Republic of the Philippines
Camarines Sur Polytechnic Colleges
Nabua, Camarines Sur
ISO 9001:2015 Certified
COLLEGE of COMPUTER STUDIES

CS 319 - INFORMATION ASSURANCE AND SECURITY

PROJECT REPORT

1. Executive Summary

The current project consists of developing a personal expenses tracker application in terms of being secure, intuitive, and cross-platform in nature. **The Cryptics Legion** enables users to create, log and have a number of accounts and safely track expenses, track budgets and expenses trends. The primary aim was to put the principles of software engineering, in particular, security, decorum design, and data management in practical and meaningful application. Making the necessary adjustments produces a working expense tracker that illustrates the best growth strategies, offline-first design, and incorporation of real-time multi-currency analytics as a Python application developed with Flet.

2. Framework Chosen & Rationale

This project is based on **Flet (Flutter to Python)** as the main framework since it is possible to create cross-platform desktop and mobile applications using Python within a very short time depending on Flet. Flet does not require the native development of an app on Android, iOS, Windows, MacOS, or Linux, so it is not necessary to divert the efforts to those platforms. It is possible to focus on the application logic, security, and data handling. It also encourages the current design of UI, interactive dashboard, and the seamless work of events, which is why it is perfect to create a rich features tracker of expenses.

Supporting tools and technologies were chosen to ensure security, portability, and simplicity:

- **SQLite** for lightweight, offline-first, file-based database storage
- **Fixer.io API** for real-time currency exchange rates
- **python-dotenv** for managing configuration and environment variables
- **Flet CLI** for building and deploying cross-platform applications

These technologies provide a balance of simplicity, functionality, and security, making Cryptics Legion a robust, user-friendly, and reliable expense management platform.

3. Implemented Features

Baseline Features

- User registration, login, and logout
- Password hashing with bcrypt and secure authentication
- Multi-account management (Cash, Bank, Credit/Debit Cards, E-Wallets)
- Expense tracking with categories, dates, and notes
- Real-time currency conversion for ten major currencies
- Offline-first SQLite database storage

- Home dashboard displaying balances, recent transactions, and quick stats
  Transaction history with search and filtering
- Passcode lock for app access

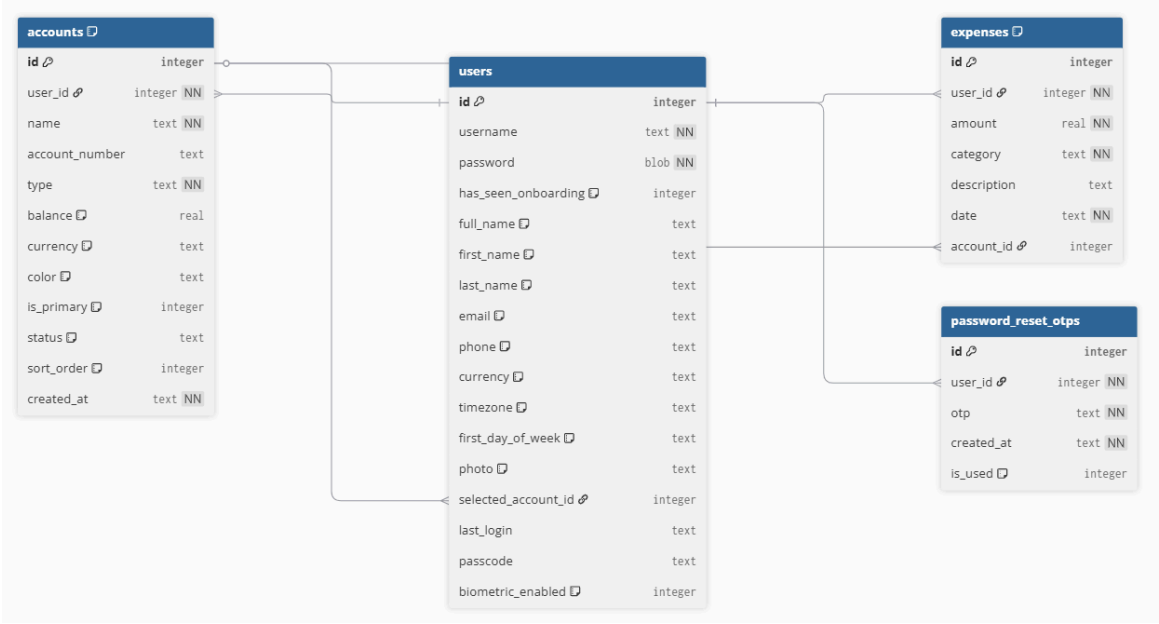
Enhancements Added

- **Advanced expense categorization and custom tags**
- **Analytics and charts for financial insights and trends**
- **Dark theme for improved visual comfort**
- **OTP-based password recovery**
- **Modular architecture separating UI, business logic, utilities, and data layers**
- **Planned features: Biometric authentication, recurring expenses, and budget alerts**

These enhancements make the system more organized, functional, and secure, providing users with a complete solution for tracking, analyzing, and managing their finances effectively.

4. Architecture & Module Overview

The system uses a modular, layered structure that separates the user interface, backend logic, and database. This makes the application easier to understand, maintain, and extend.



