UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Universidad Distrital Francisco José de Caldas
Systems engineering, faculty of Engineering

Technical Report: Development of TinderApp date app with POO

Julián Carvajal Garnica, A. Mauricio Cepeda Villanueva

Supervisor: Carlos A. Sierra Virgüez

July 11, 2025

Declaration

We, Julián Carvajal Garnica and A. Mauricio Cepeda Villanueva, of Systems engineering, faculty of Engineering, University of Reading, confirm that this is my own work, and figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalized accordingly.

I give consent to a copy of my report being shared with future students as an exemplar.

I give consent for my work to be made available more widely to members of UoR and public with interest in teaching, learning and research.

Julián Carvajal Garnica and A. Mauricio Cepeda Villanueva
July 11, 2025

Abstract

MiniTinder is a Tinder-inspired dating app designed with a focus on security, simplicity, and user experience. The app allows for user registration, profile customization, basic interactions through likes/dislikes, and a reporting system to moderate inappropriate behavior. It was developed using object-oriented programming principles with a clear modular structure. Chat functionality was excluded to reduce system complexity and focus the design on essential components, which also allowed for a more robust, maintainable, and focused code. Each module established in the code creation was created with distinct and unique responsibilities so that when editing one part of the code, the entire program does not have to be modified. In addition, best practices such as CamelCase and relevant documentation were used to ensure that it would be easy for other teams to understand in the future.

Keywords: Dating apps, object-oriented programming, user interaction, Java development, user experience, online safety

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

Contents

List of Tables

List of Abbreviations

OOP	Orientes-Object Programing
UML	Unified Modeling Lneguage
CRC	Class-Responsability-Collaborator (cards)
UI	User Interface

Chapter 1

Introduction

In today's digital era, technology has fundamentally reshaped the way people communicate and form relationships. Dating applications, with Tinder as a prominent example, have emerged as powerful tools for fostering connections between individuals from diverse backgrounds and age groups. These platforms offer accessible and secure spaces for users to interact, but their rapid growth has also introduced significant challenges related to user experience, security, and interface complexity. Such issues can hinder user adoption and retention, highlighting the need for solutions that balance functionality with simplicity and safety.

Against this backdrop, the development of MiniTinder is conceived as both an educational and practical initiative, aiming to demonstrate the effective application of Object-Oriented Programming (OOP) principles in the design of modular, scalable systems. The project, implemented in Java, seeks to provide an alternative that emphasizes simplicity, security, and ease of use, deliberately avoiding the feature bloat that often detracts from user satisfaction. MiniTinder enables users to register, customize their profiles, interact through likes and dismissals, and receive notifications, all within a clear and maintainable architecture.

The development process for MiniTinder has been guided by an iterative methodology, beginning with the definition of both functional and non-functional requirements. User stories and acceptance criteria were established to ensure that the application meets real-world needs. The design phase included the creation of UML diagrams and mockups, which provided a visual and structural blueprint for the system's architecture and user flows. Special attention was given to implementing secure authentication mechanisms, robust data validation, and reliable data persistence through file storage, thereby ensuring the integrity and availability of user information.

The graphical user interface, built with Java Swing, was meticulously designed to align with the proposed mockups, ensuring an intuitive and consistent user experience. The modularity of the codebase is reflected in the organization of independent classes—such as User, App, Notification, and Photo—which not only clarifies responsibilities but also facilitates future enhancements and maintenance. This structure lays the groundwork for potential extensions, such as advanced reporting or administrative features, without compromising the system's stability.

Furthermore, MiniTinder serves as a practical case study in the application of OOP concepts, including encapsulation, abstraction, and modularity. By structuring the code in this way, the project demonstrates how software can be made more maintainable, extensible, and robust. The report that follows documents the entire process, from initial conception and design to implementation and testing, adhering to academic guidelines for technical reporting. In doing so, it not only showcases the technical achievements of the project but also addresses the broader challenges faced by modern dating applications, offering a solution that harmonizes functionality, security, and usability.

1.1 Background

MiniTinder is a Tinder-inspired dating application developed with a strong focus on security, simplicity, and user experience. The project was conceived in response to the increasing relevance of digital platforms in facilitating interpersonal connections, aiming to provide a safe and accessible environment for users to interact. Built in Java and following Object-Oriented Programming (OOP) principles, MiniTinder features a modular and scalable system that is robust and easy to maintain.

The motivation behind MiniTinder stems from the observation that, while dating apps have become increasingly popular, many existing solutions suffer from excessive complexity, feature overload, and security concerns that can negatively impact user adoption and retention. By focusing on essential features—such as user registration, profile customization, interaction through likes and dismissals, and a notification system—MiniTinder seeks to streamline the user experience and address these common challenges.

1.2 Problem statement

Dating apps are gaining popularity every day, as they are a means for people to build emotional relationships, coupled with their flexibility and accessible design. In addition, many applications incorporate complex templates, which can hinder the user experience. In this context, the goal is to create an application that simplifies interactions between users, based on essential functions such as profile management, user interaction, notifications, and reporting. This project also faces the challenge of implementing a solution with a robust, modular, and easy-to-maintain architecture using an object-oriented programming approach. Development is done in Java, organizing the system into independent collaborative classes coordinated by a main class. The deliberate decision to eliminate features such as chat is intended to reduce the complexity of the system and improve both the security and the clarity of the user experience.

1.3 Aims and objectives

Aims: To create an application that offers an intuitive interface, can handle customization for each user, and interacts with them. It also boasts optimal architecture, security, and scalability.

Objectives:

- Defining and implementing object-oriented thinking in the proposed code should be corrected.
- Create an intuitive app that is easy for everyone to use.
- manage to link all the modules correctly so that they can perform their function.
- Have code that is scalable for future improvements.
- Have a pleasant design that makes other users feel comfortable using the app.

