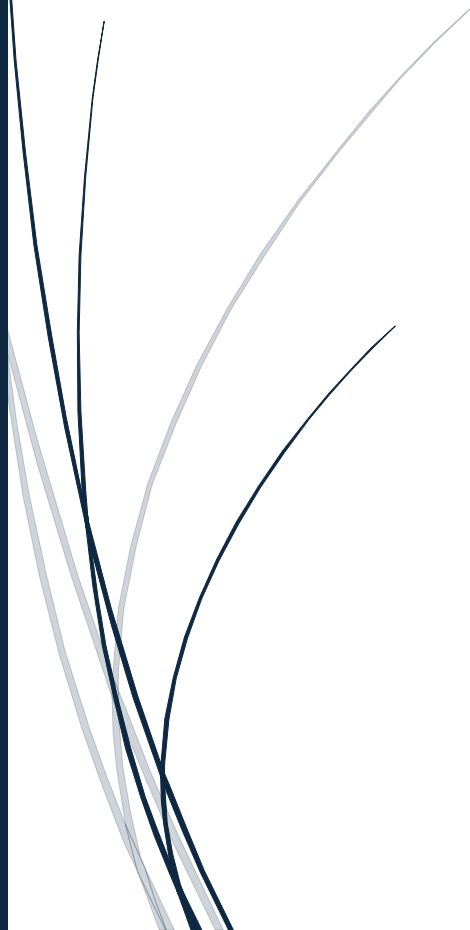


20/02/2025

PDS – Group 24

Relazione tecnica

Politecnico di Torino



Andrea Fontana s324581
Davide Jannussi s331391
Grazia De Mola s326640
Rebecca Audisio s334032

Sommario

Introduzione	2
User Manual - installazione	2
User Manual – funzionamento generale	3
Gestione del programma	6
File di configurazione.....	6
AppState.....	8
MyApp	10
Canali	12
Threads.....	13
Logica User Panel (GUI)	15
Struttura del programma	17
analytics.rs	17
detector.rs	18
first_sign.rs	21
confirm_sign.rs	21
main.rs	21
transfer.rs	22
utils.rs	23
ui	23
mod.rs.....	23
analytics.rs.....	25
backup.rs	25
Info.rs.....	26

Introduzione

Un'applicazione per PC progettata per eseguire backup in caso di schermo inutilizzabile. L'utente può attivare il backup tramite un comando grafico tracciato con il mouse, senza la necessità di interagire con una GUI tradizionale. Supporta anche diverse modalità di backup, inclusi file di tipo specifico e intere cartelle.




User Manual - installazione

Il programma è disponibile sia per Windows che per Linux (testato su Ubuntu) rispettivamente nei file:

- backup_app_group24_windows.zip
- backup_app_group24_linux.zip

Dal repository **github** andare nella cartella **download/** e scegliere la versione per il sistema operativo desiderato.

All'estrazione dell'archivio saranno presenti i seguenti files/directory:

 images	Cartella di file
 Sounds	Cartella di file
 backup_app_group24	Applicazione

L'applicazione è già pronta per essere eseguita.

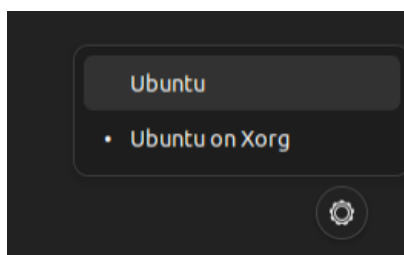
Il file eseguibile è stato creato lanciando il seguente comando dal terminale dell'ambiente di sviluppo:

- `cargo build --release`

Per ottenere la cartella di installazione si prende l'eseguibile, le cartelle **images/** e **Sounds/** e si comprimono in un archivio. (*Questo lavoro è già stato fatto dagli autori di questo programma*)

Nota: Spostare l'eseguibile dalla cartella di installazione potrebbe compromettere il corretto funzionamento del programma.

Su **Linux** l'applicazione è stata testata con VirtualBox. Per risolvere alcuni problemi della GUI si è impostato Ubuntu su Xorg:



La differenza principale tra Ubuntu e Ubuntu on Xorg riguarda il server grafico utilizzato per gestire l'interfaccia utente:

- **Ubuntu (predefinito)** → Usa Wayland come server grafico.
- **Ubuntu on Xorg** → Usa Xorg (X11) come server grafico.

Nota bene: durante le prime compilazioni del programma (su Linux) potrebbe essere necessario installare alcune librerie di sistema (*libglib2.0-dev pkg-config, libgdk-pixbuf2.0-dev libgtk-3-dev, libasound2-dev*)

User Manual – funzionamento generale

Una volta aperta l'applicazione si aprirà un'interfaccia GUI in cui è presente una barra laterale che l'utente può utilizzare per spostarsi tra i vari pannelli e verrà visualizzato il pannello backup.

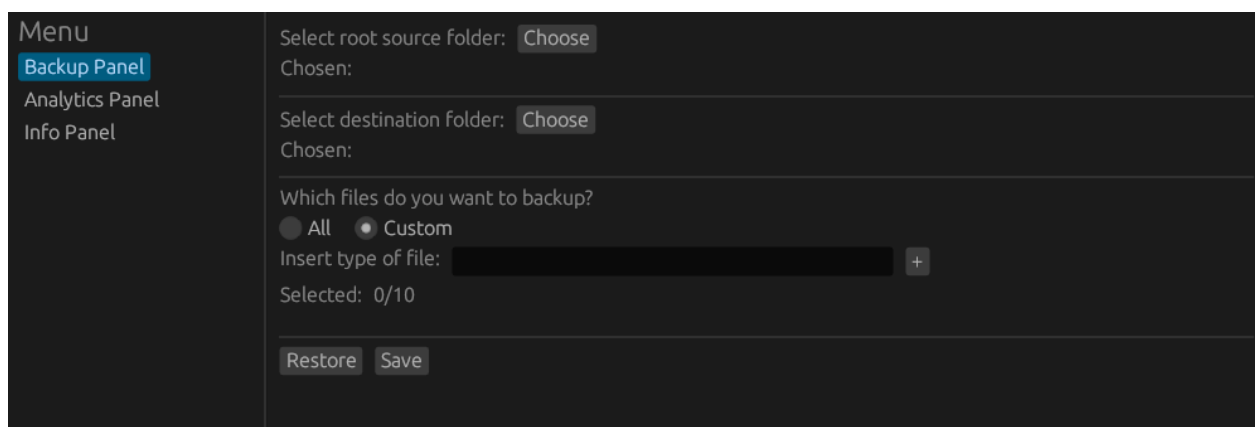
Nel pannello backup è possibile scegliere la cartella che si vuole copiare durante il backup e la cartella di destinazione mediante i pulsanti con la scritta “Choose”.

Dopo aver scelto le cartelle, l'utente può decidere se copiare tutto il contenuto della cartella o se vuole copiare solo i file di un determinato formato inserendo l'estensione nella cella di input (es. “.txt”, “.png”) e cliccando il pulsante “+”.

L'utente può rimuovere un'estensione cliccando sull'etichetta relativa.

