Phase 1: Requirements + UML + Refactoring

Group Number: 03, Student Names: Nahdjay Lin, David Wicker, Davide Lacovone, Antek Radowski.

Wednesday, March 6, 2024

Table of contents

1. Requirement Analysis (4%)

1.1 Current System Analysis

User Authentication: Allow users to sign up and log in to the platform.

Profile Management: Enable users to create and personalize their profiles with profile pictures, bios, and other information.

Image Sharing: Allow users to upload images with captions and tags to share moments with others.

Interaction Features: Enable users to like images, follow other users, and engage in comments and conversations.

Explore Feature: Provide a feature for users to discover trending content and new profiles.

Real-Time Notifications: Notify users of new interactions, such as likes and followers, in real-time.

Navigation and Search: Enhance navigation with features like search functionality for finding people and content easily.

1.2 Stakeholder Engagement

Smooth Onboarding Process: Users should have a frustration-free experience during sign-up and sign-in, with clear error indications to guide them in inputting valid credentials.

Flexible Profile Management: Users should be able to customize their profiles even after the initial setup, including the ability to change profile pictures and bios, to enhance personalization and user engagement.

Seamless Interactivity: Interactions such as liking pictures and following other users should be seamless and provide immediate feedback to enhance user engagement. It's essential for the app to be responsive and engaging.

Comments and Moderation: While not an urgent feature, comments can significantly enhance user engagement. However, moderation tools are necessary to ensure a positive environment and foster healthy discussions within the community.

Robust Security Measures: With more interaction features, robust security measures are needed to protect user data and interactions, ensuring a safe online space for users.

Enhanced Navigation and Search: While not an immediate requirement, the addition of a search functionality is something users will eventually expect to find people and content easily, enhancing overall navigation.

Dynamic Notifications: Enhancing notifications beyond just likes, such as notifying users when they have a new follower, can create a more dynamic and connected community, keeping users engaged and connected.

Design Improvements: Even small design improvements can significantly impact the user experience, making the app visually appealing and user-friendly.

User-Centric Development: Throughout the development process, it's crucial to keep the user at the center of every decision and gather feedback from the user community to ensure their voices are heard.

1.3 Requirements Specification

1.3.1 Functional Requirements

User Sign-Up and Login: This functionality enables users to create an account and log in to the application. It serves as the foundation for user interaction with the platform, allowing users to access personalized features and content.

- User registration with a unique username, password, and bio.
- Secure authentication mechanisms for user login.

Profile Management: Users can personalize their profiles by adding profile pictures and bios. Profile customization enhances user engagement and fosters a sense of identity within the platform.

- Uploading profile pictures and bios during sign-up.
- Flexibility for users to change profile pictures.

Image Sharing: Users can upload images with captions and share them with others. Image sharing is the core functionality of the application, allowing users to express themselves and connect with others through visual content.

- Uploading images in various formats.
- Viewing and sharing images with other users.

Interactivity: Users can engage with each other by liking images, following other users, and receiving real-time notifications. Interactivity enhances user engagement and fosters a sense of community within the platform.

- Liking images posted by other users.
- Following other users to receive updates.
- Real-time notifications for new interactions.

1.3.2 Non-Functional Requirements

User Experience: Ensuring a smooth and intuitive user experience throughout the application, including the sign-up/login process and profile customization. A positive user experience is essential for user retention and satisfaction.

- Clear error indications during sign-up/login.
- Flexibility in profile customization.
- Intuitive interface design.

Security: Implementing robust security measures to protect user data and interactions. Security is crucial for maintaining user trust and safeguarding sensitive information.

- Secure storage and management of user data.
- Implementation of encryption and authentication mechanisms.
- Regular security audits and updates.

Performance: Ensuring efficient performance of the application, especially during user interactions and navigation. Performance optimization enhances user satisfaction and platform usability.

- Efficient handling of user interactions (likes, follows).
- Responsive and glitch-free functionality.
- Smooth navigation across different features.

Usability: Enhancing the overall usability of the application through intuitive design and functionality. Usability improvements contribute to user engagement and retention.

- User-friendly interface design.
- Intuitive profile customization options.
- Engaging notifications to keep users connected.

1.4 Prioritization of Requirements

Indicate the priority of the requirements listed above and provide a rationalization for your prioritization. You don't need to justify each requirement individually but may do so for categories of requirements. For example, if privacy is emphasized as crucial in the interview transcripts, you can explain giving high priority to all privacy-related requirements.

Requirement ID	Requirement Description	Priority
REQ-01	Sign-Up and Login Process	High
REQ-02	Profile Flexibility	Medium
REQ-03	Seamless Interactivity	Medium
REQ-04	Commenting and Moderation	Medium
REQ-05	Security Measures	High
REQ-06	Navigation Enhancement	Medium
REQ-07	Notification Enhancement	Medium
REQ-08	Design Improvement	Medium
REQ-09	Performance Optimization	Low
REQ-10	Usability Enhancement	Low

Rationalization for Requirement Prioritization:

REQ-01: Sign-Up and Login Process

Given a high priority due to its critical role as the first interaction point for users. A smooth onboarding experience is crucial for attracting and retaining users, making it essential to prioritize.

REQ-02: Profile Flexibility

Ranked as medium priority since while important for user engagement, it doesn't directly affect the initial user acquisition phase. However, it contributes to user satisfaction and personalization, warranting attention.

REQ-03: Seamless Interactivity

Medium priority as it enhances user engagement, but not as critical as foundational features like sign-up and login. Still important for user retention and satisfaction in the long term.

REQ-04: Commenting and Moderation

Given a medium priority as it fosters community engagement and user interaction, which is crucial for user retention. However, it's not as vital as core functionalities in the initial stages.

REQ-05: Security Measures

Assigned a high priority due to its critical role in protecting user data and maintaining trust. Security is fundamental for user confidence and platform viability.

REQ-06: Navigation Enhancement

Ranked as medium priority as it contributes to overall usability and user experience. While not as critical as core functionalities, it's essential for long-term retention and engagement.

REQ-07: Notification Enhancement

Given a medium priority as it improves user engagement and connectivity. While important, it's not as critical as foundational features but contributes to long-term user satisfaction.

REQ-08: Design Improvement

Medium priority since it enhances the visual appeal and user experience, but may not directly impact functionality or initial user acquisition. Still essential for long-term user engagement.

REQ-09: Performance Optimization

Assigned a low priority as it's important for overall usability but may not be critical in the initial stages of development. Can be addressed iteratively over time for long-term scalability.

REQ-10: Usability Enhancement

Given a low priority as it focuses on improving user experience and accessibility, which may not directly impact initial functionality or user acquisition. Still important for long-term retention and satisfaction.

