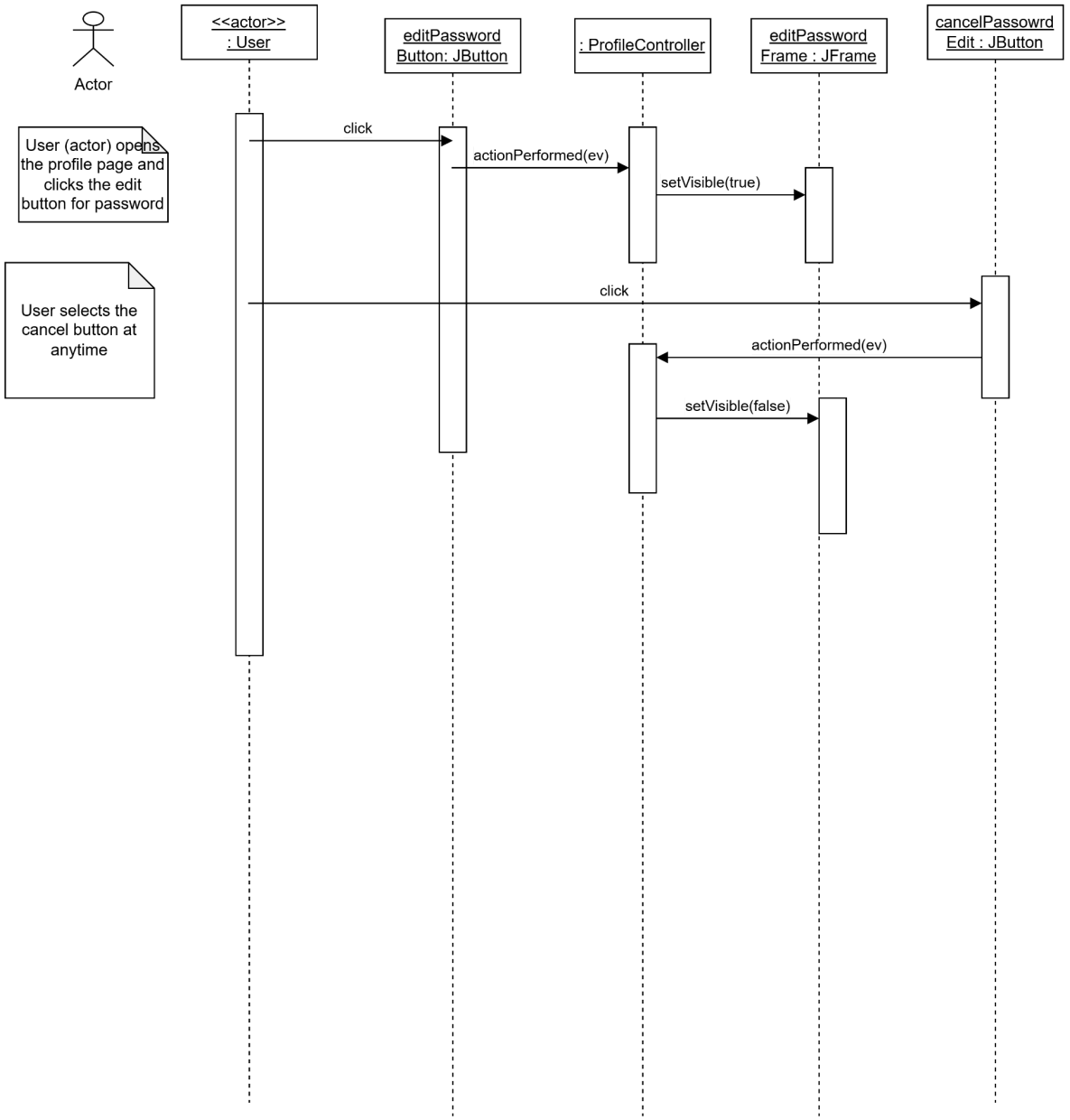
- Variation #1: Password length constraint met


Actor
<<actor>>
: User
editPasswordButton:
JButton
: ProfileController
editPassword
Frame : JFrame
savePassword
Button : JButton
passwordField:
JTextfiled
reentryPassword
Field: JTextfiled
ProfilePanel :
ProfilePanel
: UserAccount
User enters password of wrong length, go back to entering password and clicking save. When error does not occur, continue sequence as usual.
click
actionPerformed(ev)
setVisible(true)
click
actionPerformed(ev)
getText()
getText()
checkPasswordLength()
displayPasswordLengthError()
checkPasswordLength()
changePassword(String password)
setPassword(String newPwd)
setVisible(false)

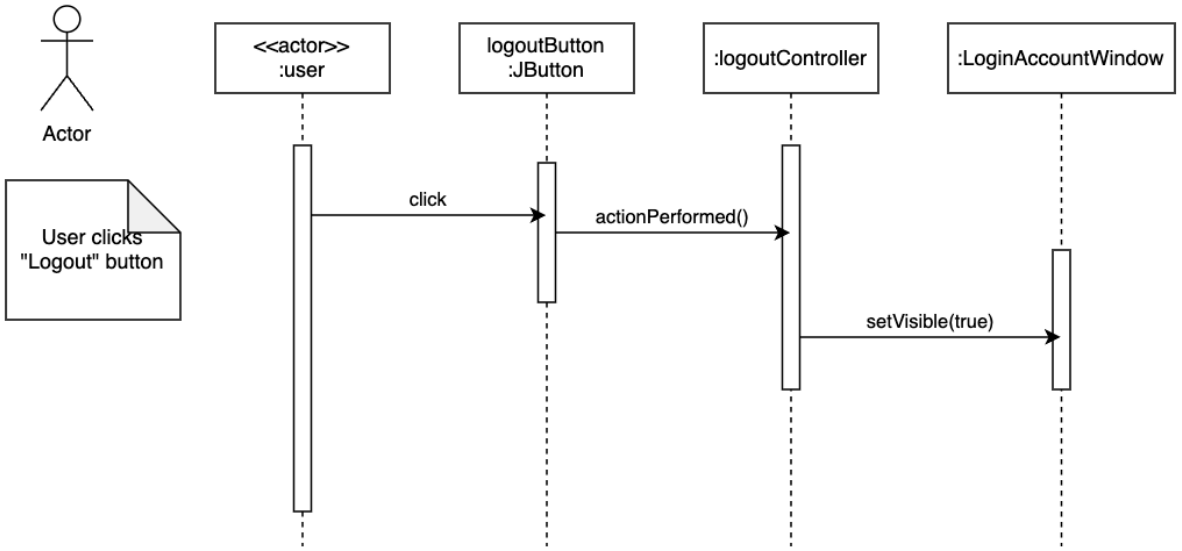
- Variation #2: Passwords do not match



- Variation #3: User selects cancel button

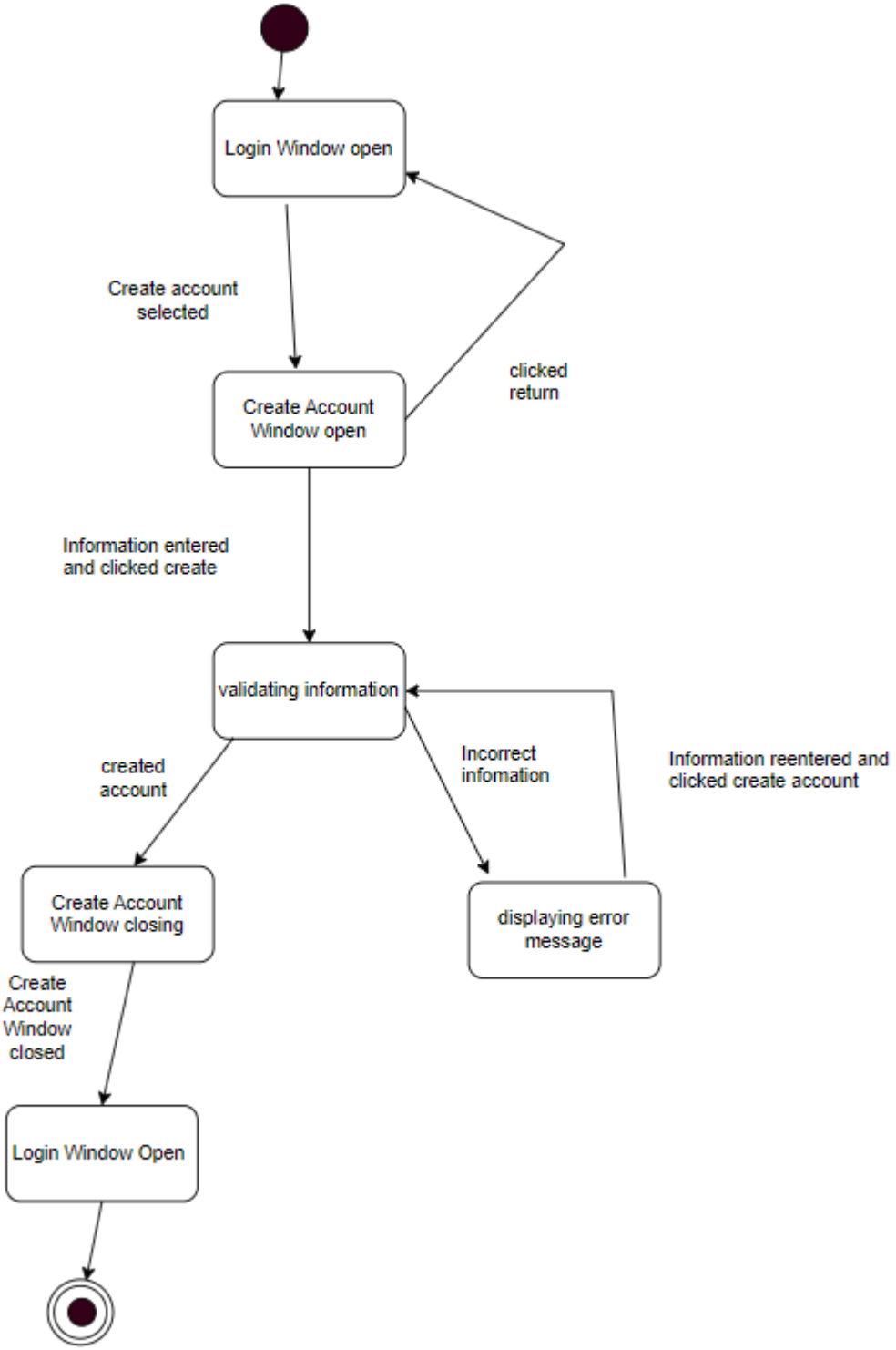

Actor
<<actor>>
: User
editPassword
Button: JButton
: ProfileController
editPassword
Frame : JFrame
cancelPassowrd
Edit : JButton
User (actor) opens the profile page and clicks the edit button for password
click
actionPerformed(ev)
setVisible(true)
User selects the cancel button at anytime
click
actionPerformed(ev)
setVisible(false)

13. Use case: Log out (author Selin)

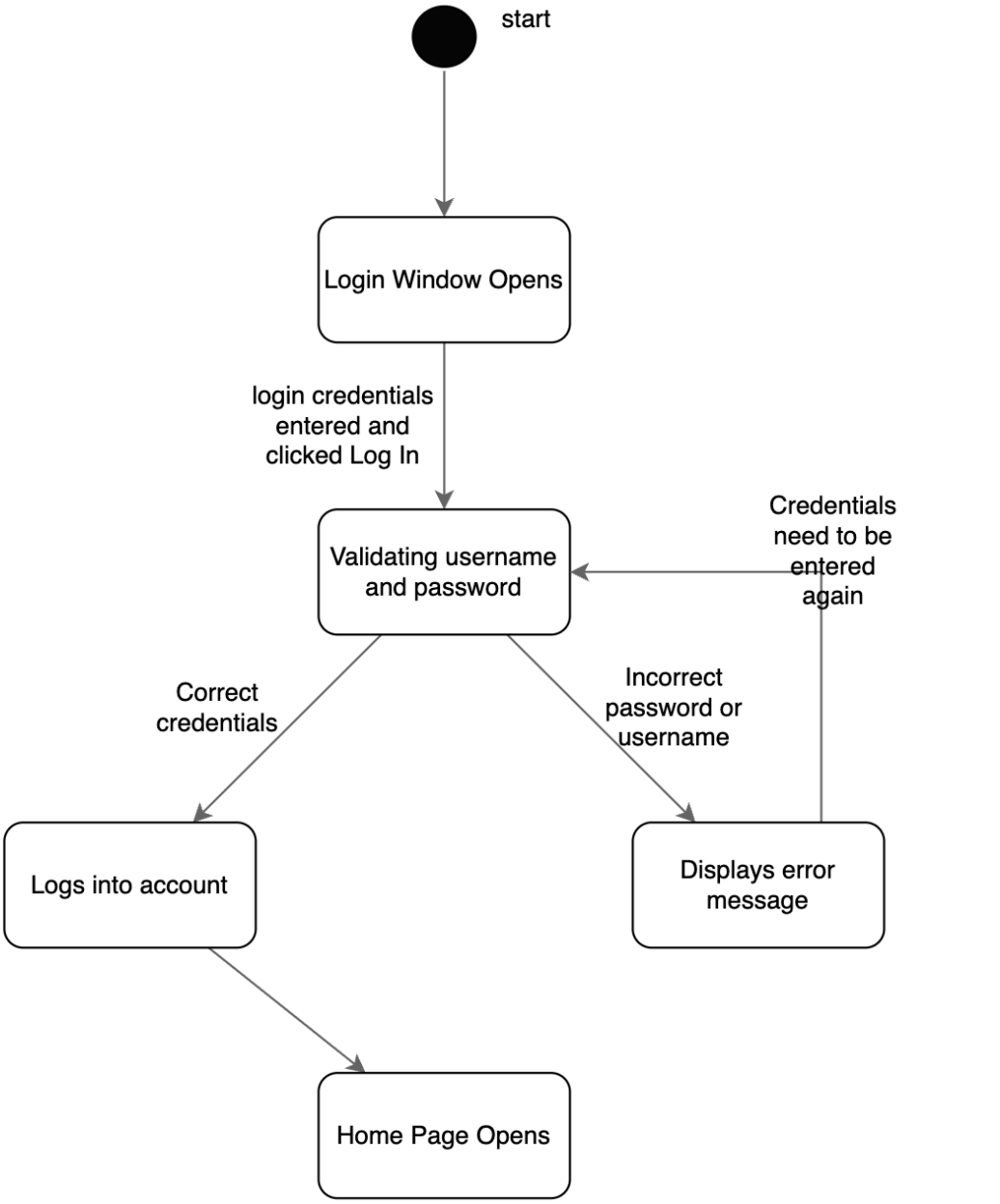


D. State diagram

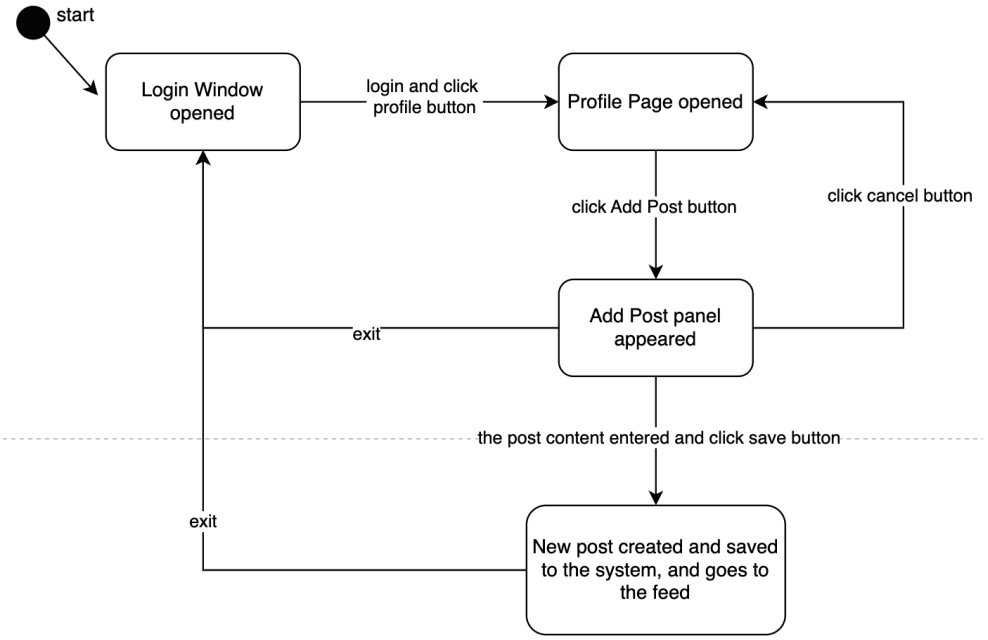
1. Use case: Create account (author Tabbie)



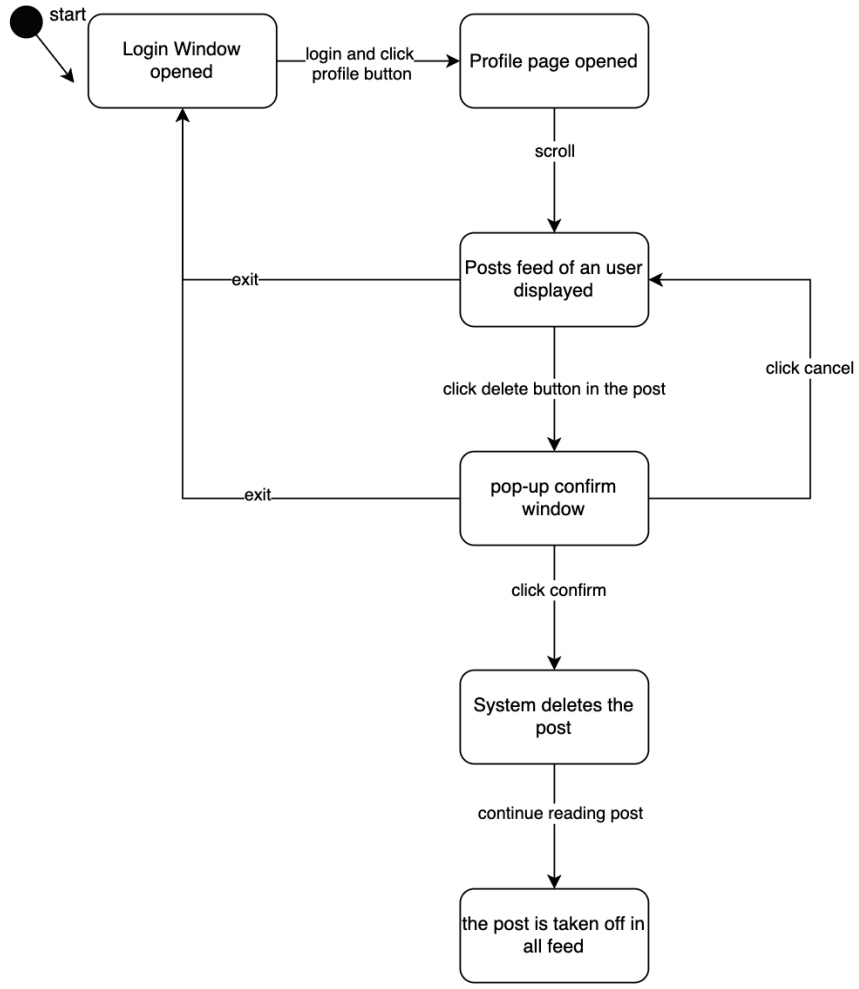
2. Use case: Log in (author Selin)



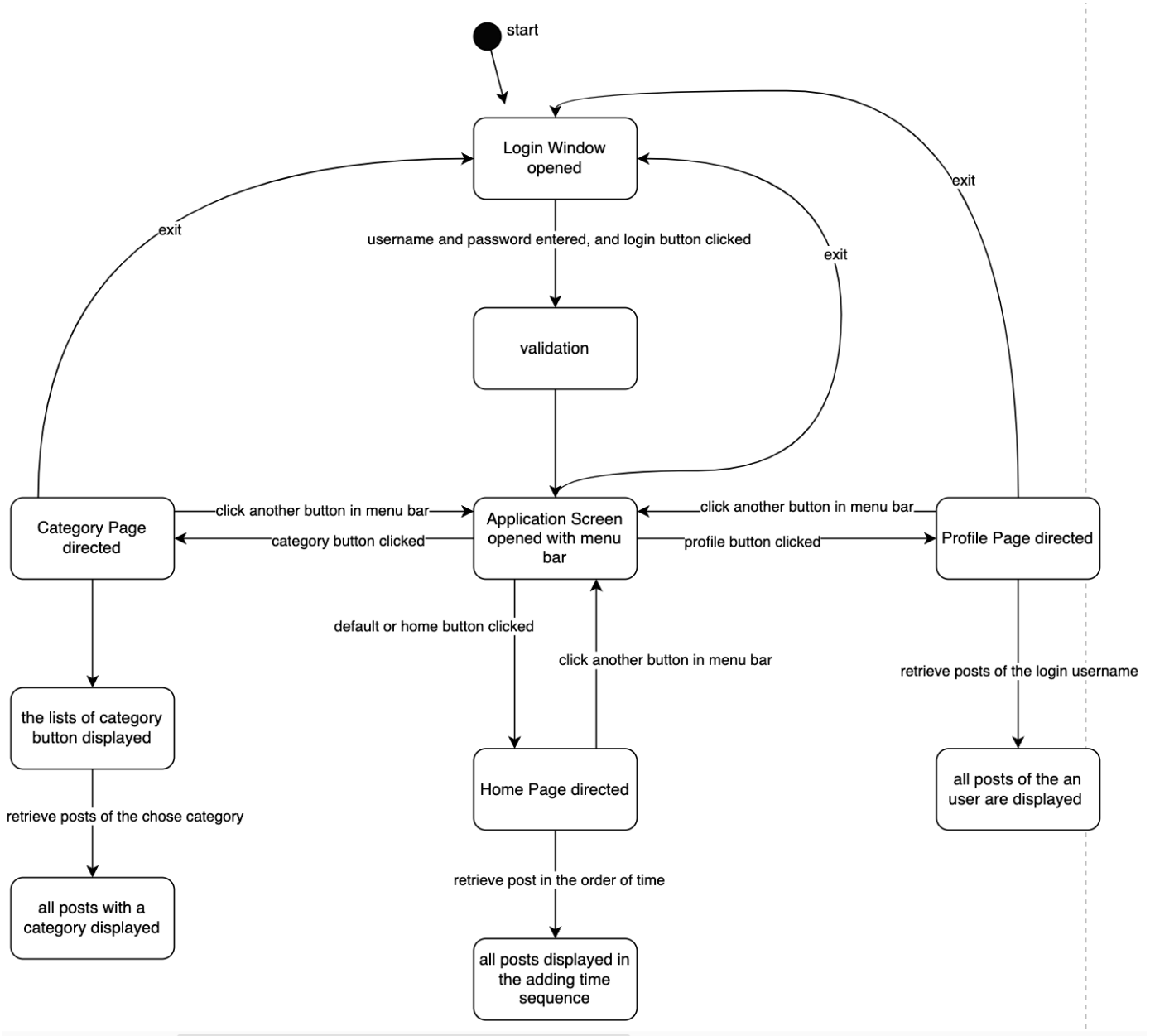
3. Use case: Write new post (author Hieu)



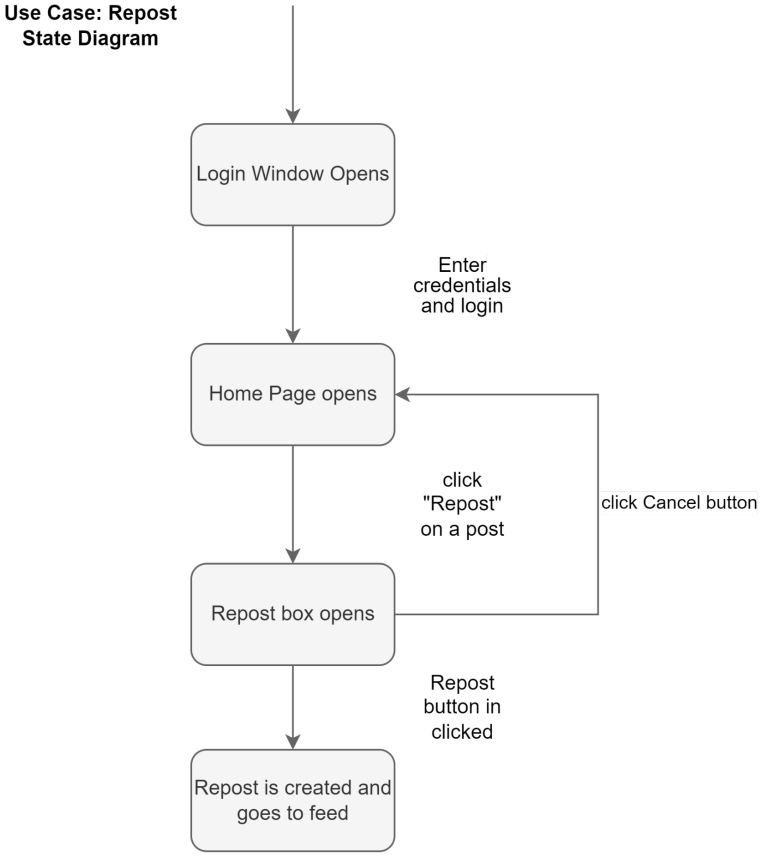
4. Use case: Delete a post (author Hieu)