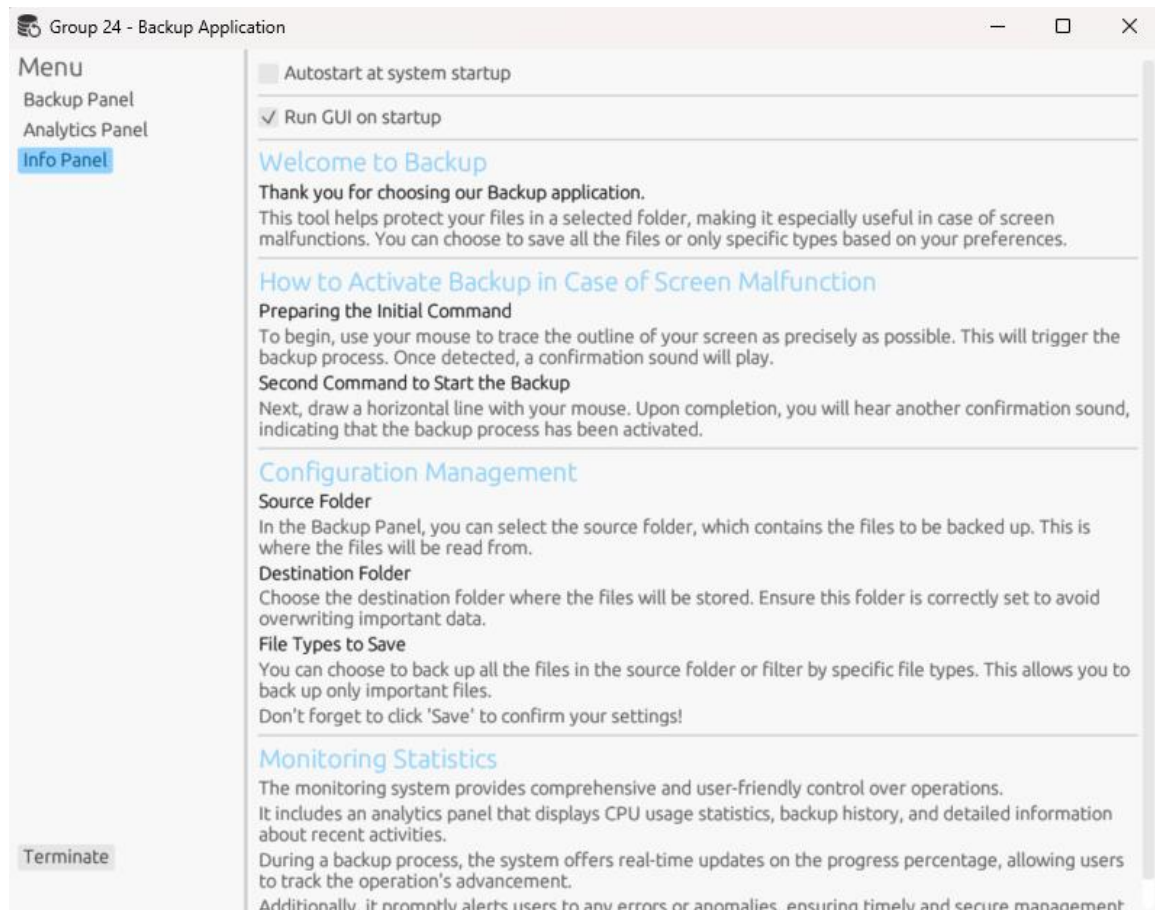
È importante che alla fine della configurazione l'utente clicchi sul pulsante “**Save**”, per salvare la configurazione.

Il tasto “**Restore**” permette all'utente di tornare alla configurazione precedente alle modifiche.



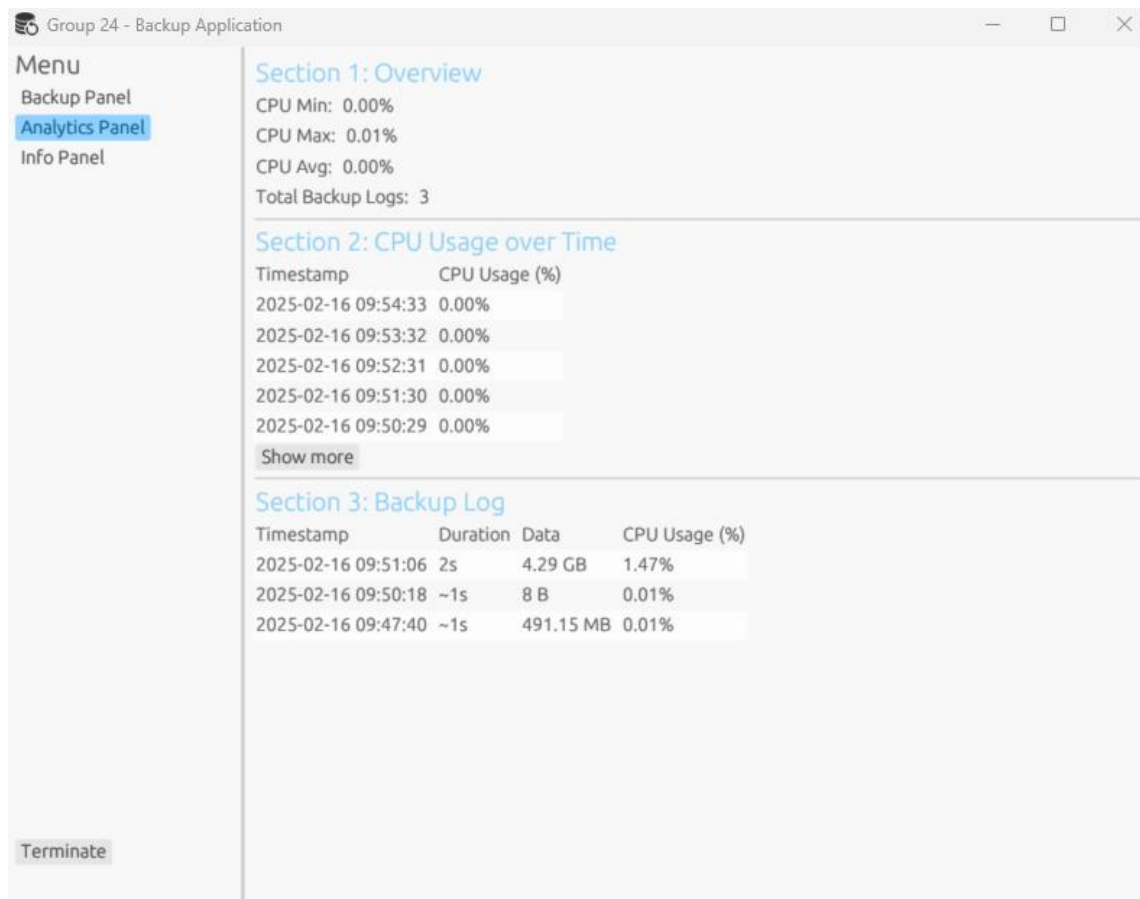
L'info panel è un pannello che riporta le informazioni per utilizzare correttamente l'applicazione, specificando a cosa servono e come utilizzare i vari pannelli.

Nella parte superiore del pannello sono presenti due checkbox, la prima permette di far partire l'applicazione automaticamente dopo l'avvio del sistema facendo partire il monitoraggio del mouse e la seconda permette di scegliere se far apparire all'avvio l'interfaccia GUI o meno.

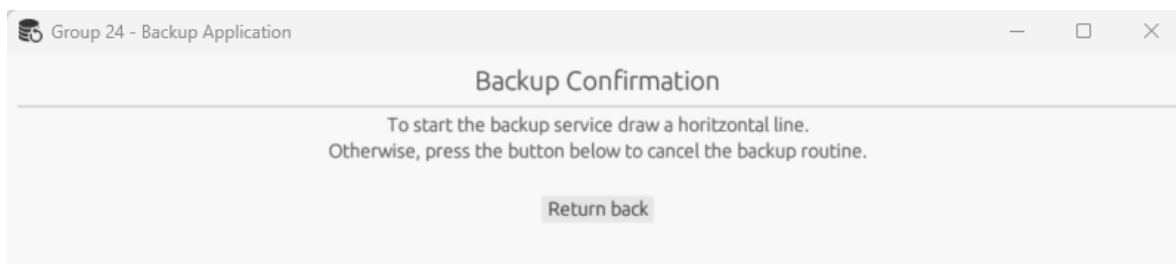


Analytics Panel mostra le statistiche sul consumo di CPU e backup effettuati:

- **Overview:** Mostra statistiche generali sull'uso della CPU e il numero totale di log di backup disponibili.
- **CPU usage over time:** Visualizza l'uso della CPU in una tabella, mostrando gli ultimi log registrati. Sono presenti pulsanti per vedere più o meno log.
- **Backup Log:** Visualizza i dettagli dei log di backup in una tabella, con pulsanti per vedere più o meno log ("*Show more*").

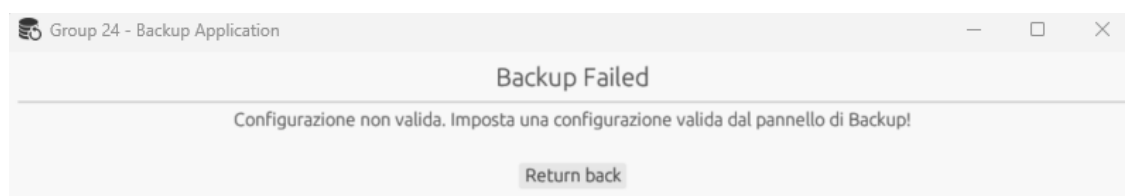


Per far partire il backup (anche in caso di malfunzionamento dello schermo) l'utente deve tracciare il contorno dello schermo, con più precisione possibile. L'applicazione emetterà un suono per avvisare l'utente che la sequenza è stata riconosciuta.