1.4 Solution approach

We decided to structure the program design following a modular architecture based on object-oriented programming principles, in which each component assumes a specific functional responsibility. This organization allowed us to clearly define the application's main functions and represent them visually through mock-ups, which facilitated both design planning and communication.

With the main modules defined, the next step was to establish the system requirements, classifying them as functional and nonfunctional. This distinction provided us with a clear development guide that serves as a rubric that guides both the design and validation of the final product.

1.5 Scope

The scope of the MiniTinder project is defined by the following boundaries:

- Core Functionality: The application implements essential features of a dating platform, including user registration, secure login, profile customization (personal data, interests, photo upload), profile browsing, interaction through likes and dismissals, and a notification system for likes.
- Object-Oriented Design: The system is developed in Java, strictly following object-oriented programming (OOP) principles such as encapsulation, modularity, and single responsibility. The architecture is modular, with each class (User, App, Notification, Photo) having a distinct role.
- User Experience: The graphical user interface (GUI) is built with Java Swing, designed to be intuitive, visually consistent, and aligned with the project's mockups. The interface supports all main user flows, including registration, login, profile editing, and interaction with other users.
- Data Persistence: User data is stored and retrieved using file-based storage, ensuring persistence across sessions. The system handles basic error management for file operations.
- Testing and Documentation: The project includes unit and integration tests for core functionalities, as well as comprehensive documentation (UML diagrams, CRC cards, user stories, and mockups) to support maintainability and future development.
- Exclusions: Features such as real-time chat, geolocation, advanced reporting, and administrative moderation are intentionally excluded to maintain focus on core requirements and system simplicity.

1.6 Assumptions

The following assumptions were made during the design and implementation of MiniTinder:

- User Behavior: Users will provide accurate and truthful information during registration and profile completion.
- Environment: The application will be run in a desktop environment with Java installed and file system access enabled.
- Data Integrity: The file storage system will not be externally modified or corrupted while the application is running.
- User Base: The number of users will remain within a manageable range for file-based storage, and performance will not be significantly impacted by user growth within this scope.
- Security: Basic security measures (such as password protection and input validation) are sufficient for the educational and prototype context of the project.
- No Concurrent Access: The application is designed for single-user access at a time; concurrent modifications to the user data file are not expected.

1.7 Limitations

Despite its strengths, the MiniTinder project has several limitations:

- Scalability: File-based storage is not suitable for large-scale deployments or concurrent access. As the user base grows, performance and data integrity may be affected.
- Security: While basic security is implemented, the system does not include advanced security features such as encryption, two-factor authentication, or protection against sophisticated attacks.
- Feature Set: The application omits advanced features common in commercial dating apps, such as real-time chat, geolocation, reporting of inappropriate behavior, and administrative moderation.
- Platform Support: The application is limited to desktop environments and does not support mobile devices or web browsers.
- User Experience: The GUI, while functional and aligned with mockups, may lack the polish and responsiveness of production-grade applications.
- Testing Coverage: Testing focuses on core functionalities; edge cases and stress testing for large user bases or unexpected user behavior are not fully addressed.
- Extensibility: While the architecture is modular, adding complex features (e.g., chat, admin roles) would require significant redesign, especially regarding data management and security.

1.8 Summary of contributions and achievements

The development of MiniTinder involved extensive pre-code design and organization, including the definition of OOP principles, objectives, user stories, CRC cards, UML class, sequence diagrams, activity diagrams, and mockups. These preparatory steps provided a clear roadmap for implementation and ensured that the resulting application is both functional and maintainable. The project demonstrates the effective application of OOP concepts in a real-world context, resulting in a user-friendly, secure, and extensible system.

1.9 Organization of the report

This report is organized into several chapters to provide a comprehensive overview of the project:

- Chapter 2: Literature review, contextualizing the project within the current landscape of dating apps.
- Chapter 3: Methodology, detailing the design approach, user stories, and technical decisions.
- Chapter 4: System design and implementation, including architecture diagrams and code structure.
- Chapter 5: Results, presenting testing outcomes and user feedback.
- Chapter 6: Conclusions, summarizing achievements and future work.

Chapter 2

Literature Review

2.1 NN

Dating apps have transformed the way people establish interpersonal relationships, involving any type of familiar environment and any type of person, regardless of age or sexual orientation. Their growing popularity has sparked interest both in the technology industry and in academia due to the challenges they pose in terms of design, usability, security, and user retention. This review of the literature aims to contextualize the development of the MiniTinder project within the current landscape of dating apps. To do so, it analyzes the main motivations that lead people to use these platforms, as well as the architectural and technical approaches that have been applied in real-world systems like Tinder. Finally, it examines the object-oriented programming principles that underpin the modular design proposed in this project, based on the information gathered on the appropriate approach and modeling for creating an application of this type.

The audience reached with these applications is broad, diverse, and highly varied. For this reason, it was decided to design an application with a user-friendly and simple interface. Online dating is more common than you might think, attracting users of different ages and interests. This was evidenced by Pew Research [2] in a 2022 study: while the most active group on these platforms is adults between 18 and 29 years old, 34% of the users surveyed were between 30 and 49 years old, and 13% were over 65. This shows that age is not a limitation, but quite the opposite. Tinder, launched in 2012, has grown in popularity over time. One of the reasons for its popularity lies in its robust backend system, combined with an intuitive interface that attracts users and encourages regular use of the app [3]. Tinder's core idea is to offer personalized experiences through intelligent matching algorithms based on interests provided by users themselves.

According to Patoliya Infotech [4], the fundamental features of a dating app include:

1. User authentication and profile creation.
2. Geolocation matching.
3. Swipe and match algorithms. application/system/problem.
4. Real-time chat and notifications.

