**Credit Card Vault System Documentation**



Table of Contents

[1. Introduction 2](#_Toc195735335)

[1.1 Purpose 2](#_Toc195735336)

[1.2 Scope 2](#_Toc195735337)

[1.3 Target Audience 2](#_Toc195735338)

[1.4 System Overview 3](#_Toc195735339)

[2. System Architecture 3](#_Toc195735340)

[2.1 High-Level Architecture 3](#_Toc195735341)

[2.2 Component Diagram 3](#_Toc195735342)

[2.3 Database Design 4](#_Toc195735343)

[2.4 Security Architecture 4](#_Toc195735344)

[3. Functional Description 5](#_Toc195735345)

[3.1 User Authentication System 5](#_Toc195735346)

[3.2 Card Management System 5](#_Toc195735347)

[3.3 Admin Control Panel 6](#_Toc195735348)

[3.4 Security Features 7](#_Toc195735349)

[4. System Design 7](#_Toc195735350)

[4.1 Class Diagram 7](#_Toc195735351)

[4.2 Data Flow Diagrams 9](#_Toc195735352)

[4.4 Entity Relationship Diagram 10](#_Toc195735353)

[4.5 Database Schema 11](#_Toc195735354)

[5. Implementation 11](#_Toc195735355)

[5.1 Technology Stack 11](#_Toc195735356)

[5.4 User Interface 12](#_Toc195735357)

[6. Testing Strategy 12](#_Toc195735358)

[6.1 Test Planning 12](#_Toc195735359)

[6.2 Test Cases 13](#_Toc195735360)

[6.3 Test Execution 13](#_Toc195735361)

[6.4 Test Results 14](#_Toc195735362)

[7. User Guide 14](#_Toc195735363)

[7.1 Installation 14](#_Toc195735364)

[8. Quality Plan 15](#_Toc195735365)

[**Quality Plan for Credit Card Vault App** 15](#_Toc195735366)

[9. Appendices 17](#_Toc195735367)

# 1. Introduction

## 1.1 Purpose

The Credit Card Vault System is designed to securely store and manage credit card information. It provides a secure environment for users to store their credit card details with encrypted protection, ensuring sensitive financial information remains protected from unauthorized access.

## 1.2 Scope

This document provides a comprehensive description of the Credit Card Vault System, including its architecture, functionality, implementation details, testing procedures, and user guide. The document serves as a reference for developers, testers, and users of the system.

## 1.3 Target Audience

This documentation is intended for:

- Software developers maintaining or extending the system

- Quality assurance personnel testing the system

- System administrators configuring and managing the system

- End users who need to understand the system's capabilities and operation

## 1.4 System Overview

The Credit Card Vault is an application built with Python and Tkinter that provides secure storage of credit card information. The system implements role-based access control with three user types: admin, merchant, and customer. All sensitive credit card data is encrypted using Fernet symmetric encryption, and user passwords are secured with SHA-256 hashing. The application connects to a MySQL database for data persistence.

Key features include:

- Secure user authentication

- Role-based access control

- Encrypted storage of credit card details

- Card information masking for display

- User management for administrators

- Simple and intuitive graphical user interface

# 2. System Architecture

## 2.1 High-Level Architecture

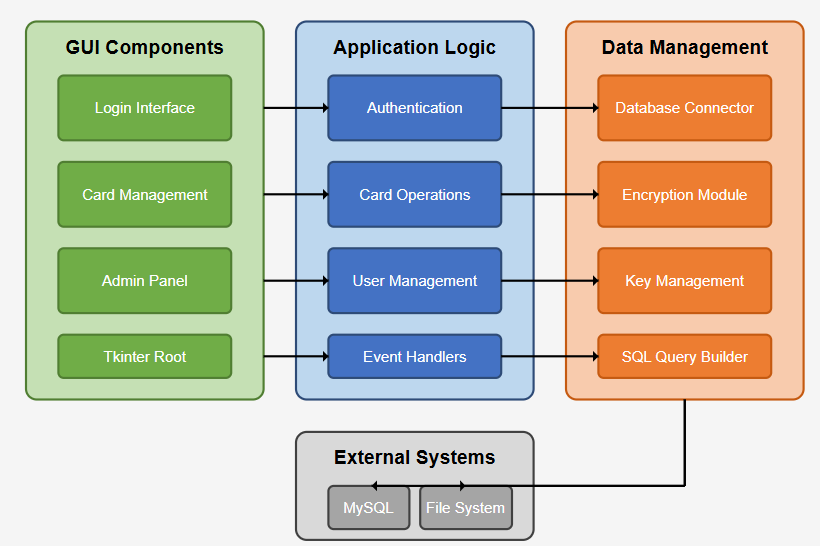
The Credit Card Vault system follows a three-tier architecture:

1. Presentation Layer: A Tkinter-based GUI that provides interfaces for login, card management, and administrative functions.

