Workshops of the Object-Oriented Programming project

Julián Carvajal Garnica

20242020024

Andrés Mauricio Cepeda Villanueva

20242020010

Faculty of Engineering, Universidad Distrital Francisco José de Caldas

Object Oriented Programming

Carlos Andrés Sierra

Bogotá D.C.

2025

INTRODUCTION:

In a world where daily interaction with technological tools is increasingly common, the development of applications that facilitate communication between users has become both feasible and relevant. With this in mind, we propose the creation of a dating application inspired by Tinder, aiming to promote affective connections between individuals through digital interaction. The project is developed under the principles of Object-Oriented Programming (OOP), using Java as the main language. The application includes essential features such as user profile creation, an interaction system (likes, matches), and a notification mechanism, all structured through a modular and scalable object-oriented design.

The final version of the project is expected to be a functional and modular dating application that demonstrates the correct application of object-oriented principles. It should allow users to register, customize their profiles, interact with others through likes or dismisses, and receive notifications in response to those interactions. Additionally, the codebase should be clean, reusable, and scalable, serving as a solid foundation for future extensions or improvements.

OBJETIVES:

General Objectives

To build a functional prototype of a dating application inspired by Tinder, using Java and Object-Oriented Programming (OOP) principles, aiming to promote secure and engaging user interactions through clean, modular, and maintainable architecture.

Specific Objectives:

- To implement a user authentication system that allows account creation and secure login, ensuring controlled access to the application.
- To enable users to customize their profiles by editing personal data and uploading photos, enhancing personalization and user identity.
- To develop an interaction and match mechanism that lets users like or dismiss profiles and receive notifications when mutual interest occurs, fostering engagement.
- To design the application architecture using OOP principles such as encapsulation and modularity, supported by UML diagrams that document system structure and behavior.

Functional Requirements:

- Users must be able to register, log in, and edit their profile. This includes uploading a photo, writing a bio, selecting interests, and modifying personal information.

- Users must be able to view other profiles and perform actions such as "Like" or "Dismiss."
- When a user expresses interest (Like), the other user must receive a notification so they are informed and can potentially view the profile, indicating whether mutual interest exists.

Non-Functional Requirements:

- The application must have a clear, intuitive, and easy-to-navigate interface, accessible from various devices.
- The system must implement security mechanisms such as account verification and protection against common threats, ensuring user safety.
- The design must be modern, aesthetically pleasing, and consistent, using a color palette and typography that promote a comfortable visual experience.
- The application must maintain fast response times and support progressive user growth without compromising the user experience or system stability.

USER STORIES:

USER STORY

Title: User registration	**Priority:** High	**Estimate: 1 week**

User Story:
As a new user,
I want to register using my email and password,
so that I can access the application.

Acceptance Criteria:
Given that a new user opens the registration interface,
When they enter a valid email and password and press the registration button,
Then their account is successfully created, and a confirmation message is displayed.

USER STORY

Title: Complete new user information	**Priority:** High	**Estimate: 1 week**

User Story:
As a newly registered user,
I want to enter personal information such as my name, birthday, gender, sexual orientation, interests, and lifestyle,
so that I can complete my profile and start using the app properly.

Acceptance Criteria:
Given that a user has just completed the registration form
When they access the app for the first time
Then they are redirected to a profile completion screen where they can enter their name, date of birth, gender, sexual orientation, interests, and lifestyle before being able to interact with other users.

USER STORY

Title: Registered user	**Priority:** High	**Estimate: 1 week**

User Story:
As a registered user,
I want to log in using my email and password,
so that I can access my account.

Acceptance Criteria:
Given that the user exists and the email and password match,
When the user enters their email and password and presses the login button,
Then the system grants access to their account and redirects them to the home screen.

USER STORY

Title: Upload image to the user's profile	**Priority:** High	**Estimate: 1 week**

User Story:
As a registered user,
I want to upload an image to my profile,
so that other users can see it.

Acceptance Criteria:
Given that the user has successfully completed the registration process,
When they access the upload section and select an image from their device,
Then the image is uploaded to their profile.

USER STORY

Title: Using interaction with profiles.	**Priority:** High	**Estimate: 2 weeks**

User Story:
As a user of the app,
I want to interact with other users' profiles using options such as dismiss or like.
so that I can interact with people I'm interested in and inhance my experience on the platform.

Acceptance Criteria:
Given that a user has completed their profile setup,
When they begin interacting with the profiles shown on the main screen,
Then they can tap the "Like" or "Dismiss" buttons and if mutual interest is detected, a match notification is displayed.

USER STORY

Title: Receive notification when someone likes your profile	**Priority:** High	**Estimate: 1 week**

User Story:

As a user of the app,
I want to receive a notification when someone likes my profile,
so that I can see the alert.

Acceptance Criteria:
Given that another user has liked my profile,
When I receive a notification about that like,
Then I should be able to see the notification.

USER STORY

Title: Log out	**Priority:** Media	**Estimate: 1 week**

User Story:

As a user of the app,
I want to able to log out of my account,
so that I can protect my information on shared devices.

Acceptance Criteria:
Given that the user is on the main screen,
When they tap the “Log out” button and confirm the action,
Then they are signed out and redirected to the welcome/login screen.

USER STORY

Title: Edit profile	**Priority:** Media	**Estimate: 2 weeks**

User Story:

As a registered user,
I want to edit my profile information such as name, sexual orientation, interests, lifestyle,
so that I can update my profile as I evolve or change preferences.

Acceptance Criteria:
Given that a user is logged in,
When they navigate to the profile settings,
Then they are able to update their personal information and save the changes.

USER STORY

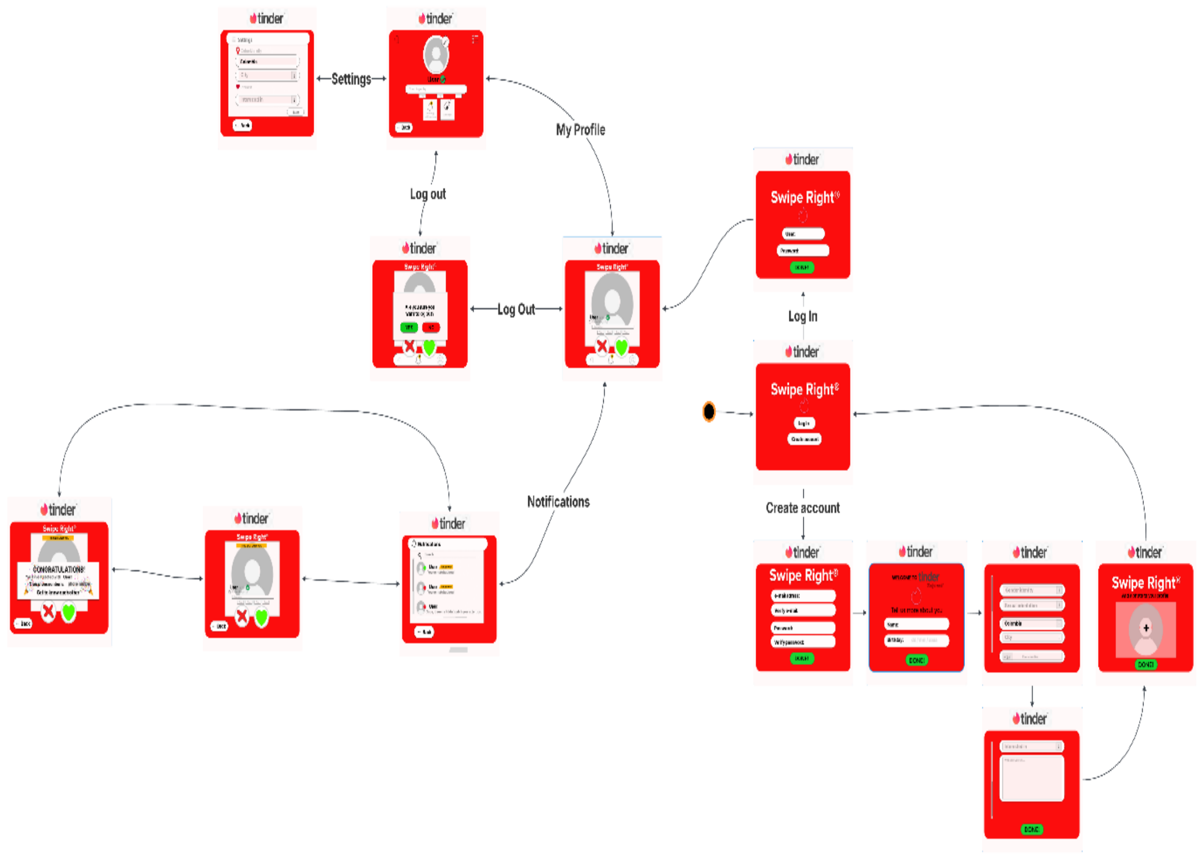
Title: View profile from notification	**Priority:** Low	**Estimate: 2 week**

User Story:

As a user of the app,
I want to tap on a notification when someone likes me,
so that I can view their profile and decide whether to interact.

Acceptance Criteria:
Given that a notification has been received,
When the user taps the notification,
Then they are redirected to the liker's profile.

MOCKUPS:



[CLICK HERE TO SEE THE MOCKUPS]

CRC CARDS:

Class Name: User	
Responsibilities: - Register a new account - Authenticate with email and password - Change password - Save and update personal information - Give like to other users	**Collaborators:** - App - Notification - Photo

Class Name: Notification	
Responsibilities: - Add a new notification to the user - Store all received notifications - Display notifications if needed	**Collaborators:** - User - App

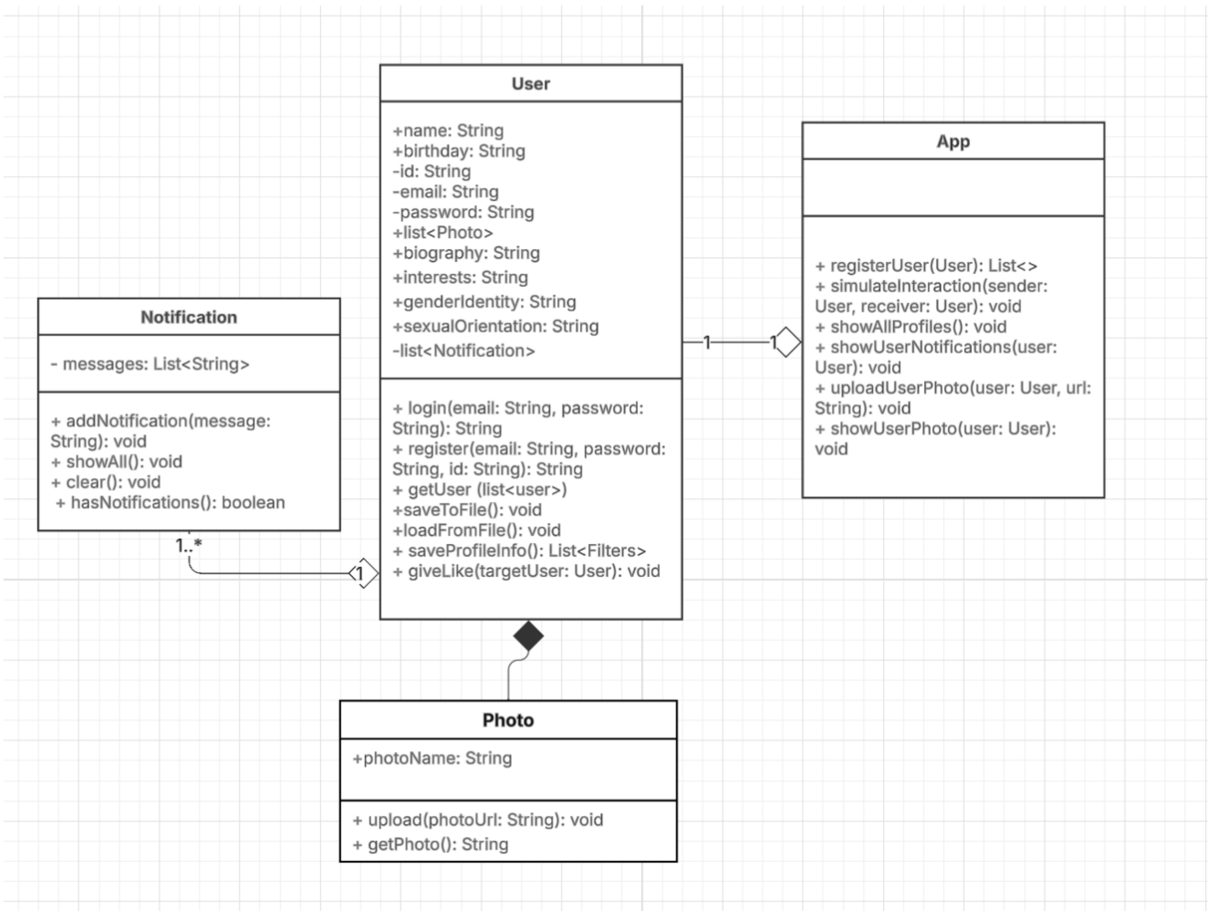
Class Name: App	
Responsibilities: - Register new users - Simulate users' interactions - Display all other users' profiles - Display notifications for a specific user - Manage user photo uploads and viewing	**Collaborators:** - User - Notification - Photo

Class Name: Photo	
Responsibilities: - Upload a profile picture - Store the image path or file	**Collaborators:** - User - App

.

