POLITECNICO DI TORINO

Programmazione di sistema

Backup Emergency

Leonardo MORACCI

Paolo MUCCILLI

Miriam IVALDI

Massimo PORCHEDDU

Indice

Capitolo 1

Introduzione

L'obiettivo principale di questo progetto è lo sviluppo di un applicativo che consenta agli utenti desktop di poter effettuare il backup dei dati presenti nel PC anche nei casi in cui lo schermo del computer risulti inaccessibile. Questo tipo di contesto può verificarsi più spesso del previsto in seguito a malfunzionamenti generali del dispositivo, problemi hardware o situazioni in cui l'accesso allo schermo è compromesso da altri fattori.

Inoltre, un altro aspetto fondamentale della soluzione progettata è quello di renderla fruibile anche da utenti con competenze tecniche limitate, al fine di rendere il backup un'operazione accessibile a tutti. L'applicazione mette a disposizione opzioni configurabili che consentono la scelta della sorgente di backup nonché la possibilità di scegliere il tipo di dati da salvare. È importante sottolineare che, le modalità tradizionali per poter effettuare un backup, tendenzialmente sfruttano l'accesso a interfacce visive complesse che quindi si rendono inutilizzabili nel caso in cui lo schermo non sia agibile. La particolarità di questa applicazione è proprio quella di consentire l'esecuzione dell'operazione di backup con semplici comandi del mouse, al fine di garantire un livello di accessibilità superiore.

L'applicazione proposta è inoltre caratterizzata dalla sua capacità di operare in background con un consumo minimo di risorse di sistema, assicurando quindi un impatto trascurabile sulle prestazioni del PC stesso. Un'apposita funzione di logging si occupa di monitorare quanto appena detto.

Per concludere il software sviluppato consente di essere eseguito nei seguenti sistemi operativi: Windows, MacOs, Ubuntu.

Capitolo 2

Requisiti e Specifiche del Progetto

2.1 Requisiti funzionali

- Attivazione del backup tramite comando del mouse: disegno di un rettangolo che tocca tutti gli angoli dello schermo partendo da quello in alto a sinistra procedendo in senso orario.

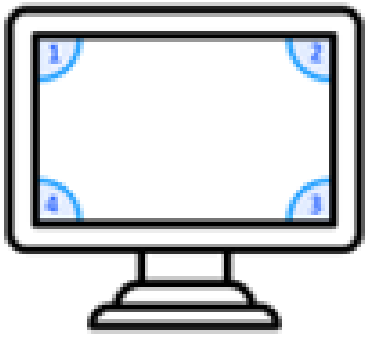

Figura 2.1: first step

- Annullamento del backup: tornando nell'angolo in alto a sinistra.

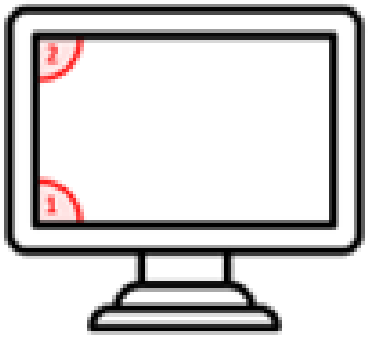

Figura 2.2: second step

- Conferma del backup: tramite un secondo comando del mouse tracciando una partendo dall'angolo in basso a sinistra fino all'angolo in basso a destra.

Figura 2.3: third step

- Visualizzazione di una finestra di conferma.

2.2 Requisiti non funzionali

- L'applicazione deve essere installata automaticamente durante la fase di bootstrap del PC.
- Esecuzione in background con consumo minimo di CPU.
- Il consumo di CPU deve essere monitorato e salvato su un file di log ogni 2 minuti.
- Creazione di un file di log con la dimensione totale del backup

Capitolo 3

Architettura del software

La struttura dell'applicazione è modulare, ogni modulo è responsabile di una funzionalità specifica. Questa architettura favorisce la manutenibilità e la leggibilità del codice e anche la possibilità di ampliare il progetto con nuove funzionalità.

