

POLITECNICO DI TORINO

Course Project



Backup App in Rust

Authors:

Avantaggiato Vincenzo, Bianco Pasquale,
Genco Alessandro

July 2024

Table of Contents

1 Backup App	1
1.1 App description	1
1.2 Requirements	1
1.3 Non Functional Requirements	2
1.4 Screenshots	2
2 App Structure	4
2.1 project folder	4
2.2 src	5
2.3 bin	5
2.4 release	5
3 GUI and Sounds	6
3.1 GUI	6
3.2 Sounds	6
4 Events and Mouse Tracking	8
4.1 Events	8
4.2 Recognizing shapes	8
5 Backup	9
6 Uninstall	10
7 Cargo.toml	11
7.1 Crates Used	11
7.2 Conditional Dependency (Linux Only)	12
7.3 Build Dependency	12
7.4 Profile Customizations	12
7.5 Package Metadata	12

Chapter 1

Backup App

The application is a portable program able to be run on every OS, so Linux, Windows and MacOS. It provides itself executable files able to be run on the device without any installation, thanks to the structure of the project.

1.1 App description

The application provides a useful way to avoid data loss due to a non-working monitor, while the user is still able to move your mouse – so, the system is able to track it. When a critical situation happens (or if the user just wants to back-up some data), it is possible to virtually draw a simple gesture with the mouse—tracing a rectangle following the monitor edges—and then confirm with a swipe from left to right. At app startup, user can decide both *source* and *destination* folders to be copied and even which type of files, selected by their extension. The app will produce a copy of the current folder with the timestamp of the backup, to make user aware of it.

1.2 Requirements

What user can do by the app:

- User can choose which **source/destination** directories to use for copying data;
- User can choose which **type** of file to be copied, in order to be able to avoid large or unnecessary files;
- After the first execution, the app will run automatically at every **system startup**, so user does not have to worry about manually starting it at every

working session;

- The app runs in the **background**, ensuring it does not interfere with user's workspace.

1.3 Non Functional Requirements

Here what user should expect from the app:

- **Reliability:** The app is built to ensure a reliable behaviour even in critical situation: the only condition is to have a working mouse or track-pad and a turned on PC;
- **Usability:** The app is easy to use and control, thanks to a simple GUI and a simple gesture that can be reproduced easily in every situation, even blindly. Moreover, app provides acoustic notification to make user aware of the current state of the backup.
- **Security:** The app operates without any data leaks or malicious operations under the hood;
- **Availability:** The app will always be on, as long as user's PC is turned on;
- **Portability:** The app works on Windows, MacOS, and Linux.

