Rust - Emergency BackUp

Politecnico di Torino

Questo documento descrive il progetto di sviluppo dell'applicazione Emergency Backup, realizzata in linguaggio Rust per il backup dei dati in situazioni critiche.

Progetto a cura di:
Matteo Vincenzo Petrera (s331356)
Gianluca Maida (s334263)
Davide Proglio (324103)

Febbraio 2025

Indice

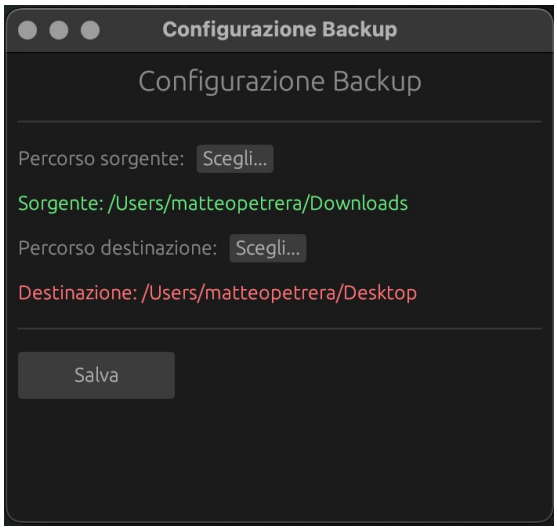
1 Introduzione

Il presente progetto mira a sviluppare un'applicazione desktop in linguaggio RUST, progettata per consentire il backup dei dati anche in situazioni critiche, come l'inutilizzabilità dello schermo. L'applicazione sarà installata durante la fase di bootstrap del sistema operativo, garantendo la sua disponibilità immediata all'avvio del PC e funzionando in background con un impatto minimo sul consumo della CPU. Il software sarà pienamente compatibile con i principali sistemi operativi: Windows, macOS e Ubuntu, offrendo una soluzione versatile e adattabile a diverse esigenze e ambienti di lavoro. L'applicazione permetterà agli utenti di avviare il backup tramite comandi specifici eseguiti con il mouse, eliminando la necessità di un'interfaccia grafica tradizionale. Questo approccio semplifica l'uso anche per utenti con poca esperienza tecnica. L'utente potrà personalizzare la scelta della sorgente e della destinazione del backup tramite un file di configurazione. L'applicazione sarà ottimizzata per garantire un basso consumo di CPU, rimanendo attiva in background senza compromettere le prestazioni complessive del sistema. Al termine di ogni operazione di backup, verrà generato un file di log nella stessa destinazione di backup. Questo file conterrà un riepilogo della procedura di backup e statistiche utili per l'analisi delle prestazioni e per l'ottimizzazione continua del sistema.

2 Funzionamento del Progetto

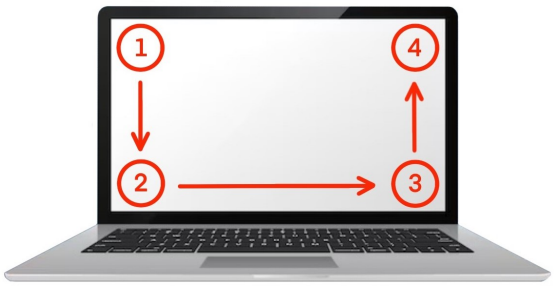
In questa sezione verrà illustrato come utilizzare correttamente l'applicazione per eseguire con sucesso la procedura di backup.

- **Step 1**: Se è la prima volta che si esegue l'applicazione sarà presentata all'utente una semplice finestra di configurazione che gli permetterà di personalizzare sorgente e destinazione della procedura di backup. Dal secondo avvio in poi questa finestra di setup non sarà più presentata, è tuttavia possibile per l'utente cambiare le informazioni iniziali aprendo l'eseguibile "setup", accedendo nuovamente alla finestra di configurazione. Di seguito è riportata la finestra di setup:



Configurazione backup

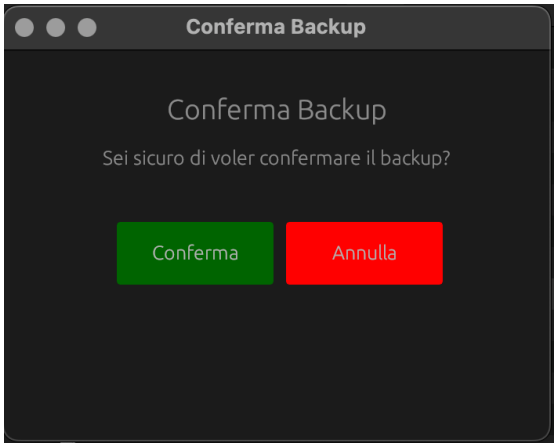
- **Step 2**: In seguito alla configurazione, o ad avvii successivi al primo, l'attivazione della procedura di backup comincia con il riconoscimento di una sequenza di movimenti da svolgere attraverso il mouse. In particolare per avviare la procedura di backup ci si deve spostare ordinatamente con il mouse dall'angolo superiore sinistro dello schermo a quello in inferiore sinistro per proseguire con quello inferiore destro e infine con quello superiore destro. Di seguito è riportata un'immagine esplicativa della sequenza da eseguire:



Sequenza attivazione backup

- **Step 3**: Dopodichè sarà presentata all'utente una schermata di conferma di backup dalla quale sarà possibile avviare effettivamente il backup o annullarlo. Nel caso lo schermo fosse inutilizzabile (schermo nero), sarà possibile avviare il backup tracciando un meno con il mouse sullo schermo

(cliccando e tenendo premuto sul mouse si disegna una linea orizzontale), al rilascio del mouse verrà emesso un segnale acustico per indicare il corretto avvio del backup.



Conferma backup - finestra di conferma

Conferma backup - tracciamento "-"

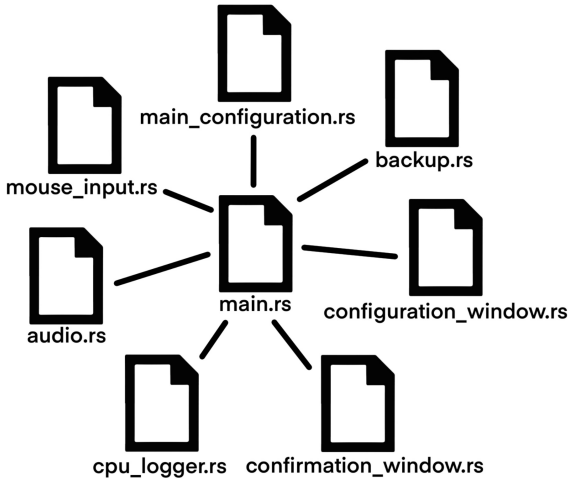
- **Step 4**: Al termine del backup verrà emesso un secondo segnale acustico per indicare la corretta conclusione della procedura di salvataggio. Nella destinazione definita durante la procedura di setup saranno disponibili una copia dei file presenti nella sorgente di backup e un file di log contenente la dimensione del backup e il consumo di CPU.

3 Architettura e Implementazione del software

3.1 Sistema Architetturale

L'applicazione è organizzata in moduli, ciascuno dedicato a una funzionalità specifica. Questo approccio architetturale migliora la leggibilità e la manutenibilità del codice, facilitando inoltre l'integrazione di nuove funzionalità nel pro-

getto. Di seguito viene riportata un'immagine esplicativa dell'organizzazione strutturale del software:



Schema architetturale

3.2 Liguaggio

La scelta del linguaggio **Rust** per questo progetto è motivata dalla sua capacità di combinare sicurezza e prestazioni elevate. Grazie al suo sistema di gestione della memoria, Rust evita errori comuni come i null pointer e le race conditions, assicurando applicazioni più sicure. Inoltre, supporta la concorrenza in modo sicuro, ideale per operazioni multi-thread come il monitoraggio degli eventi del mouse. La velocità di esecuzione e il controllo delle risorse, insieme a un ricco ecosistema di librerie, rendono Rust la scelta perfetta per sviluppare applicazioni robuste e performanti.

3.3 Librerie Principali

Di seguito è riportato un breve riepilogo delle librerie impiegate all'interno del progetto:

- **sysinfo**: utilizzata per monitorare l'uso della CPU nei processi, in particolare per registrare il consumo delle risorse nel modulo 'cpu_logger.rs'.

