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

Part 1 of the Object-Oriented Programming project

Selected project: Tinder (dating app)

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 develop a functional prototype of a dating application inspired by Tinder, using Java and Object-Oriented Programming (OOP) principles. The project aims to promote safe and dynamic user interaction by implementing core functionalities such as registration, profile management, user interaction (likes/dismiss), match notifications. All structured under a modular and maintainable architecture.

Specific Objectives:

- To design and implement a secure user authentication system that supports account creation and login functionalities.
- To allow users to personalize their profiles by editing personal information, uploading photos, and selecting interests and lifestyle preferences.
- To develop an interaction system that enables users to like or dismiss other profiles based on their preferences.
- To implement a match detection mechanism that notifies users when mutual interest occurs, enhancing engagement within the app.

- To model the application's architecture using object-oriented principles such as encapsulation, inheritance, and modularity, ensuring that the code is maintainable and extensible.

- To visualize the system’s structure and logic through UML class and sequence diagrams that reflect the interaction between components.

Functional Requirements:

- Users will be able to register, log in, and edit their profile. This includes uploading a photo, writing a bio, selecting interests, and modifying their personal information.
- Users will be able to view other profiles and perform actions such as "Interested" or "Not Interested."
- When one of the two users makes a positive interaction (Interested/Like), the other user will be notified so they have this information and can view their profile, indicating whether they are compatible.

Non-Functional Requirements:

- The application must have a clear, intuitive, and easy-to-navigate interface, accessible from a variety of devices.
- Implementar mecanismos de seguridad para los usuarios como rectificación de ingreso de cuentas y protección contra amenazas comunes.
- The design should be modern, pleasing, and consistent, using a color palette and typography that promote a comfortable visual experience.
- The application must maintain rapid response times and support progressive user growth without compromising the experience or system stability.

USER STORIES:

USER STORY

Title: User registration	**Priority:** High	**Estimate: 6 weeks**

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: 6 weeks**

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: 6 weeks**

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: 6 weeks**

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: 6 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: 6 weeks**

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: 6 weeks**

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: 6 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: 6 weeks**

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.

Changes made:

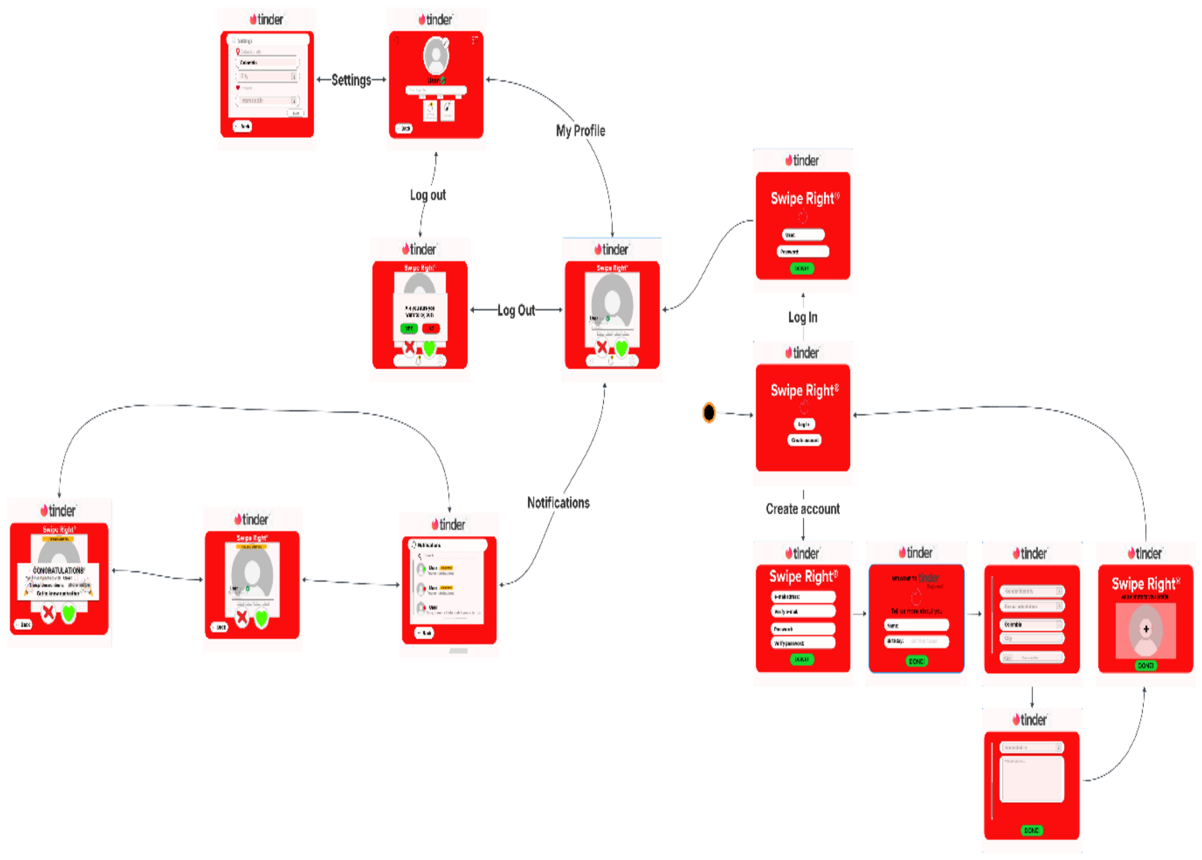
- **For workshop 2:**

We decided to change the user stories, because we discarded the chat and report, also we added a notification option to facilitate the match. Also, thanks to the Mockups and the UML diagram, we found ourselves in need of adding more stories to be able to organize ourselves better.

- **For workshop 3:**

It was decided to make changes to the user stories presented since, in terms of functionality and needs seen in the diagrams made, it was necessary to add the Registered User, Complete new user information and Log out stories. In addition, modifications were made to the existing user stories based on the required program update.

MOCKUPS:



CLICK HERE TO SEE THE MOCKUPS

Changes made:

- **For workshop 2:**

In the presented mockup, changes were made compared to the first, it was done in a more organized and easy to understand way, prioritizing a graphic part that is pleasant for its readers, going through the Frame 3 line we can see the creation of a user account even requesting new values such as password, biography and add tags, the location activation frame was eliminated, on the other hand, the frame [Star 4.0] was added the option to log out, view notifications and report user, frames [4.1, 4.2, 4.3] are created for a better fluidity in the program, they were added compared to the previous mockUp where we can view a frame to see notifications, a cleaner frame to view my profile and a frame to log out, within viewing my profile the options to make modifications to our profile such as interests or tags are also added, also modify profile photo.

- **For workshop 3:**

To maintain a clean and minimalistic project structure, certain features such as user reporting, multiple photo uploads, and complex interaction methods were intentionally omitted. The welcome screen remains intuitive, offering straightforward options to either create an account or log in. When creating an account, users are prompted to provide essential but useful information that facilitates relevant interactions.