Legend:

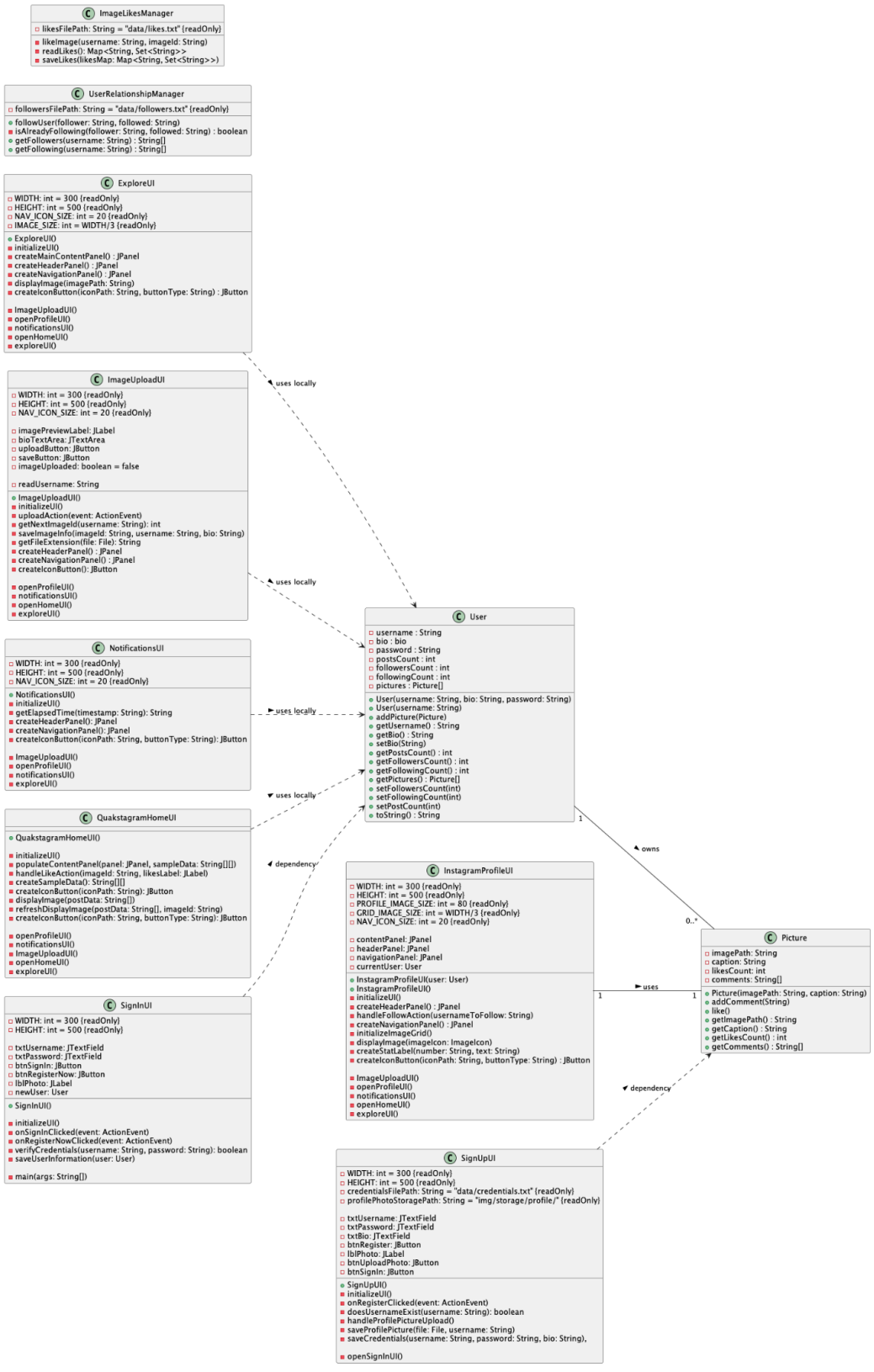
- **High Priority:** Implementing a rainbow color scheme and supporting voice commands are essential for ensuring accessibility and attractiveness of the interface.
- **Medium Priority:** Unique aspects like allowing users to upload cat pictures and generating unicorn avatars, which can significantly improve user satisfaction.
- **Low Priority:** While beneficial, integrating with fictional API XYZ and including a feature to find the nearest coffee shop are considered low priority.