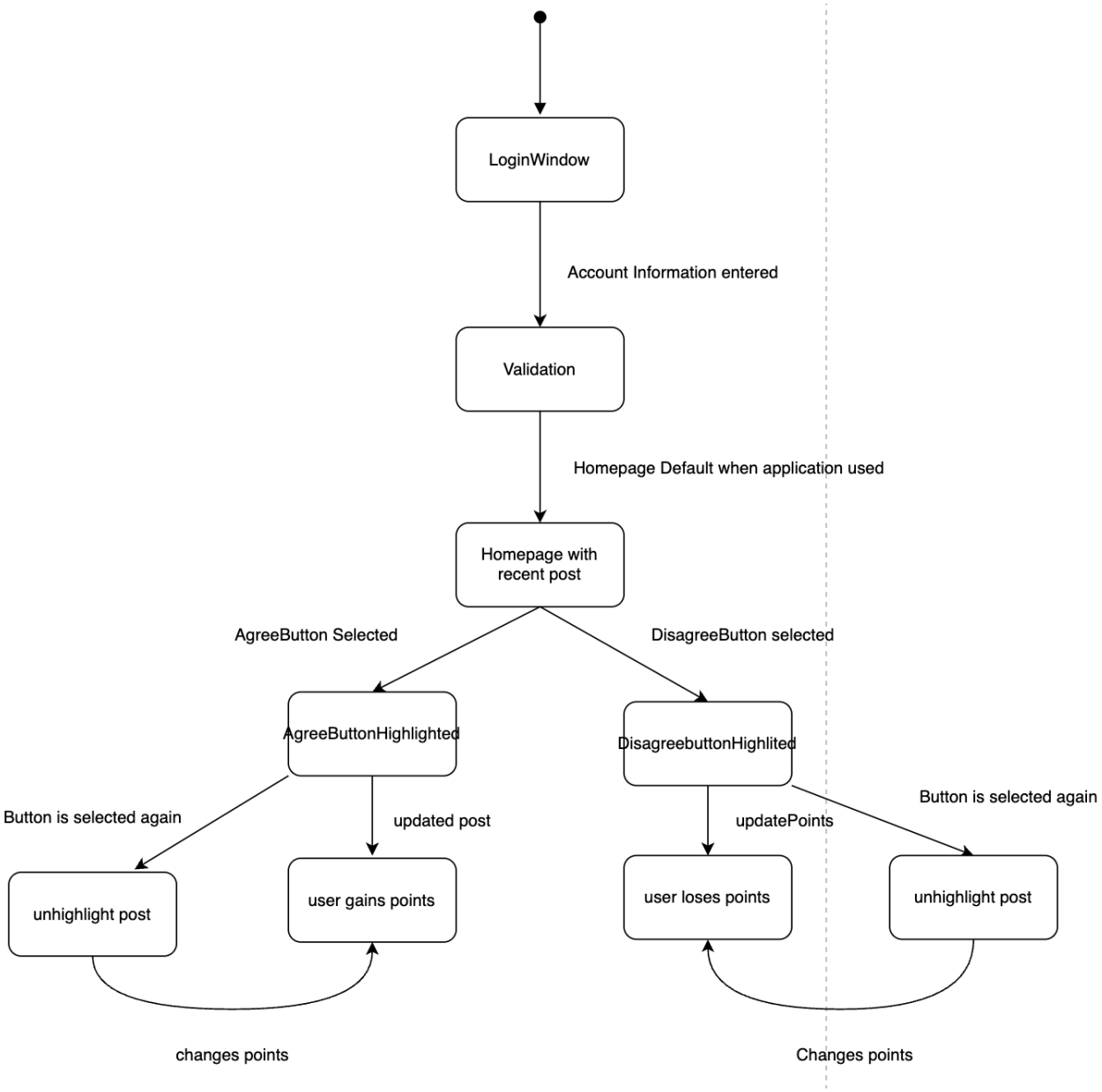
5. Use case: Read recent post and own post and category post (author Hieu)



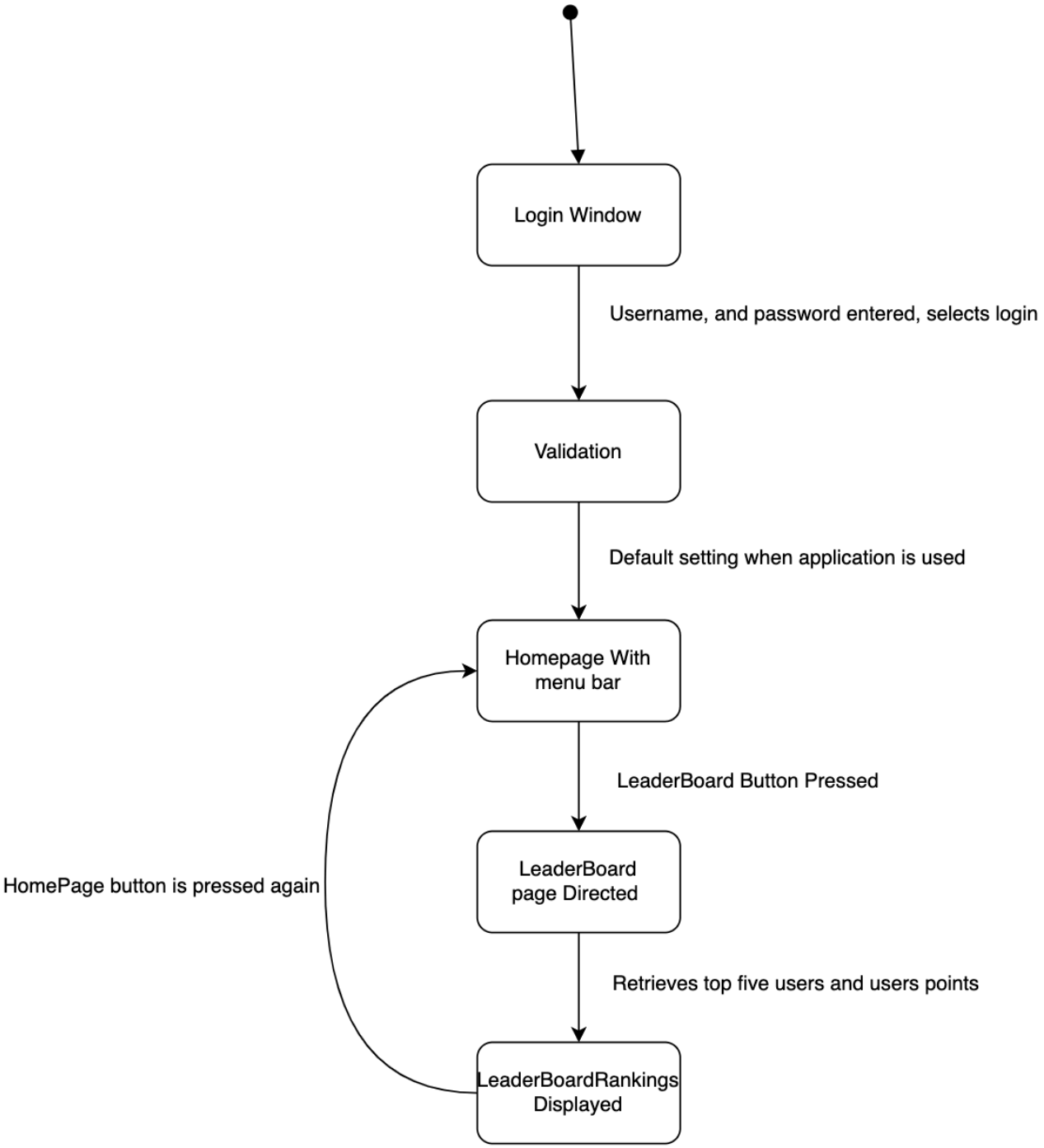
6. Use case: Repost (author Selin)



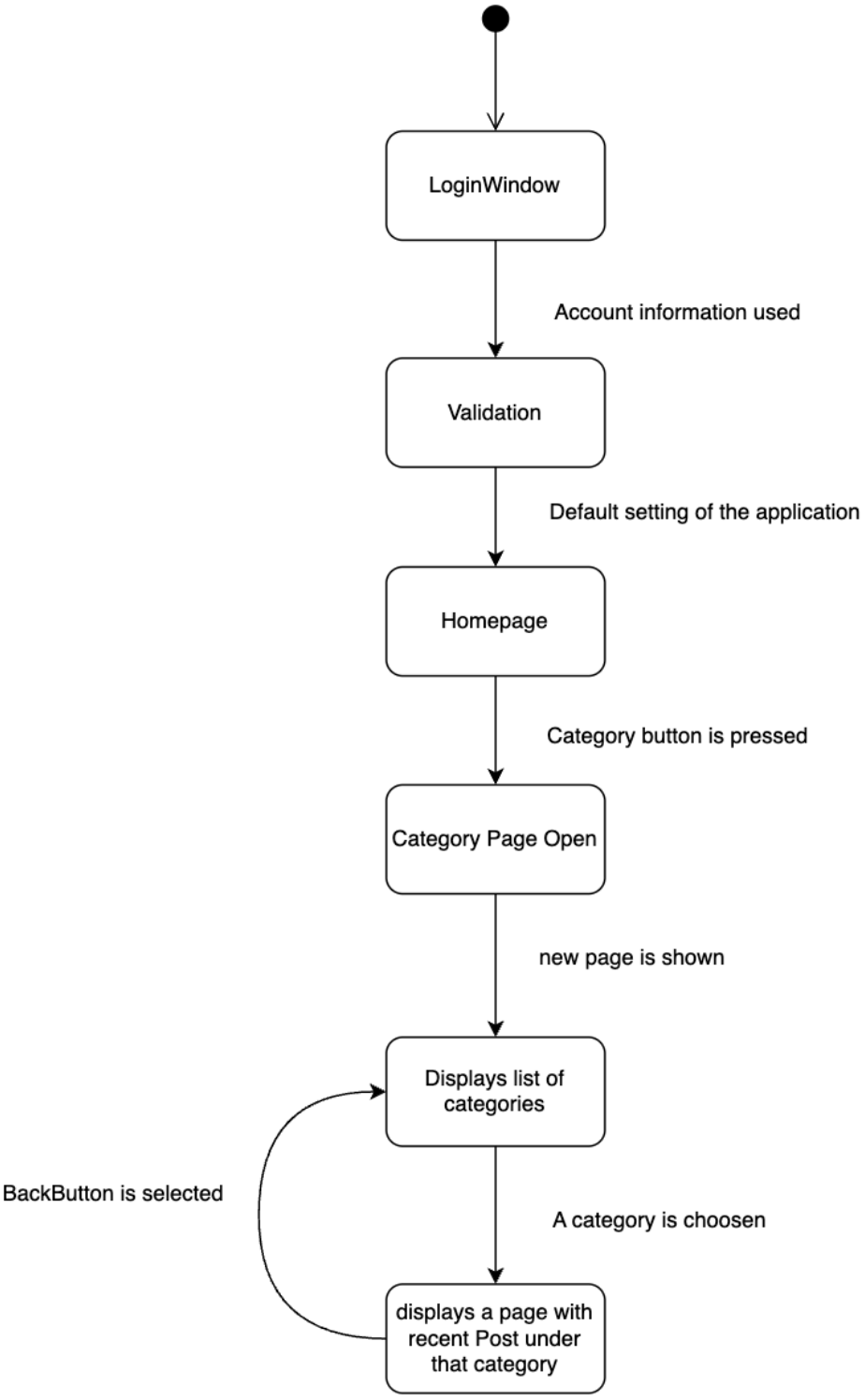
6. Use case: Agree/disagree post (author Lewis)



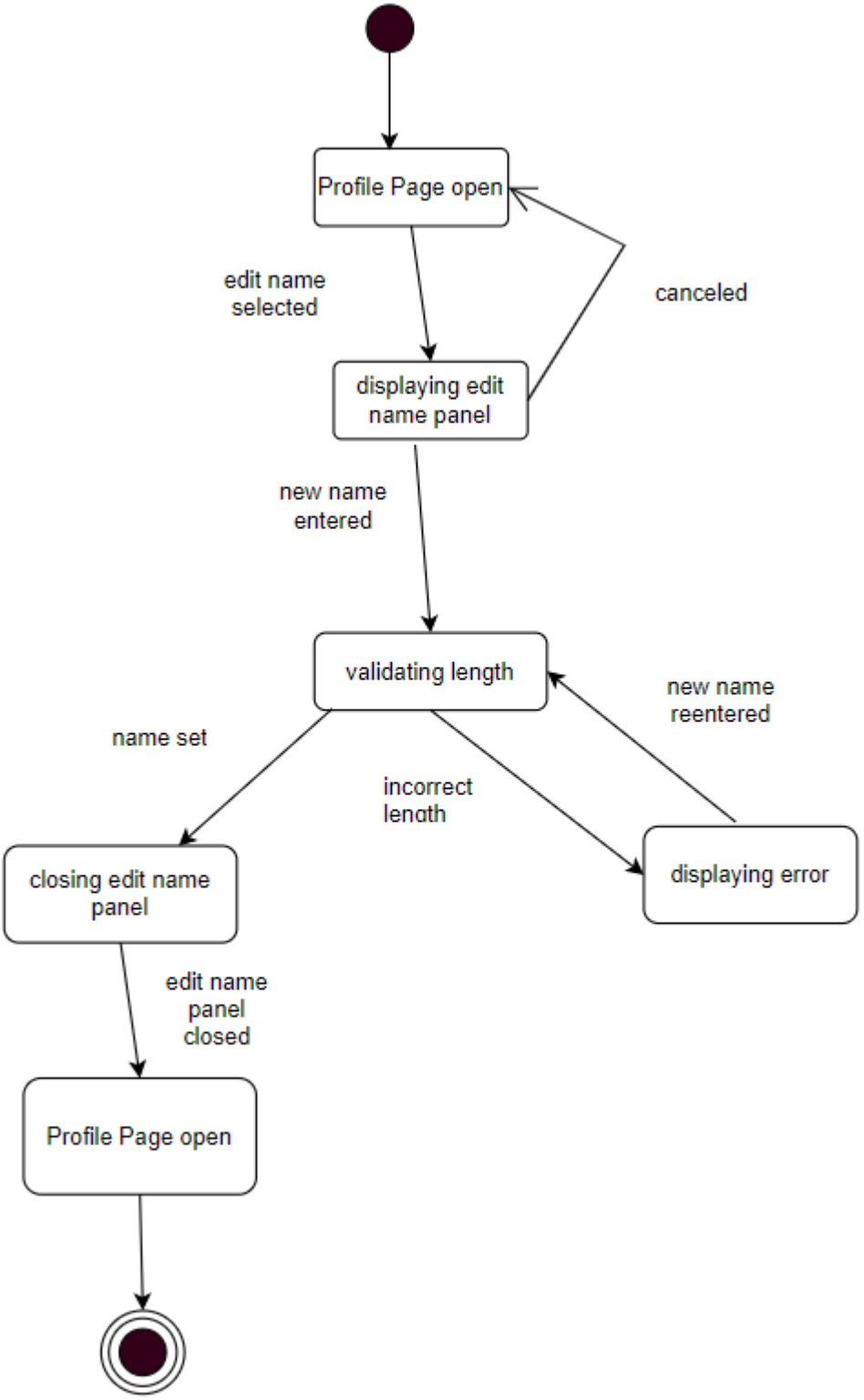
7. Use case: Check leaderboard (author Lewis)



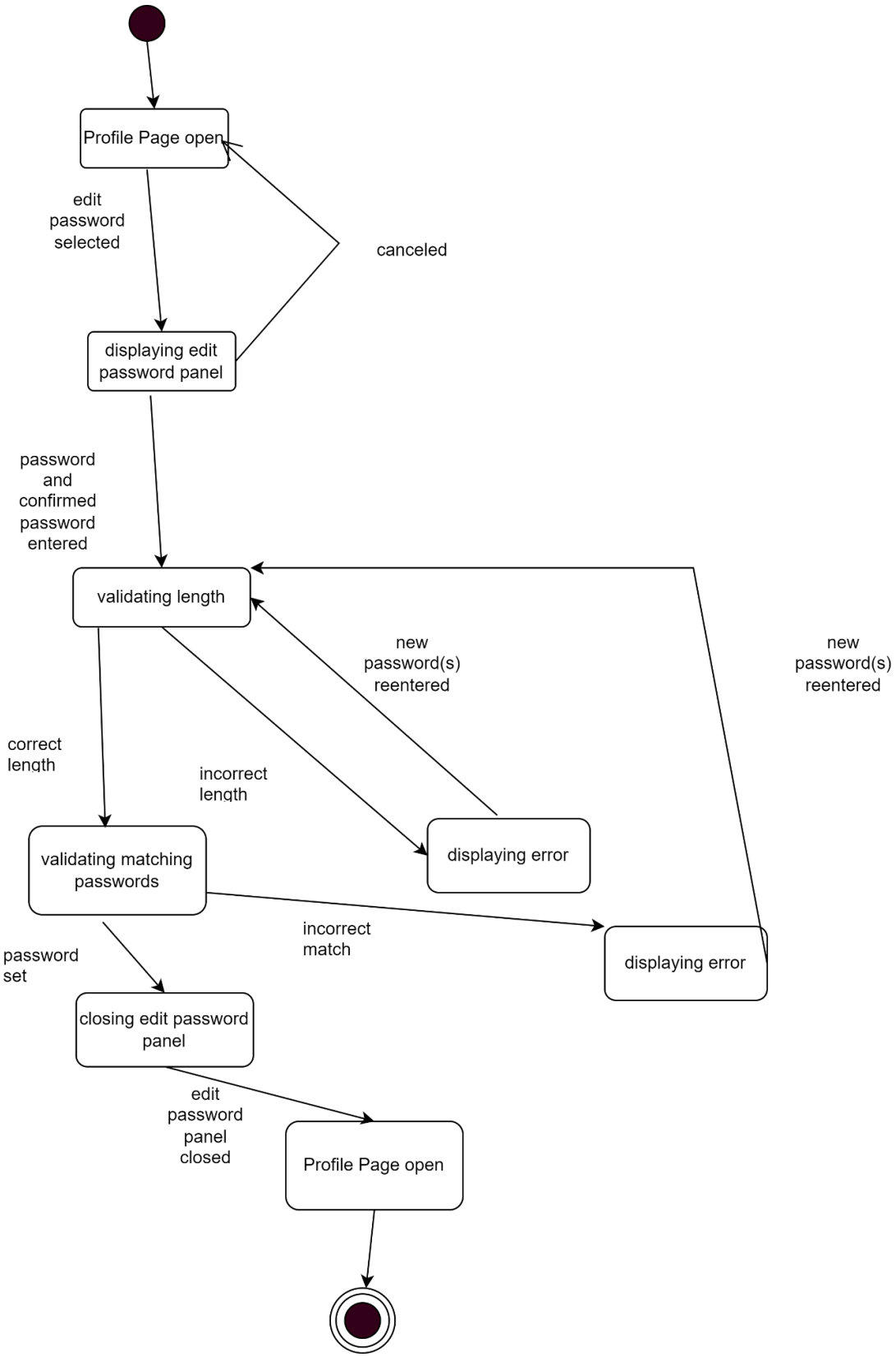
8. Use case: Find topic using category (author Lewis)



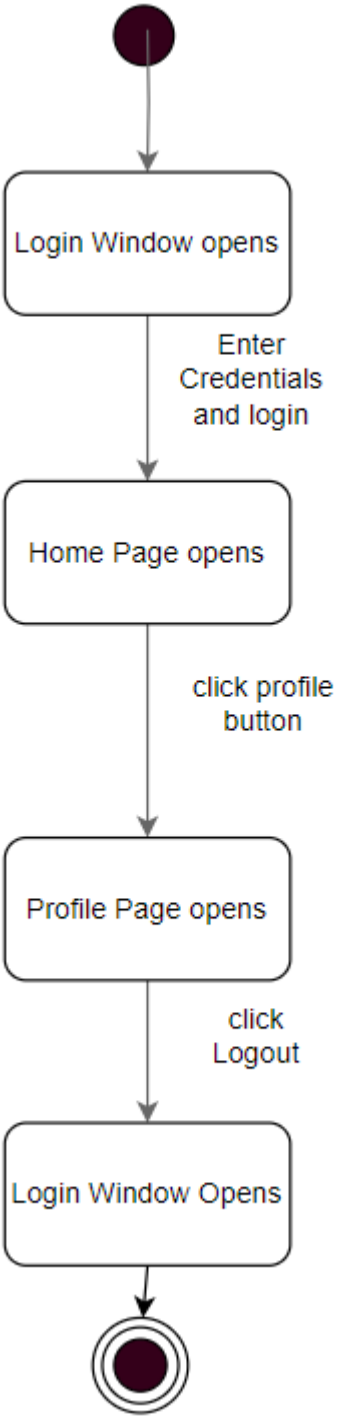
9. Use case: Edit profile name (author Tabbie)



10. Use case: Edit profile password(author Tabbie)

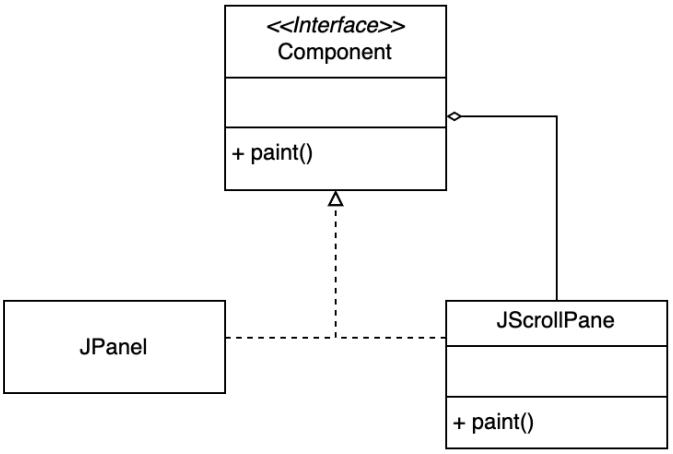


11. Use case: Log out (author Selin)



E. Pattern

1. Decorator Pattern



Mapping table:

Name in Design Pattern	**Actual name**
Component	Component
ConcreteComponent	JPanel
Decorator	JScrollPane
method()	paint()

2. Observer Pattern

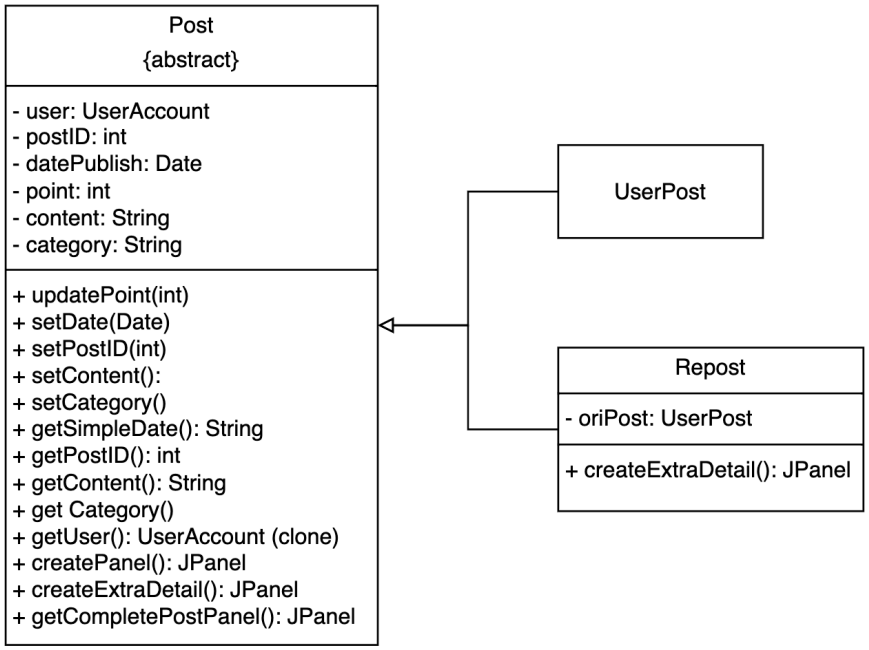
Mapping table:

Name in Design Pattern	**Actual name**
Subject	JButton
Observer	ActionListener
ConcreteObserver	The class that implements the ActionListener interface type
attach()	addActionListener
notify()	actionPerformed

3. Singleton Pattern

Name in Design Pattern	**Actual name**
Singleton class	NetworkSystem
Instance variable	instance
Instance method	getInstance()

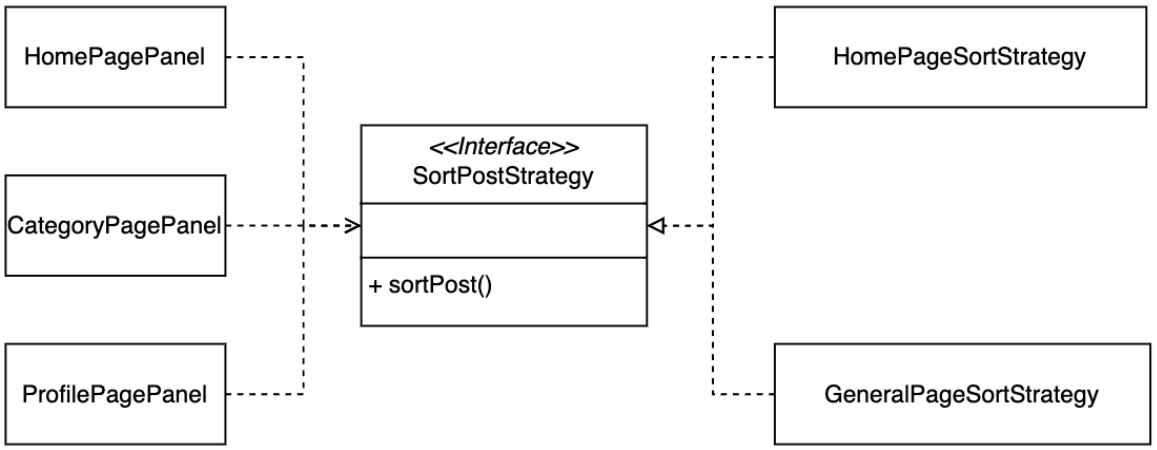
4. Template Method Pattern



Mapping table:

Name in Design Pattern	**Actual name**
AbstractClass	Post
ConcreteClass	UserPost, Repost
templateMethod()	getCompletePostPanel()
primitiveOp1()	getExtraDetail()