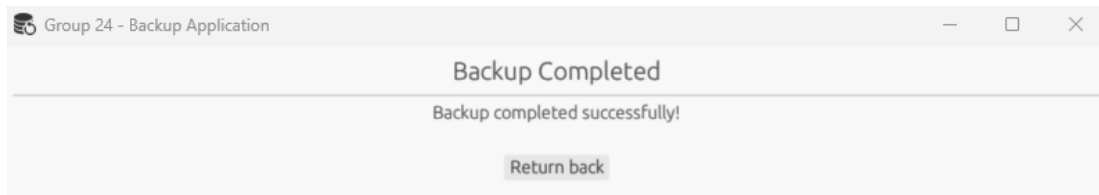


A questo punto l'utente deve tracciare un segno orizzontale, per confermare il backup, altrimenti può decidere di tornare alla schermata iniziale della GUI.

Configurazione non valida:

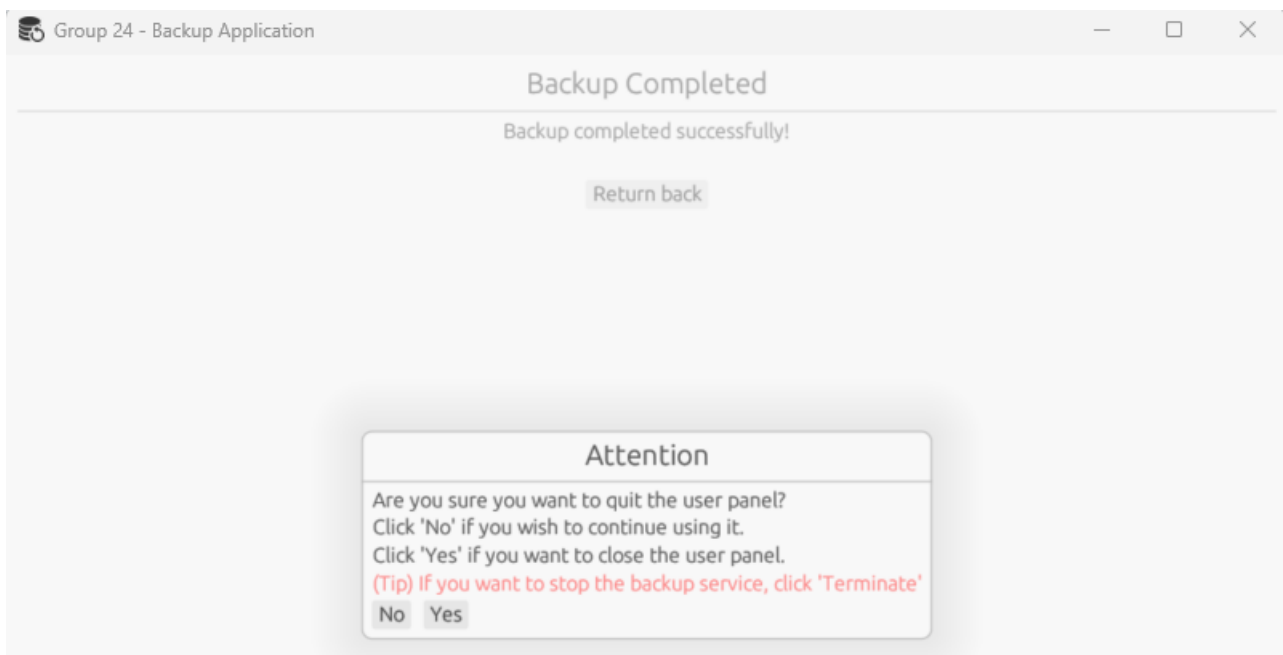


Infine, quando la copia di tutti i file sarà completata, si sentirà un altro suono di conferma.



Nel caso di errore o di interruzione da parte dell'utente si sentirà un suono di allarme.

L'utente può chiudere l'interfaccia GUI chiudendo la finestra mediante il pulsante "X", ma l'applicazione continuerà a funzionare monitorando il movimento del mouse.



Se invece l'utente non vuole più utilizzare l'applicazione, deve cliccare sul tasto "Terminate".

Quando la GUI non è visibile l'utente può in ogni momento cliccare sull'eseguibile per mostrarla. La GUI verrà riaperta dall'ultima finestra dalla quale era stata chiusa.

Gestione del programma

File di configurazione

config_build.toml

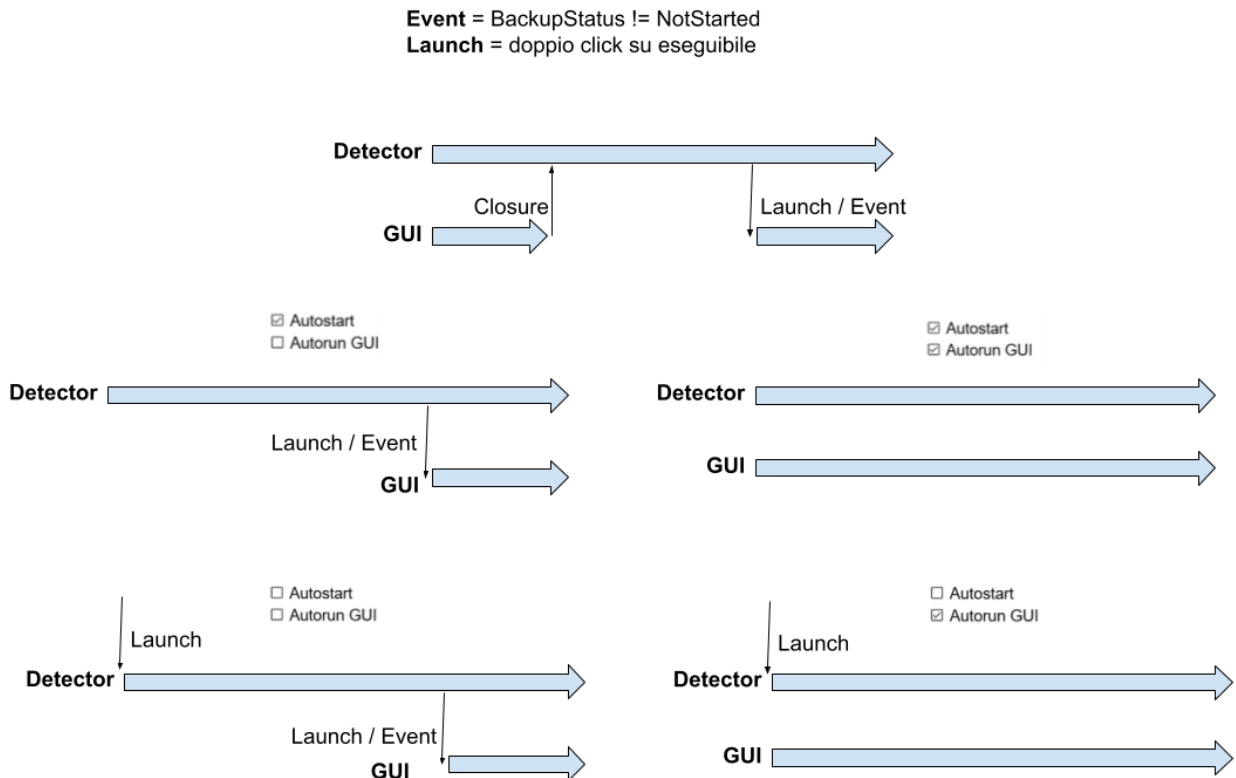
- **source_folder** → percorso della cartella da dove selezionare i file per il backup
- **destination_folder** → percorso della cartella dove salvare i file del backup

