Algorithms and Programming
Report (L1BC)

Computer Science Undergraduate Program
Universitas Bina Nusantara
Jakarta

Final Project Report

Submitted by:
Bunleap Sorn (2702345693)

Submitted to:
Jude Joseph Lamug Martinez MCS
Algorithms and Programming
COMP6047001

Finance Tracker Documentation

Project Overview

The Finance Tracker project aims to create a Python-based financial management system that allows users to register, log in, and track their financial transactions. The system focuses on data security, efficient transaction management, and user-friendly interactions. The core functionality includes user registration, login, and the ability to add, update, delete, and view financial transactions associated with individual user accounts.

Key Features

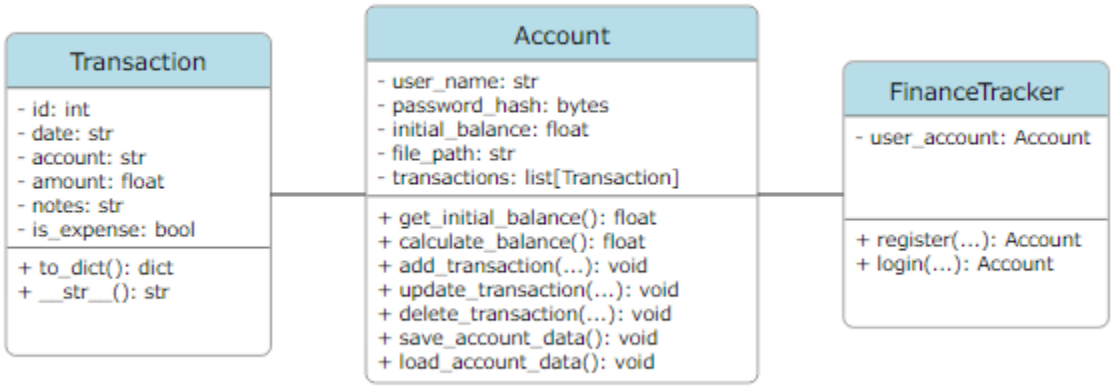
1. User Registration and Login:
   - Users can register by providing a unique user name, a secure password, and an initial balance.
   - Passwords are securely hashed using the bcrypt library.
   - Existing users can log in by entering their user name and password.

2. Financial Transaction Management:
   - Users can add new financial transactions, specifying the date, account, amount, notes, and whether it is an expense.
   - Transactions can be updated, deleted, and viewed.
   - The system calculates and displays the user's account balance based on the transactions.

3. Data Persistence:
   - User account data, including transactions, is stored persistently in JSON files.
   - File naming follows the convention "{user_name}_account.json" for unique identification.

Solution Design

1. Class Structure



Transaction Class
Attributes:

- id: Transaction ID
- date: Transaction date
- account: Transaction account
- amount: Transaction amount
- notes: Additional notes for the transaction
- is_expense: Boolean indicating whether the transaction is an expense or not

Methods:

- to_dict(): Convert transaction data to a dictionary.
- __str__(): Return a string representation of the transaction.

Account Class
Attributes:

- user_name: User name
- password_hash: Hashed password using bcrypt
- initial_balance: Initial account balance
- file_path: File path for data persistence
- transactions: List of transactions associated with the account

Methods:

- get_initial_balance(): Retrieve the initial balance.
- calculate_balance(): Calculate the current account balance based on transactions.
- add_transaction(): Add a new transaction to the account.
- update_transaction(): Update an existing transaction.
- delete_transaction(): Delete a transaction.
- save_account_data(): Save account data to a JSON file.
- load_account_data(): Load account data from a JSON file.

FinanceTracker Class
Attributes:

- user_account: Reference to the current user's account

Methods:

- register(user_name, password, initial_balance): Register a new user account.
- login(user_name, password): Log in to an existing user account.

2. Data Flow

- User Registration:
    - The user provides a user name, password, and initial balance.
    - Passwords are hashed using bcrypt.
    - An Account object is created, and user data is saved to a JSON file.

- User Login:
    - The user enters username and password.
    - The password is verified against the hashed version stored in the account file.
    - If successful, an Account object for the user will be loaded.
- Transaction Management:
    - Users can add, update, delete, and view transactions through the Account class methods.
    - The account balance is updated after each transaction operation.
- Data Persistence:
    - Account data, including transactions, is saved and loaded using JSON files.