The home screen presents a suggested profile with simple interaction buttons: like or dismiss. Additional screens include notifications, profile settings, and logout, all designed with focused and specific functionality. Regarding user interactions, actions such as likes are stored and will only trigger a notification if a mutual interest (match) occurs. This streamlined logic simplifies the user experience and backend processing.

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>Changes made:</u>

- **For workshop 2:**

We decided to add some classes and change some responsibilities on the CRC Cards, for better organization, the idea came from the UML diagrams, it helped us realize that the classes needed to be better broken down.

Also, we decided to discard the chat on the app, because we considered that if we put a chat, we wouldn't be able to finish all the requirements.

- **For workshop 3:**

While refining our application design, we applied fundamental SOLID principles to improve structure and clarity. We focused on ensuring that each class had a single, well-defined responsibility. As the project scope became clearer, several components originally created were removed to align the implementation with actual functional needs and available development time:

The MatchSystem class was removed, and its responsibilities were redistributed between the User and App classes, simplifying interaction and matching logic. The

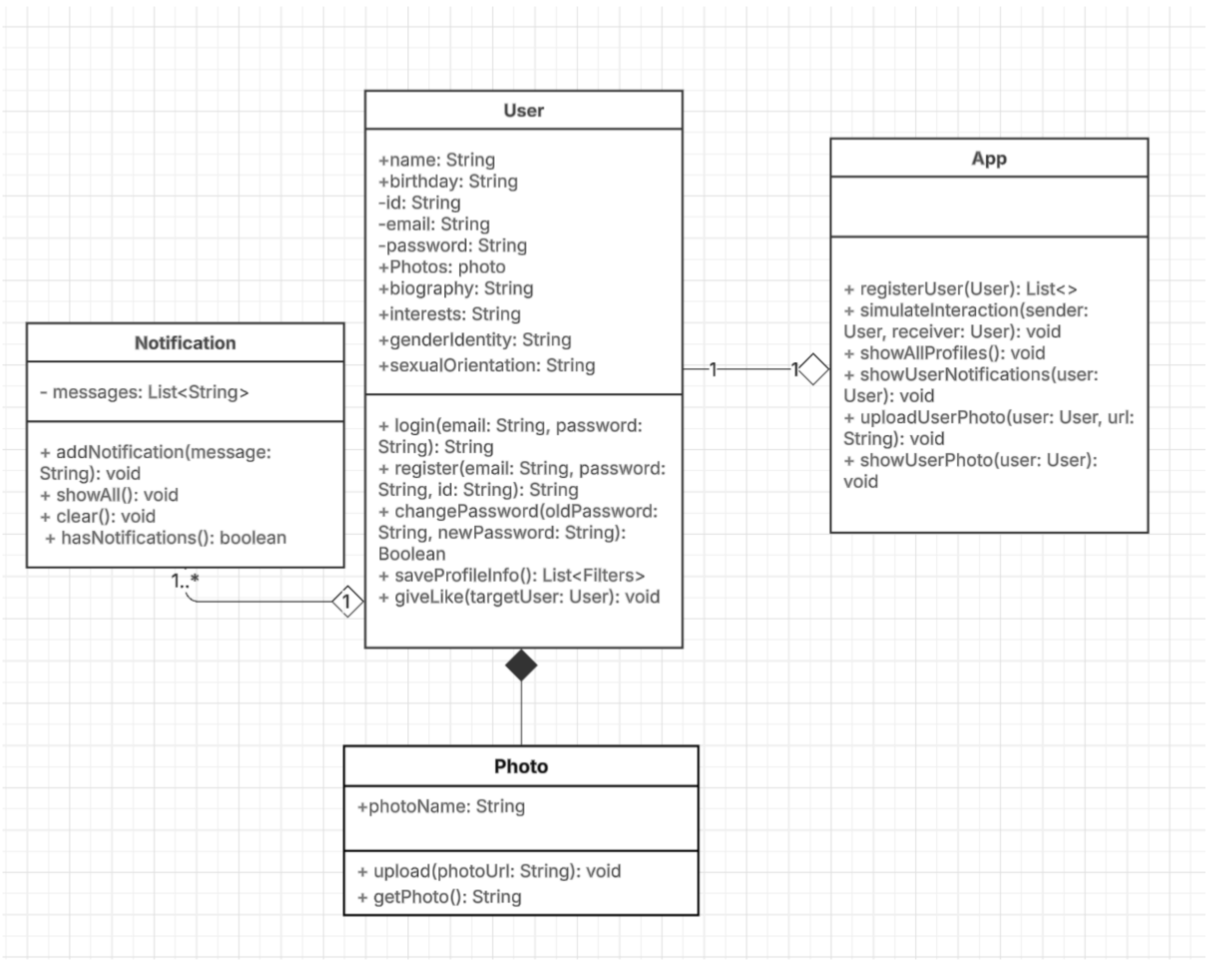
Reports, Admin, and Account classes were also removed, as the application was distributed in a simpler and clearer way. Instead, two necessary classes were added:

Photo, integrated as a composition within the User class, manages the storage and retrieval of the profile picture. App was introduced to coordinate user registration, interaction management, notification display, and photo management.

Part 2 of the Object-Oriented Programming project

UML DIAGRAMS:

Class diagram:


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
Photo
+photoName: String
+ upload(photoUrl: String): void
+ getPhoto(): String
1
1
1..*
1