1.4 Screenshots

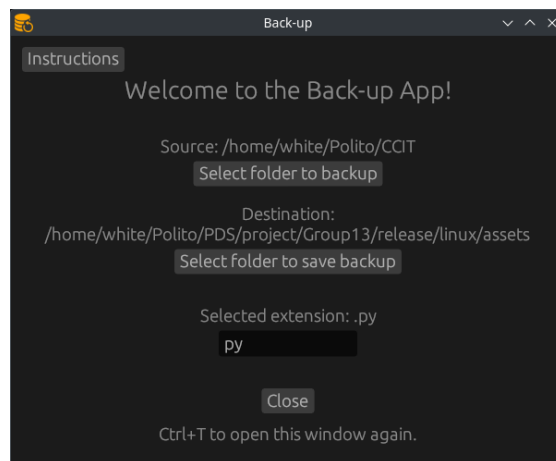


Figure 1.1: GUI displayed at application startup

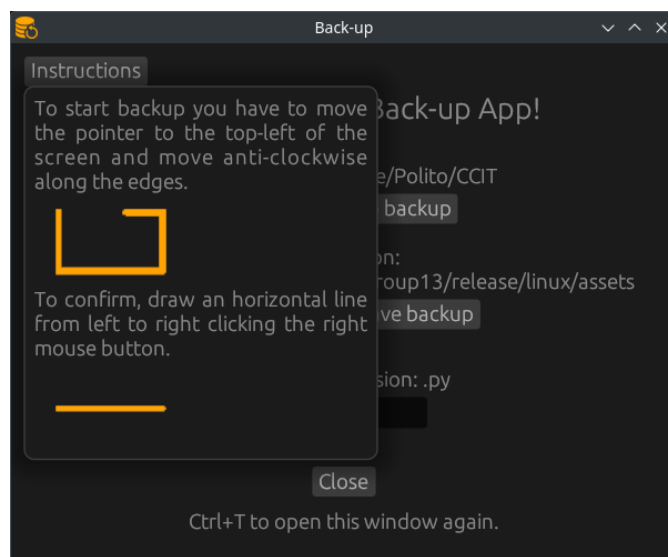


Figure 1.2: Instructions about how to use the application, directly in the GUI

Chapter 2

App Structure

The application is composed as a standard configuration already provided by the IDE, also because there was no need to make it more modular: it would have over-complicated it.

2.1 project folder

- *src*: it contains all source files;
- *assets*: contains all media that are managed in the app, such as .mp3 files, .gif and the .ico logo;
- *release*: contains executable files for each OS, each in their representative folder;
- **build.rs**: file containing information about how to build the project when using the command `cargo build --release --workspace`;
- **lin_build_release.sh**: bash file to build release in Linux
- **macos_build_release.sh**: bash file to build release in MacOS
- **win_build_release.bat**: bash file to build release in Windows
- **Cargo.lock**
- **Cargo.toml**

2.2 src

- *bin*: his folder contains all files we want it to create an executable from;
- **main.rs**: managed app entry point and avoid process duplication;
- **app.rs**, Sec. 3.1: GUI built in egui for the startup of the application;
- **config.rs**: it creates the config.json file containing user preferences for source, destination and extension of files to be copied;
- **events.rs**: module managing all events produced by the application and their outputs;
- **fs_copy.rs**: module to perform copy files when the backup is started;
- **lib.rs**: contains all modules and components used, in order to simplify coding and readability;
- **mouse_tracker.rs**: module controlling mouse tracking movement;
- **popup.rs**, Sec. 3.1: module for the popup: a GUI to be visualized when the user trace the rectangle, asking for confirmation;
- **sound.rs**, Sec. 3.2: module to control sound playing;

2.3 bin

- **spawn_gui.rs**: exe for spawning the gui;
- **spawn_popup.rs**: exe for spawning the popup;
- **uninstall.rs**: exe for uninstalling the app;
- **launcher_backup.rs**: only compiled in MacOS, used for a cleaner launching of the application;

2.4 release

- *linux*: contains all executables ans assets for Linux;
- *macos*: contains all executables ans assets for MacOS;
- *windows*: contains all executables ans assets for Windows.

Chapter 3

GUI and Sounds

3.1 GUI

Graphical User Interface is realized using the crate *egui*.

There are two main GUI windows:

- the **app**, which contains the logic to setup the backup and is spawned when running the app or when clicking the combination *Ctrl+T*. It lets the user select the folder to backup and the destination folder, using OS file system navigation. A text box lets the user write the extension to backup, if the field is empty all files are backed-up. User choices are serialized and saved in the file `assets/config.json` every time the user writes on the text box or selects a directory. Coherence between multiple instances of the same GUI is also implemented: when the window is focused, it reads again the content of the configuration file (in a normal program execution, it is however impossible to have multiple windows of this GUI opened).
- the **popup**, which spawns when the user traces the rectangle. It shows the options Yes or No, choices that are sent to the caller through a channel. If the user closes the window, the channel closes and the caller can interpret it as a No. This GUI cannot be minimized, maximized or resized. Moreover, it always spawns above other windows, in order to be always trackable by the user.

This two GUIs are spawned and managed in two separate processes.

3.2 Sounds

To reproduce sounds, the crate *rodio* (which requires crate *alsa* in Linux) is used.

All sounds are played in a separate thread, in order not to block the app. Moreover, we chose to use sounds that describe what's happening, to give a feedback to the user who can't see the screen.

Chapter 4

Events and Mouse Tracking

4.1 Events

Using crate rdev, both mouse and keyboard events are tracked. Mouse coordinates are generated and sent to the algorithm that recognizes shapes. Keys *Ctrl* and *T* are listened, in order to spawn the GUI window when both are clicked.