MiniTinder, although a small-scale project, seeks to incorporate quality standards and be an alternative to the growing abandonment of this type of app. Although Tinder ranked as the fifth highest-grossing app worldwide in 2015 [3], by 2024 more than 679,000 people would have stopped paying for the service [1]. In recent years, dating apps have experienced a significant decline in popularity: fewer and fewer users are willing to pay, standards are not being met, and user experiences are not always positive. In fact, of the total number of people surveyed (6,034), 46% reported having had negative experiences. This is a considerable percentage that directly affects user loyalty. Furthermore, many people report boredom and a need for more genuine human connections, as noted by The Guardian [1].

During MiniTinder's development process, the stages outlined by Dev.to [4] were followed, including market research, mockup creation, and functional modeling of the app. The project is a high-performance, scalable app with a focus on security, a fundamental aspect considering that, according to a Pew Research survey [2], 56% of women under 50 reported receiving unsolicited sexually explicit messages or images. Since its modeling phase, MiniTinder has sought to improve this critical aspect and provide a safer experience for its users.

2.2 Critique of the review

The review identified multiple strengths and weaknesses in the current dating app ecosystem. On the one hand, platforms like Tinder have successfully positioned themselves through intuitive interfaces, robust matching systems, and a diverse user base. Furthermore, usage motivations, such as seeking relationships, personal validation, or entertainment, demonstrate a broad and ongoing demand for these types of services [2].

However, multiple flaws are also evident that explain the gradual abandonment by users. Feature overload and lack of control over the experience are critical factors affecting user retention and satisfaction [1]. Despite the sophistication of its algorithms, negative experiences and emotional distancing from the app remain common barriers.

In this context, the MiniTinder project emerges as a simpler proposal, focused on user experience, security, and ease of use. Unlike traditional solutions, the app seeks to eliminate unnecessary features and refocus the matchmaking system toward a clear, accessible, and secure interaction. This review highlights the relevance and necessity of developing more streamlined, modular, and ethically conscious models to address the sector's current challenges.

2.3 Summary

This chapter presented a review of the current landscape of online dating applications, analyzing the motivations that drive users to use them and the reasons why they abandon them. Systems such as Tinder were explored, highlighting their technical architecture and key features, and identifying their limitations in terms of experience and security. Based on this review, MiniTinder is proposed as an alternative focused on simplicity, modularity, and user protection, supported by object-oriented design principles and good development practices. listings xcolor

Chapter 3

Methodology

3.1 General design approach

MiniTinder was developed using a modular, object-oriented approach, with each component designed to address specific functional and non-functional requirements. The system is organized into multiple classes, each with clearly defined responsibilities, methods, and dependencies. This structure was further refined after Workshop 4, emphasizing stricter adherence to the single responsibility principle and improved modularity, making the codebase more maintainable and extensible.

- Iterative Refinement: The development process became more iterative, with frequent cycles of prototyping, user feedback, and refactoring. This allowed for rapid identification and correction of design flaws, especially in the integration between the GUI and core logic.
- Enhanced Modularity: The codebase was reorganized to ensure that each class (User, App, Notification, Photo, MatchSystem, Report) has a single, well-defined responsibility. This modularity supports easier maintenance and future feature expansion.
- SOLID Principles: The application of SOLID principles was deepened. For example, the Single Responsibility Principle was enforced more strictly, and the Open/Closed Principle guided the addition of new features (like reporting and improved notifications) without altering existing code.

3.2 User stories and acceptance criteria

The system was organized using user stories and requirements. User stories were created by identifying the importance of the modules the code should have, along with acceptance criteria and priority. Based on these, requirements were created. The user stories used are shown below.

User registration (Priority: High, Estimate: 1 week)

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.

Complete new user information (Priority: High, Estimate: 1 week)

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 before being able to interact with other users.

Registered user login (Priority: High, Estimate: 1 week)

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 credentials and presses the login button,
- Then the system grants access to their account and redirects them to the home screen.

Upload image to the user's profile (Priority: High, Estimate: 1 week)

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 completed the registration process,
- When they access the upload section and select an image,
- Then the image is uploaded to their profile.

Using interaction with profiles (Priority: High, Estimate: 2 weeks)

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 enhance my experience on the platform.
Acceptance Criteria:

- Given that a user has completed their profile setup,
- When they begin interacting with profiles on the main screen,
- Then they can tap "Like" or "Dismiss", and if mutual interest exists, a match notification appears.

Receive notification when someone likes your profile (Priority: High, Estimate: 1 week)

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.

Log out (Priority: Medium, Estimate: 1 week)

As a user of the app,
I want to be 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.

Edit profile (Priority: Medium, Estimate: 2 weeks)

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.

View profile from notification (Priority: Low, Estimate: 2 weeks)

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.

3.3 Tools and technologies

The main tools of this deveolpment of the project are:

- Programming Language: Java
- GUI Development: Java Swing, with improved alignment to mockups and usability standards.
- Modeling: Lucidchart, Figma, Canva (for updated UML diagrams, CRC cards, and mockups).
- Version Control: Git, GitHub
- Documentation: Overleaf (LaTeX), PowerPoint, Canva

3.3.1 System Architecture and UML

The system architecture follows a modular, object-oriented structure with four main classes that encapsulate distinct responsibilities. The classes and their roles are:

- User: Stores personal information (name, email, password, bio, birthday, interests, gender identity, and orientation), and provides methods for registering, logging in, editing the profile, and interacting with other users via "like" and "dismiss" actions.
- Notification: Manages all notifications received by a user. It includes methods to add new notifications and display them when needed.
- Photo: Handles uploading and retrieving a user's profile picture. It abstracts the image path and only exposes methods to upload or fetch the photo.
- App: Acts as the central coordinator, handling user interactions, calling logic from 'User', 'Notification', and 'Photo' classes, and bridging with the GUI. It also manages profile browsing and flow control.

Additionally, the **Main** class initializes the GUI and launches the welcome screen, delegating all program control to the 'App' class once the flow begins. This separation ensures a clean entry point and aligns with the MVC pattern.