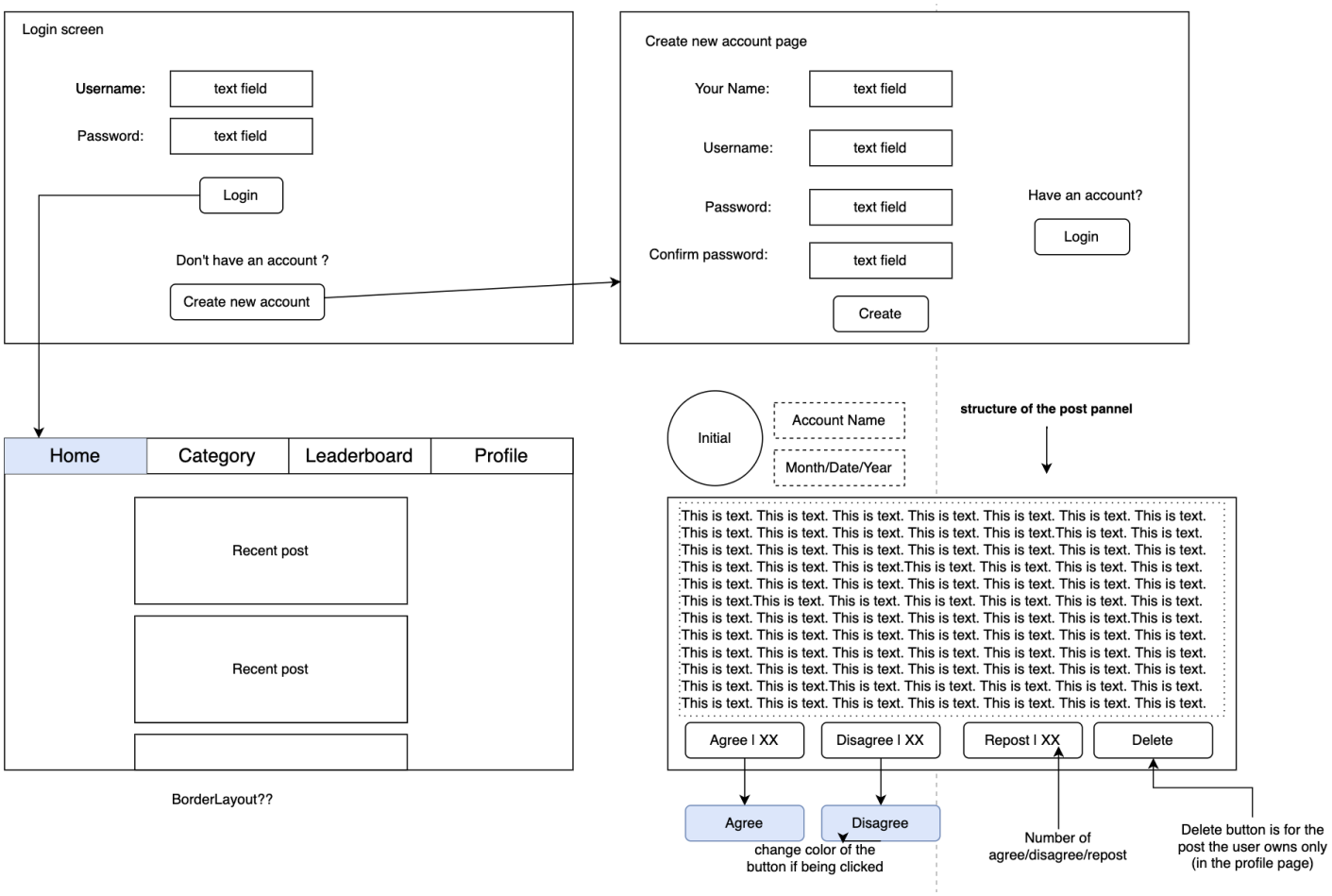
5. Strategy method pattern

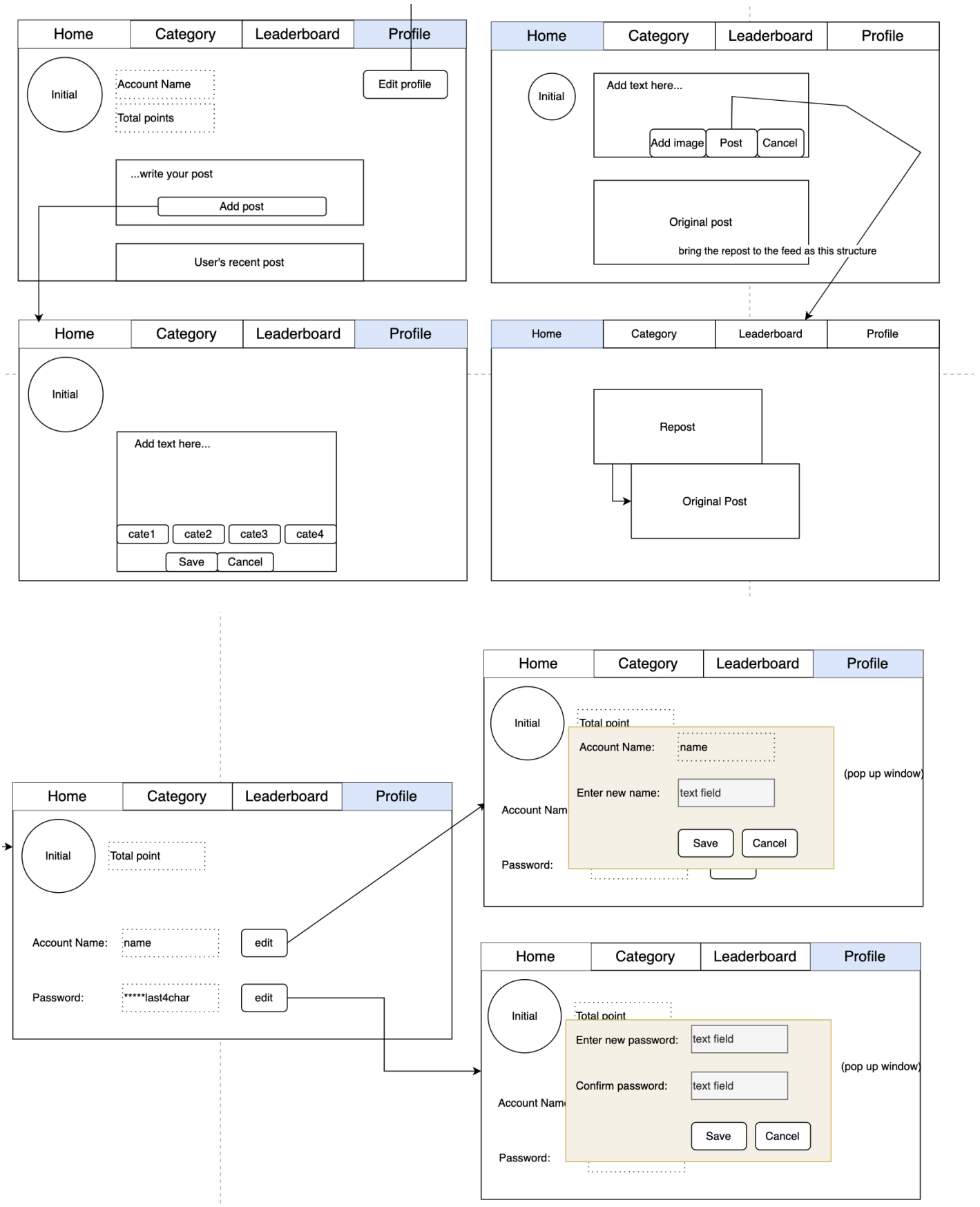


Mapping table

Name in Design Pattern	**Actual name**
Context	HomePagePanel, ProfilePagePanel, CategoryPagePanel
Strategy	SortPostStrategy
ConcreteStrategy	ProfilePageSortStrategy, GeneralPageSortStrategy
doWork()	sortPost()

F. User interface




Home
Category
Leaderboard
Profile
Initial
Account Name
Total points
Edit profile
...write your post
Add post
User's recent post
Add text here...
Add image
Post
Cancel
Original post
bring the repost to the feed as this structure
cate1
cate2
cate3
cate4
Save
Cancel
Repost
Original Post
Total point
Account Name:
name
Enter new name:
text field
Save
Cancel
Password:
(pop up window)
edit
*****last4char
Enter new password:
Confirm password:
Account Nam

G. Code

Network System:

```
package core;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import controller.LoginController;
import controller.NetworkWindowController;
import gui.CreateAccountWindow;
import gui.LoginWindow;
import gui.NetworkWindow;


/**
 * @author all
 * Network system, the class for the project
 * @class invariant: only one instance of NetworkSystem is created
 */
public class NetworkSystem implements Cloneable, Serializable {
    //attributes
    private static NetworkSystem instance = new NetworkSystem();

    private HashMap<String, UserAccount>  userAccountsList = new HashMap<>();
    private UserAccount currentUser;

    private ArrayList<Post> postsList = new ArrayList<>();
    private int postID; //this will set the id for the new post

    private NetworkWindow networkWindow =  new NetworkWindow();
```

```
    private NetworkWindowController networkWindowController = new
NetworkWindowController(this.networkWindow);

    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    private final int WIDTH = screenSize.width;
    private final int HEIGHT = screenSize.height;

    private LoginWindow loginWindowView;

    private HashMap<String, Category> categories = new HashMap<>();

    private static final String USER_ACCOUNTS_FILE = "/userAccounts.ser";
    private static final String POSTS_FILE = "/posts.ser";
    private String dataDirectory = System.getProperty("user.dir") + "/data";  // For example, "data" directory
under your project folder


    /**
     * Constructor of the NetworkSystem
     * @precondition n/a
     * @postcondition loadConfig() and ensureDirectoryExists is called
     */
    private NetworkSystem(){
        this.ensureDirectoryExists();
        this.loadSystemData();
    }

    /**
     * @author Tabbie Brantley
     * getInstance returns the only instance
     * @return instance as a NetworkSystem
     * @precondition n/a
     * @postcondition n/a
     */
    public static synchronized NetworkSystem getInstance() {
        if (instance == null) {
            System.out.println("Creating new NetworkSystem instance...");
            instance = new NetworkSystem();
        }
        return instance;
    }

    /**
     * @author TabbieBrantley
```

```
 * ensuresDirectoryExists makes sure that the directory exists, and if it does not
 * it creates it
 * @precondition dataDirectory must have a valid path
 * @postcondition the directory is created if it does not exist
 */
private void ensureDirectoryExists() {
    Path path = Paths.get(this.dataDirectory);
    if (!Files.exists(path)) {
        try {
            Files.createDirectories(path);  // Create the directory if it doesn't exist
            System.out.println("Created directory: " + this.dataDirectory);
        } catch (IOException e) {
            System.err.println("Error creating directory: " + e.getMessage());
        }
    }
}

/**
 * @author TabbieBrantley
 * loadSystemsData loads the UserAccount and Post data
 * @precondition n/a
 * @postcondition this.UserAccountsList and this.postsLists should have the
 * proper data
 */
private void loadSystemData() {
    this.userAccountsList = this.loadUserAccounts();
    this.postsList = this.loadPosts();
}

/**
 * @author Tabbie Brantley
 * loadUserAccounts retrieves all the user account files in USER_ACCOUNTS_FILE
 * @precondition n/a
 * @postocondition the user accounts are added to the userAccountsList HashMap
 */
private HashMap<String, UserAccount> loadUserAccounts() {
    try (ObjectInputStream inStream = new ObjectInputStream(
            new FileInputStream(this.dataDirectory + USER_ACCOUNTS_FILE))) {
        return (HashMap<String, UserAccount>) inStream.readObject();
    } catch (IOException | ClassNotFoundException e) {
        System.err.println("Error loading user accounts: " + e.getMessage());
    }
    return new HashMap<>();
}
```

```
/**
 * @author Tabbie Brantley
 * loadPosts retrieves all the post files in POSTS_FILE
 * @precondition n/a
 * @postocondition the posts are added to the postsList
 */
private ArrayList<Post> loadPosts() {
    try (ObjectInputStream inStream = new ObjectInputStream(
            new FileInputStream(this.dataDirectory + POSTS_FILE))) {
        return (ArrayList<Post>) inStream.readObject();
    } catch (IOException | ClassNotFoundException e) {
        System.err.println("Error loading posts: " + e.getMessage());
    }
    return new ArrayList<>(); // Return empty list if loading fails
}


/**
 * @author Tabbie Brantley
 * addUserAccounts creates and adds a new UserAccount
 * @param accName the account name as a String
 * @param name the account user name as a String
 * @param pw the password as a String
 * @precondition the username should not already exist
 * @postcondition the new user is added to userAccountsList
 */
public void addUserAccount(String accName, String name, String pw) {
    if (this.userAccountsList == null) {
        this.userAccountsList = new HashMap<>();
    }
    UserAccount user = new UserAccount(accName, name, pw);
    if (this.checkUserExists(name)) {
        System.out.println("Username already exists");
        return;
    }
    this.userAccountsList.put(user.getUsername(), user);
    this.saveUserAccounts();
}

/**
 * @author Tabbie Brantley
 * addPosts adds a new post to the postsList
```

```
 * @param post the post that is being added
 * @precondition n/a
 * @postcondition the new post is added to postsList
 */
public void addPosts(Post post) {
    if (this.postsList == null) {
        this.postsList= new ArrayList<>();
    }
    if (postsList.isEmpty()){
       post.setID(1);
    } else {
    post.setID(postsList.get(postsList.size()-1).getPostID()+1);
    }
    this.postsList.add(post);
    //postID += 1;
    System.out.println("Added successfully");
    this.savePosts();
}

/**
 * @author Hieu Truong
 * Remove a post in the postsList
 * @param post
 */
public void removePost(Post post){
    Iterator<Post> iterator = postsList.iterator();
    while (iterator.hasNext()){
        Post currentPost = iterator.next();
        if (currentPost.getPostID() == post.getPostID()){
            iterator.remove();
            break;
        }
    }
}

/**
 * @author Tabbie Brantley
 * saveUserAccounts saves all the user account objects to the USER_ACCOUNT_FILE
 * @precondition n/a
 * @postcondition n/a
 */
public void saveUserAccounts() {
    try (ObjectOutputStream outStream = new ObjectOutputStream(
            new FileOutputStream(this.dataDirectory + USER_ACCOUNTS_FILE))) {
```

```
            outStream.writeObject(this.userAccountsList);
            System.out.println("User accounts saved.");
        } catch (IOException e) {
            System.err.println("Error saving user accounts: " + e.getMessage());
        }
    }

    /**
     * @author Tabbie Brantley
     * savePosts saves all the post to the POSTS_FILE
     * @precondition n/a
     * @postcondition n/a
     */
    public void savePosts() {
        try (ObjectOutputStream outStream = new ObjectOutputStream(
                new FileOutputStream(this.dataDirectory + POSTS_FILE))) {
            outStream.writeObject(this.postsList);
            System.out.println("Posts saved.");
        } catch (IOException e) {
            System.err.println("Error saving posts: " + e.getMessage());
        }
    }

    /**
     * @author Tabbie Brantley
     * set current user sets the curUser to the UserAccount that is signed in
     * @param curUser the current user as a UserAccount
     * @precondition n/a
     * @postcondition n/a
     */
    public void setCurrentUser(UserAccount curUser){
        this.currentUser = curUser;
    }

    /**
     * @author Tabbie Brantley
     * getCurrentUser returns the current user
     * @precondition n/a
     * @return this.currentUser as a UserAccount
     * @precondition a user must be signed in
     * @postcondition n/a
     */
    public UserAccount getCurrentUser(){
        return this.currentUser;
```

```
}

/**
 * @author Tabbie Brantley
 * getNetworkWindow returns the network window
 * @return this.networkWindow as a NetworkWindow
 * @precondition n/a
 * @postcondition n/a
 */
public NetworkWindow getNetworkWindow(){
    return this.networkWindow;
}

/**
 * @author Tabbie Brantley
 * checkUserExists checks if the UserAccount is in the hashmap
 * @param userNm the key as a String
 * @return boolean value, true if exists, false if not
 */
public boolean checkUserExists(String userNm){
    return this.userAccountsList.containsKey(userNm);
}

/**
 * @author Tabbie Brantley
 * Accessor for the userAccountsList HashMap
 * @return this.userAccountsList as a HashMap
 * @precondition n/a
 * @postcondition n/a
 */
public HashMap<String, UserAccount> getUserAccountsList() {
    return this.userAccountsList;
}

 /**
 * @author Tabbie Brantley
 * Accessor for the postsList ArrayList
 * @return this.postsList as an ArrayList
 * @precondition n/a
 * @postcondition n/a
 */
public ArrayList<Post> getPostsList() {
    return this.postsList;
}
```

```
/**
 * @author Tabbie Brantley
 * Accessor for the HEIGHT
 * @return this.HEIGHT as an int
 * @precondition n/a
 * @postcondition n/a
 */
public int getHeight(){
    return this.HEIGHT;
}

/**
 * @author Tabbie Brantley
 * Accessor for the WIDTH
 * @return this.WIDTH  as an int
 * @precondition n/a
 * @postcondition n/a
 */
public int getWidth(){
    return this.WIDTH;
}

/**
 * @author Tabbie Brantley
 * usernameExists checks if a username is used in the HashMap usesAccountsLists
 * @param un the username as a String
 * @return boolean value, true if this.userAccountsList containts
 * the key, and false if not
 * @precondition n/a
 * @postcondition n/a
 */
public boolean usernameExists(String un){
    return this.userAccountsList.containsKey(un);
}

/**
 * @author Tabbie Brantley
 * loginUser logs in a user account
 * @param un the username as a String
 * @param pwd the password as a String
 * @return true when passwords match, else will return false
 * @precondition this.usernameExists(un) == ture
 * @postconition this.currentUser == this.userAccountsList.get(un), else return false
```

```
 */
public boolean loginUser(String un, String pwd){
    if (this.usernameExists(un)){
        if (this.userAccountsList.get(un).getPassword().equals(pwd)){
            this.setCurrentUser(this.userAccountsList.get(un));
            return true;
        }
    }
    return false;
}

/**
 * @author Tabbie Brantley
 * openApplication will display the networkWindow homepage
 * @precondition this.getCurrentUser() != null
 * @postcondition the application is displayed using the current user's information
 */
public void openApplication(){
    this.getNetworkWindow().displayWindow();
    this.networkWindow.displayHomePage(this.getCurrentUser());
}


/**
* @author Selin Topac
* user will click logout button, application will save info and open login window
* @preconditon this.getCurrentUser() != null
* @preconditon user must click logout button (in profile page)
* @postconditon this.getCurrentUser() == null
* @postconditon network window is closed, login window is opened
*/
public void logout(){
    System.out.println("logout working");
    this.currentUser = null;
    this.savePosts();
    this.saveUserAccounts();
    this.networkWindow.closeWindow();
    this.networkWindow = new NetworkWindow();
    this.networkWindowController = new NetworkWindowController(this.networkWindow);
    this.login();
}

/**
* @author Selin Topac
```

```
* opens the login window
* @preconditon this.LoginWindowView == null
* @preconditon this.getCurrentUser() == null
* @postcondition this.LoginWindowView != null
*/
public void login(){
    this.loginWindowView = new LoginWindow();
    CreateAccountWindow createAccountView = new CreateAccountWindow();
    LoginController loginController = new LoginController(loginWindowView, createAccountView, this);
}
/**
 * @author Tabbie Brantley
* saveSystemState saves the UserAccounts and posts
* @precondition n/a
* @postcondition n/a
*/
public void saveSystemState(){
    this.saveUserAccounts();
    this.savePosts();
    System.out.println("Everything saved!");
}


/**
* @author Lewis Cox
* gets a category by the name
* @param categoryName
* @return the category object
 */
public Category getCategory(String categoryName){
    return categories.get(categoryName);
}
/**
* @author Lewis Cox
* gets a list of all categories
* @return list of categories
*/

/**
 *
 * @return
 */
public List<String> getAllCategories(){
    return new ArrayList<>(categories.keySet());
```

```
    }

    /**
     * @author Lewis Cox
     * @return
     */
    public List<UserAccount> getTopUsers() {
        List<UserAccount> sortedUsers = new ArrayList<>(this.userAccountsList.values());
        sortedUsers.sort((u1, u2) -> Integer.compare(u2.getPoints(), u1.getPoints()));
        return sortedUsers;
    }

    /**
     * @author Hieu Truong
     * @param username the username as a String
     * @param val the value will be add to point
     * @precondition the userAccountList contains key username
     * @postcondition point update to userAccountList.get(username)
     */
    public void updatePointForUser(String username, int val){
        userAccountsList.get(username).updatePoints(val);
    }

    /**
     * Main method for running the application
     */
    public static void main(String[] args) {
        NetworkSystem networkSystem = NetworkSystem.getInstance();
        networkSystem.loadPosts();
        networkSystem.loadUserAccounts();

        // Print all user accounts to verify the new user was added
        System.out.println("Current users: " + networkSystem.userAccountsList);
        System.out.println("Current posts: " + networkSystem.postsList);

        networkSystem.login();
        networkSystem.savePosts();
        networkSystem.saveUserAccounts();
    }

}
```

Category:

```
package core;
```

```
/**
 * @author Lewis Cox
 * category in the system
 */
public class Category {
    private String name;

    /**
     * Constructs a Category object
     *
     * @param name of the category
     */
    public Category(String name) {
        this.name = name;
    }

    /**
     * gets the name of the category
     *
     * @return the name of the category
     */
    public String getName() {
        return name;
    }

    /**
     * updates the name of the category
     *
     * @param name the new name for the category
     */
    public void setName(String name) {
        this.name = name;
    }
}
```

Post:

```
package core;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.io.Serializable;
```

```
import java.time.LocalDate;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.SwingConstants;
import javax.swing.border.Border;
import javax.swing.border.EmptyBorder;

import controller.PostObserver;
import utility.SortPostStrategy;


/**
 * Post is an Abstract class representing a Post in the system.
 * Acts as the Subject in the Observer pattern and the Model in the MVC architecture.
 *
 * @author Hieu Truong
 */
public abstract class Post implements Serializable {
    private final String content;
    private String datePulish;
    private String category;
    private int point;
    private final UserAccount owner;
    private int ID;

    private HashMap<String, String> interactList;
    private JPanel postPanel;
    private JButton[] buttons;
    private JTextArea pointArea;

    private final int width = NetworkSystem.getInstance().getWidth();
    private final int height = NetworkSystem.getInstance().getHeight();

    /**
```

```
     * Constructs a Post object with the specified content and owner.
     * @param content: the content of the post.
     * @param owner: the UserAccount that owns this post.
     */
    public Post(String content, UserAccount owner) {
        this.content = content;
        this.owner = owner;
        this.point = 0;
        this.interactList = new HashMap<>();
        this.category = "";
        ZoneId zoneId = ZoneId.of("America/New_York");
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
        this.datePulish = LocalDate.now(zoneId).format(formatter);
    }

    /**
     * Sets the unique ID for this post.
     *
     * @param id the unique identifier to set.
     */
    public void setID(int id) {
        this.ID = id;
    }

    /**
     * Updates the points of the post.
     *
     * @param number the amount to increment or decrement the points.
     *
     * @Precondition: number is int and !=0
     * @Postcondition: The points of the post are updated by number, the textArea of point is updated
     */
    public void updatePoint(int number) {
        NetworkSystem.getInstance().updatePointForUser(owner.getUsername(), number);
        point += number;
        pointArea.setText("Point: " + point);
        postPanel.repaint();
    }

    /**
     * Sets the category of the post.
     *
     * @param category the category to set.
     */
```

```
public void setCategory(String category) {
    this.category = category;
}

/**
 * Gets the owner of the post.
 *
 * @return the UserAccount object that owns the post.
 */
public UserAccount getOwner() {
    return owner;
}

/**
 * Gets the category of the post.
 *
 * @return the category of the post, or an empty string if the category is null.
 */
public String getCategory() {
    return category == null ? "" : category;
}

/**
 * Gets the ID of the post.
 *
 * @return the post's unique identifier.
 */
public int getPostID() {
    return ID;
}

/**
 * Adds an interaction to the post.
 * @param username: the username of the account that interact with this post
 * @param interact: the type of interaction
 */
public void addInteract(String username, String interact){
    interactList.put(username, interact);
}

/**
 * Remove an interaction to the post
 * @param username: the username of the account that interact with this post
 * @precondition the interactList contain the username
```

```
     * @postcondition the value with the key of username is removed
     */
    public void removeInteract(String username){
        if (interactList.containsKey(username)){
            interactList.remove(username);
        }
    }

    /**
     * Get the kind of interaction
     * @param username: the username of the account that interact with this post
     * @return the interaction String
     */
    public String getInteract(String username){
        if (interactList.containsKey(username)){
            return interactList.get(username);
        }
        return "";
    }

    /**
     * @return the String represents a post object
     */
    @Override
    public String toString(){
        return content + " " + owner.getAccountName();
    }

    /**
     * Creates a JPanel representing the post's basic information.
     * The panel contains the post's avatar, author, date, points, and category.
     * @return JPanel representing the post's basic information.
     */
    public JPanel getThisPostPanel() {

        JPanel thisPanel = new JPanel(new BorderLayout());

        //set user interface of the avatar
        JLabel avatar = new JLabel(String.valueOf(owner.getAccountName().charAt(0)).toUpperCase(),
SwingConstants.CENTER);
        avatar.setPreferredSize(new Dimension(30, 30));
        avatar.setOpaque(true);
        avatar.setBackground(Color.PINK);
        avatar.setBorder(BorderFactory.createLineBorder(Color.BLACK));
```

```
    //set user interface of the author
    JTextArea author = new JTextArea(owner.getAccountName() + " | " + datePulish
            + " | Post ID: " + ID + " | ");
    author.setOpaque(false);
    author.setEditable(false);

    //set  user interface of the point
    pointArea = new JTextArea("Point: " + point);
    pointArea.setEditable(false);
    pointArea.setOpaque(false);

    //set  user interface of the category
    JTextArea categoryArea = new JTextArea(" | " + category);
    categoryArea.setEditable(false);
    categoryArea.setOpaque(false);

    //set user interface of the information of the post
    JPanel infoPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
    infoPanel.add(avatar);
    infoPanel.add(author);
    infoPanel.add(pointArea);
    infoPanel.add(categoryArea);

    thisPanel.add(infoPanel, BorderLayout.NORTH);

    //set user interface for the content
    JTextArea contentArea = new JTextArea(content);
    contentArea.setEditable(false);
    contentArea.setLineWrap(true);
    contentArea.setWrapStyleWord(true);
    //contentArea.setBounds(15, 0, width / 5 * 3 - 40, height / 15 * 5);
    contentArea.setBounds(15, 0, contentArea.getWidth() - 40, contentArea.getHeight() * 5);
    contentArea.setBorder(new EmptyBorder(10, 10, 10, 10));
    thisPanel.add(contentArea, BorderLayout.CENTER);

    return thisPanel;
}

/**
 * Abstract method to retrieve additional details for the post.
 *
 * @return a JPanel containing the additional details.
 */
```

```
public abstract JPanel getExtraDetail();

/**
 * Constructs a complete JPanel for the post, including basic and additional details.
 *
 * @param displayStrategy the strategy for displaying the post.
 * @return the complete JPanel for the post.
 */
public JPanel getCompletePostPanel(SortPostStrategy displayStrategy) {
    postPanel = new JPanel(new BorderLayout());
    postPanel.setBackground(Color.WHITE);

    Border innerBorder = BorderFactory.createEmptyBorder(15, 15, 15, 15);
    Border outerBorder = BorderFactory.createLineBorder(Color.LIGHT_GRAY, 2);
    postPanel.setBorder(BorderFactory.createCompoundBorder(outerBorder, innerBorder));

    JPanel contentPanel = new JPanel();
    contentPanel.setLayout(new BoxLayout(contentPanel, BoxLayout.Y_AXIS));
    contentPanel.add(getThisPostPanel());

    //if this post is a repost, it needs to add the information of the original post
    if (getClass().getSimpleName().equals("Repost")) {
        contentPanel.add(getExtraDetail());
    }

    ArrayList<String> buttonNameList = new ArrayList<>(Arrays.asList("Agree", "Disagree", "Repost"));
    //if the page is not profile page, it cannot have the delete button
    if (displayStrategy.getClass().getSimpleName().equals("ProfilePageSortStrategy")) {
        buttonNameList.add("Delete");
    }

    //set the button panel, which contains multiple buttons
    buttons = new JButton[buttonNameList.size()];
    JPanel buttonPanel = new JPanel(new GridLayout(1, buttonNameList.size()));
    buttonPanel.setBackground(Color.WHITE);

    for (int i = 0; i < buttonNameList.size(); i++) {
        buttons[i] = new JButton(buttonNameList.get(i));
        buttons[i].setBackground(Color.WHITE);
        buttons[i].setContentAreaFilled(true);
        buttons[i].setOpaque(true);
        buttons[i].addActionListener(new PostObserver(this));
        buttons[i].setBorderPainted(false);
        buttonPanel.add(buttons[i]);
```

```
        }

        // Highlight buttons based on the current user interaction with this post
        String currentUserInteraction =
getInteract(NetworkSystem.getInstance().getCurrentUser().getUsername());
        if (currentUserInteraction.equals("Agree")) {
            buttons[0].setBackground(new Color(219, 231, 252));
        } else if (currentUserInteraction.equals("Disagree")) {
            buttons[1].setBackground(new Color(219, 231, 252));
        }

        postPanel.add(contentPanel, BorderLayout.CENTER);
        postPanel.add(buttonPanel, BorderLayout.SOUTH);

        return postPanel;
    }

    /**
     * Gets the "Agree" button for the post.
     *
     * @return the JButton for "Agree".
     */
    public JButton getAgreeButton() {
        return buttons[0];
    }

    /**
     * Gets the "Disagree" button for the post.
     *
     * @return the JButton for "Disagree".
     */
    public JButton getDisagreeButton() {
        return buttons[1];
    }

}
```

