Banking System - Project Report

Project Title:

Banking System (Using Functions, Dictionaries & File Handling)

Technologies Used:

- Python
- JSON for persistent file storage

Concepts Covered:

- Functions
- Dictionaries
- File Handling
- Basic User Authentication

Project Implementation Summary

Implemented Features:

1. **User Registration:**
    - Allows creation of a new user account.
    - Stores user data including password and starting balance.
2. **User Login (Authentication):**
    - Users log in with a username and password.
    - Only logged-in users can access account features.
3. **Deposit Functionality:**
    - Logged-in users can deposit funds.
    - Balance is updated and saved in the JSON file.
4. **Withdraw Functionality:**
    - Users can withdraw money if they have enough balance.
    - Prevents overdrafts.
5. **Check Balance:**
    - Displays the current account balance.

6. **Transfer Money**

- Enables transferring money between two existing users.
- Validates sufficient balance before allowing the transfer.

7. Generate Monthly Statement (CSV)

- Allows users to generate a bank statement.
- A CSV file is created showing username, balance, and timestamp.

8. **Currency Converter**

- Converts the user's balance (or any custom amount) from one currency to another.
- Fetches real-time exchange rates from the ExchangeRate-API.

9. **Simple Chatbot Assistant**

- Answers basic user questions about how to use the banking system.
- Helps users understand deposit, withdrawal, transfer, and balance check processes

File Handling:

- Account data is stored in a file called users.json.
- Ensures data persistence across sessions.

How different functionalities work:

- When the program starts, it loads all user accounts from users.json.
- A new user can register or an existing user can log in.
- After login, users can perform deposit, withdraw, transfer, check balance, currency conversion, and chatbot interaction.
- The banking menu is available until the user logs out.
- All changes are immediately saved to ensure persistence even if the program closes unexpectedly.

Limitations & Challenges

1. **Plain Text Passwords**
   - Passwords are stored in the JSON file without encryption.
   - Real systems must hash passwords for security.

2. **Input Validation**
    - Some basic input validation (try-except blocks) is done, but more robust validation is needed for better user experience.
3. **Currency Converter Dependency**
    - The currency converter depends on an external API.
    - If the API is unavailable or network connection fails, the feature does not work.
4. **Limited Scalability**
    - Storing data in a JSON file is fine for a few users but not suitable for thousands of users or multi-user concurrent access.
5. **No Transaction History**
    - The system does not store full transaction histories (deposits, withdrawals, transfers) — only the final balance is maintained.
6. **No Two-Factor Authentication (2FA)**
    - Only password-based login is used.
    - Adding email/SMS OTP would greatly improve security.
7. **Command-Line Interface Only**
    - The system currently runs on the terminal.
    - No graphical user interface (GUI) for easier interaction.

.

Challenges Faced During Development:

- **Handling File Format Errors**
    - Ensuring correct reading and writing of users.json without corrupting the file.
- **Managing Currency API Integration**
    - Handling API failures and bad responses required careful exception handling.
- **Structuring the Code**
    - Keeping functions modular (register, login, deposit, withdraw, etc.) for easier debugging and future extension.
- **Dealing with String Formatting**
    - Formatting balance amounts with dollar signs and two decimal places using f-strings.
- **File Overwriting**
    - Making sure new actions append or update user data properly instead of overwriting everything accidentally.

Future Implementation Plan

Planned Enhancements:

- **Password Hashing**
    - Implement bcrypt to hash and verify passwords securely.
    - Protect user credentials from data breaches.
- **Transaction History**
    - Record each transaction (deposit, withdraw, transfer) with timestamp, type, and amount.
    - Display transaction history when the user requests it.
- **Improved Error Handling**
    - Validate all inputs more strictly.
    - Handle API failures gracefully (e.g., try another currency API if one fails).
- **Upgrade to SQLite Database**
    - Replace users.json with an SQLite database.
    - Allow multi-user support and better data integrity.
- **Forgot Password Recovery System**
    - Implement security questions or email OTP to recover lost accounts**.**
- **Two-Factor Authentication (2FA)**
    - Add OTP verification during login for extra security.
- **Graphical User Interface (GUI)**
    - Build a simple GUI with Tkinter for easier use.
    - Alternatively, develop a web application using Flask.
- **Deployment as Web Service**
    - Turn the system into a Flask or FastAPI app.
    - Deploy it online for access via browser or mobile apps.