- **backup_type** →
 - **total**: tutti i file di source_folder vengono salvati in destination folder;
 - **custom**: solamente i file corrispondenti alle estensioni selezionate vengono trasferiti in destination_folder
- **file_types** → contiene l'elenco delle estensioni dei file, massimo 10
- **display** → determina comparsa user panel (GUI) quando l'applicazione viene lanciata per la prima volta oppure dopo il riavvio se si è scelto di farla partire automaticamente; se il valore è '**false**' allora lo user panel (GUI) non viene mostrato. Nota bene: la GUI può comparire anche se questo campo è '**false**' in quanto la comparsa/scomparsa della GUI viene gestita da un campo della struct **AppState**, che è '**dinamico**' e quindi riflette il comportamento da parte dell'utente di utilizzo dell'applicazione.

lock.toml

- **boot_time** → contiene il timestamp esatto (data e orario, fino ai minuti) di quando è stato fatto il boot del sistema operativo. Viene utilizzato per implementare il paradigma della **SINGLE INSTANCE** → Mentre il sistema operativo è in esecuzione, solamente una istanza dell'applicazione può essere eseguita.
 - Se quando lancio l'applicazione il boot_time è maggiore del precedente salvato nella configurazione, allora lo sovrascrivo → si tratta di un nuovo avvio, quindi viene rispettata la **SINGLE INSTANCE**;
 - Se il boot_time è identico o inferiore a quello salvato nella configurazione allora vuol dire che sto tentando di aprire di nuovo l'applicazione mentre è già in esecuzione. Questo può essere visto come un tentativo di aprire lo user panel (la **GUI**), quindi si scrive il campo **show_gui** a true, sempre nel file **lock.toml**. Questo campo servirà per mostrare, eventualmente, la GUI, se non è già stata aperta.
- **show_gui** → Serve per lanciare la GUI, mentre l'applicazione è già attiva ma lo user panel è stato chiuso. Questo campo viene monitorato da un thread apposito, sotto la funzione "**monitor_lock_file**", ogni secondo. Se il file **lock.toml** è stato modificato si controlla se **show_gui = true**; se sì, e il valore di **AppState.display** è "**false**", quindi la GUI non è visibile al momento, allora viene messo nel canale condiviso in trasmissione il valore "**showGUI**". Dopodiché **show_gui** viene impostato a false.

Funzionamento avvio automatico GUI:



AppState

Struct che contiene le principali informazioni che servono all'applicazione, non solo alla GUI (user panel), per funzionare → anche quando la GUI non viene mostrata, **AppState** rimane salvato. La nuova GUI lanciata si appoggia sull'**AppState** corrente.

Viene incapsulato in un **Arc<Mutex<AppState>>**, perché deve essere condiviso da più thread parallelamente in esecuzione.

- **Arc** permette di creare più riferimenti a **AppState** mantenendolo in vita finché esiste almeno un riferimento.
- **Mutex<T>** è necessario perché **AppState** è mutabile, e Rust non permette di modificare un valore da più thread senza protezione contro le race condition.

Viene inizializzato a partire dalla lettura del file di configurazione **config_build.toml**, dai registri di sistema per capire se l'applicazione deve essere lanciata in automatico all'avvio del sistema operativo.

Campi:

- **current_panel** → contiene l'informazione sulla UI attuale da mostrare all'utente (Backup; Analytics; Info)
- **source_folder** → cartella sorgente correntemente selezionata
- **destination_folder** → cartella destinazione correntemente selezionata
- **backup_type** → tipo di backup correntemente selezionato
- **file_types** → estensioni dei file da salvare
- **new_file_type** → estensione da inserire
- **info_message** → campo testuale da mostrare per modale di info (info_modal, che può essere a titolo di *avviso* oppure di *successo*)
- **info_source** → discrimina il modale info_modal in *avviso* oppure in *successo*
- **show_info_modal** → serve per renderizzare il modale di info
- **error_message** → campo testuale che viene riempito in caso di errori durante l'inserimento dell'estensione dei file oppure quando si prova a salvare la configurazione dinamica nel file statico **config_build.toml** (cartelle selezionate, estensioni di file, ecc..)
- **error_source** → discrimina il modale di errore in *FileTypeError* oppure in *SaveOperationError*
- **exit_message** → mostra schermata di errore anomalo che porta alla chiusura incondizionata del programma
- **show_error_modal** → server per renderizzare il modale di errore
- **show_confirmation_modal** → renderizza il modale per chiedere all'utente la conferma di chiusura dell'applicazione, quando tenta di chiudere lo user panel (GUI)
- **display** → display "dinamico", serve al programma per capire quando la GUI non è mostrata
- **backup_status** → enum contenente uno dei seguenti valori:
 - **NotStarted**: selezione di default, il backup non è ancora iniziato. Viene mostrato il **pannello principale**, altrimenti viene mostrato il **pannello di backup**

- **ToConfirm**: selezionato quando il detector è in attesa del secondo tracciamento di conferma, prima di avviare il backup
- **InProgress**: selezionato quando viene lanciato il thread per il backup
- **CompletedSuccess**: selezionato per informare l'utente che il backup è stato completato con successo
- **Canceled**: l'utente ha deciso di interrompere il backup in corso
- **CompletedError(String)**: selezionato quando si è verificata una situazione con errore nel programma
- **auto_start_enabled** → permette all'utente di lanciare il programma in automatico dopo il boot del sistema operativo. Il valore viene preso dai registri di sistema (Windows) oppure da una apposita directory (Linux) mediante la funzione **check_auto_start_status**
- **run_gui** → display "statico", serve al programma per capire se dopo il primo avvio dell'applicazione bisogna mostrare la GUI oppure no

MyApp

Struct che contiene:

- **Arc<Mutex<AppState>>**
- Il canale, in trasmissione, per comunicare con il detector (**Sender<String>**)
 - Quando **BackupStatus::ToConfirm**
 - Invia "**resetWaiting**" sul canale condiviso in trasmissione verso il detector, quando si vuole annullare il backup dopo il tracciamento del rettangolo e prima del tracciamento della linea orizzontale
- Il canale, in trasmissione, per comunicare lo stop al thread di backup (**Sender<String>**)
 - Quando **BackupStatus::InProgress**
 - Invia "**stop**" sul canale condiviso in trasmissione verso il thread di backup, quando si vuole annullare il backup dopo che è iniziato → Questo messaggio viene controllato prima di ogni copia di file, da parte del thread di backup
- Barra di progresso da mostrare durante il trasferimento dei file (**Arc<Mutex<f32>>**)

- File che viene mostrato solamente durante il processo di backup, rappresenta il file corrente che si sta copiando verso la destinazione:

Arc<Mutex<Option<String>>>

- implementa il tratto **App**
 - Il trait **App** è fornito da **eframe** ed è utilizzato per definire il comportamento di una applicazione GUI basata su **egui**.
 - Questo trait permette di **collegare la logica della UI all'aggiornamento continuo** dell'interfaccia grafica.
 - metodo **update**:
 - viene chiamato a ogni frame per aggiornare l'interfaccia grafica
 - Se il backup non è ancora iniziato, si mostra il pannello principale (**main_panel**).
 - Se il backup è in corso, si mostra la finestra del backup (**show_backup_window**).
 - Se esiste un messaggio di uscita (**AppState.exit_message** non è None), viene mostrato un pannello di uscita con il messaggio di errore e la funzione update termina (return).
 - Se **AppState.show_confirmation_modal** è true, si mostra un modale di conferma uscita per evitare che l'utente chiuda l'applicazione per errore. Gestito da funzione grafica **render_modal_exit**
 - Gestisce la chiusura dello user panel (GUI)
 - In caso di conferma, imposta **AppState.display** a false e quindi il **frame**.
 - metodo **on_close_event**:
 - Viene chiamato quando l'utente cerca di chiudere la finestra, e può impedire la chiusura se necessario.
 - Deve restituire **true** se la finestra può essere chiusa o **false** se deve rimanere aperta.
 - Viene mostrato il modale di conferma chiusura

Canali

“showGUI”

- Trasmissione:
 - thread che monitora il campo **show_gui** nel file **lock.toml**;
 - thread che esegue la procedura di backup, dopo che questo termina, indipendentemente dall’esito
 - thread detector:
 - dopo tracciamento della prima sequenza
 - dopo tracciamento seconda sequenza
- Ricezione:
 - thread principale “main” → lo stato del programma viene aggiornato per mostrare la GUI

“resetWaiting”