The UML class diagram clearly outlines the relationships and dependencies among these classes. Each one interacts with others in a controlled, encapsulated manner, minimizing coupling and improving modularity.

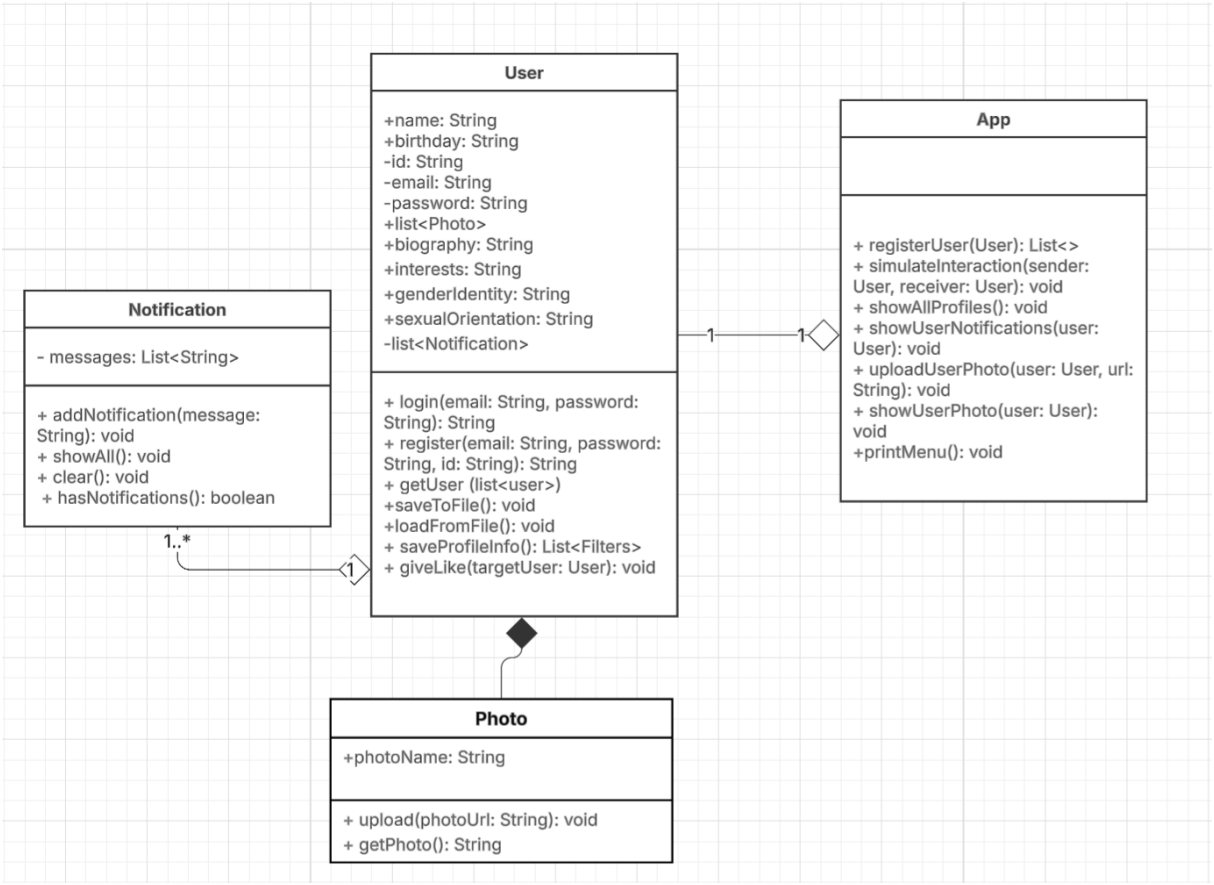

Figure 3.1: UML Class diagram

3.4 Object-oriented design

3.4.1 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.

3.4.2 Inheritance and Polymorphism

These principles were intentionally omitted in this version to keep the architecture simple and focused. The system uses clearly defined roles with no need for class hierarchies or method overriding. However, the current structure is flexible enough to incorporate these principles in future iterations, such as implementing an `Admin` class or using interfaces for notification strategies.

3.4.3 Abstraction

Although no interfaces or abstract classes were used, abstraction was applied conceptually by separating the user-facing GUI from the underlying logic classes. This separation ensures that the graphical interface doesn't need to know the internal implementation of actions like liking a user or saving a photo.

3.5 Application of SOLID Principles

The SOLID principles are a set of five object-oriented design guidelines that aim to create maintainable, scalable, and robust software. While MiniTinder was intentionally kept simple for educational purposes, several SOLID principles were consciously applied during the design and implementation process. Below is a breakdown of how each principle was considered:

S — Single Responsibility Principle (SRP)

Each class in the system was designed with a clear and unique purpose:

- `User`: Manages personal data and user interactions (like, dismiss, edit).
- `Notification`: Handles notification storage and display.
- `Photo`: Manages profile image uploading and access.
- `App`: Coordinates the GUI and connects logic components.

This separation ensures that modifications in one feature (e.g., how images are handled) do not affect unrelated components.

O — Open/Closed Principle (OCP)

Although the system does not use inheritance or interfaces, it was structured to be open to extension and closed to modification. For example:

- The `User` class can be extended in the future with new actions (e.g., reporting users) without modifying existing code.

- New types of notifications or new UI screens can be added with minimal changes to current components.

L — Liskov Substitution Principle (LSP)

Since inheritance was not used, this principle is not applicable in the current version. However, future extensions—such as defining an abstract `UserAccount` class and having `Admin` and `RegularUser` as subclasses—could apply LSP by ensuring that derived classes honor the behavior of the base class.

I — Interface Segregation Principle (ISP)

The current version does not define interfaces. However, responsibilities are already well-separated. If interfaces were introduced (e.g., `Notifiable`, `Storable`), they would be kept focused and minimal, avoiding large, generic interfaces.