2. Application Layer: Core business logic implemented in Python, handling user authentication, encryption/decryption, and card management operations.

3. Data Layer: MySQL database that stores user information and encrypted credit card data.

## 2.2 Component Diagram



## 2.3 Database Design

The system uses a MySQL database with two primary tables:

1. Users Table - Stores user authentication and role information

2. CreditCards Table - Stores encrypted credit card information

## 2.4 Security Architecture

The security architecture includes multiple layers of protection:

1. Authentication Layer: Verifies user identity through username/password combinations, with passwords stored as SHA-256 hashes.

2. Authorization Layer: Role-based access control ensures users can only access operations appropriate for their role.

3. Encryption Layer: Uses Fernet symmetric encryption to protect sensitive card data at rest.

4. Input Validation: Parameterized queries prevent SQL injection attacks.

5. Card Data Masking: Credit card numbers are masked when displayed, showing only the last four digits.

# 3. Functional Description

## 3.1 User Authentication System

The authentication system handles user login, verification, and session management.

**Key Functions**

- User login with username and password

- Password hashing using SHA-256

- Role-based access control

- Session management to track current user and role

**Workflow**

1. User enters username and password

2. System hashes the password

3. System verifies credentials against the database

4. Upon successful authentication, the system loads appropriate screens based on user role

5. Failed authentication attempts display error messages

## 3.2 Card Management System

The card management system allows users to add, view, and delete credit card information.

**Key Functions**

- Add new credit cards with encryption

- View stored cards with proper masking

- Delete stored cards

**Workflow for Adding Cards**

1. User enters card details (name, number, expiration date, CVV)

2. System encrypts sensitive fields (card number, CVV)

3. System stores encrypted data in the database

4. Confirmation message displayed to user

**Workflow for Viewing Cards**

1. User requests to view stored cards

2. System retrieves encrypted card data

3. System decrypts data

4. System masks card numbers (shows only last 4 digits)

5. System displays masked information

## 3.3 Admin Control Panel

c

The admin panel provides administrative functions for user and card management.

**Key Functions**

- View all system users

- View all stored cards across users

Access Control

- Only users with the 'admin' role can access these functions

- Attempts by non-admin users to access these functions are blocked

## 3.4 Security Features

The system implements several security features to protect sensitive data.