- Trasmissione:
 - Da pannello di backup dopo il tracciamento della prima sequenza → l’utente decide di annullare la sequenza e di tornare in uno stato di ascolto
- Ricezione:
 - thread apposito presente nel thread detector, reimposta il tracciamento dei bordi lungo lo schermo da zero

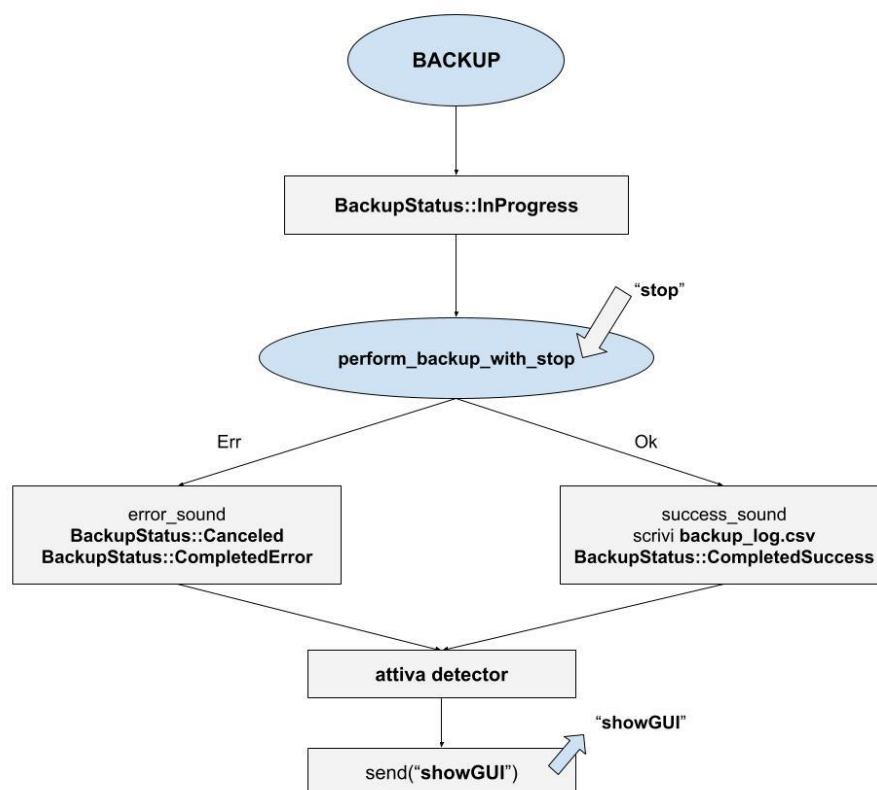
“stop”

- Trasmissione:
 - Dal pannello di backup mentre è in atto il trasferimento dei dati, l’utente decide di interrompere il processo.
- Ricezione:
 - thread backup, all’inizio della copia di ogni file controlla se nel canale è stato inviato un segnale di terminazione. Se l’utente decide di interrompere il backup mentre è in corso il trasferimento di un dato file, questo verrà copiato interamente e poi il backup verrà interrotto prima della prossima copia.

Threads

Backup

- Viene lanciato all'interno della funzione **avvia_backup** dopo che la seconda sequenza di tracciamento è stata riconosciuta dal thread di backup. Il detector viene disattivato
- Viene eseguita la copia dei file, e viene interrotta nel caso venga ricevuto “**stop**” sul canale condiviso
- Vengono emessi degli effetti sonori in caso sia di successo che di errore (interruzione), e vengono modificati gli stadi della enum **BackupStatus** per mostrare all’utente lo stato di avanzamento del backup
- In caso di successo vengono salvate delle metriche (timestamp, durante trasferimento, dimensione totale file copiati, cpu utilizzata) nel file **backup_log.csv**
- Alla fine della procedura si attiva il detector per riconoscere una nuova sequenza e viene mandato “**showGUI**” sul canale condiviso per far apparire la GUI



Listener “resetWaiting”

- Lanciato dal thread detector, la sua unica funzione è quella di rimanere in ascolto per messaggi “**resetWaiting**” sul canale condiviso tra detector e user panel

Listener segnali

- Sia per Windows che per Linux, cattura segnali del tipo SIGINT e SIGTERM, il programma viene terminato in modo corretto, eliminando il file **lock.toml**, che impedirebbe una riapertura dell'applicazione dopo una chiusura imprevista
- SIGKILL non può essere catturato, in questo caso l'applicazione deve essere reinstallata.

Logging CPU usage

- registra consumo CPU in file **cpu_usage_log.csv** ogni minuto, mediante funzione **log_cpu_usage_to_csv**
- timestamp, CPU occupata

Detector

- implementa la logica di riconoscimento del segno tracciato dall'utente per avviare il backup
- logica applicativa principale contenuta nel file **detector.rs**
- Ha bisogno dell'**AppState** condiviso, il canale in trasmissione per inviare “**showGUI**”, il canale in ricezione per ascoltare “**resetWaiting**”, il canale in ricezione per ascoltare “**stop**”, una variabile atomica thread-safe per abilitare/disabilitare il detector durante la fase di backup

Sound Playback

- segue la riproduzione audio in un **thread separato**.
- il **thread termina automaticamente** dopo aver riprodotto l'audio.

Monitoring del file lock.toml

- controlla campo **show_gui** e utilizza canale condiviso per trasmettere “**showGUI**”

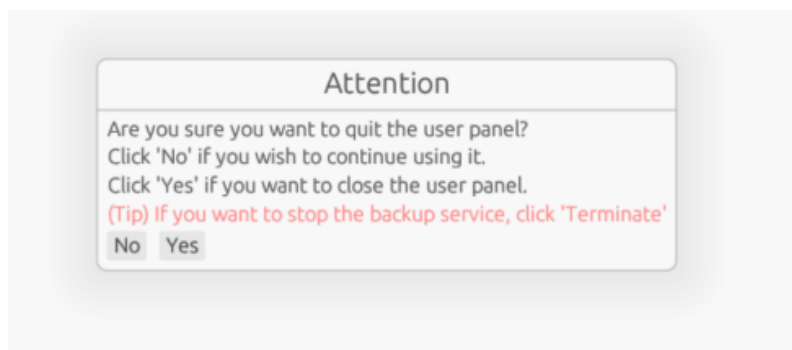
Programma principale: main.rs

Logica User Panel (GUI)

La GUI rappresenta un componente importante dell'applicazione, in quanto permette una interazione diretta con l'utente, per la configurazione del backup (**schermata backup**), il monitoraggio delle statistiche di consumo CPU e backup effettuati (**schermata analytics**), settaggio di avvio in automatico a ogni avvio e se mostrare o meno lo user panel al prossimo riavvio (**schermata info**).

Al primo avvio viene mostrata la GUI che permette di configurare il file **config_build.toml** per effettuare un backup. Se il file in questione non dovesse avere una configurazione valida, il backup non potrà essere effettuato.

Durante l'utilizzo della GUI si può decidere di chiudere lo user panel, premendo sull'apposita X di chiusura del frame, compare dunque un apposito messaggio che chiede conferma dell'azione all'utente.



Durante la comparsa di questa finestra, e in generale di ogni modale, quindi che copre la GUI sottostante, si potrà interagire solamente col modale stesso, in quanto c'è un **overlay** che **blocca** la user interface. Questo avviene sia per il main panel, che per la sidebar (a sinistra), e quando viene mostrata la UI per il backup in corso.

Una volta che la GUI è stata chiusa, l'applicazione continua a essere eseguita; infatti, sono presenti molteplici thread che sono ancora in esecuzione, tra cui quello **“detector”**, che è in grado di far apparire la GUI ogni qual volta si verifica una interazione sul **canale condiviso** tra main e detector, come ad esempio “prima sequenza di backup riconosciuta” oppure “backup completato”.

La GUI compare, ovviamente, se **AppState.display** è false, altrimenti significa che già viene mostrata all'utente e quindi, anche se si verifica interazione sul canale condiviso, non c'è necessità di segnalare lo stato di far apparire un nuovo user panel, proprio perché è già presente.

La GUI può essere riaperta anche cliccando due volte sull'eseguibile, infatti nel file **lock.toml** viene impostato il parametro **“show_gui”** a true, che segnala al **thread di monitoring del file lock.toml** di inviare al canale condiviso con il **main** il messaggio **“showGUI”**.