Repost:

```
package core;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

/**
 * The Repost class represents a post that is a repost of an original post.
 * It extends the Post class and adds functionality specific to reposting
```

```
 * @author Hieu Truong
 */
public class Repost extends Post {
    private final Post originalPost;

    /**
     * Constructs a Repost object with the given content, owner, and original post.
     * The constructor calls the superclass {@link Post}'s constructor to initialize the post
     * @param content: The content of the repost.
     * @param owner: The UserAccount that created the repost.
     * @param oriPost: The UserPost that is being reposted.
     * @precondition content cannot be null or empty, and owner and oriPost must be valid.
     * @postcondition A new Repost object is created with the specified content, owner, and original post.
     */
    public Repost(String content, UserAccount owner, Post oriPost){
        //calls the superclass Post constructor to initialize the post
        super(content, owner);
        this.originalPost = oriPost;
    }

    /**
     * Returns the additional details for this repost.
     * This method returns a JPanel that represents the original post, with an added border for visual
separation.
     * @return A JPanel representing the original post with added padding.
     * @precondition The Repost object has been created and contains a reference to an original post.
     * @postcondition The JPanel of the original post is returned
     */
    @Override
    public JPanel getExtraDetail() {
        JPanel oriPanel = originalPost.getThisPostPanel();
        oriPanel.setBorder(new EmptyBorder(10, 25, 10, 5)); // Add padding to the original post panel
        return oriPanel;
    }
}
```

UserAccount:

```
package core;
/**
 * @author Tabbie Brantley
 * UserAccount class for the users is Serializable
 * @class invariant the USERNAME String cannot be changed
 * */
```

```
import java.io.Serializable;

public class UserAccount implements Serializable{
    private final String USERNAME;
    private String accountName;
    private String password;
    private int points;

    /**
     * Constructor for UserAccount
     * @param accountNm the account name as a String
     * @param userNm the username as a String
     * @param pwd the password as a String
     * @precondition ((8 <= accountName.length()) && (accountName.length.() <= 20))
     * && ((8 <= userNm.length()) && (userNm.length <= 20))
     * && ((8 <= pwd.length()) && (pwd.length <= 20))
     * @postcondition points are initialized to 0
     */
    public UserAccount(String accountNm, String userNm, String pwd){
        this.accountName = accountNm;
        this.USERNAME = userNm;
        this.password = pwd;
        this.points = 0;
    }

    /**
     * Accessor for username
     * @return USERNAME as a String
     * @precondition n/a
     * @postcondition n/a
     */
    public String getUsername(){
        return this.USERNAME;
    }

    /**
     * Accessor for password
     * @return password as a String
     * @precondition n/a
     * @postcondition n/a
     */
    public String getPassword(){
        return this.password;
    }
```

```
/**
 * Accessor for points
 * @return points as an int
 * @precondition n/a
 * @postcondition n/a
 */
public int getPoints(){
    return this.points;
}

/**
 * Mutator for points
 * @param val the value that the points should increase or decrease by as an int
 * @precondition n/a
 * @postcondition this.points >= 0
 */
public void updatePoints(int val){
    this.points += val;
}

/**
 * Mutator for accountName
 * @param newName the new name as a String
 * @precondition (8 <= newName.length()) && (newName.length() <= 20)
 * @postcondition the accountName is changed
 */
public void setName(String newName){
    this.accountName = newName;
}
/**
 * Mutator for password
 * @param newPWD the new password as a String
 * @precondition (8 <= newPwd.length()) && (newPwd.length() <= 20)
 * @postcondition the password is changed
 */
public void setPassword(String newPwd){
    this.password = newPwd;
}

/**
 * Accessor for accountName
 * @return accountName as a String
 * @precondition n/a
```

```
     * @postcondition n/a
     */
    public String getAccountName(){
        return this.accountName;
    }
    @Override
    public String toString(){
        return USERNAME + " " + points;
    }
}
```

UserPost:

```
package core;

import javax.swing.JPanel;
/**
 * The UserPost class represents an original post created by a user in the system.
 * It extends the Post class and provides additional details specific to the user.
 * @author Hieu Truong
 */
public class UserPost extends Post {

    /**
     * Constructs a UserPost object with the given content and owner.
     * @param content: The content of the post.
     * @param owner: the UserAccount, who that created the post.
     * @precondition content cannot be null or empty, and owner must be a valid UserAccount.
     * @postcondition A new UserPost object is created with the specified content and owner.
     */
    public UserPost(String content, UserAccount owner){
        //calls the superclass {@link Post}'s constructor to initialize the post.
        super(content, owner);
    }

    /**
     * Returns additional details for this post.
     * In this case, it returns null, meaning no extra details are provided.
     * @return JPanel representing the additional details for the post, or null if no extra details are
provided.
     * @precondition The UserPost object has been created.
     * @postcondition null is returned, as no extra details are defined for the UserPost class.
     */
    @Override
    public JPanel getExtraDetail() {
        return null;
```

```
    }
}
```

CategoryPageController:

```
package controller;

import gui.CategoryPage;


/**
 * @author Lewis Cox
 * CategoryPageController is the logic behind updating posts
 * it gets posts by category and updates the view accordingly
 */
public class CategoryPageController {

    private CategoryPage categoryPage;

    /**
     * constructs a CategoryPageController with the given CategoryPage
     *
     * @param categoryPage CategoryPage instance to be controlled
     */
    public CategoryPageController(CategoryPage categoryPage) {
        this.categoryPage = categoryPage;
    }

    /**
     * updates the CategoryPage with posts from the specified category
     *
     * @param category The category to filter posts by
     */
    public void updateCategoryPosts(String category) {
        categoryPage.updateCategoryPosts(category);
    }
}
```

LeaderboardController:

```
package controller;

import java.util.List;
import core.*;
import gui.LeaderboardPage;
```

```
public class LeaderboardController {
    private LeaderboardPage leaderboardPage;
    private List<UserAccount> users;

    /**
     * @author Lewis Cox
     * Constructor to initialize the controller
     * @param leaderboardPage the LeaderboardPage to update
     * @param users the list of users for the leaderboard
     */
    public LeaderboardController(LeaderboardPage leaderboardPage, List<UserAccount> users) {
        this.leaderboardPage = leaderboardPage;
        this.users = users;
        updateLeaderboard();
    }

    /**
     * Updates the leaderboard by sorting the user list and refreshing the page
     */
    public void updateLeaderboard() {
        if (users == null || users.isEmpty()) {
            leaderboardPage.populateLeaderboard(List.of());
        } else {
            users.sort((u1, u2) -> Integer.compare(u2.getPoints(), u1.getPoints()));
            leaderboardPage.populateLeaderboard(users);
        }
    }

    /**
     * updates the list and refreshes the leaderboard
     * @param users the new list of users
     */
    public void setUsers(List<UserAccount> users) {
        this.users = users;
        updateLeaderboard();
    }
}
```

LoginController:

```
package controller;
//imports
import java.awt.event.*;
import javax.swing.*;
```

```
import core.NetworkSystem;
import gui.CreateAccountWindow;
import gui.LoginWindow;

/**
 * @author Tabbie Brantley and Selin Topac
 * Login Controller for logging into the system and creating an account
 * @class invariant the currentSystem attribute is only for one instance (Singleton pattern)
 */
public class LoginController {
    //LonginWindow attribute loginWindowView
    private LoginWindow loginWindowView;
    //CreateAccountWindow attribute createAccountWindowView
    private CreateAccountWindow createAccountWindowView;
    private NetworkSystem currentSystem;

    /*
     * Constructor for LoginController
     * @param lView as LoginWindowView
     * @param curSystem the current NetworkSystem
     * @precondition n/a
     * @postcondition createAccountWindow is initiliazed
     */
    public LoginController(LoginWindow lView, CreateAccountWindow createView, NetworkSystem
curSystem) {
        this.loginWindowView = lView;
        this.createAccountWindowView = createView;
        this.currentSystem = curSystem;
        this.createListeners();
        this.createWindowListeners();
    }

    /**
     * createtListeners for creating the listeners
     * @precondition createAccountWindowView.getCreateAccountButton() != null &&
     * createAccountWindowView.getReturnToLoginButton() != null &&
     * createAccountWindowView.getJTextFields() != null
     * @postcondition Listeners for the create button and return button are created
     */
    public void createListeners(){

        //loginWindowView Listeners
        JButton loginButton = this.loginWindowView.getLoginButton();
        JButton goToCreateAccountButton = this.loginWindowView.getCreateAccountButton();
```

```
JTextField[] loginTextFields = this.loginWindowView.getJTextFields();
String[] loginInfo = new String[loginTextFields.length];

//add actionPerformed to the loginButton
loginButton.addActionListener(new
ActionListener() {
   public void actionPerformed(ActionEvent event){
      for (int i = 0; i < loginTextFields.length; i++){
         loginInfo[i] = loginTextFields[i].getText();
      }
      //check fields match length constraints
      if (LoginController.this.checkLoginLengths(loginInfo[0], loginInfo[1]) == false) {
         LoginController.this.loginWindowView.displayLengthConstraintMessage();
      }
      //check that username exists
      else if (LoginController.this.checkUsernameExists(loginInfo[0]) == false) {
         LoginController.this.loginWindowView.displayUsernameNotExistMessage();
      }
      else if (LoginController.this.verifyLogin(loginInfo[0], loginInfo[1]) == false){
         LoginController.this.loginWindowView.displayPasswordIsNotCorrectMessage();
      }
      else{
         LoginController.this.loginWindowView.closeLoginWindow();
         LoginController.this.currentSystem.openApplication();
      }

   }
});

//add actionPerformed to goToCreateAccountButton
goToCreateAccountButton.addActionListener(new
ActionListener() {
   public void actionPerformed(ActionEvent event){
      //close loginView and open createAccountView
      LoginController.this.loginWindowView.closeLoginWindow();
      LoginController.this.createAccountWindowView.openCreateAccountWindow();
}
});


//createAccountWindowView Listeners
JButton createButton = this.createAccountWindowView.getCreateAccountButton();
JButton returnButton = this.createAccountWindowView.getReturnToLoginButton();
```

```
        JTextField[] createTextFields = this.createAccountWindowView.getJTextFields();

        String[] accountInfo = new String[createTextFields.length];

        //add action performed for returnButton
        returnButton.addActionListener(new
           ActionListener() {
              public void actionPerformed(ActionEvent event){
                 //close createAccountWindowView and ope loginWindowView
                 LoginController.this.createAccountWindowView.closeCreateAccountWindow();
                 LoginController.this.loginWindowView.openLoginWindow();
        }
        });

        //add actionPerformed to createButton
        createButton.addActionListener(new
        ActionListener() {
           public void actionPerformed(ActionEvent event){
              for (int i = 0; i < createTextFields.length; i++){
                 accountInfo[i] = createTextFields[i].getText();
              }
              //check fields match length constraints
              if (LoginController.this.checkCreateLengths(accountInfo[0], accountInfo[1], accountInfo[2]) ==
false) {
                 LoginController.this.createAccountWindowView.displayLengthConstraintMessage();
              }
              //check that passwords match
              else if (LoginController.this.checkPasswords(accountInfo[2], accountInfo[3]) == false) {
                 LoginController.this.createAccountWindowView.displayPasswordNotMatchMessage();
              }
              else if (LoginController.this.currentSystem.checkUserExists(accountInfo[1])){

LoginController.this.createAccountWindowView.displayAccountNameAlreadyUsedMessage();
              } else {
                 currentSystem.addUserAccount(accountInfo[0], accountInfo[1], accountInfo[2]);
                 LoginController.this.createAccountWindowView.closeCreateAccountWindow();
                 LoginController.this.loginWindowView.openLoginWindow();
              }
           }
        });
     }


     /**
```

```
 * cheackLoginLenghts to match constraints
 * @param username as String
 * @param pwd as String
 * @return boolean value whether all fields match constraints
 * @precondition n/a
 * @postcondition n/a
 */
private boolean checkLoginLengths(String username, String pwd){
    return ((8 <= username.length()) && ((username.length()) <= 20)
            && ((8 <= pwd.length()) && ((pwd.length()) <= 20)));
}

/**
 * cheackUsernameExists to match constraints
 * @param username as String
 * @param pwd as String
 * @return boolean value username exists in the system
 * @precondition n/a
 * @postcondition n/a
 */
private boolean checkUsernameExists(String username){
    return this.currentSystem.usernameExists(username);
}

/**
 * verfifyLogin to check if user is able to login
 * @param username as String
 * @param pwd as String
 * @return boolean value username exists in the system
 * @precondition n/a
 * @postcondition n/a
 */
private boolean verifyLogin(String username, String pwd){
    return this.currentSystem.loginUser(username, pwd);
}


/**
 * checkCreatelenghts to match constraints
 * @param name as String
 * @param username as String
 * @param pwd as String
 * @return boolean value whether all fields match constraints
 * @precondition n/a
```

```
     * @postcondition n/a
     */
    public boolean checkCreateLengths(String name, String username, String pwd){
        return (((8 <= name.length()) && (name.length()<= 20))
                && ((8 <= username.length()) && ((username.length()) <= 20)
                && ((8 <= pwd.length()) && ((pwd.length()) <= 20))));
    }

    /**
     * Check if passwords match
     * @param pwd as String
     * @param confirmPwd as String
     * @return boolean value whether they match
     * @precondition n/a
     * @postcondition n/a
     */
    public boolean checkPasswords(String pwd, String confirmPwd){
        return (pwd.equals(confirmPwd));
    }

    /**
     * createWindowListeners creates listeners for the loginWindowView frame
     * and the createAccountWindowView frame
     * @precondition n/a
     * @postcondition n/a
     */
    public void createWindowListeners(){
        JFrame lFrame = this.loginWindowView.getLoginWindow(); // Assuming `getFrame()` returns the
main JFrame
        lFrame.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                NetworkSystem.getInstance().saveSystemState();
            }
        });
        JFrame cFrame = this.createAccountWindowView.getCreateAccountFrame(); // Assuming
`getFrame()` returns the main JFrame
        cFrame.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                NetworkSystem.getInstance().saveSystemState();
            }
        });
    }
```

```
}
```

LogoutController:

```
package controller;
import java.awt.event.*;
import javax.swing.*;

import core.NetworkSystem;
import gui.LogoutPanel;

/* Selin Topac */

public class LogoutController {
    private LogoutPanel logoutwindowView;
    private NetworkSystem currentSystem;


        /* Constructor for LogoutController */
        public LogoutController(LogoutPanel lView, NetworkSystem curSystem) {
            this.logoutwindowView = lView;
            this.currentSystem = curSystem;
            this.createListeners();
        }


    public void createListeners(){
        JButton logoutButton = this.logoutwindowView.getLogoutButton();

        logoutButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event){
                System.out.println("Logout button clicked");
                currentSystem.logout();

                //System.exit(0);


            }
        });
    }

}
```

NetworkWindowController:

```
package controller;
/**
```

```
 * @author TabbieBrantley
 * NetworkWindowController is the controller for the NetworkWindow
 * @class invariant listeners for the menubar buttons are added
 */

 //improts
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;

import core.NetworkSystem;
import gui.NetworkWindow;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;

public class NetworkWindowController {
   //NetworkWindow view attribute
   private NetworkWindow windowView;

   /**
    * Contsructor for the NetworkWindowController
    * @param netWindow the view as a NetworkWindow
    * @precondition n/a
    * @postcondition listeners are created
    */
   public NetworkWindowController(NetworkWindow netWindow){
      this.windowView = netWindow;
      this.createListeners();
      this.addNetworkWindowListener();
   }

   /**
    * createListeners creates listeners for the buttons that are on the MenuBarPanel
    * @precondition n/a
    * @postcondition n/a
    */
   private void createListeners(){
      //get all the buttons
      JButton homeB = this.windowView.getMenuBar().getHomeButton();
      JButton catB = this.windowView.getMenuBar().getCategoriesButton();
      JButton leadB = this.windowView.getMenuBar().getLeaderboardButton();
      JButton profileB = this.windowView.getMenuBar().getProfileButton();
```

```
        //adding ActionListener to the homeB
        homeB.addActionListener(new
        ActionListener() {
            public void actionPerformed(ActionEvent event){

NetworkWindowController.this.windowView.displayHomePage(NetworkSystem.getInstance().getCurrentU
ser());
            }
        });

        //adding ActionListener to the catB
        catB.addActionListener(new
        ActionListener() {
            public void actionPerformed(ActionEvent event){

NetworkWindowController.this.windowView.displayCategoriesPage(NetworkSystem.getInstance().getCurr
entUser());
            }
        });

        //adding ActionListener to the leadB
        leadB.addActionListener(new
        ActionListener() {
            public void actionPerformed(ActionEvent event){

NetworkWindowController.this.windowView.displayLeaderboardPage(NetworkSystem.getInstance().getC
urrentUser());
            }
        });

        //adding ActionLister to the profileB
        profileB.addActionListener(new
        ActionListener() {
            public void actionPerformed(ActionEvent event){

NetworkWindowController.this.windowView.displayProfilePage(NetworkSystem.getInstance().getCurrent
User());
            }
        });
    }
    /**
    * addNetworkWindowLIsteners adds a listener when the
    * windowView is closed
```

```
    * @precondition n/a
    * @postcondition n/a
    */
    public void addNetworkWindowListener(){
    JFrame frame = this.windowView.getWindowFrame();
    frame.addWindowListener(new WindowAdapter() {
        @Override
        public void windowClosing(WindowEvent e) {
            NetworkSystem.getInstance().saveSystemState();
        }
        });
    }
}
```

PostObserver:

```
package controller;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextArea;

import core.NetworkSystem;
import core.Post;
import core.Repost;

/**
 * The PostObserver class is responsible for handling user interactions with a post.
 * This includes actions such as agreeing, disagreeing, reposting, and deleting a post.
 * It updates the post's state and refreshes the profile feed based on the actions performed.
 * @author Hieu Truong
 */
public class PostObserver implements ActionListener {
    private Post post;

    /**
     * Constructs a PostObserver for the specified Post.
```

```
     * @param post: the Post object to be observed and interacted with.
     */
    public PostObserver(Post post) {
        this.post = post;
    }

    /**
     * Handles the actions performed on the observed post.
     * This includes actions such as agreeing, disagreeing, reposting, and deleting.
     * The post's points and interactions are updated accordingly, and the UI is refreshed.
     * @precondition e != null && e.getSource() instance of JButton
     * @postcondition The state of the post and user interface is updated based on the action performed.
     * @param e the ActionEvent triggered by the user's interaction.
     */
    @Override
    public void actionPerformed(ActionEvent e) {
        String currAccount = NetworkSystem.getInstance().getCurrentUser().getUsername();
        String action = e.getActionCommand();

        if (e.getSource() instanceof JButton) {
            if (action.equals("Agree")) {
                handleAgreeAction(currAccount);
            } else if (action.equals("Disagree")) {
                handleDisagreeAction(currAccount);
            } else if (action.equals("Repost")) {
                handleRepostAction();
            } else {
                handleDeleteAction();
            }
        }
    }

    /**
     * Handles the Agree action, updating the post's points and interactions.
     * @precondition  The action command of the event must be "Agree"
     * @postcondition The post's interaction is updated to "Agree" or cleared if re-clicked, change the
button color
     * @param currAccount: the username of the current user.
     */
    private void handleAgreeAction(String currAccount) {
        if (post.getInteract(currAccount).equals("Agree")) {
            //current user re-clicks agree, deduct 1 point, de highlight the agree button
            post.updatePoint(-1);
            post.removeInteract(currAccount);
```

```
            post.getAgreeButton().setBackground(Color.WHITE);
        } else if (post.getInteract(currAccount).equals("Disagree")) {
            //disagree->agree: +1+1; high-light agree; de-highlight disagree
            post.updatePoint(+2);
            post.addInteract(currAccount, "Agree");
            post.getAgreeButton().setBackground(new Color(219, 231, 252));
            post.getDisagreeButton().setBackground(Color.WHITE);
        } else {
            //from nothing->agree: +1; highlight agree button
            post.updatePoint(1);
            post.addInteract(currAccount, "Agree");
            post.getAgreeButton().setBackground(new Color(219, 231, 252));
        }
    }

    /**
     * Handles the Disagree action, updating the post's points and interactions.
     * @precondition The action command of the event must be "Disagree".
     * @postcondition The post's interaction is updated to "Disagree" or cleared if re-clicked, and change
the buttons color
     * @param currAccount the username of the current user.
     */
    private void handleDisagreeAction(String currAccount) {
        if (post.getInteract(currAccount).equals("Disagree")) {
            //current user re-clicks disagree, add 1 point, de-highlight the disagree button
            post.updatePoint(1);
            post.removeInteract(currAccount);
            post.getDisagreeButton().setBackground(Color.WHITE);
        } else if (post.getInteract(currAccount).equals("Agree")) {
            //agree->disagree: +1+1; high-light disagree; de-highlight agree
            post.updatePoint(-2);
            post.addInteract(currAccount, "Disagree");
            post.getDisagreeButton().setBackground(new Color(219, 231, 252));
            post.getAgreeButton().setBackground(Color.WHITE);
        } else {
            //from nothing->disagree: -1; highlight disagree button
            post.updatePoint(-1);
            post.addInteract(currAccount, "Disagree");
            post.getDisagreeButton().setBackground(new Color(219, 231, 252));
        }
    }


    /**
```

```
    * @author Selin Topac
    * Handles the Repost action
    * @precondition a post should already exist
    * @postcondition a frame that includes the original post and a new panel with a text area for the
repost is returned
    *
    * @return a frame that includes the original post and a new panel with a text area
    */
   private void handleRepostAction() {
      // creates a frame
      JFrame repostFrame = new JFrame();
      repostFrame.setLayout(new BorderLayout());
      // creates a text area for the user to add their repost text
      JTextArea textField = new JTextArea(5,50);
      textField.setForeground(Color.BLACK);
      textField.setLineWrap(true);
      textField.setWrapStyleWord(true);
      JPanel repostPanel = new JPanel();
      repostPanel.add(textField);

      // this will allow the user to cancel or post their repost after they add text
      // if their is no text and they click repost, it is not posted
      JButton repost = new JButton("Repost");
      repost.addActionListener((ActionEvent e) -> {
         if (!textField.getText().equals("")) {
            Post newRepost = new Repost(textField.getText(),
NetworkSystem.getInstance().getCurrentUser(), post);
            newRepost.setCategory(post.getCategory());
            NetworkSystem.getInstance().addPosts(newRepost);
         }
         repostFrame.dispose();
      });
      JButton cancel = new JButton("Cancel");
      cancel.addActionListener((ActionEvent e) -> {
         repostFrame.dispose();
      });

      // create a new panel for the buttons to be next to each other
      JPanel buttonPanel = new JPanel();
      buttonPanel.setLayout(new GridLayout(1,2));
      buttonPanel.add(repost);
      buttonPanel.add(cancel);
```

```
        // this will combine the panels into the frame
        repostFrame.add(post.getThisPostPanel(), BorderLayout.NORTH);
        repostFrame.add(repostPanel, BorderLayout.CENTER);
        repostFrame.add(buttonPanel, BorderLayout.SOUTH);
        repostFrame.setSize(500, 600);
        repostFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        repostFrame.setVisible(true);
        System.out.println("clicked repost");
        // return repostFrame;

    }


    /**
     * Handles the Delete action, prompting the user for confirmation and deleting the post if confirmed.
     * @precondition The action command of the event must be "Delete".
     * @postcondition If confirmed, the post is deleted from the network system, and the profile feed is
refreshed.
     */
    private void handleDeleteAction() {
        //set up the asking to confirm
        JFrame askingFrame = new JFrame("Confirm");
        askingFrame.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();

        JLabel command = new JLabel("Confirm to delete this post ?");
        JButton confirm = new JButton("Confirm");
        confirm.addActionListener((ActionEvent e) -> {
            askingFrame.dispose();
            NetworkSystem.getInstance().removePost(post);
            NetworkSystem.getInstance().getNetworkWindow().getProfilePagePanel().refreshProfileFeed();
        });
        JButton cancel = new JButton("Cancel");
        cancel.addActionListener((ActionEvent e) -> {
            askingFrame.dispose();
        });

        JPanel buttonPanel = new JPanel();
        buttonPanel.add(confirm);
        buttonPanel.add(cancel);

        c.gridx = 0;
        c.gridy = 0;
        askingFrame.add(command, c);
```

```
        c.gridx = 1;
        c.gridy = 0;
        askingFrame.add(buttonPanel, c);

        askingFrame.pack();
        askingFrame.setVisible(true);
    }
}
```