CLLICK HERE TO SEE THE CLASS DIAGRAM

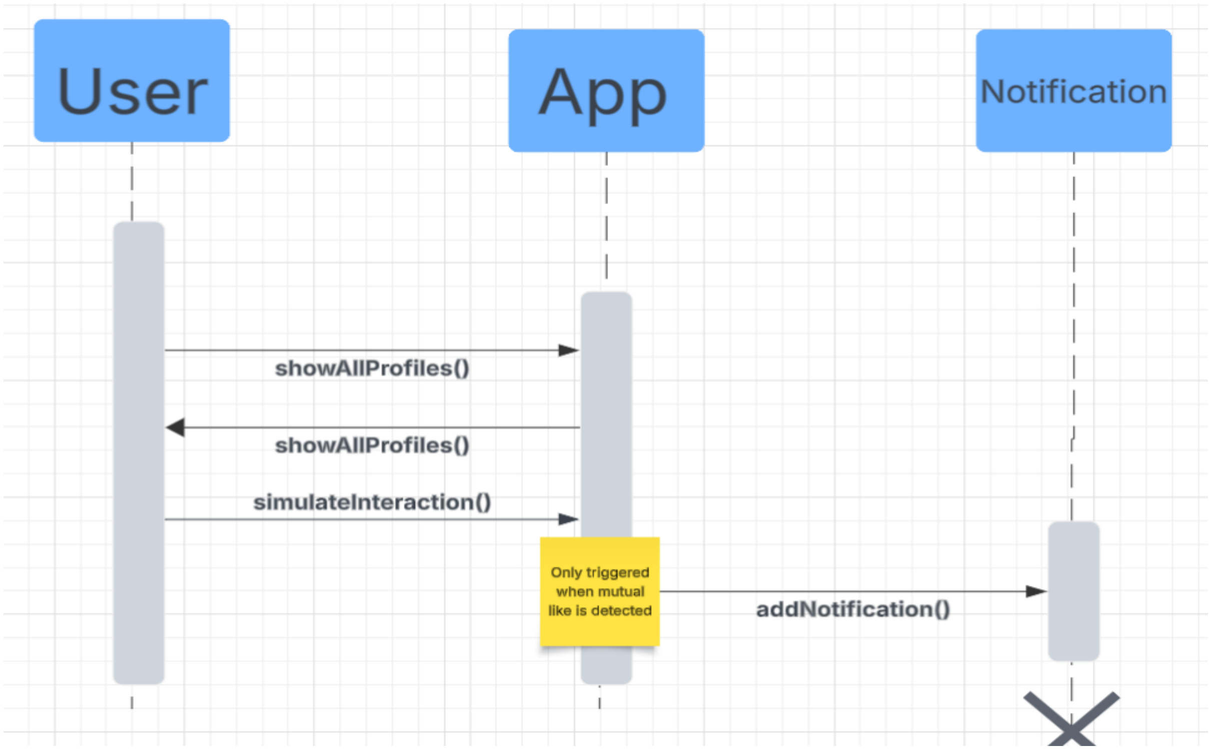
Changes made:

- **For workshop 3:**

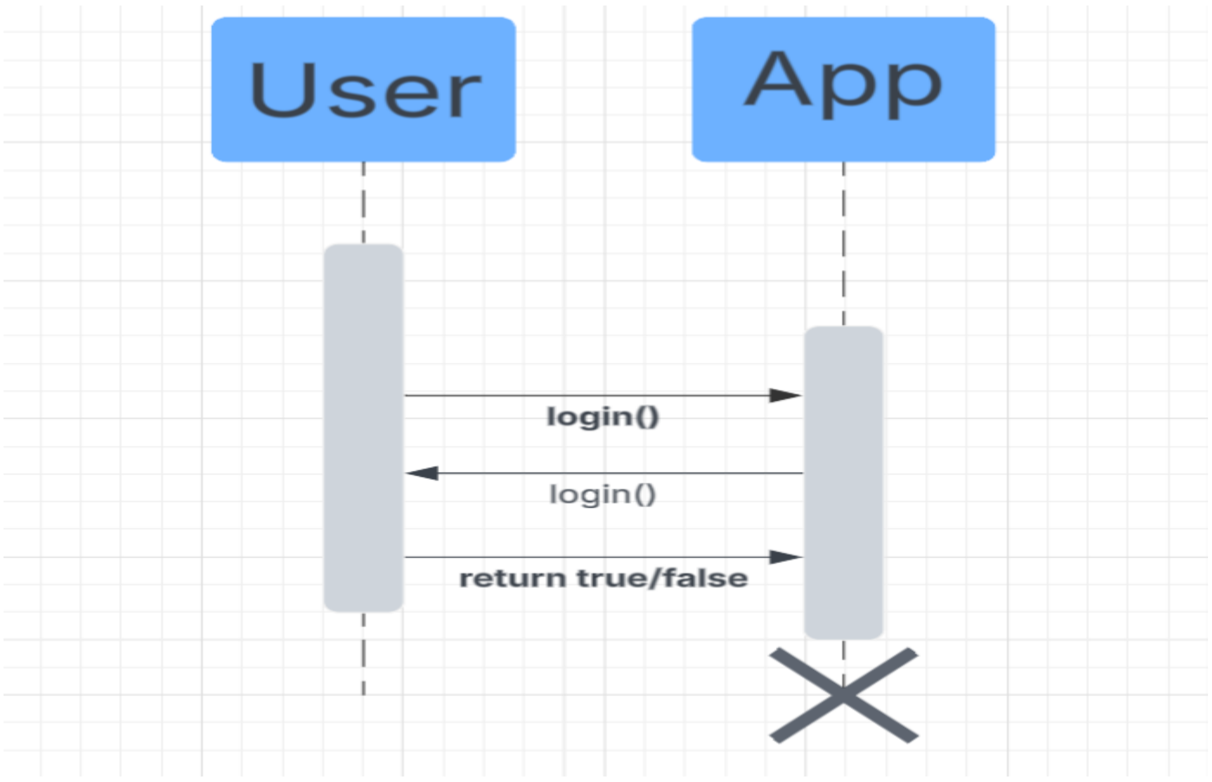
Thanks to SOLID principles and the change in core responsibilities we had proposed, we decided to discard the report. Along with this, the admin class should also be discarded, since it makes no sense to have the admin class if it doesn't have a specific method. This also aims to have better development investment in core functionalities.

Sequence diagram:

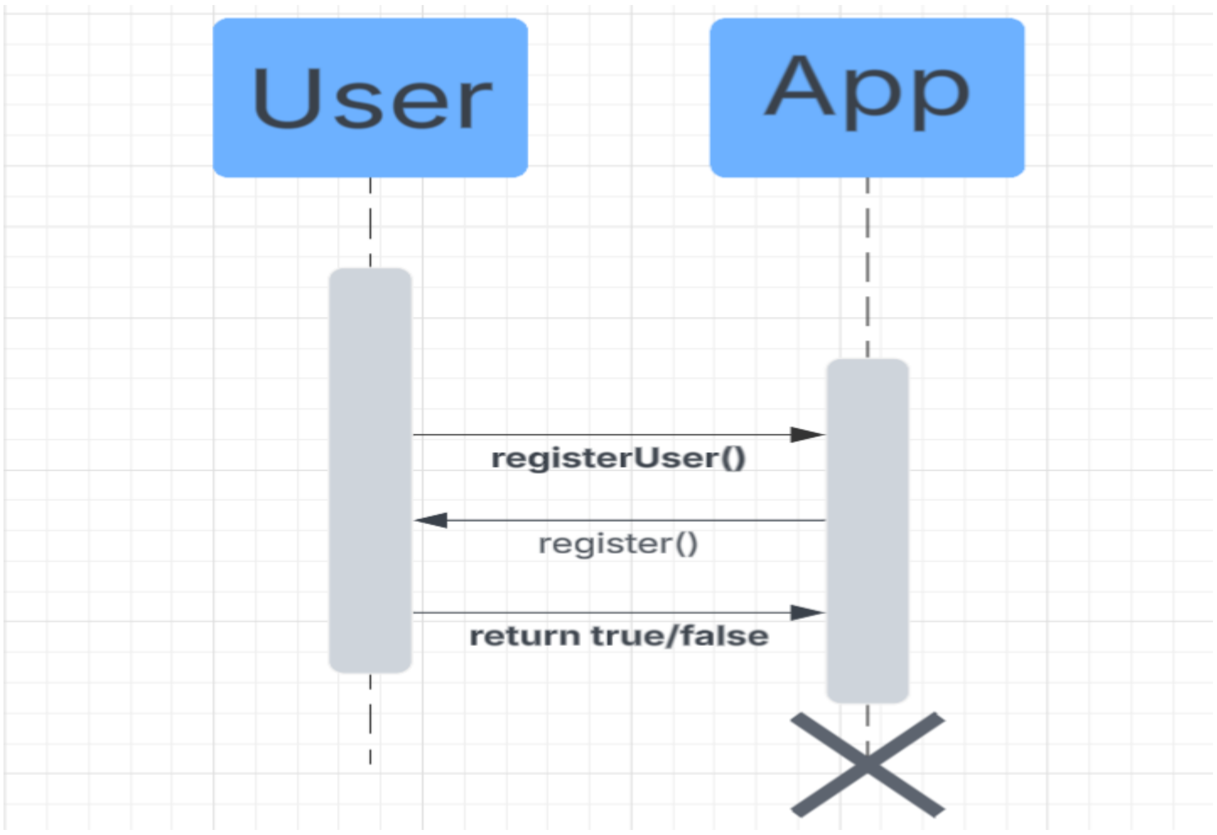
Mutual like:



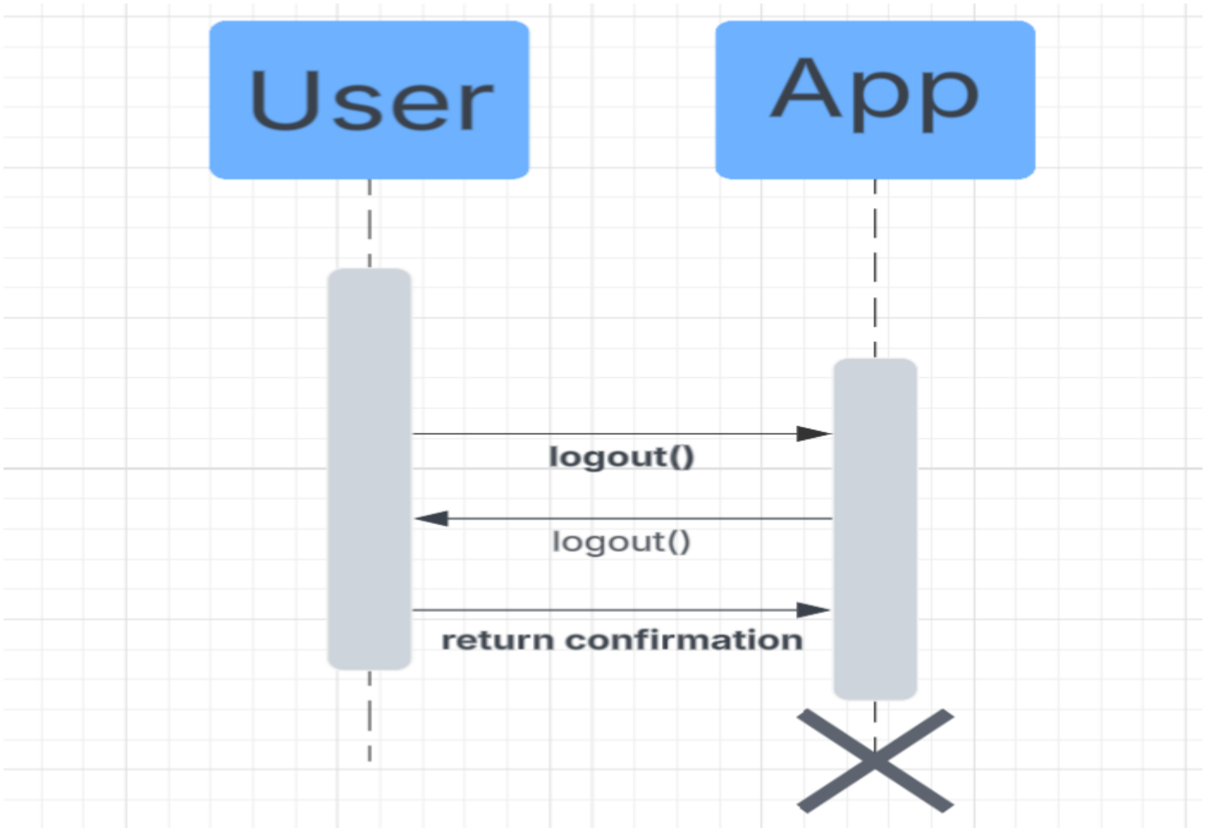
Log in:



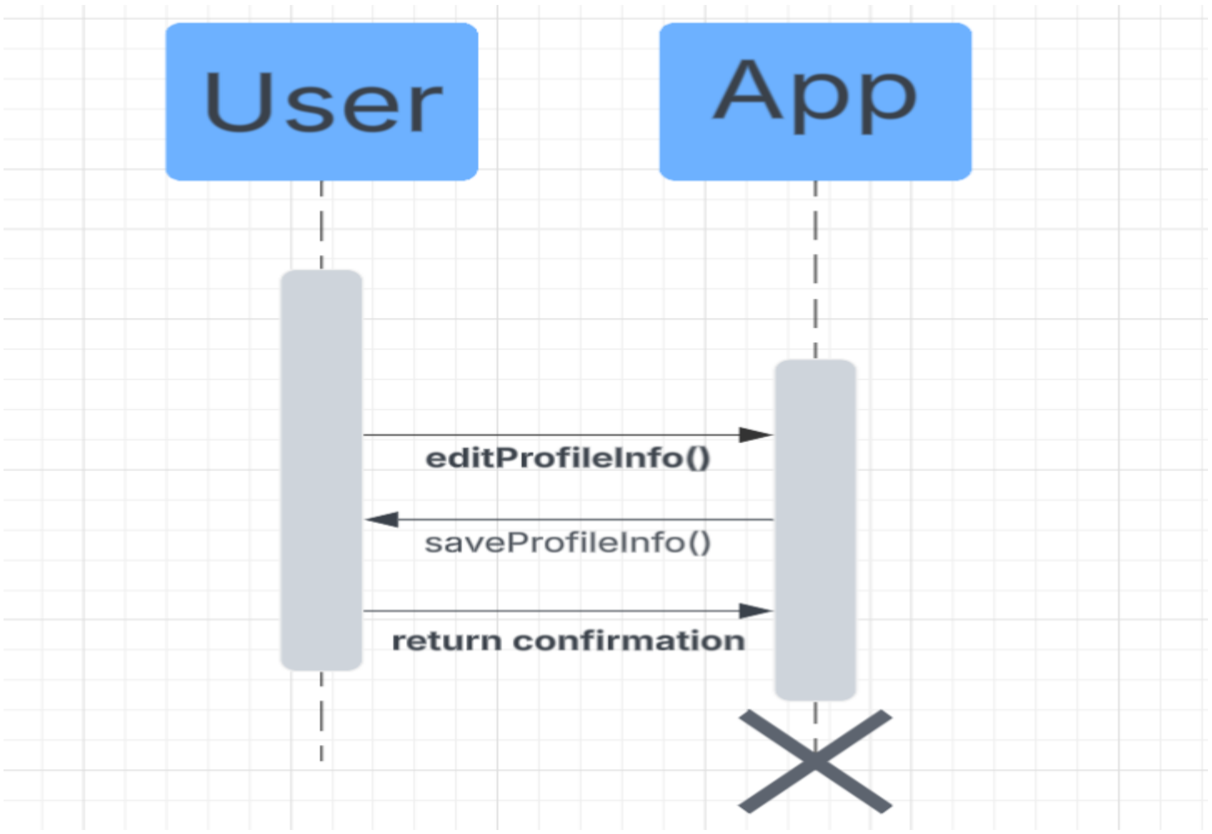
User registration:



Log out:



Edit profile:



Changes made:

- **For workshop 3:**

Se realizaron ajustes importantes en los diagramas de secuencia para mantener coherencia con el nuevo diseño del sistema:

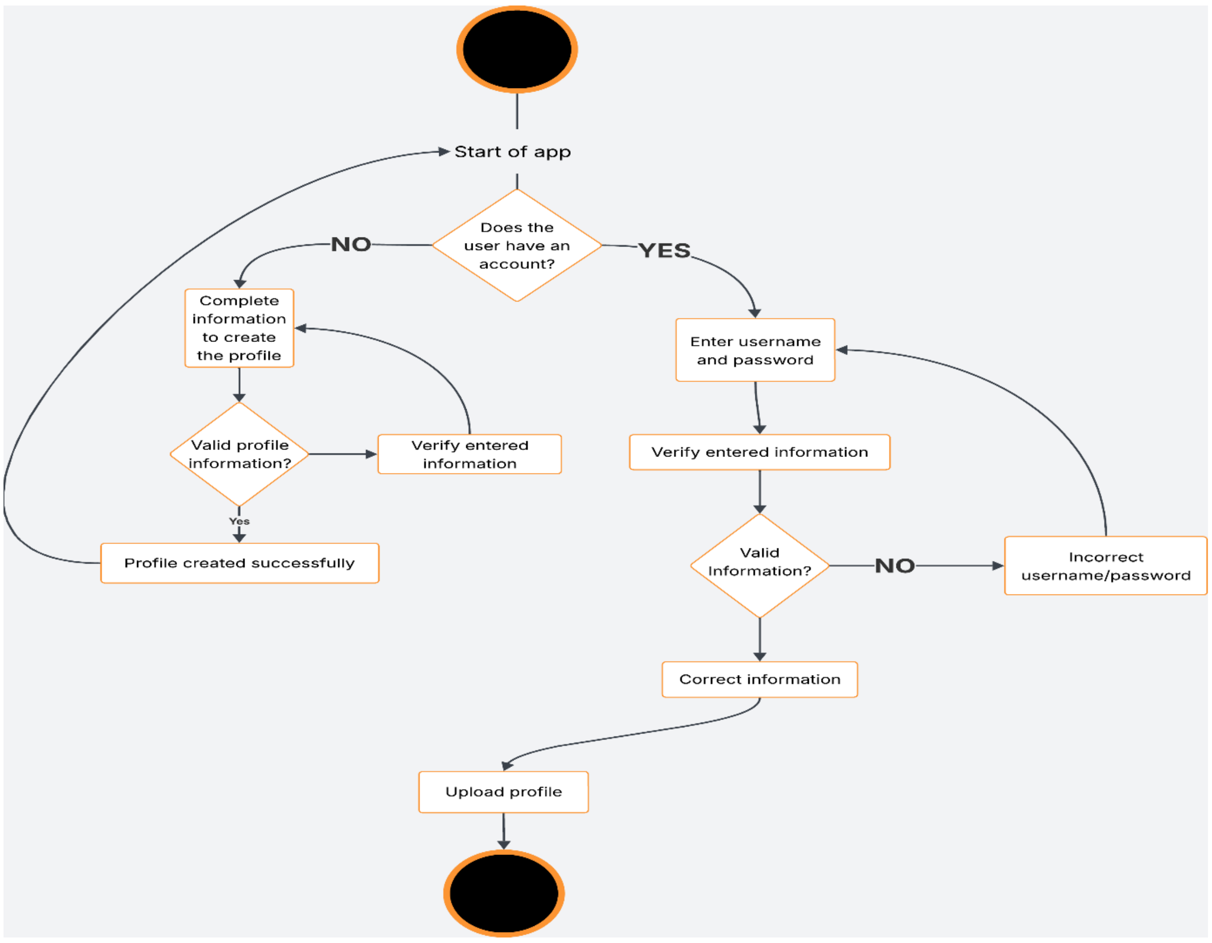
Se eliminaron los diagramas de Reports, Notifications y MatchSystem, ya que estas clases dejaron de existir en el modelo actual por cambios en los requerimientos. En reemplazo, se crearon y reorganizaron diagramas que reflejan con precisión las funcionalidades implementadas hasta ahora:

- **Login**: muestra cómo el usuario accede con login().
- **Log Out**: refleja el flujo de cierre de sesión del usuario.
- **Edit Profile**: representa la actualización de datos personales con saveProfileInfo().
- **Mutual Like**: reemplaza a MatchSystem, simulando la lógica de like recíproco y notificación mediante simulateInteraction().