Questo messaggio verrà letto dal main quando entrerà in un apposito **loop**.

- Se è stato scelto di mostrare la GUI all'avvio dell'applicazione, il programma non entra subito nel loop, ma entrerà solamente quando l'utente chiude lo user panel, quindi quando il frame grafico viene chiuso dopo la conferma dell'utente.
- Se è stato scelto di non mostrare la GUI all'avvio dell'applicazione, il programma entra subito nel loop.

Gestione dell'avvio iniziale della GUI → **AppState.display** (valore dinamico) viene sincronizzato con **AppState.run_gui** (valore statico), che rappresenta il campo **"display"** preso dal file di configurazione **lock.toml**.

La GUI viene avviata con **eframe::run_native**, che gestisce il ciclo di rendering e gli eventi della GUI.

L'istanza dell'App si trova nella struct **MyApp**.

Se si è scelto di non mostrare la GUI immediatamente all'avvio oppure lo user panel precedente è stato chiuso, allora si entra nel loop del **main.rs**:

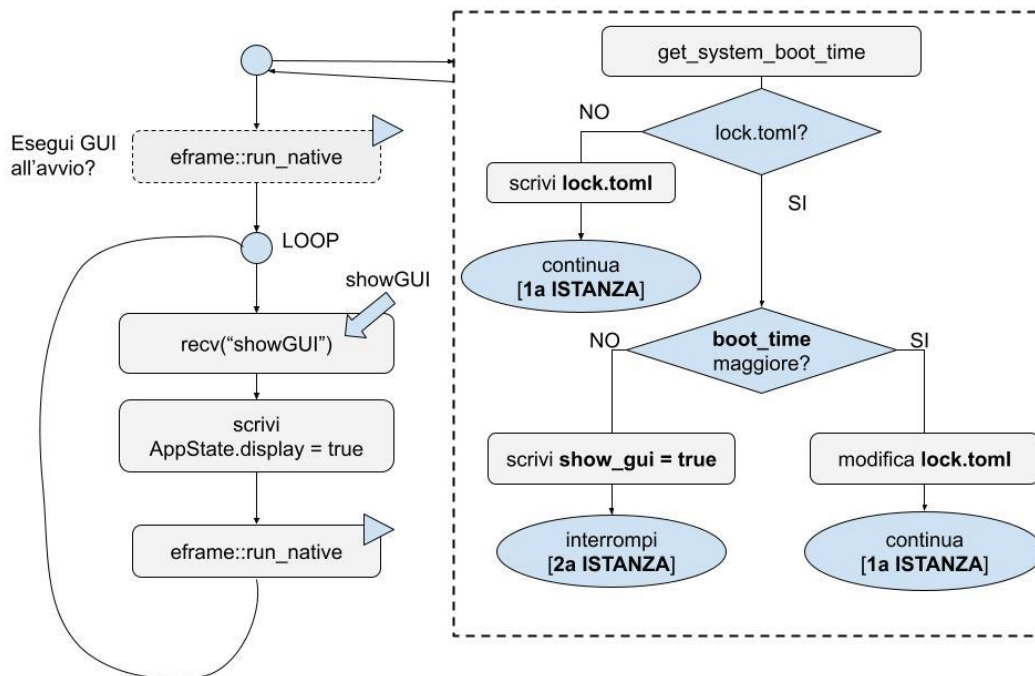
- Si mette in attesa, quindi bloccante, sul canale rx, in ricezione, del messaggio **"showGUI"**; non appena arriva **AppState.display** diventa **true**;
- Si procede con l'esecuzione di una nuova GUI con **eframe::run_native**, passando una nuova struct **MyApp** (che incapsula sempre lo stato condiviso dell'applicazione **AppState**)
- Si ritorna all'attesa del canale condiviso, in ricezione, non appena la GUI viene chiusa.

showGUI viene trasmesso sul **canale condiviso** dai seguenti:

- **Dal thread di backup**, non appena questo termina, sia in situazioni di successo, di errore, oppure quando viene interrotto dall'utente. (funzione **avvia_backup** lanciata dal **detector**, che sospende la sua esecuzione)
- **Dal thread detector**, dopo che si è tracciato il contorno rettangolare lungo lo schermo.
- **Dal thread detector**, dopo che si è tracciato il segno di conferma.
- **Dal thread monitoring del file lock.toml**, dopo che viene letto **show_gui = true** in **lock.toml**
- **Nota bene**: in tutti i casi sopra menzionati, il messaggio viene inviato solamente quando la GUI non è attiva, cioè quando **AppState.display == false**, altrimenti

non c'è bisogno di inviare il messaggio perché lo user panel viene già mostrato all'utente.

Grafico con funzionamento:



Struttura del programma

analytics.rs

Il modulo serve per monitorare e registrare i dettagli relativi al backup dei dati e all'utilizzo della CPU. Il codice utilizza diverse librerie per gestire operazioni sui file, la gestione delle date e la raccolta delle statistiche di sistema. Due principali funzioni vengono implementate: una per registrare i dettagli del backup in un file CSV e l'altra per registrare l'utilizzo della CPU ogni minuto.

Il codice utilizza le seguenti librerie:

- `std::fs::OpenOptions` e `std::fs:` per la gestione dei file.
- `std::io::{Write, BufWriter}`: per la scrittura bufferizzata nei file.
- `chrono::Local`: per ottenere la data e l'ora locale.
- `std::{thread, time::Duration}`: per gestire il ritardo nelle operazioni.

- `systemstat::{System, Platform}`: per raccogliere le statistiche del sistema.

Funzione **initialize_log_file**:

- Questa funzione ha lo scopo di inizializzare il file di log CSV. Verifica se il file esiste e se è vuoto. Se il file è vuoto o non esiste, scrive l'intestazione passata come parametro. La funzione restituisce un **BufWriter** per la scrittura bufferizzata.

Funzione **log_backup_data_to_csv**:

- Questa funzione registra i dettagli del backup (timestamp, dimensione totale, durata e utilizzo della CPU) in un file CSV. Chiama **initialize_log_file** per inizializzare il file e poi scrive i dettagli. La funzione gestisce eventuali errori durante la scrittura e il flush dei dati nel file.

Funzione **log_cpu_usage_to_csv**:

- Questa funzione registra l'utilizzo della CPU in un file CSV ogni minuto. Inizializza il file di log usando **initialize_log_file**, crea un'istanza di `System` per raccogliere le statistiche del sistema e poi entra in un ciclo infinito dove, ogni minuto, ottiene e registra il carico della CPU.

Le funzioni sono dotate di meccanismi di gestione degli errori per assicurare che eventuali problemi durante l'accesso o la scrittura nei file vengano segnalati.

detector.rs

Il modulo `detector.rs` implementa la logica di riconoscimento del segno tracciato dall'utente per avviare il backup.

Avvio del Detector

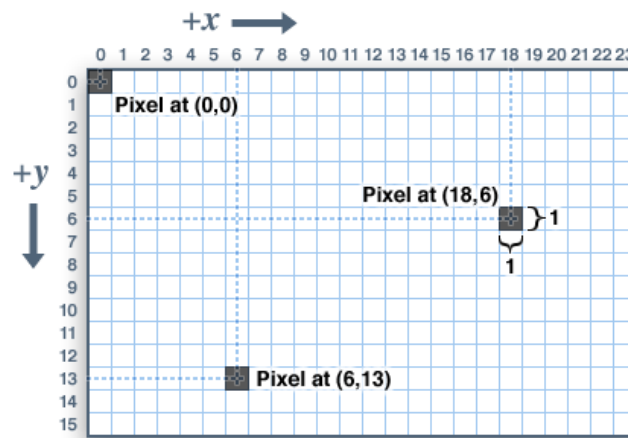
Vengono acquisite le coordinate dello schermo mediante la funzione **get_screen_resolution**, (per Windows viene corretta anche la **DPI awareness**, in caso di ridimensionamento dello schermo).

Il thread **Listener “resetWaiting”** viene lanciato.

Si inizializza una procedura di ascolto di eventi del sistema (**pressione tasto sinistro, rilascio tasto sinistro, movimento del mouse**).