**Key Features**

- Fernet symmetric encryption for card data

- SHA-256 hashing for passwords

- Parameterized SQL queries to prevent injection attacks

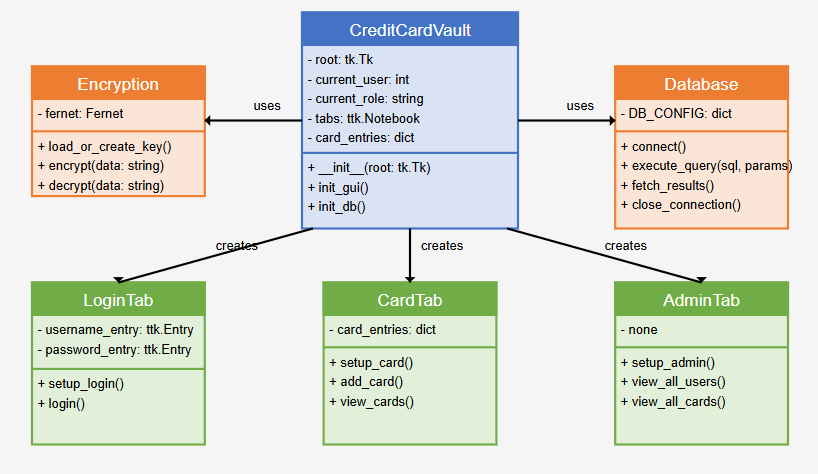
- Card number masking in display

- Role-based access control

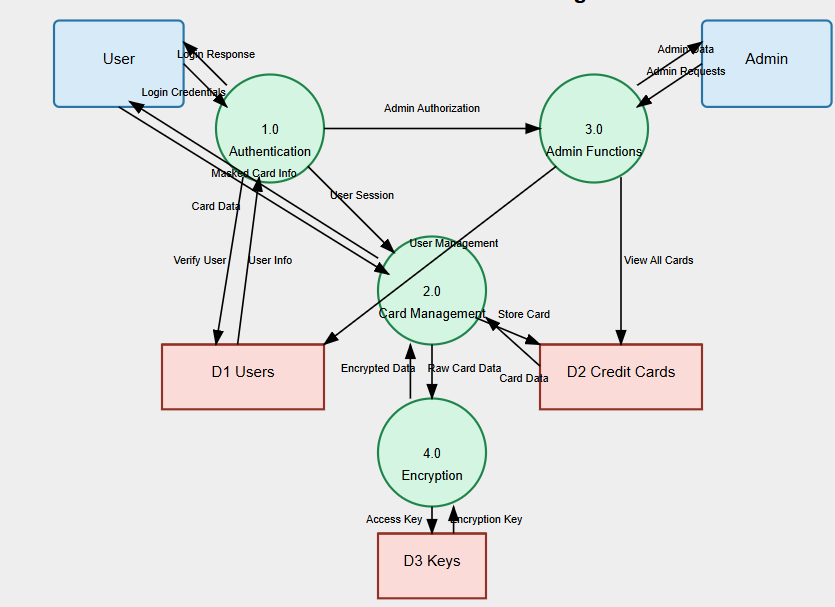
- Secure key management

# 4. System Design

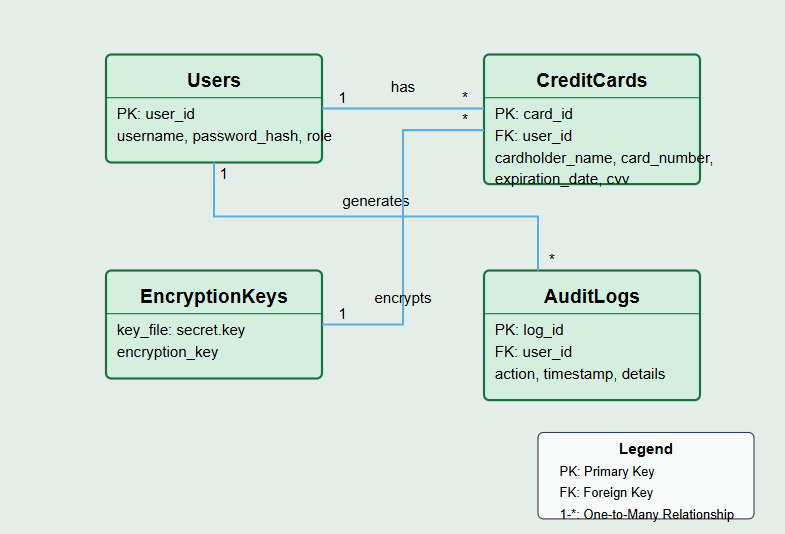
## 4.1 Class Diagram



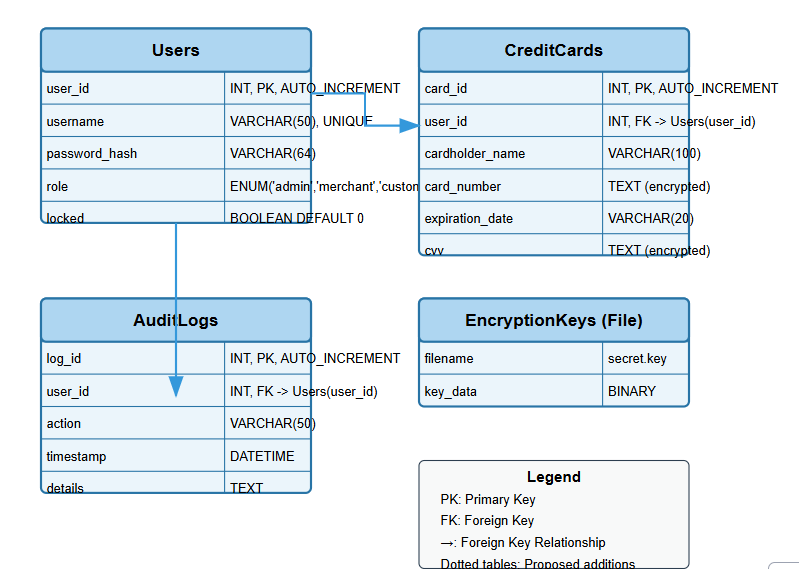
## 4.2 Data Flow Diagrams



## 4.4 Entity Relationship Diagram



## 4.5 Database Schema



# 5. Implementation

## 5.1 Technology Stack

The Credit Card Vault system is implemented using the following technologies

- Programming Language: Python 3.x

- GUI Framework: Tkinter

- Database: MySQL

-Encryption: cryptography.fernet (symmetric encryption)

