Quackstagram Report

Group Number: 46

Student Names: Ethan Goetsch, Mihai Solescu, Alvaro Murillo Terre

1. Requirement Analysis

1. 1 Current System Analysis

The current Quackstagram codebase contains functionality for displaying various interfaces, such as sing-in, sign-up, profile page, explore page, homepage, page where you can upload an image and a notifications page. Each page contains interactive elements enabling user input and navigation. The input is verified against a database that contains user credentials and profile information. A user profile consists of a profile picture, posts, amount of followers, amount of people following, a username and a bio. Users are able to post pictures with captions.

1. 2 Stakeholder Engagement

A highly requested feature from users is a messaging system, wherein two users can message each other, sending text or images. Users should also be able to form groups with one or more users allowing a user to send messages to all other users in the group.

Another feature requested by stakeholders is content moderation, safety and structure through admin users. Wherein, admin users can help address user concerns and can manage user content.

1.3 Requirement Specification

1.3.1 Functional Requirements

- User accounts with a username and password
- Save user account and their details
- Verify user account details
- Register a new account
- Log in to an existing account
- User accounts with the following functionality:
  - Have a bio and profile picture
  - Can follow other users
  - Can upload photos, with optional captions
  - Can explore other user's content
  - Receive notifications from other users
  - Can see the users that follow them
  - Can see the users that they follow
  - Can receive notifications when a new user follows you
  - Can see suggested posts
  - Can see the posts of the users they follow

1.3.2 Non-Functional Requirements

- Modern and intuitive UI
- Secure credentials

1.4 Prioritization of Requirements

Requirement ID	Requirement Description	Priority
REQ-01	User account with a username and password	Hight
REQ-02	Save user account details	High

Requirement ID	Requirement Description	Priority
REQ-03	Verify user account details	High
REQ-04	Allow users to create new accounts	High
REQ-05	Allow users to login to existing accounts	High
REQ-06	Basic user profile, such as username, bio, profile picture	Medium
REQ-07	Allow users to follow other users	Medium
REQ-08	Allow users to post content	Medium
REQ-09	Allow users to explore other user content	Medium
REQ-10	Allow users to receive notifications from other users	Low
REQ-11	Modern and intuitive UI	Low
REQ-12	Securely store and retrieve data	High

Rationalization for Requirement Prioritization:

High Priority:

User accounts with basic functionality, such as created new accounts and logging in to existing accounts as well as saving account details is essential to the functionality of the software. Those details should also be managed securely to ensure user privacy, without it our users are vulnerable.

Medium Priority:

User accounts with interactive functionality, such as engaging with other users, posting content and viewing content is important because it is required for the user to interact with the software.

Low Priority:

A notification system is beneficial to the user experience it is not required in order to interact with the software.

2. UML Modeling

2.1 Current System UML Diagrams

Refer to Appendix Items A - G

2.2 Use Cases

2.2. 1Create Post

Actors: User, System

Pre-Conditions: User must be signed-in

Flows:

ID	Actor	Action	Notes
BF-1	User	Press the “Create Post” button	
BF-2	System	Display “Create Post” Page	
BF-3	User	User optionally enters a caption	
BF-4	User	User uploads image	
BF-5	System	Copies post into system	AF-1 Wrong Image format
BF-6	System	Notifies the user that the image has been uploaded	
AF-1-1	System	Notifies the user that there is an error with the image format	
AF-1-2	System	GOTO BF-3	

2.2.2 Follow a User

Actors: User, System

Pre-conditions: User and user to be followed must have an account on Quackstagram

Flow:

ID	Actors	Action	Notes
BF-1	User	Navigate to explore page	
BF-2	System	Display the explore page	
BF-3	User	Browse explore page AND select post	
BF-4	System	Display Selected Post	
BF-5	User	Navigate the profile of the user that created the post	
BF-6	System	Display the profile of the user that selected the post	
BF-7	User	Follow the selected user by pressing “Follow”	AF-1 The user is the same user as the current user AF-2 The user is already followed by the current user
BF-8	System	Change the text on the follow button to “Following” AND update the follower/following counts of the current and selected user	
AF-1-1			There is no “unfollow” button, so the user does not have the option of following himself
AF-2-1	User	Press “Unfollow” button	Instead of the text “follow”, the button has the text “following”, indicating to the user that the user that he is already following the selected user
AF-2-2	System	Nothing happens	

2.2.3 Sign In

Actors: User, System

Pre-conditions: User is in the Sign in page

Flows:

ID	Actors	Action	Notes
BF-1	User	Start the application	
BF-2	System	Display the Sign-in page	
BF-3	User	Modify the text fields AND press “Sign-in” button	AF-1: User does not have an account- *See Sign-up use case
BF-4	System	Outputs “It did” in default CLI AND displays user profile page	AF-2: Inputted incorrect credentials
AF-2	System	Outputs “It didn’t in default CLI”	

2.2.4 Sign up

Actors: User, System

Pre-conditions: User is in the SignIn Page

Flows:

ID	Actors	Action	Notes
BF-1	User	Press the “No account? register here” button	
BF-2	System	Display the sign up page	
BF-3	User	modify text in text field AND press register button	AF-1: Uploads profile picture as well
BF-4	System	Opens the picture upload dialog	
BF-5	User	Selects Picture OR closes dialog	
BF-6	System	If a picture was selected, saves the picture	
BF-7	System	Creates a user profile AND displays the user Sign-in	

		page	
AF-1-1	User	Press the “upload” picture button	
AF-1-2	System	Opens the picture upload dialog	
AF-1-3	User	Selects a picture OR closes the dialog	
AF-1-4	System	If a picture was selected, save the picture	
AF-1-4		GOTO BF-4	

2.2.5 Like a post

Actors: User, System

Pre-conditions: User is viewing post of current user

Flows:

ID	Actors	Action	Notes
BF-1	User	Finds a post on the explore page	
BF-2	System	Displays the post from the explore page	
BF-3	User	Selects to view the post of the user	
BF-4	System	Displays the post of the current user	
BF-5	User	Presses the “Like” button on the post	AF-1: The user wants to unlike the post
BF-6	System	Changes the text of the “like” button AND displays it to the user	
BF-7	System	Updates the count of the number of likes for the current users post	
AF-1-1	User	Presses the “like” button again to unlike the current users post	

AF-1-2	System	The text of the “like”button is changed AND is displayed to the user		
AF-1-3	System	Updates the count of the number of likes for the current users post		

2.2.6 Edit User profile

Actors: User, System

Pre-conditions: User is signed into Quackstagram

Flows:

ID	Actors	Action	Notes	
BF-1	User	Goes to profile page		
BF-2	System	Displays profile page		
BF-3	User	Selects “Edit profile page”		
BF-4	System	Displays “Edit profile” page		
BF-5	User	Modify Bio OR Upload new profile picture	AF-1: Image in Wrong format	
BF-6	System	Updates the bio OR profile picture		
BF-7	System	Display updated profile page		
AF-1-1	System	Display “Error” message and notify user of incorrect image format		
AF-1-2		GOTO BF-4		

3. Refactoring

Magic Strings

Path Files

Before

Hard Coded Strings throughout the application

```
private boolean verifyCredentials(String username, String p
assword) {
    try (BufferedReader reader = new BufferedReader(new Fil
eReader("data/credentials.txt"))) {
        String line;
        while ((line = reader.readLine()) != null) {
            String[] credentials = line.split(":");
            if (credentials[0].equals(username) && credenti
als[1].equals(password)) {
            String bio = credentials[2];
            // Create User object and save information
        newUser = new User(username, bio, password); // Ass
uming User constructor takes these parameters
        saveUserInformation(newUser);

                return true;
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return false;
}
```

```
    private void saveCredentials(String username, String pa
ssword, String bio) {
        try (BufferedWriter writer = new BufferedWriter(new
```

```
FileWriter("data/credentials.txt", true))) {
            writer.write(username + ":" + password + ":" +
bio);
            writer.newLine();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
   try (BufferedReader reader = Files.newBufferedReader(Pat
hs.get("data", "notifications.txt"))) {
    String line;
    while ((line = reader.readLine()) != null) {
        String[] parts = line.split(";");
        if (parts[0].trim().equals(currentUsername)) {
            // Format the notification message
            String userWhoLiked = parts[1].trim();
            String imageId = parts[2].trim();
            String timestamp = parts[3].trim();
            String notificationMessage = userWhoLiked + " l
iked your picture - " + getElapsedTime(timestamp) + " ago";

            // Add the notification to the panel
            JPanel notificationPanel = new JPanel(new Borde
rLayout());
            notificationPanel.setBorder(BorderFactory.creat
eEmptyBorder(5, 10, 5, 10));

            JLabel notificationLabel = new JLabel(notificat
ionMessage);
            notificationPanel.add(notificationLabel, Border
Layout.CENTER);

            // Add profile icon (if available) and timestam
p
            // ... (Additional UI components if needed)

            contentPanel.add(notificationPanel);
```

```
        }
    }
}
```