Qualora si dovesse verificare un **evento**, come pressione del tasto sinistro del mouse, ma il detector non è attivo (variabile atomica), allora l'evento viene ignorato. Tuttavia, **l'event listener** continuerà ad essere in ascolto per altri eventi di sistema.

Alla pressione del tasto sinistro del mouse, il modulo memorizza le coordinate fino al rilascio.



Verifica del Contorno

Una volta rilasciato il pulsante, il detector controlla se l'utente ha tracciato correttamente il contorno dello schermo (entro una certa tolleranza). In caso positivo, emette un segnale audio e invia un messaggio sul canale ("**showGUI**") per notificare di far apparire la finestra di conferma.

Controllo di Annullamento

In parallelo, un secondo thread resta in ascolto di eventuali messaggi di annullamento del backup da parte dell'utente attraverso la gui. Se arriva un messaggio di reset ("**resetWaiting**"), il backup viene annullato e il detector torna al riconoscimento del contorno.

Conferma e Avvio del Backup

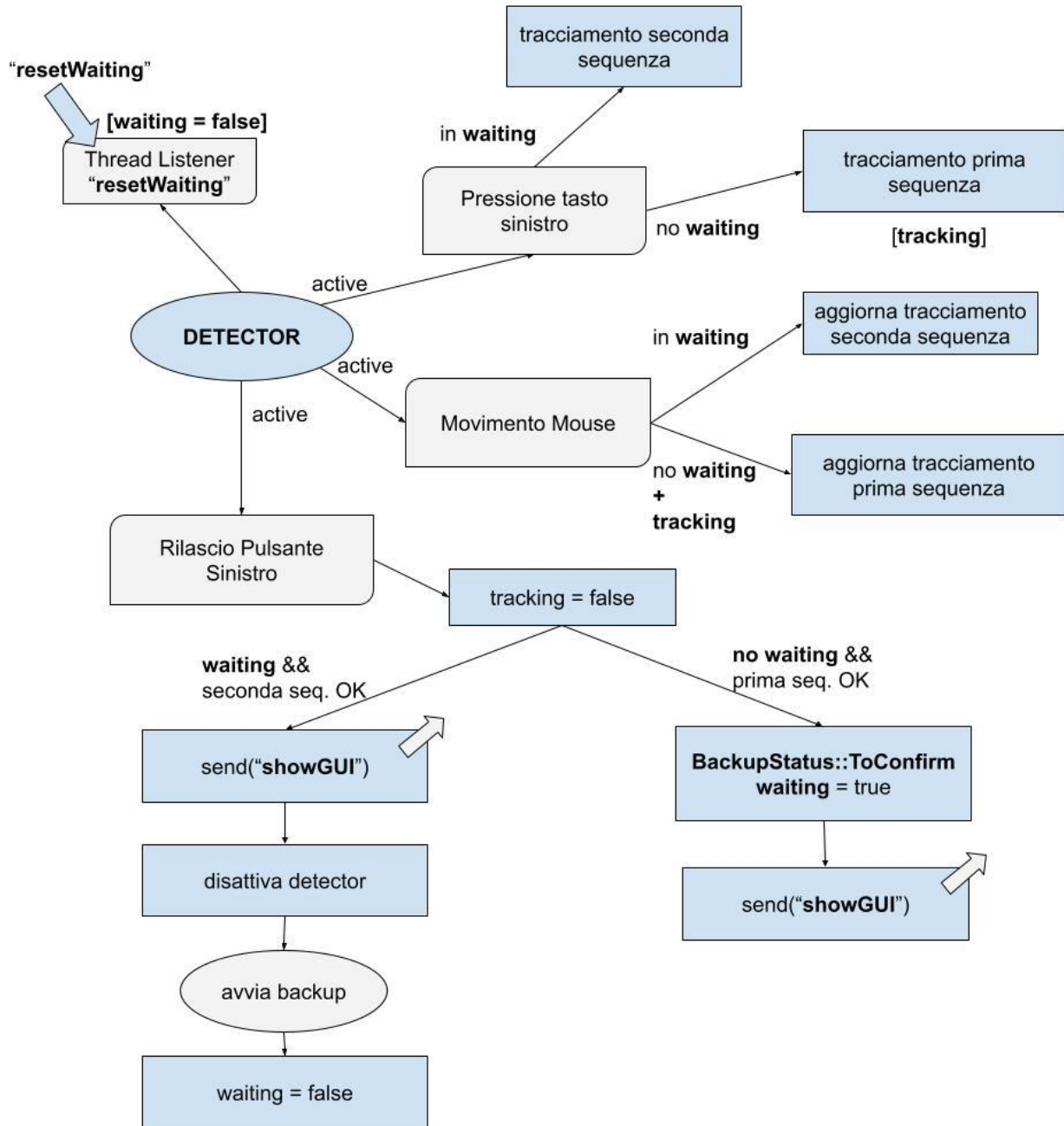
Se invece l'utente traccia la linea orizzontale di conferma, il detector viene fermato e si avvia un nuovo thread dedicato al backup.

Notifica alla GUI

Al termine del processo di backup, un altro messaggio viene inviato alla GUI per mostrarne l'esito in una schermata. Infine, il processo riparte da zero e il detector torna al riconoscimento del contorno.

Stati del detector:

waiting = in attesa della seconda sequenza
tracking = tracciamento attivo prima sequenza
active = detector attivo



first_sign.rs

Registra i punti lungo i quattro lati dello schermo e li confronta con una griglia immaginaria suddivisa in segmenti. Se ogni segmento viene coperto almeno una volta, il contorno risulta completato.

- **update_edges_rectangle** → Questo metodo serve per accumulare i punti che si trovano sui bordi del rettangolo, evitando duplicati.
- **is_contour_complete** → verifica se ogni bordo ha abbastanza punti **distribuiti uniformemente**. Se almeno un bordo **non è stato completamente coperto**, la funzione ritorna false.
- **check_coverage** → verifica se un bordo ha abbastanza punti distribuiti nei segmenti stabiliti

confirm_sign.rs

Verifica che l'utente abbia tracciato una linea verticale

main.rs

Questo file rappresenta il modulo principale. Contiene la logica di gestione dell'esecuzione dell'applicazione, il controllo **dell'istanza singola e l'interfaccia grafica**.

Gestione del file di lock: evita l'esecuzione di più istanze controllando il tempo di avvio del sistema e la presenza del file **lock.toml**.

Configurazione dell'ambiente: imposta la directory di lavoro sull'eseguibile mediante la funzione **set_working_directory_to_executable()**

Gestione dei segnali: intercetta i segnali per terminare in modo sicuro.

- Condizioni di **panic**, **SIGINT**, e **SIGTERM** vengono catturati e rimuovono i seguenti file: **lock.toml** e **cpu_usage_log.csv**.
- **GESTIONE ARRESTO ANOMALO:** La rimozione del file **lock.toml** è fondamentale, altrimenti in caso di errore imprevisto durante l'esecuzione del programma, l'utente non riuscirebbe ad aprire nuovamente l'applicazione (data l'esistenza del file **lock.toml** che implica che una **istanza** dell'applicazione è in esecuzione, e quindi non può esserne lanciata un'altra)

Logging della CPU: avvia un thread separato per registrare l'uso della CPU.

Comunicazione: utilizza canali `mpsc` per la comunicazione con la gui, il backup e il detector.

Legge configurazione: **AppState** viene inizializzato in base ai valori letti dal file **config_build.toml** qualora esistesse, altrimenti si procede per l'inizializzazione di default

Avvio thread di monitoring del file lock.toml

Avvio del detector: esegue in un thread separato il monitoraggio delle attività del mouse del detector.

Avvio interfaccia grafica: viene avviata l'interfaccia grafica utilizzando **`eframe::run_native()`** e inizializzata **MyApp**. Inoltre, quando l'interfaccia viene chiusa, il thread principale resta in attesa di messaggi da altri componenti. Se riceve **"showGUI"** la gui viene riavviata. Avviene la sincronizzazione tra il valore di display statico (**`run_gui`**) con quello dinamico (**`display`**) della struct **AppState**, per mostrare o meno la GUI al primo avvio dell'applicazione.