2. UML Modeling (4%)

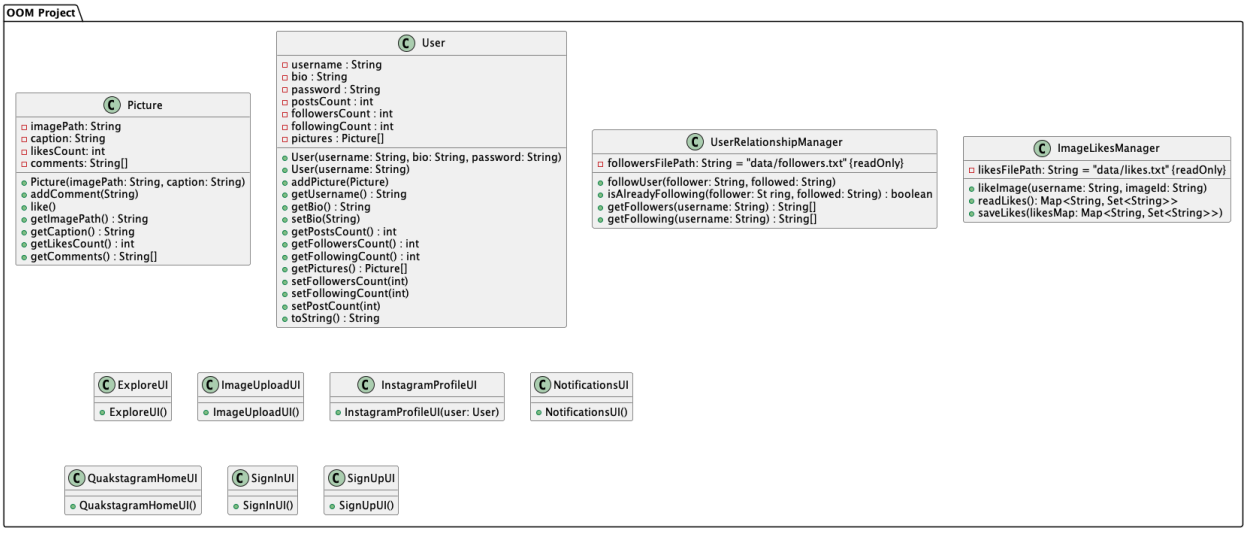
2.1 Current System UML Diagrams

2.1.1 Structural Diagrams

1. Class Diagram *(Navigation interactions are ignored in order to increase clarity)*



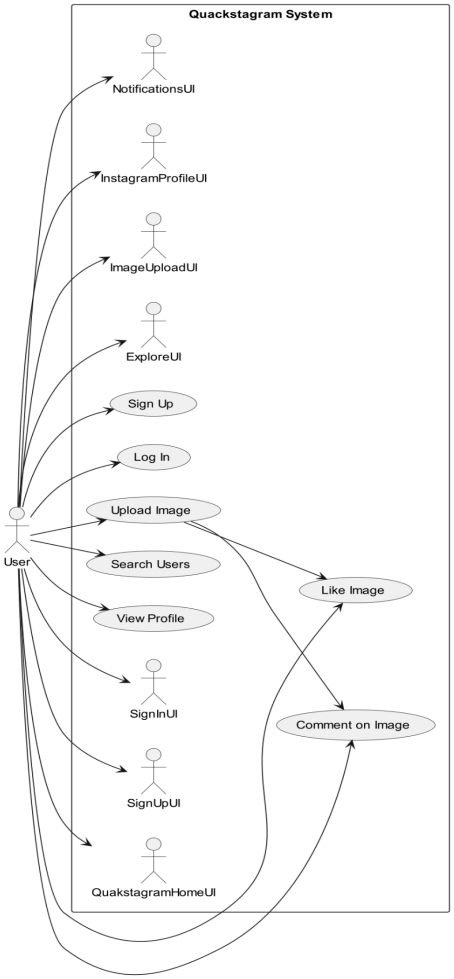
2. Package Diagram



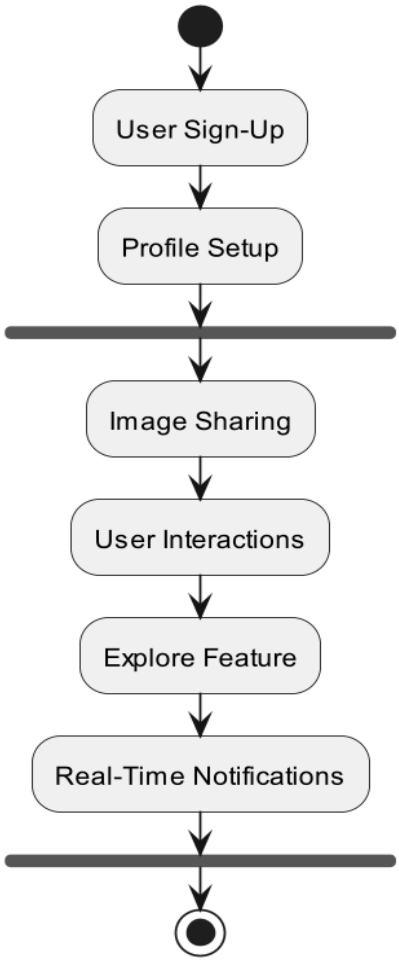
The package diagram highlights the need for compartmentalization and layering between the classes of the project, as the initial code-base does not respect OOP design principles, resulting in duplicate code.