After

Class stores all the paths for the application

```
public class Paths
{
    // root path
    public static final Path rootPath = Path.of("src/refact
ored/");
    // data path
    public static final Path dataPath = Path.of("data/");
    // contents
    public static final Path contentsPath = Path.of(dataPat
h.toString(), "contents/");
    // profile pictures
    public static final Path profilePicturesPath = Path.of
(contentsPath.toString(), "/profile_pictures/");
    // uploads
    public static final Path uploadsPath = Path.of(contents
Path.toString(), "/uploads/");

    //icons
    public static final Path iconsPath = Path.of(dataPath.t
oString(), "/icons");
    public static final Path logoPath = Path.of(iconsPath.t
oString(), "/DACS.png");
    public static final Path homeIconPath = Path.of(iconsPa
th.toString(), "/home.png");
    public static final Path searchIconPath = Path.of(icons
Path.toString(), "/search.png");
    public static final Path addIconPath = Path.of(iconsPat
h.toString(), "/add.png");
    public static final Path heartIconPath = Path.of(iconsP
```

```
ath.toString(), "/heart.png");
    public static final Path profileIconPath = Path.of(icon
sPath.toString(), "/profile.png");

    //content
    public static final Path contentsDBPath = Path.of(conte
ntsPath.toString(), "/contents.json");
    //posts
    public static final Path postsDBPath = Path.of(contents
Path.toString(), "/posts.json");
    //messages
    public static final Path messagesDBPath = Path.of(dataP
ath.toString(), "/messages.json");
    //comments
    public static final Path commentsDBPath = Path.of(dataP
ath.toString(), "/comments.json");
    //follows
    public static final Path followsDBPath = Path.of(dataPa
th.toString(), "/follows.json");
    //likes
    public static final Path likesDBPath = Path.of(dataPat
h.toString(), "/likes.json");
    //data
    public static final Path usersDBPath = Path.of(dataPat
h.toString(), "/users.json");
}
```

```
public abstract class DBManager<T> {

    @SuppressWarnings("unchecked")
    protected static <T> ArrayList<T> retrieve(Path from)
    {
        // try-with-resources to auto-close the file
        try (FileInputStream fos = new FileInputStream(fro
m.toFile());
        ObjectInputStream oos = new ObjectInputStream(fos))
        {
            // if the file contains a null object, return a
```

```
n empty list
            ArrayList<T> temp = (ArrayList<T>) oos.readObje
ct();
            if (temp != null)
            {
                return temp;
            }
            return new ArrayList<T>();
        } catch (Exception e)
        {
            //maybe do something here
            e.printStackTrace();
            return new ArrayList<>();
        }
    }

    protected static <T> void store(ArrayList<T> from, Path
to)
    {
        // try-with-resources to auto-close the file
        try (FileOutputStream fos = new FileOutputStream(t
o.toFile());
        ObjectOutputStream oos = new ObjectOutputStream(fo
s))
        {
            // if the object is null, write an empty list t
o file
            if(from == null)
            {
                from = new ArrayList<T>();
            }
            oos.writeObject(from);
        } catch (Exception e)
        {
            //maybe do something here
            e.printStackTrace();
        }
```

```
    }
}
```

Rationale

1. **Enhances Maintainability**
    a. With dedicated paths for each data file it is easier to change any data files since there is only a single reference to it throughout the code base
2. **Clear Responsibility**
    a. A single class responsible for data paths helps maintain single responsibility throughout the application. The class has a single clear purpose.
3. **Scalability**
    a. As new data files are added it is clear and concise where they must be referenced

Button Types

Before

Hard coded button strings and responses to those string

```
    private JButton createIconButton(String iconPath, Strin
g buttonType) {
        ImageIcon iconOriginal = new ImageIcon(iconPath);
        Image iconScaled = iconOriginal.getImage().getScale
dInstance(NAV_ICON_SIZE, NAV_ICON_SIZE, Image.SCALE_SMOOT
H);
        JButton button = new JButton(new ImageIcon(iconScal
ed));
        button.setBorder(BorderFactory.createEmptyBorder
());
        button.setContentAreaFilled(false);

        // Define actions based on button type
        if ("home".equals(buttonType)) {
```

```
            button.addActionListener(e -> openHomeUI());
        } else if ("profile".equals(buttonType)) {
            button.addActionListener(e -> openProfileUI());
        } else if ("notification".equals(buttonType)) {
            button.addActionListener(e -> notificationsUI
());
        } else if ("explore".equals(buttonType)) {
            button.addActionListener(e -> exploreUI());
        }
        return button;


    }
```

After

Enum for button page type

```
    /**
     * the nav bar needs to know which frame it belongs to
in order to close that frame on button click.
     * there are ways of getting the reference of the frame
from the JPannel, but it has to attached first.
     * @param this
     * @return
     */
    public static JPanel createNavigationPanel(JFrame topFr
ame, IAction<PageType> navigateAction)
    {
        JPanel navigationPanel = new JPanel();
        navigationPanel.setBackground(new Color(249, 249, 2
49));
        navigationPanel.setLayout(new BoxLayout(navigationP
anel, BoxLayout.X_AXIS));
        navigationPanel.setBorder(BorderFactory.createEmpty
Border(5, 5, 5, 5));

        // Buttons
        navigationPanel.add(createNavButton(navigateAction,
```

```
PageType.HOME, Paths.homeIconPath, topFrame));
        navigationPanel.add(Box.createHorizontalGlue());
        navigationPanel.add(createNavButton(navigateAction,
PageType.EXPLORE, Paths.searchIconPath, topFrame));
        navigationPanel.add(Box.createHorizontalGlue());
        navigationPanel.add(createNavButton(navigateAction,
PageType.UPLOAD, Paths.addIconPath, topFrame));
        navigationPanel.add(Box.createHorizontalGlue());
        navigationPanel.add(createNavButton(navigateAction,
PageType.NOTIFICATION, Paths.heartIconPath, topFrame));
        navigationPanel.add(Box.createHorizontalGlue());
        navigationPanel.add(createNavButton(navigateAction,
PageType.PROFILE, Paths.profileIconPath, topFrame));

        return navigationPanel;
    }
```

Rationale

1. **Enhances Robustness**
    a. Referencing a type instead of a string improves robustness by being resilient to name changes
    b. Due to a reference, the types can be debugged and tracked
        i. I.e. If a value is deleted, the compiler will warn against it
2. **Promotes Open for Extension Closed for Modification principle**
    a. The buttons functionality can be extended without modifying the creation of the button

UI Element Factory

Before

UI classes contain duplicate code to create UI elements

```
    private JPanel createHeaderPanel() {

        // Header Panel (reuse from InstagramProfileUI or c
```

```
ustomize for home page)
         // Header with the Register label
         JPanel headerPanel = new JPanel(new FlowLayout(Flo
wLayout.CENTER));
         headerPanel.setBackground(new Color(51, 51, 51));
// Set a darker background for the header
         JLabel lblRegister = new JLabel(" Upload Image
🐥");
         lblRegister.setFont(new Font("Arial", Font.BOLD, 1
6));
         lblRegister.setForeground(Color.WHITE); // Set the
text color to white
         headerPanel.add(lblRegister);
         headerPanel.setPreferredSize(new Dimension(WIDTH,
40)); // Give the header a fixed height
         return headerPanel;
   }

   private JPanel createNavigationPanel() {
       // Create and return the navigation panel
        // Navigation Bar
        JPanel navigationPanel = new JPanel();
        navigationPanel.setBackground(new Color(249, 249, 2
49));
        navigationPanel.setLayout(new BoxLayout(navigationP
anel, BoxLayout.X_AXIS));
        navigationPanel.setBorder(BorderFactory.createEmpty
Border(5, 5, 5, 5));

        navigationPanel.add(createIconButton("img/icons/hom
e.png", "home"));
        navigationPanel.add(Box.createHorizontalGlue());
        navigationPanel.add(createIconButton("img/icons/sea
rch.png","explore"));
        navigationPanel.add(Box.createHorizontalGlue());
        navigationPanel.add(createIconButton("img/icons/ad
d.png"," "));
        navigationPanel.add(Box.createHorizontalGlue());
```

```
        navigationPanel.add(createIconButton("img/icons/hea
rt.png","notification"));
        navigationPanel.add(Box.createHorizontalGlue());
        navigationPanel.add(createIconButton("img/icons/pro
file.png", "profile"));

        return navigationPanel;
   }

    private JButton createIconButton(String iconPath, Strin
g buttonType) {
        ImageIcon iconOriginal = new ImageIcon(iconPath);
        Image iconScaled = iconOriginal.getImage().getScale
dInstance(NAV_ICON_SIZE, NAV_ICON_SIZE, Image.SCALE_SMOOT
H);
        JButton button = new JButton(new ImageIcon(iconScal
ed));
        button.setBorder(BorderFactory.createEmptyBorder
());
        button.setContentAreaFilled(false);

        // Define actions based on button type
        if ("home".equals(buttonType)) {
            button.addActionListener(e -> openHomeUI());
        } else if ("profile".equals(buttonType)) {
            button.addActionListener(e -> openProfileUI());
        } else if ("notification".equals(buttonType)) {
            button.addActionListener(e -> notificationsUI
());
        } else if ("explore".equals(buttonType)) {
            button.addActionListener(e -> exploreUI());
        }
        return button;


    }
```