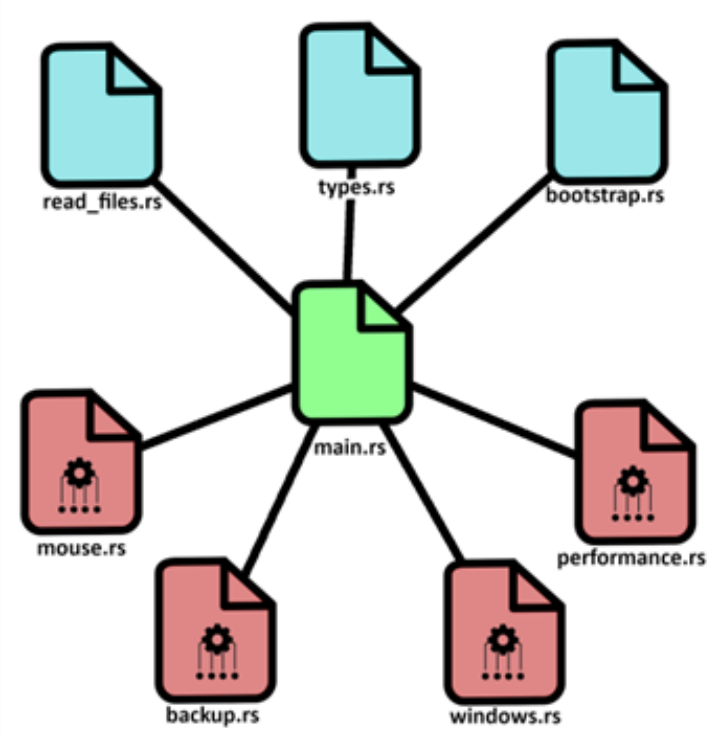

Figura 3.1: Architettura del software

3.1 Moduli principali

1. **main.rs**

   Questo file funge da entry point dell'applicazione. È il componente in cui vengono inizializzati i vari moduli, avviati i thread e configurati i parametri per il corretto funzionamento del sistema.

2. **read_files.rs**

Questo file fornisce le funzionalità necessarie per la gestione del file di configurazione del programma: definisce i parametri relativi al processo di backup: directory sorgente, directory destinazione e tipi di file da includere nel backup.

3. **types.rs**

   Contiene l'enum BackupState, essenziale per la gestione e il corretto funzionamento degli stati dell'applicazione.

4. **bootstrap.rs**

   Questo file gestisce la configurazione necessaria per impostare l'avvio automatico dell'applicazione durante la fase di bootstrap del sistema operativo. Questa funzionalità assicura che il software sia sempre attivo in background al momento dell'accensione del dispositivo. Il modulo è stato progettato per supportare il bootstrap su tutti i principali sistemi operativi, inclusi Windows, macOS e Ubuntu.

5. **mouse.rs**

   Questo file implementa la funzione mouse_movements, che si occupa di gestire il monitoraggio continuo delle coordinate del mouse per rilevare pattern di movimento, come il tracciamento di un rettangolo lungo i bordi dello schermo. Una volta che il pattern viene riconosciuto la funzione modifica lo stato dell'enum BackUpState (Idle, Confirming, Confirmed, BackingUp). Questo processo deve essere pressocché sempre in ascolto, quindi, è necessario che venga eseguito dentro un thread separato.

6. **backup.rs**

   Il modulo backup.rs rappresenta il fulcro del software, occupandosi dell'effettiva esecuzione del backup dei dati. L'operazione viene svolta all'interno di un thread separato, in modo che il programma rimanga reattivo durante tutte le fasi.

7. **window.rs**

   Mostra una finestra di conferma nel caso in cui l'utente disponga di uno schermo funzionante. Questa finestra è utile per avvisare l'utente quando il backup viene avviato accidentalmente, offrendo la possibilità di annullare l'operazione. La finestra può essere chiusa tramite i tasti dedicati o utilizzando gesture sullo schermo.