2.1.2 Behavioral Diagrams

1. Diagram 1 (Use Case)

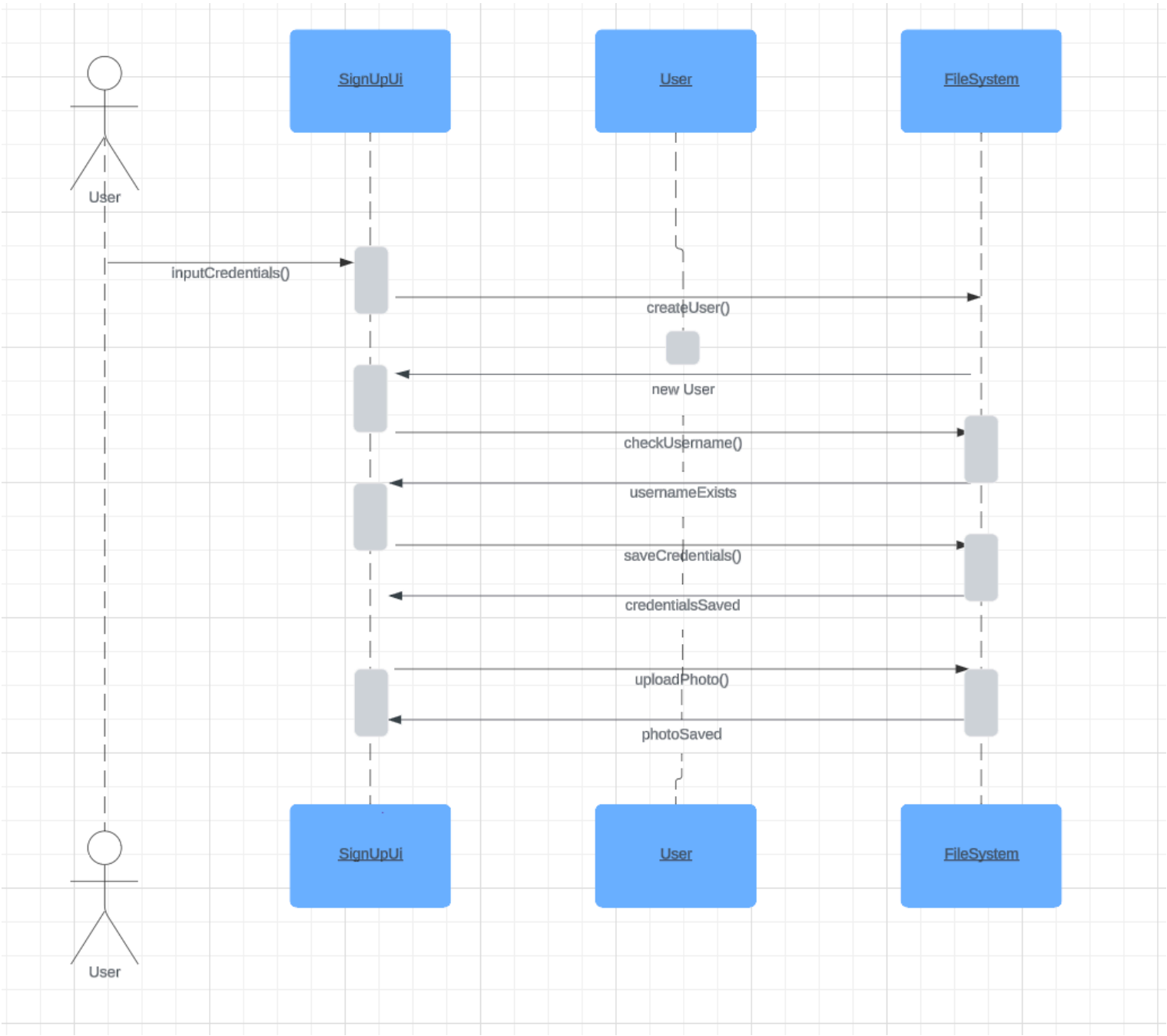


2. Diagram 2 (ActivityDiagram)

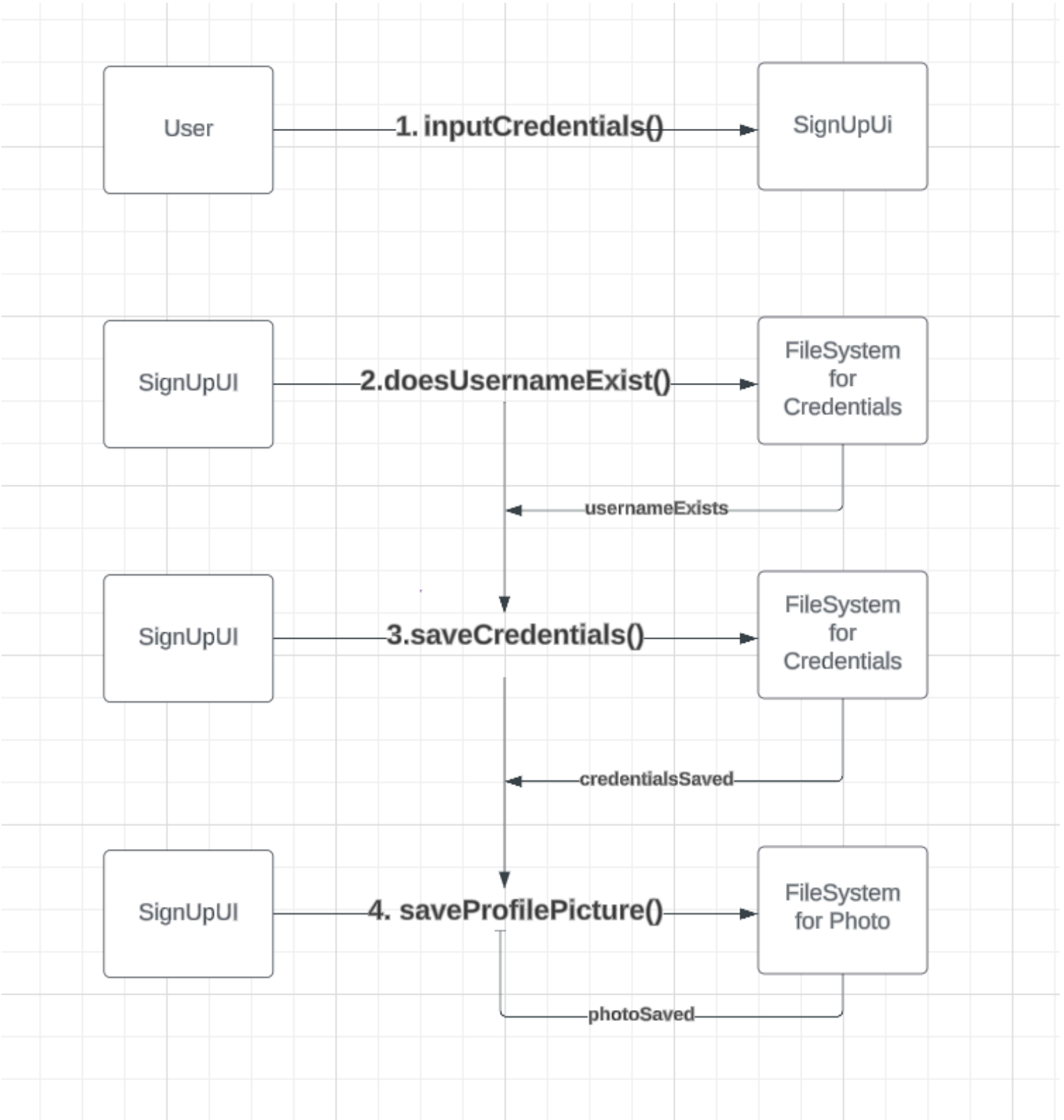


2.1.3 Interaction Diagrams

1. Diagram 1 (Sequence diagram)



2. Diagram 2 (Communication diagram)



2.2 Use Cases

1. Profile Customization: Users can personalize their profiles with a profile picture, bio, and highlights to express their personality.
2. Photo and Video Uploads: The main feature allows users to effortlessly upload and share their visual content with followers.
3. Interactive Storytelling: ThroughQuackstagram Stories, users can share their day-to-day activities in a more dynamic and interactive format.
4. Discovering Content: The Explore page aids users in discovering new content based on their interests and interactions.
5. Engagement through Likes andComments: Users can engage with content by liking photos, videos, and leaving comments, fostering interaction.
6. Direct Messaging: Quackstagram's direct messaging feature enables private conversations, making it easy for users to connect on a more personal level.

3. Refactoring (4%)

ExploreUI

Removed unused imports.

Before refactoring:

```
public ExploreUI() {
        setTitle("Explore");
        setSize(WIDTH, HEIGHT);
        setMinimumSize(new Dimension(WIDTH, HEIGHT));
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new BorderLayout());
        initializeUI();
    }

    private void initializeUI() {

        getContentPane().removeAll(); // Clear existing components
        setLayout(new BorderLayout()); // Reset the layout manager

        JPanel headerPanel = createHeaderPanel(); // Method from your
InstagramProfileUI class
        JPanel navigationPanel = createNavigationPanel(); // Method from your
InstagramProfileUI class
        JPanel mainContentPanel = createMainContentPanel();

        // Add panels to the frame
        add(headerPanel, BorderLayout.NORTH);
        add(mainContentPanel, BorderLayout.CENTER);
        add(navigationPanel, BorderLayout.SOUTH);

        revalidate();
        repaint();


    }
```

After refactoring:

```java
public ExploreUI() {
        configureFrame();
        initializeUI();
    }

    private void configureFrame() {
        setTitle("Explore");
        setSize(WIDTH, HEIGHT);
        setMinimumSize(new Dimension(WIDTH, HEIGHT));
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new BorderLayout());
    }

    private void initializeUI() {
        resetContentPane();
        addPanelsToFrame();
    }

    private void resetContentPane() {
        getContentPane().removeAll();
        setLayout(new BorderLayout());
    }

    private void addPanelsToFrame() {
        add(createHeaderPanel(), BorderLayout.NORTH);
        add(createMainContentPanel(), BorderLayout.CENTER);
        add(createNavigationPanel(), BorderLayout.SOUTH);
        refreshUI();
    }

    private void refreshUI() {
        revalidate();
        repaint();
    }
```

Reasoning:

The refactored code prioritizes the creation of methods for specific functionalities, resulting in improved readability and comprehension of the codebase.

By breaking down complex processes into smaller, modular methods, the refactored code becomes more maintainable. This approach facilitates easier addition of new functionalities and promotes better organization within the codebase, leading to a streamlined development and debugging process. Removal of unused imports from the class further reduces clutter.

ImageUploadUI

Removed not used variables.

Before refactoring:

```
private void initializeUI() {
        JPanel headerPanel = createHeaderPanel(); // Reuse the
createHeaderPanel method
        JPanel navigationPanel = createNavigationPanel(); // Reuse the
createNavigationPanel method

        // Main content panel
        JPanel contentPanel = new JPanel();
        contentPanel.setLayout(new BoxLayout(contentPanel, BoxLayout.Y_AXIS));

        // Image preview
        imagePreviewLabel = new JLabel();
        imagePreviewLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        imagePreviewLabel.setPreferredSize(new Dimension(WIDTH, HEIGHT / 3));

        // Set an initial empty icon to the imagePreviewLabel
        ImageIcon emptyImageIcon = new ImageIcon();
        imagePreviewLabel.setIcon(emptyImageIcon);

        contentPanel.add(imagePreviewLabel);

        // Bio text area
        bioTextArea = new JTextArea("Enter a caption");
        bioTextArea.setAlignmentX(Component.CENTER_ALIGNMENT);
        bioTextArea.setLineWrap(true);
        bioTextArea.setWrapStyleWord(true);
        JScrollPane bioScrollPane = new JScrollPane(bioTextArea);
        bioScrollPane.setPreferredSize(new Dimension(WIDTH - 50, HEIGHT / 6));
        contentPanel.add(bioScrollPane);

        // Upload button
        uploadButton = new JButton("Upload Image");
        uploadButton.setAlignmentX(Component.CENTER_ALIGNMENT);
        uploadButton.addActionListener(this::uploadAction);
        contentPanel.add(uploadButton);

        // Save button (for bio)
        saveButton = new JButton("Save Caption");
        saveButton.setAlignmentX(Component.CENTER_ALIGNMENT);
        saveButton.addActionListener(this::saveBioAction);

        // Add panels to frame
        add(headerPanel, BorderLayout.NORTH);
        add(contentPanel, BorderLayout.CENTER);
        add(navigationPanel, BorderLayout.SOUTH);
    }

    private JPanel createHeaderPanel() {
```