- Password Hashing: hashlib (SHA-256)

- Database Connector: mysql-connector-python

## 5.4 User Interface

The user interface is implemented using Tkinter and consists of three main tabs

1. Login Tab: Provides fields for username and password entry, with a login button.

2. Card Management Tab: Contains fields for entering card details and buttons for managing cards.

3. Admin Panel Tab: Provides administrative functions like viewing all users and cards.

# 6. Testing Strategy

## 6.1 Test Planning

The testing strategy for the Credit Card Vault system includes

1. Unit Testing: Testing individual components and functions in isolation.

2. Integration Testing: Testing interactions between components.

3. System Testing: Testing the complete system from end to end.

4. Security Testing: Specific tests for security features.

Test Environment

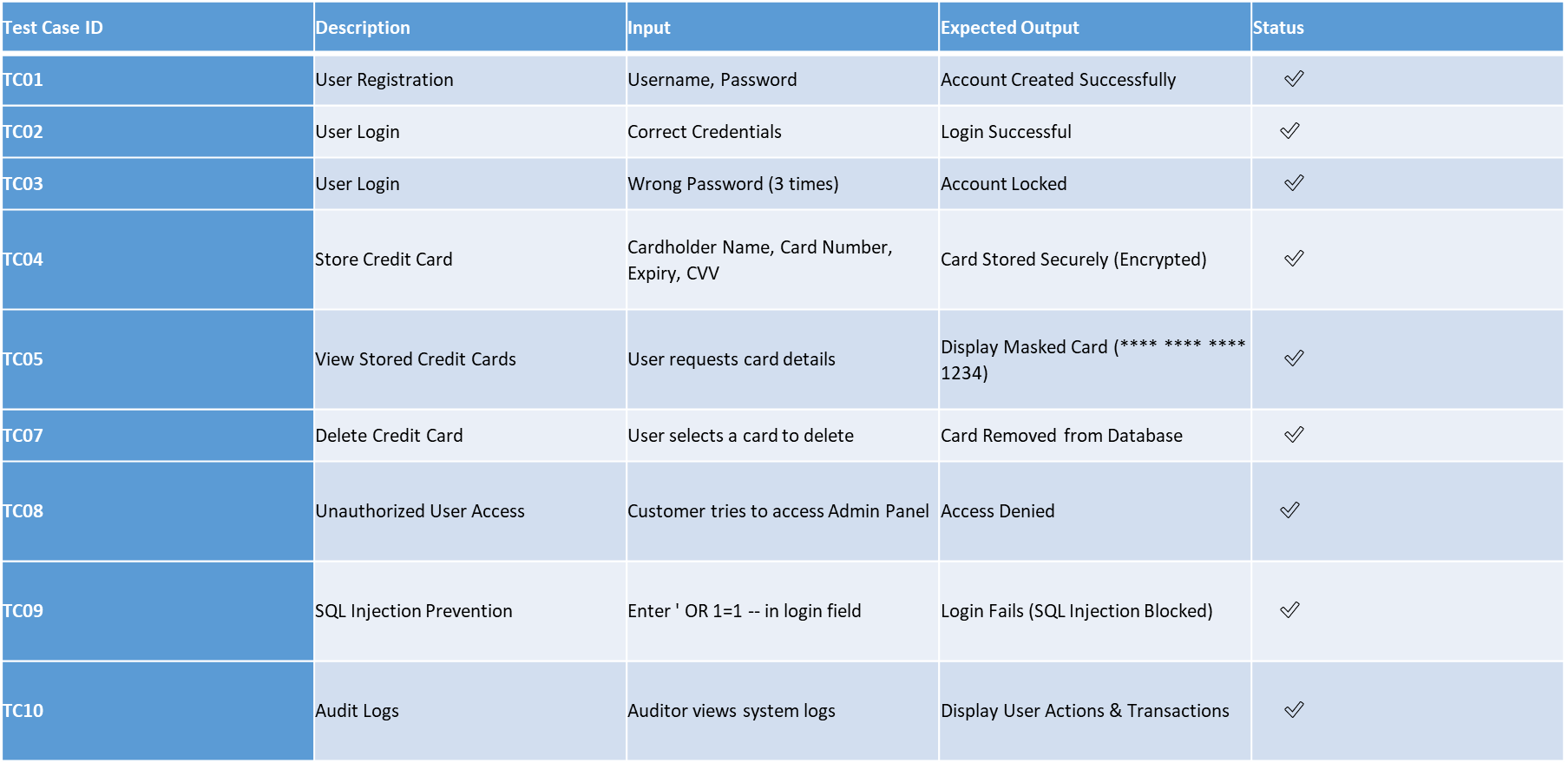
- Development environment with local MySQL database

