Torin Flanagan: 1169130

Due Date: 19/09/2025

Word Count: 1,152

Course Coordinator: Andrew Lang

Tutor: Shena Machin

Task 2: Key-Value Database

ICT 320 - Database Programming

Contents

[Introduction 1](#_Toc209191914)

[Login Page Flow Diagram: 2](#_Toc209191915)

[Design 3](#_Toc209191916)

[Data Storage Model: 3](#_Toc209191917)

[Core Functions: 3](#_Toc209191918)

[Design Choices and Rationale: 3](#_Toc209191919)

[Testing 4](#_Toc209191920)

[Visual Testing Tkinter GUI: 4](#_Toc209191921)

[Automated Tests: 7](#_Toc209191922)

[Results: 7](#_Toc209191923)

[Automated Test Python Script And Terminal Output: 8](#_Toc209191924)

[Unit Tests: 9](#_Toc209191925)

[Results: 9](#_Toc209191926)

[Unit Test Python Script And Terminal Output: 11](#_Toc209191927)

[Future 12](#_Toc209191928)

[References 12](#_Toc209191929)

**I acknowledge the use of ChatGPT in this assessment for helping develop the code used to generate the python script.**

# Introduction

For the *Find a Campsite* service, a secure login system was developed using Python for scripting, Redis as the database, and Tkinter for the GUI. Redis, an in-memory key-value database, enables real-time data handling with millisecond responses (Redis, 2025). Tkinter, a built-in Python library, provides a lightweight GUI wrapper ideal for quick development (GeeksforGeeks, 2017). Both were integrated into the Python script to support account creation, login, and password recovery.

The system first connects to Redis to store all user records. A CSV file containing initial user details (username, password, first name) is loaded, with passwords encrypted using AES-based Fernet encryption. Fernet was chosen for its symmetric authenticated cryptography, requiring a secret key for secure access (Cryptography, 2014). Each account is also assigned a randomised security question for password recovery.

The GUI delivers three key functions:

1. Login – users authenticate with email and password, verified against encrypted Redis records.
2. Create Account – new users register with email, name, password (encrypted), and a security question.
3. Forgot Password – existing users reset passwords by correctly answering their security question; CSV-imported users also complete this step as they bypass account creation.

Testing combined automated tests, which validated functionality before execution (Gillis, 2023), and unit tests, which checked isolated functions such as account creation, login, and password reset using controlled inputs and asserted outputs (AWS, 2024).

Overall, the system integrates database storage, encryption, and testing within a single Python application. Redis ensures fast, scalable data access, while security is strengthened through encrypted passwords and recovery questions, delivering a reliable and interactive login solution.

For the script to run smoothly, ***Python 3*** must be installed alongside all required libraries. Imports including ***tkinter***, ***random***, ***os***, ***csv***, and ***unittest*** come pre-installed with Python. The ***redis***and ***cryptography***libraries aren’t built in and must be installed with ***pip install redis***and ***pip install cryptography***. Additionally, a running Redis server is also required for database connection. With these dependencies met, the script will function correctly without further configurations.

## Login Page Flow Diagram:

# Design

Data Storage Model:The system uses a Redis key-value structure, where each user account is stored as a hash object with the email as the unique key and associated values including first name, encrypted password, security question, and answer. Security questions are stored in a Redis list for random allocation, while a simple string confirms Python–Redis connectivity.

|  |  |
| --- | --- |
| **Field** | **Value** |
| first\_name | Test |
| security\_question | What was your favourite place to visit as a child? |
| Password | gAAAAABoy9u3O-qJ4odqVxEhqaZ-A-vldKcNSzw9UohrU048BuaZjjSmL6WmJEyBQeFxPvf8auMcbdIkOUMtMknrywJd2le4Lg== |
| security\_answer | Test |

Key: user:ABC@email.com

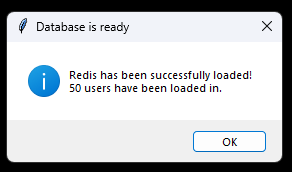
## Core Functions:

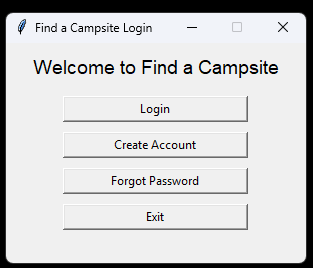
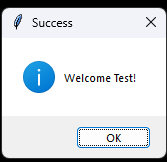
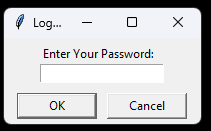
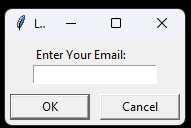
* **Account Creation;** stores email as the key and records first name, encrypted password, and a randomly assigned security question with the entered answer.
* **Login;** verifies the entered email and password against stored values.
* **Forgot Password;** retrieves the user’s security question, validates the answer, and allows new password update. For CSV imported users, they are assigned a random security question due to not having an answer allocated to their random security question yet.
* **Exit;** terminates the program when selected.

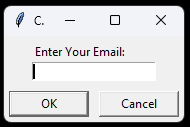
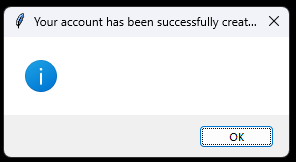
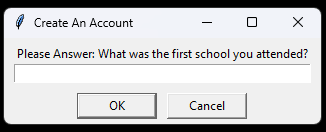
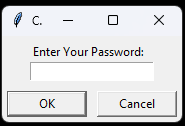
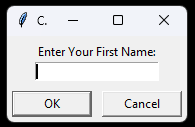
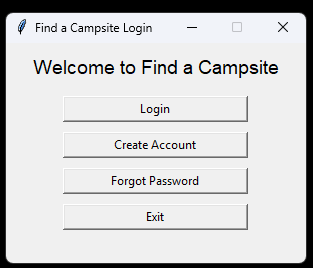
Design Choices and Rationale:Redis was selected for its required use, speed, and efficient key-value model suited to login operations. A CSV file is imported to test database integration with the Python program, supporting smooth data updates. AES Fernet encryption secures stored passwords, ensuring confidentiality of credentials. Tkinter provides a user-friendly GUI instead of a command-line interface. Automated and unit testing ensure each function operates correctly and reliably.

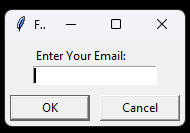
# Testing

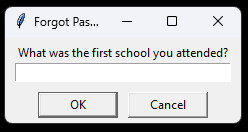
## Visual Testing Tkinter GUI:

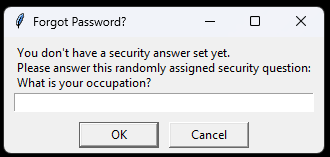
Initial data load:

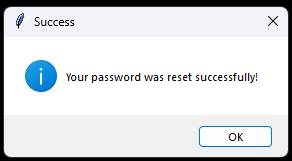
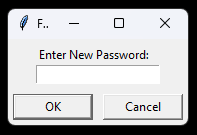
Step-by-step login process:

****Step-by-step create account process:

Step-by-step forgot password process:

If previously filled mandatory security question.

Otherwise, if haven’t previously filled mandatory security question.

Continues as normal.

## Automated Tests:

Core login functions were tested to ensure correct behaviour under defined conditions. Account creation verified that new records were initialised with proper fields, login was validated through both successful and failed attempts, and password resets confirmed that encrypted passwords were updated correctly. Mock data simulated user input without GUI interaction, with passwords stored, retrieved, and compared for accuracy. Conditional checks asserted outcomes, reported as either [PASS] or [FAIL].