After refactoring:

```
private void initializeUI() {
        JPanel headerPanel = createHeaderPanel();
        JPanel navigationPanel = createNavigationPanel();
        JPanel contentPanel = createContentPanel();

        add(headerPanel, BorderLayout.NORTH);
        add(contentPanel, BorderLayout.CENTER);
        add(navigationPanel, BorderLayout.SOUTH);
    }

    private JPanel createHeaderPanel() {
        JPanel headerPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
        headerPanel.setBackground(new Color(51, 51, 51));
        JLabel lblRegister = new JLabel("Upload Image ");
        lblRegister.setFont(new Font("Arial", Font.BOLD, 16));
        lblRegister.setForeground(Color.WHITE);
        headerPanel.add(lblRegister);
        headerPanel.setPreferredSize(new Dimension(WIDTH, 40));
        return headerPanel;
    }

    private JPanel createContentPanel() {
        JPanel contentPanel = new JPanel();
        contentPanel.setLayout(new BoxLayout(contentPanel, BoxLayout.Y_AXIS));

        imagePreviewLabel = createImagePreviewLabel();
        bioTextArea = createBioTextArea();
        uploadButton = createUploadButton();
        saveButton = createSaveButton();

        contentPanel.add(imagePreviewLabel);
        contentPanel.add(new JScrollPane(bioTextArea));
        contentPanel.add(uploadButton);
        contentPanel.add(saveButton);

        return contentPanel;
    }

    private JLabel createImagePreviewLabel() {
        JLabel label = new JLabel();
        label.setAlignmentX(Component.CENTER_ALIGNMENT);
        label.setPreferredSize(new Dimension(WIDTH, HEIGHT / 3));
        return label;
    }

    private JTextArea createBioTextArea() {
        JTextArea textArea = new JTextArea("Enter a caption");
        textArea.setAlignmentX(Component.CENTER_ALIGNMENT);
        textArea.setLineWrap(true);
        textArea.setWrapStyleWord(true);
        return textArea;
    }

    private JButton createUploadButton() {
        JButton button = new JButton("Upload Image");
        button.setAlignmentX(Component.CENTER_ALIGNMENT);
        button.addActionListener(this::uploadAction);
        return button;
    }

    private JButton createSaveButton() {
        JButton button = new JButton("Save Caption");
        button.setAlignmentX(Component.CENTER_ALIGNMENT);
        button.addActionListener(this::saveBioAction);
        return button;
    }
```

Reasoning: The refactored code prioritizes the creation of methods for specific functionalities, resulting in improved readability and comprehension of the codebase.

By breaking down complex processes into smaller, modular methods, the refactored code becomes more maintainable. This approach facilitates easier addition of new code and promotes better organization within the codebase. Removal of unused variables from the class further enhances code cleanliness.

InstagramProfileUI

Removed unused imports and unused openProfileUI method.

NotificationsUI

Removed unused variable.

UserRelationShipManager

Before refactoring:

```
public void followUser(String follower, String followed) throws IOException {
        if (!isAlreadyFollowing(follower, followed)) {
            try (BufferedWriter writer = new BufferedWriter(new FileWriter(followersFilePath, true))) {
                writer.write(follower + ":" + followed);
                writer.newLine();
            }
        }
    }

    // Method to check if a user is already following another user
    private boolean isAlreadyFollowing(String follower, String followed) throws IOException {
        try (BufferedReader reader = new BufferedReader(new FileReader(followersFilePath))) {
            String line;
            while ((line = reader.readLine()) != null) {
                if (line.equals(follower + ":" + followed)) {
                    return true;
                }
            }
        }
        return false;
    }
```

After refactoring:

```
public void followUser(String follower, String followed) throws IOException {
        if (!isAlreadyFollowing(follower, followed)) {
            appendFollowRelationship(follower, followed);
        }
    }

    // Check if a user is already following another user
    private boolean isAlreadyFollowing(String follower, String followed) throws IOException {
        List<String> followers = getFollowers(followed);
        return followers.contains(follower);
    }
private void appendFollowRelationship(String follower, String followed) throws IOException {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(followersFilePath, true))) {
            writer.write(follower + ":" + followed);
            writer.newLine();
        }
    }
```

Reasoning: isAlreadyFollowing - simplified by reusing getFollowers

appendFollowRelationship - extracted the logic to append a follow relationship to a new method, improving followUser readability.

SignUpUI

Removed unused imports.