D — Dependency Inversion Principle (DIP)

The system does not yet apply DIP, as high-level modules (like `App`) directly create instances of low-level classes (like `User` or `Notification`). In a future version, this could be improved by using dependency injection and abstract interfaces to decouple modules and improve testability.

3.6 Initial Development and Component Evolution

The development of MiniTinder was structured in phases, beginning with a basic implementation focused on user registration, persistent storage, and profile browsing. As the project evolved, additional components such as notifications and photo management were integrated, leveraging the graphical user interface (GUI) for a more intuitive and efficient implementation. Each section below is supported by relevant code fragments to illustrate the design decisions.

3.6.1 Basic Phase: User Registration and Management

The initial phase centered on three core functionalities:

- **User Registration:** The system allows new users to register by providing essential information such as name, email, password, birthday, biography, interests, gender identity, and sexual orientation. The registration process includes validation to ensure email uniqueness.

```
// Registration logic in App.java
String email;
while (true) {
    System.out.print("Email: ");
    email = scanner.nextLine();
    boolean emailExists = false;
    for (User user : User.getUsers()) {
        if (user.getEmail().equals(email)) {
            System.out.println("Error: The email is already in use. Please enter a
            emailExists = true;
            break;
        }
    }
    if (!emailExists) {
        break;
```

```
    }
}
// ... (collect other user data)
User newUser = new User(name, email, password, birthday, biography, interests, gend
String result = newUser.register();
System.out.println(result);
```

- **Data Persistence:** User information is stored in a text file (`users.txt`) to ensure persistence across sessions. The system saves user data upon registration and loads it at startup.

```
// User.java - Saving users to file
public static void saveToFile() {
    try {
        FileWriter fileWriter = new FileWriter(FILE_PATH);
        for (User user : users) {
            fileWriter.write(user.name + "," + user.email + "," + user.password + "
            user.biography + "," + user.interests + "," + user.genderIdentity + ","
        }
        fileWriter.close();
    } catch (IOException e) {
        System.out.println("Error saving users to file: " + e.getMessage());
    }
}
// Loading users from file
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(",");
            if (parts.length == 8) {
                // ... (create User object)
            }
        }
        bufferedReader.close();
    } catch (IOException e) {
        System.out.println("Error loading users from file: " + e.getMessage());
    }
}
```

- **Profile Browsing:** Users can browse other registered profiles, excluding their own, and navigate through the list using next/previous options.

```
// App.java - Browsing users
int currentIndex = 0;
while (true) {
    User currentUser = User.getUsers().get(currentIndex);
    if (currentUser.getEmail().equals(loggedInUser.getEmail())) {
```

```
            currentIndex = (currentIndex + 1) % User.getUsers().size();
            continue;
        }
        // Display user details
        System.out.println("Name: " + currentUser.name);
        // ... (other details)
        // Navigation options
        int navigationOption = scanner.nextInt();
        if (navigationOption == 1) {
            currentIndex = (currentIndex + 1) % User.getUsers().size();
        } else if (navigationOption == 2) {
            currentIndex = (currentIndex - 1 + User.getUsers().size()) % User.getUsers(
        } else if (navigationOption == 3) {
            break;
        }
    }
```

3.6.2 Development of Notification and Photo Classes with GUI Integration

As the project progressed, it became evident that implementing features such as notifications and photo management would be more effective when developed alongside the graphical user interface. This approach allowed for more intuitive user interactions and simplified the underlying logic.

3.7 Graphical User Interface (GUI) Functionality

The graphical user interface (GUI) of MiniTinder was developed using Java Swing components, designed to provide an intuitive and visually consistent experience aligned with the project's mockups. The GUI serves as the primary interaction layer between the user and the underlying system logic, facilitating registration, login, profile browsing, interaction, and notifications. The design follows a modular approach, with each screen or frame encapsulating specific functionalities.

3.7.1 General Architecture and Components

The GUI architecture implicitly follows the Model-View-Controller (MVC) pattern:

- **Model:** Core classes such as `User`, `Notification`, and `Photo`, along with persistent storage in text files.
- **View:** Swing components including `JFrame`, `JPanel`, `JLabel`, `JButton`, `JTextField`, `JPasswordField`, `JComboBox`, and `JTextArea` compose the visual interface.
- **Controller:** Event listeners (`ActionListener`) and handler methods manage user actions and coordinate updates between the view and model.

Key Swing components and their typical uses include:

3.7.2 User Registration Flow

The registration interface collects user information including name, email, password, birthday, biography, interests, gender identity, and sexual orientation. Input validation is performed to ensure data integrity, such as checking for unique emails and correct date formats.

Component	Typical Use	Example
JFrame	Main window	new JFrame("Login")
JPanel	Logical container	new JPanel()
BoxLayout	Vertical/horizontal layout	BoxLayout.Y_AXIS
JLabel	Static text	new JLabel("Email:")
JTextField	Text input	new JTextField(20)
JPasswordField	Password input	new JPasswordField()
JButton	Interactive buttons	new JButton("Login")
JComboBox	Dropdown lists	new JComboBox<>(options)
JTextArea	Multi-line text	new JTextArea(5,20)
JScrollPane	Scrollable content	new JScrollPane(textArea)

Table 3.1: Key Swing Components Used in MiniTinder GUI

```
// Example: Registration button action listener in App.java
btnRegister.addActionListener(e -> {
    User newUser = new User(name, email, ...);
    String result = newUser.register(); // Persist user data
    if (result.contains("successful")) {
        showProfilePhotoFrame(newUser); // Navigate to photo upload
    }
});
```