After

Single UI Element Factory to create and return common UI elements

```
public class UIElementFactory
{
    private static final int NAV_ICON_SIZE = 20;

    public static JPanel createHeaderPanel(int width, Strin
g title)
    {
        // Header with the Register label
        JPanel headerPanel = new JPanel(new FlowLayout(Flow
Layout.CENTER));
        headerPanel.setBackground(new Color(51, 51, 51));
// Set a darker background for the header
        JLabel lblRegister = new JLabel(title);
        lblRegister.setFont(new Font("Arial", Font.BOLD, 1
6));
        lblRegister.setForeground(Color.WHITE); // Set the
text color to white
        headerPanel.add(lblRegister);
        headerPanel.setPreferredSize(new Dimension(width, 4
0)); // Give the header a fixed height
        return headerPanel;
    }

    /**
     * the nav bar needs to know which frame it belongs to
in order to close that frame on button click.
     * there are ways of getting the reference of the frame
from the JPannel, but it has to attached first.
     * @param this
     * @return
     */
    public static JPanel createNavigationPanel(JFrame topFr
ame, IAction<PageType> navigateAction)
    {
        JPanel navigationPanel = new JPanel();
        navigationPanel.setBackground(new Color(249, 249, 2
```

```
49));
        navigationPanel.setLayout(new BoxLayout(navigationP
anel, BoxLayout.X_AXIS));
        navigationPanel.setBorder(BorderFactory.createEmpty
Border(5, 5, 5, 5));

        // Buttons
        navigationPanel.add(createNavButton(navigateAction,
PageType.HOME, Paths.homeIconPath, topFrame));
        navigationPanel.add(Box.createHorizontalGlue());
        navigationPanel.add(createNavButton(navigateAction,
PageType.EXPLORE, Paths.searchIconPath, topFrame));
        navigationPanel.add(Box.createHorizontalGlue());
        navigationPanel.add(createNavButton(navigateAction,
PageType.UPLOAD, Paths.addIconPath, topFrame));
        navigationPanel.add(Box.createHorizontalGlue());
        navigationPanel.add(createNavButton(navigateAction,
PageType.NOTIFICATION, Paths.heartIconPath, topFrame));
        navigationPanel.add(Box.createHorizontalGlue());
        navigationPanel.add(createNavButton(navigateAction,
PageType.PROFILE, Paths.profileIconPath, topFrame));

        return navigationPanel;
    }

    private static JButton createNavButton(IAction<PageType
> navigateAction, PageType pageType, Path iconPath, JFrame
topFrame)
    {
        ImageIcon iconOriginal = new ImageIcon(iconPath.toS
tring());
        Image iconScaled = iconOriginal.getImage().getScale
dInstance(NAV_ICON_SIZE, NAV_ICON_SIZE, Image.SCALE_SMOOT
H);
        JButton button = new JButton(new ImageIcon(iconScal
ed));
        button.setBorder(BorderFactory.createEmptyBorder
());
```

```
        button.setContentAreaFilled(false);
        button.addActionListener(e -> navigateAction.execut
e(pageType));
        return button;
    }

    public static JScrollPane createImageGridPanel(int GRID
_IMAGE_SIZE, Iterable<Post> posts, IAction2<Post, ImageIcon
> imageClickedListener)
    {
        // panel to be decorated with scroll bar
        JPanel contentPanel = new JPanel();
        contentPanel.setLayout(new GridLayout(0, 3,2, 2));
// Grid layout for image grid

        // get user posts
        for (Post post : posts)
        {
            Path path = post.getFilePath();
            ImageIcon imageIcon = new ImageIcon(new ImageIc
on(path.toString()).getImage().getScaledInstance(GRID_IMAGE
_SIZE, GRID_IMAGE_SIZE, Image.SCALE_SMOOTH));
            JLabel imageLabel = new JLabel(imageIcon);

            // Add a mouse listener to the image label
            imageLabel.addMouseListener(new MouseAdapter()
{
                @Override
                public void mouseClicked(java.awt.event.Mou
seEvent e) {
                    imageClickedListener.execute(post, imag
eIcon);
                }
            });
            contentPanel.add(imageLabel);
        }

        // scroll pane decorator
```

```
        JScrollPane scrollPane = new JScrollPane(contentPan
el);
        scrollPane.setHorizontalScrollBarPolicy(JScrollPan
e.HORIZONTAL_SCROLLBAR_NEVER);
        scrollPane.setVerticalScrollBarPolicy(JScrollPane.V
ERTICAL_SCROLLBAR_AS_NEEDED);

        return scrollPane;
    }
}
```

Rationale

1. **Promotes Single Responsibility**
    a. UI does not need to create its UI elements
2. **Enhances Scalability**
    a. Easily allows for implementing new features to all UI elements
        i. I.e. A Dark and Light mode for the UI visuals
            1. Abstract Factory → Dark UI Element Factory | Light UI Element Factory
3. **Enhances Maintainability**
    a. A dedicated class for for UI element creation allows for easy modification to the creation of UI elements
        i. I.e. Changing the icon displayed for the profile profile
4. **Respects DRY**
    a. Reduces and removes duplicate code for creating common UI elements

Model-View-Controller

Coupled UI

Before

High Coupling

Each UI class instantiates every other UI class

```
 private void ImageUploadUI() {
        // Open InstagramProfileUI frame
        this.dispose();
        ImageUploadUI upload = new ImageUploadUI();
        upload.setVisible(true);
    }

    private void openProfileUI() {
        // Open InstagramProfileUI frame
        this.dispose();
        InstagramProfileUI profileUI = new InstagramProfile
UI();
        profileUI.setVisible(true);
    }

     private void notificationsUI() {
        // Open InstagramProfileUI frame
        this.dispose();
        NotificationsUI notificationsUI = new Notifications
UI();
        notificationsUI.setVisible(true);
    }

    private void openHomeUI() {
        // Open InstagramProfileUI frame
        this.dispose();
        QuakstagramHomeUI homeUI = new QuakstagramHomeUI();
        homeUI.setVisible(true);
    }

    private void exploreUI() {
        // Open InstagramProfileUI frame
        this.dispose();
        ExploreUI explore = new ExploreUI();
        explore.setVisible(true);
    }
```

After

Loose

A single UI Manager handles navigation between UI and instantiates the relevant UI

```
public class UIManager
{
    private IPageController signInPage;
    private IPageController signUpPage;
    private IPageController homePage;
    private IPageController explorePage;
    private IPageController uploadPage;
    private IPageController notificationPage;
    private IPageController profilePage;

    private IPageController activePage;

    public void openSignUp()
    {
        signUpPage = new SignUpController(this);
        setActivePage(signUpPage);
    }

    public void openSignIn()
    {
        signInPage = new SignInController(this);
        setActivePage(signInPage);
    }

    public void openHome(int userId)
    {
        homePage = new HomeController(this);
        setActivePage(homePage);
    }

    public void openExplore(int userId)
    {
        OtherUsersPosts posts = new OtherUsersPosts(current
```

```
UserId);
        explorePage = new ExploreController(this, posts);
        setActivePage(explorePage);
    }

    private void openUpload(int userId)
    {
        uploadPage = new UploadController(this, userId, Use
rDBManager.getUsername(userId));
        setActivePage(uploadPage);
    }

    private void openNotifications(int userId)
    {
        notificationPage = new NotificationController(this,
currentUserId);
        setActivePage(notificationPage);
    }

    public void openProfile(int userId)
    {
        profilePage = new ProfileController(this, userId, c
urrentUserId == userId, FollowDBManager.isAFollowingB(curre
ntUserId, userId), PostDBManager.getUserPosts(userId));
        setActivePage(profilePage);
    }

    public void navigateToPage(PageType page)
    {
        switch (page)
        {
            case EXPLORE:
                openExplore(currentUserId);
                break;
            case HOME:
                openHome(currentUserId);
                break;
            case NOTIFICATION:
```

```
                openNotifications(currentUserId);
                break;
            case PROFILE:
                openProfile(currentUserId);
                break;
            case UPLOAD:
                openUpload(currentUserId);
                break;
            default:
                break;
        }
    }
}
```

Rationale

1. **Enhances Maintainability**
    a. Navigation between pages is handled in a single class, which allows for easy modification to the handling of the navigation
2. **Enhances Robustness**
    a. Loosely coupled classes allows for easy modification to a class without causing changes in other classes

Responsibility

Before

UI has no single responsibility. Controls Logic, Saving/Loading Data, Navigation and Interaction

