Source Code: AI Generated Report

Generated on: 2026-02-07 16:43:43

```
import sqlite3
from sqlite3 import Error
from cryptography.fernet import Fernet
import logging

# Setup logging
logging.basicConfig(level=logging.INFO)

# Database setup
def create_connection():
    conn = None
    try:
        conn = sqlite3.connect(':memory:')
        logging.info("Connection to SQLite DB successful")
    except Error as e:
        logging.error(f"Error connecting to database: {e}")
    return conn

def create_table(conn):
    try:
        sql_create_passwords_table = """ CREATE TABLE IF NOT EXISTS passwords (
                                        id integer PRIMARY KEY,
                                        service_name text NOT NULL,
```

```
                                        encrypted_password text NOT NULL
                                    ); """
        cursor = conn.cursor()
        cursor.execute(sql_create_passwords_table)
        logging.info("Passwords table created successfully")
    except Error as e:
        logging.error(f"Error creating table: {e}")

# Encryption utilities
def create_key():
    return Fernet.generate_key()

def encrypt_password(key, password):
    try:
        fernet = Fernet(key)
        encrypted = fernet.encrypt(password.encode())
        return encrypted.decode()
    except Exception as e:
        logging.error(f"Error encrypting password: {e}")
        return None

# API logic
def save_password(conn, service_name, password):
    key = create_key()
    encrypted_password = encrypt_password(key, password)
    if encrypted_password:
```

```
        try:
            sql = ''' INSERT INTO passwords(service_name, encrypted_password)
                      VALUES(?, ?) '''
            cursor = conn.cursor()
            cursor.execute(sql, (service_name, encrypted_password))
            conn.commit()
            logging.info("Password saved successfully")
            return "Password saved successfully!"
        except Error as e:
            logging.error(f"Error saving password: {e}")
            return "Error saving password"
    return "Error encrypting password"


def get_stats(conn):
    try:
        cursor = conn.cursor()
        cursor.execute("SELECT COUNT(*) FROM passwords")
        total_services = cursor.fetchone()[0]

        cursor.execute("SELECT AVG(LENGTH(encrypted_password)) FROM passwords")
        average_password_length = cursor.fetchone()[0] or 0

        return {
            "total_services": total_services,
            "average_password_length": average_password_length
        }
```

```
    except Error as e:
        logging.error(f"Error retrieving stats: {e}")
        return {"total_services": 0, "average_password_length": 0}

# Main execution
if __name__ == "__main__":
    conn = create_connection()
    create_table(conn)

    # Mock data
    print(save_password(conn, "example_service", "my_secure_password"))
    stats = get_stats(conn)
        print(f"Total services: {stats['total_services']}, Average password length:
{stats['average_password_length']:.2f}")
```