Before refactoring:

```
public SignUpUI() {
        setTitle("Quackstagram - Register");
        setSize(WIDTH, HEIGHT);
        setMinimumSize(new Dimension(WIDTH, HEIGHT));
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new BorderLayout(10, 10));
        initializeUI();
    }

    private void initializeUI() {
        // Header with the Register label
        JPanel headerPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
        headerPanel.setBackground(new Color(51, 51, 51)); // Set a darker background for the header
        JLabel lblRegister = new JLabel("Quackstagram ");
        lblRegister.setFont(new Font("Arial", Font.BOLD, 16));
        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

        // Profile picture placeholder without border
        lblPhoto = new JLabel();
        lblPhoto.setPreferredSize(new Dimension(80, 80));
        lblPhoto.setHorizontalAlignment(JLabel.CENTER);
        lblPhoto.setVerticalAlignment(JLabel.CENTER);
        lblPhoto.setIcon(new ImageIcon(new ImageIcon("img/logos/DACS.png").getImage().getScaledInstance(80, 80, Image.SCALE_SMOOTH)));
        JPanel photoPanel = new JPanel(); // Use a panel to center the photo label
        photoPanel.setLayout(new FlowLayout(FlowLayout.CENTER));
        photoPanel.add(lblPhoto);
```

```
// Text fields panel
JPanel fieldsPanel = new JPanel();
fieldsPanel.setLayout(new BoxLayout(fieldsPanel, BoxLayout.Y_AXIS));
fieldsPanel.setBorder(BorderFactory.createEmptyBorder(5, 20, 5, 20));

txtUsername = new JTextField("Username");
txtPassword = new JTextField("Password");
txtBio = new JTextField("Bio");
txtBio.setForeground(Color.GRAY);
txtUsername.setForeground(Color.GRAY);
txtPassword.setForeground(Color.GRAY);

fieldsPanel.add(Box.createVerticalStrut(10));
fieldsPanel.add(photoPanel);
fieldsPanel.add(Box.createVerticalStrut(10));
fieldsPanel.add(txtUsername);
fieldsPanel.add(Box.createVerticalStrut(10));
fieldsPanel.add(txtPassword);
fieldsPanel.add(Box.createVerticalStrut(10));
fieldsPanel.add(txtBio);
btnUploadPhoto = new JButton("Upload Photo");

btnUploadPhoto.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        handleProfilePictureUpload();
    }
});
JPanel photoUploadPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
photoUploadPanel.add(btnUploadPhoto);
fieldsPanel.add(photoUploadPanel);
```

```
    // Register button with black text
    btnRegister = new JButton("Register");
    btnRegister.addActionListener(this::onRegisterClicked);
    btnRegister.setBackground(new Color(255, 90, 95)); // Use a red color that matches the mockup
    btnRegister.setForeground(Color.BLACK); // Set the text color to black
    btnRegister.setFocusPainted(false);
    btnRegister.setBorderPainted(false);
    btnRegister.setFont(new Font("Arial", Font.BOLD, 14));
    JPanel registerPanel = new JPanel(new BorderLayout()); // Panel to contain the register button
    registerPanel.setBackground(Color.WHITE); // Background for the panel
    registerPanel.add(btnRegister, BorderLayout.CENTER);



    // Adding components to the frame
    add(headerPanel, BorderLayout.NORTH);
    add(fieldsPanel, BorderLayout.CENTER);
    add(registerPanel, BorderLayout.SOUTH);
     // Adding the sign in button to the register panel or another suitable panel
    btnSignIn = new JButton("Already have an account? Sign In");
    btnSignIn.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            openSignInUI();
        }
    });
    registerPanel.add(btnSignIn, BorderLayout.SOUTH);
}
```

```
private void onRegisterClicked(ActionEvent event) {
    String username = txtUsername.getText();
    String password = txtPassword.getText();
    String bio = txtBio.getText();

    if (doesUsernameExist(username)) {
        JOptionPane.showMessageDialog(this, "Username already exists. Please choose a different username.", "Error", JOptionPane.ERROR_MESSAGE);
    } else {
        saveCredentials(username, password, bio);
        handleProfilePictureUpload();
        dispose();

    // Open the SignInUI frame
    SwingUtilities.invokeLater(() -> {
        SignInUI signInFrame = new SignInUI();
        signInFrame.setVisible(true);
    });
    }
}
```

After refactoring:

```
public SignUpUI() {
        initializeFrame();
        initializeUIComponents();
        layoutComponents();
    }

    private void initializeFrame() {
        setTitle("Quackstagram - Register");
        setSize(WIDTH, HEIGHT);
        setMinimumSize(new Dimension(WIDTH, HEIGHT));
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new BorderLayout(10, 10));
    }

    private void initializeUIComponents() {
        lblPhoto = createPhotoLabel();
        txtUsername = createTextField("Username");
        txtPassword = createTextField("Password");
        txtBio = createTextField("Bio");
        btnUploadPhoto = createButton("Upload Photo", this::handleProfilePictureUpload);
        btnRegister = createButton("Register", this::onRegisterClicked);
        btnSignIn = createButton("Already have an account? Sign In", this::openSignInUI);
    }

    private JLabel createPhotoLabel() {
        JLabel photoLabel = new JLabel();
        photoLabel.setPreferredSize(new Dimension(80, 80));
        photoLabel.setHorizontalAlignment(JLabel.CENTER);
        photoLabel.setVerticalAlignment(JLabel.CENTER);
        photoLabel.setIcon(new ImageIcon(
                new ImageIcon("img/logos/DACS.png").getImage().getScaledInstance(80, 80, Image.SCALE_SMOOTH)));
        return photoLabel;
    }
```

```
private JTextField createTextField(String placeholder) {
    JTextField textField = new JTextField(placeholder);
    textField.setForeground(Color.GRAY);
    return textField;
}

private JButton createButton(String text, ActionListener listener) {
    JButton button = new JButton(text);
    button.addActionListener(listener);
    return button;
}

private void layoutComponents() {
    add(createHeaderPanel(), BorderLayout.NORTH);
    add(createFieldsPanel(), BorderLayout.CENTER);
    add(createRegisterPanel(), BorderLayout.SOUTH);
}

private JPanel createHeaderPanel() {
    JPanel headerPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
    headerPanel.setBackground(new Color(51, 51, 51));
    JLabel lblRegister = new JLabel("Quackstagram 🦆");
    lblRegister.setFont(new Font("Arial", Font.BOLD, 16));
    lblRegister.setForeground(Color.WHITE);
    headerPanel.add(lblRegister);
    headerPanel.setPreferredSize(new Dimension(WIDTH, 40));
    return headerPanel;
}
```

```
private JPanel createFieldsPanel() {
    JPanel fieldsPanel = new JPanel();
    fieldsPanel.setLayout(new BoxLayout(fieldsPanel, BoxLayout.Y_AXIS));
    fieldsPanel.setBorder(BorderFactory.createEmptyBorder(5, 20, 5, 20));

    fieldsPanel.add(Box.createVerticalStrut(10));
    fieldsPanel.add(createPhotoPanel());
    fieldsPanel.add(Box.createVerticalStrut(10));
    fieldsPanel.add(txtUsername);
    fieldsPanel.add(Box.createVerticalStrut(10));
    fieldsPanel.add(txtPassword);
    fieldsPanel.add(Box.createVerticalStrut(10));
    fieldsPanel.add(txtBio);
    fieldsPanel.add(createPhotoUploadPanel());

    return fieldsPanel;
}

private JPanel createPhotoPanel() {
    JPanel photoPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
    photoPanel.add(lblPhoto);
    return photoPanel;
}

private JPanel createPhotoUploadPanel() {
    JPanel photoUploadPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
    photoUploadPanel.add(btnUploadPhoto);
    return photoUploadPanel;
}
```

```
private JPanel createRegisterPanel() {
    JPanel registerPanel = new JPanel(new BorderLayout());
    registerPanel.setBackground(Color.WHITE);
    registerPanel.add(btnRegister, BorderLayout.CENTER);
    registerPanel.add(btnSignIn, BorderLayout.SOUTH);
    return registerPanel;
}

private void onRegisterClicked(ActionEvent event) {
    String username = txtUsername.getText();
    String password = txtPassword.getText();
    String bio = txtBio.getText();

    if (doesUsernameExist(username)) {
        JOptionPane.showMessageDialog(this, "Username already exists. Please choose a different username.", "Error",
                JOptionPane.ERROR_MESSAGE);
    } else {
        saveCredentials(username, password, bio);
        handleProfilePictureUpload(null);
        dispose();
        openSignInUI(null);
    }
}
```

Reasoning:

The original method was complex, so we decomposed it into smaller, more manageable components, improving code readability and comprehension.

Breaking down the monolithic method makes the code easier to understand and maintain, with each method serving a clear purpose.

Smaller, well-defined methods enhance maintainability by facilitating easier debugging, modification, and addition of new features.

QuackstagramHomeUI

Removed unused method createIconButton.