- **chrono**: serve per ottenere e formattare timestamp, utili nei log delle attività come il monitoraggio CPU e i backup.
- **lazy_static**: facilita l'inizializzazione di variabili globali, ad esempio per gestire configurazioni o parametri statici.
- **rdev**: usata per ascoltare eventi relativi al mouse (movimento e clic). Fondamentale nel tracciare i movimenti del mouse e nell'attivare le azioni come il backup se viene disegnato un segno meno.
- **scrap**: impiegata per ottenere le dimensioni dello schermo nel modulo "mouse_input.rs" e per il tracciamento e la gestione della selezione dei punti (1, 2, 3, 4).
- **device_query**: adoperata per ottenere lo stato del mouse, inclusa la posizione e il suo movimento, usata per tracciare la posizione del cursore in tempo reale.
- **rodio**: usata per la riproduzione di file audio nel modulo "audio.rs", per notifiche sonore all'inizio e dopo il backup.
- **egui ed eframe**: impiegate per creare interfacce grafiche, come le finestre di configurazione e conferma del backup ("configuration_window.rs", "confirmation_window.rs").
- **rfd**: semplifica la selezione di cartelle o file tramite finestre di dialogo native, usata per configurare i percorsi di backup.
- **auto-launch**: gestisce l'avvio automatico dell'app all'accensione del sistema, configurato e disattivato in "uninstall.rs".

3.4 Moduli Principali

3.4.1 main.rs e main_configuration.rs

Gestiscono l'automazione dell'avvio dell'applicazione, la configurazione iniziale dei percorsi di input/output e il monitoraggio delle risorse di sistema. Se i percorsi di configurazione non sono presenti, il programma avvia un processo di configurazione interattivo. Una volta configurato correttamente, il programma prosegue con l'esecuzione del modulo principale, gestendo l'input del mouse e altre operazioni a seconda delle necessità.

3.4.2 mouse_input.rs

Il file mouse_input.rs implementa la gestione degli input del mouse, con funzionalità avanzate per il monitoraggio del movimento del cursore e il riconoscimento di azioni specifiche. Include anche il controllo della posizione del mouse, l'interazione con l'utente, e l'avvio di una sequenza di operazioni di backup in caso di azioni corrette. Funzioni principali:

- Monitoraggio della sequenza di spostamento del mouse negli angoli dello schermo (**handle_event**): la funzione handle_event è il cuore del sistema di rilevamento degli eventi del mouse. Monitora il movimento del mouse per riconoscere la sequenza di punti negli angoli dello schermo. Una volta che questa sequenza è completata, invia un segnale tramite un canale per aprire una finestra di conferma (**confirmation_window.rs**). La funzione **initialize_confirmation_channel** crea un canale di comunicazione tra il rilevamento degli angoli e la finestra di conferma.

- Monitoraggio del "segno meno" (**track_minus_sign**): La funzione si occupa di monitorare e rilevare se il movimento del mouse forma un segno meno, ovvero una linea orizzontale con una tolleranza verticale molto piccola. Quando questo segno viene riconosciuto, viene emesso un segnale audio e viene avviato il processo di backup dei files.

3.4.3 backup.rs

Il modulo backup.rs automatizza la copia sicura di file specifici da una sorgente a una destinazione, supportando la gestione di directory annidate e fornendo un resoconto dettagliato dell'operazione eseguita. La funzione principale del modulo è chiamata **backup**, questa gestisce il processo di copia dei file da una cartella di origine a una di destinazione. La funzione riceve tre parametri: il percorso della cartella sorgente, quello della cartella di destinazione e un elenco di tipi di file da includere nel backup. Il processo inizia registrando l'orario di avvio per calcolare la durata totale del backup. Viene poi creata la directory di destinazione, se non esiste già. La funzione interna **recursive_backup** viene utilizzata per esplorare ricorsivamente tutte le sottocartelle e i file all'interno della cartella sorgente. Durante questa esplorazione, ogni file viene verificato per controllare se la sua estensione corrisponde a uno dei tipi di file specificati. Se la verifica ha esito positivo, il file viene copiato nella destinazione e viene registrata la sua dimensione per calcolare il totale dei dati trasferiti. Al termine del processo di backup, viene generato un file di log chiamato **backup_log.txt** nella cartella di destinazione. Questo file contiene informazioni sul tempo impiegato per completare il backup, la quantità totale di dati copiati e un registro dettagliato di tutti i file trasferiti, con indicazione del loro percorso e della dimensione.

3.4.4 cpu_logger.rs

Il modulo gestisce la registrazione periodica dell'utilizzo della CPU del processo corrente. La funzione principale **log_cpu_usage** inizializza un oggetto System per monitorare le risorse di sistema e apre un file di log chiamato **cpu_usage_log.txt**, dove verranno memorizzati i dati raccolti. All'interno di un ciclo infinito, la funzione aggiorna le informazioni sul processo corrente, identificato tramite il suo PID, e ne rileva l'utilizzo della CPU. Per ogni rilevazione, viene registrato un timestamp insieme alla percentuale di utilizzo della CPU e

questi dati vengono sia scritti nel file di log che stampati a schermo. Il monitoraggio viene effettuato ogni 120 secondi (2 minuti) per garantire aggiornamenti regolari senza sovraccaricare il sistema.