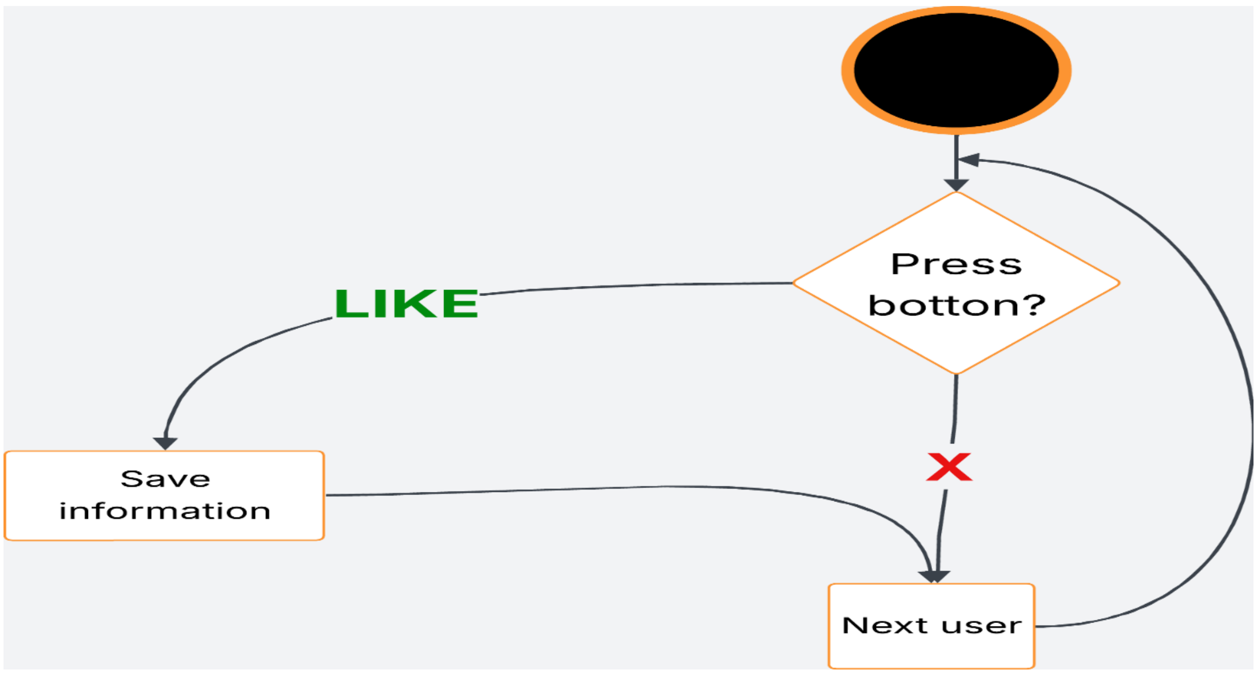
Estos cambios aseguran que los diagramas de secuencia estén alineados con el diagrama de clases, los diagramas de actividad y los flujos reales de la aplicación.

Activity diagram:

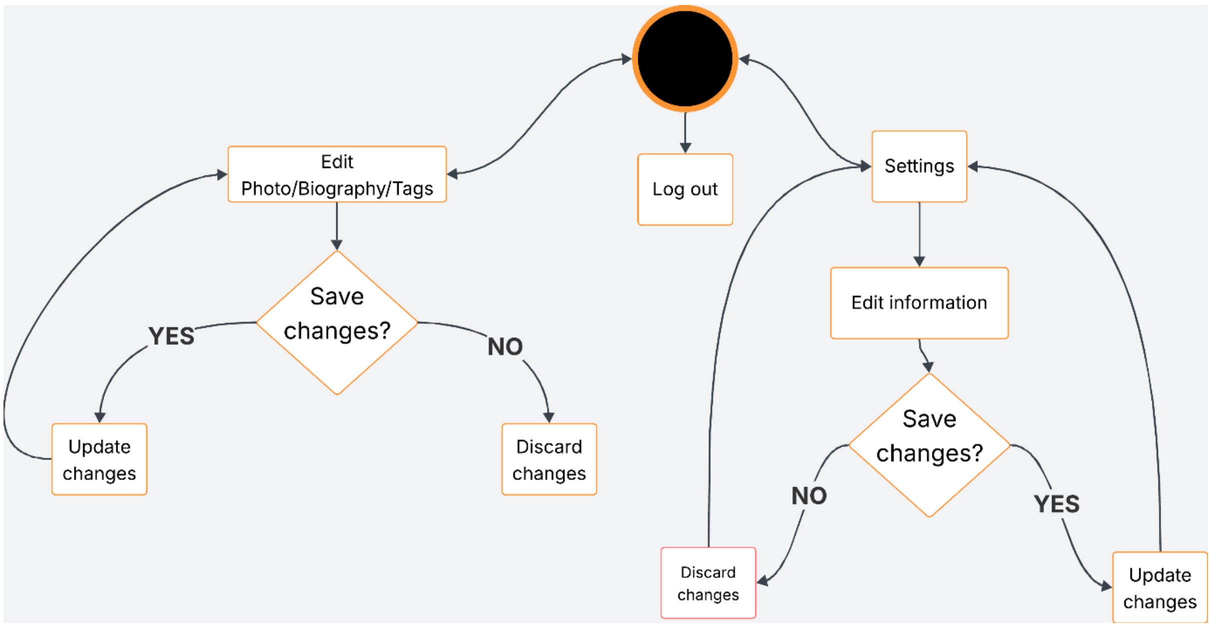
Log in or register:


Start of app
Does the user have an account?
NO
YES
Complete information to create the profile
Valid profile information?
Verify entered information
Yes
Profile created successfully
Enter username and password
Verify entered information
Valid Information?
NO
Incorrect username/password
Correct information
Upload profile

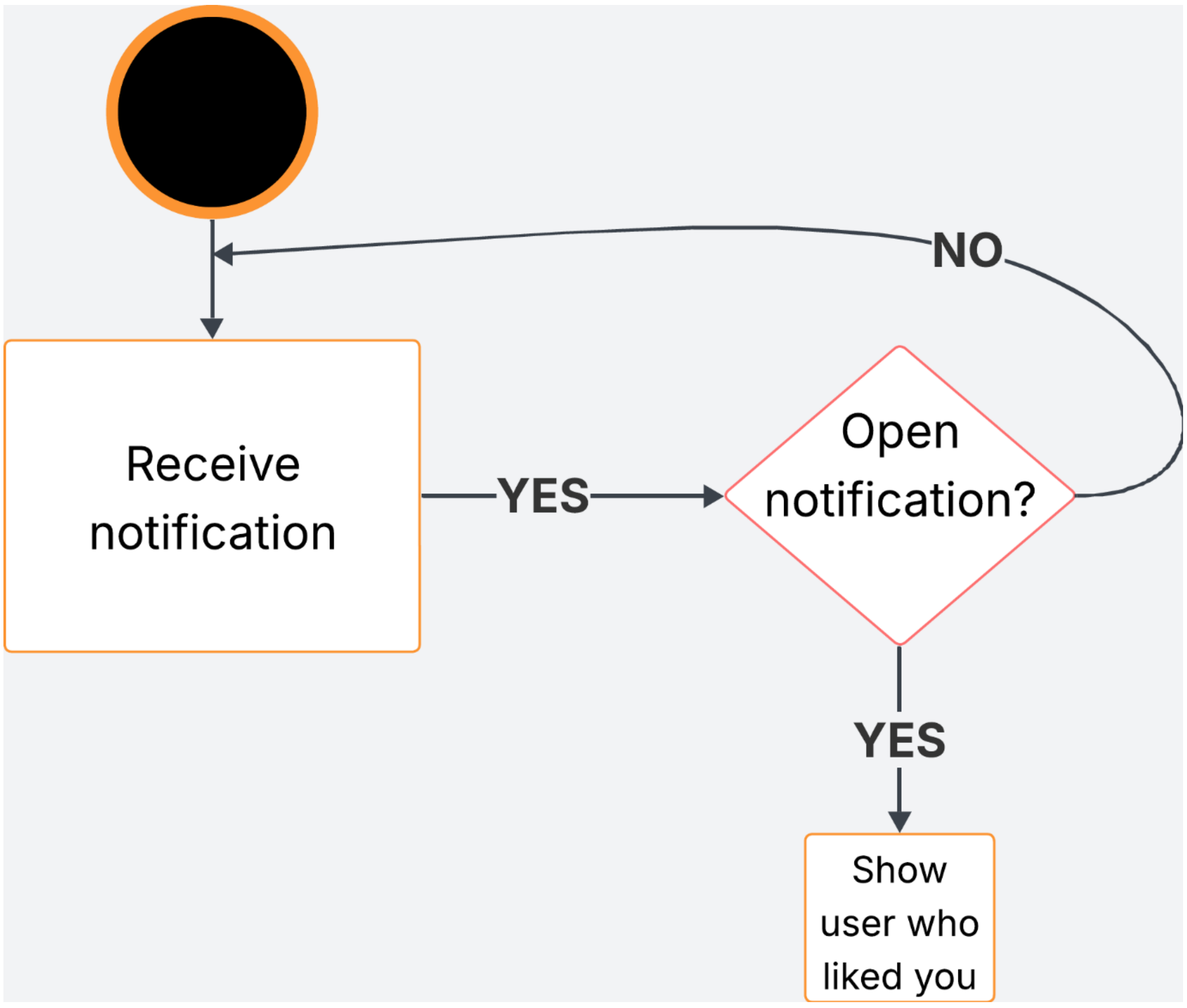
Frame principal:



My profile:



Notification:



<u>Changes made:</u>

- **For workshop 3:**

Among the changes made to this section, as part of the feedback, activity diagrams were added considering the main functionalities such as registration, log in, main screen, notifications, profile settings, having a clean and necessary integration for a correct implementation in the project.

IMPLEMENTATION PLAN FOR OOP CONCEPTS:

Implementation Plan for OOP Concepts

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

Encapsulation:

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

- <u>App:</u>
  Coordinates the system through public methods without exposing its internal logic.

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

- <u>Photo:</u>
  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.

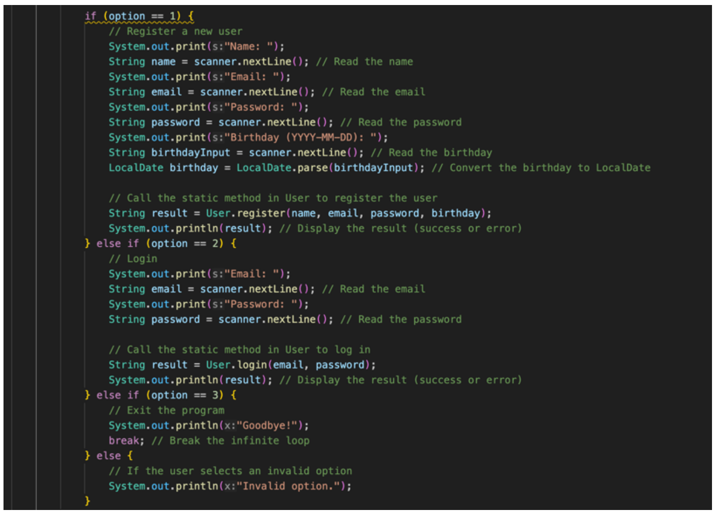
- Notification:
  - Data-based: Displays only relevant messages, hiding the structure or source of the data.
  - Behavioral: Exposes key actions such as displaying, adding, or clearing notifications, without revealing how they are stored or managed.

- 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

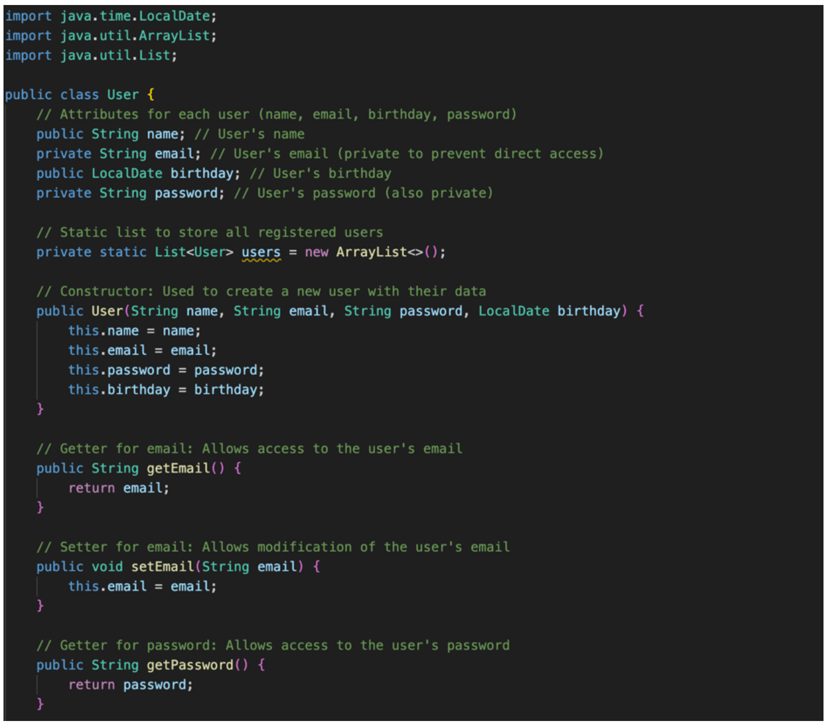
CLICK HERE:

Code snippets:



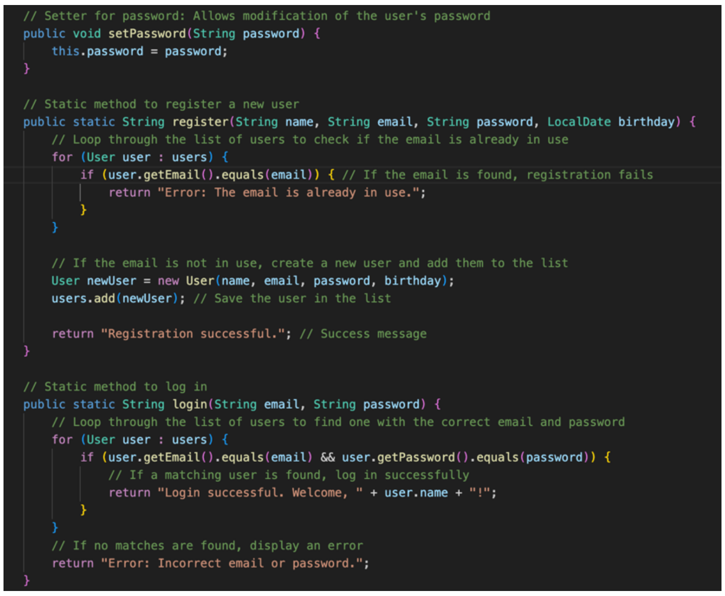
```java
if (option == 1) {
    // Register a new user
    System.out.print(s:"Name: ");
    String name = scanner.nextLine(); // Read the name
    System.out.print(s:"Email: ");
    String email = scanner.nextLine(); // Read the email
    System.out.print(s:"Password: ");
    String password = scanner.nextLine(); // 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.



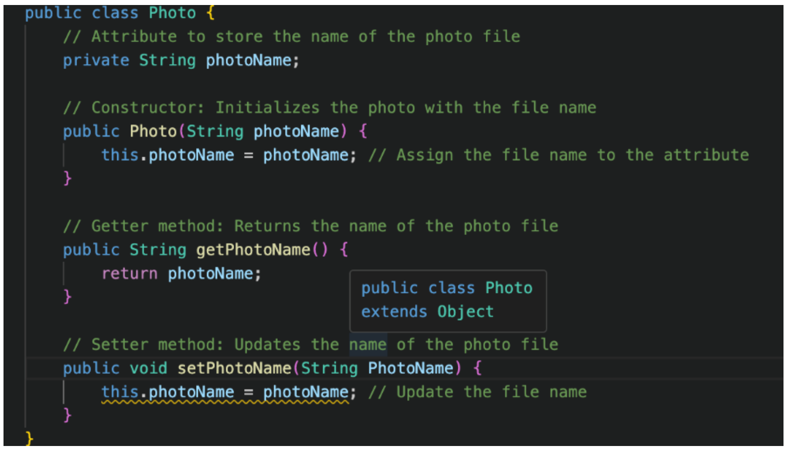
```java
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