<u>UML DIAGRAMS:</u>

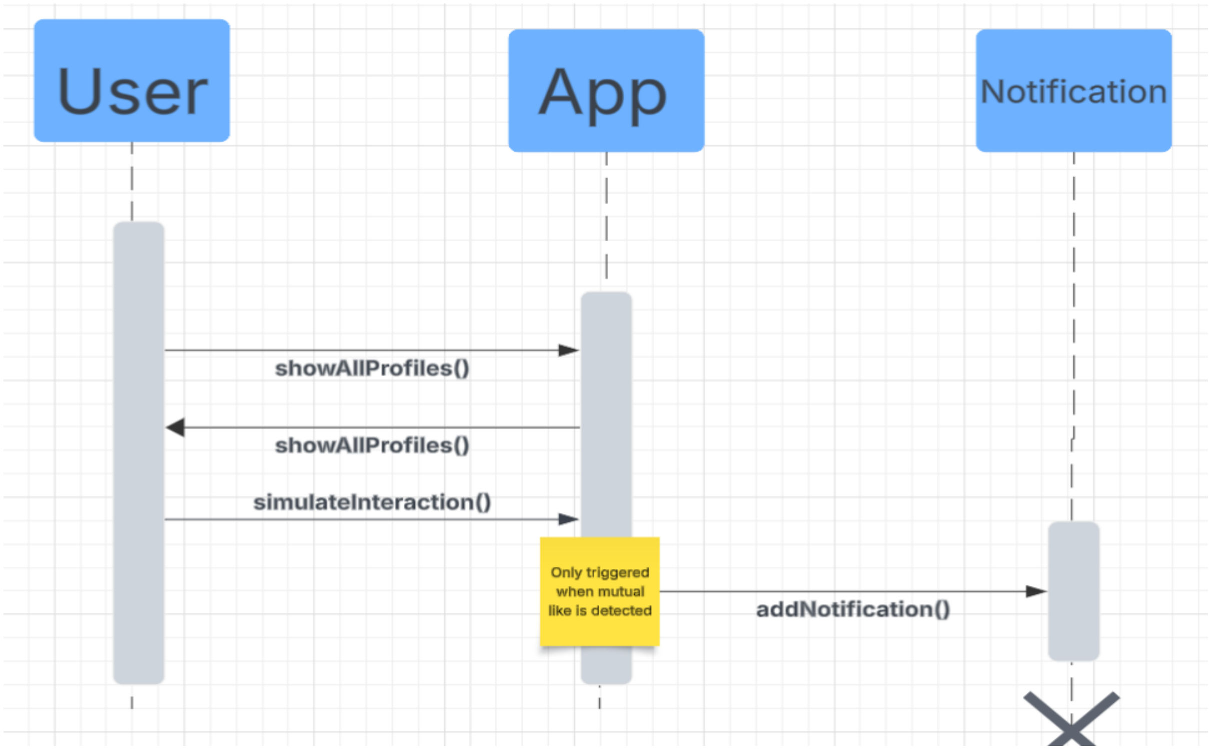
<u>Class diagram:</u>


User
+name: String
+birthday: String
-id: String
-email: String
-password: String
+list<Photo>
+biography: String
+interests: String
+genderIdentity: String
+sexualOrientation: String
-list<Notification>
+ login(email: String, password: String): String
+ register(email: String, password: String, id: String): String
+ getUser (list<user>)
+saveToFile(): void
+loadFromFile(): void
+ saveProfileInfo(): List<Filters>
+ giveLike(targetUser: User): void
App
+ registerUser(User): List<>
+ simulateInteraction(sender: User, receiver: User): void
+ showAllProfiles(): void
+ showUserNotifications(user: User): void
+ uploadUserPhoto(user: User, url: String): void
+ showUserPhoto(user: User): void
Notification
- messages: List<String>
+ addNotification(message: String): void
+ showAll(): void
+ clear(): void
+ hasNotifications(): boolean
Photo
+photoName: String
+ upload(photoUrl: String): void
+ getPhoto(): String
1
1
1..*
1

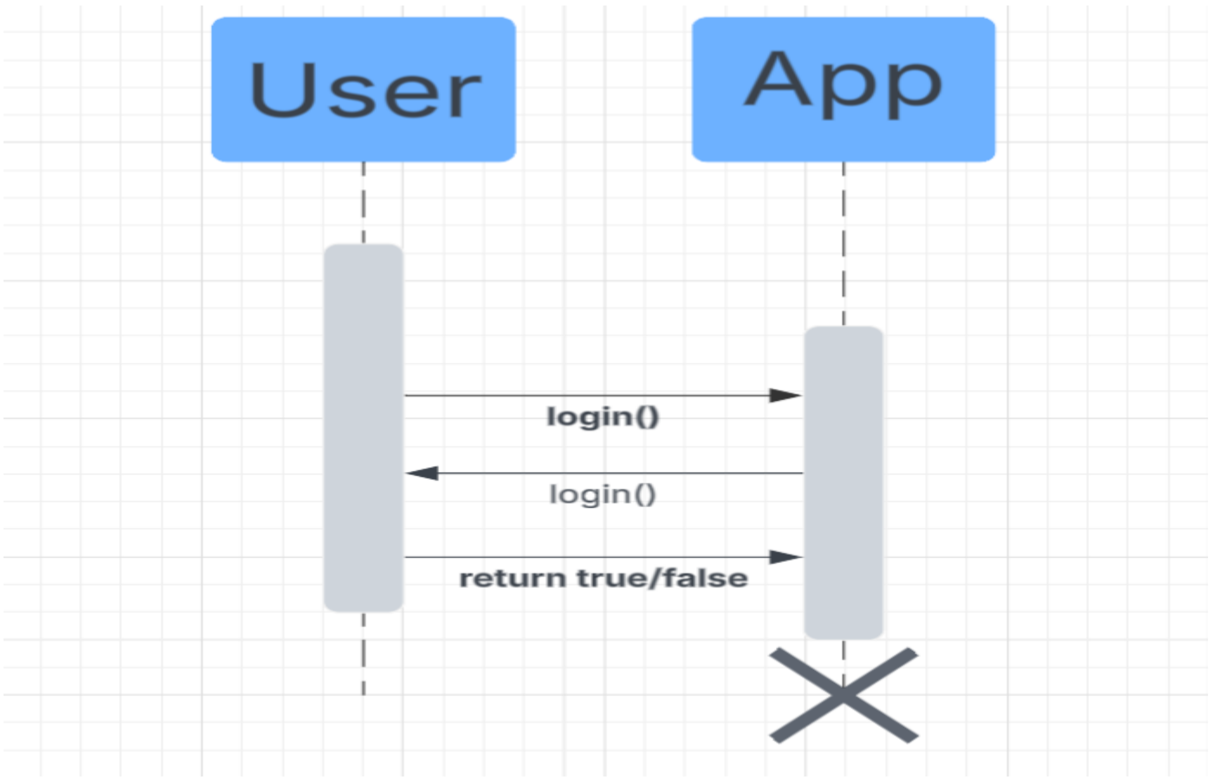
<u>CLLICK HERE TO SEE THE CLASS DIAGRAM</u>

Sequence diagram:

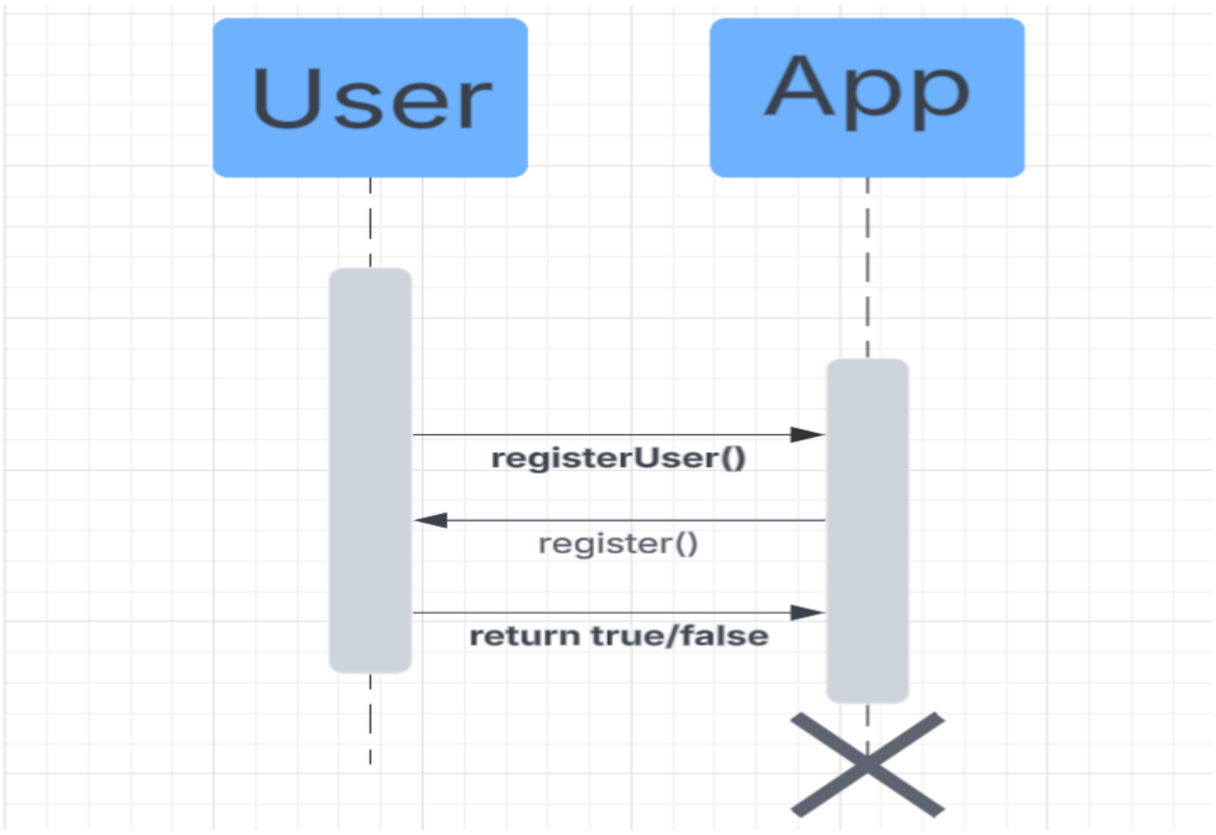
Mutual like:



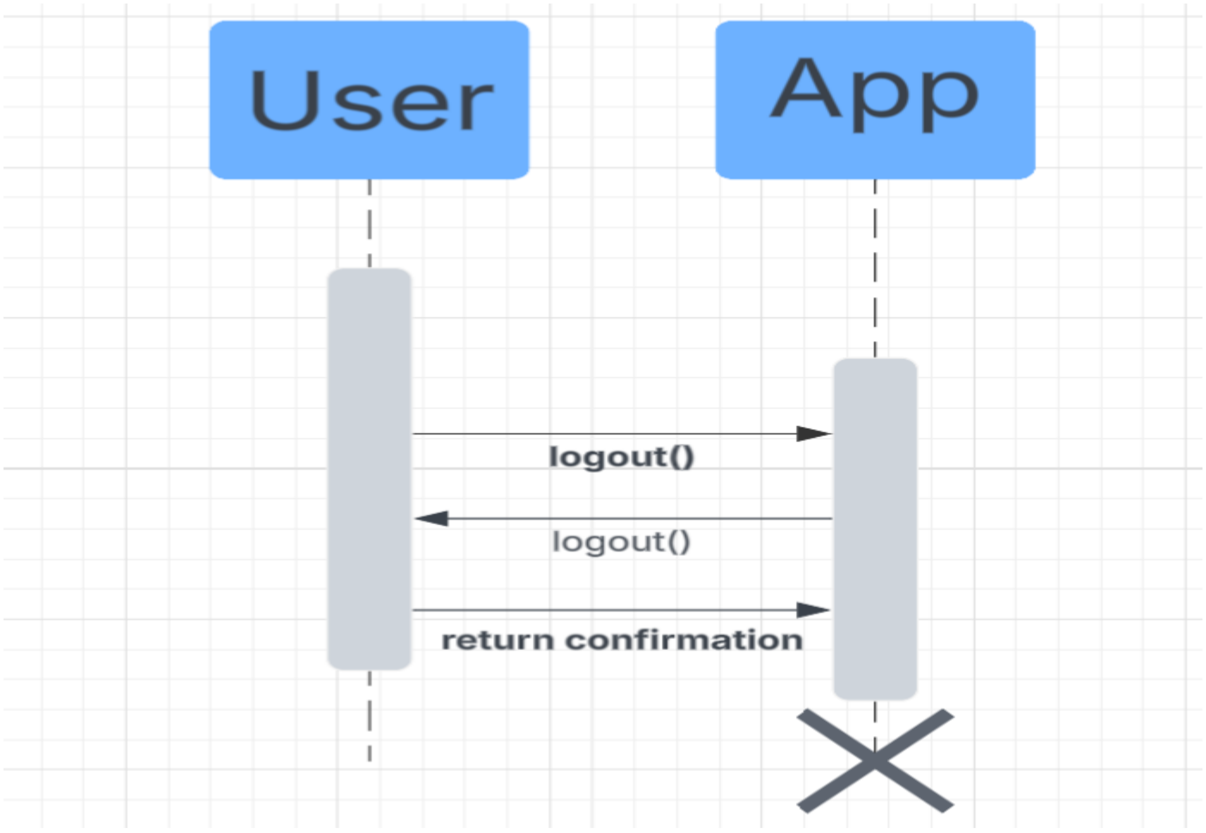
Log in:



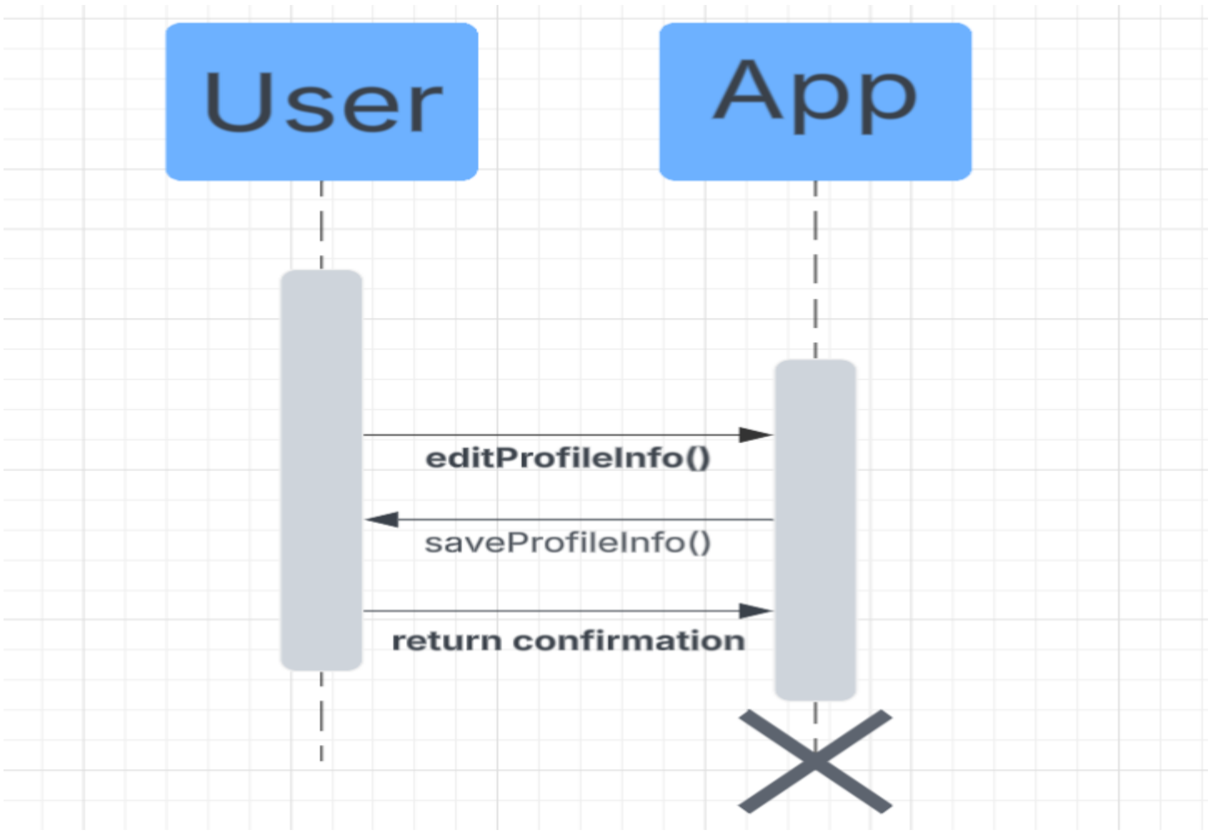
User registration:



Log out:

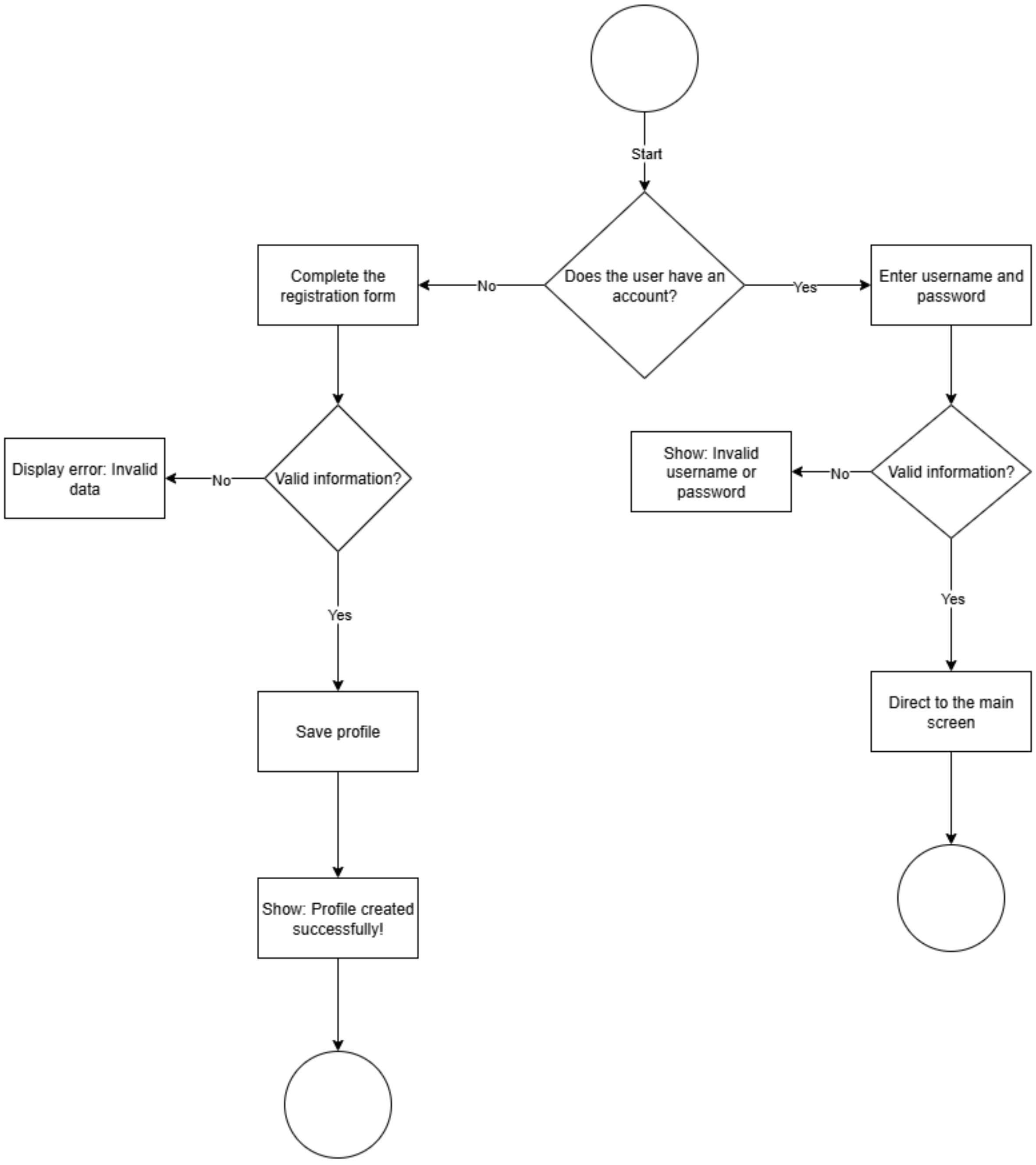


Edit profile:

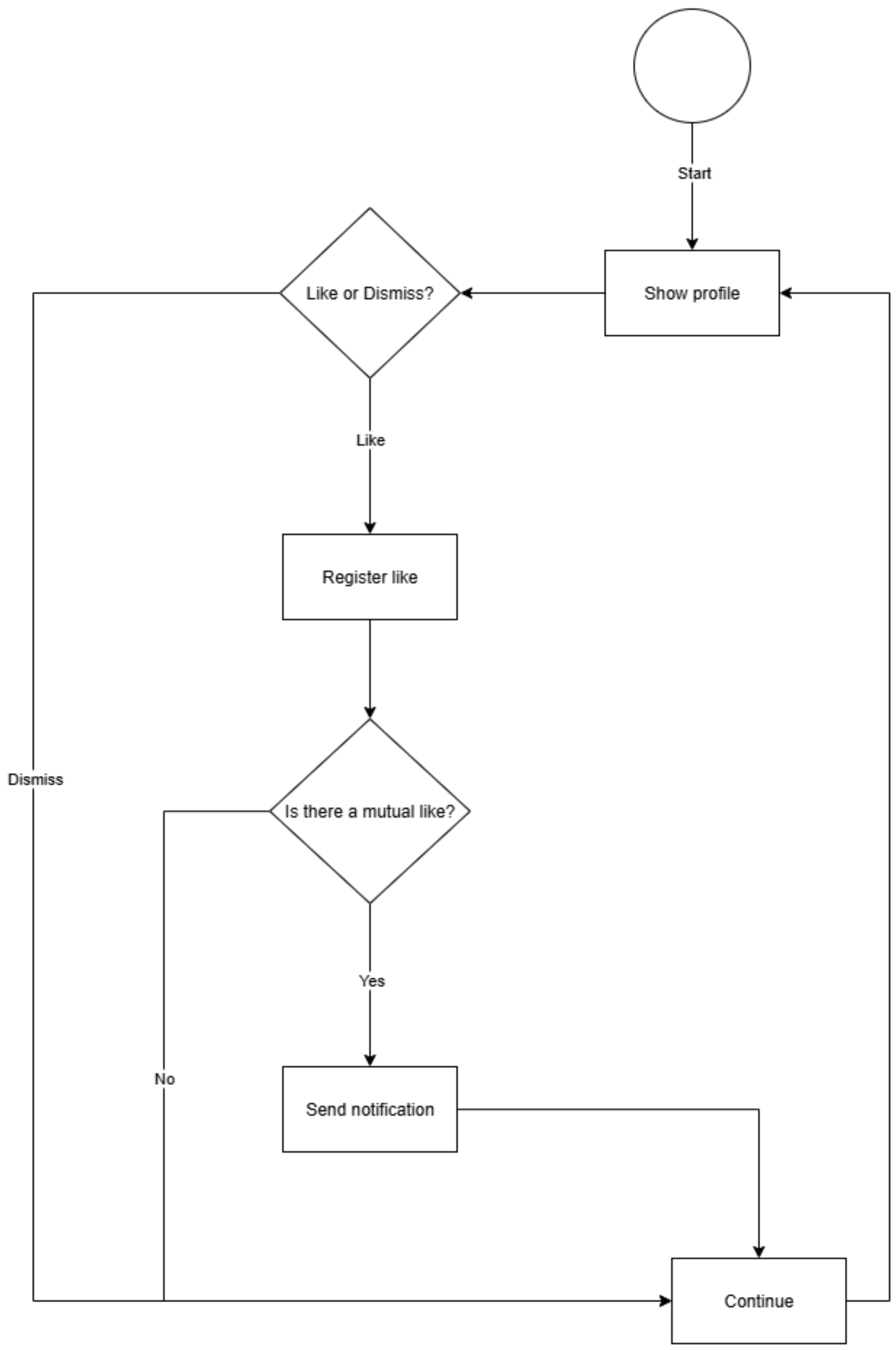


Activity diagram:

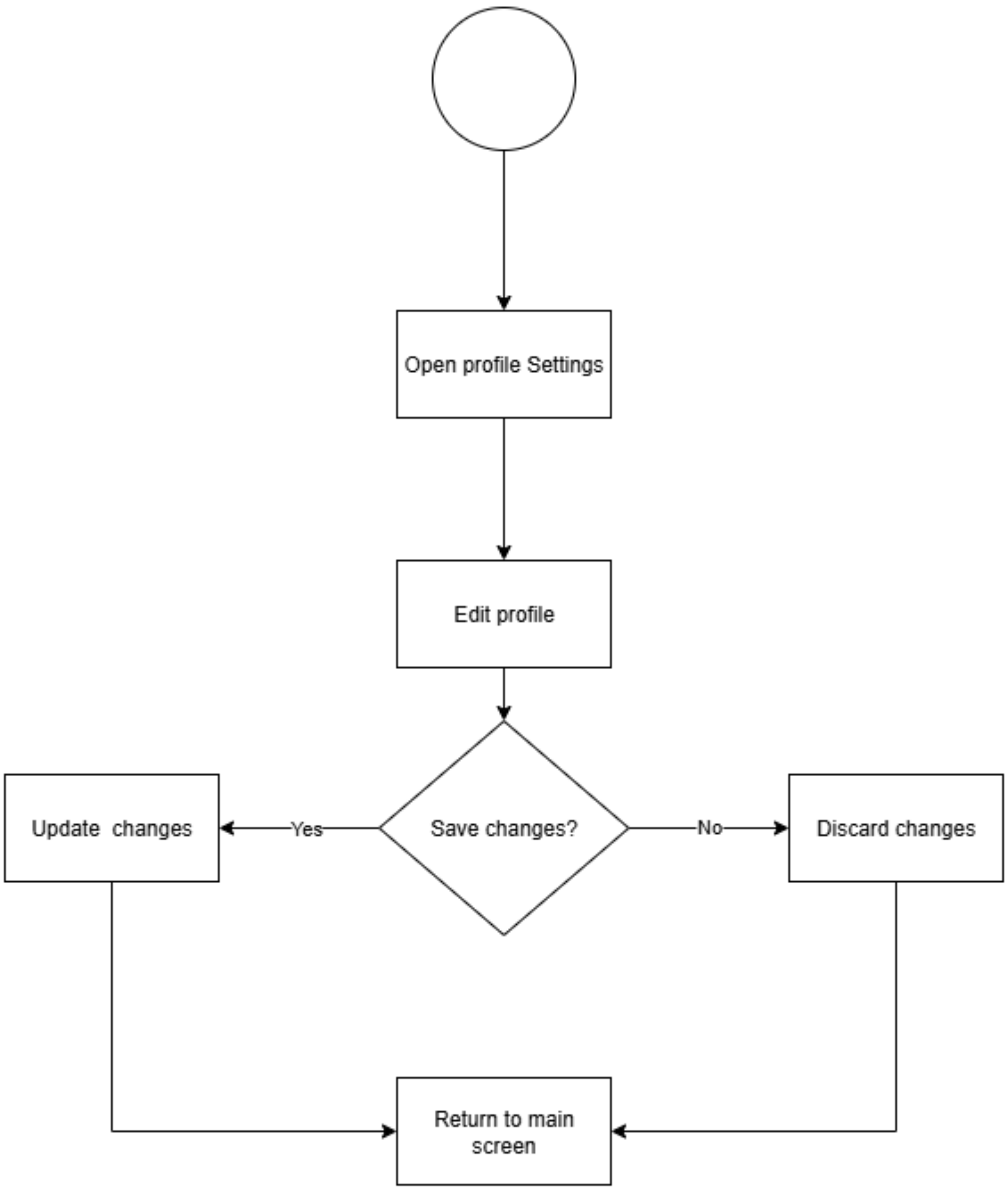
Log in or register:


Start
Does the user have an account?
No
Yes
Complete the registration form
Enter username and password
Valid information?
No
Display error: Invalid data
Yes
Save profile
Show: Profile created successfully!
Valid information?
No
Show: Invalid username or password
Yes
Direct to the main screen

Frame principal:



My profile:


Open profile Settings
Edit profile
Save changes?
Yes
No
Update changes
Discard changes
Return to main screen

IMPLEMENTATION PLAN FOR OOP CONCEPTS

Why don't we use inheritance and polymorphism?

We decided not to implement inheritance and polymorphism because we didn't really consider it because we don't have more than one way to perform the actions. You could consider inheritance in the login, creating a user registration class and a class that grants access to the admin. Since we decided to keep it simple, we didn't include a variation in access, but if it were desired in the future, this would be our implementation of inheritance and polymorphism.

Implementation Plan for OOP Concepts

	User	App	Notification	Photo
Encapsulation	✓	✓	✓	✓
Inheritance	✗	✗	✗	✗
Polymorphism	✗	✗	✗	✗
Abstraction	✓	✓	✓	✓

Encapsulation:

- User:
  Private attributes such as password, email, and ID are protected using public methods (register(), login()), ensuring data integrity.

- App:
  Coordinates the system through public methods without exposing its internal logic.

- Notification:
  It internally manages its list of messages and exposes only specific actions (addNotification(), showAll()).

- Photo:
  Hides the real URL of the photo and manipulates it only through controlled methods (upload(), getPhoto()).

Abstraction:

- User:
  - Data abstraction: Only what is necessary for authentication and profiling is exposed.
  - Behavioral abstraction: Methods such as login(), giveLike(), or saveProfileInfo() hide the technical details of processing.

- App:
  Behavioral: It operates as a flow controller. The user only interacts with registerUser() and simulateInteraction(), without knowing how the classes relate internally.

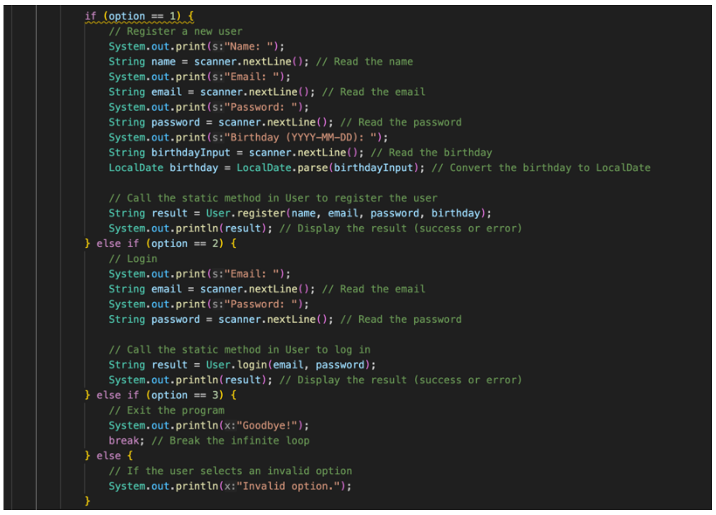
- Notification:
  - Data-based: Displays only relevant messages, hiding the structure or source of the data.
  - Behavioral: Exposes key actions such as displaying, adding, or clearing notifications, without revealing how they are stored or managed.

- Photo:
  - Data: Exposes only the photo string needed to load or display the image.
  - Behavioral: Methods allow manipulation without directly accessing the structure.

THE WORK IN PROGRESS CODE

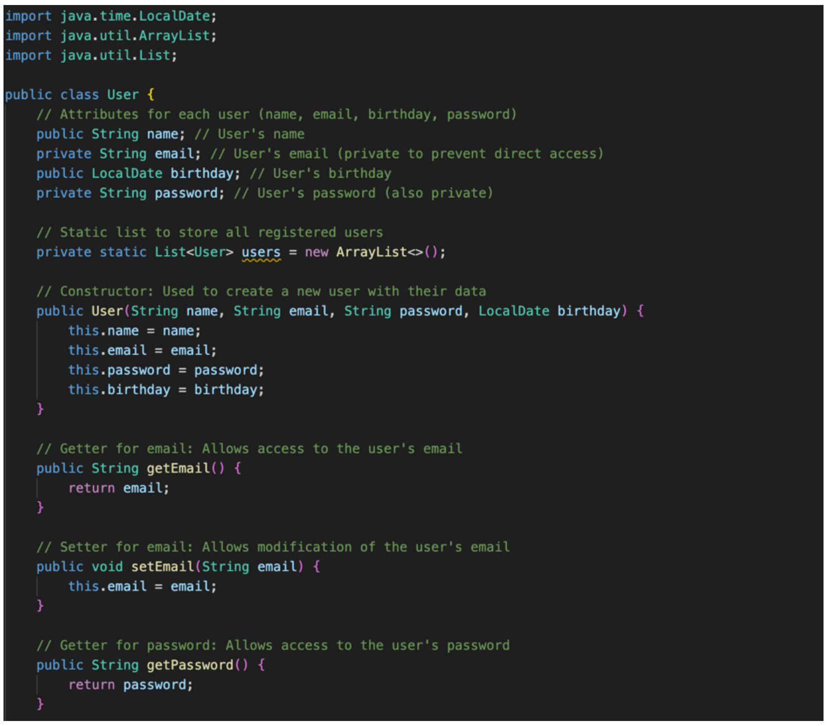
CLICK HERE:

Code snippets:



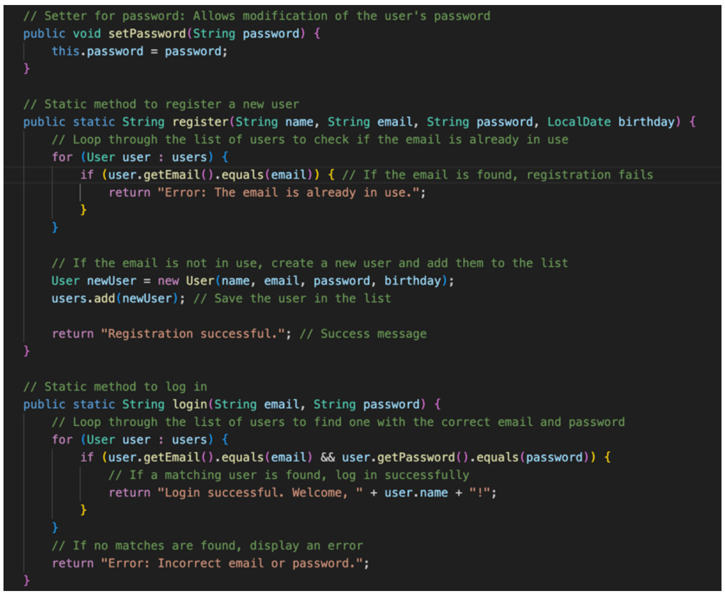
```java
if (option == 1) {
    // Register a new user
    System.out.print(s:"Name: ");
    String name = scanner.nextLine(); // Read the name
    System.out.print(s:"Email: ");
    String email = scanner.nextLine(); // Read the email
    System.out.print(s:"Password: ");
    String password = scanner.nextLine(); // Read the password
    System.out.print(s:"Birthday (YYYY-MM-DD): ");
    String birthdayInput = scanner.nextLine(); // Read the birthday
    LocalDate birthday = LocalDate.parse(birthdayInput); // Convert the birthday to LocalDate

    // Call the static method in User to register the user
    String result = User.register(name, email, password, birthday);
    System.out.println(result); // Display the result (success or error)
} else if (option == 2) {
    // Login
    System.out.print(s:"Email: ");
    String email = scanner.nextLine(); // Read the email
    System.out.print(s:"Password: ");
    String password = scanner.nextLine(); // Read the password

    // Call the static method in User to log in
    String result = User.login(email, password);
    System.out.println(result); // Display the result (success or error)
} else if (option == 3) {
    // Exit the program
    System.out.println(x:"Goodbye!");
    break; // Break the infinite loop
} else {
    // If the user selects an invalid option
    System.out.println(x:"Invalid option.");
}
```

This is the app class, here, The user must enter their personal data so that it is sent to the user class method, in order to register and then save this data in the list.



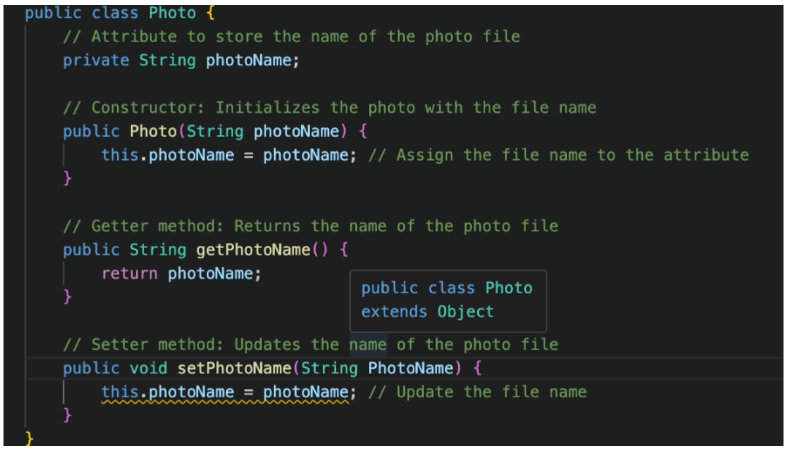
```java
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class User {
    // Attributes for each user (name, email, birthday, password)
    public String name; // User's name
    private String email; // User's email (private to prevent direct access)
    public LocalDate birthday; // User's birthday
    private String password; // User's password (also private)

    // Static list to store all registered users
    private static List<User> users = new ArrayList<>();

    // Constructor: Used to create a new user with their data
    public User(String name, String email, String password, LocalDate birthday) {
        this.name = name;
        this.email = email;
        this.password = password;
        this.birthday = birthday;
    }

    // Getter for email: Allows access to the user's email
    public String getEmail() {
        return email;
    }

    // Setter for email: Allows modification of the user's email
    public void setEmail(String email) {
        this.email = email;
    }

    // Getter for password: Allows access to the user's password
    public String getPassword() {
        return password;
    }
```

Here is the first part of the User class, you can se some attributes defined and the creation of the constructor User, also the encapsulation of the email and password.