3.4.5 confirmation_window.rs

Il modulo gestisce un'interfaccia grafica per confermare l'esecuzione di un backup. La funzione principale **run** avvia una finestra tramite la libreria "eframe" con dimensioni predefinite.. All'interno di questa finestra viene chiesto all'utente se desidera procedere con il backup, offrendo due pulsanti: "Conferma" e "Annulla". Se l'utente clicca su "Conferma", viene avviata la funzione backup, utilizzando i percorsi di origine e destinazione definiti nelle variabili globali e un set di tipi di file specifici (txt, jpg, png). Se il backup va a buon fine o si verifica un errore, viene stampato un messaggio corrispondente e l'applicazione si chiude. Se l'utente sceglie "Annulla", il processo termina immediatamente senza eseguire alcuna operazione.

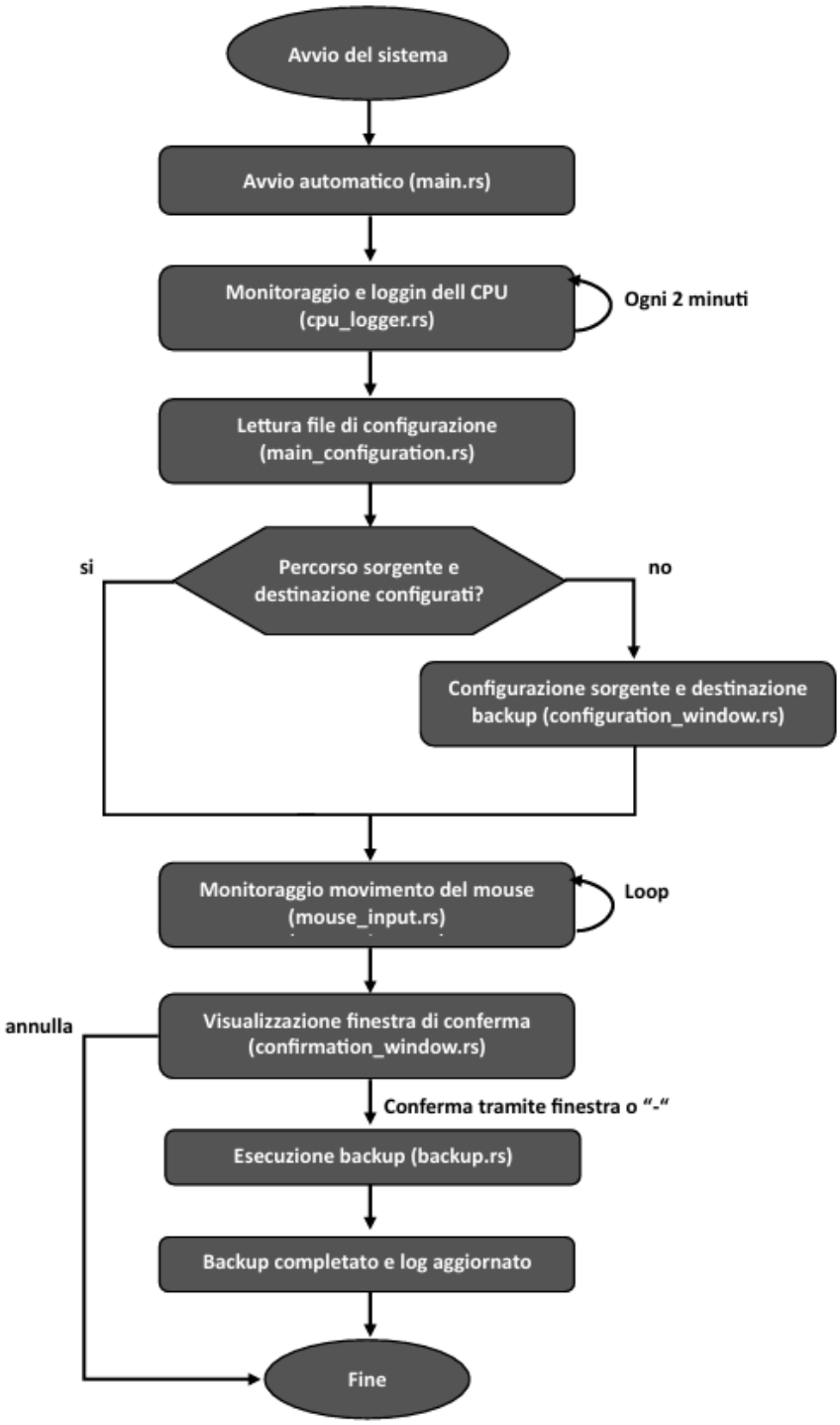
3.4.6 configuration_window.rs

Il modulo gestisce una finestra grafica per la configurazione dei percorsi di origine e destinazione del backup. La funzione principale **run_configuration_window** avvia l'interfaccia usando la libreria "eframe", permettendo all'utente di selezionare le cartelle tramite un semplice pulsante che apre un selettore di file. L'applicazione memorizza temporaneamente i percorsi scelti. Una volta selezionati entrambi i percorsi, l'utente può cliccare sul pulsante "Salva", che registra i percorsi in un file CSV chiamato **configuration.csv**, situato all'interno della cartella configuration_csv del progetto. Se l'operazione va a buon fine, la finestra si chiude automaticamente. In caso contrario, viene mostrato un messaggio di errore se non sono stati selezionati entrambi i percorsi.

3.4.7 audio.rs

Il modulo gestisce la riproduzione di file audio utilizzando la libreria "rodio". Contiene due funzioni principali, **play_sound** e **play_sound2**, che riproducono rispettivamente i file audio "audio.wav" che indica l'inizio del processo di backup e "audio2.wav" che ne indica la corretta terminazione. Entrambe le funzioni inizializzano uno stream audio e un sink per la gestione del flusso audio. Dopo aver aperto il file audio corrispondente e decodificato il contenuto, il file viene riprodotto. L'esecuzione della funzione resta in attesa fino al termine della riproduzione del suono.

3.5 Flusso di esecuzione



