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)

Objectives:

Develop a small-scale version of the Tinder app that allows for user matching, providing a visually appealing, intuitive and functional experience.

- **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 gives a positive interaction (Interested/like) the other user will be notified, so the person could see the profile of them.
    - Features to report and block users for inappropriate behavior.

- **Non-Functional Requirements**
    - The application must have a clear, intuitive, and easy-to-navigate interface, accessible from a variety of devices.
    - Implement mechanisms such as secure authentication, encryption of sensitive data, and protection against common threats. Content blocking, reporting, and control options should be included.
    - 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:**

User Story:

As a new user,
I want to register using email and password,
so that access the application.

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

USER STORY

Title: Complete user profile	**Priority:** High	**Estimate:**

User Story:

As a registered user,
I want to enter data such as my name, birthday, gender, sexual orientation, interests, lifestyle,
so that complete the creation of my profile and start the experience in the app.

Acceptance Criteria:
Given a user who has just successfully registered,
When access the app home page for the first time,
Then They are redirected to a profile creation completion screen where they can enter their name, date of birth, gender, sexual orientation, interests, and lifestyle before continuing to use the app.

USER STORY

Title: Upload images to the user's profile	**Priority:** High	**Estimate:**

User Story:

As a registered user,
I want to upload an image to my profile,
so that I personalize my profile and make it more attractive to other users.

Acceptance Criteria:
Given a user has successfully completed registration,
When you access the upload section and upload an image to your profile,
Then you select and upload an image from your device and receive confirmation when the image upload is successful.

USER STORY

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

User Story:

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

Acceptance Criteria:
Given an app user has completed creating their profile and has watched or skipped the tutorial,
When you start interacting with the profiles shown,
Then swipe or tap buttons to like or dismiss, and if a match is made, receive a notification of success.

USER STORY

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

User Story:

As a user of the app,
I want to receive a notification when someone likes my profile,
so that I can check out the person who liked me and decide whether to interact with them.

Acceptance Criteria:
Given that another user has liked my profile,
When I receive a notification about that like,
Then I should be able to access the liker’s profile directly from the notification to decide whether to match

USER STORY

Title: Report Inappropriate User	**Priority:** High	**Estimate:**

User Story:

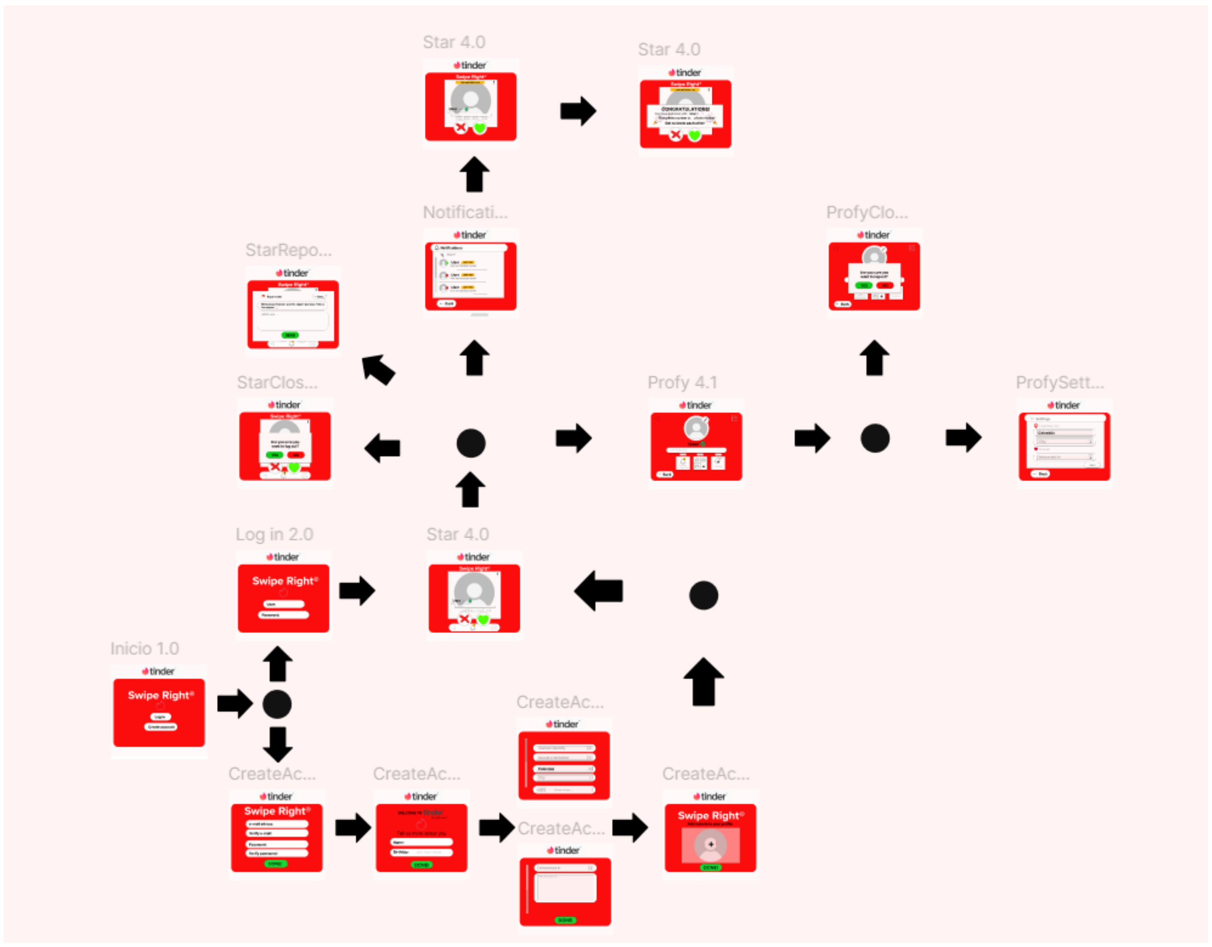
As a user of the app,
I want to be able to report other users for inappropriate behavior,
so that the platform can review and take action to maintain a safe community.