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Input** | **Expected Output** | **Result in Script** |
| Account Creation | username = “test@email.com”  first\_name = “Test”  password = “Password123”  security\_question = random  security\_answer = “Answer123” | first\_name (and other fields) stored correctly | [PASS] Account Creation |
| Successful Login | Password = “Password123” (matches encrypted password) | Decrypted password equals input | [PASS] Successful Login |
| Failed Login | wrong\_pass = “WrongPass” | Decrypted password doesn’t equal input | [PASS] Login Failure Was Correctly Ejected. |
| Forgot Password | new\_password = “NewPassword123” | Decrypted password equals updated password | [PASS] Forgot Password |

## Results:

1.  **Account Creation** – validates new records contain the correct fields; fails if any value is missing or incorrect.
2.  **Successful Login** – confirms encrypted passwords can be decrypted and matched with valid input.
3.  **Failed Login** – ensures incorrect passwords are rejected, validating authentication logic.
4.  **Forgot Password** – checks password reset correctly encrypts and updates stored credentials.

## A screen shot of a computer AI-generated content may be incorrect.Automated Test Python Script And Terminal Output:

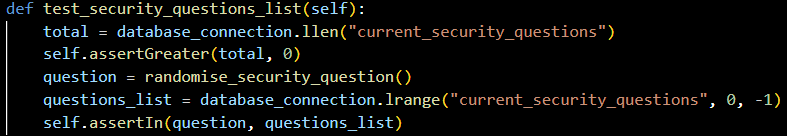
## Unit Tests:

Unit tests validated individual components of the login system in isolation, avoiding GUI interaction. Account creation checked that user data was stored in Redis with proper encryption, the security questions list was verified to exist and return valid random entries, and login success ensured encrypted passwords matched valid input while rejecting incorrect ones. Mock data was inserted directly into Redis, retrieved, and validated using assertions such as assertEqual, assertIn, and assertGreater. Test data was removed after each run to keep the database clean and tests repeatable.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Input** | **Expected Output** | **Result in Script** |
| Account Creation | username="unit\_test\_account"  first\_name="Unit"  password="UnitPass123"  security\_question=random  security\_answer="UnitAnswer" | Redis hash stores correct values matching input | OK |
| Security Questions List | Redis list current\_security\_questions | Length of list > 0, randomised question exists in list | OK |
| Login Success | username="unit\_test\_login", password="LoginPass123" | Decrypted password equals input, does not equal “WrongPass” | OK |

## Results:

1. A computer screen with text on it

   AI-generated content may be incorrect. **Account Creation** – confirms data is stored correctly and encrypted passwords can be decrypted for validation.
2.  **Security Questions List** – verifies the Redis list is populated, and the randomisation function only selects valid entries.
3. A screen shot of a computer code

   AI-generated content may be incorrect. **Login Success** – ensures valid credentials are authenticated and incorrect ones are rejected, protecting system integrity.

**A black background with white text

AI-generated content may be incorrect.Important** – Cleanup is performed after each test to remove inserted data from Redis, ensuring tests remain isolated and repeatable.

## A screen shot of a computer program AI-generated content may be incorrect.Unit Test Python Script And Terminal Output:

A black background with white and yellow lines

AI-generated content may be incorrect.

# Future

Future improvements for the *Find a Campsite* login system could include:

* Adding a function to export newly added user data from Redis to a CSV, preserving appended security questions and answers.
* Developing the GUI into a full web application, similar to modern login pages.
* Exploring alternative data storage methods, such as document databases, to compare functionality with Redis.
* Linking the login system to the Find a Campsite website, extending its function beyond basic authentication.
* Implementing multi-factor authentication (MFA) to enhance security using multiple forms of verification.
* Separating the script into a dedicated function file and a testing file, following best practices in software development.

# References

* AWS. (2024). What is Unit Testing? - Unit Testing Explained - AWS. Amazon Web Services, Inc. https://aws.amazon.com/what-is/unit-testing/
* Cryptography. (2014). Fernet (symmetric encryption) — Cryptography 2.9.dev1 documentation. Cryptography.io. https://cryptography.io/en/latest/fernet/
* GeeksforGeeks. (2017, June 17). Python Tkinter. GeeksforGeeks. https://www.geeksforgeeks.org/python/python-gui-tkinter/
* Gillis, A. (2023). What is Automated Testing and How Does it Work? SearchSoftwareQuality. https://www.techtarget.com/searchsoftwarequality/definition/automated-software-testing
* Redis. (2025, June 11). About Redis Stack - Redis. https://redis.io/about/about-stack/