(I hope the shear size of this class proves my point)

```
public class InstagramProfileUI extends JFrame {

    private static final int WIDTH = 300;
    private static final int HEIGHT = 500;
    private static final int PROFILE_IMAGE_SIZE = 80; // Ad
```

```
justed size for the profile image to match UI
    private static final int GRID_IMAGE_SIZE = WIDTH / 3;
// Static size for grid images
    private static final int NAV_ICON_SIZE = 20; // Correct
ed static size for bottom icons
    private JPanel contentPanel; // Panel to display the im
age grid or the clicked image
    private JPanel headerPanel;   // Panel for the header
    private JPanel navigationPanel; // Panel for the naviga
tion
    private User currentUser; // User object to store the c
urrent user's information

    public InstagramProfileUI(User user) {
        this.currentUser = user;
         // Initialize counts
        int imageCount = 0;
        int followersCount = 0;
        int followingCount = 0;

        // Step 1: Read image_details.txt to count the numb
er of images posted by the user
    Path imageDetailsFilePath = Paths.get("img", "image_det
ails.txt");
    try (BufferedReader imageDetailsReader = Files.newBuffe
redReader(imageDetailsFilePath)) {
        String line;
        while ((line = imageDetailsReader.readLine()) != nu
ll) {
            if (line.contains("Username: " + currentUser.ge
tUsername())) {
                imageCount++;
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
```

```
    // Step 2: Read following.txt to calculate followers an
d following
    Path followingFilePath = Paths.get("data", "following.t
xt");
    try (BufferedReader followingReader = Files.newBuffered
Reader(followingFilePath)) {
        String line;
        while ((line = followingReader.readLine()) != null)
{
            String[] parts = line.split(":");
            if (parts.length == 2) {
                String username = parts[0].trim();
                String[] followingUsers = parts[1].split
(";");
                if (username.equals(currentUser.getUsername
())) {
                    followingCount = followingUsers.length;
                } else {
                    for (String followingUser : followingUs
ers) {
                        if (followingUser.trim().equals(cur
rentUser.getUsername())) {
                            followersCount++;
                        }
                    }
                }
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    String bio = "";

    Path bioDetailsFilePath = Paths.get("data", "credential
s.txt");
    try (BufferedReader bioDetailsReader = Files.newBuffere
dReader(bioDetailsFilePath)) {
```

```
        String line;
        while ((line = bioDetailsReader.readLine()) != nul
l) {
            String[] parts = line.split(":");
            if (parts[0].equals(currentUser.getUsername())
&& parts.length >= 3) {
                bio = parts[2];
                break; // Exit the loop once the matching b
io is found
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    System.out.println("Bio for " + currentUser.getUsername
() + ": " + bio);
    currentUser.setBio(bio);



    currentUser.setFollowersCount(followersCount);
    currentUser.setFollowingCount(followingCount);
    currentUser.setPostCount(imageCount);

    System.out.println(currentUser.getPostsCount());

     setTitle("DACS Profile");
        setSize(WIDTH, HEIGHT);
        setMinimumSize(new Dimension(WIDTH, HEIGHT));
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new BorderLayout());
        contentPanel = new JPanel();
        headerPanel = createHeaderPanel();       // Initial
ize header panel
        navigationPanel = createNavigationPanel(); // Initi
alize navigation panel

        initializeUI();
```

```
    }


      public InstagramProfileUI() {

        setTitle("DACS Profile");
        setSize(WIDTH, HEIGHT);
        setMinimumSize(new Dimension(WIDTH, HEIGHT));
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new BorderLayout());
        contentPanel = new JPanel();
        headerPanel = createHeaderPanel();       // Initial
ize header panel
        navigationPanel = createNavigationPanel(); // Initi
alize navigation panel
        initializeUI();
    }

    private void initializeUI() {
        getContentPane().removeAll(); // Clear existing com
ponents

        // Re-add the header and navigation panels
        add(headerPanel, BorderLayout.NORTH);
        add(navigationPanel, BorderLayout.SOUTH);

        // Initialize the image grid
        initializeImageGrid();

        revalidate();
        repaint();
    }

    private JPanel createHeaderPanel() {
        boolean isCurrentUser = false;
        String loggedInUsername = "";

        // Read the logged-in user's username from users.tx
```

```
t
    try (BufferedReader reader = Files.newBufferedReader(Pa
ths.get("data", "users.txt"))) {
        String line = reader.readLine();
        if (line != null) {
            loggedInUsername = line.split(":")[0].trim();
            isCurrentUser = loggedInUsername.equals(current
User.getUsername());
        }
    } catch (IOException e) {
        e.printStackTrace();
    }


       // Header Panel
        JPanel headerPanel = new JPanel();
        try (Stream<String> lines = Files.lines(Paths.get
("data", "users.txt"))) {
            isCurrentUser = lines.anyMatch(line -> line.sta
rtsWith(currentUser.getUsername() + ":"));
        } catch (IOException e) {
            e.printStackTrace();  // Log or handle the exce
ption as appropriate
        }

        headerPanel.setLayout(new BoxLayout(headerPanel, Bo
xLayout.Y_AXIS));
        headerPanel.setBackground(Color.GRAY);

        // Top Part of the Header (Profile Image, Stats, Fo
llow Button)
        JPanel topHeaderPanel = new JPanel(new BorderLayout
(10, 0));
        topHeaderPanel.setBackground(new Color(249, 249, 24
9));

        // Profile image
        ImageIcon profileIcon = new ImageIcon(new ImageIcon
```

```
("img/storage/profile/"+currentUser.getUsername()+".png").g
etImage().getScaledInstance(PROFILE_IMAGE_SIZE, PROFILE_IMA
GE_SIZE, Image.SCALE_SMOOTH));
        JLabel profileImage = new JLabel(profileIcon);
        profileImage.setBorder(BorderFactory.createEmptyBor
der(10, 10, 10, 10));
        topHeaderPanel.add(profileImage, BorderLayout.WES
T);

        // Stats Panel
        JPanel statsPanel = new JPanel();
        statsPanel.setLayout(new FlowLayout(FlowLayout.CENT
ER, 10, 0));
        statsPanel.setBackground(new Color(249, 249, 249));
        System.out.println("Number of posts for this user"+
currentUser.getPostsCount());
        statsPanel.add(createStatLabel(Integer.toString(cur
rentUser.getPostsCount()) , "Posts"));
        statsPanel.add(createStatLabel(Integer.toString(cur
rentUser.getFollowersCount()), "Followers"));
        statsPanel.add(createStatLabel(Integer.toString(cur
rentUser.getFollowingCount()), "Following"));
        statsPanel.setBorder(BorderFactory.createEmptyBorde
r(25, 0, 10, 0)); // Add some vertical padding


// Follow Button
// Follow or Edit Profile Button
// followButton.addActionListener(e -> handleFollowAction(c
urrentUser.getUsername()));
JButton followButton;
    if (isCurrentUser) {
        followButton = new JButton("Edit Profile");
    } else {
        followButton = new JButton("Follow");

        // Check if the current user is already being follo
wed by the logged-in user
```

```
        Path followingFilePath = Paths.get("data", "followi
ng.txt");
        try (BufferedReader reader = Files.newBufferedReade
r(followingFilePath)) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] parts = line.split(":");
                if (parts[0].trim().equals(loggedInUsernam
e)) {
                    String[] followedUsers = parts[1].split
(";");
                    for (String followedUser : followedUser
s) {
                        if (followedUser.trim().equals(curr
entUser.getUsername())) {
                            followButton.setText("Followin
g");
                            break;
                        }
                    }
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        followButton.addActionListener(e -> {
            handleFollowAction(currentUser.getUsername());
            followButton.setText("Following");
        });
    }

followButton.setAlignmentX(Component.CENTER_ALIGNMENT);
followButton.setFont(new Font("Arial", Font.BOLD, 12));
followButton.setMaximumSize(new Dimension(Integer.MAX_VALU
E, followButton.getMinimumSize().height)); // Make the butt
on fill the horizontal space
followButton.setBackground(new Color(225, 228, 232)); // A
soft, appealing color that complements the UI
```

```
followButton.setForeground(Color.BLACK);
followButton.setOpaque(true);
followButton.setBorderPainted(false);
followButton.setBorder(BorderFactory.createEmptyBorder(10,
0, 10, 0)); // Add some vertical padding


        // Add Stats and Follow Button to a combined Panel
        JPanel statsFollowPanel = new JPanel();
        statsFollowPanel.setLayout(new BoxLayout(statsFollo