ProfileController:

```
package controller;
/**
 * @author Tabbie Brantley
 * The ProfileController class is the controller for the Profile
 * @class invariant user information can be updated only through the ProfileController
 */

import javax.swing.*;

import core.UserAccount;
import gui.ProfilePanel;

import java.awt.*;
import java.awt.event.*;

public class ProfileController {
    //attributes
    private ProfilePanel profileView;
    private JButton editNameB;
    private JButton cancelNameB;
    private JButton saveNameB;
    private JButton editPasswordB;
    private JButton cancelPasswordB;
    private JButton savePasswordB;
    private UserAccount currentUser;

    /**
     * Constructor for profile panel
     * @param profilePanelView the view as a ProfilePanel
     * @param curUser the current user as a UserAccount
     * @precondition n/a
     * @postcondition listeners are created
     */
    public ProfileController(ProfilePanel profilePanelView, UserAccount curUser){
```

```
    this.profileView = profilePanelView;
    this.currentUser = curUser;
    this.createProfileListeners();
}

/**
 * createPorfileLIsteners vreates listeners for the profile
 * @precondition n/a
 * @postcondition listeners are creates
 */
public void createProfileListeners(){

    this.editNameB = profileView.getEditNameButton();
    //add action listener to editNameB
    this.editNameB.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent event){
            ProfileController.this.profileView.editName();
        }
    });

    this.cancelNameB = profileView.getCancelNameButton();

    //add action listener to cancelNameB
    this.cancelNameB.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent event){
            ProfileController.this.profileView.cancelNameEdit();
        }
    });

    this.saveNameB = profileView.getSaveNameButton();

    //add action listener to saveNameB
    this.saveNameB.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent event){
            String newName = profileView.getNewName().getText();
            if (ProfileController.this.checkNameLength(newName)){
                ProfileController.this.changeName(newName);
                ProfileController.this.profileView.closeEditName();
            }
            else{
                ProfileController.this.profileView.displayNameLengthErrorMessage();
            }
        }
    });
```

```
    this.editPasswordB = profileView.getEditPasswordButton();
    //add action listener to editPasswordB
    this.editPasswordB.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent event){
            ProfileController.this.profileView.editPassword();
        }
    });

    this.cancelPasswordB = profileView.getCancelPasswordEditButton();
    //add action listener to cancelPasswordB
    this.cancelPasswordB.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent event){
            ProfileController.this.profileView.cancelPasswordEdit();
        }
    });

    this.savePasswordB = profileView.getSavePasswordButton();
    //add action listener to savePasswordB
    this.savePasswordB.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent event){
            String newPassword = profileView.getNewPassword().getText();
            String confirmedPassword = profileView.getConfirmedPassword().getText();
            if ((ProfileController.this.checkPasswordLength(newPassword)) &&
                (ProfileController.this.checkPasswordsMatch(newPassword, confirmedPassword))){
                ProfileController.this.changePassword(newPassword);
                ProfileController.this.profileView.closeEditPassword();
            } else if(!ProfileController.this.checkPasswordLength(newPassword)){
                ProfileController.this.profileView.displayPasswordLengthErrorMessage();
            } else {
                ProfileController.this.profileView.displayPasswordMatchErrorMessage();
            }

        }
    });
}

/**
 * checkNameLength checks the length of a String to meet requirements
 * @param name the name as a String
 * @return boolean value, true if length is between [8, 20]
 * @precondtion n/a
 * @postcondition n/a
```

```
 */
public boolean checkNameLength(String name){
    return ((8 <= name.length()) && (name.length()<= 20));
}

/**
 * changeName changes the name of the currentUser
 * @param name the new name as a String
 * @precondition n/a
 * @postcondition currentUser's name is updated and the profileView displays the new name
 */
public void changeName(String name){
    this.currentUser.setName(name);
    this.profileView.updateName(name);
}

/**
 * checkPassowordLength checks the length of a String to meet requirements
 * @param pwd is the new password as a String
 * @return boolean value, true if length is between [8, 20]
 * @precondtion n/a
 * @postcondition n/a
 */
public boolean checkPasswordLength(String pwd){
    return ((8 <= pwd.length()) && (pwd.length()<= 20));
}


/**
 * checkPassowordsMatch checks whether two passwords are the same
 * @param pwd is the new password, and confirmedPwd is the confirmed password
 * @return boolean value, true if they the sequence of characters equal each other,
 * false if not
 * @precondtion n/a
 * @postcondition n/a
 */
public boolean checkPasswordsMatch(String pwd, String confirmedPwd){
    return pwd.equals(confirmedPwd);
}

/**
 * changePassword changes the password of the currentUser
 * @param pwd is the new password as a String
 * @precondition n/a
```

```
     * @postcondition currentUser's password is updated
     */
    public void changePassword(String pwd){
        this.currentUser.setPassword(pwd);
    }
}
```

CategoryButtonPanel:

```
package gui;

import java.awt.Component;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;


/**
 * @author  Lewis Cox
 * CategoryButtonPanel contains buttons for each category allowing users to filter posts
 * by category
 *
 */
public class CategoryButtonPanel extends JPanel {

    /**
     * constructs the CategoryButtonPanel
     */
    public CategoryButtonPanel() {
        this.setLayout(new FlowLayout());

        // create buttons for each category
        String[] categories = {"Sports", "Technology", "Art", "News"};

        for (String category : categories) {
            JButton categoryButton = new JButton(category);
            categoryButton.addActionListener(new CategoryButtonListener(category));
            this.add(categoryButton);
        }
    }
```

```
    /**
     * updates the CategoryPage with the selected category
     */
    private class CategoryButtonListener implements ActionListener {
        private String category;

        /**
         * constructs a listener for a specific category button
         *
         * @param category category that the button represents
         */
        public CategoryButtonListener(String category) {
            this.category = category;
        }

        @Override
        public void actionPerformed(ActionEvent e) {

            CategoryPage categoryPage = (CategoryPage)
SwingUtilities.getAncestorOfClass(CategoryPage.class, (Component) e.getSource());
            if (categoryPage != null) {
                categoryPage.updateCategoryPosts(category);
            }
        }
    }
}
```

CategoryPage:

```
package gui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;

import javax.swing.BoxLayout;
import javax.swing.JPanel;
import javax.swing.JScrollPane;

import core.NetworkSystem;
import core.Post;
import utility.GeneralPageSortStrategy;
```

```
/**
 * @author Lewis Cox
 * CategoryPage is a JPanel that shows the posts and are  filtered by category
 *
 */
public class CategoryPage extends JPanel{
    private JPanel postPanel;
    private JScrollPane scrollPane;

    /**
     *
     * CategoryPage
     * layout with category selection buttons and post display panel
     */
    public CategoryPage(){
        postPanel = new JPanel();
        postPanel.setLayout(new BoxLayout(postPanel, BoxLayout.Y_AXIS));

        this.setLayout(new BorderLayout());


        JPanel west = new JPanel();
        JPanel east = new JPanel();


        int panelWidth = NetworkSystem.getInstance().getWidth() / 5;
        west.setPreferredSize(new Dimension(panelWidth, this.getHeight()));
        east.setPreferredSize(new Dimension(panelWidth, this.getHeight()));


        west.setBackground(new Color(219, 231, 252));
        east.setBackground(new Color(219, 231, 252));


        this.add(new CategoryButtonPanel(), BorderLayout.NORTH);


        this.add(postPanel, BorderLayout.CENTER);


        this.add(west, BorderLayout.WEST);
        this.add(east, BorderLayout.EAST);
```

```
        setScrollPane();
        this.add(scrollPane, BorderLayout.CENTER);

    }

    /**
     * displayed posts based on the selected category
     *
     * @param category the category whose posts should be displayed
     */
    public void updateCategoryPosts(String category) {
        postPanel.removeAll();

        // loops through the list of all posts
        for (Post post : NetworkSystem.getInstance().getPostsList()) {
            // post category matches the selected category and adds it to the panel
            if (post.getCategory().equals(category)) {
                postPanel.add(post.getCompletePostPanel(new GeneralPageSortStrategy()));
            }
        }


        this.revalidate();
        this.repaint();
    }

    /**
     * Wraps the post panel in a scroll pane.
     *
     */
    public final void setScrollPane() {
        scrollPane = new JScrollPane(postPanel);
        scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
        scrollPane.getVerticalScrollBar().setValue(0);
    }
}
```

CreateAccountWindow:

```
package gui;
/**
 * @author Tabbie Brantley
 * CreateAccountWindow window view for creating an account
 * @class invariant.....
 */
```

```
//imports
import java.awt.BorderLayout;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;

public class CreateAccountWindow {
    //frame and panel
    private JFrame createAccountFrame;
    private JPanel createAccountPanel;

    //buttons
    private JButton createAccountButton;
    private JButton returnToLoginButton;

    //labels
    private JLabel instructionsLabel;
    private JLabel accountNameLabel;
    private JLabel usernameLabel;
    private JLabel passwordLabel;
    private JLabel confirmPasswordLabel;

    //JTextFields and JTextField arra
    private JTextField[] textFields = new JTextField[4];
    private JTextField accountNameTextField;
    private JTextField usernameTextField;
    private JTextField passwordTextField;
    private JTextField confirmPasswordTextField;

    //GridBagConstraint
    private GridBagConstraints c;

    /**
     * Constructor for CreateAccountWindow
```

```
     * @precondition n/a
     * @postcondition JFrame window is created and is set to be visible
     */
    public CreateAccountWindow(){

        //creating JFrame and JPanel with JPanel has GridBagLayout
        this.createAccountFrame = new JFrame();
        this.createAccountFrame.setLayout(new BorderLayout());
        this.createAccountPanel = new JPanel();
        this.createAccountFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //creating instructions label
        this.instructionsLabel = new JLabel("Enter in new account information. Press Create Account when
done and you will be brought"
        + " back to the login page. Press Return to Login if you do not wish to create a new account.");
        this.instructionsLabel.setBorder(new EmptyBorder(200, 300, 0, 200));

        //creating create account panel
        this.createAccountPanel.setLayout(new GridBagLayout());
        this.c = new GridBagConstraints();
        this.c.fill = GridBagConstraints.HORIZONTAL;
        this.c.insets = new Insets(5, 5, 5,  5);

        //Creating labels
        this.accountNameLabel = new JLabel("Enter an account name:");
        this.usernameLabel = new JLabel("Enter a username:");
        this.passwordLabel = new JLabel("Enter a password:");
        this.confirmPasswordLabel = new JLabel("Confirm password:");

        //Createing text fields
        this.accountNameTextField = new JTextField(20);
        this.usernameTextField  = new JTextField(40);
        this.passwordTextField = new JTextField(40);
        this.confirmPasswordTextField = new JTextField(40);
        this.textFields[0] = this.accountNameTextField;
        this.textFields[1] = this.usernameTextField;
        this.textFields[2] = this.passwordTextField;
        this.textFields[3] = this.confirmPasswordTextField;

        //creating buttons
        this.createAccountButton = new JButton("Create Account");
        this.returnToLoginButton = new JButton("Return to Login");

        //adding labels, text fields, and buttons to the panel
```

```
this.c.gridx = 0;
this.c.gridy = 0;
this.createAccountPanel.add(this.accountNameLabel, this.c);

this.c.gridx = 1;
this.c.gridy = 0;
this.createAccountPanel.add(this.accountNameTextField, this.c);

this.c.gridx = 0;
this.c.gridy = 1;
this.createAccountPanel.add(this.usernameLabel, this.c);

this.c.gridx = 1;
this.c.gridy = 1;
this.createAccountPanel.add(this.usernameTextField, this.c);

this.c.gridx = 0;
this.c.gridy = 2;
this.createAccountPanel.add(this.passwordLabel, this.c);

this.c.gridx = 1;
this.c.gridy = 2;
this.createAccountPanel.add(this.passwordTextField, this.c);

this.c.gridx = 0;
this.c.gridy = 3;
this.createAccountPanel.add(this.confirmPasswordLabel, this.c);

this.c.gridx = 1;
this.c.gridy = 3;
this.createAccountPanel.add(this.confirmPasswordTextField, this.c);

this.c.gridx = 0;
this.c.gridy = 4;
this.createAccountPanel.add(this.returnToLoginButton, this.c);

this.c.gridx = 1;
this.c.gridy = 4;
this.createAccountPanel.add(this.createAccountButton, this.c);

//adding panel to frame, and setting frame to MAXIMIZED_BOTH, and setting visible true to pane
this.createAccountFrame.add(this.instructionsLabel, BorderLayout.NORTH);
this.createAccountFrame.add(createAccountPanel, BorderLayout.CENTER);
this.createAccountFrame.setExtendedState(JFrame.MAXIMIZED_BOTH);
```

```
    //this.createAccountFrame.setVisible(true);
}

/**
 * Getter for createAccountButton
 * @return createAccountBUtton as JButton
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getCreateAccountButton(){
    return this.createAccountButton;
}

/**
 * Getter for eturnToLoginButton
 * @return eturnToLoginButton as JButton
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getReturnToLoginButton(){
    return this.returnToLoginButton;
}

/**
 * Getter for the textFields
 * @return textFields as JTextField[]
 * @precondition n/a
 * @postcondition the reference of the textfields is not returned, only a copy
 */
public JTextField[] getJTextFields() {
    return this.textFields.clone();
}

/**
 * openCreateAccountWindow toopen the window
 * @precondition n/a
 * @postcondition createAccountFrame is visible
 */
public void openCreateAccountWindow(){
    this.createAccountFrame.setVisible(true);
}

/**
 * closeCreateAccountWindow to close the window
```

```
     * @precondition n/a
     * @postcondition createAccountFrame is no longer visible
     */
    public void closeCreateAccountWindow(){
        this.createAccountFrame.setVisible(false);
    }

    /**
     * displayLengthConstraintMessage displays message for constraint in a JOptionPane
     * @precondition n/a
     * @postcondition n/a
     */
    public void displayLengthConstraintMessage(){
        JOptionPane.showMessageDialog(this.createAccountFrame, "Error: Account Name, Username, and
password must be between 8-20 characters.");
    }

   /**
     * displayPasswordNotMatchMessage displays message for unmatching passwords in a JOptionPane
     * @precondition n/a
     * @postcondition n/a
     */
    public void displayPasswordNotMatchMessage(){
        JOptionPane.showMessageDialog(this.createAccountFrame, "Error: Passwords do not match.");
    }

    /**
     * displayAccountNameAlreadyUsedMessage displays message for a username already in use in a
JOptionPane
     * @precondition n/a
     * @postcondition n/a
     */
    public void displayAccountNameAlreadyUsedMessage(){
        JOptionPane.showMessageDialog(this.createAccountFrame, "Error: Username has already been
used. Enter a different "
        + "username.");
    }

    /**
     * Accessor for the createAccountFrame
     * @return this.createAccountFrame as a JFrame
     * @precondition n/a
     * @postcondition n/a
     */
```

```
    public JFrame getCreateAccountFrame(){
        return this.createAccountFrame;
    }
}
```

FeedPanel:

```
package gui;


import java.awt.Color;
import java.util.ArrayList;

import javax.swing.BoxLayout;
import javax.swing.JPanel;

import utility.SortPostStrategy;
import core.Post;

/**
 * The FeedPanel class represents a panel for displaying a feed of posts.
 * It dynamically populates itself based on the sorting strategy provided.
 * The feed is displayed as a vertical list of posts.
 * @author Hieu Truong
 */
public class FeedPanel extends JPanel {
    private SortPostStrategy sortStrategy;
    private ArrayList<Post> posts;

    /**
     * Constructs a FeedPanel object using the specified sorting strategy.
     * This constructor initializes the feed by sorting posts and displaying them.
     *
     * @param strategy the sorting strategy used to determine the order of posts.
     * @throws IllegalArgumentException if the strategy is null
     * @precondition strategy != null: A valid sorting strategy must be provided.
     * @postcondition The panel is initialized with posts sorted according to the given strategy.
     */
    public FeedPanel(SortPostStrategy strategy) {
        if (strategy == null) {
            throw new IllegalArgumentException("Sorting strategy cannot be null.");
        }
        sortStrategy = strategy;
        posts = strategy.sortPost();
        this.displayFeed();
```

```
        this.repaint();
    }

    /**
     * Displays the feed by adding the sorted posts to the panel in a vertical layout.
     * Each post is represented as a complete post panel, rendered in the order specified by the sorting
strategy.
     *
     * @return the current instance of FeedPanel for method chaining.
     * @pre posts != null && !posts.isEmpty(): Posts must be initialized and non-empty.
     * @post The feed panel is populated with all posts in the correct order.
     */
    public final FeedPanel displayFeed() {
        this.setBackground(Color.WHITE);
        this.setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
        System.out.println("Display profile feed posts");
        for (int i = 0; i < posts.size(); i++) {
            this.add(posts.get(i).getCompletePostPanel(sortStrategy));
            System.out.println(posts.get(i));
        }
        return this;
    }
}
```

HomePageFeedPanel:

```
package gui;
import java.awt.BorderLayout;

import javax.swing.JPanel;
import utility.GeneralPageSortStrategy;

/**
 * @author Tabbie Brantley
 * HomePageFeedPanel is the panel that holds the home page feed
 * @class invariant only the HomePageSortStrategy is used
 */
public class HomePageFeedPanel extends JPanel{

    private FeedPanel feedPanel;

    /**
     * Constructor for HomePageFeedPanel
     * @precondition n/a
     * @postcondition n/a
```

```
     */
    public HomePageFeedPanel(){
        this.feedPanel = new FeedPanel(new GeneralPageSortStrategy());
        this.setLayout(new BorderLayout());;
        this.add(this.feedPanel, BorderLayout.CENTER);
    }

     /**
     * createHomePageFeedPanel repaints to refresh the panel
     * @return the current instance
     * @precondition n/a
     * @postcondition n/a
     */
    public HomePageFeedPanel createHomePageFeedPanel(){
        this.feedPanel.repaint();
        return this;
    }

}
```

HomePagePanel:

```
package gui;
/**
 * @author Tabbie Brantley
 * HomePagePanel is the panel that holds the home page
 * @class invariant the HomePageSortStrategy will only be used with the homeFeedPanel
 */

//imports
import core.NetworkSystem;
import core.UserAccount;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;

import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.SwingUtilities;

public class HomePagePanel extends JPanel {
    private HomePageFeedPanel homeFeedPanel;
    private JScrollPane scrollPane;

    /**
     * Constructor for HomePagePanel
```

```
     * @precondition n/a
     * @postcondition n/a
     */
    public HomePagePanel() {
        UserAccount curUser = NetworkSystem.getInstance().getCurrentUser();
        this.homeFeedPanel = new HomePageFeedPanel();
        JPanel west = new JPanel();
        JPanel east = new JPanel();

        int panelWidth = NetworkSystem.getInstance().getWidth() / 5;
        west.setPreferredSize(new Dimension(panelWidth, this.getHeight()));
        east.setPreferredSize(new Dimension(panelWidth, this.getHeight()));

        west.setBackground(new Color(219, 231, 252));
        east.setBackground(new Color(219, 231, 252));

        this.setLayout(new BorderLayout());
        setScrollPane();
        this.add(scrollPane, BorderLayout.CENTER); // Add the scroll pane here
        this.add(west, BorderLayout.WEST);
        this.add(east, BorderLayout.EAST);
    }

    /**
     * setScrollPane sets a scroll pane for the homeFeedPanel
     * @precondition n/a
     * @postcodntion n/a
     */
    public final void setScrollPane() {
        scrollPane = new JScrollPane(this.homeFeedPanel);
        scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
        scrollPane.getVerticalScrollBar().setValue(0);
        SwingUtilities.invokeLater(() -> scrollPane.getVerticalScrollBar().setValue(0));
    }
}
```

LeaderboardPage:

```
package gui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.util.List;
```

```
import javax.swing.DefaultListModel;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;
import javax.swing.SwingConstants;

import core.NetworkSystem;
import core.UserAccount;

/**
 * @author Lewis Cox
 * LeaderboardPage displays the leaderboard for users and it sorted by points
 */
public class LeaderboardPage extends JPanel {
    private JList<String> leaderboardList;
    private JScrollPane scrollPane;
    private JPanel listUser;
    /**
     * Constructor for the LeaderboardPage
     * Initializes the layout and adds users to the leaderboard
     *
     * @param users list of UserAccount objects to be displayed in the leaderboard
     */
    public LeaderboardPage(List<UserAccount> users) {
        this.setLayout(new BorderLayout());


        JPanel west = createSidePanel();
        JPanel east = createSidePanel();


        JPanel centerPanel = new JPanel(new BorderLayout());
        centerPanel.setBackground(Color.WHITE);

        // Add leaderboard rank label to the center panel
        JLabel leaderboardLabel = new JLabel("LEADERBOARD RANK", SwingConstants.CENTER);
        leaderboardLabel.setOpaque(true);
        leaderboardLabel.setBackground(new Color(173, 216, 230));
        leaderboardLabel.setPreferredSize(new Dimension(centerPanel.getWidth(), 100));
        centerPanel.add(leaderboardLabel, BorderLayout.NORTH); // Add label to the top of the center
panel
```

```
        // Populate the leaderboard and add the scrollPane to the center panel
        populateLeaderboard(users);
        centerPanel.add(scrollPane, BorderLayout.CENTER);

        this.add(centerPanel, BorderLayout.CENTER);
        this.add(west, BorderLayout.WEST);
        this.add(east, BorderLayout.EAST);
    }

    private JPanel createSidePanel() {
        JPanel sidePanel = new JPanel();
        int panelWidth = NetworkSystem.getInstance().getWidth() / 5;
        sidePanel.setPreferredSize(new Dimension(panelWidth, this.getHeight()));
        sidePanel.setBackground(new Color(219, 231, 252));
        return sidePanel;
    }

    /**
     * adds to the leaderboard with the list of users
     *
     * @param users list of UserAccount objects to be displayed in the leaderboard
     */
    public void populateLeaderboard(List<UserAccount> users) {

        DefaultListModel<String> listModel = new DefaultListModel<>();

        // sorts users by points in descending order and add to the list model
        users.stream()
            .sorted((u1, u2) -> Integer.compare(u2.getPoints(), u1.getPoints()))
            .limit(5)
            .forEach(user -> listModel.addElement(user.getUsername() + ": " + user.getPoints() + " points"));


        leaderboardList = new JList<>(listModel);
        leaderboardList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

        scrollPane = new JScrollPane(leaderboardList);
        scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
        scrollPane.getVerticalScrollBar().setValue(0);
    }
}
```