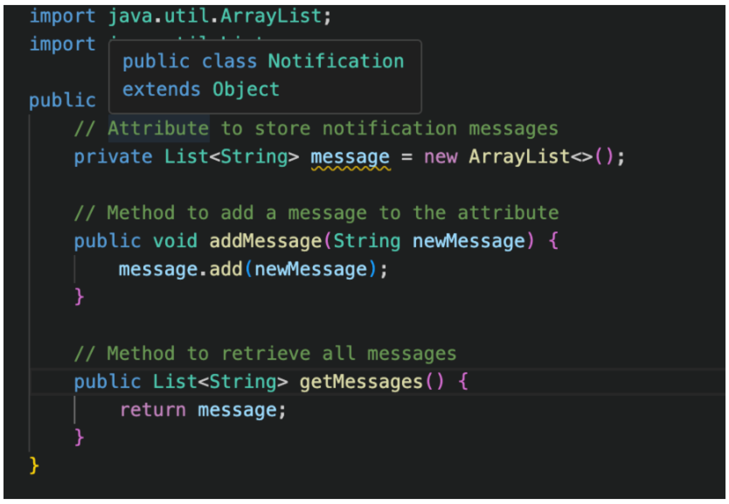
```java
    // Setter for password: Allows modification of the user's password
    public void setPassword(String password) {
        this.password = password;
    }

    // Static method to register a new user
    public static String register(String name, String email, String password, LocalDate birthday) {
        // Loop through the list of users to check if the email is already in use
        for (User user : users) {
            if (user.getEmail().equals(email)) { // If the email is found, registration fails
                return "Error: The email is already in use.";
            }
        }

        // If the email is not in use, create a new user and add them to the list
        User newUser = new User(name, email, password, birthday);
        users.add(newUser); // Save the user in the list

        return "Registration successful."; // Success message
    }

    // Static method to log in
    public static String login(String email, String password) {
        // Loop through the list of users to find one with the correct email and password
        for (User user : users) {
            if (user.getEmail().equals(email) && user.getPassword().equals(password)) {
                // If a matching user is found, log in successfully
                return "Login successful. Welcome, " + user.name + "!";
            }
        }
        // If no matches are found, display an error
        return "Error: Incorrect email or password.";
    }
```

This is the second part of the User class, as you can se, we are defining the register method, using all the attributes that we defined.



```java
public class Photo {
    // Attribute to store the name of the photo file
    private String photoName;

    // Constructor: Initializes the photo with the file name
    public Photo(String photoName) {
        this.photoName = photoName; // Assign the file name to the attribute
    }

    // Getter method: Returns the name of the photo file
    public String getPhotoName() {
        return photoName;                       public class Photo
    }                                           extends Object

    // Setter method: Updates the name of the photo file
    public void setPhotoName(String PhotoName) {
        this.photoName = photoName; // Update the file name
    }
}
```

This is the photo class, and we only defined the attributes for now



```java
import java.util.ArrayList;
import     public class Notification
           extends Object
public
    // Attribute to store notification messages
    private List<String> message = new ArrayList<>();

    // Method to add a message to the attribute
    public void addMessage(String newMessage) {
        message.add(newMessage);
    }

    // Method to retrieve all messages
    public List<String> getMessages() {
        return message;
    }
}
```

This is the notification class, as you can see, we only defined the attributes for this class, for now

The code is working, it only takes your info defined on the user class, and makes the register process, and u can loggin in the app, but we figure out how to do it with the list, also we make the error message if the password is wrong or if the user doesn't exist.

In future snippets we want to have the full app with the minimum requirements.

SOLID-Focused Implementation:

Single responsibility:

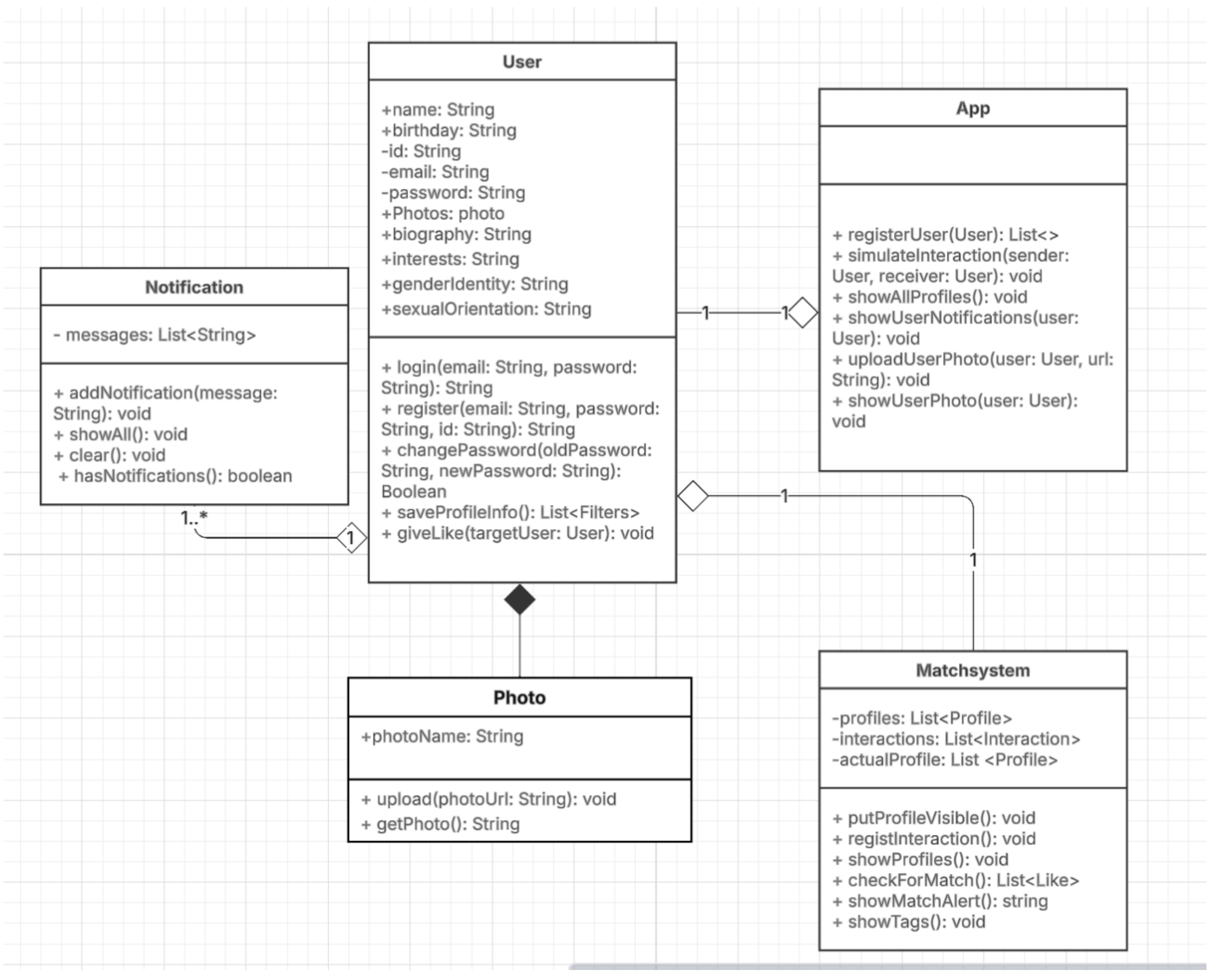
In the first principle of single responsibility, we organize classes with a single key purpose. In this case, by changing the diagram to focus on key responsibilities of the program, we apply the single responsibility principle without having to modify it again, since each class has its own purpose.

As an example of this, you can look at the CRC cards and the methods in the UML diagram, where you can identify that each module of the program is based on a single responsibility.

Open/Closed:

In the second SOLID principle, we can be sure that the program is ready so that if an extra responsibility needs to be added, it can be added without having to modify the rest of the modules, since each module works with a single responsibility, and would not suffer any damage or modification if you want to add a class, for example, if you wanted to add the match system, you can create another class that would be added to run from the app like the others without modifying the other classes.

As an example of Open/Closed, we will show the UML diagram when in the future if required we add the match system


User
+name: String
+birthday: String
-id: String
-email: String
-password: String
+Photos: photo
+biography: String
+interests: String
+genderIdentity: String
+sexualOrientation: String
+ login(email: String, password: String): String
+ register(email: String, password: String, id: String): String
+ changePassword(oldPassword: String, newPassword: String): Boolean
+ saveProfileInfo(): List<Filters>
+ giveLike(targetUser: User): void
App
+ registerUser(User): List<>
+ simulateInteraction(sender: User, receiver: User): void
+ showAllProfiles(): void
+ showUserNotifications(user: User): void
+ uploadUserPhoto(user: User, url: String): void
+ showUserPhoto(user: User): void
Notification
- messages: List<String>
+ addNotification(message: String): void
+ showAll(): void
+ clear(): void
+ hasNotifications(): boolean
1..*
1
1
1
1
1
Photo
+photoName: String
+ upload(photoUrl: String): void
+ getPhoto(): String
Matchsystem
-profiles: List<Profile>
-interactions: List<Interaction>
-actualProfile: List <Profile>
+ putProfileVisible(): void
+ registInteraction(): void
+ showProfiles(): void
+ checkForMatch(): List<Like>
+ showMatchAlert(): string
+ showTags(): void

Liskov substitution:

In the third SOLID principle, we cannot verify if it can be applied, since we do not have classes of higher or lower hierarchy, since there is no inheritance or polymorphism in our program.

Interface segregation:

In the fourth principle of interface segregation, we cannot be sure of this either since we do not use interfaces in our program, but we can say with certainty that the methods they have are well implemented and all are necessary.

Dependency Inversion

For the fifth SOLID principle, our program does not have a hierarchy or crucial dependency, in fact, the only existing dependency is that of the Photo class with the User class, this is because we take the Photo class as a type of data that would be anchored to the attributes that the user profile should have within the app.

Swing-based GUI Prototype

The creation of an initial graphical interface improves the system's usability, allowing for more fluid user interaction. This was implemented considering the previously developed logic, the object-oriented principles applied, and the mockups created during the design phase.

- The initial implementation of showWelcomeScreen() creates a JFrame titled "TinderApp", which serves as the entry point to the application.

- This frame uses basic Swing components, such as:

    - JLabel to display a welcoming message with "Welcome to" and "TinderApp", following the style planned in the mockups.
    - JButton to provide interactive options: "Log In" and "Create Account".

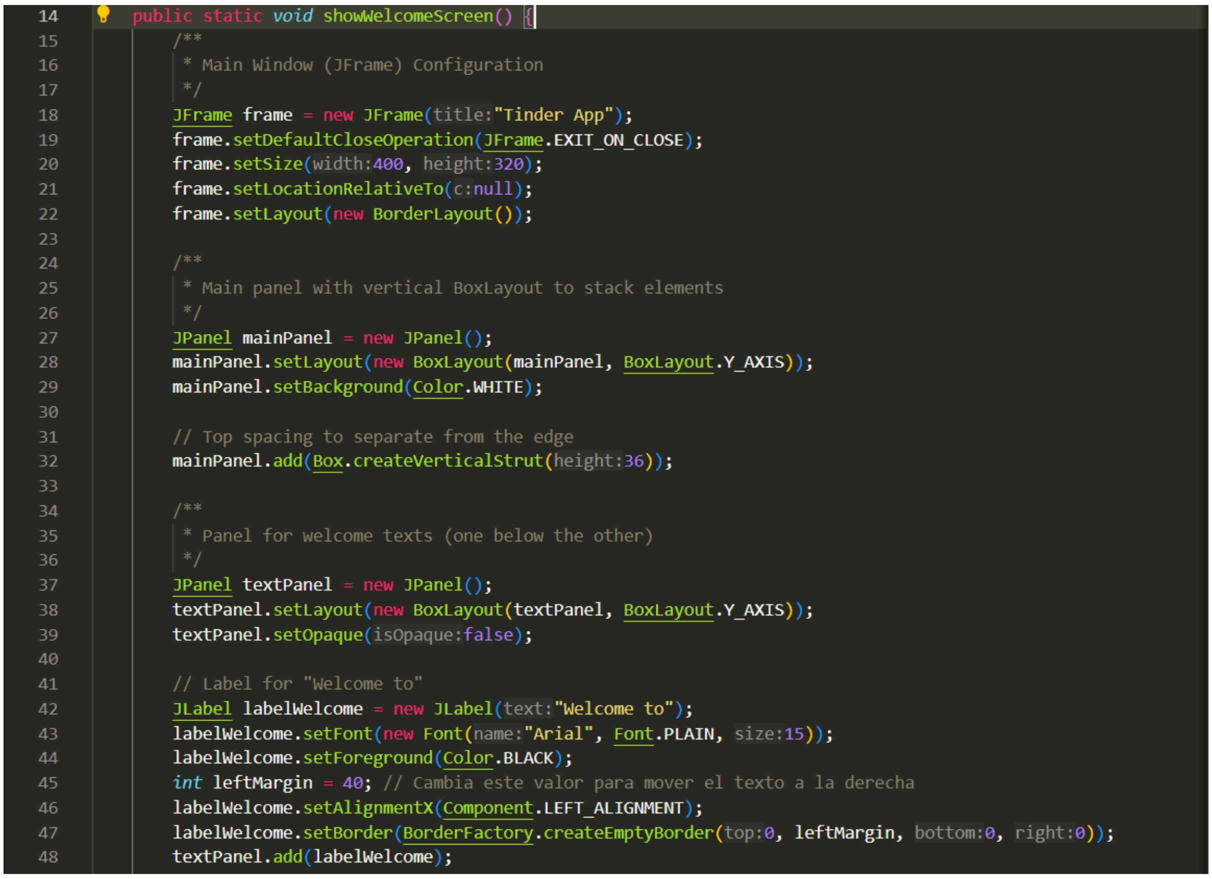
- All these components are organized within JPanels, stacked vertically using BoxLayout, which results in a visually clean and organized interface.