wPanel, BoxLayout.Y_AXIS));
        statsFollowPanel.add(statsPanel);
        statsFollowPanel.add(followButton);
        topHeaderPanel.add(statsFollowPanel, BorderLayout.C
ENTER);

        headerPanel.add(topHeaderPanel);

     // Profile Name and Bio Panel
JPanel profileNameAndBioPanel = new JPanel();
profileNameAndBioPanel.setLayout(new BorderLayout());
profileNameAndBioPanel.setBackground(new Color(249, 249, 24
9));

JLabel profileNameLabel = new JLabel(currentUser.getUsernam
e());
profileNameLabel.setFont(new Font("Arial", Font.BOLD, 14));
profileNameLabel.setBorder(BorderFactory.createEmptyBorder
(10, 10, 0, 10)); // Padding on the sides

JTextArea profileBio = new JTextArea(currentUser.getBio());
System.out.println("This is the bio "+currentUser.getUserna
me());
profileBio.setEditable(false);
profileBio.setFont(new Font("Arial", Font.PLAIN, 12));
profileBio.setBackground(new Color(249, 249, 249));
profileBio.setBorder(BorderFactory.createEmptyBorder(0, 10,
10, 10)); // Padding on the sides
```

```
profileNameAndBioPanel.add(profileNameLabel, BorderLayout.N
ORTH);
profileNameAndBioPanel.add(profileBio, BorderLayout.CENTE
R);

headerPanel.add(profileNameAndBioPanel);



        return headerPanel;

    }


   private void handleFollowAction(String usernameToFollow)
{
    Path followingFilePath = Paths.get("data", "following.t
xt");
    Path usersFilePath = Paths.get("data", "users.txt");
    String currentUserUsername = "";

    try {
        // Read the current user's username from users.txt
        try (BufferedReader reader = Files.newBufferedReade
r(usersFilePath)) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] parts = line.split(":");
               currentUserUsername = parts[0];
            }
        }

        System.out.println("Real user is "+currentUserUsern
ame);
        // If currentUserUsername is not empty, process fol
lowing.txt
        if (!currentUserUsername.isEmpty()) {
```

```
            boolean found = false;
            StringBuilder newContent = new StringBuilder();

            // Read and process following.txt
            if (Files.exists(followingFilePath)) {
                try (BufferedReader reader = Files.newBuffe
redReader(followingFilePath)) {
                    String line;
                    while ((line = reader.readLine()) != nu
ll) {
                        String[] parts = line.split(":");
                        if (parts[0].trim().equals(currentU
serUsername)) {
                            found = true;
                            if (!line.contains(usernameToFo
llow)) {
                                line = line.concat(line.end
sWith(":") ? "" : "; ").concat(usernameToFollow);
                            }
                        }
                        newContent.append(line).append
("\n");
                    }
                }
            }

            // If the current user was not found in followi
ng.txt, add them
            if (!found) {
                newContent.append(currentUserUsername).appe
nd(": ").append(usernameToFollow).append("\n");
            }

            // Write the updated content back to following.
txt
            try (BufferedWriter writer = Files.newBufferedW
riter(followingFilePath)) {
                writer.write(newContent.toString());
```

```
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}




    private JPanel createNavigationPanel() {
        // Navigation Bar
        JPanel navigationPanel = new JPanel();
        navigationPanel.setBackground(new Color(249, 249, 2
49));
        navigationPanel.setLayout(new BoxLayout(navigationP
anel, BoxLayout.X_AXIS));
        navigationPanel.setBorder(BorderFactory.createEmpty
Border(5, 5, 5, 5));

        navigationPanel.add(createIconButton("img/icons/hom
e.png", "home"));
        navigationPanel.add(Box.createHorizontalGlue());
        navigationPanel.add(createIconButton("img/icons/sea
rch.png","explore"));
        navigationPanel.add(Box.createHorizontalGlue());
        navigationPanel.add(createIconButton("img/icons/ad
d.png","add"));
        navigationPanel.add(Box.createHorizontalGlue());
        navigationPanel.add(createIconButton("img/icons/hea
rt.png","notification"));
        navigationPanel.add(Box.createHorizontalGlue());
        navigationPanel.add(createIconButton("img/icons/pro
file.png", "profile"));

        return navigationPanel;
```

```
    }

private void initializeImageGrid() {
    contentPanel.removeAll(); // Clear existing content
    contentPanel.setLayout(new GridLayout(0, 3, 5, 5)); //
Grid layout for image grid

    Path imageDir = Paths.get("img", "uploaded");
    try (Stream<Path> paths = Files.list(imageDir)) {
        paths.filter(path -> path.getFileName().toString().
startsWith(currentUser.getUsername() + "_"))
             .forEach(path -> {
                 ImageIcon imageIcon = new ImageIcon(new Im
ageIcon(path.toString()).getImage().getScaledInstance(GRID_
IMAGE_SIZE, GRID_IMAGE_SIZE, Image.SCALE_SMOOTH));
                 JLabel imageLabel = new JLabel(imageIcon);
                 imageLabel.addMouseListener(new MouseAdapt
er() {
                     @Override
                     public void mouseClicked(MouseEvent e)
{
                         displayImage(imageIcon); // Call m
ethod to display the clicked image
                     }
                 });
                 contentPanel.add(imageLabel);
             });
    } catch (IOException ex) {
        ex.printStackTrace();
        // Handle exception (e.g., show a message or log)
    }

    JScrollPane scrollPane = new JScrollPane(contentPanel);
    scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HOR
IZONTAL_SCROLLBAR_NEVER);
    scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTI
CAL_SCROLLBAR_AS_NEEDED);
```

```
    add(scrollPane, BorderLayout.CENTER); // Add the scroll
pane to the center

    revalidate();
    repaint();
}



    private void displayImage(ImageIcon imageIcon) {
        contentPanel.removeAll(); // Remove existing conten
t
        contentPanel.setLayout(new BorderLayout()); // Chan
ge layout for image display

        JLabel fullSizeImageLabel = new JLabel(imageIcon);
        fullSizeImageLabel.setHorizontalAlignment(JLabel.CE
NTER);
        contentPanel.add(fullSizeImageLabel, BorderLayout.C
ENTER);

        JButton backButton = new JButton("Back");
        backButton.addActionListener(e -> {
            getContentPane().removeAll(); // Remove all com
ponents from the frame
            initializeUI(); // Re-initialize the UI
        });
        contentPanel.add(backButton, BorderLayout.SOUTH);

        revalidate();
        repaint();
    }



    private JLabel createStatLabel(String number, String te
xt) {
        JLabel label = new JLabel("<html><div style='text-a
```

```
lign: center;'>" + number + "<br/>" + text + "</div></html
>", SwingConstants.CENTER);
        label.setFont(new Font("Arial", Font.BOLD, 12));
        label.setForeground(Color.BLACK);
        return label;
    }

    private JButton createIconButton(String iconPath, Strin
g buttonType) {
        ImageIcon iconOriginal = new ImageIcon(iconPath);
        Image iconScaled = iconOriginal.getImage().getScale
dInstance(NAV_ICON_SIZE, NAV_ICON_SIZE, Image.SCALE_SMOOT
H);
        JButton button = new JButton(new ImageIcon(iconScal
ed));
        button.setBorder(BorderFactory.createEmptyBorder
());
        button.setContentAreaFilled(false);

        // Define actions based on button type
        if ("home".equals(buttonType)) {
            button.addActionListener(e -> openHomeUI());
        } else if ("profile".equals(buttonType)) {
            //
        } else if ("notification".equals(buttonType)) {
            button.addActionListener(e -> notificationsUI
());
        } else if ("explore".equals(buttonType)) {
            button.addActionListener(e -> exploreUI());
        } else if ("add".equals(buttonType)) {
            button.addActionListener(e -> ImageUploadUI());
        }
        return button;


    }

    private void ImageUploadUI() {
```

```
        // Open InstagramProfileUI frame
        this.dispose();
        ImageUploadUI upload = new ImageUploadUI();
        upload.setVisible(true);
    }

    private void openProfileUI() {
        // Open InstagramProfileUI frame
        this.dispose();
        InstagramProfileUI profileUI = new InstagramProfile
UI();
        profileUI.setVisible(true);
    }

     private void notificationsUI() {
        // Open InstagramProfileUI frame
        this.dispose();
        NotificationsUI notificationsUI = new Notifications
UI();
        notificationsUI.setVisible(true);
    }

    private void openHomeUI() {
        // Open InstagramProfileUI frame
        this.dispose();
        QuakstagramHomeUI homeUI = new QuakstagramHomeUI();
        homeUI.setVisible(true);
    }

    private void exploreUI() {
        // Open InstagramProfileUI frame
        this.dispose();
        ExploreUI explore = new ExploreUI();
        explore.setVisible(true);
    }


}
```