8. **performance.rs**

   Al suo interno è presente la logica che consente di monitorare le performance del programma e di scrivere un file di log che mantenga traccia delle stesse, sin tanto che il software resta in esecuzione.

Capitolo 4

Implementazione

4.1 Tecnologie utilizzate

Il linguaggio utilizzato per lo sviluppo è Rust, scelto per la sua capacità di gestire in modo efficiente la concorrenza e coordinare thread indipendenti, caratteristiche fondamentali per un'applicazione che opera in background. Grazie alla gestione nativa delle risorse permette inoltre di garantire un basso impatto sulle prestazioni del sistema. Queste caratteristiche rendono Rust particolarmente adatto per sviluppare applicazioni che richiedono un funzionamento continuo e affidabile su più piattaforme.

4.2 Librerie principali

Sono state utilizzate diverse librerie per implementare tutte le funzionalità dell'applicazione:

- **winit:** è stata utilizzata per la gestione della finestra di conferma.
- **pixels:** permette di disegnare pixel per pixel il contenuto della finestra di conferma, consentendo una personalizzazione completa.
- **sysinfo:** utilizzata per monitorare l'utilizzo della CPU
- **auto_launch:** implementa le funzionalità necessarie per l'avvio automatico del bootstrap del SO.
- **fs_extra:** utilizzata per la copia ricorsiva di directory e file.

4.3 Descrizione dei moduli

4.3.1 main.rs

responsabile della gestione del ciclo di vita del programma e della coordinazione di tutti i moduli:

- All'avvio, il programma verifica la presenza di un file di configurazione. Se l'utente non specifica un file tramite linea di comando, ne viene creato uno predefinito: config.toml, nella directory di configurazione dell'applicazione. Questo file contiene diverse informazioni come:
  - la directory sorgente da cui effettuare il backup
  - percorso destinazione o stringa di errore
  - tipi di file di cui effettuare il backup
  - percorso del file di log per il monitoraggio del consumo di CPU.

```
[backup]
destination_directory = "Error to find USB drive"
file_types = ["*"]
source_directory = "C:\\Users\\morac\\Desktop\\curriculum"

[cpu_logging]
log_path = "C:\\Users\\morac\\AppData\\Roaming\\Backup Emergency\\log_CPU.txt"
```

Figura 4.1: file di configurazione

- Il modulo bootstrap.rs è chiamato per configurare l'applicazione in modo che venga avviata automaticamente durante il bootstrap del PC.
- Avvia il modulo mouse.rs che ascolta e interpreta i movimenti del mouse per attivare il backup
- Avvia il modulo backup.rs che esegue il processo di copia vera a propria dei dati
- Avvia il modulo performance.rs che registra periodicamente il consumo di CPU e un file di log.
- Il modulo window.rs viene avviato nel therad principale per visualizzare la finestra di conferma.
- Attende che tutti i thread completino le loro operazioni, garantendo la terminazione ordinata del programma.

La sincronizzazione tra questi thread avviene tramite un Arc condiviso che utilizza un mutex e una Condvar.

4.3.2 mouse.rs

questo file implementa la funzione mouse_movements, che si occupa di gestire il monitoraggio continuo delle coordinate del mouse per rilevare dei pattern di movimento, nello specifico il tracciamento di un rettangolo lungo i bordi dello schermo. Una volta che il pattern viene riconosciuto la funzione modifica lo stato dell'enum BackUpState (Idle, Confirming, Confirmed). Quando il pattern è completato il

programma entra nello stato di Confirming, da qui l'utente può scegliere se confermare o annullare il backup attraverso ulteriori movimenti del mouse. Il processo è progettato per rimanere costantemente in ascolto, garantendo un funzionamento continuo. Per minimizzare l'impatto sulle risorse di sistema, utilizza un ciclo infinito con una pausa di 100 millisecondi (thread::sleep()). Questa pausa regolare riduce il carico sulla CPU, ottimizzando l'efficienza senza compromettere la reattività del programma.