LoginWindow:

```
package gui;
/**
 * @author Selin Topac
 * loginWindow window view for signing in
 * @class invariant.....
 */


//imports
import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;

public class LoginWindow {
    //frame and panel
    private JFrame loginFrame;
    private JPanel loginPanel;

    //buttons
    private JButton loginButton;
    private JButton CreateAccountButton;

    //labels
    private JLabel instructionsLabel;
    private JLabel usernameLabel;
    private JLabel passwordLabel;

    //JTextFields and JTextField arra
    private JTextField[] textFields = new JTextField[2];
    private JTextField usernameTextField;
    private JTextField passwordTextField;

    //GridBagConstraint
    private GridBagConstraints c;

    /**
     * Constructor for loginWindow
         * @return
         * @precondition n/a
         * @postcondition JFrame window is created and is set to be visible
         */
        public LoginWindow(){

        //creating JFrame and JPanel with JPanel has GridBagLayout
```

```
        this.loginFrame = new JFrame();
        this.loginFrame.setLayout(new BorderLayout());
        this.loginPanel = new JPanel();
        this.loginFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //creating instructions label
        this.instructionsLabel = new JLabel("Enter your username and password. If you do not have an
account, click Create Account");
        this.instructionsLabel.setBorder(new EmptyBorder(200, 300, 0, 200));

        //creating create account panel
        this.loginPanel.setLayout(new GridBagLayout());
        this.c = new GridBagConstraints();
        this.c.fill = GridBagConstraints.HORIZONTAL;
        this.c.insets = new Insets(5, 5, 5,  5);

        //Creating labels
        this.usernameLabel = new JLabel("Enter a username:");
        this.passwordLabel = new JLabel("Enter a password:");

        //Createing text fields
        this.usernameTextField  = new JTextField(40);
        this.passwordTextField = new JTextField(40);
        this.textFields[0] = this.usernameTextField;
        this.textFields[1] = this.passwordTextField;

        //creating buttons
        this.loginButton = new JButton("Login");
        this.CreateAccountButton = new JButton("Create an Account");

        //adding labels, text fields, and buttons to the panel

        this.c.gridx = 0;
        this.c.gridy = 1;
        this.loginPanel.add(this.usernameLabel, this.c);

        this.c.gridx = 1;
        this.c.gridy = 1;
        this.loginPanel.add(this.usernameTextField, this.c);

        this.c.gridx = 0;
        this.c.gridy = 2;
        this.loginPanel.add(this.passwordLabel, this.c);
```

```
    this.c.gridx = 1;
    this.c.gridy = 2;
    this.loginPanel.add(this.passwordTextField, this.c);

    this.c.gridx = 0;
    this.c.gridy = 4;
    this.loginPanel.add(this.CreateAccountButton, this.c);

    this.c.gridx = 1;
    this.c.gridy = 4;
    this.loginPanel.add(this.loginButton, this.c);

    //adding panel to frame, and setting frame to MAXIMIZED_BOTH, and setting visible true to pane
    this.loginFrame.add(this.instructionsLabel, BorderLayout.NORTH);
    this.loginFrame.add(loginPanel, BorderLayout.CENTER);
    this.loginFrame.setExtendedState(JFrame.MAXIMIZED_BOTH);
    this.loginFrame.setVisible(true);
}



/**
 * Getter for loginButton
 * @return loginButton as JButton
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getLoginButton(){
    return this.loginButton;
}

/**
 * Getter for eturnToLoginButton
 * @return eturnToLoginButton as JButton
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getCreateAccountButton(){
    return this.CreateAccountButton;
}

/**
 * Getter for the textFields
 * @return textFields as JTextField[]
```

```
     * @precondition n/a
     * @postcondition the reference of the textfields is not returned, only a copy
     */
    public JTextField[] getJTextFields() {
        return this.textFields.clone();
    }

    /**
     * openLoginWindow to open the window
     * @precondition n/a
     * @postcondition loginFrame is visible
     */
    public void openLoginWindow(){
        this.loginFrame.setVisible(true);
    }

    /**
     * closeloginWindow to close the window
     * @precondition n/a
     * @postcondition loginFrame is no longer visible
     */
    public void closeLoginWindow(){
        this.loginFrame.setVisible(false);
    }

    /**
     * displayLengthConstraintMessage displays message for constraint in a JOptionPane
     * @precondition n/a
     * @postcondition n/a
     */
    public void displayLengthConstraintMessage(){
        JOptionPane.showMessageDialog(this.loginFrame, "Error: Username and password must be
between 8-20 characters.");
    }

    /**
     * displayUsernameNotExistMessage displays message in a JOptionPane
     * @precondition n/a
     * @postcondition n/a
     */
    public void displayUsernameNotExistMessage(){
        JOptionPane.showMessageDialog(this.loginFrame, "Error: Username does not exist.");
    }
```

```
    /**
     * displayPasswordIsNotCorrectMessage in a JOptionPane
     * @precondition n/a
     * @postcondition n/a
     */
    public void displayPasswordIsNotCorrectMessage(){
        JOptionPane.showMessageDialog(this.loginFrame, "Password is not correct.");
    }

    /**
     * Accessor for loginFrane
     * @return this.loginFrame as a JFrame
     * @precondition n/a
     * @postcondition n/a
     */
    public JFrame getLoginWindow(){
        return this.loginFrame;
    }

}
```

LogoutPanel:

```
package gui;
/**
 * @author Tabbie Brantley
 * LogoutPanel extends JPanel
 * @class invariant the logoutButton is shown
 */

//imports
import java.awt.*;
import javax.swing.*;

public class LogoutPanel extends JPanel{
    private JButton logoutButton;
    private GridBagConstraints c;

    /**
    * Constructor for LogoutPanel
    * @precondition n/a
    * @postcondition n/a
    */
    public LogoutPanel(){
        this.logoutButton = new JButton("Logout");
```

```
        this.c = new GridBagConstraints();
        this.c.fill = GridBagConstraints.HORIZONTAL;
        this.c.insets = new Insets(5, 5, 5,  5);

        this.setLayout(new GridBagLayout());
        this.c.gridx = 0;
        this.c.gridy = 0;
        this.add(this.logoutButton, this.c);
    }

    /**
     * Accessor for logoutButton
     * @return this.logoutButton as JButton
     * @precondition n/a
     * @postcondition n/a
     */
    public JButton getLogoutButton(){
        return this.logoutButton;
    }
}
```

MenuBarPanel:

```
package gui;
/**
 * @auther Tabbie Brantley
 * MenuBarPanel extends JPanel and displays the menu buttons
 * @class invariant buttons are added
 */

//import
import java.awt.Color;
import java.awt.GridLayout;

import javax.swing.JButton;
import javax.swing.JPanel;

public class MenuBarPanel extends JPanel {
    private JButton homeButton;
    private JButton categoriesButton;
    private JButton leaderboardButton;
    private JButton profileButton;

    /**
     * Constructor for MenuBarPanel
```

```
 * @precondition n/a
 * @postcondition n/a
 */
public MenuBarPanel(){
    this.setLayout(new GridLayout(1, 4));
    this.setBackground(new Color(219, 231, 252));
    this.homeButton = new JButton("Home");
    this.categoriesButton = new JButton("Categories");
    this.leaderboardButton = new JButton("Leaderboard");
    this.profileButton = new JButton("Profile");

    this.add(this.homeButton);
    this.add(this.categoriesButton);
    this.add(this.leaderboardButton);
    this.add(this.profileButton);

}

/**
 * Accessor for this.homeButton
 * @return this.homeButton as a JButton
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getHomeButton(){
    return this.homeButton;
}

/**
 * Accessor for this.homeButton
 * @return this.homeButton as a JButton
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getCategoriesButton(){
    return this.categoriesButton;
}

/**
 * Accessor for this.leaderboardButton
 * @return this.leaderboardButton as a JButton
 * @precondition n/a
 * @postcondition n/a
 */
```

```
    public JButton getLeaderboardButton(){
        return this.leaderboardButton;
    }

    /**
     * Accessor for this.profileButton
     * @return this.profileButton as a JButton
     * @precondition n/a
     * @postcondition n/a
     */
    public JButton getProfileButton(){
        return this.profileButton;
    }

    /**
     * Accessor for this current instance of MenuBarPanel
     * @return this as a MenuBarPanel
     * @precondition n/a
     * @postcondition n/a
     */
    public MenuBarPanel getMenuBarPanel(){
        return this;
    }
}
```

NetworkWindow:

```
package gui;

/**
 * @author all
 * CreateAccountWindow window view for creating an account
 * @class invariant.....
 */


//imports
import java.awt.BorderLayout;
import java.awt.GridBagConstraints;
import java.util.List;

import javax.swing.JFrame;
import javax.swing.JPanel;

import core.NetworkSystem;
```

```
import core.UserAccount;

public class NetworkWindow {

    //GridBagConstraint
    private GridBagConstraints c;
    private JFrame windowFrame;
    private JPanel mainPanel;

    private ProfilePagePanel profilePage;
    private MenuBarPanel menuBarPanel;
    private HomePagePanel homePage;
    private LeaderboardPage leaderboardPage;
    private CategoryPage categoriesPage;


    /**
     * Constructor for NetworkWindow
     * @precondition n/a
     * @postcondition n/a
     */
    public NetworkWindow() {

        //creating JFrame and JPanel with JPanel has GridBagLayout
        this.windowFrame = new JFrame();
        this.windowFrame.setLayout(new BorderLayout());
        //this.windowFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        this.menuBarPanel = new MenuBarPanel();
        this.windowFrame.add(this.menuBarPanel, BorderLayout.NORTH);
        this.windowFrame.setExtendedState(JFrame.MAXIMIZED_BOTH);

    }

    /**
     * displayWindow sets the windowFrame to visible
     * @precondition n/a
     * @postcondition n/a
     */
    public void displayWindow(){
        this.windowFrame.setVisible(true);
    }
```

```
/**
 * closeWindow sets the windowFrame visible to false
 * @precondition n/a
 * @postcondition n/a
 */
public void closeWindow(){
    this.windowFrame.setVisible(false);
}

/**
 * displayProfilePage removes content from the windowFrame and then
 * add the menuBarPanel and the profilePage
 * @param curUser the current user logged into the system as a UserAccount
 * @preocdintion n/a
 * @postcondition n/a
 */
public void displayProfilePage(UserAccount curUser){
    this.windowFrame.getContentPane().removeAll();
    this.windowFrame.add(this.menuBarPanel, BorderLayout.NORTH);

    this.profilePage = new ProfilePagePanel();

    windowFrame.add(this.profilePage, BorderLayout.CENTER);
    windowFrame.revalidate();
    windowFrame.repaint();
}

/**
 * Accessor for profilePage
 * @return this.profilePage as a ProfilePagePanel
 * @precondition n/a
 * @postcondition n/a
 */
public ProfilePagePanel getProfilePagePanel(){
    return this.profilePage;
}

/**
 * displayHomePage removes content from the windowFrame and then
 * add the menuBarPanel and the HomePage
 * @param curUser the current user logged into the system as a UserAccount
 * @preocdintion n/a
 * @postcondition n/a
 */
```

```
    public void displayHomePage(UserAccount curUser){
        this.windowFrame.getContentPane().removeAll();

        this.windowFrame.add(this.menuBarPanel, BorderLayout.NORTH);

        this.homePage = new HomePagePanel();

        this.windowFrame.add(homePage, BorderLayout.CENTER);
        this.windowFrame.revalidate();
        this.windowFrame.repaint();
    }


/**
     * displayLeaderboardPage removes content from the windowFrame and then
     * add the menuBarPanel and the leaderboardPage
     * @param curUser the current user logged into the system as a UserAccount
     * @preocdintion n/a
     * @postcondition n/a
     */
    public void displayLeaderboardPage(UserAccount curUser){

        this.windowFrame.getContentPane().removeAll();

        this.windowFrame.add(this.menuBarPanel, BorderLayout.NORTH);


        List<UserAccount> topUsers = NetworkSystem.getInstance().getTopUsers();

        System.out.println("DISPLAY LEADER BOEAD");
        System.out.println(topUsers);

        this.leaderboardPage = new LeaderboardPage(topUsers);

        this.windowFrame.add(this.leaderboardPage, BorderLayout.CENTER);
        this.windowFrame.revalidate();
        this.windowFrame.repaint();
    }



    /**
     * displayCategoriesPage removes content from the windowFrame and then
     * add the menuBarPanel and the categoriesPage
```

```
     * @param curUser the current user logged into the system as a UserAccount
     * @preocdintion n/a
     * @postcondition n/a
     */
     public void displayCategoriesPage(UserAccount curUser) {
        this.windowFrame.getContentPane().removeAll();

        this.windowFrame.add(this.menuBarPanel, BorderLayout.NORTH);



        this.categoriesPage = new CategoryPage();

        this.windowFrame.add(this.categoriesPage, BorderLayout.CENTER);
        this.windowFrame.revalidate();
        this.windowFrame.repaint();
    }


    /**
     * Accessor for the menuBarPanel
     * @return this.menuBarPane as a MenuBarPanel
     * @precondition n/a
     * @postcondition n/a
     */
    public MenuBarPanel getMenuBar(){
        return this.menuBarPanel;
    }

    /**
     * Accessor for windowFrame
     * @return this.windowFrame as a JFrame
     * @precondition n/a
     * @postcondition n/a
     */
    public JFrame getWindowFrame(){
        return this.windowFrame;
    }
}
```

NewPostCreatePanel:

```
package gui;
import java.awt.*;
import java.awt.event.*;
```

```
import java.util.ArrayList;
import java.util.Arrays;
import javax.swing.*;
import javax.swing.border.EmptyBorder;

import core.NetworkSystem;
import core.Post;
import core.UserPost;

/**
 * NewPostCreatePanel is a JPanel for creating a new post
 * Allowing users to input content, select a category, and save or cancel the operation.
 * Includes interactive features such as placeholder text, category selection, and focus behavior.
 * @author Hieu
 */
public class NewPostCreatePanel extends JPanel {
    private JTextArea content;
    private final JButton save;
    private final JButton cancel;
    private JMenu categoryMenu;
    private final JMenuBar menuBar;
    private final JMenuItem[] menuItem;
    private final String ENTER_COMMAND = "Enter your opinion here";
    ArrayList<String> categoryName = new ArrayList<>(Arrays.asList("Sports", "Technology", "Art",
"News"));

    /**
     * Constructs a NewPostCreatePanel with text input, category selection, and save/cancel buttons.
     * @precondition: Requires the action on the JTextArea and the JButton
     * @postcondition: If "Save" is clicked, a new post is added to the system's network and the profile feed
is refreshed.
     */
    public NewPostCreatePanel() {
        this.setLayout(new BorderLayout());

        JLabel label = new JLabel("Write a new post");
        this.add(label, BorderLayout.NORTH);

        //set text area with placeholder text
        content = new JTextArea(ENTER_COMMAND);
        content.setForeground(Color.LIGHT_GRAY);
        content.setBorder(new EmptyBorder(10, 10, 10, 10));
        content.setLineWrap(true);
        content.setWrapStyleWord(true);
```

```
        content.setRows(1);

        content.addFocusListener(new FocusListener() {
            //Clears placeholder text when the text area gains focus.
            @Override
            public void focusGained(FocusEvent e) {
                if (content.getText().equals(ENTER_COMMAND)) {
                    content.setText("");
                    content.setForeground(Color.BLACK);
                }
            }

            //Restores placeholder text if the text area loses focus with no input.
            @Override
            public void focusLost(FocusEvent e) {
                if (content.getText().equals("")) {
                    content.setText(ENTER_COMMAND);
                    content.setForeground(Color.LIGHT_GRAY);
                }
            }
        });
        this.add(content, BorderLayout.CENTER);

        //set the action panel, which contains the buttons
        JPanel actionPanel = new JPanel();

        //save button to create and save a new post
        save = new JButton("Save");
        save.addActionListener(new ActionListener() {
            //saves the post content and category into the network system.
            @Override
            public void actionPerformed(ActionEvent e) {
                if (!content.getText().equals("") && !content.getText().equals(ENTER_COMMAND)) {
                    String contentString = content.getText();
                    Post newPost = new UserPost(contentString,
NetworkSystem.getInstance().getCurrentUser());
                    if (!categoryMenu.getText().equals("Category")) {
                        newPost.setCategory(categoryMenu.getText());
                    }
                    NetworkSystem.getInstance().addPosts(newPost);

NetworkSystem.getInstance().getNetworkWindow().getProfilePagePanel().refreshProfileFeed();
                }
            }
```

```
        });

        //cancel button to clear the input field
        cancel = new JButton("Cancel");
        cancel.addActionListener(new ActionListener() {
            //Clears the text area input.
            @Override
            public void actionPerformed(ActionEvent e) {
                content.setText("");
            }
        });

        //initialize category menu
        menuItem = new JMenuItem[categoryName.size()];
        menuBar = new JMenuBar();
        categoryMenu = new JMenu("Category");

        for (int i = 0; i < categoryName.size(); i++) {
            menuItem[i] = new JMenuItem(categoryName.get(i));
            final String name = categoryName.get(i);
            menuItem[i].addActionListener(e -> categoryMenu.setText(name));
            categoryMenu.add(menuItem[i]);
        }
        menuBar.add(categoryMenu);

        // Set up action panel
        actionPanel.setLayout(new FlowLayout(FlowLayout.RIGHT));
        actionPanel.add(menuBar);
        actionPanel.add(save);
        actionPanel.add(cancel);

        this.setBorder(BorderFactory.createLineBorder(Color.WHITE, 15));
        this.add(actionPanel, BorderLayout.SOUTH);
    }
}
```

ProfileFeedPanel:

```
package gui;

import java.awt.BorderLayout;

import javax.swing.JPanel;
```

```
import utility.ProfilePageSortStrategy;

/**
 * ProfileFeedPanel is a JPanel that displays the user's profile feed, including a post creation section
 * at the top and a feed panel that displays posts sorted based on a profile page strategy.
 * @author Tabbie, Hieu
 */
public class ProfileFeedPanel extends JPanel {
    private NewPostCreatePanel createPostPanel;
    private FeedPanel feedPanel;

    /**
     * Constructs a ProfileFeedPanel that initializes the components.
     * The panel contains a NewPostCreatePanel at the top and a FeedPanel in the center, sorted using
ProfilePageSortStrategy.
     * The layout is set to BorderLayout, and the components are added accordingly.
     * A ProfileFeedPanel is created with a NewPostCreatePanel at the top and a FeedPanel in the center
     */
    public ProfileFeedPanel() {
        this.createPostPanel = new NewPostCreatePanel();
        this.feedPanel = new FeedPanel(new ProfilePageSortStrategy());
        this.setLayout(new BorderLayout());
        this.add(this.createPostPanel, BorderLayout.NORTH);
        this.add(this.feedPanel, BorderLayout.CENTER);
    }

    /**
     * Refreshes the feed panel by repainting it.
     * Used to update the feed display after a new post has been created or the feed has changed.
     * @precondition A ProfileFeedPanel has already been initialized with a FeedPanel.
     * @postcondition The feed panel is repainted to reflect any changes.
     * @return The current instance of ProfileFeedPanel.
     */
    public ProfileFeedPanel createProfileFeedPanel() {
        this.feedPanel.repaint();
        return this;
    }
}
```

ProfilePagePanel:

```
package gui;
import java.awt.BorderLayout;
import java.awt.Color;
```

```
import java.awt.Dimension;

import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.SwingUtilities;

import controller.LogoutController;
import controller.ProfileController;
import core.NetworkSystem;
import core.UserAccount;

/**
 * ProfilePagePanel represents the panel in profile page.
 * It includes a profile panel showing user details (west), a feed panel displaying posts(center),
 * and a logout panel(east) with functionality to log out of the system.
 * @author Tabbie, Hieu
 */
public class ProfilePagePanel extends JPanel {

    private ProfilePanel profilePanel;
    private ProfileFeedPanel profileFeedPanel;
    private final LogoutPanel logoutPanel;
    private JScrollPane scrollPane;

    /**
     * Constructs a ProfilePagePanel.
     * Set up the profile panel, profile feed panel, and logout panel, and their controller
     * @precondition The current user is retrieved from the NetworkSystem.
     * @postcondition A ProfilePagePanel is created with all necessary components,
     *                including the profile, feed, and logout panels,
     *                and their respective controllers.
     */
    public ProfilePagePanel() {
        UserAccount curUser = NetworkSystem.getInstance().getCurrentUser();
        this.profilePanel = new ProfilePanel(curUser);
        this.profileFeedPanel = new ProfileFeedPanel();
        this.logoutPanel = new LogoutPanel();

        //set the width and color for the west and east panel
        int panelWidth = NetworkSystem.getInstance().getWidth() / 5;
        this.profilePanel.setPreferredSize(new Dimension(panelWidth, this.getHeight()));
        this.logoutPanel.setPreferredSize(new Dimension(panelWidth, this.getHeight()));

        this.profileFeedPanel.setBackground(new Color(219, 231, 252));
```

```
        this.logoutPanel.setBackground(new Color(219, 231, 252));

        LogoutController logoutController = new LogoutController(this.logoutPanel,
NetworkSystem.getInstance());
        ProfileController profileController = new ProfileController(profilePanel, curUser);

        this.setLayout(new BorderLayout());

        this.add(this.profilePanel, BorderLayout.WEST);
        this.add(this.logoutPanel, BorderLayout.EAST);
        setScrollPane();
        this.add(scrollPane, BorderLayout.CENTER);
    }

    /**
     * Wraps the profile feed panel in a scroll pane to make the feed scrollable.
     * The scroll pane's vertical scroll bar is set to always be visible.
     * @precondition The profile feed panel is initialized.
     * @postcondition A JScrollPane containing the profile feed panel is set,
     *                and the vertical scroll bar is made visible.
     */
    public final void setScrollPane() {
        scrollPane = new JScrollPane(this.profileFeedPanel);
        scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
        scrollPane.getVerticalScrollBar().setValue(0);
        SwingUtilities.invokeLater(() -> scrollPane.getVerticalScrollBar().setValue(0));
    }

    /**
     * Returns the profile panel associated with this profile page.
     * @return The ProfilePanel instance.
     * @precondition The ProfilePagePanel must have been initialized.
     * @postcondition The current ProfilePanel is returned.
     */
    public ProfilePanel getProfilePanel() {
        return this.profilePanel;
    }

    /**
     * Refreshes the profile feed by resetting the profile feed panel and the scroll pane.
     * This is typically used to update the feed after a change has occurred, such as a new post.
     * @precondition The ProfileFeedPanel is updated and requires a refresh.
     * @postcondition The profile feed and scroll pane are reset, and the layout is refreshed.
     */
```

```
    public void refreshProfileFeed() {
        // Set profileFeedPanel to a new object (after updating FeedPanel)
        profileFeedPanel = new ProfileFeedPanel();
        // Remove the current scroll pane and reset it
        remove(scrollPane);
        setScrollPane();
        add(scrollPane, BorderLayout.CENTER);

        // Refresh the layout
        revalidate();
        repaint();
    }

    public void refreshProfilePoint(){
        profilePanel.updatePointLabel();
    }
}
```