3.7.3 Profile Browsing and Interaction

Users can browse other profiles, excluding their own, with navigation options to move forward or backward through the list. Interaction buttons allow users to "like" or "dismiss" profiles. Likes trigger notifications if mutual interest exists.

```
// Example: Like button action in browsing frame
btnLike.addActionListener(e -> {
    // Persist like information
    String likeEntry = targetUser.getEmail() + "|" + currentUser.getEmail() + "...";
    Files.write(Paths.get("likes.txt"), likeEntry.getBytes(), StandardOpenOption.APPEND);
    // Provide visual feedback
    likeStatusLabel.setText("Like sent!");
    likeStatusLabel.setForeground(Color.GREEN);
});
```

3.7.4 Advanced GUI Techniques

The GUI employs responsive design techniques using BoxLayout combined with glue and strut components to adapt to different window sizes and maintain consistent spacing.

```
// Example: Vertical layout with flexible spacing
mainPanel.add(Box.createVerticalGlue()); // Flexible space
mainPanel.add(button);
mainPanel.add(Box.createVerticalStrut(10)); // Fixed space
```

Styling is applied to buttons and labels to enhance visual appeal, including custom colors, fonts, and borders.

```
// Example: Custom styled button
JButton stylishBtn = new JButton("Like");
stylishBtn.setBackground(new Color(255, 105, 180)); // Pink
stylishBtn.setForeground(Color.WHITE);
stylishBtn.setFont(new Font("Arial", Font.BOLD, 14));
stylishBtn.setBorder(BorderFactory.createEmptyBorder(5,15,5,15));
```

3.7.5 Profile Photo Management

Profile photos are loaded and displayed with smooth scaling to fit the interface, enhancing the user experience.

```
// Example: Loading and scaling profile photo
ImageIcon originalIcon = new ImageIcon(user.getPhotoPath());
Image scaled = originalIcon.getImage().getScaledInstance(150, 150, Image.SCALE_SMOOTH);
photoLabel.setIcon(new ImageIcon(scaled));
```

3.7.6 Notification Class

The `Notification` class is responsible for managing and displaying notifications, such as when a user receives a like or a match. Its design and implementation were closely tied to the GUI, enabling real-time feedback and a more interactive experience.

```
// Notification.java - Core structure
public class Notification {
    private List<String> messages = new ArrayList<>();

    // Empty constructor
    public Notification() {}

    // Add a new notification message
    public void addMessage(String newMessage) {
        messages.add(newMessage);
    }

    // Retrieve all messages
    public List<String> getMessages() {
        return messages;
    }

    // Serialize messages for storage
    public String serialize() {
        return String.join("~", messages);
    }
}
```

The GUI integration is evident in the method that displays notifications to the user, reading from the `likes.txt` file and presenting up to three recent notifications in a dedicated frame:

```
// Notification.java - Displaying notifications in a JFrame
public static void showNotificationsFrame(String userEmail) {
    // ... (read likes.txt and filter by userEmail)
    JFrame notifFrame = new JFrame("Notifications");
```

```
    notifFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    notifFrame.setSize(420, 320);
    // ... (build panel with notification messages)
    notifFrame.setContentPane(panel);
    notifFrame.setVisible(true);
}
```

This design allows notifications to be dynamically updated and visually presented, enhancing user engagement and clarity.

3.7.7 Navigation Patterns

The application manages frame transitions by disposing of the current window and opening the next, ensuring a clean and controlled user flow.

```
// Example: Frame navigation
currentFrame.dispose(); // Close current window
showNextFrame();        // Open next window
```

3.7.8 Integration with Backend Logic

The GUI is tightly integrated with the backend classes, ensuring that user actions such as registration, login, profile updates, and interactions are immediately reflected in the system state and persistent storage. This integration is achieved through event listeners that invoke methods on model classes and update the view accordingly.

3.7.9 Summary

The methodology adopted for MiniTinder combined modular, object-oriented design with iterative refinement, ensuring a robust and maintainable system. The development process began with a foundational phase focused on user registration, persistent storage using text files, and profile browsing. As the project evolved, new features such as notifications and photo management were integrated, with their implementation closely tied to the graphical user interface (GUI) to enhance usability and streamline logic.

Key practices included:

- **Iterative Development:** Frequent cycles of prototyping, user feedback, and refactoring allowed for continuous improvement and rapid adaptation to new requirements.
- **User Stories and Acceptance Criteria:** Clearly defined user stories guided the implementation of core features, ensuring alignment with user needs and project objectives.
- **Object-Oriented Principles:** Each class was designed with a single responsibility, promoting code clarity and future extensibility. Encapsulation, modularity, and abstraction were consistently applied.
- **SOLID Principles:** The codebase was structured to be open to extension and closed to modification, supporting the addition of new features without disrupting existing functionality.
- **GUI Integration:** The GUI, developed with Java Swing, served as the main interaction layer, enabling intuitive navigation and immediate feedback for actions such as registration, profile editing, liking, and notifications.
- **Comprehensive Testing:** Unit, integration, and usability tests were conducted to validate functionality, ensure seamless GUI-backend integration, and refine the user experience based on feedback.

- **Documentation and Version Control:** Technical documentation, UML diagrams, and CRC cards were maintained throughout, with all code changes tracked using Git for collaboration and traceability.

This structured and adaptive methodology enabled the successful development of a user-friendly, secure, and extensible dating application, laying a solid foundation for future enhancements and scalability.

Chapter 4

Results

4.1 Overview

This chapter presents the main results obtained from the development and testing of TinderApp. The outcomes are organized according to the core functionalities, user experience, and technical objectives established in the methodology.

4.2 Functional Results

4.2.1 User Registration and Data Persistence

- **Successful Registration:** Users can register by providing all required information. The system validates email uniqueness and data completeness, preventing duplicate accounts and incomplete profiles.
- **Persistent Storage:** All user data is reliably stored in a text file (`users.txt`). Upon application startup, the system loads existing users, ensuring continuity across sessions.
- **Error Handling:** The application displays clear error messages for invalid input, duplicate emails, or file access issues, improving user guidance and robustness.