4.3.3 backup.rs

rappresenta il fulcro del software, occupandosi del backup vero e proprio dei dati. L'operazione viene svolta all'interno di un thread separato, in modo che il programma rimanga reattivo durante tutte le fasi. Questo file gestisce diversi aspetti dell'operazione di backup:

- **Identificazione della destinazione del backup:** viene rilevata automaticamente la posizione di dischi esterni collegati al sistema (vengono supportati i principali sistemi operativi come Windows, macOS, Ubuntu), e in caso di errore nell'individuazione del dispositivo, viene riprodotto un segnale acustico per informare l'utente della mancata presenza di un dispositivo esterno.
- **Copia dei dati:** all'interno del modulo è presente la funzione copy_dir_recursive che copia ricorsivamente i file e le cartelle presenti nella directory sorgente, in quella di destinazione. La funzione should_copy_file consente il filtraggio dei file in base alle estensioni configurate (se configurato con *, copia tutto).
- **Logging e feedback:** al termine del processo di backup viene creato un file di log nella directory destinazione, contenente informazioni come il tempo impiegato e la dimensione complessiva dei dati copiati.
- **Ottimizzazione per sistemi operativi diversi:** vengono utilizzati comandi specifici per ciascun sistema operativo per individuare i dischi esterni, wmic su windows, diskutil su macOS e lsblk su Ubuntu.
- **Gestione dello stato:** il modulo interagisce con il sistema di sincronizzazione degli stati tramite un Mutex e una Condvar. In questa maniera controlla lo stato del programma (Idle, BackingUp ecc.) e notifica le altre parti dell'applicazione al termine del processo.

Inoltre, questo modulo integra una funzionalità di logging dei dettagli della cartella e sulla dimensione totale del backup, utilizzando la funzione calculate_directory_size().

4.3.4 window.rs

implementa la visualizzazione di una finestra di conferma. La finestra è sincronizzata con lo stato del programma tramite un mutex e una Condvar associati all'enum BackupState. Quando lo stato cambia a Confirming la finestra diventa visibile e quando lo stato cambia a Idle o Confirmed, la finestra ritorna ad essere nascosta.

La finestra può essere chiusa tramite i comandi di sistema (clic su “chiudi”) oppure tramite gesture dell’utente attraverso il mouse: movimento verso l’angolo superiore sinistro per annullare o verso l’angolo inferiore destro per confermare.

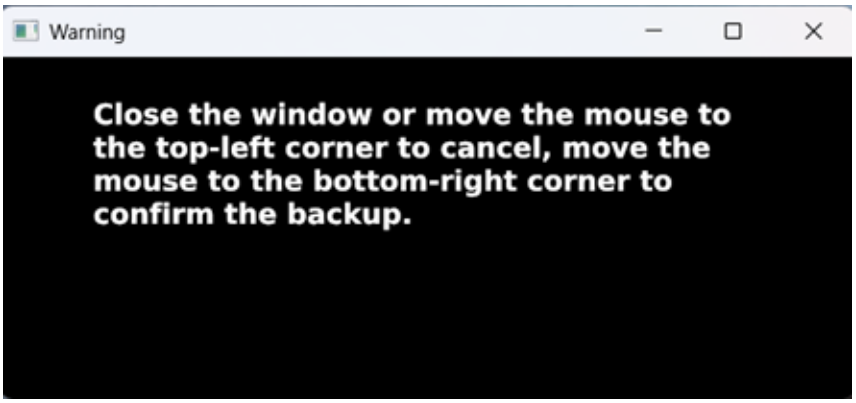

Figura 4.2: finestra di avviso

4.3.5 performance.rs

Viene implementata una logica che periodicamente rileva il consumo di risorse e registra i dati in un file di log. La funzione get_cpu_usage utilizza la libreria sysinfo per recuperare il consumo di CPU del processo corrente. Aggiorna le informazioni sui processi del sistema e individua il consumo di CPU associato al PID del programma. La funzione append_to_log si occupa di scrivere i dati sul consumo di CPU in un file di log: ogni voce del log contiene data e ora precise e la percentuale di utilizzo di CPU. La funzione log_cpu_usage_periodically utilizza un ciclo infinito per monitorare e registrare l’utilizzo della CPU a intervalli configurabili in secondi.