3. File Management

- Account data is saved and loaded using JSON files.
- Files are named based on the user name for easy identification.
- Error handling is implemented to manage file not found, JSON decoding errors, and missing keys during loading.

How it works

The Enhanced Finance Tracker is a Python program designed to manage financial transactions for individual users. It includes features for user registration, login, and tracking financial transactions with associated accounts. The program utilizes JSON for data persistence, the bcrypt library for secure password hashing, and object-oriented programming concepts to model transactions and user accounts.

Classes

1. Transaction Class
    - The Transaction class represents individual financial transactions. Each transaction has attributes such as ID, date, account, amount, notes, and whether it is an expense or not. The class provides methods to convert transaction data to a dictionary (to_dict()) and generate a string representation of the transaction (__str__()).

2. Account Class
    - The Account class represents user accounts and their associated financial data. It includes attributes like user name, password hash, initial balance, a list of transactions, and a file path for data persistence. Methods within the class handle operations such as calculating the account balance, adding, updating, and deleting transactions, as well as saving and loading account data from JSON files.

3. FinanceTracker Class
    - The FinanceTracker class manages user registration, login, and serves as a high-level interface for interacting with user accounts. It utilizes the Account class and bcrypt for secure password handling. The register method creates a new user account, and the

login method allows users to log in to existing accounts. Both methods handle appropriate error checking and provide feedback to the user.

Data Structures

1. Transaction Data Structure
    - A transaction is represented as a Python object of the Transaction class, encapsulating attributes such as ID, date, account, amount, notes, and whether it is an expense or not.

2. Account Data Structure
    - The Account class encapsulates user-specific data, including user name, password hash, initial balance, and a list of transactions. This data is organized and stored in a JSON file format, allowing for easy persistence and retrieval.

Algorithms

1. Password Hashing
    - User passwords are securely hashed using the bcrypt library. This adds a layer of security by storing only the hashed version of passwords in the account data file.

2. Transaction Balance Calculation
    - The calculate_balance method in the Account class computes the account balance by iterating through the list of transactions and adjusting the balance based on whether each transaction is an income or expense.

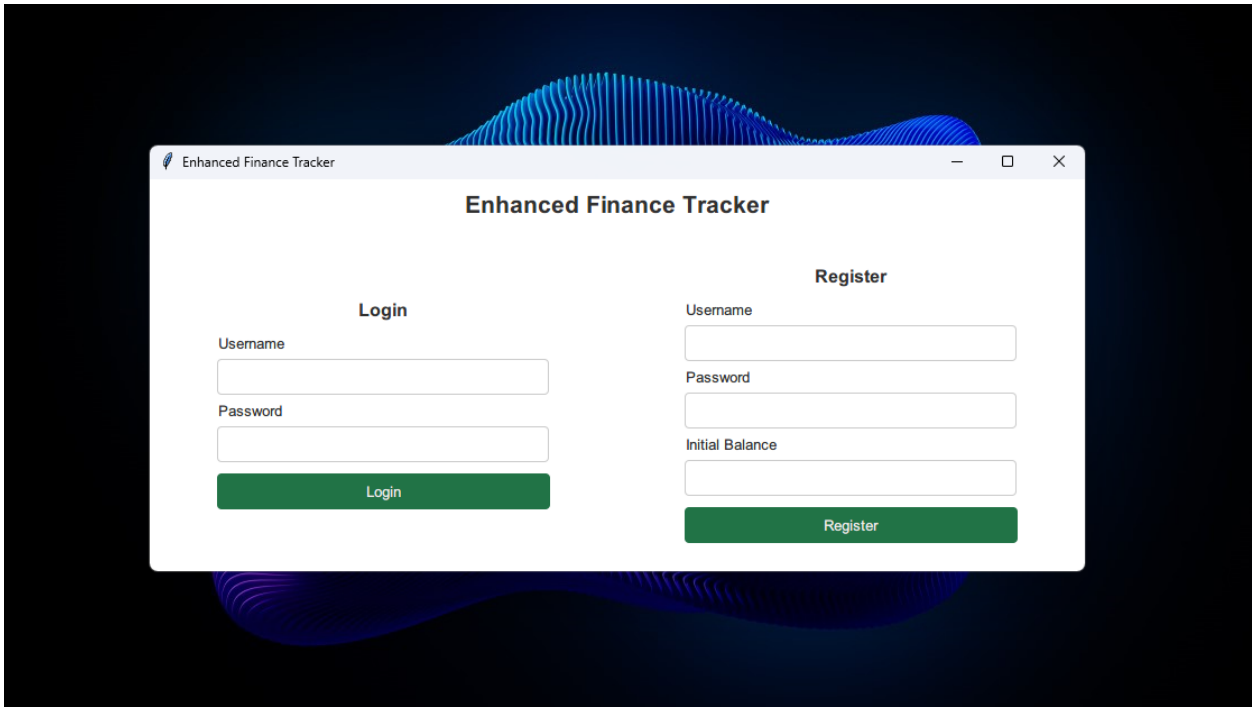
3. User Registration and Login
    - The FinanceTracker class handles user registration and login. During registration, it checks for the existence of an account file, hashes the provided password, creates a new Account object, and saves the data to a JSON file. The login process involves checking the existence of an account file, loading the data, and verifying the password hash for successful login.

