TINDER APP

SIMPLIFIED SOLUTION FOR CONNECTING USERS

School of Engineering

Object-oriented programming

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Julián Carvajal Garnica & Mauricio Cepeda Villanueva

INTRODUCTION

Dating apps like Tinder and Bumble have transformed how people interact, but they face a critical challenge: more than 60% of users abandon these apps due to usability and security issues.
This small-scale Tinder project prioritizes a simple, intuitive, and secure experience. It also serves as a case study to demonstrate how SOLID principles, modular design (UML), and OOP best practices enable the development of a maintainable and scalable application, implementing:

- Customizable profiles (photos, bio, interests).
- Like/dislike system with match notifications.
- User-friendly graphical interface (Swing) based on realistic mockups.

This demonstrates how a clean and deliberate design can address both the technical challenges and social dynamics faced by these platforms.

METHODOLOGY

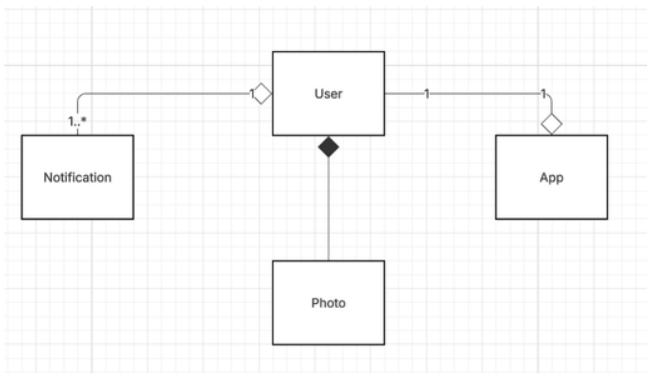
Previously challenged with creating a secure and usable app, we followed a tiered approach that linked conceptual design with technical development:

1.F**rom Requirements to Design**

- **User Stories** were transformed into:
  - Mockups
- Key features **prioritized** such as:
  - Like/dislike system
  - Profile customization

2.**Architectural model**

- **UML class diagram** [Fig. 1.0]:
  - **Main classes:** App, user Notification, Photo
  - Clear relationships
- **CRC Cards** with unique liability assignments



[Fig. 1.0] Simplified UML class diagram

3. **Implementation with SOLID** [Fig. 2.0]

Principle	Applied	Example
SRP	Yes	Photo only manage images
OCP	Yes	Extensible for future chat

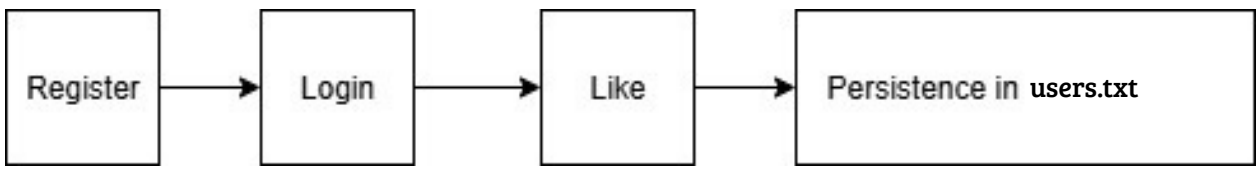
[Fig. 2.0] SOLID Principles Table

- **Encapsulation:**
  - **Private attributes** User.password,
  - Controlled public methods login(), register()
- **Validations:**
  - Unique email
  - Date format (YYYY-MM-DD)

4. **Testing and validation**

Each module was validated by:

- **Unit tests:**
  - Key methods such as Notification.addNotification()
- **Complete flows**



[Fig. 3.0] Complete flows

- **Modular architecture:**
  - The clear separation between classes User, Photo, App, Notification allowed adding functionality without modifying existing code

- **CRC Cards:**
  - They avoided crossing responsibilities

Notification
only manages messages, not interactions

RESULTS

1. **Implemented Functionalities**
   - The application fulfills all the planned basic functions:
     - User registration and login
     - Like/dislike system
     - Notifications when there is interest
     - Customizable profiles with photos and information
2. **Important Validations**
   - The system correctly verifies:
     - Make each email unique
     - That the data is formatted correctly
     - Don't miss out on important information
3. **Organization of the Code**
   - We use the SOLID principles correctly:
     - Each part of the program has a single responsibility
     - It's easy to add new features without messing up what already works

This prototype, although simpler than professional apps, demonstrated that the likes, profiles, and notifications system works correctly using files as a database. It's stable with multiple users, but would require a traditional database to scale. The results demonstrate that the design is robust and ready for expansion.

CONCLUSIONS

The development of the TinderApp demonstrated the effectiveness of OOP principles by successfully implementing a functional system for user registration, interactions through likes, and notification management, supported by a graphical interface developed in Java Swing. The modular architecture, based on SOLID and encapsulation, guaranteed a maintainable and scalable code, ready for future enhancements such as the integration of relational databases or advanced matching algorithms. The prototype validated the proposed design and its potential to evolve into a more complex product.

BIBLIOGRAPHY