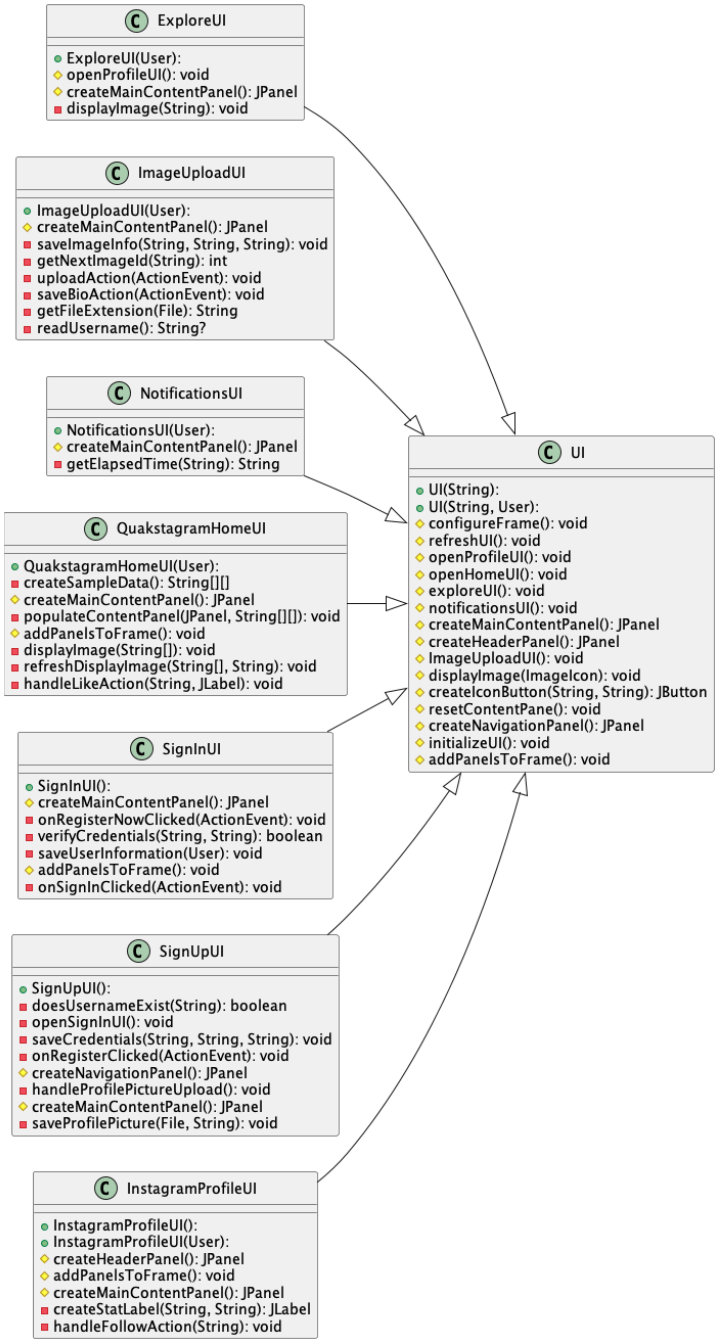
4. Object-Oriented Design (4%)

The main issue with the provided code-base with regards to OOP Principles is the absence of relationships between the numerous UI classes.

All of the UI classes, despite being structured slightly differently due to a lack of consistency, contain common key components and functionalities, which could be extracted into a parent abstract class, as seen in the updated Class Diagram – such a diagram merely shows the relationship between an abstract UI class and the various UI classes that would extend it.

As shown in the code snippet below, an abstract UI class could extend Swing's JFrame, since all of the UI Classes inherit from such a class, and could incorporate all common constants and utility methods.

```
13  public abstract class UI extends JFrame {
14      protected static final int WIDTH = 300;
15      protected static final int HEIGHT = 500;
16      protected static final int NAV_ICON_SIZE = 20;
17      protected static final int IMAGE_SIZE = WIDTH / 3;
18
19      protected JPanel contentPanel;
20      protected JPanel headerPanel;
21      protected JPanel navigationPanel;
22
23      protected String title;
24      protected User loggedInUser;
25
26      public UI(String title) {
27          this.title = title;
28          configureFrame();
29          initializeUI();
30      }
31
32      public UI(String title, User user) {
33          this.title = title;
34          this.loggedInUser = user;
35          configureFrame();
36          initializeUI();
37      }
```



We decided to add a protected "loggedInUser" variable of type **User,** this variable can be initialized through the constructor of the abstract UI class. This allows for every piece of the UI to have access to the information of the currently logged in user, which may be useful for future features.

```
39      // Draw content methods
40      protected abstract JPanel createMainContentPanel();
41
42  >   protected JPanel createHeaderPanel() {…
54
55  >   protected JPanel createNavigationPanel() {…
75
76      // UI Utility methods
77      protected void configureFrame() {
78          setTitle(this.title);
79          setSize(WIDTH, HEIGHT);
80          setMinimumSize(new Dimension(WIDTH, HEIGHT));
81          setDefaultCloseOperation(EXIT_ON_CLOSE);
82          setLayout(new BorderLayout());
83      }
84
85      protected void initializeUI() {
86          resetContentPane();
87          addPanelsToFrame();
88      }
89
90      protected void resetContentPane() {
91          getContentPane().removeAll();
92          setLayout(new BorderLayout());
93      }
94
95      protected void addPanelsToFrame() {
96          this.headerPanel = createHeaderPanel();
97          this.contentPanel = createMainContentPanel();
98          this.navigationPanel = createNavigationPanel();
99
100         add(headerPanel, BorderLayout.NORTH);
101         add(contentPanel, BorderLayout.CENTER);
102         add(navigationPanel, BorderLayout.SOUTH);
103         refreshUI();
104     }
105
106     protected void refreshUI() {
107         revalidate();
108         repaint();
109     }
```

As shown above, the abstract UI class would also standardize the layout of the pages of the application, distinguishing between a Header Panel, a Navigation Panel and a Main Content Panel, all with their respective draw functions which all return **JPanel** objects. Such methods would be called from an **addPanelsToFrame()** function and displayed correctly according to the desired layout. This implementation also recalls the [refactoring showcased in this report.]

```
154     // Navigation methods
155 >   protected void ImageUploadUI() {…
161
162 >   protected void openProfileUI() {…
180
181 >   protected void notificationsUI() {…
187
188 >   protected void openHomeUI() {…
194
195 >   protected void exploreUI() {…
201 }
```