Gestione UI: viene implementato il tratto **App** per la struct **MyApp** per definire il comportamento della ui, sfruttando due metodi:

- **update:** viene eseguito continuamente per aggiornare e renderizzare l'interfaccia.
- **on_close_event:** gestisce la chiusura della finestra attivando un modale per chiedere conferma all'utente prima di chiudere l'applicazione.

transfer.rs

Questo file contiene le funzioni che permettono di eseguire il backup, ossia la copia dei file dalla cartella sorgente a quella di destinazione, con la possibilità di interrompere il processo su richiesta.

Inizialmente vengono recuperate le informazioni di backup che sono state configurate dall'utente. Viene verificato che la configurazione da **config_build.toml** sia valida e quale modalità è stata scelta, se quella totale o quella personalizzata, quindi con l'elenco delle estensioni dei file scelti.

Prima di avviare il processo che permette la copia dei file, viene avviato un cronometro per misurare la durata dell'operazione, emesso un suono per indicare l'inizio del backup e vengono contati i file totali per permettere di monitorare i progressi durante il backup.

Viene quindi chiamata la funzione **backup_folder_with_stop**, che si occupa di copiare i file in modo ricorsivo e permette di raccogliere i dati per visualizzare una barra di progresso e i dati utili per poter monitorare le prestazioni durante il processo. Il

processo monitora costantemente se è stato inviato un comando di interruzione “**stop**” sul canale condiviso.

Se l’utente decide di fermare il backup, il programma interrompe la copia, riproduce un suono di avviso e restituisce l’errore.

Se l’utente non interrompe il backup e questo va a buon fine, viene emesso un suono di conferma operazione, raccolti i dati sul consumo di CPU e salvate le metriche di backup (timestamp, dimensione totale trasferita, tempo totale trascorso, utilizzo CPU) nel file di log **backup_log.csv**

utils.rs

Contiene una serie di funzioni utili per la corretta esecuzione del programma, tra queste viene menzionata “**monitor_lock_file**” che lancia il thread di monitoring del file **lock.toml** all’avvio del programma.

ui

modulo che gestisce la UI (User Interface) e contiene i seguenti file: **mod.rs**; **analytics.rs**; **backup.rs**; **info.rs**.

mod.rs

Questo file rappresenta il nucleo dell’interfaccia grafica dell’applicazione. Dichiarare e inizializza le strutture dati necessarie per la navigazione tra le diverse sezioni dell’applicazione (**PanelType**) e per distinguere tra i diversi stati del backup (**BackupStatus**).

Un altro elemento fondamentale è la struttura **AppState**, che raccoglie tutte le informazioni necessarie per gestire lo stato dell’applicazione.

La logica dell'app è gestita dalla struttura **MyApp**, che è responsabile per l'aggiornamento e la visualizzazione della GUI.

Contiene inoltre la definizione dei pannelli e dei modali presenti nell’interfaccia grafica.

exit_panel → renderizza pannello che comporta alla chiusura dell’applicazione, poiché si è verificato un comportamento anomalo.

main_panel →

- Contiene una **sidebar** (renderizzata con la funzione **render_sidebar**), che permette di navigare tra le tre sezioni dello user panel, ovvero: Backup, Analytics, Info. Mediante il pulsante “**Terminate**” è anche possibile interrompere l'intera applicazione: in questo caso è importante che il campo “**display**” del file **config_build.toml**, sia reimpostato a true.
- Contiene un pannello principale (renderizzato con la funzione **render_main_content**). In base alla selezione compiuta nella sidebar viene mostrata la seguente interfaccia:
 - **Backup** (PanelType::Backup) → dettaglio in **backup.rs**
 - **Analytics** (PanelType::Analytics) → dettaglio in **analytics.rs**
 - **Info** (PanelType::Info) → dettaglio in **info.rs**
- Sopra l'overlay, visualizza modale di errore mediante la funzione **render_error_modal** oppure renderizza modale di *avviso/successo* mediante la funzione **render_success_modal**

show_backup_window → Mostra il pannello di backup al posto del pannello principale quando c'è un backup in corso (**NOT BackupStatus::NotStarted**)

- è qui che viene trasmesso sul canale condiviso il messaggio “**resetWaiting**”, quando l'utente decide di annullare la conferma del riconoscimento della prima sequenza di tracciamento (**BackupStatus::ToConfirm**)
- Viene renderizzata una finestra apposita per ognuno dei diversi stati in cui si trova l'applicazione
- In caso di backup in progressione (**BackupStatus::InProgress**), viene renderizzata l'apposita barra di avanzamento mediante la funzione **render_backup_progress**
 - Da questa finestra è possibile trasmettere il messaggio “**stop**” nel canale condiviso con il thread di backup, qualora l'utente decidesse di annullare il backup

render_modal_exit → renderizza modale di conferma uscita dall'applicazione, sopra il pannello principale mostrato. Se l'utente decide di chiudere lo user panel allora **AppState.display** = false e la GUI viene interrotta (tuttavia lo stato rimane sempre salvato in **AppState**).

analytics.rs

Si occupa di mostrare l'interfaccia “**Analytics**”, mediante la funzione **show_analytics_panel**.

La funzione principale crea il pannello di analytics utilizzando l'interfaccia grafica fornita da egui. È suddivisa in tre sezioni principali:

- **Panoramica:** Mostra statistiche generali sull'uso della CPU e il numero totale di log di backup disponibili.
- **Uso della CPU nel tempo:** Visualizza l'uso della CPU in una tabella, mostrando gli ultimi log registrati. Sono presenti pulsanti per vedere più o meno log.
- **Log di Backup:** Visualizza i dettagli dei log di backup in una tabella, con pulsanti per vedere più o meno log.

Il codice utilizza due variabili globali per tracciare quanti log devono essere mostrati:

- static mut SHOWN_LOGS_CPU: usize = 5;
- static mut SHOWN_LOGS_BACKUP: usize = 5;

backup.rs

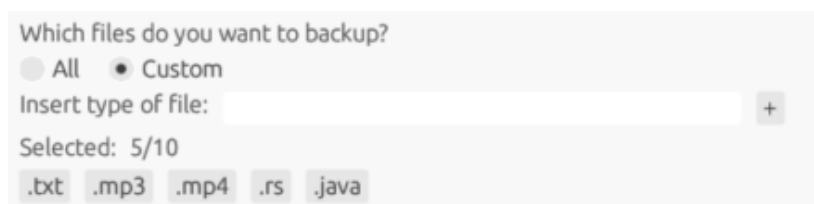
Si occupa di mostrare l'interfaccia “**Backup**”, mediante la funzione **show_backup_panel**

Contiene il pannello per la configurazione delle cartelle per il backup.

Sono presenti due bottoni che permettono di scegliere le cartelle: una di origine che rappresenta quindi la cartella che si vuole copiare, e una di destinazione in cui verranno salvati i file.

A seguire è presente una sezione in cui è possibile scegliere se copiare tutti i file della cartella o solo uno o più formati.

Nel caso si decidesse di salvare solamente determinati tipi di file, appare un apposito pannello per l'inserimento dell'estensione dei file. Una estensione scelta può sempre essere rimossa.



Infine, sono presenti due pulsanti:

- **Restore:** permette di tornare alla configurazione precedentemente salvata nel file **config_build.toml**
- **Save:** permette di salvare la configurazione appena inserita nel file **config_build.toml**. Dopo aver premuto il pulsante è possibile che un modale appaia in caso di errore, ad esempio non è possibile selezionare una cartella di origine vuota o uguale alla cartella di destinazione, oppure potrebbe comparire un modale di richiesta di conferma delle scelte effettuate, ad esempio nel caso la cartella di destinazione non fosse vuota. Il salvataggio della configurazione viene effettuato mediante la funzione: **save_folders**

Info.rs

Si occupa di mostrare l'interfaccia "Info", mediante la funzione **show_info_panel**

All'inizio del pannello sono presenti due checkbox:

- **Avvio Automatico:** permette di abilitare o disabilitare l'avvio automatico dell'applicazione all'avvio del sistema. Funzione principale: **toggle_auto_start**
- **GUI all'avvio:** permette di scegliere se all'avvio del sistema l'interfaccia utente verrà visualizzata. Il salvataggio di questa scelta si riflette nel campo "**display**" del file **config_build.toml**, mediante la funzione **update_config_file_display**

Successivamente sono presenti le informazioni per un corretto utilizzo dell'applicazione divise in tre sezioni: come attivare il backup, come configurare le cartelle e cosa contiene il pannello di statistiche.