After

UI has been separated into a Model, View and Controller. Each with a single responsibility

Model is omitted due to being saved in text files and accessed by multiple Database classes

```
public class ProfileController implements IPageController
{
    private final UIManager manager;
    private final ProfilePage page;

    private final int userID;
    private final boolean isOwner;

    public ProfileController(UIManager uiManager, int userI
D, boolean isOwner, boolean isFollowing, List<Post> posts)
    {
        this.manager = uiManager;
        this.userID = userID;
        this.page = new ProfilePage(userID, isOwner, isFoll
owing, id -> handleFollowOrEditAction(id), pageType -> mana
ger.navigateToPage(pageType), posts);
        this.isOwner = isOwner;
    }

    private void handleFollowOrEditAction(int id)
    {
        if (isOwner)
            handleEdit(id);
        else
            handleFollow(id);
    }

    private void handleFollow(int userIdToFollow)
    {
        FollowDBManager.createFollow(userID, userIdToFollo
w);
```

```
        boolean isFollowing = FollowDBManager.isAFollowingB
(manager.getCurrentUserId(), userIdToFollow);
        int followees = FollowDBManager.getFolloweeCount(us
erIdToFollow);

        page.updateEditOrFollowButtonLabel(isFollowing ? "U
nfollow" : "Follow");
        page.updateFolloweeCount(followees); // followers a
ctually retrieved in updateFollowerCount
    }

    private void handleEdit(int userId)
    {

    }

    @Override
    public void open()
    {
        page.setVisible(true);
    }

    @Override
    public void close()
    {
        page.dispose();
    }
}

public class ProfilePage extends JFrame
{
    private static final int WIDTH = 300;
    private static final int HEIGHT = 500;
    private static final int GRID_IMAGE_SIZE = WIDTH / 3;
// Static size for grid images
    private static final int PROFILE_IMAGE_SIZE = 80; // Ad
justed size for the profile image to match UI
```

```
    private final boolean isOwner;
    private final int userId;

    private final IAction<Integer> followAction;
    private final IAction<PageType> navigateAction;

    private final List<Post> posts;

    private JPanel headerPanel;   // Panel for the header
    private JPanel contentPanel; // Panel to display the im
age grid or the clicked image
    private JPanel navigationPanel; // Panel for the naviga
tion

    private JButton editOrFollowButton;
    private JLabel followersLabel;

    private boolean isFollowing;

    public ProfilePage(int userId, boolean isOwner, boolean
isFollowing, IAction<Integer> followAction, IAction<PageTyp
e> navigateAction, List<Post> posts)
    {
        this.userId = userId;
        this.isOwner = isOwner;
        this.isFollowing = isFollowing;

        this.followAction = followAction;
        this.navigateAction = navigateAction;

        this.posts = posts;

        setTitle("DACS Profile");
        setSize(WIDTH, HEIGHT);
        setMinimumSize(new Dimension(WIDTH, HEIGHT));
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new BorderLayout());
```

```
        initializeUI();
    }

    private void initializeUI()
    {
        getContentPane().removeAll(); // Clear existing com
ponents

        headerPanel = createProfileHeader(); // state-depen
dent : either follow button or edit profile button

        /// content grid //TODO : Make selected images full
screen
        JScrollPane contentScrollPane = UIElementFactory.cr
eateImageGridPanel(GRID_IMAGE_SIZE, posts, (post, imageIco
n) -> displayImage(post, imageIcon));
        contentPanel = new JPanel(new BorderLayout());
        contentPanel.add(contentScrollPane, BorderLayout.CE
NTER);
        navigationPanel = UIElementFactory.createNavigation
Panel(this, navigateAction); // state-independent : always
opens the current user profile

        // Re-add the header and navigation panels
        add(headerPanel, BorderLayout.NORTH);
        add(contentPanel, BorderLayout.CENTER);
        add(navigationPanel, BorderLayout.SOUTH);

        revalidate();
        repaint();
    }

    public void displayImage(Post post, ImageIcon imageIco
n)
    {
        contentPanel.removeAll(); // Remove existing conten
t
        contentPanel.setLayout(new BorderLayout()); // Chan
```

```
ge layout for image display

        // Add the full-size image to the panel
        JLabel fullSizeImageLabel = new JLabel(imageIcon);
        fullSizeImageLabel.setHorizontalAlignment(JLabel.CE
NTER);
        contentPanel.add(fullSizeImageLabel, BorderLayout.C
ENTER);

        // Add a back button to return to the grid view
        JButton backButton = new JButton("Back");
        backButton.addActionListener(e ->
        {
            getContentPane().removeAll(); // Remove all com
ponents from the frame
            initializeUI(); // Re-initialize the UI
        });
        contentPanel.add(backButton, BorderLayout.SOUTH);

        revalidate();
        repaint();
    }

    private JPanel createProfileHeader()
    {
        // profile's user account
        String username = UserDBManager.getUsername(userI
d);
        String bio = UserDBManager.getBio(userId);
        int postsCount = PostDBManager.getPostCount(userI
d);
        int followingCount = FollowDBManager.getFollowerCou
nt(userId);
        int followeeCount = FollowDBManager.getFolloweeCoun
t(userId);

        JPanel headerPanel = new JPanel();
        headerPanel.setLayout(new BoxLayout(headerPanel, Bo
```

```
xLayout.Y_AXIS));
        headerPanel.setBackground(Color.GRAY);

        // Top Part of the Header (Profile Image, Stats, Fo
llow Button)
        JPanel topHeaderPanel = new JPanel(new BorderLayout
(10, 0));
        topHeaderPanel.setBackground(new Color(249, 249, 24
9));

        // Profile image
        ImageIcon profileIcon = new ImageIcon(new ImageIcon
(Path.of(Paths.profilePicturesPath.toString(), username +
".png").toString()).getImage().getScaledInstance(PROFILE_IM
AGE_SIZE, PROFILE_IMAGE_SIZE, Image.SCALE_SMOOTH));
        JLabel profileImage = new JLabel(profileIcon);
        profileImage.setBorder(BorderFactory.createEmptyBor
der(10, 10, 10, 10));
        topHeaderPanel.add(profileImage, BorderLayout.WES
T);

        // Stats Panel
        JPanel statsPanel = new JPanel();
        statsPanel.setLayout(new FlowLayout(FlowLayout.CENT
ER, 10, 0));
        statsPanel.setBackground(new Color(249, 249, 249));
        System.out.println("Number of posts for this user "
+ postsCount);
        statsPanel.add(createStatLabel(Integer.toString(pos
tsCount) , "Posts"));
        followersLabel = createStatLabel(Integer.toString(f
olloweeCount), "Followers");
        statsPanel.add(followersLabel);
        statsPanel.add(createStatLabel(Integer.toString(fol
lowingCount), "Following"));
        statsPanel.setBorder(BorderFactory.createEmptyBorde
r(25, 0, 10, 0)); // Add some vertical padding
```

```
        // Follow or Edit Profile Button (depending on the
user)
        editOrFollowButton = new JButton(isOwner ? "Edit Pr
ofile" : isFollowing ? "Unfollow" : "Follow");
        editOrFollowButton.addActionListener(e -> followAct
ion.execute(userId));

        editOrFollowButton.setAlignmentX(Component.CENTER_A
LIGNMENT);
        editOrFollowButton.setFont(new Font("Arial", Font.B
OLD, 12));
        editOrFollowButton.setMaximumSize(new Dimension(Int
eger.MAX_VALUE, editOrFollowButton.getMinimumSize().heigh
t)); // Make the button fill the horizontal space
        editOrFollowButton.setBackground(new Color(225, 22
8, 232)); // A soft, appealing color that complements the U
I
        editOrFollowButton.setForeground(Color.BLACK);
        editOrFollowButton.setOpaque(true);
        editOrFollowButton.setBorderPainted(false);
        editOrFollowButton.setBorder(BorderFactory.createEm
ptyBorder(10, 0, 10, 0)); // Add some vertical padding

        // Add Stats and Follow Button to a combined Panel
        JPanel statsFollowPanel = new JPanel();
        statsFollowPanel.setLayout(new BoxLayout(statsFollo
wPanel, BoxLayout.Y_AXIS));
        statsFollowPanel.add(statsPanel);
        statsFollowPanel.add(editOrFollowButton);
        topHeaderPanel.add(statsFollowPanel, BorderLayout.C
ENTER);

        headerPanel.add(topHeaderPanel);

        // Profile Name and Bio Panel
        JPanel profileNameAndBioPanel = new JPanel();
        profileNameAndBioPanel.setLayout(new BorderLayout
());
```

```
        profileNameAndBioPanel.setBackground(new Color(249,
249, 249));

        JLabel profileNameLabel = new JLabel(username);
        profileNameLabel.setFont(new Font("Arial", Font.BOL
D, 14));
        profileNameLabel.setBorder(BorderFactory.createEmpt
yBorder(10, 10, 0, 10)); // Padding on the sides

        JTextArea profileBio = new JTextArea(bio);
        System.out.println("This is the bio "+username + "
" + userId);
        profileBio.setEditable(false);
        profileBio.setFont(new Font("Arial", Font.PLAIN, 1
2));
        profileBio.setBackground(new Color(249, 249, 249));
        profileBio.setBorder(BorderFactory.createEmptyBorde
r(0, 10, 10, 10)); // Padding on the sides

        profileNameAndBioPanel.add(profileNameLabel, Border
Layout.NORTH);
        profileNameAndBioPanel.add(profileBio, BorderLayou
t.CENTER);

        headerPanel.add(profileNameAndBioPanel);

        return headerPanel;
    }

    private JLabel createStatLabel(String number, String te
xt)
    {
        JLabel label = new JLabel("<html><div style='text-a
lign: center;'>" + number + "<br/>" + text + "</div></html
>", SwingConstants.CENTER);
        label.setFont(new Font("Arial", Font.BOLD, 12));
        label.setForeground(Color.BLACK);
        return label;
```

```
    }

    public void updateEditOrFollowButtonLabel(String strin
g)
    {
        editOrFollowButton.setText(string);
    }

    public void updateFolloweeCount(int followeeCount)
    {
        followersLabel.setText("<html><div style='text-alig
n: center;'>" + followeeCount + "<br/>" + "Followers" + "</
div></html>");
    }
}
```

Rationale

1. **Promotes Single Responsibility principle**
    a. Each class has a single responsibility
2. **Clarity**
    a. Easier to understand the purpose of each class and maintain code readability
3. **Enhances Modularity**
    a. Splitting a large UI class into smaller parts allows for those parts to be reused elsewhere

4. Object-Oriented Design

I have not included additional Design Decision code snippets as they are the same code snippets previously shown

Factory Pattern

Code Smell

Duplicate Code

UI classes contained many **duplicate** methods and fields. Most of the functionality for the duplicate code was the creation of UI elements.

Design Pattern

The Factory pattern was the perfect pattern to solve this issue, since the duplicate code pertained to the creation of UI elements. The creation of those UI elements was then moved into a factory that concerned itself with the creation of each individual UI element.

Justification

Duplicate code reduced maintainability and scalability of the code base, as changing how the application looked required you to modify many classes. Implementing new features would also become more difficult as the feature would have to be implemented multiple times.