ProfilePanel:

```
package gui;
/**
 * @author Tabbie Brantley
 * The ProfileView class is the view for the Profile
 * @class invariant the view will always show the current UserAccount name
 */

//imports
import java.awt.BorderLayout;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

import core.UserAccount;


public class ProfilePanel extends JPanel{
```

```
//attributes for main profilePanel
private JPanel userInfoPanel;
private JLabel currentNameLabel;
private JLabel currentPointsLabel;
private UserAccount user;
private JButton editNameButton;
private JButton editPasswordButton;

//attributes for editNameFrame
private JFrame editNameFrame;
private JPanel editNamePanel;
private JButton saveNameButton;
private JLabel enterNameLabel;
private JTextField nameField;
private JButton cancelNameEditButton;

//attributes for editPassword frame
private JFrame editPasswordFrame;
private JPanel editPasswordPanel;
private JButton savePasswordButton;
private JLabel enterPasswordLabel;
private JLabel confirmPasswordLabel;
private JTextField passwordField;
private JTextField reentryPasswordField;
private JButton cancelPasswordEditButton;


private GridBagConstraints c;

/**
 * Constructor for profile panel
 * @param curUser the current user as a UserAccount
 * @precondtion n/a
 * @postcondition view is shown
 */
public ProfilePanel(UserAccount curUser){
    this.user = curUser;

    //setting c info
    this.c = new GridBagConstraints();
    this.c.fill = GridBagConstraints.HORIZONTAL;
    this.c.insets = new Insets(5, 5, 5,  5);
```

```
//initializing the profilePanel
this.setLayout(new BorderLayout());

//initializing the userInfoPanel
this.userInfoPanel = new JPanel();
this.userInfoPanel.setLayout(new GridBagLayout());
this.currentNameLabel = new JLabel("Account Name: " + user.getAccountName());
this.currentPointsLabel = new JLabel("Total Points: " + Integer.toString(user.getPoints()));

//adding labels and buttons
this.c.gridx = 0;
this.c.gridy = 0;
this.userInfoPanel.add(this.currentNameLabel, this.c);

this.c.gridx = 0;
this.c.gridy = 1;
this.userInfoPanel.add(this.currentPointsLabel, this.c);

this.editNameButton = new JButton("Edit name");
this.editPasswordButton = new JButton("Edit password");

this.c.gridx = 0;
this.c.gridy = 2;
this.userInfoPanel.add(this.editNameButton, this.c);

this.c.gridx = 0;
this.c.gridy = 3;
this.userInfoPanel.add(this.editPasswordButton, this.c);

this.add(userInfoPanel, BorderLayout.WEST);

//setting up the editNameFrame
this.editNameFrame = new JFrame();
this.editNamePanel = new JPanel();
this.editNamePanel.setLayout(new GridBagLayout());

this.enterNameLabel = new JLabel("Enter new name:");
this.nameField = new JTextField();
this.saveNameButton = new JButton("Save Name");
this.cancelNameEditButton = new JButton("Cancel");

this.c.gridx = 0;
this.c.gridy = 0;
this.editNamePanel.add(this.enterNameLabel, this.c);
```

```
this.c.gridx = 1;
this.c.gridy = 0;
this.editNamePanel.add(this.nameField, this.c);

this.c.gridx = 0;
this.c.gridy = 1;
this.editNamePanel.add(this.saveNameButton, this.c);

this.c.gridx = 1;
this.c.gridy = 1;
this.editNamePanel.add(this.cancelNameEditButton, this.c);

this.editNameFrame.add(this.editNamePanel);
this.editNameFrame.pack();

//setting up the editPasswordFrame
this.editPasswordFrame = new JFrame();
this.editPasswordPanel = new JPanel();
this.editPasswordPanel.setLayout(new GridBagLayout());

this.enterPasswordLabel = new JLabel("Enter New Password:");
this.confirmPasswordLabel = new JLabel("Confirm Password:");
this.savePasswordButton = new JButton("Save Password");
this.passwordField = new JTextField();
this.reentryPasswordField = new JTextField();
this.cancelPasswordEditButton = new JButton("Cancel");


this.c.gridx = 0;
this.c.gridy = 0;
this.editPasswordPanel.add(this.enterPasswordLabel, this.c);

this.c.gridx = 1;
this.c.gridy = 0;
this.editPasswordPanel.add(this.passwordField, this.c);

this.c.gridx = 0;
this.c.gridy = 1;
this.editPasswordPanel.add(this.confirmPasswordLabel, this.c);

this.c.gridx = 1;
this.c.gridy = 1;
this.editPasswordPanel.add(this.reentryPasswordField, this.c);
```

```
    this.c.gridx = 0;
    this.c.gridy = 2;
    this.editPasswordPanel.add(this.savePasswordButton, this.c);

    this.c.gridx = 1;
    this.c.gridy = 2;
    this.editPasswordPanel.add(this.cancelPasswordEditButton, this.c);

    this.editPasswordFrame.add(this.editPasswordPanel);
    this.editPasswordFrame.pack();
}

/**
 * getProfilePanel accessor for the profile panel
 * @return the profilePanels reference
 * @precondition n/a
 * @postcondition n/a
 */
public JPanel getProfilePanel(){
    return this;
}

/**
 * getEditNameButton accessor for the editNameButton
 * @return the editNameButton reference
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getEditNameButton(){
    return this.editNameButton;
}

 /**
 * getCancelNameButton accessor for the cancelNameButton
 * @return the cancelNameButton reference
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getCancelNameButton(){
    return this.cancelNameEditButton;
}

 /**
```

```
 * getSaveNameButton accessor for the saveNameButton
 * @return the saveNameButton reference
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getSaveNameButton(){
    return this.saveNameButton;
}

/**
 * getNewName accessor for the nameField
 * @return the nameField reference
 * @precondition n/a
 * @postcondition n/a
 */
public JTextField getNewName(){
    return this.nameField;
}

/**
 * getEditPasswordButton accessor for the editPasswordButton
 * @return the editPasswordButton reference
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getEditPasswordButton(){
    return this.editPasswordButton;
}

/**
 * getCancelPasswordButton accessor for the cancelPasswordButton
 * @return the cancelPasswordButton reference
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getCancelPasswordEditButton(){
    return this.cancelPasswordEditButton;
}

/**
 * getSavePasswordButton accessor for the savePasswordButton
 * @return the savePasswordButton reference
 * @precondition n/a
 * @postcondition n/a
```

```
 */
public JButton getSavePasswordButton(){
    return this.savePasswordButton;
}


/**
 * getNewPassword accessor for the passwordField
 * @return the passwordField reference
 * @precondition n/a
 * @postcondition n/a
 */
public JTextField getNewPassword(){
    return this.passwordField;
}

/**
 * getConfirmedPassword accessor for the reentryPasswordField
 * @return the reentryPasswordField reference
 * @precondition n/a
 * @postcondition n/a
 */
public JTextField getConfirmedPassword(){
    return this.reentryPasswordField;
}

/**
 * editName to display the edit name frame
 * @precodition n/a
 * @postcodntion editNameFrame is set to be visible
 */
public void editName(){
    this.editNameFrame.setVisible(true);
}

/**
 * cancelName to stop displaying the editNameFrame
 * @precodition n/a
 * @postcodntion editNameFrame is set to not be visible
 */
public void cancelNameEdit(){
    this.editNameFrame.setVisible(false);
}
```

```
/**
 * displayNameLengthErrorMessage displays a JOptionPane with the
 * corresponding error
 * @procondion n/a
 * @postcondition n/a
 */
public void displayNameLengthErrorMessage(){
    JOptionPane.showMessageDialog(null, "Error: Name must be between 8-20 characters.");
}

/**
 * closeEditName closes the editName frame
 * @precodition n/a
 * @postcodntion editNameFrame is set to not be visible
 */
public void closeEditName(){
    this.editNameFrame.setVisible(false);
}

/**
 * editPassowrd to display the edit name frame
 * @precodition n/a
 * @postcodntion editPasswordFrame is set to be visible
 */
public void editPassword(){
    this.editPasswordFrame.setVisible(true);
}

/**
 * cancelPasswordEdit to stop displaying the editPasswordFrame
 * @precodition n/a
 * @postcodntion editNameFrame is set to not be visible
 */
public void cancelPasswordEdit(){
    this.editPasswordFrame.setVisible(false);
}


/**
 * displayPasswordLengthErrorMessage displays a JOptionPane with the
 * corresponding error
 * @procondion n/a
 * @postcondition n/a
 */
```

```
public void displayPasswordLengthErrorMessage(){
    JOptionPane.showMessageDialog(null, "Error: Password must be between 8-20 characters.");
}

/**
 * displayPasswordMatchErrorMessage displays a JOptionPane with the
 * corresponding error
 * @procondion n/a
 * @postcondition n/a
 */
public void displayPasswordMatchErrorMessage(){
    JOptionPane.showMessageDialog(null, "Error: Passwords do not match.");
}

 /**
 * closeEditPassword closes the editPassword frame
 * @precodition n/a
 * @postcodntion editNameFrame is set to not be visible
 */
public void closeEditPassword(){
    this.editPasswordFrame.setVisible(false);
}

/**
 * updateName sets the currentNameLabel text to a new name
 * @param nm is the new name as a String
 * @precondtion n/a
 * @postcondition the current instance of ProfilePanel is repainted to display
 * the new name
 */
public void updateName(String nm){
    this.currentNameLabel.setText("Account Name: " + nm);
    this.repaint();
}

/**
 * Update the point label
 * @precondtion n/a
 * @postcondition the current instance (currentPointsLabel) of ProfilePanel is repainted to display
 */
public void updatePointLabel(){
    this.currentPointsLabel.setText("Total Points: " + Integer.toString(user.getPoints()));
    this.repaint();
    System.out.println("Update point");
```

```
    }
}
```

GeneralPageSortStrategy:

```
package gui;
/**
 * @author Tabbie Brantley
 * The ProfileView class is the view for the Profile
 * @class invariant the view will always show the current UserAccount name
 */

//imports
import java.awt.BorderLayout;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

import core.UserAccount;


public class ProfilePanel extends JPanel{

    //attributes for main profilePanel
    private JPanel userInfoPanel;
    private JLabel currentNameLabel;
    private JLabel currentPointsLabel;
    private UserAccount user;
    private JButton editNameButton;
    private JButton editPasswordButton;

    //attributes for editNameFrame
    private JFrame editNameFrame;
    private JPanel editNamePanel;
    private JButton saveNameButton;
    private JLabel enterNameLabel;
```

```
private JTextField nameField;
private JButton cancelNameEditButton;

//attributes for editPassword frame
private JFrame editPasswordFrame;
private JPanel editPasswordPanel;
private JButton savePasswordButton;
private JLabel enterPasswordLabel;
private JLabel confirmPasswordLabel;
private JTextField passwordField;
private JTextField reentryPasswordField;
private JButton cancelPasswordEditButton;


private GridBagConstraints c;

/**
 * Constructor for profile panel
 * @param curUser the current user as a UserAccount
 * @precondtion n/a
 * @postcondition view is shown
 */
public ProfilePanel(UserAccount curUser){
    this.user = curUser;

    //setting c info
    this.c = new GridBagConstraints();
    this.c.fill = GridBagConstraints.HORIZONTAL;
    this.c.insets = new Insets(5, 5, 5,  5);

    //initializing the profilePanel
    this.setLayout(new BorderLayout());

    //initializing the userInfoPanel
    this.userInfoPanel = new JPanel();
    this.userInfoPanel.setLayout(new GridBagLayout());
    this.currentNameLabel = new JLabel("Account Name: " + user.getAccountName());
    this.currentPointsLabel = new JLabel("Total Points: " + Integer.toString(user.getPoints()));

    //adding labels and buttons
    this.c.gridx = 0;
    this.c.gridy = 0;
    this.userInfoPanel.add(this.currentNameLabel, this.c);
```

```
this.c.gridx = 0;
this.c.gridy = 1;
this.userInfoPanel.add(this.currentPointsLabel, this.c);

this.editNameButton = new JButton("Edit name");
this.editPasswordButton = new JButton("Edit password");

this.c.gridx = 0;
this.c.gridy = 2;
this.userInfoPanel.add(this.editNameButton, this.c);

this.c.gridx = 0;
this.c.gridy = 3;
this.userInfoPanel.add(this.editPasswordButton, this.c);

this.add(userInfoPanel, BorderLayout.WEST);

//setting up the editNameFrame
this.editNameFrame = new JFrame();
this.editNamePanel = new JPanel();
this.editNamePanel.setLayout(new GridBagLayout());

this.enterNameLabel = new JLabel("Enter new name:");
this.nameField = new JTextField();
this.saveNameButton = new JButton("Save Name");
this.cancelNameEditButton = new JButton("Cancel");

this.c.gridx = 0;
this.c.gridy = 0;
this.editNamePanel.add(this.enterNameLabel, this.c);

this.c.gridx = 1;
this.c.gridy = 0;
this.editNamePanel.add(this.nameField, this.c);

this.c.gridx = 0;
this.c.gridy = 1;
this.editNamePanel.add(this.saveNameButton, this.c);

this.c.gridx = 1;
this.c.gridy = 1;
this.editNamePanel.add(this.cancelNameEditButton, this.c);

this.editNameFrame.add(this.editNamePanel);
```

```
    this.editNameFrame.pack();

    //setting up the editPasswordFrame
    this.editPasswordFrame = new JFrame();
    this.editPasswordPanel = new JPanel();
    this.editPasswordPanel.setLayout(new GridBagLayout());

    this.enterPasswordLabel = new JLabel("Enter New Password:");
    this.confirmPasswordLabel = new JLabel("Confirm Password:");
    this.savePasswordButton = new JButton("Save Password");
    this.passwordField = new JTextField();
    this.reentryPasswordField = new JTextField();
    this.cancelPasswordEditButton = new JButton("Cancel");


    this.c.gridx = 0;
    this.c.gridy = 0;
    this.editPasswordPanel.add(this.enterPasswordLabel, this.c);

    this.c.gridx = 1;
    this.c.gridy = 0;
    this.editPasswordPanel.add(this.passwordField, this.c);

    this.c.gridx = 0;
    this.c.gridy = 1;
    this.editPasswordPanel.add(this.confirmPasswordLabel, this.c);

    this.c.gridx = 1;
    this.c.gridy = 1;
    this.editPasswordPanel.add(this.reentryPasswordField, this.c);

    this.c.gridx = 0;
    this.c.gridy = 2;
    this.editPasswordPanel.add(this.savePasswordButton, this.c);

    this.c.gridx = 1;
    this.c.gridy = 2;
    this.editPasswordPanel.add(this.cancelPasswordEditButton, this.c);

    this.editPasswordFrame.add(this.editPasswordPanel);
    this.editPasswordFrame.pack();
}

/**
```

```
 * getProfilePanel accessor for the profile panel
 * @return the profilePanels reference
 * @precondition n/a
 * @postcondition n/a
 */
public JPanel getProfilePanel(){
    return this;
}

/**
 * getEditNameButton accessor for the editNameButton
 * @return the editNameButton reference
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getEditNameButton(){
    return this.editNameButton;
}

 /**
 * getCancelNameButton accessor for the cancelNameButton
 * @return the cancelNameButton reference
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getCancelNameButton(){
    return this.cancelNameEditButton;
}

 /**
 * getSaveNameButton accessor for the saveNameButton
 * @return the saveNameButton reference
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getSaveNameButton(){
    return this.saveNameButton;
}

/**
 * getNewName accessor for the nameField
 * @return the nameField reference
 * @precondition n/a
 * @postcondition n/a
```

```
 */
public JTextField getNewName(){
    return this.nameField;
}

/**
 * getEditPasswordButton accessor for the editPasswordButton
 * @return the editPasswordButton reference
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getEditPasswordButton(){
    return this.editPasswordButton;
}

/**
 * getCancelPasswordButton accessor for the cancelPasswordButton
 * @return the cancelPasswordButton reference
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getCancelPasswordEditButton(){
    return this.cancelPasswordEditButton;
}

/**
 * getSavePasswordButton accessor for the savePasswordButton
 * @return the savePasswordButton reference
 * @precondition n/a
 * @postcondition n/a
 */
public JButton getSavePasswordButton(){
    return this.savePasswordButton;
}


/**
 * getNewPassword accessor for the passwordField
 * @return the passwordField reference
 * @precondition n/a
 * @postcondition n/a
 */
public JTextField getNewPassword(){
    return this.passwordField;
```

```
}

/**
 * getConfirmedPassword accessor for the reentryPasswordField
 * @return the reentryPasswordField reference
 * @precondition n/a
 * @postcondition n/a
 */
public JTextField getConfirmedPassword(){
    return this.reentryPasswordField;
}

/**
 * editName to display the edit name frame
 * @precodition n/a
 * @postcodntion editNameFrame is set to be visible
 */
public void editName(){
    this.editNameFrame.setVisible(true);
}

/**
 * cancelName to stop displaying the editNameFrame
 * @precodition n/a
 * @postcodntion editNameFrame is set to not be visible
 */
public void cancelNameEdit(){
    this.editNameFrame.setVisible(false);
}

/**
 * displayNameLengthErrorMessage displays a JOptionPane with the
 * corresponding error
 * @procondion n/a
 * @postcondition n/a
 */
public void displayNameLengthErrorMessage(){
    JOptionPane.showMessageDialog(null, "Error: Name must be between 8-20 characters.");
}

/**
 * closeEditName closes the editName frame
 * @precodition n/a
 * @postcodntion editNameFrame is set to not be visible
```

```
 */
public void closeEditName(){
    this.editNameFrame.setVisible(false);
}

/**
 * editPassowrd to display the edit name frame
 * @precodition n/a
 * @postcodntion editPasswordFrame is set to be visible
 */
public void editPassword(){
    this.editPasswordFrame.setVisible(true);
}

/**
 * cancelPasswordEdit to stop displaying the editPasswordFrame
 * @precodition n/a
 * @postcodntion editNameFrame is set to not be visible
 */
public void cancelPasswordEdit(){
    this.editPasswordFrame.setVisible(false);
}


/**
 * displayPasswordLengthErrorMessage displays a JOptionPane with the
 * corresponding error
 * @procondion n/a
 * @postcondition n/a
 */
public void displayPasswordLengthErrorMessage(){
    JOptionPane.showMessageDialog(null, "Error: Password must be between 8-20 characters.");
}

/**
 * displayPasswordMatchErrorMessage displays a JOptionPane with the
 * corresponding error
 * @procondion n/a
 * @postcondition n/a
 */
public void displayPasswordMatchErrorMessage(){
    JOptionPane.showMessageDialog(null, "Error: Passwords do not match.");
}
```

```
    /**
     * closeEditPassword closes the editPassword frame
     * @precodition n/a
     * @postcodntion editNameFrame is set to not be visible
     */
    public void closeEditPassword(){
        this.editPasswordFrame.setVisible(false);
    }

    /**
     * updateName sets the currentNameLabel text to a new name
     * @param nm is the new name as a String
     * @precondtion n/a
     * @postcondition the current instance of ProfilePanel is repainted to display
     * the new name
     */
    public void updateName(String nm){
        this.currentNameLabel.setText("Account Name: " + nm);
        this.repaint();
    }

    /**
     * Update the point label
     * @precondtion n/a
     * @postcondition the current instance (currentPointsLabel) of ProfilePanel is repainted to display
     */
    public void updatePointLabel(){
        this.currentPointsLabel.setText("Total Points: " + Integer.toString(user.getPoints()));
        this.repaint();
        System.out.println("Update point");
    }
}
```

ProfilePageSortStrategy:

```
package utility;
import java.util.ArrayList;

import core.NetworkSystem;
import core.Post;

/**
 * ProfilePageSortStrategy implements SortPostStrategy interface.
 * It is used to sort and filter posts based on the current user's profile.
 * This strategy is typically used to display posts on the user's profile page.
```

```
 * @author Hieu Truong
 */
public class ProfilePageSortStrategy implements SortPostStrategy {

    /**
     * Sorts the posts to only include those created by the current user.
     * The posts are returned in reverse order, with the most recent posts appearing first.
     * @return An ArrayList of Post objects created by the current user, in reverse chronological order.
     * @precondition The NetworkSystem must be initialized and contain a list of posts.
     *               The current user must be set in the NetworkSystem.
     * @postcondition list of posts in the profile page made by the current user is returned.
     */
    @Override
    public ArrayList<Post> sortPost() {
        //get the list of all posts in the system
        ArrayList<Post> allPosts = NetworkSystem.getInstance().getPostsList();
        ArrayList<Post> requirePost = new ArrayList<>();

        //go through the posts in reverse order (most recent first)
        for (int i = allPosts.size()-1; i>= 0; i--) {

            if
(allPosts.get(i).getOwner().getUsername().equals(NetworkSystem.getInstance().getCurrentUser().getUse
rname())){
                requirePost.add(allPosts.get(i));
            }
        }

        //check
        System.out.println("REQUIRE:" + requirePost);
        return requirePost;
    }
}
```

SortPostStrategy:

```
package utility;
import java.util.ArrayList;

import core.Post;

/**
 * The SortPostStrategy interface defines a strategy for sorting posts.
 * This interface is used in the context of a post feed, where posts need to be sorted
 * @author Hieu Truong
 */
```

```
public interface SortPostStrategy {

    /**
     * Sorts the posts based on a specific strategy.
     * The implementing class should define how the posts are sorted and filtered.
     * @return An ArrayList of sorted and filtered Post objects.
     * @precondition The system should have a list of posts available for sorting.
     * @postcondition A sorted and filtered list of posts is returned according to the specific strategy.
     */
    public ArrayList<Post> sortPost();
}
```

Test files:

NetworkSystemTest:

```
package core;



import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;

import java.awt.Dimension;
import java.awt.Toolkit;
import java.util.Arrays;
import java.util.List;

import utility.ProfilePageSortStrategy;

/**
 * Test class for NetworkSystem
 * Directly tests handleAgreeAction, handleDisagreeAction, handleRepostAction, and handleDeleteAction
methods.
 * @author Tabbie Brantley and Selin Topac
 */
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
```

```
import gui.*;
import core.*;


public class NetworkSystemTest {

    public NetworkSystemTest() {
    }

    @BeforeAll
    public static void setUpClass() {
    }

    @AfterAll
    public static void tearDownClass() {
    }

    @BeforeEach
    public void setUp() {
    }

    @AfterEach
    public void tearDown() {
    }


    /**
     * Test of setCurrentUser method, of class NetworkSystem.
     */
    @Test
    public void testSetCurrentUser() {
        System.out.println("setCurrentUser");
        System.out.println("getCurrentUser");
        UserAccount user = new UserAccount("accountName", "username", "password");
        NetworkSystem instance = NetworkSystem.getInstance();
        instance.getUserAccountsList().put("username", user);
        instance.setCurrentUser(user);
        UserAccount expResult = user;
        UserAccount result = instance.getCurrentUser();
        assertEquals(expResult, result);
    }


    /**
```

```
 * @author Tabbie Brantley
 * Test of getCurrentUser method, of class NetworkSystem.
 */
@Test
public void testGetCurrentUser() {
    System.out.println("getCurrentUser");
    UserAccount user = new UserAccount("accountName", "username", "password");
    NetworkSystem instance = NetworkSystem.getInstance();
    instance.getUserAccountsList().put("username", user);
    instance.setCurrentUser(user);
    UserAccount expResult = user;
    UserAccount result = instance.getCurrentUser();
    assertEquals(expResult, result);
}

/**
 * @author Tabbie Brantley
 * Test of checkUserExists method, of class NetworkSystem.
 */
@Test
public void testCheckUserExists() {
    System.out.println("checkUserExists");
    NetworkSystem instance = NetworkSystem.getInstance();
    String un = "username";
    String pwd = "password";
    String accountName = "accountName";
    //when it should exist
    instance.addUserAccount(accountName, un, pwd);
    boolean expResult = true;
    boolean result = instance.checkUserExists(un);
    assertEquals(expResult, result);

    //when it shouldn't exist
    boolean expResult2 = false;
    boolean result2 = instance.checkUserExists("incorrect");
    assertEquals(expResult, result);
}



/**
 * @author Tabbie Brantley
 * Test of getHeight method, of class NetworkSystem.
 */
```

```
@Test
public void testGetHeight() {
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    System.out.println("getHeight");
    int expResult = screenSize.height;
    int result = NetworkSystem.getInstance().getHeight();
    assertEquals(expResult, result);

}

/**
 * @author Tabbie Brantley
 * Test of getWidth method, of class NetworkSystem.
 */
@Test
public void testGetWidth() {
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    System.out.println("getWidth");
    int expResult = screenSize.width;
    int result = NetworkSystem.getInstance().getWidth();
    assertEquals(expResult, result);
}



/**
 * @author Tabbie Brantley
 * Test of usernameExists method, of class NetworkSystem.
 */
@Test
public void testUsernameExists() {
    NetworkSystem instance = NetworkSystem.getInstance();
    String un = "username";
    String pwd = "password";
    String accountName = "accountName";
    //when it should exist
    instance.addUserAccount(accountName, un, pwd);
    boolean expResult = true;
    boolean result = instance.usernameExists(un);
    assertEquals(expResult, result);

    //when it shouldn't exist
    boolean expResult2 = false;
    boolean result2 = instance.usernameExists("incorrect");
```

```
    assertEquals(expResult, result);
}

/**
* @author Tabbie Brantley
* Test of loginUser method, of class NetworkSystem. Making sure the correct user was logged in
*/
@Test
public void testLoginUser() {
    System.out.println("loginUser");
    NetworkSystem instance = NetworkSystem.getInstance();
    String un = "username";
    String pwd = "password";
    String accountName = "accountName";
    //when it should log in
    instance.addUserAccount(accountName, un, pwd);
    boolean expResult = true;
    boolean result = instance.loginUser(un, pwd);
    assertEquals(expResult, result);

    //when it shouldn't log in
    boolean expResult2 = false;
    boolean result2 = instance.loginUser(un, "incorrect");
    assertEquals(expResult, result);
}


 /**
 * @author TabbieBrantley
 * Test of logout method, of class NetworkSystem. Testing that the current user is set to null
 */
@Test
public void testLogout() {
    System.out.println("logout");
    NetworkSystem instance = NetworkSystem.getInstance();
    instance.setCurrentUser(new UserAccount("accountname", "username", "password"));
    instance.logout();
    assertEquals(null, instance.getCurrentUser());
}

/**

/**
 * @author Tabbie Brantley
```

```
     *
     * test for updating points
     */
    @Test
    public void testUpdatePointForUser(){
        System.out.println("updatePointForUser");
        NetworkSystem.getInstance().addUserAccount("name", "usernametest", "password");
        int val = 1;
        NetworkSystem instance = NetworkSystem.getInstance();
        instance.updatePointForUser("usernametest", val);
        assertEquals(instance.getUserAccountsList().get("usernametest").getPoints(), val);
    }

    /**
     * @author Selin Topac
     *
     * test for addinging a user account
     */
    @Test
    public void testAddUserAccount() {

        NetworkSystem.getInstance().addUserAccount("accounttest", "testname", "password");

        assertEquals(1, NetworkSystem.getInstance().getUserAccountsList().size());

        assertNotNull(NetworkSystem.getInstance().getUserAccountsList().get("testname"));

    }

}
```

