Muhammad Umer

Budget Safe (Budget Tracker with Data Privacy and Security)

Quick Submit
Quick Submit
National University of Computer and Emerging Sciences, Islamabad

Document Details

Submission ID
trn:oid:::1:3424229325

Submission Date
Nov 25, 2025, 1:07 PM GMT+5

Download Date
Nov 25, 2025, 1:09 PM GMT+5

File Name
Project_Result_Report_25K0983_25K0916_1.pdf

File Size
767.6 KB

10 Pages

674 Words

3,801 Characters

0% detected as AI

The percentage indicates the combined amount of likely AI-generated text as well as likely AI-generated text that was also likely AI-paraphrased.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Detection Groups

0 AI-generated only 0%
Likely AI-generated text from a large-language model.

0 AI-generated text that was AI-paraphrased 0%
Likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.


NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES

University Name: FAST NUCES Karachi Campus

Department: Department of Computer Science

Course: Programming Fundamentals

Project Title: Budget Safe (Budget Tracker with Data Privacy and Security)

Submitted By: Muhammad Umer (25K-0983) & Khizar Khurseed (25K-0916)

Submitted To: Sir Sheeraz Iqbal

Semester: Fall 2025

Date: 22 Nov 2025

Abstract

Budget Safe is a budget tracker system which ensures user-specific budget record management as well as having some basic data privacy and security features

1. Introduction

Most people want to manage their budget effectively and for this purpose, they use either apps or websites. But ignoring the fact that their data can be compromised. That's our problem statement which we have addressed by introducing **"Budget Safe"**

Budget Safe is a functional Budget Tracking project. Users can:

- Register and login securely
- Add, view, edit, delete expenses or income
- Generate monthly reports or financial alerts

2. Objectives

- To develop a secure user account system
- To preserve data by storing it in a file
- To allow users to add, view, and track their transaction records with ensuring basic data privacy

3. System Design

Flow of the program:

Start
→ Home Menu → Login/ Register
→ If Login Success → Load User Dashboard
→ Dashboard Menu → Add/ View/ Edit/ Delete transactions
→ Dashboard Menu → Generate Monthly Report/ Alerts
→ Save data to file to avoid data loss when program stops
Exit

Algorithm

1. Start the program
2. Display home menu (login / signup)
3. If user registers:
    - Prompt user for username
    - Prompt user for password
    - Hash password using hashing algorithm
    - Save these credentials to file
4. If user logs in:
    - Validate credentials
    - If matches → load dashboard

5. Dashboard operations:
    - Add record → encrypt amount → save to file
    - View record → decrypt amount → display
    - Edit/delete → rewrite records in file
    - Monthly report → total sum/ expense by month
    - Alert → compare total expense with given limit

6. Continue until user exits

Input & Output

Input:

- Username
- Password

- Date
- Amount
- Transaction type (income/expense)

Output:

- Login success or failure
- Transaction recorded successfully
- Perform CRUD operations on transaction

4. Implementation

Language used: C

Compiler/IDE: GCC/ Visual Studio (VS) Code

Key Features

- ➢ CLI interface with clean menu navigation
- ➢ Password hashing
- ➢ Basic data encryption
- ➢ CRUD operations on transactions (Add/View/Edit/Delete)
- ➢ Monthly report
- ➢ Expense alert

Code Snippet

1. Password Hash Function

```
unsigned long hashString(const char *str) {
    unsigned long hash = 5381;
    int c;
    while (*str != '\0') {
        c= *str;
        hash = ((hash << 5) + hash) + c;  // (hash * 33) + 112
        str++;
    }
    return hash;
}
```

2. Data structure

```
struct user {
    char username[50];
    unsigned long passwordHash;
};
```

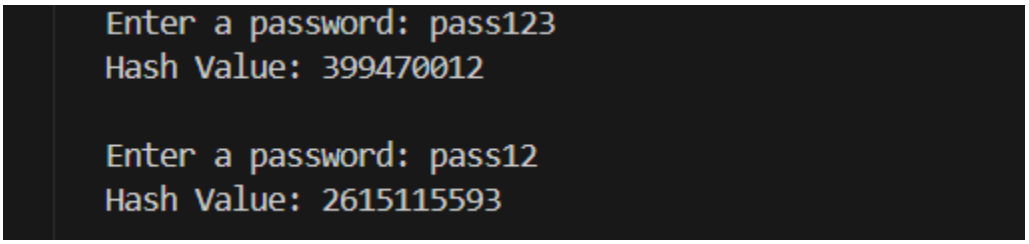
```
struct budgetRecord{
    char date[30]; // in specific format DD-MM-YYYY
    float amount;
    char type[20]; // is it income or expense
};
```

3. Encryption algorithm

```
float encryptAmount(float amount) {
    return (amount *4) + 147.59;
}
```

Sample Output

For hashing function



5. Testing & Results

Case	Input	Expected Output	Actual Output	Status
1	Login (correct credentials)	Login successful	Login successful!	Pass
2	Login (wrong password)	Invalid password	Invalid username or password. Attempts left n out of t (where n = attempts made by user & t=total attempts made)	Pass
3	Add transaction (correct data input)	Record added!	(Save the record in a user specific text file) Record added!	Pass
4	Add transaction (with incorrect data format)	Cause runtime errors	Data got saved but it was useless to use in future	Fail
5	View transaction	Display all records	Display all records but decrypting the amount value	Pass

The system performed consistently for the most of time as it was intended. However, case 4 failed because it will be considered in next versions.

6. Conclusion, Limitations & References

Conclusion

This project showcases how by implementing the fundamental concepts of programming, a CLI based application can be built similar to Budget Safe. Additionally, security and data privacy features can be added to the application in order to provide a safe environment for users to interact with.

Limitations

- Password hashing is basic & obsolete
- Amount encryption is easily reversible
- Basic analytics

Future Enhancements

- Use of stronger encryption for all data as financial data is crucial
- Use of stronger hashing algorithms such as 256
- Add backup system for data
- Add charts/graphs for good analysis

References

- Money Tracker-Expense & Budget App (By Horoscope365)