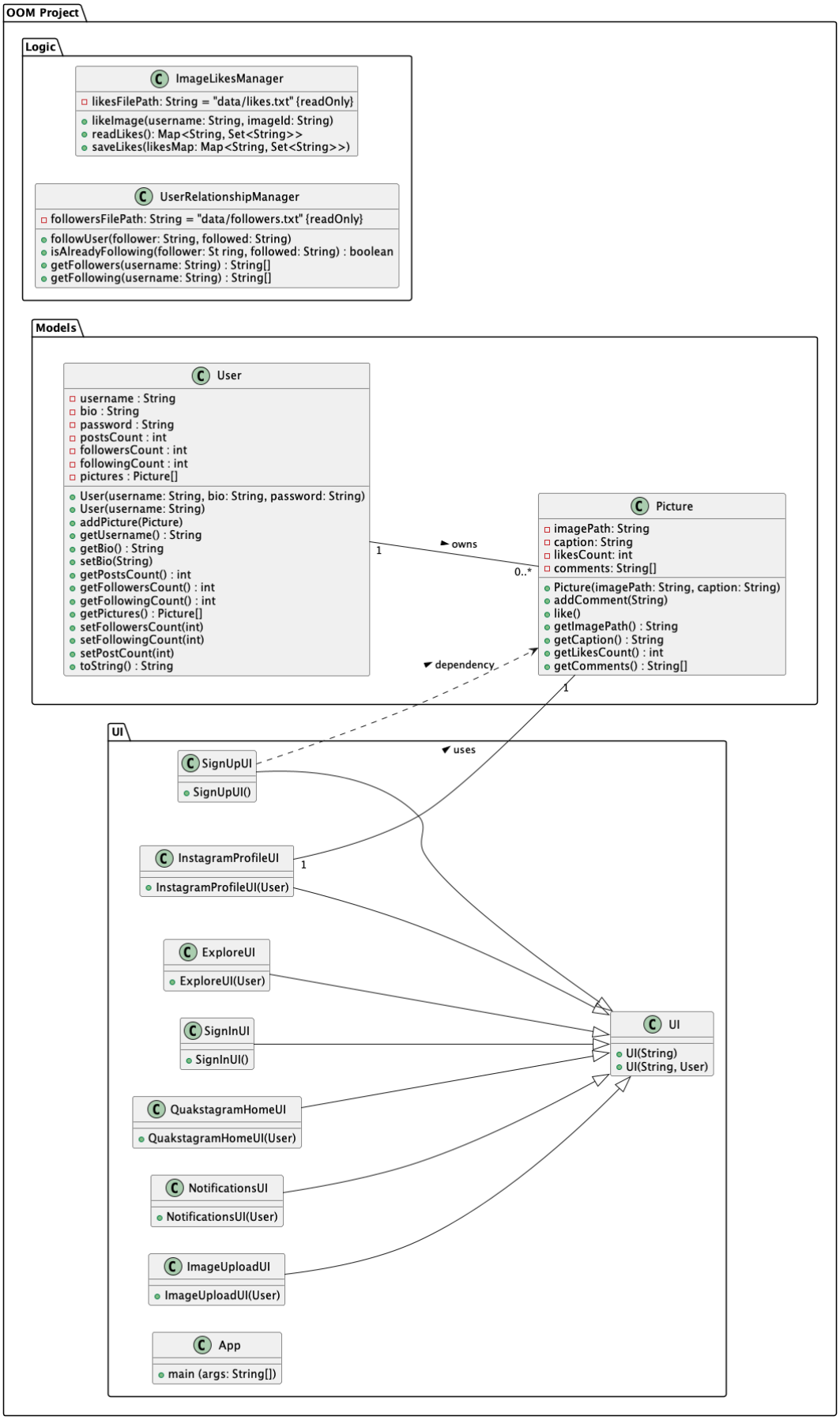
Furthermore, 5 common navigation functions have been identified and moved to the UI class. These are called from the child classes when navigating from page to page. In one instance, the **openProfileUI** method is overridden by the child class **ExploreUI.**

```
22  public class ExploreUI extends UI {
23
24      private static final int IMAGE_SIZE = WIDTH / 3;
25
26      public ExploreUI(User user) {
27          super(title:"Explore", user);
28      }
29
30      // Override draw content methods
31      @Override
32  >   protected JPanel createMainContentPanel() {…
76
77      // Utility methods
78  >   private void displayImage(String imagePath) {…
198
199     // Override navigation methods
200     @Override
201     protected void openProfileUI() {
202         // Open InstagramProfileUI frame
203         this.dispose();
204         InstagramProfileUI profile = new InstagramProfileUI();
205         profile.setVisible(b:true);
206     }
207 }
208
```

As an example implementation of a child class of the UI abstract class, we can consider the ExploreUI class, shown below.

The ExploreUI class overrides the method **createMainContentPanel** of the parent class in order to display the content relevant to this specific page.

Finally, the refactoring also included the separation of the classes into different packages, to better handle code readability and to respect OOP principles, as seen in the Package Diagram below.


OOM Project
Logic
ImageLikesManager
- likesFilePath: String = "data/likes.txt" {readOnly}
+ likeImage(username: String, imageId: String)
+ readLikes(): Map<String, Set<String>>
+ saveLikes(likesMap: Map<String, Set<String>>)
UserRelationshipManager
- followersFilePath: String = "data/followers.txt" {readOnly}
+ followUser(follower: String, followed: String)
+ isAlreadyFollowing(follower: St ring, followed: String) : boolean
+ getFollowers(username: String) : String[]
+ getFollowing(username: String) : String[]
Models
User
- username : String
- bio : String
- password : String
- postsCount : int
- followersCount : int
- followingCount : int
- pictures : Picture[]
+ User(username: String, bio: String, password: String)
+ User(username: String)
+ addPicture(Picture)
+ getUsername() : String
+ getBio() : String
+ setBio(String)
+ getPostsCount() : int
+ getFollowersCount() : int
+ getFollowingCount() : int
+ getPictures() : Picture[]
+ setFollowersCount(int)
+ setFollowingCount(int)
+ setPostCount(int)
+ toString() : String
owns
1
0..*
Picture
- imagePath: String
- caption: String
- likesCount: int
- comments: String[]
+ Picture(imagePath: String, caption: String)
+ addComment(String)
+ like()
+ getImagePath() : String
+ getCaption() : String
+ getLikesCount() : int
+ getComments() : String[]
dependency
1
UI
uses
SignUpUI
+ SignUpUI()
InstagramProfileUI
+ InstagramProfileUI(User)
1
ExploreUI
+ ExploreUI(User)
SignInUI
+ SignInUI()
QuakstagramHomeUI
+ QuakstagramHomeUI(User)
NotificationsUI
+ NotificationsUI(User)
ImageUploadUI
+ ImageUploadUI(User)
App
+ main (args: String[])
UI
+ UI(String)
+ UI(String, User)

5. Proposal for New Functionality (Not graded)

New Functionality Proposal: Database Interface Class

What: Introduce a Database class acting as an interface to centralize data access operations in Quackstagram. This class will abstract data manipulation tasks, simplifying future transitions to alternative storage solutions such as SQL databases.

Why: The Database interface addresses current limitations in data management by providing a unified access point for data operations. This enhances maintainability and prepares the system for scalability and security improvements.

Expected Impact: Implementing the Database interface class will streamline data access and manipulation processes, improving code readability and maintainability. Additionally, it lays the foundation for future security enhancements and scalability improvements, ensuring the long-term viability of the Quackstagram platform.

New Functionality Proposal: Edit Profile Page

What: Introduce the ability for users to edit their profiles within Quackstagram. Currently, this feature is absent from the platform, limiting user customization options.

Why: The Edit Profile Page functionality directly addresses user expectations and demands for personalization within the platform. By enabling users to modify their profiles, Quackstagram enhances user engagement and satisfaction.

Expected Impact: The addition of the Edit Profile Page feature will significantly enhance the user experience by empowering users to personalize their profiles according to their preferences. This increased flexibility and customization options are likely to boost user retention and overall platform usage.