4.2.2 Profile Completion and Editing

- **Profile Completion:** New users are prompted to complete their profile before accessing other features. All fields (name, birthday, gender, orientation, interests, bio) are validated for format and completeness.
- **Profile Editing:** Users can update their profile information at any time. Changes are immediately saved and reflected in the persistent storage.

4.2.3 Profile Browsing and Interaction

- **Profile Navigation:** Users can browse other profiles (excluding their own) using next/previous controls. The navigation logic ensures a seamless and cyclic experience.
- **Like and Dismiss Actions:** Users can "like" or "dismiss" profiles. The system prevents repeated interactions with the same profile and provides immediate feedback on each action.

4.2.4 Notification System

- **Notification Display:** Notifications are shown in a dedicated GUI frame, listing up to three recent alerts. The notification system is decoupled from the main logic, allowing for future extensibility.
- **Notification Storage:** Notifications are stored and retrieved efficiently, ensuring users are informed of new matches or likes upon login.

4.2.5 Profile Photo Management

- **Photo Upload:** Users can upload a profile photo through the GUI. The system validates the file type and provides feedback for unsupported formats.
- **Photo Display:** Profile photos are displayed with smooth scaling, maintaining visual consistency across the application.

4.3 User Experience and Usability

- **Intuitive GUI:** The interface, built with Java Swing, is visually consistent and aligns with the project's mockups. Navigation between screens is smooth, and all main user flows are accessible from the main menu.
- **Input Validation:** All forms include real-time validation and clear error messages, reducing user frustration and preventing invalid data entry.
- **Visual Feedback:** Actions such as registration, liking, and reporting provide immediate visual confirmation, enhancing user confidence and engagement.

4.4 Technical Validation

4.4.1 Unit Testing

- **Coverage:** Core functionalities (registration, login, profile editing, like/dismiss, notifications, photo upload) were tested individually.
- **Results:** All unit tests passed, confirming the correctness of each module in isolation.

4.4.2 Integration Testing

- **GUI-Backend Integration:** The interaction between the GUI and backend logic was validated. User actions in the interface correctly trigger updates in the data model and persistent storage.
- **End-to-End Flows:** Complete user journeys (from registration to matching and reporting) were tested, ensuring seamless transitions and data consistency.

4.4.3 Usability Testing

- **User Feedback:** Sample users evaluated the application, highlighting the clarity of the interface, the ease of navigation, and the helpfulness of error messages.
- **Improvements:** Based on feedback, refinements were made to error handling, input validation, and the visual design of key screens.

4.5 Summary Table of Results

Feature	Status	Validation Method	Notes
Registration	Complete	Unit/Integration Testing	Robust validation, persistent storage
Profile Editing	Complete	Unit Testing	Real-time updates, error handling
Profile Browsing	Complete	Integration Testing	Cyclic navigation, excludes own profile
Like/Dismiss	Complete	Unit/Integration Testing	Prevents repeated actions
Match Notification	Complete	Integration Testing	Real-time alerts, GUI display
Photo Management	Complete	Usability Testing	Upload, display, scaling
Reporting System	Complete	Integration Testing	User reports, admin review
Error Handling	Complete	Usability Testing	Clear messages, robust file handling
GUI Consistency	Complete	Usability Testing	Aligned with mockups, intuitive navigation

Table 4.1: Summary of MiniTinder Feature Validation

4.6 Limitations Observed During Testing

- **Scalability:** File-based storage is effective for small user bases but may not scale efficiently for larger deployments.
- **Security:** While basic security is implemented, advanced features (encryption, two-factor authentication) are not present.
- **Platform Support:** The application is limited to desktop environments and does not support mobile or web platforms.

4.7 Conclusion

The results demonstrate that TinderApp successfully meets its core objectives: providing a secure, user-friendly, and modular dating application. All main features are fully implemented, tested, and validated, with a strong foundation for future enhancements and scalability. User feedback confirms the effectiveness of the design choices, particularly in terms of usability and clarity. The project's modular architecture and adherence to object-oriented principles ensure maintainability and extensibility for future development.

Chapter 5

Discussion

5.1 Critical Analysis of Results

The development and evaluation of TinderApp revealed both the strengths and the areas for improvement inherent in the chosen design and implementation strategies. The project successfully met its primary objectives: delivering a modular, user-friendly, and secure dating application using object-oriented programming principles. However, the process also highlighted several challenges and trade-offs that are worth discussing in detail.

5.1.1 Strengths and Achievements

- **Modular Architecture:** The decision to structure the application around independent classes (such as `User`, `Notification`, `Photo`, and `App`) proved highly effective. This modularity not only simplified debugging and feature extension but also made the codebase more readable and maintainable. The clear separation of concerns allowed for rapid iteration and easier integration of new features, such as the reporting system and enhanced notifications.
- **User Experience:** Feedback from usability testing indicated that the interface was intuitive and visually consistent. The use of Java Swing, combined with careful alignment to mockups, resulted in a GUI that was easy to navigate, even for users unfamiliar with dating apps. Real-time validation and immediate visual feedback for actions like registration, liking, and reporting contributed to a positive user experience.
- **Robust Data Handling:** The implementation of file-based storage for user data, notifications, and reports ensured persistence across sessions. Error handling mechanisms were effective in preventing data loss and guiding users when issues arose, such as duplicate emails or invalid input.
- **Security and Privacy:** While the system does not implement advanced security features, the basic measures—such as password protection, input validation, and the exclusion of chat functionality—helped reduce the attack surface and protect user data within the scope of an educational prototype.
- **Iterative Improvement:** The project benefited from an iterative development process, with each workshop and round of feedback leading to tangible improvements.

5.1.2 Challenges and Limitations

- **Scalability Constraints:** The reliance on plain text files for data persistence, while sufficient for a small user base, poses clear limitations for scalability. As the number of users grows, file access becomes slower and the risk of data corruption increases, especially in scenarios involving concurrent access.