Acceptance Criteria:
Given I am logged in and viewing another user's profile,
When I click on the "Report" button and write the reason for the report,
Then the report is submitted to the system with the reported user's ID, my ID, and the selected reason.
And I receive confirmation that my report was successfully submitted.

We decided to change the user stories, because we discarded the chat and 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.

<u>Mockups</u>:



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.

https://www.figma.com/design/Fjw8QTU486SoliSIwdrVPO/Mockup-WorkShop?node-id=34-2&t=xkDMfsgrW6raHPS9-1

CRC Cards:

Class Name: Account	
Responsibilities: - Manage account (register and login) - Change Password	**Collaborators:** - User - Admin

Class Name: User	
Responsibilities: - Edit personal information - Upload profile images	**Collaborators:** - Account - MatchSystem

Class Name: Admin	
Responsibilities: - View Reports - Block Users - Delete Profiles - Admin Level - Admin Name	**Collaborators:** - Account - Reports

Class Name: Reports

Responsibilities:	**Collaborators:**
- Submit Reports - Report ID - Reported Users - Reason	- User - Admin

Class Name: MatchSystem

Responsibilities:	**Collaborators:**
- Show other user profiles - Register like/dislike actions - Represent a mutual interest between two users - Detect Matchs	- User - Notification

Class Name: Notifications

Responsibilities:	**Collaborators:**
- Create a space when the user could see, who give them like - Allow users to see who give them a like	- User - MatchSystem