Interface with the central logic

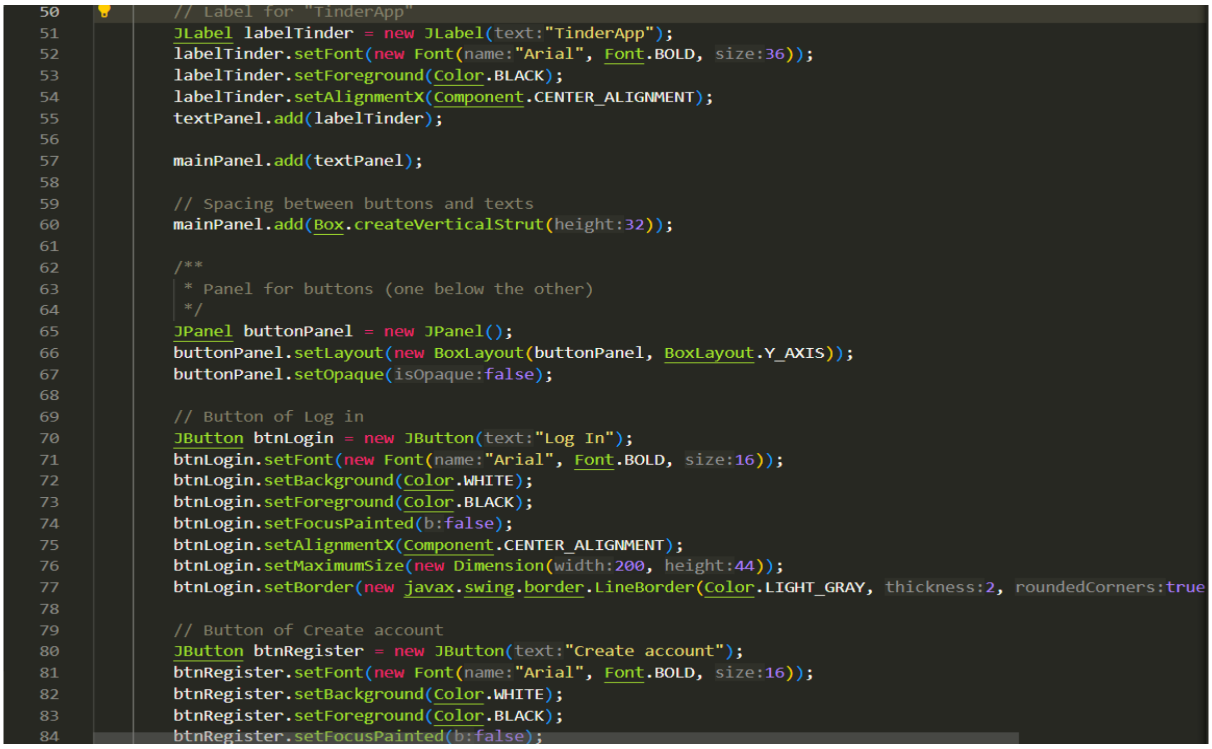
Each button is wired to an ActionListener that bridges the graphical interface with the internal logic:

- Pressing "Log In" closes the current frame and calls startConsoleApp(2), launching the login flow defined in the core classes.
- Pressing "Create Account" invokes showRegisterFrame(), opening a new JFrame with fields that collect user data.

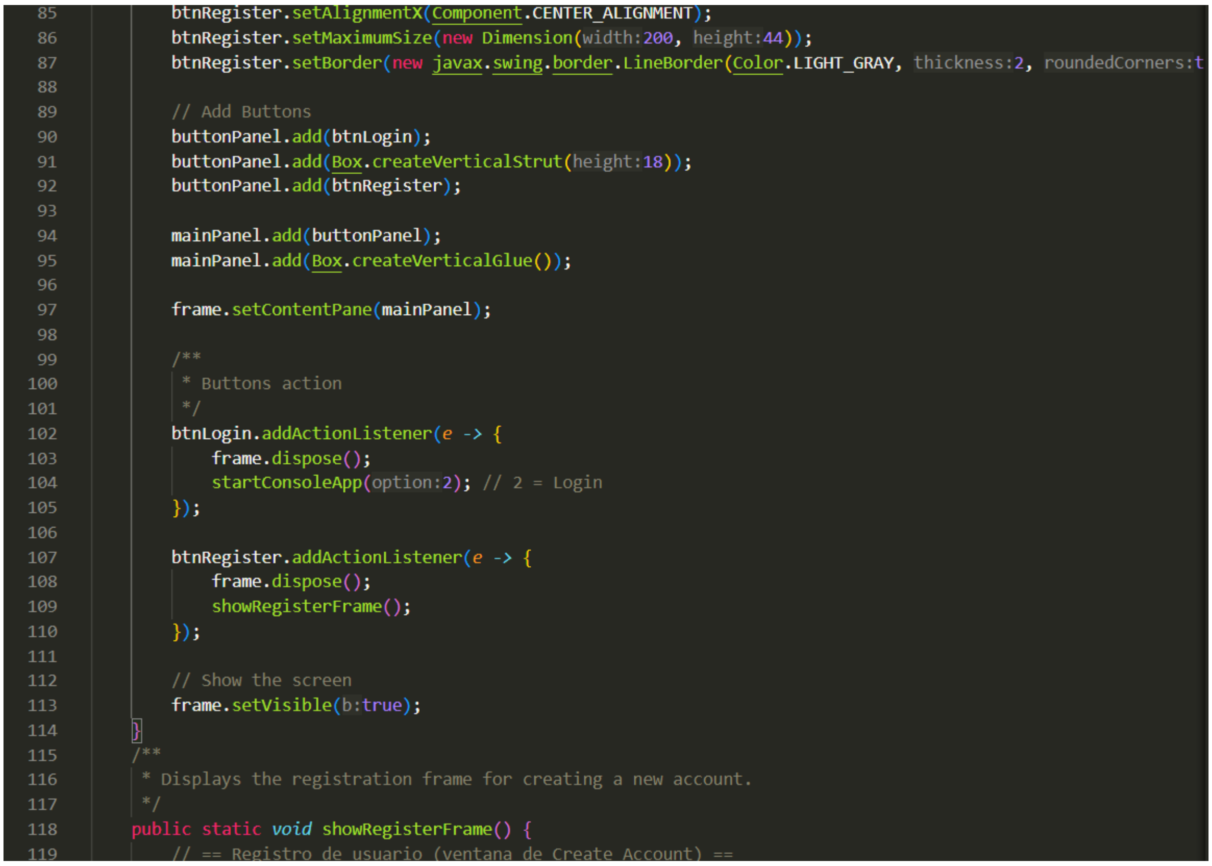
This interaction is aligned with the activity and sequence diagrams, ensuring that the GUI triggers the same flows described in the UML models.



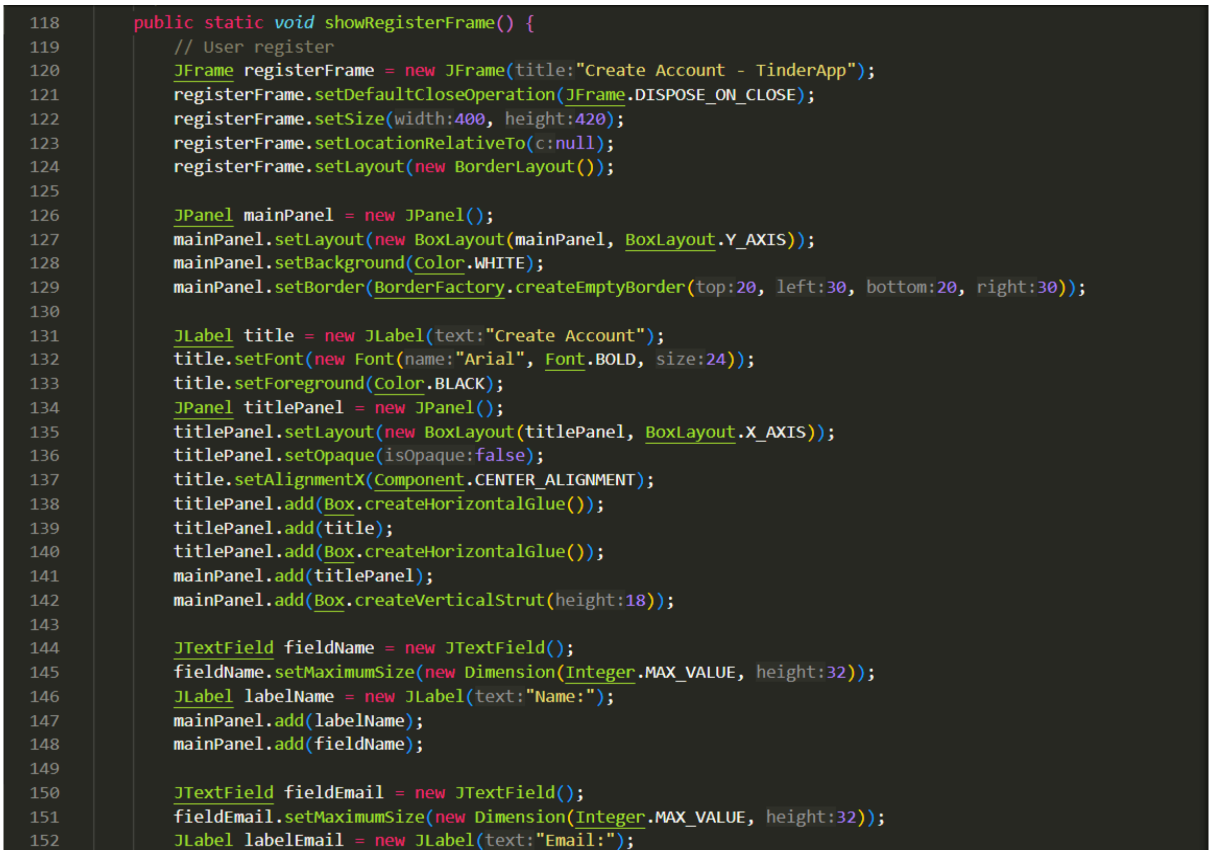
```java
public static void showWelcomeScreen() {
    /**
     * Main Window (JFrame) Configuration
     */
    JFrame frame = new JFrame(title:"Tinder App");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(width:400, height:320);
    frame.setLocationRelativeTo(c:null);
    frame.setLayout(new BorderLayout());

    /**
     * Main panel with vertical BoxLayout to stack elements
     */
    JPanel mainPanel = new JPanel();
    mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));
    mainPanel.setBackground(Color.WHITE);

    // Top spacing to separate from the edge
    mainPanel.add(Box.createVerticalStrut(height:36));

    /**
     * Panel for welcome texts (one below the other)
     */
    JPanel textPanel = new JPanel();
    textPanel.setLayout(new BoxLayout(textPanel, BoxLayout.Y_AXIS));
    textPanel.setOpaque(isOpaque:false);

    // Label for "Welcome to"
    JLabel labelWelcome = new JLabel(text:"Welcome to");
    labelWelcome.setFont(new Font(name:"Arial", Font.PLAIN, size:15));
    labelWelcome.setForeground(Color.BLACK);
    int leftMargin = 40; // Cambia este valor para mover el texto a la derecha
    labelWelcome.setAlignmentX(Component.LEFT_ALIGNMENT);
    labelWelcome.setBorder(BorderFactory.createEmptyBorder(top:0, leftMargin, bottom:0, right:0));
    textPanel.add(labelWelcome);
```