4.3.6 read_file.rs

Questo modulo fornisce le funzionalità necessarie per la lettura del file di configurazione del programma. La struttura BackupConfig definisce i parametri relativi al processo di backup come directory sorgente, directory destinazione e tipi di file da includere nel backup. Fornisce inoltre un’implementazione del tratto Default per inizializzare i valori predefiniti, utile per quando il file di configurazione non è presente. La struttura CpuLoggingConfig specifica il percorso del file in cui viene salvato il log del consumo di CPU. La struttura Config combina i parametri di configurazione del backup e del logging. La funzione read_config legge il file di configurazione .toml e lo deserializza nella struttura Config.

4.3.7 bootstrap.rs

l’avvio automatico dell’applicazione viene fatto utilizzando la libreria auto_launch. L’applicazione, quindi, viene registrata nei processi di avvio automatico del sistema operativo, in modo che venga eseguita ogni volta che il PC si accende. Il percorso dell’eseguibile viene recuperato utilizzando env::current_exe, il nome dell’applicazione

e il percorso di avvio vengono configurati con AutoLaunchBuilder e l'avvio automatico viene abilitato tramite auto_launch.enable.

4.3.8 types.rs

definisce gli stati condivisi tra i moduli. Contiene l'enum BackupState per rappresentare i diversi stati dell'applicazione.

- Idle → l'applicazione è inattiva.
- Confirming → l'utente sta confermando o annullando l'operazione
- Confirmed → l'utente ha confermato l'operazione
- BackingUp → il backup è in corso

4.4 Flusso di esecuzione

L'applicazione segue un flusso ben definito per gestire lo stato del sistema e l'interazione dell'utente, come illustrato nei diagrammi forniti. La variabile state è racchiusa nel Mutex e condivisa tra i threads, la variabile count è locale al thread mouse_movements.

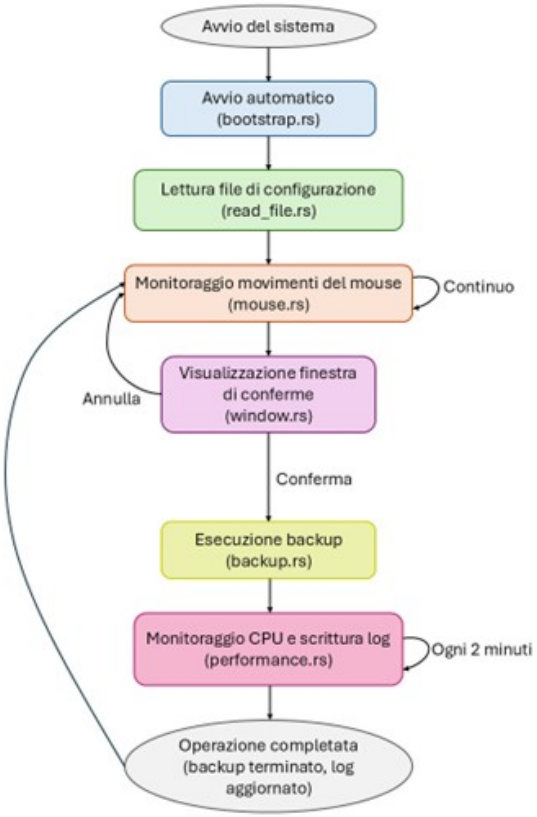

Figura 4.3: diagramma di flusso esecuzione

Di seguito viene descritto il comportamento in dettaglio:

1. **Stato iniziale (state: Idle count: 0)**

   L'applicazione inizia in uno stato di inattività, monitorando le interazioni dell'utente tramite il mouse. Questo stato è chiamato Idle e corrisponde al punto di partenza del ciclo. Il thread dedicato al mouse è attivo e in ascolto degli input per rilevare eventuali azioni.