The Factory pattern helped extract the duplicate functionality into a single class, which is easy to modify and scale with the application. This also promoted single responsibility by handling the creation of UI elements. Allowing the UI classes to worry about its usage rather than its creation.

MVC Pattern

Code Smell

Bloaters

UI classes were **Bloaters**. Containing many methods, fields and lines of code.

Each UI class contained methods for user interaction, data storage and retrieval, verification and displaying UI elements.

Design Pattern

The Model-View-Controller pattern was used to extract functionality from a single bloater class into smaller classes. These are the model, view and controller classes.

The model contains the data for the specific UI.

The view displays the data and receives user interaction.

The controller formats the data to be displayed and handles user interaction to modify the model and update the view as needed.

Justification

The MVC pattern was used since it was the perfect pattern to decouple data, logic and visuals into individual elements. The pattern helps improve the maintainability and scalability of UI whilst promoting single responsibility between different elements.

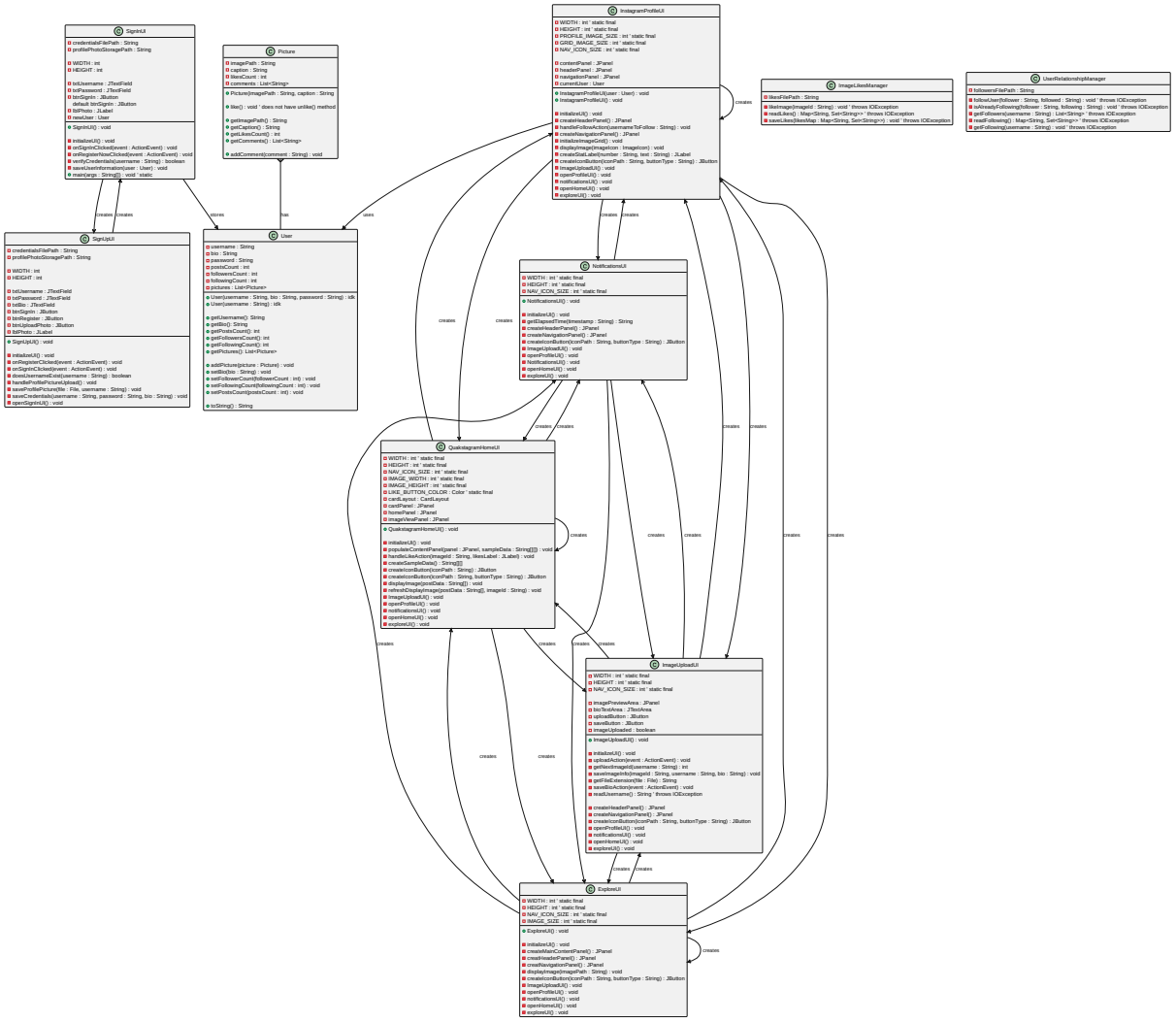
This pattern helped reduce the size of the **Bloater** classes by extracting their functionality into separate classes.

5. Proposal for New Functionality

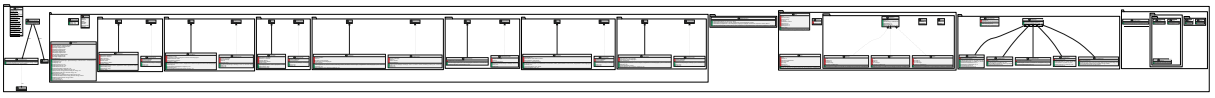
The additional functionality we want to implement is a messaging systems between users. Users would be able to message each other through direct messages or in groups. This system would then provide notifications to users as well employ an admin system. Further functionality we wish to implement is an admin system where admin users have authority to remove posts and moderate users. Messaging groups would then have administrators that would moderate the group.

These new functionalities would have a significant impact on our application, as we would have similar functionality to our competitor's products, allowing us to compete with them. With the use of moderators and admins our app would become safer, which would give us an edge over our competitors as well as provide security to our users.