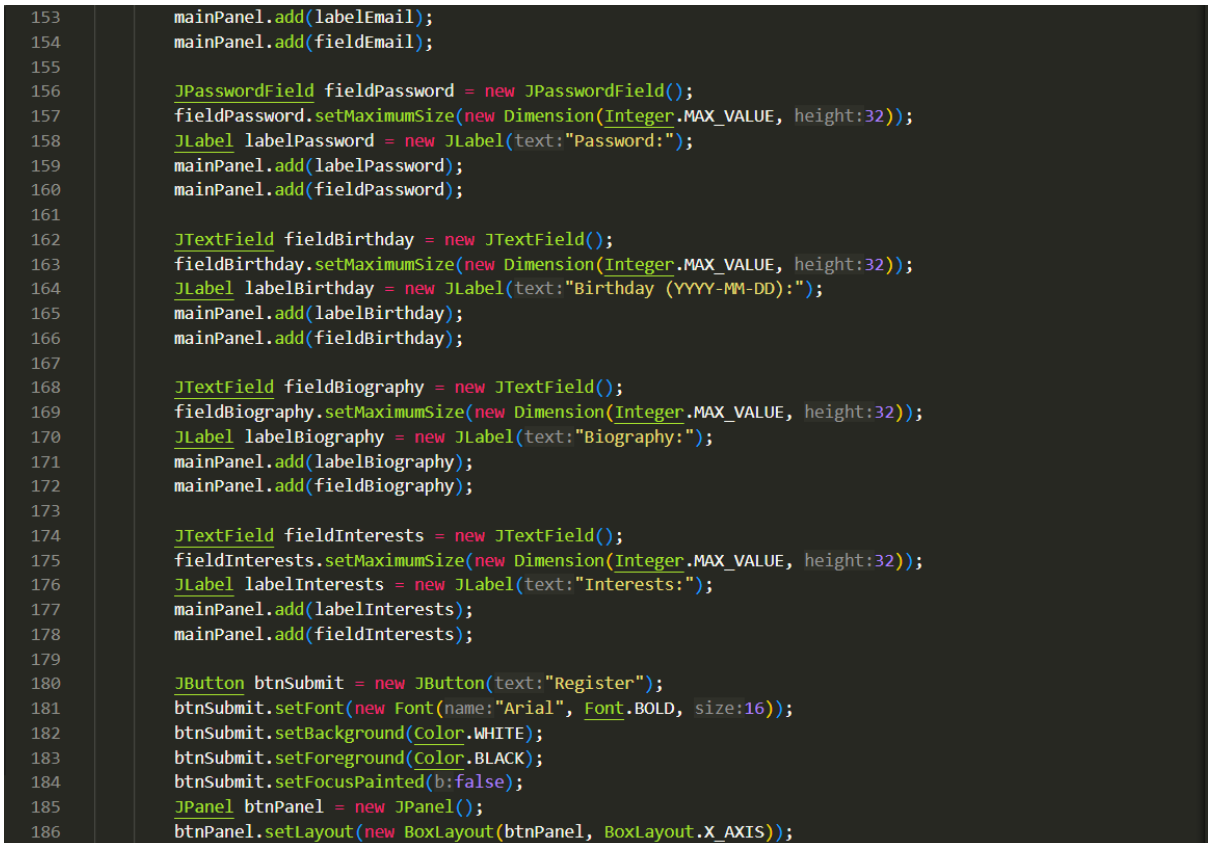
```java
    // Label for "TinderApp"
    JLabel labelTinder = new JLabel(text:"TinderApp");
    labelTinder.setFont(new Font(name:"Arial", Font.BOLD, size:36));
    labelTinder.setForeground(Color.BLACK);
    labelTinder.setAlignmentX(Component.CENTER_ALIGNMENT);
    textPanel.add(labelTinder);

    mainPanel.add(textPanel);

    // Spacing between buttons and texts
    mainPanel.add(Box.createVerticalStrut(height:32));

    /**
     * Panel for buttons (one below the other)
     */
    JPanel buttonPanel = new JPanel();
    buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.Y_AXIS));
    buttonPanel.setOpaque(isOpaque:false);

    // Button of Log in
    JButton btnLogin = new JButton(text:"Log In");
    btnLogin.setFont(new Font(name:"Arial", Font.BOLD, size:16));
    btnLogin.setBackground(Color.WHITE);
    btnLogin.setForeground(Color.BLACK);
    btnLogin.setFocusPainted(b:false);
    btnLogin.setAlignmentX(Component.CENTER_ALIGNMENT);
    btnLogin.setMaximumSize(new Dimension(width:200, height:44));
    btnLogin.setBorder(new javax.swing.border.LineBorder(Color.LIGHT_GRAY, thickness:2, roundedCorners:true

    // Button of Create account
    JButton btnRegister = new JButton(text:"Create account");
    btnRegister.setFont(new Font(name:"Arial", Font.BOLD, size:16));
    btnRegister.setBackground(Color.WHITE);
    btnRegister.setForeground(Color.BLACK);
    btnRegister.setFocusPainted(b:false);
```



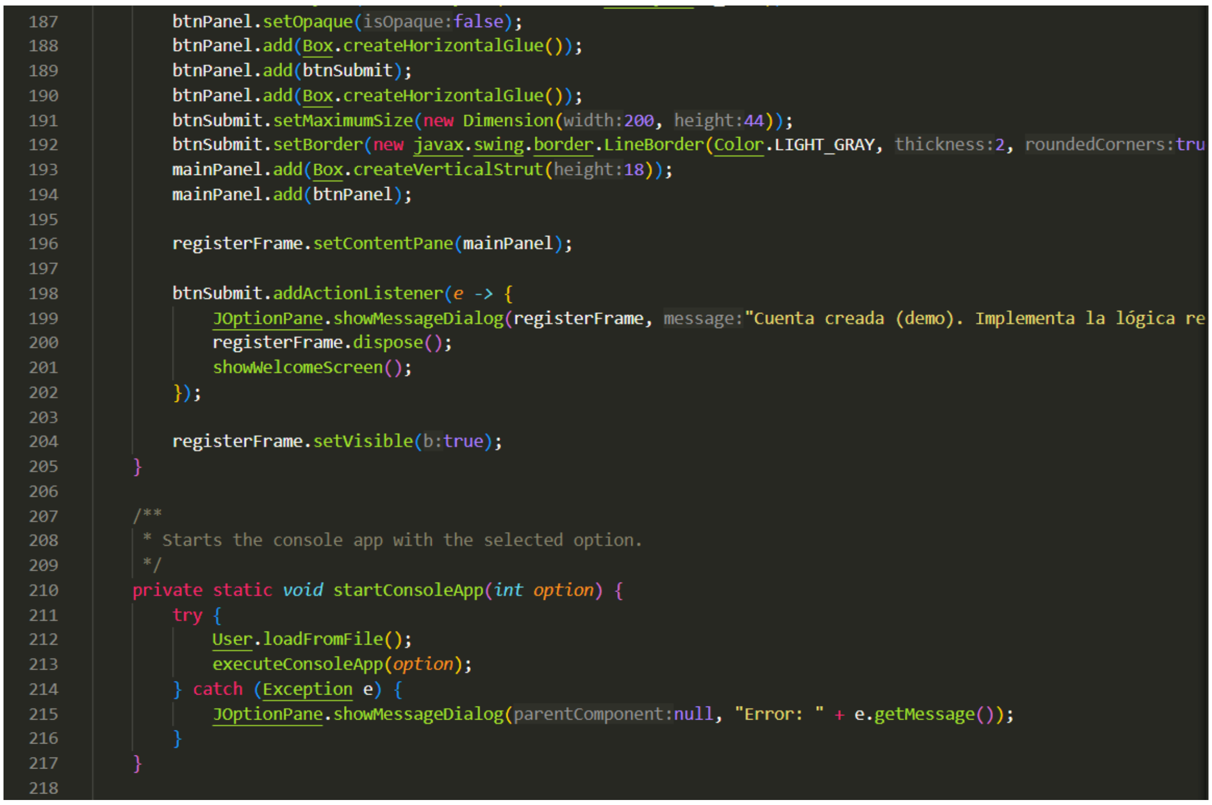
```java
        btnRegister.setAlignmentX(Component.CENTER_ALIGNMENT);
        btnRegister.setMaximumSize(new Dimension(width:200, height:44));
        btnRegister.setBorder(new javax.swing.border.LineBorder(Color.LIGHT_GRAY, thickness:2, roundedCorners:t

        // Add Buttons
        buttonPanel.add(btnLogin);
        buttonPanel.add(Box.createVerticalStrut(height:18));
        buttonPanel.add(btnRegister);

        mainPanel.add(buttonPanel);
        mainPanel.add(Box.createVerticalGlue());

        frame.setContentPane(mainPanel);

        /**
         * Buttons action
         */
        btnLogin.addActionListener(e -> {
            frame.dispose();
            startConsoleApp(option:2); // 2 = Login
        });

        btnRegister.addActionListener(e -> {
            frame.dispose();
            showRegisterFrame();
        });

        // Show the screen
        frame.setVisible(b:true);
    }
    /**
     * Displays the registration frame for creating a new account.
     */
    public static void showRegisterFrame() {
        // == Registro de usuario (ventana de Create Account) ==
```



```java
    public static void showRegisterFrame() {
        // User register
        JFrame registerFrame = new JFrame(title:"Create Account - TinderApp");
        registerFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        registerFrame.setSize(width:400, height:420);
        registerFrame.setLocationRelativeTo(c:null);
        registerFrame.setLayout(new BorderLayout());

        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));
        mainPanel.setBackground(Color.WHITE);
        mainPanel.setBorder(BorderFactory.createEmptyBorder(top:20, left:30, bottom:20, right:30));

        JLabel title = new JLabel(text:"Create Account");
        title.setFont(new Font(name:"Arial", Font.BOLD, size:24));
        title.setForeground(Color.BLACK);
        JPanel titlePanel = new JPanel();
        titlePanel.setLayout(new BoxLayout(titlePanel, BoxLayout.X_AXIS));
        titlePanel.setOpaque(isOpaque:false);
        title.setAlignmentX(Component.CENTER_ALIGNMENT);
        titlePanel.add(Box.createHorizontalGlue());
        titlePanel.add(title);
        titlePanel.add(Box.createHorizontalGlue());
        mainPanel.add(titlePanel);
        mainPanel.add(Box.createVerticalStrut(height:18));

        JTextField fieldName = new JTextField();
        fieldName.setMaximumSize(new Dimension(Integer.MAX_VALUE, height:32));
        JLabel labelName = new JLabel(text:"Name:");
        mainPanel.add(labelName);
        mainPanel.add(fieldName);

        JTextField fieldEmail = new JTextField();
        fieldEmail.setMaximumSize(new Dimension(Integer.MAX_VALUE, height:32));
        JLabel labelEmail = new JLabel(text:"Email:");
```



```
        mainPanel.add(labelEmail);
        mainPanel.add(fieldEmail);

        JPasswordField fieldPassword = new JPasswordField();
        fieldPassword.setMaximumSize(new Dimension(Integer.MAX_VALUE, height:32));
        JLabel labelPassword = new JLabel(text:"Password:");
        mainPanel.add(labelPassword);
        mainPanel.add(fieldPassword);

        JTextField fieldBirthday = new JTextField();
        fieldBirthday.setMaximumSize(new Dimension(Integer.MAX_VALUE, height:32));
        JLabel labelBirthday = new JLabel(text:"Birthday (YYYY-MM-DD):");
        mainPanel.add(labelBirthday);
        mainPanel.add(fieldBirthday);

        JTextField fieldBiography = new JTextField();
        fieldBiography.setMaximumSize(new Dimension(Integer.MAX_VALUE, height:32));
        JLabel labelBiography = new JLabel(text:"Biography:");
        mainPanel.add(labelBiography);
        mainPanel.add(fieldBiography);

        JTextField fieldInterests = new JTextField();
        fieldInterests.setMaximumSize(new Dimension(Integer.MAX_VALUE, height:32));
        JLabel labelInterests = new JLabel(text:"Interests:");
        mainPanel.add(labelInterests);
        mainPanel.add(fieldInterests);

        JButton btnSubmit = new JButton(text:"Register");
        btnSubmit.setFont(new Font(name:"Arial", Font.BOLD, size:16));
        btnSubmit.setBackground(Color.WHITE);
        btnSubmit.setForeground(Color.BLACK);
        btnSubmit.setFocusPainted(b:false);
        JPanel btnPanel = new JPanel();
        btnPanel.setLayout(new BoxLayout(btnPanel, BoxLayout.X_AXIS));
```



```
        btnPanel.setOpaque(isOpaque:false);
        btnPanel.add(Box.createHorizontalGlue());
        btnPanel.add(btnSubmit);
        btnPanel.add(Box.createHorizontalGlue());
        btnSubmit.setMaximumSize(new Dimension(width:200, height:44));
        btnSubmit.setBorder(new javax.swing.border.LineBorder(Color.LIGHT_GRAY, thickness:2, roundedCorners:tru
        mainPanel.add(Box.createVerticalStrut(height:18));
        mainPanel.add(btnPanel);

        registerFrame.setContentPane(mainPanel);

        btnSubmit.addActionListener(e -> {
            JOptionPane.showMessageDialog(registerFrame, message:"Cuenta creada (demo). Implementa la lógica re
            registerFrame.dispose();
            showWelcomeScreen();
        });

        registerFrame.setVisible(b:true);
    }

    /**
     * Starts the console app with the selected option.
     */
    private static void startConsoleApp(int option) {
        try {
            User.loadFromFile();
            executeConsoleApp(option);
        } catch (Exception e) {
            JOptionPane.showMessageDialog(parentComponent:null, "Error: " + e.getMessage());
        }
    }

```

As can be seen in the code screenshots, lines 10 to 219 correspond to the implementation of the graphical interface using Java Swing, which provides the visual foundation of this application. This section of the code strategically leverages Swing components (such as JFrame, JPanel, JLabel, JButton, JTextField, and JPasswordField) to deliver a user-friendly, structured and intuitive interface, aligned with the usability objectives defined for the project.