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.



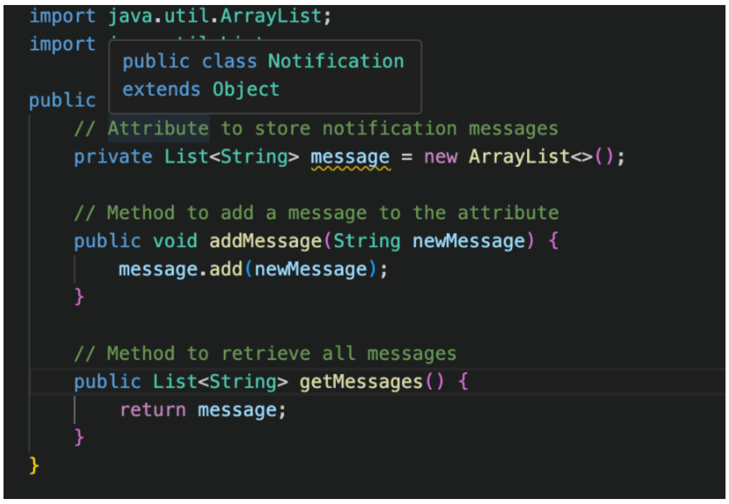
```java
    // Setter for password: Allows modification of the user's password
    public void setPassword(String password) {
        this.password = password;
    }

    // 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;
    }

    // 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
    // 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.

Part 3 of the Object-Oriented Programming project

SOLID-Focused Implementation:

Single responsibility:

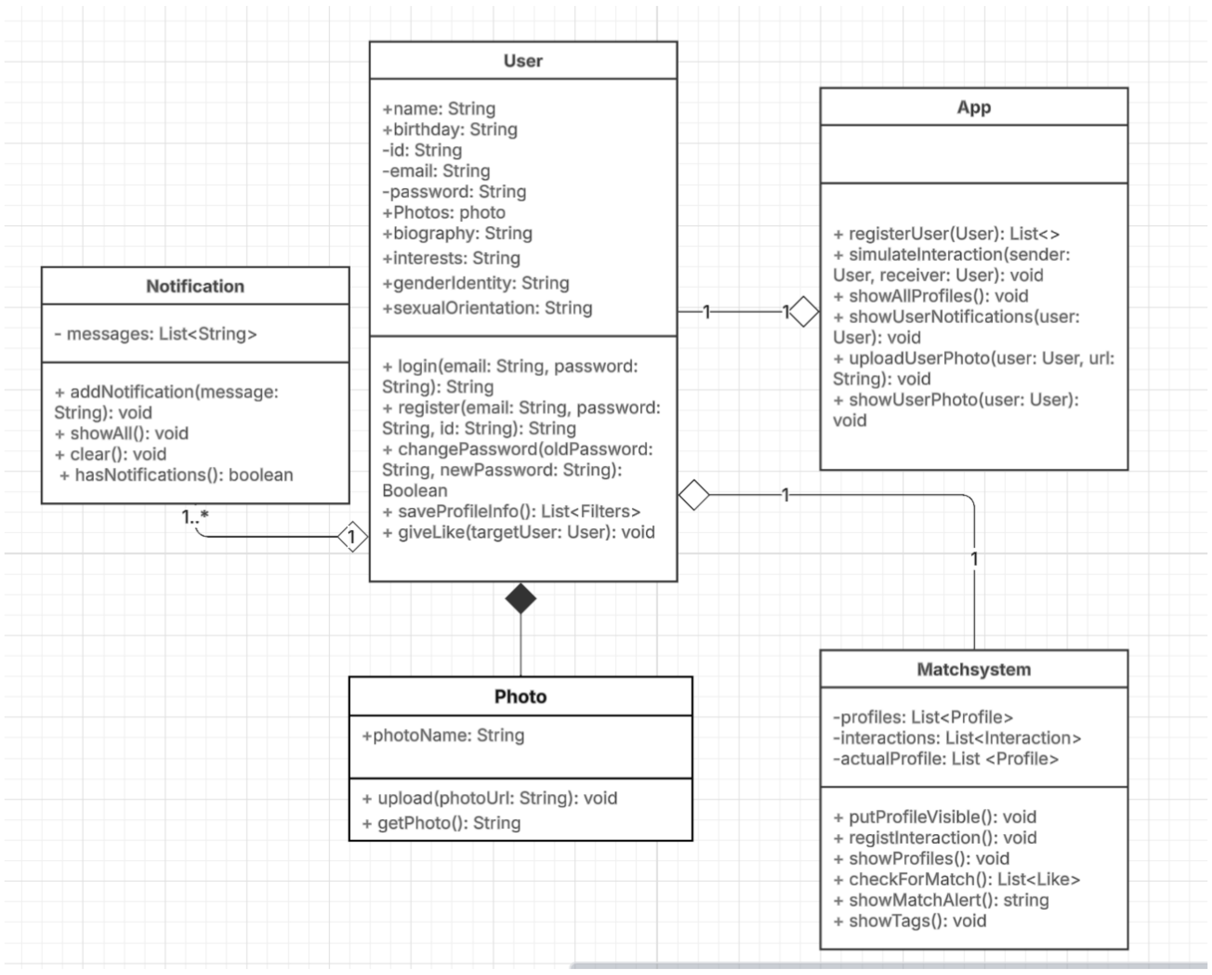
In the first principle of single responsibility, we organize classes with a single key purpose. In this case, by changing the diagram to focus on key responsibilities of the program, we apply the single responsibility principle without having to modify it again, since each class has its own purpose.

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
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
1
1
1..*
1
1
1

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.