File Management

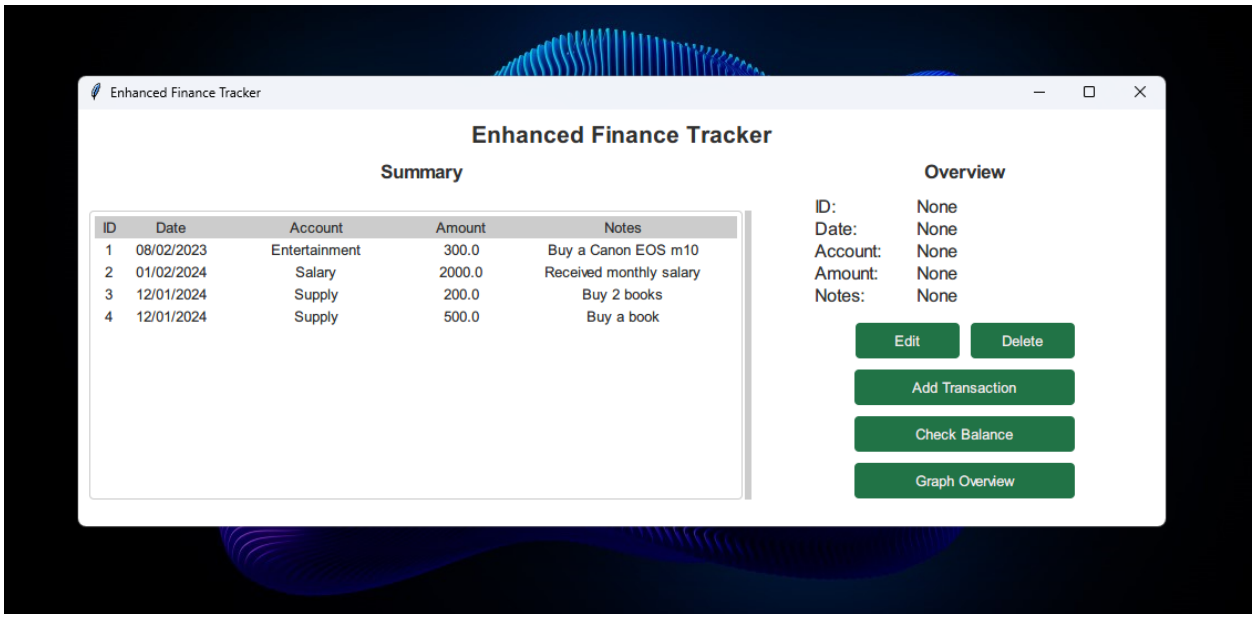
1. JSON Data Persistence
    - User account data, including transactions, is saved and loaded using JSON files. The save_account_data and load_account_data methods in the Account class manage the serialization and deserialization of account information.

2. File Naming Convention
    - Account data files are named based on the user name, following a convention of "{user_name}_account.json". This ensures unique file names for each user.