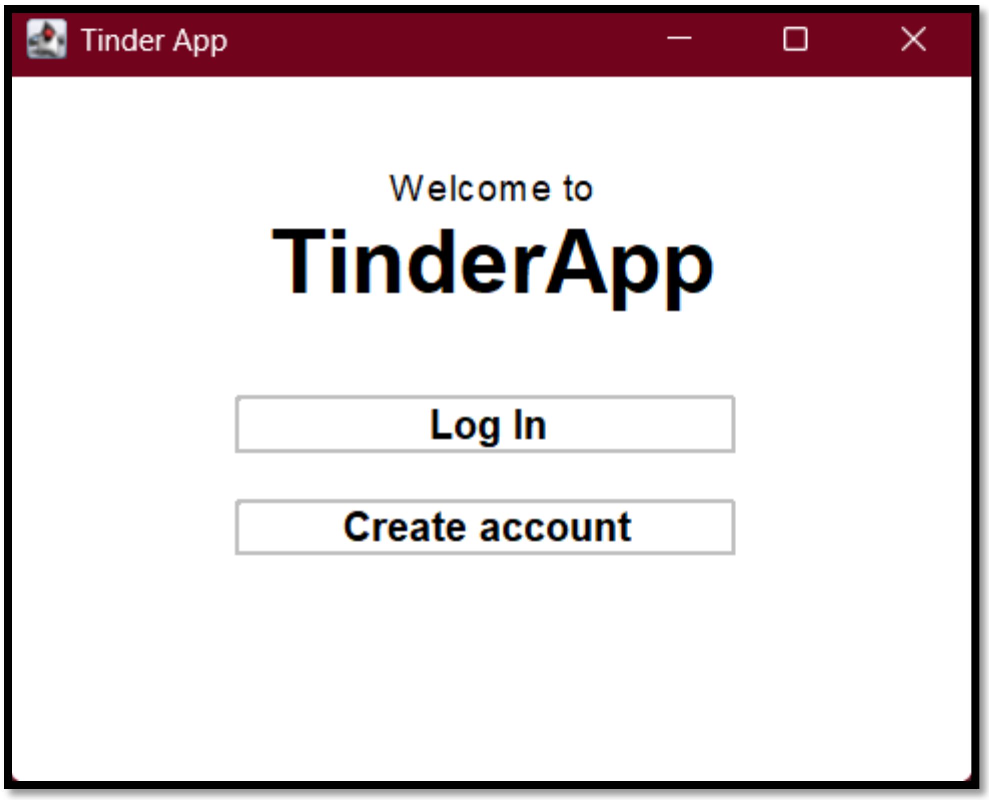
This graphical layer was built upon the robust internal logic previously established in the application's core. By following this approach, the interface is not an isolated aesthetic element, but rather a direct extension of the system's functionalities, ensuring a seamless connection between what the user sees and what the underlying OOP-based logic executes.

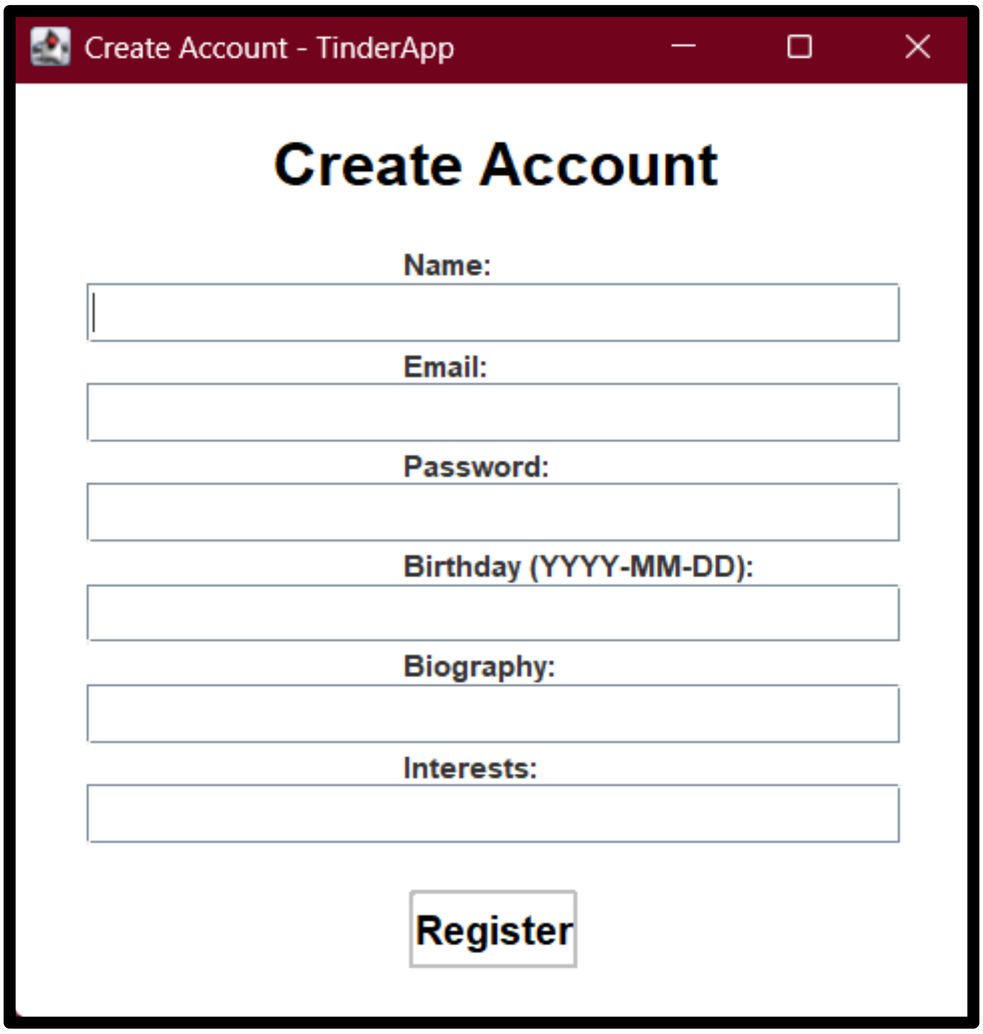
Specifically:

- Lines 10 - 150 handle the welcome screen (showWelcomeScreen), constructing a JFrame titled "TinderApp" that includes welcome messages and buttons for "Log In" and "Create account". Each button is wired to trigger the appropriate next step in the workflow, seamlessly invoking either the login or the registration process.
- Lines 150 - 219 define the registration screen (showRegisterFrame), dynamically generating input fields for user details such as name, email, password, birthday, biography, and interests. These are grouped and styled using JPanel containers with BoxLayout, maintaining alignment with the project's mockups and visual identity.

This meticulous design ensures that the interface not only meets functional requirements but also enhances the user experience, offering clear flows for critical processes such as account creation and access.

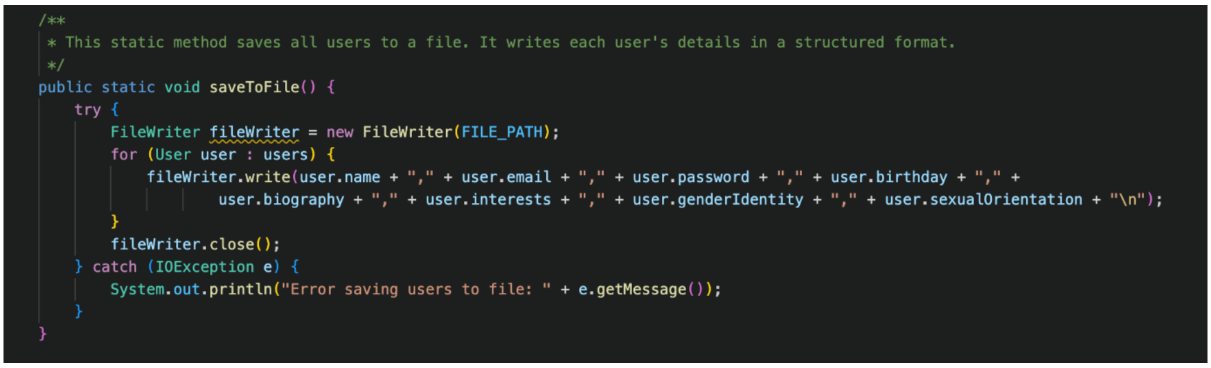
Additionally, by following the architecture established in the UML diagrams, these graphical screens faithfully represent the sequence and activity diagrams, transitioning the abstract system behaviors into tangible interactions.



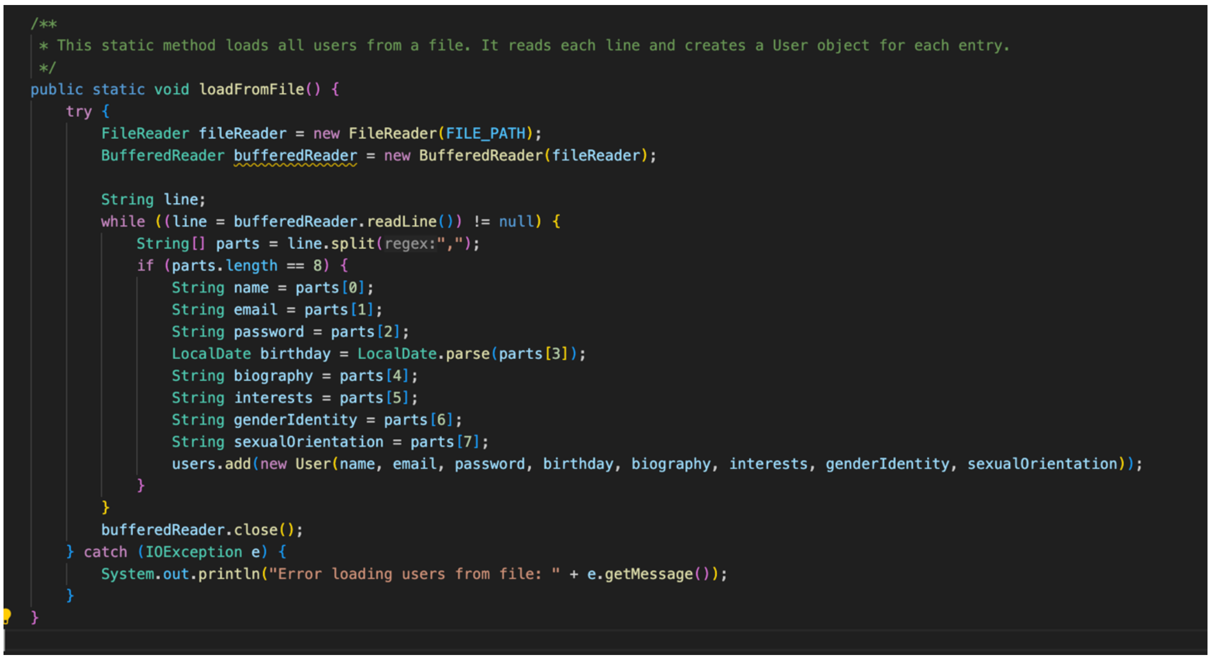


<u>FILE STORAGE:</u>

The file saving functionality was implemented using basic Java file handling classes, such as `FileWriter` and `BufferedReader`. In the User class, a static list called `users` was added to store all registered users. To save user data, the static method `saveToFile` was created, which iterates through the `users` list and writes each user's attributes to the `users.txt` file, separating values with commas for easy parsing later. Similarly, the static method loadFromFile was implemented to read the file line by line, split the data using the `split(",")` method, and create User objects with the extracted values, adding them to the `users` list. This ensures that user data persists between program executions, allowing previously registered users to be available when the application starts.



```java
/**
 * This static method saves all users to a file. It writes each user's details in a structured format.
 */
public static void saveToFile() {
    try {
        FileWriter fileWriter = new FileWriter(FILE_PATH);
        for (User user : users) {
            fileWriter.write(user.name + "," + user.email + "," + user.password + "," + user.birthday + "," +
                    user.biography + "," + user.interests + "," + user.genderIdentity + "," + user.sexualOrientation + "\n");
        }
        fileWriter.close();
    } catch (IOException e) {
        System.out.println("Error saving users to file: " + e.getMessage());
    }
}
```



```java
/**
 * This static method loads all users from a file. It reads each line and creates a User object for each entry.
 */
public static void loadFromFile() {
    try {
        FileReader fileReader = new FileReader(FILE_PATH);
        BufferedReader bufferedReader = new BufferedReader(fileReader);

        String line;
        while ((line = bufferedReader.readLine()) != null) {
            String[] parts = line.split(regex:",");
            if (parts.length == 8) {
                String name = parts[0];
                String email = parts[1];
                String password = parts[2];
                LocalDate birthday = LocalDate.parse(parts[3]);
                String biography = parts[4];
                String interests = parts[5];
                String genderIdentity = parts[6];
                String sexualOrientation = parts[7];
                users.add(new User(name, email, password, birthday, biography, interests, genderIdentity, sexualOrientation));
            }
        }
        bufferedReader.close();
    } catch (IOException e) {
        System.out.println("Error loading users from file: " + e.getMessage());
    }
}
```

In the users.txt file, each user's information is stored as a single line in a comma-separated format. The attributes saved include the user's name, email, password, birthday, biography, interests, gender identity, and sexual orientation. For example, a line in the file might look like this: `John,john@example.com, password, 1990-01-01, Short bio, Reading, Cisgender Man, Heterosexual`. This format ensures that all user data is organized and can be easily parsed when the application reads the file. Each line represents one user, and the attributes are separated by commas to allow the program to split the data and recreate `User` objects during the loading process.

We handle errors with cycles so that if files are missing in the archives, we have a conditional that displays an error message.