- Python unittest framework for automated tests

- Manual testing for UI components

## 6.2 Test Cases

The system has been tested with the following test cases:



## 6.3 Test Execution

The automated tests are implemented using Python's unittest framework.

1. Authentication Tests

- Test user login with correct credentials

- Test login failure with incorrect credentials

- Test account locking after multiple failures

2. Card Operation Tests

- Test adding cards with encryption

- Test viewing cards with masking

- Test deleting cards

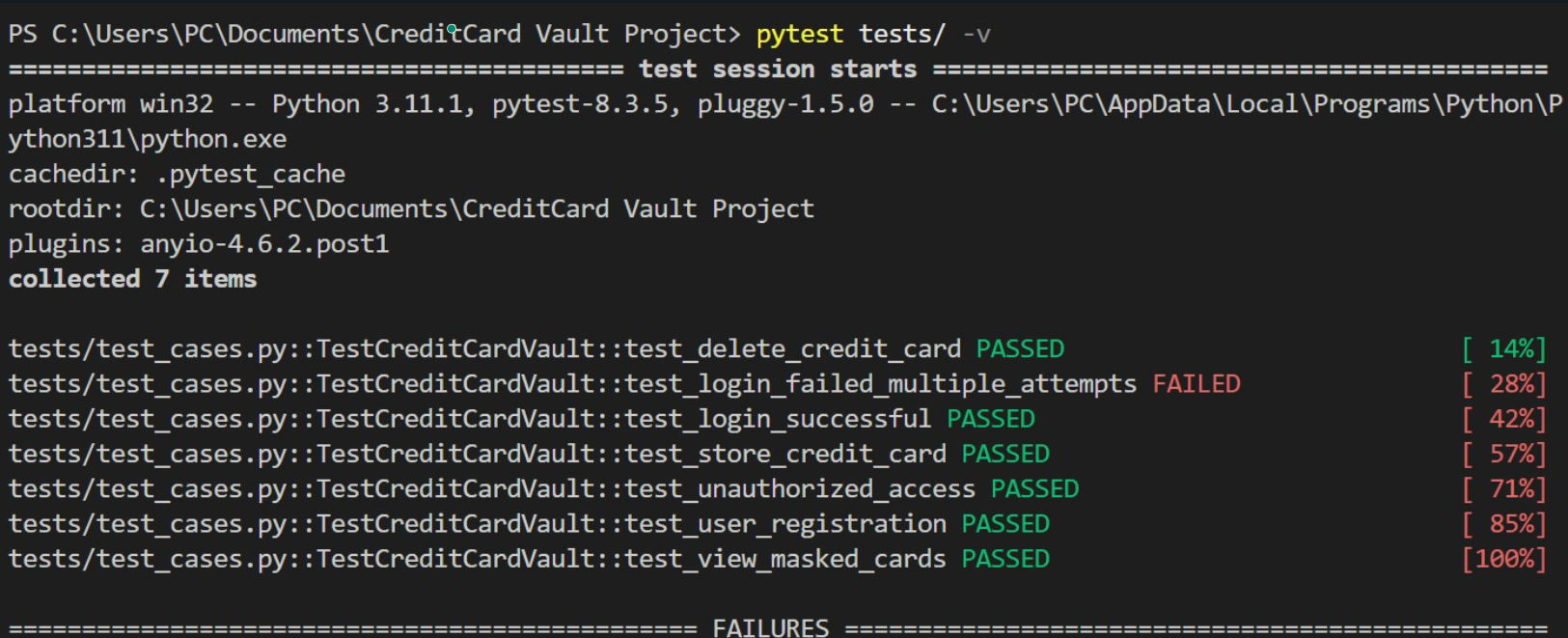
3. Security Tests

- Test encryption/decryption process

- Test role-based access control

- Test SQL injection prevention

## 6.4 Test Results



# 7. User Guide

## 7.1 Installation

To install the Credit Card Vault system

1. Prerequisites

- Python 3.6 or higher

- MySQL Server

- Required Python packages: tkinter, mysql-connector-python, cryptography

2. Installation Steps

- Clone or download the application source code

- Install required Python packages:

pip install mysql-connector-python cryptography

- Create a MySQL database named ""