Module Breakdown

- UI (Flet): This deals with all screens and elements that the end user interacts with, such as accounts interrupt through authentication pages, accounts dashboards, expense forms, charts, lists, modals, and theme pages. It is smooth and responsive with a Material Design that is available on desktop and mobile.
- Routes: Controls movement between large parts of the app like login, onboarding, accounts overview, expense tracker, settings, and password reset. All routes deal with screen transitions and make sure that users can access only pages that are relevant to them (e.g., logged in, onboarding incomplete).
- Services: This includes the main application logic such as account management, processing of expenses, formatting of currency, updating of user profiles, OTP issuance and validation checks. Other business rules implemented by these service modules include validation of categories, primary account processing and updating balances.
- Models: Performs the layout of the data prior to its being made in the database. Every model corresponds to the tables in your SQLite database users, accounts, expenses, and password resets and provides a consistent validation, types management and objects representation across the application.
- **SQLite Database: Stores persistent user data across four main tables:**
    - **users** – User credentials, preferences, profile data, settings (timezone, currency, first day of week), passcode, and biometric flags.
    - **accounts** – Account records including balances, types (cash, savings, e-wallet, etc.), currency, colors, and creation timestamps.
    - **expenses** – All user expenses, linked to accounts and users, with categories, dates, amounts, descriptions, and tags.
    - **password_reset_otps** – Secure one-time codes for account recovery with timestamps and usage flags.

5. Threat Model & Security Controls

Risks were studied to ensure that security of the application was maintained and addressed the risks by dealing with them in the system. Below, there is the description of the key threats, their potential effects, and the mitigation controls that were implemented in the authentication, account management, and expense tracking modules.

TABLE 1. Threats and Security Controls

Threat	Description	Security Controls / Mitigation
Weak or reused passwords	Users may choose simple or repeated passwords that can be guessed or brute-forced	Enforced minimum password strength, bcrypt hashing, and rejection of overly common passwords
Unauthorized access	Attackers attempt to log in using stolen credentials or guesswork	Passcode support, optional biometric flag, login attempt

		restrictions, session validation
Manipulation of account balances	A malicious actor tries to alter user account balances, statuses, or sort order	Strict backend validation, locked primary account rules, controlled update operations
Fake or modified expenses	Users or attackers insert fraudulent expenses to manipulate reports	Required fields (amount, category, date), validation rules, and owner-check enforcement
SQL Injection	Malicious input attempts to modify SQL queries	Parameterized SQL queries across all CRUD operations
Data tampering in storage	Local database is altered manually or by malware	Hashed credentials, structured schema relationships, and consistent foreign key integrity

This table appropriates the major security concerns that arise as far as a financial tracking system is concerned. The app reduces the risk of unauthorized access to the database, transaction manipulation, and User information safety by taking care of the strength of authentication, database integrity, and OTP safety, and session security. These are protection mechanisms which are coded into the logic layers and validation modules to ensure a secure and reliable system.

6. Design Decisions & Trade-offs

There are some fundamental choices which affected the design of the application. The app uses the Flet interface because it is decidedly simple and capable of providing a native-like interface with the help of Python only. Although this accelerates development as well as provides a uniform appearance across platforms, it is associated with a disadvantage of little detailed control over the low-level manipulation and session strategies.

The data server relies on **SQLite** which was chosen due to its lightweight nature and its ease of deployment with local user data. This makes it perfect to use the app in tracking personal finance on one device, but would also make the app inherently limited in terms of concurrency, real time synchronization, and the number of users being supported.

An authentication, account, expenses, and OTP recovery service are divided into separate components of the structure based on the modular structure. It enhances testing and maintainability but makes it hard to test in comparison to monolithic code. Three-way authentication with security options like bcrypt hashing, OTP expiration implementation and stringent validation and reliability at the expense of additional implementation time.

In general, both design decisions are better suited to the constraints of portability, fast development as well as offline usability of a personal finance tool, at the cost of heavy-duty capabilities needed by large-scale distributed systems.

7. Limitations & Future Work

Limitations

Although the app has been able to deliver major functionalities like account monitoring, expense management, and safe password activation, there exist significant drawbacks that have an impact on the production maturity of the application. The convenient SQLite is not ideal to use to sync and have many devices or to act multi-core, which confines the application to the local environment usage. The system also lacks multi-factor authentication, device verification, or anomaly detection, which has become common in the modern finance tools.

The storage of backup and logs is performed on the local disk, and hence this data is subjected to manipulation in case the device is impaired. The modern UI still remains functional but does not have the dynamic animations, responsive templates, and customization features that other, more advanced personal finance apps offer. Moreover, certain sophisticated functionalities, including automatic budgeting, revenue monitoring, repetitive costs and real-time currency translation are not fully adopted.

Future Work

Several enhancements can significantly improve scalability, security, and user experience. Migrating the backend from SQLite to a more robust database such as PostgreSQL or Supabase would allow cloud syncing and multi-device access. Integrating multi-factor authentication, email verification, and device fingerprinting would provide stronger protection for user accounts.

The UI can be upgraded with better visual dashboards, animations, dark mode improvements, and customizable themes. Adding intelligent features such as spending predictions, category insights, budgeting goals, and auto-categorization using machine learning could expand usability. Enhancing backups through encrypted cloud storage or dedicated secure vaults would increase reliability. Finally, expanding the infrastructure with Docker, CI/CD pipelines, and modular API endpoints would bring the system closer to real production standards.