2. **Gestione degli Input del mouse**

   L'utente interagisce con specifiche aree dello schermo (ad esempio, angoli o posizioni predefinite), che causano la transizione tra gli stati.

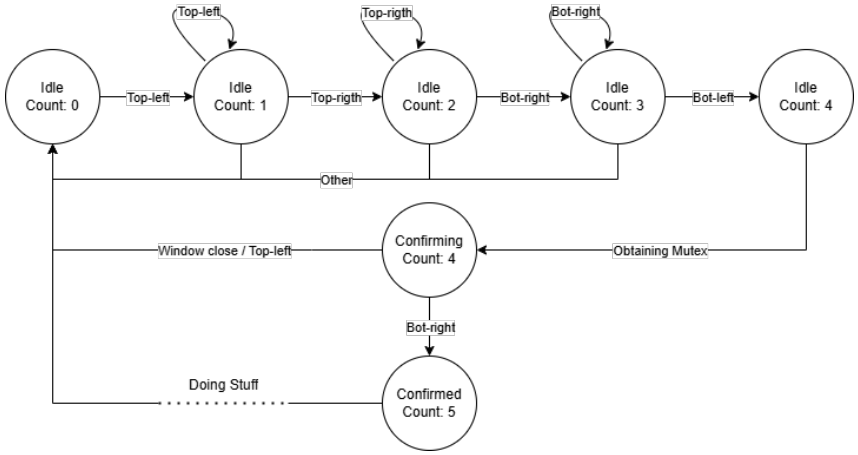

Figura 4.4: diagramma di flusso degli eventi del mouse

   Ogni interazione dell'utente incrementa il contatore count, seguendo questo percorso:

   (a) **Top-left:** Transizione dallo stato state: Idle count: 0 a state: Idle count: 1

   (b) **Top-right:** Transizione da state: Idle count: 1 a state: Idle count: 2

   (c) **Bot-right:** Proseguimento verso state: Idle count: 3

   (d) **Bot-left:** Per ultimo state: Idle count: 4.

   (e) Il thread richiede il possesso del mutex e imposta state: Confirming

3. **Conferma del back-up**

   Quando si raggiunge lo state Confirming e count: 4, il sistema attiva la finestra di conferma. Questa finestra è visibile solo se lo schermo è attivo e operativo. Il suo scopo è prevenire avvii accidentali del backup, richiedendo un'azione esplicita da parte dell'utente. Lo stato di conferma segue due possibili percorsi:

   - **Conferma positiva (Bot-right):** L'utente conferma l'operazione, portando lo state a Confirmed e count: 5.

- **Annullamento(Window close/Top-left):** L'utente chiude la finestra o annulla l'operazione, riportando il sistema allo state iniziale: Idle count: 0.

4. **Avvio del back-up**

   Se l'operazione viene confermata, il sistema acquisisce il controllo del mutex. Successivamente, la variabile state passa a BackingUp, in cui il thread dedicato al backup inizia l'operazione. Durante questa fase, le risorse vengono gestite in modo efficiente per garantire un basso impatto sulle prestazioni del sistema, ovvero il thread mouse_movements rimane in attesa della CondVar.

5. **Ciclo continuo**

   Dopo il completamento del backup, il sistema ritorna allo stato iniziale state: Idle count: 0, pronto per gestire nuove richieste. Il ciclo garantisce un monitoraggio continuo senza interruzioni.