6. Appendix

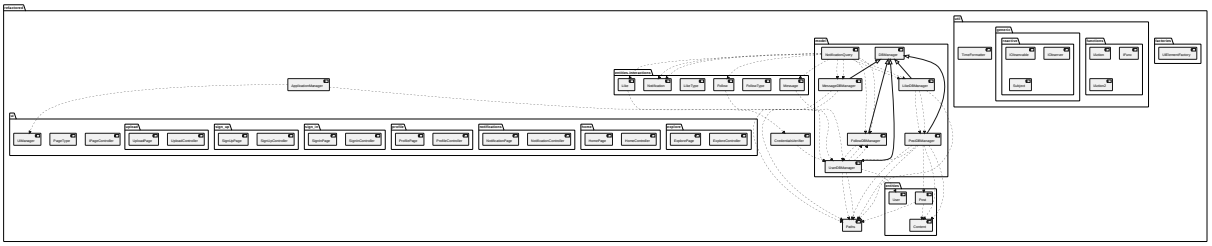


Item A: Initial Code Class Diagram



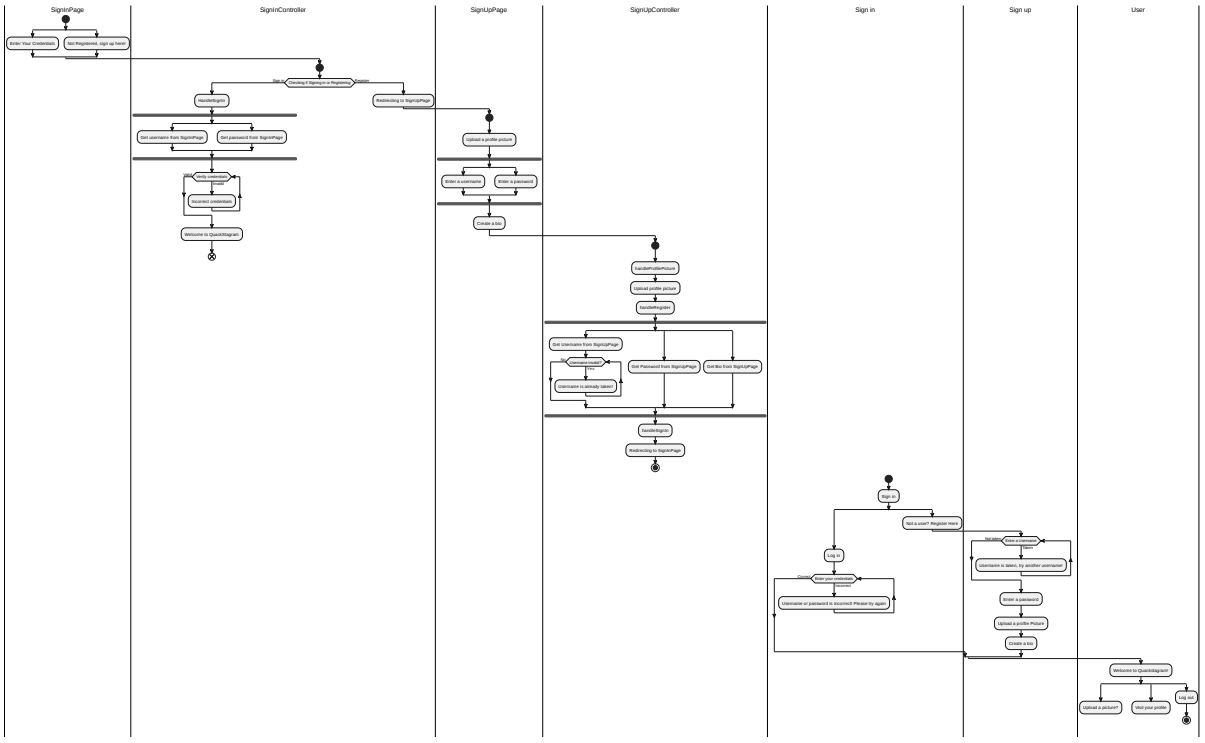
Item B: Refactored Code Class Diagram

For a detailed view, see `class_diagram_refactored.svg` *in* `diagrams`

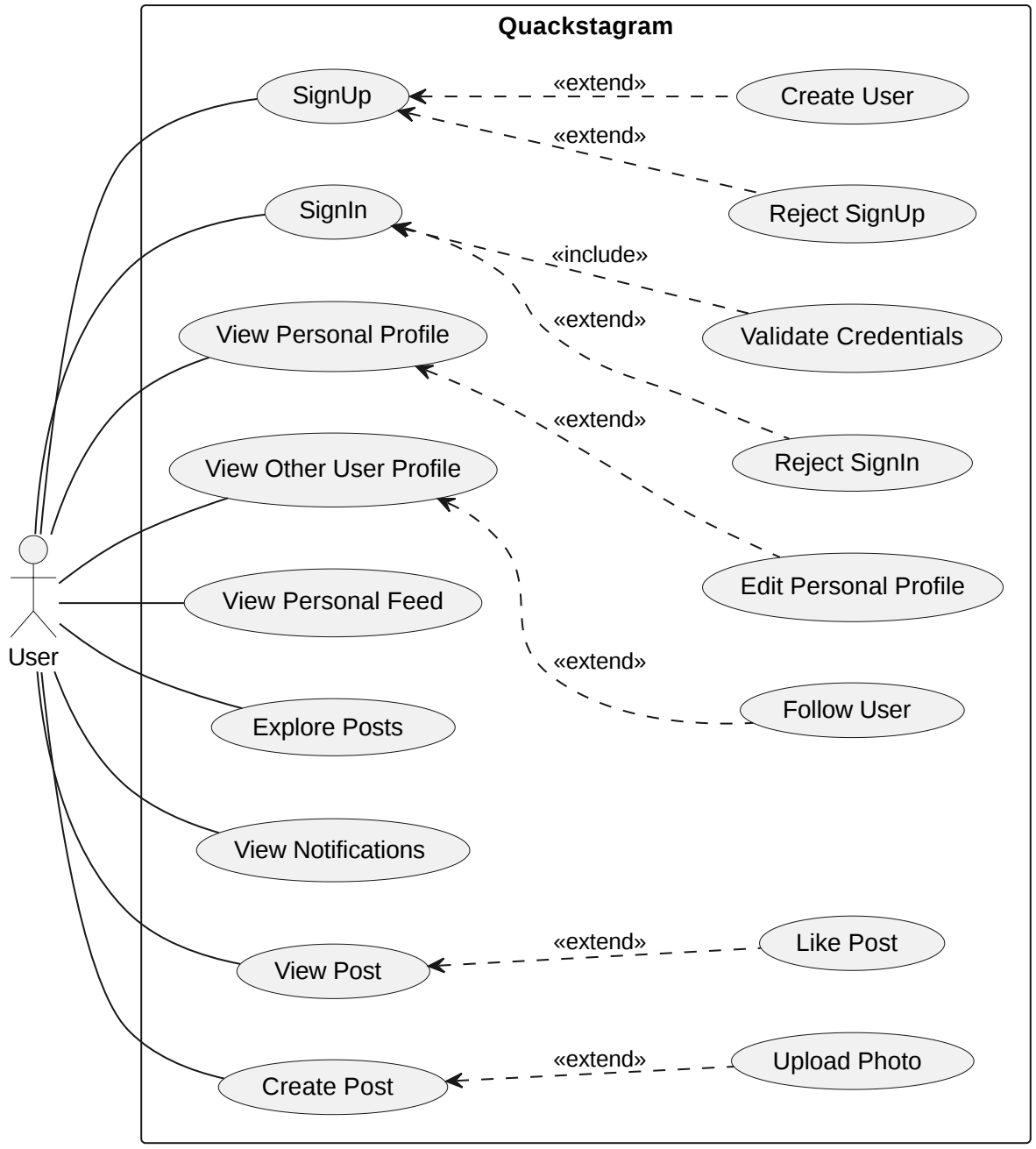


Item C: Refactored Code Package Diagram

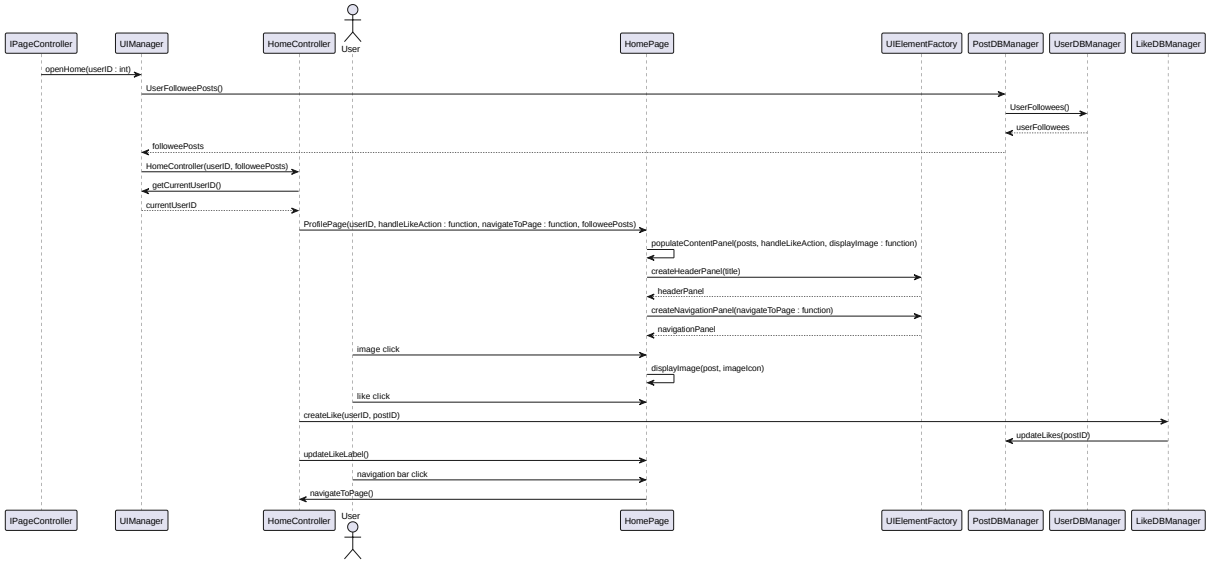
For a detailed view, see `package_diagram_refactored.svg` *in* `diagrams`



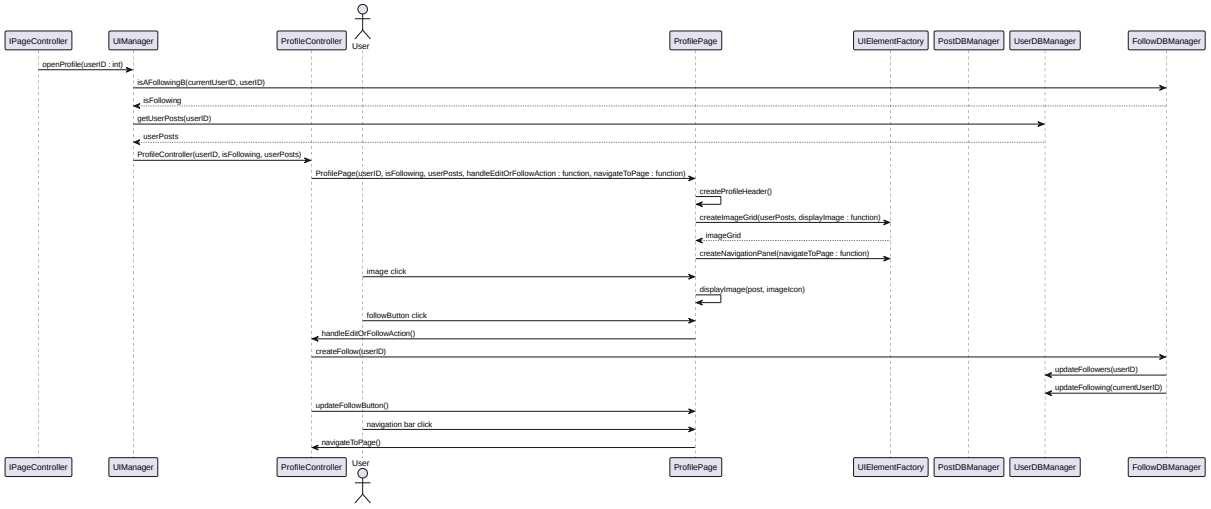
Item D: Activity Diagram



Item E: Use Case Diagram



Item F: Home Sequence Diagram



Item G: Profile Sequence Diagram