

**System Requirements**

**Group 2 Semester Project**

**NotPad--**

| Group 2 Semester Project Members | |
| --- | --- |
| Gayendra Baijnath | 1052667 |
| Serina Rebecca Garrett | 1037042 |
| Irshad Mohamed | 1052668 |
| Kiran Doobay | 1053135 |
| Ravindra Persaud | 1052507 |
| Kyahri Whyte | 1053151 |
| Dinar Mohabir | 1052382 |

**INTRODUCTION**

**1.1** **Introduction**

The purpose of this document is to define and describe the technical requirements of this project NotPad--. It outlines the application functionality, the major files , and any limitations and constraints involved in the development process. This documentation aims to be a guide for future developers and maintainers, or anyone who intends to modify the code.

**1.2** **Target Audience**

The primary users for the system are students and instructors of the University of Guyana, including Mrs. Andreasa Morris-Martin, PhD. The constraints for this project includes our deadline for the document which is April 30, 2025. The intention is that the application is a single user application with the ability to record personal messages or notes. No advanced technical knowledge is required beyond interacting with a terminal and understanding menu-based input.

**1.3** **Overview**

The application is a message/note storing app that allows the user to store notes/messages that can be later retrieved via the app. The user can choose to view either all messages or search for specific messages via different search methods. Messages are stored and retrieved from a .dat file with a generated ID and user-given title. The user can encrypt or censor their message before it is stored, then go back and modify the contents of the message at a later time. Encrypted messages can be decrypted for viewing.

**GENERAL DESCRIPTION**

**2.1 Product Functions**

This application is coded in the C programming language and operated entirely through a command-line interface. This application, NotPad-- should enable you to store, view, search, delete, encrypt, censor, and modify messages through a menu-drive interface. The user can store their messages in plain text form, censored or encrypted using a column-array transposition algorithm. Each message is assigned an ID, title and a value that determines if a message is encrypted and can be decrypted. To view their messages, the app can display all messages or search for a specific message(s) using an ID, title and phrase.

**2.2 Objectives**

* Allows easy entry and retrieval of messages.
* Supports encryption as an option to maintain confidentiality.
* Enables censorship of user-determined words regardless of capitalisation differences.
* Facilitates secure modification or deletion of messages.
* Operates without needing a graphical environment or internet access.

**3. FUNCTIONAL REQUIREMENTS**

| **Priority** | **Description** | **Dependency** |
| --- | --- | --- |
| High | The system must allow users to store a new message | Depends on Library  <Stdio.h>  <Stdlib.h>  <String.h>  <ctype.h> |
| High | The system must allow users to view all stored messages | <Stdio.h>  <Stdlib.h>  <String.h> |
| High | The system must search for and display a message by ID/Title | Depends on reading and parsing stored messages |
| High | The system must allow users to delete a note/message by ID | Depends on reading, identifying, and rewriting file content. |
| Medium | Users can censor specific words from a message before storing | Depends on redact() function implementation |
| Medium | Users can encrypt a message using columnar transposition before storing | Depends on encrypt\_columnar function |
| Medium | Users can decrypt a message using a key to view it in plain text | Depends on decrypt\_columnar() function |
| Low | The user can search message by word/phrase within the content | Depends on message storage and indexing implementation |
| Low | The user can modify an existing message | Depends on search, decryption and update mechanism |

**6.1 Security**

The program shall not keep encryption keys in plaintext anywhere in files or in the application. W

The encrypted messages shall not be decipherable without using the correct decryption key.

The program shall sanitize any user input to avoid buffer overflows or injection attacks (especially crucial in C).

**6.2 Reliability**

The program shall gracefully handle invalid inputs, i.e., non-existent message IDs or poorly formatted commands.

The program must fail gracefully, with informative error messages, and without crashing.

No data must be lost or corrupted when reading/writing files (e.g., interrupted writes must not delete already-existing content).

**6.3 Maintainability**

Code will follow modular design, using functions (i.e., read\_string, write\_string) to delineate functionality.

It must have understandable and consistent naming conventions throughout the codebase.

Good inline comments and documentation must explain logic, especially in complex areas like encryption and redaction.

**6.4 Portability**

The software must be written in standard ANSI C so that it will compile under multiple platforms (e.g., Windows, Linux).

There must be no reliance on platform-specific libraries or system calls.

File paths and I/O operations must be generalized for non-OS-dependence.

**6.5 Usability**

The user interface (text-based) must have good menus and instructions for all operations.

Users must be prompted for confirmations for destructive operations like deletion.

The software should include a user manual that provides step-by-step directions on storing, encrypting, and reading messages.

**7. Operational Scenarios**

**7.1 Add a New Message (Unchanged)**

Scenario: A user wants to save an unchanged message.

Steps:

User runs the program and selects "Store a new note/message".

Choose option: "Store message/note unchanged".

Provide title of message: Meeting Notes.

Provide message: Team meeting at 10am on Monday.

System saves the message into the proper file with a system-generated ID.

**7.2 View All Messages**

Scenario: A user wants to view an overview of all saved messages.

Steps:

User selects the "View all notes/messages" menu item.

System displays messages with:

Message ID

Title

First N characters of the message (as developer-specified)

Example Output:

ID: 1 | Title: Meeting Notes | Preview: Team meeting at 10.

ID: 2 | Title: Reminder | Preview: Buy groceries and pick.

**7.3 Store an Encrypted Message**

Scenario: User needs to save a secret message securely.

Steps:

User selects "Store a new note/message".

Choose option: "Encrypt the message/note before storing"

Entered encryption key: keys

Entered title: Secret Plan

Entered message: Attack at dawn

System encrypts the message in columnar transposition and stores it.

Stored Message Example:

ID: 003 | Title: Secret Plan | Encrypted Content: tk-nActwaaa-t-d-

**7.4 View and Decrypt a Message**

Situation: A user wants to view an encrypted message.

Actions:

User chooses "Search for and view a message".

Entered title or ID: 003

System searches and displays encrypted content.

User chooses: "Decrypt this message"

Inputs key: keys

System decrypts message and displays: Attack at dawn

**7.5 Censor a Message Before Storing**

Scenario: A user intends to store a message after censoring out offensive words.

Steps:

Clicks on "Store a new note/message"

Chooses: "Censor the message/note before storing"

Inputs title: Student Feedback

Inputs censor words: bad, boring

Inputs message: The lecture was boring and the slides were bad

System censors and stores:

The lecture was \*\*\*\*\*\* and the slides were \*\*\*

**7.6 Delete a Message**

Scenario: An individual wanting to delete an unwanted note.

Steps:

Selects "Delete a note/message"

Types ID or title: 002

System finds the message and prompts: "Are you sure you want to delete?"

User assents.

Message is forever erased from the file.

**8. Conclusion**

This System Requirements Specification captures the purpose, scope, and detailed requirements of the Message/Note Storing Application intended for CSE1201. It provides an organized foundation for the implementation stage such that all concerned parties, namely team members and reviewers, are able to observe the purpose and operational behavior of the system.

By expressing the operation scenarios, constraints, and functional and non-functional requirements, this document makes clean communication between developers, instructors, and evaluators easier.

While this version meets the present course requirements, future improvements such as a graphical user interface, multi-user support, or cloud storage might further improve usability and scalability.

By signing off on this document, the design and development will proceed based on the agreed specifications.

This project adheres to the group-2-semester-project.zip project standards for software quality requirements and evaluation, particularly focusing on maintainability, portability, and usability.