Flusso di esecuzione

4 Ottimizzazione

Nel progetto sono state adottate diverse ottimizzazioni a livello di codice e di architettura, con l'obiettivo di migliorare le prestazioni, la sicurezza e l'affidabilità dell'applicazione. Di seguito vengono esplorate le principali ottimizzazioni implementate.

4.1 Gestione Efficiente delle Risorse e Concorrenza

Rust è stato scelto come linguaggio per il progetto proprio per le sue eccellenti capacità di gestione delle risorse e della concorrenza. Il sistema di ownership e borrowing di Rust assicura che le risorse siano correttamente gestite durante l'esecuzione, senza necessità di garbage collector. Questo approccio permette di evitare memory leaks e race conditions, che sono particolarmente critiche in applicazioni che interagiscono con hardware o gestiscono grandi quantità di dati, come nel caso di un'applicazione di backup.
In moduli come mouse_input.rs, la gestione concorrente degli eventi del mouse è stata implementata utilizzando thread separati, sfruttando il modello di concorrenza di Rust tramite Mutex e Arc. La libreria lazy_static è stata utilizzata per inizializzare variabili globali che devono essere accessibili da più moduli senza il rischio di condizioni di gara. Le variabili come le dimensioni dello schermo, coordinate di mouse, e stato di disegno vengono gestite in modo sicuro tramite Mutex e Arc. Questo approccio minimizza il rischio di errori dovuti a accessi concorrenti, ottimizzando la gestione della memoria e migliorando la sincronizzazione tra i vari componenti del progetto.

4.2 Ottimizzazione delle Interazioni con il Disco e File I/O

Nel modulo di backup, backup.rs, l'operazione di scrittura e lettura di file è stata ottimizzata per minimizzare i tempi di attesa. L'uso di buffer durante la lettura dei file e l'uso di modalità di apertura dei file come OpenOptions::append() garantiscono che le operazioni di I/O siano veloci e sicure. Inoltre, i percorsi di origine e destinazione per i file da copiare vengono gestiti in modo efficiente, riducendo il numero di operazioni di sistema necessarie. Nel caso della scrittura su file CSV nel modulo configuration_window.rs, è stata utilizzata una logica di scrittura in append, evitando di sovrascrivere file già esistenti.