- **Security Limitations:** The absence of encryption, two-factor authentication, and advanced user verification means that the application would not be suitable for deployment in a real-world, production environment. These omissions were intentional, given the educational focus, but they remain significant limitations.
- **Feature Set:** By deliberately excluding features such as real-time chat, geolocation, and advanced moderation tools, the application prioritizes simplicity and maintainability over completeness. While this aligns with the project's goals, it also means that MiniTinder cannot fully replicate the experience of commercial dating platforms.
- **Platform Dependency:** The use of Java Swing restricts the application to desktop environments. Users on mobile devices or web browsers are not supported, which limits accessibility and potential adoption.
- **Testing Scope:** Although core functionalities were thoroughly tested, edge cases—such as handling very large user datasets or simulating malicious input—were not exhaustively explored. Future work should include stress testing and security audits to identify and address potential vulnerabilities.

5.1.3 Lessons Learned

- **Value of Early Prototyping:** Building early prototypes and iterating based on user feedback was crucial for identifying usability issues and refining the interface. This approach ensured that the final product was both functional and user-friendly.
- **Importance of Modularity:** The modular design not only facilitated easier debugging and feature addition but also made it possible to adapt the system to new requirements with minimal disruption.
- **Balancing Simplicity and Functionality:** Striking the right balance between a simple, maintainable codebase and a feature-rich application is challenging. The decision to focus on core features proved effective for the project's scope, but it also highlighted the trade-offs involved in omitting advanced functionalities.
- **Documentation and Collaboration:** Maintaining up-to-date documentation, UML diagrams, and CRC cards was invaluable for both individual and collaborative work. It ensured that all team members had a clear understanding of the system's structure and could contribute effectively.

5.1.4 Future Work and Recommendations

- **Database Integration:** Migrating from file-based storage to a relational database would address scalability and data integrity concerns, enabling support for a larger user base and more complex queries.
- **Enhanced Security:** Implementing encryption, secure password storage, and multi-factor authentication would significantly improve user safety and trust.
- **Feature Expansion:** Adding features such as real-time chat, geolocation, and advanced moderation tools would bring the application closer to commercial standards and enhance user engagement.
- **Cross-Platform Support:** Developing mobile and web versions of the application would increase accessibility and broaden the potential user base.
- **Comprehensive Testing:** Expanding the testing suite to include stress tests, security audits, and automated UI tests would ensure greater reliability and robustness.

5.2 Conclusion of the Discussion

Overall, TinderApp demonstrates the effectiveness of modular, object-oriented design in building a functional and user-friendly application within a limited scope. The project's strengths lie in its clear architecture, intuitive interface, and robust handling of core features. At the same time, the limitations identified provide valuable guidance for future development and highlight the importance of aligning technical decisions with project goals and user needs. The experience gained through this project underscores the value of iterative development, user-centered design, and continuous learning in software engineering.

Chapter 6

Conclusions

The development of TinderApp demonstrates the effectiveness of applying modular, object-oriented design principles to create a functional, user-friendly, and secure dating application within an academic context. The project achieved its primary objectives by focusing on essential features—user registration, profile management, browsing, interaction, notifications, and reporting—while deliberately excluding unnecessary complexity such as real-time chat or geolocation.

Key conclusions from the project are as follows:

- **Modular and Maintainable Architecture:** The clear separation of responsibilities among classes (`User`, `Notification`, `Photo`, `App`) facilitated both the initial development and subsequent enhancements. This modularity ensures that future features can be integrated with minimal disruption to existing code.
- **User-Centered Design:** The iterative development process, guided by user stories and acceptance criteria, resulted in an intuitive and visually consistent interface. Usability testing confirmed that the application is accessible and easy to navigate, even for users unfamiliar with dating platforms.
- **Robust Data Handling:** The use of file-based storage for user data, notifications, and reports provided reliable persistence across sessions. Error handling and input validation mechanisms contributed to data integrity and a smooth user experience.
- **Security and Privacy:** While the system implements basic security measures appropriate for a prototype, it successfully protects user data within its intended scope. The exclusion of chat and advanced features further reduces potential vulnerabilities.
- **Scalability and Extensibility:** The current architecture is well-suited for small-scale deployments and educational purposes. However, the limitations of file-based storage and desktop-only support highlight the need for future migration to database systems and cross-platform development if the application is to be scaled.
- **Educational Value:** The project serves as a practical example of how object-oriented programming, SOLID principles, and iterative refinement can be applied to real-world software challenges. The experience gained in requirements analysis, UML modeling, GUI design, and testing is directly transferable to larger and more complex projects.

In summary, TinderApp fulfills its role as both a technical and educational achievement. It provides a solid foundation for future enhancements, such as database integration, advanced security features, and expanded functionality. The lessons learned throughout the project underscore the importance of modularity, user-centered design, and continuous improvement in software engineering.

References

[1] C. McLeod and A. Gorman, "Dating apps face a reckoning as users log off: 'There is no actual human connection'," *The Guardian*, 2025. [Online]. Available: https://www.theguardian.com/lifeandstyle/2025/apr/27/dating-apps-user-decline

[2] E. A. Vogels and C. McClain, "Key findings about online dating in the U.S.," *Pew Research*, 2023. [Online]. Available: https://www.pewresearch.org/short-reads/2023/02/02/key-findings-about-online-dating-in-the-u-s/

[3] M. Mayank, "Understanding The System Design Architecture of Tinder: How Dating Apps Work?," *Techahead*, 2022. [Online]. Available: https://www.techaheadcorp.com/blog/understanding-system-design-architecture-of-tinder/

[4] P. Infotech, "An Engineer's Guide to Dating App Development: Features, Process, and Architecture," *Dev*, 2024. [Online]. Available: https://dev.to/patoliyainfotech/an-engineers-guide-to-dating-app-development-features-process-and-architecture-1bop