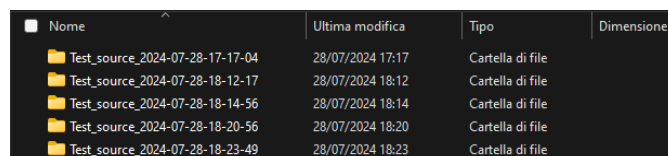
4.2 Recognizing shapes

Shape recognition and updating is performed using an ad-hoc algorithm that acts on two steps: Rectangle and Dash shape. For each shape, the algorithm class builds a `Vec<Rectangle>` composed of adjacent rectangles following the given shape, that is a rectangle at the object creation. Then, using a sliding window of three rectangles and a dynamically updated index for the vector, at each callback the caller has to call a function `track_mouse(p: Point) -> TrackingResult` able to update the object according to the given point. If the user leave the window, the vector is re-initialized and index is set to 0.

When the first shape is completed, the algorithm automatically initialize the vector for the dash shape, in order to be ready for the next gesture.

Chapter 5

Backup



Nome	Ultima modifica	Tipo	Dimensione
Test_source_2024-07-28-17-17-04	28/07/2024 17:17	Cartella di file	
Test_source_2024-07-28-18-12-17	28/07/2024 18:12	Cartella di file	
Test_source_2024-07-28-18-14-56	28/07/2024 18:14	Cartella di file	
Test_source_2024-07-28-18-20-56	28/07/2024 18:20	Cartella di file	
Test_source_2024-07-28-18-23-49	28/07/2024 18:23	Cartella di file	

Figure 5.1: Multiple backups of the same folder

Backup is realized through a recursive function that explores the directory tree and copies the file. To avoid overwriting existing files, we chose to create a folder inside the destination directory, which contains the timestamp of the backup in his name.

Backup can be very long, especially on an external disk, so we chose to run it in a separate thread, playing a descriptive notification sound when it is finished.

Chapter 6

Uninstall

The uninstall file is very simple. It does not remove the project files, but it removes the application from the startup list and closes all active processes of the app. This implementation choice allow non-expert users to have clean uninstaller, without removing all data, that are supposed to be of interest.

Chapter 7

Cargo.toml

7.1 Crates Used

- **egui**: A simple, fast, and highly portable immediate mode GUI library for Rust.
- **eframe**: A framework for building apps using **egui**.
 - **accesskit**: Adds screen reader compatibility.
 - **default_fonts**: Embeds the default fonts for **egui**.
 - **glow**: Uses the **glow** rendering backend.
- **winapi**: Provides Windows API bindings.
 - **winuser**: Windows User API bindings.
 - **wincon**: Windows Console API bindings.
- **rfd**: Cross platform library for using native file open/save dialogs.
- **log**: A lightweight logging facade for Rust.
- **tracing-subscriber**: Utilities for composing and collecting trace data.
- **futures**: Abstractions for asynchronous programming.
- **serde**: A framework for serializing and deserializing Rust data structures.
 - **derive**: Enables **derive** macros for serialization and deserialization.
- **serde_json**: A JSON serialization and deserialization library for Rust, built on **serde**.

- **image**: A library for manipulating images.
- **gif**: A library for encoding and decoding GIF files.
- **rdev**: Provides cross-platform hardware event handling.
- **chrono**: Date and time library for Rust.
- **auto-launch**: A library to manage the auto-launching of applications.
- **rodio**: A library for playing sound on multiple platforms.

7.2 Conditional Dependency (Linux Only)

- **alsa**: Provides bindings to the ALSA (Advanced Linux Sound Architecture) library.

7.3 Build Dependency

- **embed-resource**: A library to embed resources in a Rust binary.

7.4 Profile Customizations

- **Release Profile**:
 - **opt-level = 3**: Optimizes the code in release mode.
- **Development Profile**:
 - **opt-level = 2**: Optimizes all dependencies even in debug builds.

7.5 Package Metadata

- **no-console**: Disables the console window in Windows builds.