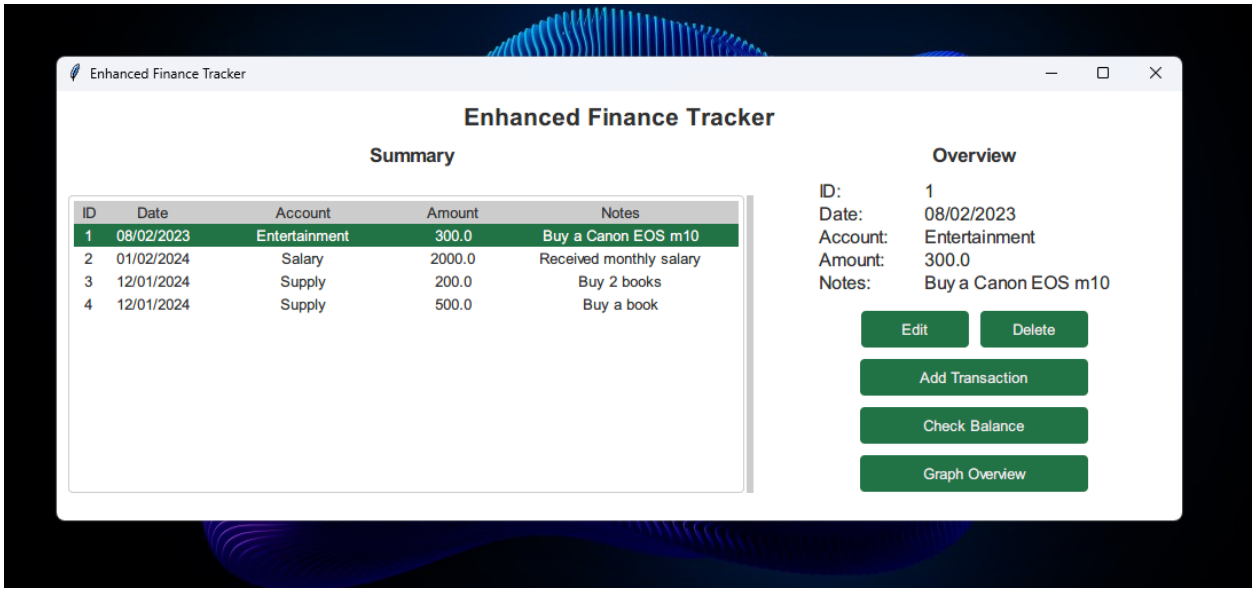
Screenshots



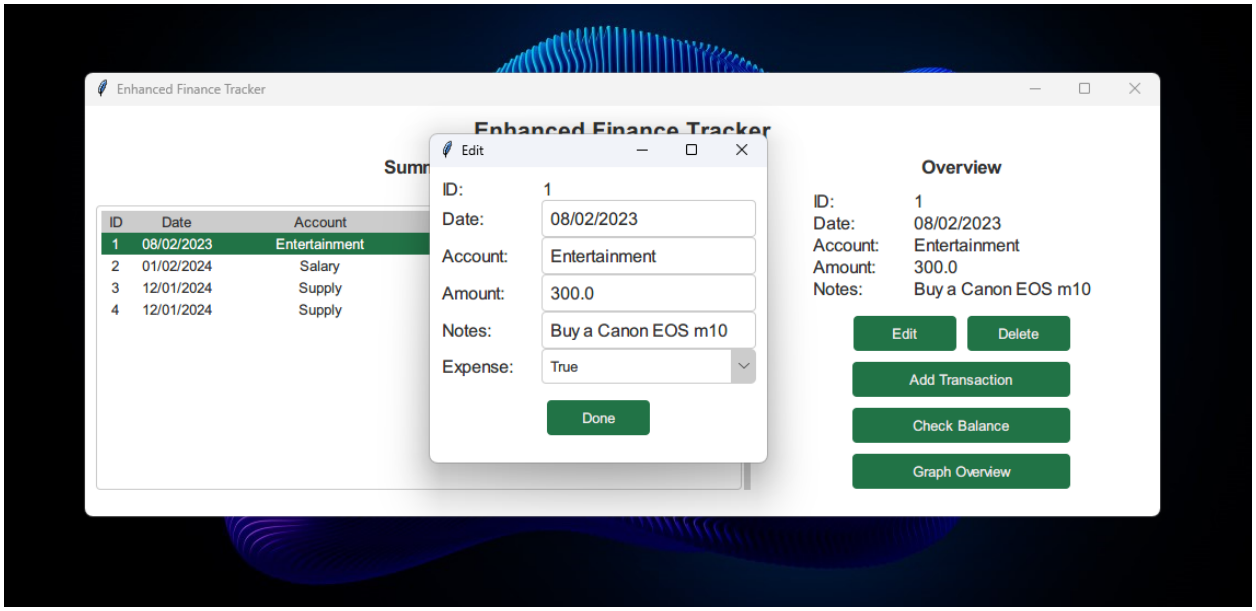
Login/Register Screen on startup



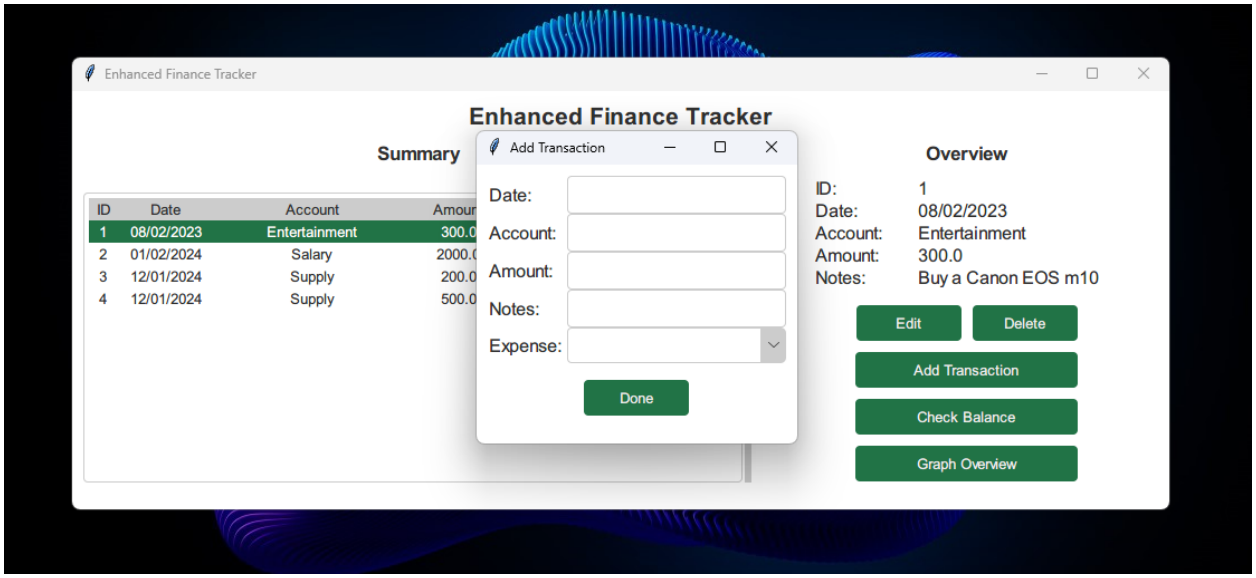
Main Application after logging in



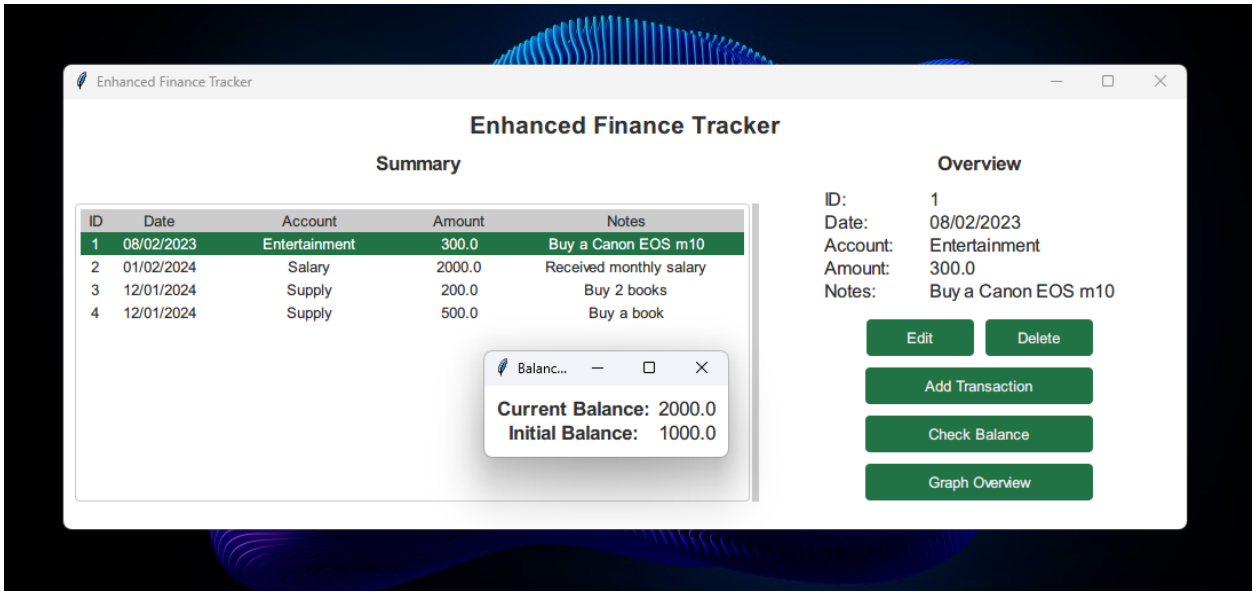
After clicking on a transaction to view it on the right side



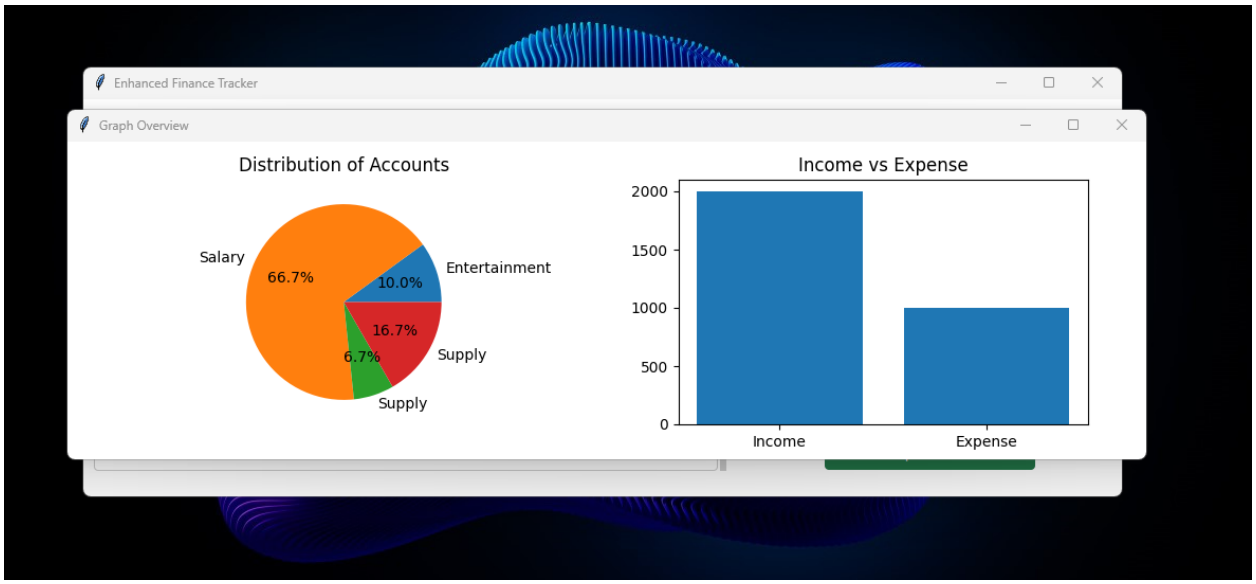
Editing the selected transaction



Adding a new transaction



Check Balance Button for checking the current balance and the initial balance.



Graph Overview Button: Show two graphs, one for the distribution of different accounts in the user's transaction and another for comparing income to expenses.

Conclusion

The Finance Tracker provides a simple and secure solution for users to manage their financial transactions, with an emphasis on data persistence, password security, and user-friendly interaction. Users can register, log in, and track their financial activities seamlessly, aided by the well-structured classes and algorithms implemented in the code.