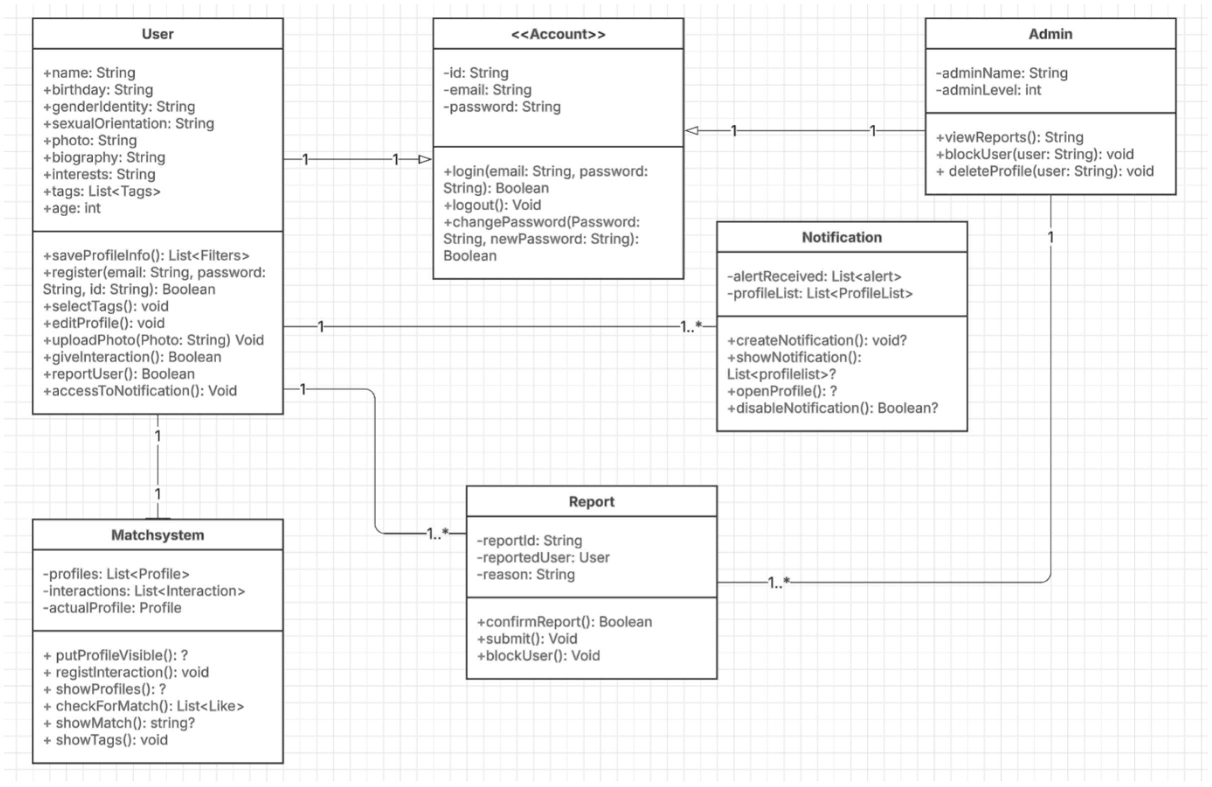
We decided to add some classes and change some responsibilities on the CRC Cards, for a better organization, the idea came from the UML diagrams, it helped us realize that the classes needed to be better broken down.

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.

Part 2 of the Object-Oriented Programming project

UML DIAGRAMS:

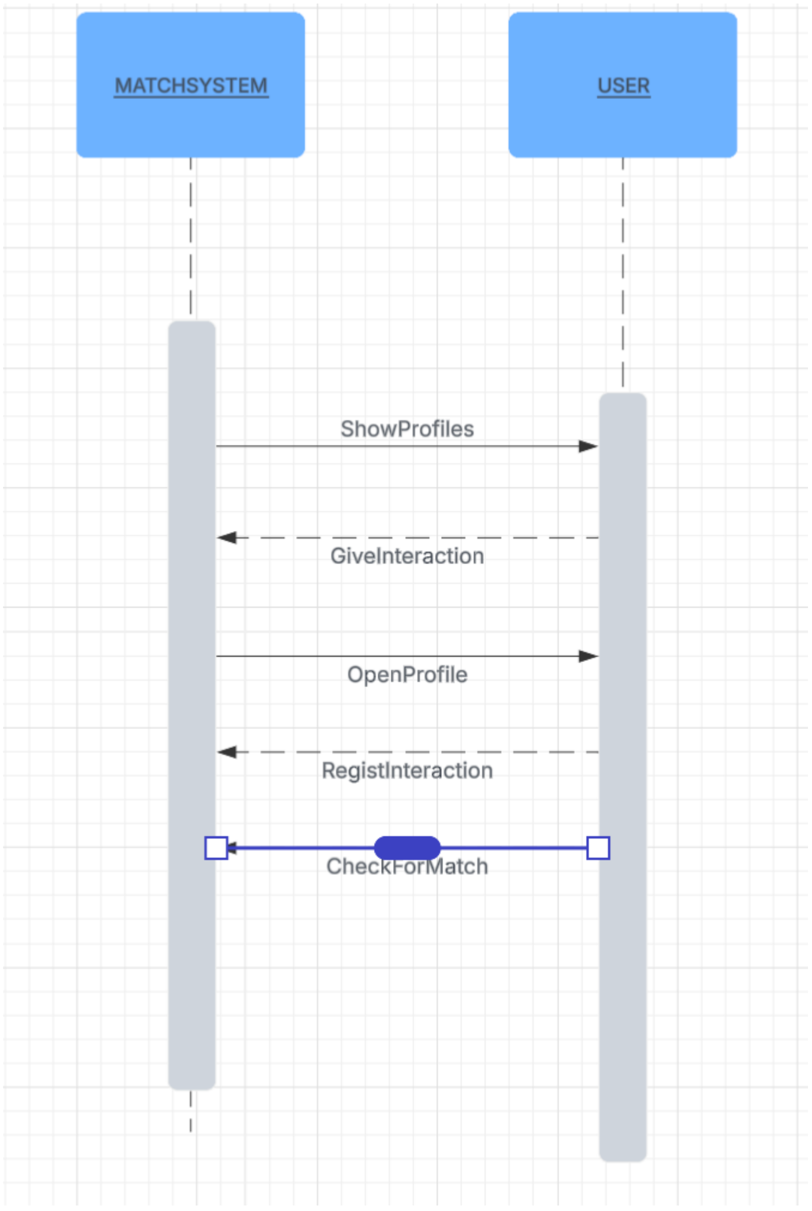
CLASS DIAGRAM


User
+name: String
+birthday: String
+genderIdentity: String
+sexualOrientation: String
+photo: String
+biography: String
+interests: String
+tags: List<Tags>
+age: int
+saveProfileInfo(): List<Filters>
+register(email: String, password: String, id: String): Boolean
+selectTags(): void
+editProfile(): void
+uploadPhoto(Photo: String) Void
+giveInteraction(): Boolean
+reportUser(): Boolean
+accessToNotification(): Void
<<Account>>
-id: String
-email: String
-password: String
+login(email: String, password: String): Boolean
+logout(): Void
+changePassword(Password: String, newPassword: String): Boolean
Admin
-adminName: String
-adminLevel: int
+viewReports(): String
+blockUser(user: String): void
+ deleteProfile(user: String): void