- Configure database connection settings in app.py

3. First Run

- Run the application:

python app.py

- The system will automatically create necessary tables and an admin user

- Default admin credentials: username "admin", password "admin123"

# 8. Quality Plan

# **Quality Plan for Credit Card Vault App**

1. Product Introduction

The Credit Card Vault App is an application developed in Python with a Tkinter GUI. It securely stores users' credit card information in an encrypted format using the Fernet symmetric encryption method and authenticates users through role-based access (Admin, Merchant, Customer). The system uses a MySQL backend and enforces strict access policies to ensure data confidentiality and integrity.

2. Product Plan

• Phase 1: Define system requirements and use cases

• Phase 2: Develop GUI layout and database schema

• Phase 3: Implement user registration and login with hashed passwords

• Phase 4: Integrate encryption for card data storage

• Phase 5: Role-based access to card and user management dashboards

• Phase 6: Internal and external testing (unit + integration)

• Phase 7: Prepare documentation and finalize deployment using optional Docker containerization

3. Quality Goals

Attribute Description

Security Encrypt sensitive data using Fernet; enforce role-based access

Usability Simple, clean GUI with tabs specific to user roles

Performance Fast data access and GUI responsiveness under normal load

Reliability Robust against invalid inputs and improper operations

Maintainability Modular code with comments and separation of concerns

Scalability Easily adaptable for web version or cloud-hosted database

4. Risks and Risk Management

|  |  |  |  |
| --- | --- | --- | --- |
| Risk | Likelihood | Impact | Mitigation Strategy |
| Key file compromise | Medium | High | Store secret.key with restricted permissions |
| SQL Injection | Medium | High | Use parameterized queries (already implemented) |
| Admin panel access by non-admin | Medium | High | Role validation on both client and server sides |
| Data corruption | Low | High | Validate encryption/decryption operations |
| GUI crash or error | Medium | Medium | Use try-except blocks and test thoroughly |

5. Means in the System and How to Assess

Feature Assessment Method

|  |  |
| --- | --- |
| Role-based dashboard | Login as different roles and test visible tabs |
| Fernet encryption of card data | Encrypt-decrypt test with assertions |
| Login validation | Attempt logins with valid/invalid credentials |
| GUI layout and responsiveness | Manual UI walkthrough on different systems |
| Database operations | Run all CRUD operations and verify success |

6. Inspection Checklist

Inspection Item Status

|  |
| --- |
| 1 App launches without error Yes |
| 2 GUI tabs change based on login role Yes |
| 3 Password hashing implemented and working Yes |
| 4 Encryption key generated and stored once Yes |
| 5 Card number and CVV encrypted before storage Yes |
| 6 Only Admin can view all users and all card details Yes |
| 7 Decrypted values are accurate and reversible Yes |
| 8 All fields are validated and sanitized Yes |
| 9 SQL queries use prepared statements Yes |
| 10 User inputs do not crash the system Yes |
| 11 Interface labels are clear and consistently styled Yes |
| 12 All card numbers are masked when displayed Yes |
| 13 Admin user is pre-seeded and secure Yes |
| 14 GUI fields clear after successful operations Yes |

# 9. Appendices

9.1 Glossary

Fernet - A symmetric encryption method that guarantees that a message encrypted using it cannot be manipulated or read without the key |

SHA-256 - A cryptographic hash function that generates a 256-bit hash value

Role-based access control - A method of regulating access to system resources based on the roles of individual users

SQL Injection - A code injection technique that exploits vulnerabilities in data-driven applications

CVV- Card Verification Value, a security feature for credit cards

9.2 References

1. Python Tkinter Documentation: https://docs.python.org/3/library/tkinter.html

2. MySQL Connector/Python Documentation: https://dev.mysql.com/doc/connector-python/en/

3. Cryptography Package Documentation: https://cryptography.io/en/latest/

4. PEP 8 Style Guide: https://www.python.org/dev/peps/pep-0008/