Digital Wallet System: A User-Centric Approach Using Object-Oriented Principles

Julian David Pulido Carreño*, David Martinez Rincon†
*Facultad de Ingeniería, Universidad Distrital Francisco José de Caldas
Email: judpulidoc@udistrital.edu.co
Email: Dmartinezr@udistrital.edu.co

***Abstract*—This paper presents a detailed analysis and design of a digital wallet system using object-oriented principles. The system includes features such as user authentication, transaction management, and notification services. The proposed design ensures modularity, maintainability, and reusability by adhering to key principles like encapsulation, inheritance, and polymorphism. Results from applying these principles to user stories demonstrate an efficient and user-friendly solution for managing digital transactions.**

I. Introduction

With the rise of digital transactions, digital wallets have become an essential tool for users to manage their finances. This paper proposes a design for a digital wallet that prioritizes user-centric features such as transaction history, notifications, and security. The system leverages object-oriented programming principles, ensuring that the solution is flexible, maintainable, and scalable. Previous solutions to digital wallet systems have focused primarily on functionality without considering extensibility or code reusability. This research aims to fill that gap by presenting a design that balances both user needs and technical efficiency.

II. Method and Materials

The digital wallet system was designed with a focus on object-oriented principles, ensuring a modular, maintainable, and scalable architecture. The system comprises various components that collectively enable essential functionalities such as user authentication, transaction management, notification services, and account handling. The following object-oriented design principles were applied: encapsulation, inheritance, polymorphism, and adherence to design principles like the Single Responsibility Principle (SRP) and Liskov Substitution Principle (LSP).

A. System Architecture

The architecture of the digital wallet is built around core objects such as `User`, `Account`, `Transaction`, and `Notification`. Each object encapsulates specific data and behaviors, interacting with one another through well-defined interfaces.

- **User Authentication:** The `Authentication` class is responsible for managing user login, logout, and password recovery. This class ensures that user credentials are securely handled, with access granted only after proper verification. Password recovery is handled via email, ensuring that the user can regain access to their account securely.
- **Account Management:** The `Account` class handles all operations related to the user's account, including balance inquiry, deposits, withdrawals, and transaction history. The account is associated with a unique identifier and maintains private attributes such as balance, which can only be accessed through getter methods to ensure encapsulation. This class also follows the SRP by exclusively focusing on account-related operations, ensuring that it remains isolated from other functionalities like authentication or notifications.
- **Transaction Management:** The `Transaction` class handles the recording of all financial activities, including deposits, withdrawals, and transfers. In this system, transactions are represented as objects, each with attributes such as the transaction amount, date, and status (pending, completed, or failed). The system uses inheritance by implementing specialized subclasses of `Transaction`, such as `Send` and `Withdraw`, which share common behaviors but extend the base class to include functionality specific to each transaction type.

B. Object-Oriented Principles

1) Encapsulation: Encapsulation ensures that each object's internal state is protected from unauthorized access and modifications. For instance, the `Account` class encapsulates sensitive information such as the user's balance, account ID, and transaction history. This data can only be accessed or modified through public methods, such as `getBalance()` or `addTransaction()`. This separation of concerns enforces security and ensures that no external components can directly alter the account's state.

Similarly, the `Transaction` class encapsulates details like transaction ID, amount, date, and status. Users and other system components can access transaction details through getter methods like `getTransactionDetails()` without altering the state of the transaction directly.

2) Inheritance: Inheritance is applied to promote code reuse and ensure that common functionalities are centralized in a parent class. For example, all types of transactions inherit from a common `Transaction` class. This base class defines shared attributes and behaviors, such as transaction

ID, amount, and date, which are common to all transactions. Specific transaction types like `Send` and `Withdraw` extend this base class and introduce unique behaviors while retaining the shared functionalities.

A similar approach could be applied to notifications. For instance, `Notification` could serve as a base class, while specific types of notifications such as `EmailNotification` and `SMSNotification` inherit from it. This structure allows the system to handle different types of notifications uniformly, while still supporting customization for each type.

3) Polymorphism: Polymorphism allows different types of transactions to be processed in a generic way, improving flexibility. In this system, methods that process transactions can treat all transactions (whether deposit, withdrawal, or transfer) as objects of the `Transaction` class. For instance, the system can loop through a user's transaction history and call the `processTransaction()` method, regardless of whether the transaction is a `Send` or a `Withdraw`. This makes the system more extensible since new transaction types can be added without modifying the core processing logic.

C. Application of Design Principles

1) Single Responsibility Principle (SRP): The Single Responsibility Principle is strictly followed in the design. Each class in the system has a clear and focused responsibility. For instance, the `Authentication` class solely manages login and logout functionality, ensuring that any changes to the authentication process do not affect other parts of the system. Similarly, the `Transaction` class deals exclusively with transaction-related operations, while the `Notification` class is responsible for sending notifications after each transaction is completed. By adhering to SRP, the system is easier to maintain and extend, as changes in one part of the system do not affect unrelated components.

2) Liskov Substitution Principle (LSP): The Liskov Substitution Principle ensures that objects of a superclass can be replaced with objects of its subclasses without affecting the correctness of the system. For instance, since `Send` and `Withdraw` are subclasses of `Transaction`, they can be used interchangeably in any part of the system that processes transactions. This guarantees that new transaction types can be integrated seamlessly without introducing errors, as long as they adhere to the structure and behaviors defined by the base `Transaction` class.

D. System Diagrams

To illustrate the interaction between these components, sequence diagrams and class diagrams were developed. The sequence diagram for a deposit transaction shows how the user interacts with the system to add funds to their account, starting with authentication and culminating in a successful transaction notification. The class diagram details the relationships between `User`, `Account`, `Transaction`, and `Notification`, showcasing how inheritance and encapsulation are implemented in the system.

E. Technologies and Tools

The system was developed using Java, leveraging its robust support for object-oriented programming. The system's architecture allows for easy integration with databases for transaction records and notifications, ensuring that the system can scale to handle a large number of users and transactions.

A class diagram and sequence diagrams were created to represent interactions within the system. An example sequence diagram for the deposit and withdrawal functionalities is included.

III. Results

The system was developed based on several user stories. For instance:

- **User Story:** As a user, I want to log in using my email and password.
- **Result:** The authentication process securely manages user login information while adhering to the Single Responsibility Principle.

Additionally, the notification system allows users to receive real-time updates on transaction statuses, contributing to user satisfaction. Tests for transaction management demonstrate successful handling of different transaction types, such as deposits and withdrawals.

IV. Conclusions

References