Notification
-alertReceived: List<alert>
-profileList: List<ProfileList>
+createNotification(): void?
+showNotification(): List<profilelist>?
+openProfile(): ?
+disableNotification(): Boolean?
Matchsystem
-profiles: List<Profile>
-interactions: List<Interaction>
-actualProfile: Profile
+ putProfileVisible(): ?
+ registInteraction(): void
+ showProfiles(): ?
+ checkForMatch(): List<Like>
+ showMatch(): string?
+ showTags(): void
Report
-reportId: String
-reportedUser: User
-reason: String
+confirmReport(): Boolean
+submit(): Void
+blockUser(): Void
1
1
1
1
1
1..*
1
1..*
1
1
1
1..*

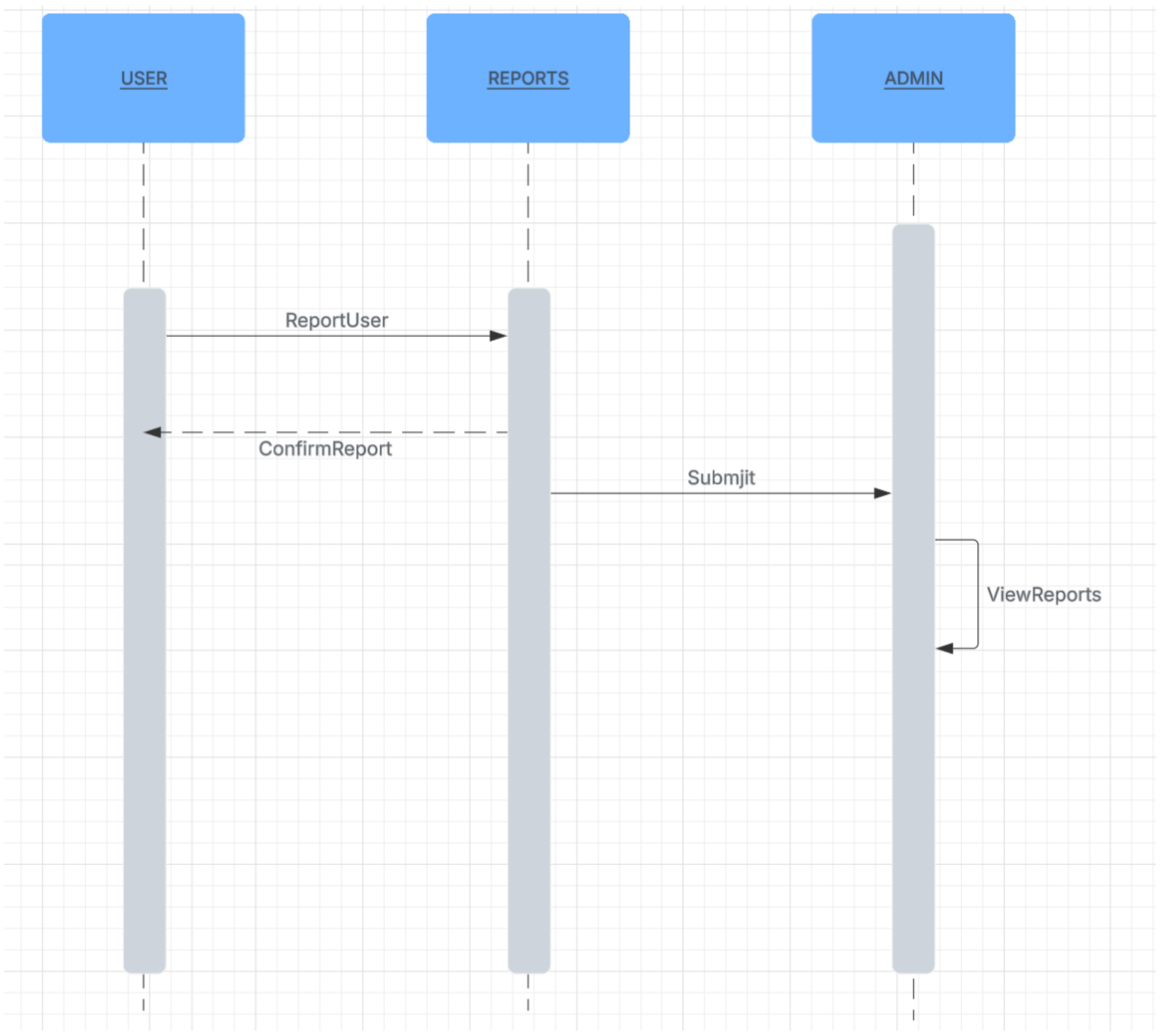
(Made in: https://lucid.app/lucidchart/03ad0193-ff99-4b08-9336-efa20bfca03c/edit?page=HWEp-vi-RSFO&invitationId=inv_c13517f7-c943-4c69-a93a-945adf5c3ba6)