4.3 Ottimizzazione della Gestione degli Eventi del Mouse

Un'altra ottimizzazione cruciale è quella relativa alla gestione degli eventi del mouse. Nel modulo mouse_input.rs, l'algoritmo per il tracciamento del segno meno sfrutta una logica di filtraggio avanzata, in modo da ignorare movimenti non significativi e focalizzarsi solo su eventi rilevanti. La funzione track_minus_sign() analizza accuratamente gli eventi del mouse, per evitare il tracciamento di segnali falsi o di interazioni non pertinenti. Inoltre, l'utilizzo del multi-threading e della sincronizzazione tramite Mutex permette di monitorare in tempo reale la

posizione del mouse senza compromettere le performance. Inoltre, una tolleranza per la precisione dei movimenti (come nel tracciamento degli angoli dello schermo) riduce il numero di eventi da elaborare, migliorando così l'efficienza del sistema.

4.4 Ottimizzazione del Flusso di Lavoro dell'Interfaccia Grafica

Per migliorare l'esperienza utente, l'interfaccia grafica è stata ottimizzata utilizzando eframe e egui. La creazione delle finestre e l'interazione con gli utenti sono stati progettati per ridurre il carico computazionale e migliorare la reattività. Per esempio, l'uso di layout dinamici e la gestione efficiente delle risorse UI contribuiscono a garantire un'interfaccia leggera e performante. Inoltre, il codice di gestione delle finestre di configurazione, come in configuration_window.rs, è stato ottimizzato per mantenere l'applicazione reattiva, anche durante operazioni di I/O.

4.5 Ottimizzazione per le Piattaforme Specifiche

Le ottimizzazioni specifiche per piattaforma, come evidenziato nel modulo uninstall.rs, permettono di disabilitare correttamente l'applicazione al termine, senza lasciare processi in background. L'uso di comandi specifici per ogni sistema operativo, come taskkill su Windows e pkill su Linux e macOS, contribuisce a garantire una disinstallazione pulita. Questa attenzione alla gestione dei processi è fondamentale per prevenire perdite di risorse e migliorare la stabilità del sistema durante l'esecuzione.

Nel complesso, le ottimizzazioni implementate nel progetto mirano a garantire alte prestazioni, affidabilità e sicurezza, sfruttando le caratteristiche avanzate di Rust come la gestione della memoria, il supporto per la concorrenza e il controllo rigoroso del sistema. Grazie all'uso di librerie efficienti e a un'attenta progettazione dell'architettura, il progetto offre un'esperienza utente fluida e reattiva, pur mantenendo elevati standard di sicurezza e performance.

5 Conclusioni

Il progetto ha raggiunto con successo gli obiettivi iniziali, sviluppando un'applicazione in linguaggio Rust per effettuare un backup dei dati in modo efficace, anche in situazioni in cui lo schermo del PC non è agibile. La funzionalità principale, ovvero l'attivazione del backup tramite un comando tracciato con il mouse, è stata implementata utilizzando la libreria rdev, che ha permesso di rilevare e gestire in modo preciso gli eventi del mouse. L'applicazione consente inoltre

di monitorare il consumo di CPU tramite la libreria sysinfo, registrando periodicamente il dato su un file di log, come richiesto, e riducendo al minimo l'uso della CPU durante l'esecuzione del programma, un aspetto essenziale per garantire l'efficienza del sistema anche in background. Un altro requisito fondamentale, ovvero la generazione di un file di log contenente informazioni sulla dimensione complessiva dei file salvati e sul tempo di CPU impiegato per il completamento dell'operazione di backup, è stato implementato con successo. L'installazione e l'avvio automatico dell'applicazione al bootstrap del sistema sono stati gestiti grazie alla libreria auto-launch, che assicura che l'app sia sempre attiva in background, pronta a eseguire il backup non appena il comando venga dato. A partire da questi successi, ci sono ancora ampie opportunità per migliorare l'applicazione. In particolare, sarebbe utile implementare funzionalità di backup incrementale, che permetterebbero di copiare solo i file modificati o aggiunti, riducendo così il tempo e lo spazio necessari per il backup. Potrebbe anche essere utile ottimizzare ulteriormente il consumo di CPU, esplorando tecniche di priorizzazione dei processi per gestire meglio le risorse del sistema durante l'esecuzione del backup. Un miglioramento dell'interfaccia utente, con l'inclusione di notifiche in tempo reale sullo stato del backup o sui consumi, contribuirebbe a rendere l'applicazione più user-friendly e informativa. Con queste migliorie, l'applicazione potrebbe evolversi, offrendo una soluzione di backup ancora più completa, versatile e user-oriented. In sintesi, il progetto ha fornito una solida base per un'applicazione di backup automatizzata e sicura, capace di rispondere a un'esigenza specifica di operabilità in situazioni critiche, ma con un ampio margine di crescita per espandere e perfezionare le sue funzionalità.