Per tutta la durata del ciclo avviene, nel thread separato: cpu_log_thread, la scrittura del file di log con il seguente formato:

Date(GG/MM/YYYY): 11/01/2025 Time: 14:30:45 - CPU usage: 15.5%

Capitolo 5

Ottimizzazione

L'applicazione è stata progettata per massimizzare le prestazioni, minimizzando l'uso delle risorse e garantendo un funzionamento fluido e reattivo. Queste ottimizzazioni vertono sia sula gestione dei thread che sulla struttura del codice, consentendo di raggiungere un equilibrio tra efficienza e modularità.

5.1 Ottimizzazione dei Thread

1. **Minimizzazione del carico della CPU**

   Per evitare che il thread mouse_movements consumi inutilmente risorse del sistema, viene messo in pausa con uno stato di sleep di 100 ms al termine di ogni ciclo operativo. In questo modo, il thread non consuma risorse in modo superfluo quando non è richiesto, garantendo una gestione efficiente delle risorse del sistema.

2. **Gestione dell'Event Loop della finestra**

   L'event loop della finestra, utilizzato per gestire la finestra di conferma, rimane in attesa dello state: Confirming, quando la finestra non è visibile. Questo evita il continuo polling di eventi inutili e riduce ulteriormente il carico del sistema. La finestra si riattiva immediatamente solo quando è necessaria un'interazione con l'utente, garantendo reattività e risparmio energetico.

3. **Sincronizzazione ottimizzata dei Thread**

   L'uso di mutex per la sincronizzazione garantisce che solo un thread acceda a risorse condivise in un determinato momento. Questo non solo evita conflitti, ma ottimizza anche il flusso di lavoro, poiché i thread non si bloccano inutilmente in attesa di risorse già disponibili.

5.2 Ottimizzazione del codice

1. **Bootstrap per configurazione automatica**

Per gestire l'auto-configurazione durante la fase di bootstrap, l'applicazione utilizza una libreria dedicata. L'adozione di questa libreria riduce la complessità del codice, inoltre questo garantisce che l'applicazione rimanga compatibile grazie agli aggiornamenti della libreria stessa.

2. **Design modulare**

   Il codice è organizzato in moduli separati per ogni componente principale dell'applicazione, come il monitoraggio del mouse, la gestione della finestra e il processo di backup. Questa modularità non solo semplifica lo sviluppo e la manutenzione, ma rende anche più semplice testare e aggiornare singole parti del sistema senza influenzare il resto dell'applicazione.

Capitolo 6

Conclusione

L'applicazione sviluppata rappresenta una soluzione versatile per la gestione dei backup su dispositivi desktop, garantendo accessibilità, affidabilità ed efficienza. La possibilità di avviare e gestire il backup anche in assenza di uno schermo funzionante la rende particolarmente utile in contesti critici, in cui le tradizionali interfacce grafiche non sarebbero utilizzabili. Attraverso un'architettura modulare e l'adozione di Rust, è stato possibile realizzare un software sicuro e facilmente manutenibile. L'uso di thread separati per le diverse operazioni, combinato con meccanismi di sincronizzazione ottimizzati e pause programmate, consente di ridurre l'impatto sulle risorse del sistema, garantendo al tempo stesso un comportamento reattivo. Le librerie selezionate per la gestione del bootstrap e del monitoraggio del consumo della CPU hanno semplificato lo sviluppo e garantito un'implementazione robusta e compatibile con i principali sistemi operativi, come Windows, macOS e Ubuntu. Questo approccio rende l'applicazione facilmente adattabile e scalabile per eventuali miglioramenti futuri. Il progetto raggiunge con successo gli obiettivi prefissati, offrendo un'esperienza d'uso intuitiva per utenti con competenze tecniche limitate, una gestione intelligente delle risorse di sistema e un'architettura che favorisce estendibilità e affidabilità. Per eventuali evoluzioni future, sarebbe possibile integrare funzionalità aggiuntive, come il supporto per servizi cloud, interfaccia remota o meccanismi di backup incrementale, ampliando ulteriormente le potenzialità dell'applicazione.