SEQUENCE DIAGRAM

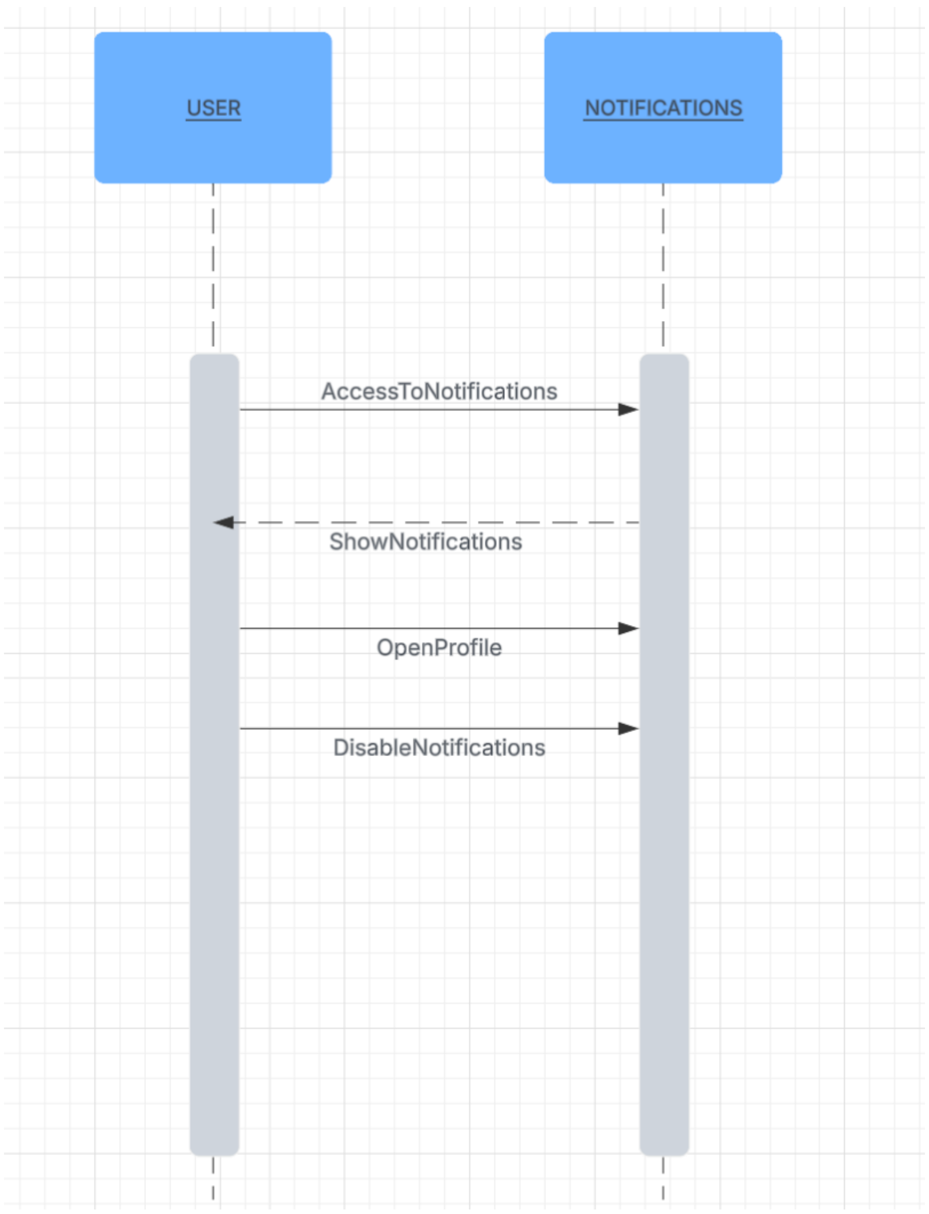
INTERACTIONS



REPORTS


USER
REPORTS
ADMIN
ReportUser
ConfirmReport
Submjit
ViewReports

NOTIFICATIONS



IMPLEMENTATION PLAN FOR OOP CONCEPTS:

ENCAPSULATION:

Our code implements encapsulation with access modifiers (private, public) in all class attributes. For example:

Attributes such as email and password in the User class are marked as private. These attributes can only be accessed or modified through public getter and setter methods, which ensures controlled access and data integrity.

Encapsulation also allows us to hide internal logic from other classes, such as password validation or profile visibility.

INHERITANCE:

We use inheritance to define a clear hierarchy in the system:

The abstract class Account defines common attributes and methods like login(), logout(), and changePassword().

Two concrete subclasses, User and Admin, inherit from Account, User extends it with behavior such as editing profiles, liking users, and chatting. Admin adds administrative capabilities like blockUser(). This promotes code reuse and clarity by separating shared and specialized behavior.

POLYMORPHISM

Polymorphism is not used in this implementation. We intentionally avoid polymorphism because the behavior of each class is clearly defined and does not require method overriding or overloading. Every role in the system has distinct responsibilities, and there is no need for shared method names with different behaviors. This decision simplifies the architecture and avoids unnecessary complexity for this scale of application.

<u>THE WORK IN PROGRESS CODE IS IN THE GITHUB</u>

<u>CLICK HERE:</u>

https://github.com/Foulsito/TinderApp/tree/main/Workshop_2/Code%20of%20TinderApp

<u>UML DIAGRAMS AND DESING RATIONALE</u>

1. User ↔ UML Class: User

Code:
In the User.java file, we can see:

Attributes such as name, birthday, gender, interests, etc., matching the class diagram.

Methods like editProfile() and uploadPhoto() are also declared, just as defined in the UML class and CRC card.

UML Alignment:
The User class inherits from Account as per the UML diagram and CRC cards. It is also related to MatchSystem, Notifications, and Reports.

2. Account ↔ UML Class: Account (Abstract)

Code:

The Account.java file shows email, password, and methods like login(), logout(), and changePassword() as expected.

UML Alignment:

Clearly matches the abstract class from the UML, which is extended by both User and Admin.

3. Admin ↔ UML Class: Admin

Code:

The Admin.java class implements methods such as viewReports() and blockUser(), as described in the UML and CRC card.

UML Alignment:

Fully corresponds to the Admin subclass of Account, managing reports and users.

4. MatchSystem ↔ UML Class: MatchSystem

Code:

Methods like registerInteraction(), checkForMatch(), and getSuggestionsFor() are found in the code and mirror the UML diagram.

UML Alignment:

Matches its CRC card: it registers likes/dislikes, shows profiles, and interacts with the Notification class.

5. Reports ↔ UML Class: Reports

Code:

The code includes functionality for submitting a report (submit()), and storing the user ID, reported user, and reason.

UML Alignment:

Mirrors the UML class and CRC card with fields reportID, reportedUser, reason, and related methods

6. Notifications ↔ UML Class: Notifications

Code:

The logic for generating and retrieving likes-based notifications is being implemented, including a list of users who liked the current user.

UML Alignment:

Aligns with the Notifications class which collaborates with User and MatchSystem.