PostTest:

```
package core;

import javax.swing.JButton;
import javax.swing.JPanel;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import utility.*;
```

```
/**
 * PostTest tests for class Post
 * @author Hieu Truong
 */
public class PostTest {

    public PostTest() {
    }

    @BeforeAll
    public static void setUpClass() {
    }

    @AfterAll
    public static void tearDownClass() {
    }

    @BeforeEach
    public void setUp() {
    }

    @AfterEach
    public void tearDown() {
    }

    /**
     * Test of setID method, of class Post.
     */
    @Test
    public void testSetID() {
        System.out.println("setID");
        int id = 1;
        Post instance = new UserPost("Test Title", new UserAccount("testName", "testUsername",
"testpPass"));
        instance.setID(id);
        assertEquals(id, instance.getPostID(), "The ID should be correctly set.");
    }

    /**
     * Test of updatePoint method, of class Post.
     */
    @Test
    public void testUpdatePoint() {
```

```
        System.out.println("updatePoint");
        int points = 5;
        Post instance = new UserPost("Test Title", new UserAccount("testName", "testUsername",
"testpPass"));
        int initialPoints = instance.getPoints(); // Assume there's a method to get points.
        instance.updatePoint(points);
        assertEquals(initialPoints + points, instance.getPoints(), "The points should be updated correctly.");
    }

    /**
     * Test of setCategory method, of class Post.
     */
    @Test
    public void testSetCategory() {
        System.out.println("setCategory");
        String category = "Sports";
        Post instance = new UserPost("Test Title", new UserAccount("testName", "testUsername",
"testpPass"));
        instance.setCategory(category);
        // TODO review the generated test code and remove the default call to fail.
        assertEquals(category, instance.getCategory(), "The points should be updated correctly.");
    }

    /**
     * Test of getOwner method, of class Post.
     */
    @Test
    public void testGetOwner() {
        System.out.println("getOwner");
        UserAccount expectedOwner = new UserAccount("testName", "testUsername", "testPass");
        Post instance = new UserPost("Test Title", expectedOwner);
        UserAccount result = instance.getOwner();
        assertEquals(result.getUsername(), expectedOwner.getUsername());
    }

    /**
     * Test of getCategory method, of class Post.
     */
    @Test
    public void testGetCategory() {
        System.out.println("getCategory");
        Post instance = new UserPost("Test Title", new UserAccount("testName", "testUsername",
"testpPass"));
        instance.setCategory("Sport");
```

```
        String expResult = "Sport";
        String result = instance.getCategory();
        assertEquals(expResult, result);
    }

    /**
     * Test of getPostID method, of class Post.
     */
    @Test
    public void testGetPostID() {
        System.out.println("getPostID");
        Post instance = new UserPost("Test Title", new UserAccount("testName", "testUsername",
"testpPass"));
        instance.setID(1);
        int expResult = 1;
        int result = instance.getPostID();
        assertEquals(expResult, result);
    }

    /**
     * Test of addInteract method, of class Post.
     */
    @Test
    public void testAddInteract() {
        System.out.println("addInteract");
        String username = "user1";
        String interact = "Agree";
        Post instance = new UserPost("Test Title", new UserAccount("testName", "testUsername",
"testPass"));
        instance.addInteract(username, interact);
        String result = instance.getInteract(username);
        assertEquals(interact, result);
    }

    /**
     * Test of removeInteract method, of class Post.
     */
    @Test
    public void testRemoveInteract() {
        System.out.println("removeInteract");
        String username = "user1";
        Post instance = new UserPost("Test Title", new UserAccount("testName", "testUsername",
"testPass"));
```

```
        instance.addInteract(username, "Agree");
        instance.removeInteract(username);
        String result = instance.getInteract(username);

        assertEquals(result, "");
    }

    /**
     * Test of getInteract method, of class Post.
     */
    @Test
    public void testGetInteract() {
        System.out.println("getInteract");
        String username = "user1";
        String interact = "Agree";
        Post instance = new UserPost("Test Title", new UserAccount("testName", "testUsername",
"testPass"));
        instance.addInteract(username, interact);
        String expResult = "Agree";
        String result = instance.getInteract(username);
        assertEquals(expResult, result);
    }

    /**
     * Test of toString method, of class Post.
     */
    @Test
    public void testToString() {
        System.out.println("toString");
        Post instance = new UserPost("Test Title", new UserAccount("testName", "testUsername",
"testPass"));
        String result = instance.toString();
        String expResult = "Test Title testName";
        assertEquals(expResult, result);
    }

    /**
     * Test of getThisPostPanel method, of class Post.
     */
    @Test
    public void testGetThisPostPanel() {
        System.out.println("getThisPostPanel");
        Post instance = new UserPost("Test Title", new UserAccount("testName", "testUsername",
"testPass"));
```

```
        JPanel result = instance.getThisPostPanel();
        assertNotNull(result, "getThisPostPanel should return a non-null JPanel.");
    }

    /**
     * Test of getExtraDetail method, of class Post.
     */
    @Test
    public void testGetExtraDetail() {
        System.out.println("getExtraDetail");
        Post instance = new UserPost("Test Title", new UserAccount("testName", "testUsername",
"testPass"));
        Post instance2 = new Repost("Test Title", new UserAccount("testName", "testUsername",
"testPass"), instance);
        JPanel result = instance2.getExtraDetail();
        assertNotNull(result, "getExtraDetail should return a non-null JPanel with extra details.");
    }

    /**
     * Test of getCompletePostPanel method, of class Post.
     */
    @Test
    public void testGetCompletePostPanel() {
        System.out.println("getCompletePostPanel");
        SortPostStrategy displayStrategy = new GeneralPageSortStrategy(); // Replace with actual strategy
implementation
        Post instance = new UserPost("Test Title", new UserAccount("testName", "testUsername",
"testPass"));
        JPanel result = instance.getCompletePostPanel(displayStrategy);
        assertNotNull(result, "getCompletePostPanel should return a non-null JPanel using the strategy.");
    }

    /**
     * Test of getAgreeButton method, of class Post.
     */
    @Test
    public void testGetAgreeButton() {
        System.out.println("getAgreeButton");
        Post instance = new UserPost("Test Title", new UserAccount("testName", "testUsername",
"testPass"));
        instance.getCompletePostPanel(new GeneralPageSortStrategy());
        JButton result = instance.getAgreeButton();
        assertNotNull(result, "getAgreeButton should return a non-null JButton.");
        assertEquals("Agree", result.getText(), "Agree button should have the correct text.");
```

```
    }

    /**
     * Test of getDisagreeButton method, of class Post.
     */
    @Test
    public void testGetDisagreeButton() {
        System.out.println("getAgreeButton");
        Post instance = new UserPost("Test Title", new UserAccount("testName", "testUsername",
"testPass"));
        instance.getCompletePostPanel(new GeneralPageSortStrategy());
        JButton result = instance.getDisagreeButton();
        assertNotNull(result, "getDisagreeButton should return a non-null JButton.");
        assertEquals("Disagree", result.getText(), "Disagreebutton should have the correct text.");
    }

    public class PostImpl extends Post {

        public PostImpl() {
            super("", null);
        }

        public JPanel getExtraDetail() {
            return null;
        }
    }

}
```

UserAccountTest:

```
/**
 * @author Tabbie Brantley
 * UserAccountTest tests for UserAccount
 */

package core;

import static org.junit.jupiter.api.Assertions.*;


public class UserAccountTest {

    public UserAccountTest() {
```

```
}

@org.junit.jupiter.api.BeforeAll
public static void setUpClass() throws Exception {
}

@org.junit.jupiter.api.AfterAll
public static void tearDownClass() throws Exception {
}

@org.junit.jupiter.api.BeforeEach
public void setUp() throws Exception {
}

@org.junit.jupiter.api.AfterEach
public void tearDown() throws Exception {
}



/**
 * Test of getUsername method, of class UserAccount.
 */
@org.junit.jupiter.api.Test
public void testGetUsername() {
    System.out.println("getUsername");
    //testing getting the username
    UserAccount user = new UserAccount("AccountName", "Username", "Password");
    String expResult = "Username";
    String notExpResult = "Bad result";
    assertEquals(expResult, user.getUsername());
    assertNotEquals(notExpResult, user.getUsername());
}

/**
 * Test of getPassword method, of class UserAccount.
 */
@org.junit.jupiter.api.Test
public void testGetPassword() {
    //testing getting the password
    System.out.println("getPassword");
    UserAccount user = new UserAccount("AccountName", "Username", "Password");
    String expResult = "Password";
    String result = user.getPassword();
```

```
    assertEquals(expResult, result);
}

/**
 * Test of getPoints method, of class UserAccount.
 */
@org.junit.jupiter.api.Test
public void testGetPoints() {
    //testing getting the points
    System.out.println("getPoints");
    UserAccount user = new UserAccount("AccountName", "Username", "Password");
    int expResult = 0;
    int result = user.getPoints();
    assertEquals(expResult, result);
}

/**
 * Test of updatePoint method, of class UserAccount.
 */
@org.junit.jupiter.api.Test
public void testUpdatePoints() {
    //testing adding a positive number
    System.out.println("updatePoints");
    int number = 1;
    UserAccount user = new UserAccount("AccountName", "Username", "Password");
    user.updatePoints(number);
    int expResult = 1;
    assertEquals(expResult, user.getPoints());

    //testing when it is 1 and making it a 0
    int number2 = -1;
    user.updatePoints(number2);
    int expResult2 = 0;
    assertEquals(expResult2, user.getPoints());

    //testing when it is 1, and going down by -2, so points should be 0
    //since cannot have negative points
    int number3 = 1;
    user.updatePoints(number3);
    //the points should be 1 now, so update by -2
    int number4 = -2;
    user.updatePoints(number4);
    int expResult3 = 0;
    assertEquals(expResult3, user.getPoints());
```

```
    }

    /**
     * Test of setName method, of class UserAccount.
     */
    @org.junit.jupiter.api.Test
    public void testSetName() {
        //testing that the name was right
        System.out.println("setName");
        UserAccount user = new UserAccount("AccountName", "Username", "Password");
        System.out.println("setName");
        String newName = "newName";
        user.setName(newName);
        assertEquals(newName, user.getAccountName());
    }

    /**
     * Test of setPassword method, of class UserAccount.
     */
    @org.junit.jupiter.api.Test
    public void testSetPassword() {
        //testing that the password was set right
        System.out.println("setPassword");
        UserAccount user = new UserAccount("AccountName", "Username", "Password");
        System.out.println("setPassword");
        String newPassword= "newPassword";
        user.setPassword(newPassword);
        assertEquals(newPassword, user.getPassword());
    }

    /**
     * Test of getAccountName method, of class UserAccount.
     */
    @org.junit.jupiter.api.Test
    public void testGetAccountName() {
        //testing getting the name
        System.out.println("getAccountName");
        UserAccount user = new UserAccount("AccountName", "Username", "Password");
        String expResult = "AccountName";
        String result = user.getAccountName();
        assertEquals(expResult, result);
    }

}
```

CategoryPageControllerTest:

```
package controller;
import org.junit.jupiter.api.AfterEach;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import gui.CategoryPage;
/*
 *
 * @author lewiscox
 * CategoryPageControllertest
 *
 */
public class CategoryPageControllerTest {

    private CategoryPageController categoryPageController;
    private CategoryPage categoryPage;

    public CategoryPageControllerTest() {
    }

    @BeforeEach
    public void setUp() {

        categoryPage = new CategoryPage();
        categoryPageController = new CategoryPageController(categoryPage);
    }

    @AfterEach
    public void tearDown() {

    }

    /**
     * Test of updateCategoryPosts method of class CategoryPageController
     */
    @Test
    public void testUpdateCategoryPosts() {
        System.out.println("updateCategoryPosts");


        String category = "Sports";
```

```
        categoryPageController.updateCategoryPosts(category);

    }

    /**
     * Test of constructor method, of class CategoryPageController
     */
    @Test
    public void testConstructor() {
        System.out.println("constructor");


        assertNotNull(categoryPageController, "CategoryPageController should not be null");
    }

}
```

LeaderboardControllerTest:

```
package  controller;
import gui.LeaderboardPage;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

/*
 * @author Lewis Cox
 * LeaderboardControllerTest
 *
 */
public class LeaderboardControllerTest {

    private LeaderboardController leaderboardController;
    private LeaderboardPage leaderboardPage;

    public LeaderboardControllerTest() {
    }

    @BeforeEach
    public void setUp() {

        leaderboardPage = new LeaderboardPage(null);
        leaderboardController = new LeaderboardController(leaderboardPage, null);
    }
```

```
    @AfterEach
    public void tearDown() {

    }

    /**
     * Test of updateLeaderboard method,
     */
    @Test
    public void testUpdateLeaderboard() {
        System.out.println("updateLeaderboard");


        leaderboardController.updateLeaderboard();


    }

    /**
     * Test of getTopPlayers method
     */
    @Test
    public void testGetTopPlayers() {
        System.out.println("getTopPlayers");




    }

    /**
     * Test of constructor method of c
     */
    @Test
    public void testConstructor() {
        System.out.println("Constructor");

        // Check if the controller is properly initialized with the leaderboardPage
        assertNotNull(leaderboardController, "LeaderboardController should not be null");
    }
}
```

LoginControllerTest:

```
/**
```

```
 * @author TabbieBrantley
 * LoginControllerTest file for LoginController class
 */
package controller;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

import gui.*;
import core.*;


public class LoginControllerTest {

    public LoginControllerTest() {
    }

    @BeforeAll
    public static void setUpClass() {
    }

    @AfterAll
    public static void tearDownClass() {
    }

    @BeforeEach
    public void setUp() {
    }

    @AfterEach
    public void tearDown() {
    }


    /**
     * Test of checkCreateLengths method, of class LoginController.
     */
    @Test
    public void testCheckCreateLengths() {
        //checking when they are a good length
```

```
        System.out.println("checkCreateLengths");
        String name = "goodlength";
        String username = "goodlength";
        String pwd = "goodlength";
        LoginController instance = new LoginController(new LoginWindow(),
                new CreateAccountWindow(), NetworkSystem.getInstance());
        boolean expResult = true;
        boolean result = instance.checkCreateLengths(name, username, pwd);
        assertEquals(expResult, result);

        //checking when they are more than 8 characters
        String lessName = "less";
        String lessUsername = "lesss";
        String lessPwd = "less";
        boolean expResult2 = false;
        boolean result2 = instance.checkCreateLengths(lessName, lessUsername, lessPwd);
        assertEquals(expResult2, result2);

        //chekcing when they are more than 20 characters
        String moreName = "123456789101112131415161718192O";
        String moreUsername = "123456789101112131415161718192O";
        String morePwd = "123456789101112131415161718192O";
        boolean expResult3 = false;
        boolean result3 = instance.checkCreateLengths(moreName, moreUsername, morePwd);
        assertEquals(expResult3, result3);

    }

    /**
     * Test of checkPasswords method, of class LoginController.
     */
    @Test
    public void testCheckPasswords() {
        //checking when the passwords match
        System.out.println("checkPasswords");
        String pwd = "matching";
        String confirmPwd = "matching";
        LoginController instance = new LoginController(new LoginWindow(), new CreateAccountWindow(),
NetworkSystem.getInstance());
        boolean expResult = true;
        boolean result = instance.checkPasswords(pwd, confirmPwd);
        assertEquals(expResult, result);

        //checking when the passwords do not match
```

```
        String badConfirmPwd = "not_matching";
        boolean expResult2 = false;
        boolean result2 = instance.checkPasswords(pwd, badConfirmPwd);
        assertEquals(expResult2, result2);

    }
}
```

NetworkWindowControllerTest:

```
package controller;
/**
 * @author TabbieBrantley
 * NetworkWindowController is the controller for the NetworkWindow
 * @class invariant listeners for the menubar buttons are added
 */

 //improts
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;

import core.NetworkSystem;
import gui.NetworkWindow;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;

public class NetworkWindowController {
    //NetworkWindow view attribute
    private NetworkWindow windowView;

    /**
     * Contsructor for the NetworkWindowController
     * @param netWindow the view as a NetworkWindow
     * @precondition n/a
     * @postcondition listeners are created
     */
    public NetworkWindowController(NetworkWindow netWindow){
        this.windowView = netWindow;
        this.createListeners();
        this.addNetworkWindowListener();
    }
```

```
    /**
     * createListeners creates listeners for the buttons that are on the MenuBarPanel
     * @precondition n/a
     * @postcondition n/a
     */
    private void createListeners(){
        //get all the buttons
        JButton homeB = this.windowView.getMenuBar().getHomeButton();
        JButton catB = this.windowView.getMenuBar().getCategoriesButton();
        JButton leadB = this.windowView.getMenuBar().getLeaderboardButton();
        JButton profileB = this.windowView.getMenuBar().getProfileButton();

        //adding ActionListener to the homeB
        homeB.addActionListener(new
        ActionListener() {
            public void actionPerformed(ActionEvent event){

NetworkWindowController.this.windowView.displayHomePage(NetworkSystem.getInstance().getCurrentU
ser());
            }
        });

        //adding ActionListener to the catB
        catB.addActionListener(new
        ActionListener() {
            public void actionPerformed(ActionEvent event){

NetworkWindowController.this.windowView.displayCategoriesPage(NetworkSystem.getInstance().getCurr
entUser());
            }
        });

        //adding ActionListener to the leadB
        leadB.addActionListener(new
        ActionListener() {
            public void actionPerformed(ActionEvent event){

NetworkWindowController.this.windowView.displayLeaderboardPage(NetworkSystem.getInstance().getC
urrentUser());
            }
        });

        //adding ActionLister to the profileB
        profileB.addActionListener(new
```

```
        ActionListener() {
            public void actionPerformed(ActionEvent event){

NetworkWindowController.this.windowView.displayProfilePage(NetworkSystem.getInstance().getCurrent
User());
            }
        });
    }
    /**
    * addNetworkWindowLIsteners adds a listener when the
    * windowView is closed
    * @precondition n/a
    * @postcondition n/a
    */
    public void addNetworkWindowListener(){
    JFrame frame = this.windowView.getWindowFrame();
    frame.addWindowListener(new WindowAdapter() {
        @Override
        public void windowClosing(WindowEvent e) {
            NetworkSystem.getInstance().saveSystemState();
        }
        });
    }
}
```

PostObserverTest:

```
package controller;

import core.Post;
import core.Repost;
import core.NetworkSystem;
import core.UserAccount;
import core.UserPost;
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;
import utility.ProfilePageSortStrategy;

/**
 * Test class for PostObserver.
 * Directly tests handleAgreeAction, handleDisagreeAction, handleRepostAction, and handleDeleteAction
methods.
 * @author thuhieu
 */
public class PostObserverTest {
```

```
    private Post mockPost;
    private PostObserver observer;
    private NetworkSystem networkSystem;
    private UserAccount currentUser;

    @BeforeEach
    public void setUp() {
        // Mock current user in NetworkSystem
        networkSystem = NetworkSystem.getInstance();

        networkSystem.addUserAccount("name1", "username1", "pass1");
        networkSystem.addUserAccount("name2", "username2", "pass2");
        networkSystem.addPosts(new UserPost("title1",
networkSystem.getUserAccountsList().get("username1")));
        networkSystem.addPosts(new UserPost("title2",
networkSystem.getUserAccountsList().get("username2")));

        mockPost = new UserPost("titl mock", networkSystem.getUserAccountsList().get("username1"));
        observer = new PostObserver(mockPost);
        networkSystem.addPosts(mockPost);

        currentUser = networkSystem.getUserAccountsList().get("username1");
        networkSystem.setCurrentUser(currentUser);
        mockPost.getCompletePostPanel(new ProfilePageSortStrategy());
    }

    @Test
    public void testHandleAgreeAction() {
        int originalPoint = mockPost.getPoints();
        // First Agree action
        observer.handleAgreeAction(currentUser.getUsername());
        assertEquals(originalPoint+1, mockPost.getPoints(), "Points should increase by 1 for Agree
action.");
        assertEquals("Agree", mockPost.getInteract(currentUser.getUsername()), "Interaction should be
'Agree'.");

        // Click Agree again (toggle off)
        observer.handleAgreeAction(currentUser.getUsername());
        assertEquals(originalPoint, mockPost.getPoints(), "Points should decrease by 1 when Agree is
toggled off.");
    }

    @Test
```

```
    public void testHandleDisagreeAction() {
        int originalPoint = mockPost.getPoints();
        // First Disagree action
        observer.handleDisagreeAction(currentUser.getUsername());
        assertEquals(originalPoint-1, mockPost.getPoints(), "Points should decrease by 1 for Disagree
action.");
        assertEquals("Disagree", mockPost.getInteract(currentUser.getUsername()), "Interaction should be
'Disagree'.");

        // Click Disagree again (toggle off)
        observer.handleDisagreeAction(currentUser.getUsername());
        assertEquals(originalPoint, mockPost.getPoints(), "Points should increase by 1 when Disagree is
toggled off.");
    }

    @Test
    public void testHandleRepostAction() {
        assertDoesNotThrow(() -> observer.handleRepostAction(), "Repost action should not throw
exceptions.");
    }

    @Test
    public void testHandleDeleteAction() {
        assertDoesNotThrow(() -> observer.handleDeleteAction(), "Delete action should not throw
exceptions.");
    }
}
```

ProfileControllerTest:

```
/**
 * @author Tabbie Brantley
 * test class for ProfileController
 */
package controller;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

import gui.*;
import core.*;
```

```
public class ProfileControllerTest {

    public ProfileControllerTest() {
    }

    @BeforeAll
    public static void setUpClass() {
    }

    @AfterAll
    public static void tearDownClass() {
    }

    @BeforeEach
    public void setUp() {
    }

    @AfterEach
    public void tearDown() {
    }

    /**
     * Test of checkNameLength method, of class ProfileController.
     */
    @Test
    public void testCheckNameLength() {
        System.out.println("checkNameLength");
        UserAccount user = new UserAccount("AccountName", "username", "password");
        ProfileController instance = new ProfileController(new ProfilePanel(user), user);

        //testing when good length
        String goodName = "good name";
        boolean expResult = true;
        boolean result = instance.checkNameLength(goodName);
        assertEquals(expResult, result);

        //testing when less than 8 characters
        String shortName = "bad";
        boolean expResult2 = false;
        boolean result2 = instance.checkNameLength(shortName);
        assertEquals(expResult2, result2);

        //testing when more than 20 characters
        String longName = "12345678910111213141516171819 20";
```

```
    boolean expResult3 = false;
    boolean result3 = instance.checkNameLength(longName);
    assertEquals(expResult3, result3);

}

/**
 * Test of changeName method, of class ProfileController.
 */
@Test
public void testChangeName() {
    //testing that the change name changed the name of the UserAccount
    System.out.println("changeName");
    UserAccount user = new UserAccount("AccountName", "username", "password");
    ProfileController instance = new ProfileController(new ProfilePanel(user), user);
    String newName = "NewAccountName";
    instance.changeName(newName);
    assertEquals(newName, user.getAccountName());
}

/**
 * Test of checkPasswordLength method, of class ProfileController.
 */
@Test
public void testCheckPasswordLength() {
    System.out.println("checkPasswordLength");
    UserAccount user = new UserAccount("AccountName", "username", "password");
    ProfileController instance = new ProfileController(new ProfilePanel(user), user);

    //testing when good length
    String goodPwd = "good password";
    boolean expResult = true;
    boolean result = instance.checkNameLength(goodPwd);
    assertEquals(expResult, result);

    //testing when less than 8 characters
    String shortPwd = "bad";
    boolean expResult2 = false;
    boolean result2 = instance.checkNameLength(shortPwd);
    assertEquals(expResult2, result2);

    //testing whe more than 20 characters
    String longPwd = "12345678910111213141516171819 20";
    boolean expResult3 = false;
```

```
        boolean result3 = instance.checkNameLength(longPwd);
        assertEquals(expResult3, result3);
    }

    /**
     * Test of checkPasswordsMatch method, of class ProfileController.
     */
    @Test
    public void testCheckPasswordsMatch() {
        System.out.println("checkPasswordsMatch");
        UserAccount user = new UserAccount("AccountName", "username", "password");
        ProfileController instance = new ProfileController(new ProfilePanel(user), user);

        //testing when passwords match
        String pwd = "matchingPwd";
        String confirmedPwd = "mathcingPwd";
        boolean expResult = true;
        boolean result = instance.checkPasswordsMatch(pwd, confirmedPwd);
        assertEquals(expResult, result);

        //testing when passwords do not match
        String badConfirmedPwd = "notMathcingPwd";
        boolean expResult2 = true;
        boolean result2 = instance.checkPasswordsMatch(pwd, badConfirmedPwd);
        assertEquals(expResult, result);

    }

    /**
     * Test of changePassword method, of class ProfileController.
     */
    @Test
    public void testChangePassword() {
        //testing that the UserAccount password was changed
        System.out.println("changePassword");
        UserAccount user = new UserAccount("AccountName", "username", "password");
        ProfileController instance = new ProfileController(new ProfilePanel(user), user);
        String newPwd = "NewPassword";
        instance.changeName(newPwd);
        assertEquals(newPwd, user.getAccountName());
    }

}
```

FeedPanelTest:

```
package gui;

import core.NetworkSystem;
import core.UserAccount;
import core.UserPost;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import utility.GeneralPageSortStrategy;

/**
 * FeedPanelTest tests for class FeedPanel
 * @author Hieu Truong
 */
public class FeedPanelTest {
    private NetworkSystem networkSystem;
    private UserAccount currentUser;

    public FeedPanelTest() {
    }

    @BeforeAll
    public static void setUpClass() {

    }

    @AfterAll
    public static void tearDownClass() {
    }

    @BeforeEach
    public void setUp() {
        networkSystem = NetworkSystem.getInstance();
        if (networkSystem.getPostsList().isEmpty()){
            networkSystem.addPosts(new UserPost("title1", new UserAccount("name1", "username1",
"pass1")));
            networkSystem.addPosts(new UserPost("title2", new UserAccount("name2", "username2",
"pass3")));
```

```
            networkSystem.addPosts(new UserPost("title3", new UserAccount("name3", "username3",
"pass3")));
            networkSystem.addPosts(new UserPost("title4", new UserAccount("name4", "username4",
"pass4")));

        }
        currentUser = networkSystem.getPostsList().get(0).getOwner();
        networkSystem.setCurrentUser(currentUser);
    }

    @AfterEach
    public void tearDown() {
    }

    /**
     * Test of displayFeed method, of class FeedPanel.
     */
    @Test
    public void testDisplayFeed() {
        System.out.println("displayFeed");
        FeedPanel instance = new FeedPanel(new GeneralPageSortStrategy());
        assertNotNull(instance);

    }

}
```

SortPostStrategyTest:

```
package utility;

import core.NetworkSystem;
import core.Post;
import core.UserPost;
import core.UserAccount;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.util.ArrayList;

import static org.junit.jupiter.api.Assertions.*;

/**
 * SortPostStrategyTest tests for SortPostStrategy and its implementations ProfilePageSortStrategy and
GeneralPageSortStrategy.
 * @author thuhieu
```

```
 */
public class SortPostStrategyTest {

    private NetworkSystem networkSystem;
    private UserAccount currentUser;

    @BeforeEach
    public void setUp() {
        networkSystem = NetworkSystem.getInstance();
        if (networkSystem.getPostsList().isEmpty()){
            networkSystem.addPosts(new UserPost("title1", new UserAccount("name1", "username1",
"pass1")));
            networkSystem.addPosts(new UserPost("title2", new UserAccount("name2", "username2",
"pass3")));
            networkSystem.addPosts(new UserPost("title3", new UserAccount("name3", "username3",
"pass3")));
            networkSystem.addPosts(new UserPost("title4", new UserAccount("name4", "username4",
"pass4")));

        }
        currentUser = networkSystem.getPostsList().get(0).getOwner();
        networkSystem.setCurrentUser(currentUser);
    }

    /**
     * Test for GeneralPageSortStrategy.
     */
    @Test
    public void testGeneralPageSortStrategy() {
        SortPostStrategy strategy = new GeneralPageSortStrategy();
        ArrayList<Post> sortedPosts = strategy.sortPost();
        Boolean expResult = true;
        Boolean result =true;
        for (int i=0; i<sortedPosts.size()-1; i++){
            if (sortedPosts.get(i).getPostID() <= sortedPosts.get(i+1).getPostID() ){
                result = false;
            }
        }
        assertEquals(result, expResult);

    }

    /**
     * Test for ProfilePageSortStrategy.
```

```
     */
    @Test
    public void testProfilePageSortStrategy() {
        SortPostStrategy strategy = new ProfilePageSortStrategy();

        ArrayList<Post> sortedPosts = strategy.sortPost();
        Boolean expResult = true;
        Boolean result =true;
        for (int i=0; i<sortedPosts.size()-1; i++){
            if (sortedPosts.get(i).getPostID() <= sortedPosts.get(i+1).getPostID() ||
                    (sortedPosts.get(i).getOwner().getUsername().equals(currentUser.getUsername()))){
                result = false;
            }
        }
        assertEquals(result, expResult);
    }
}
```