QuickCapture-egui

Francesco Paolo Carmone s308126
Daniele De Rossi s314796

codice sorgente

Funzionalità

Multiplatform Support

Easily accessible user interface (UI)

Selection Options

Hotkey Support

Output Format

Opzionali

Annotation Tools

Delay Timer

Save Options

Multi-monitor Support

Crates

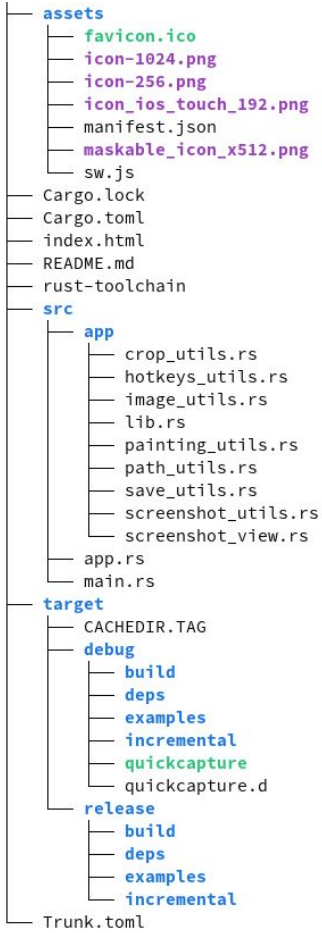
egui = "0.22.0"
egui-extras = "0.22.0"
egui-toast = "0.8.1"
egui-modal = "=0.2.4"
egui::epaint = "0.22.0"

arboard = "3.3.0"

screenshots = "0.6.0"
image = "0.24"
imageproc = "0.23.0"

Struttura cartelle

Il template iniziale prevedeva la presenza di alcune cartelle e files, tra questi: assets, src/, main.rs, Cargo.toml, Trunk.toml, imponendo una rigida struttura del codice e delle cartelle



app.rs

Contiene

- struct QuickCaptureApp
  - screenshot_image_buffer
  - painting
  - clipboard
  - toasts
- L'implementazione dei metodi *_view

```rust
pub struct QuickCaptureApp {
    pub view: Views,
    screenshot_image_buffer: Option<RgbaImage>, // The screenshot data
    screenshot_type: Option<ScreenshotType>,
    painting: Option<painting_utils::Painting>, // UI and methods to draw o
    painted_screenshot: Option<egui::TextureHandle>, // egui wants Textureh
    pub save_path: SavePath,
    screenshot_view: screenshot_view::ScreenshotView,
    update_counter: u8,      // Serve per chiamare _frame.set_visible(). Una
    keyboard_shortcuts: hotkeys_utils::AllKeyboardShortcuts,
    clipboard: Option<Clipboard>,
    toasts: Toasts,
    which_shortcut_field: String,
    modifier: Modifiers,
    key_var: String,
}
```

main.rs

Contiene

- fn update() permette l'aggiornamento del contenuto nella finestra

view è un pub `enum` `Views` di app.rs

```rust
// Called each time the UI needs repainting, which may be many times per second.
fn update(&mut self, ctx: &egui::Context, _frame: &mut eframe::Frame) {

    match self.view {
        Views::Home => {
            _frame.set_visible(true);
            _frame.set_decorations(decorated: true);
            self.home_view(ctx, _frame: _frame);
        },
        Views::Screenshot => {
            self.screenshot_view(ctx, _frame: _frame);
        },
        Views::Settings => {
            self.settings_view(ctx, _frame: _frame)
        },
        Views::Save => {
            self.save_view(ctx, _frame: _frame);
        },
    }

}
```

Files src/app/*_utils.rs

I files _utils.rs contengono l'implementazione delle funzionalità dell'applicazione, si fa riferimento specialmente a:

- (Una o più) strutture dati
- Funzioni

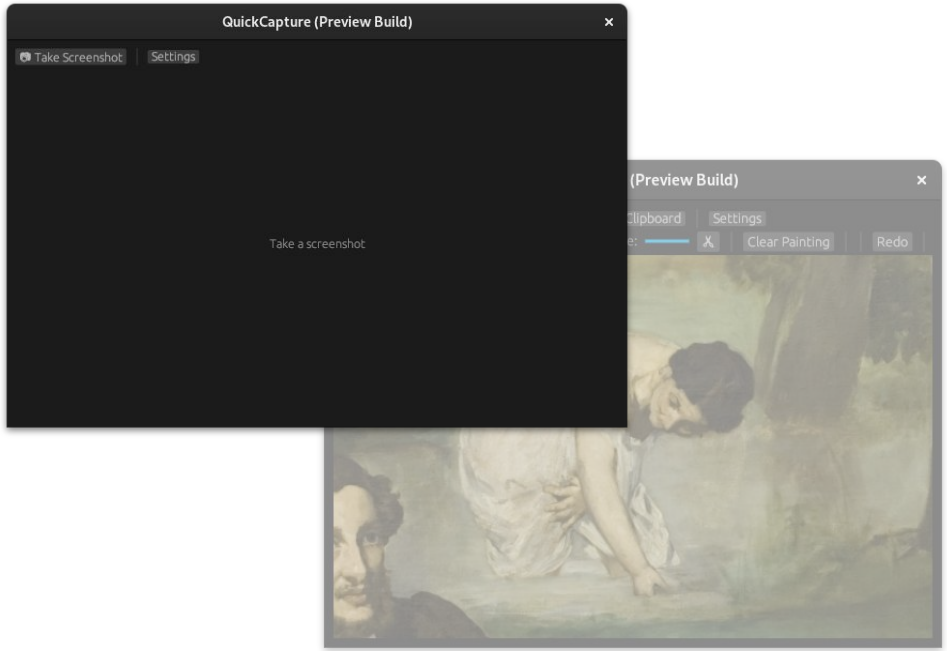
save_utils.rs

```rust
use image::ImageBuffer;
use crate::app::ImgFormats;
use std::path::PathBuf;
use chrono::{DateTime, Local};

#[derive(Debug, Clone, PartialEq, Eq)]
5 implementations
pub struct SavePath{
    pub path: PathBuf,
    pub name: String,
    pub format: ImgFormats,
    pub user_mod_name: bool,
}

impl SavePath {
    pub fn new(path: PathBuf, format: ImgFormats) -> Self {
        let date: DateTime<Local> = Local::now();
        let formatted: DelayedFormat<StrftimeItems<'_>> = date.format(fmt: "%Y-%m-%dT%H:%M:%S");
        let name: String = formatted.to_string();
        Self {
            path,
            format,
            name,
            user_mod_name: false,
        }
    }

}

pub fn save_image(save_path: &SavePath, picture: ImageBuffer<image::Rgba<u8>, Vec<u8>>){…

pub fn generate_filename() -> String {…

pub fn check_filename(name: &str) -> bool {…
```

View: Home (I)

Permette l’accesso alle funzionalità

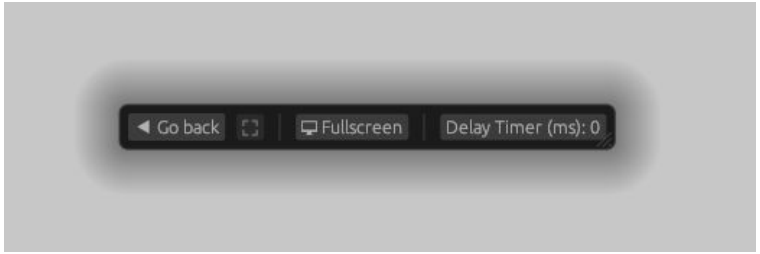
- Catturare uno screenshot
- Disegno nell’immagine
- Salvataggio
- Copia nella clipboard



View: Screenshot

Permette l’accesso alle funzionalità

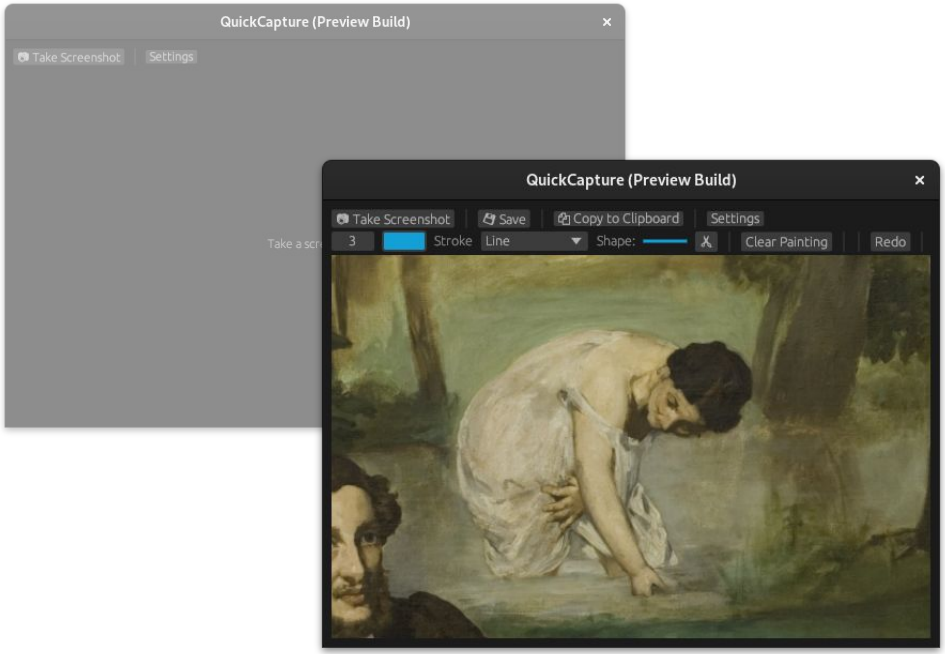
- Screenshot parziale (multischermo)
- Screenshot totale (multischermo)
- Ritardo



View: Home (II)

Permette l’accesso alle funzionalità

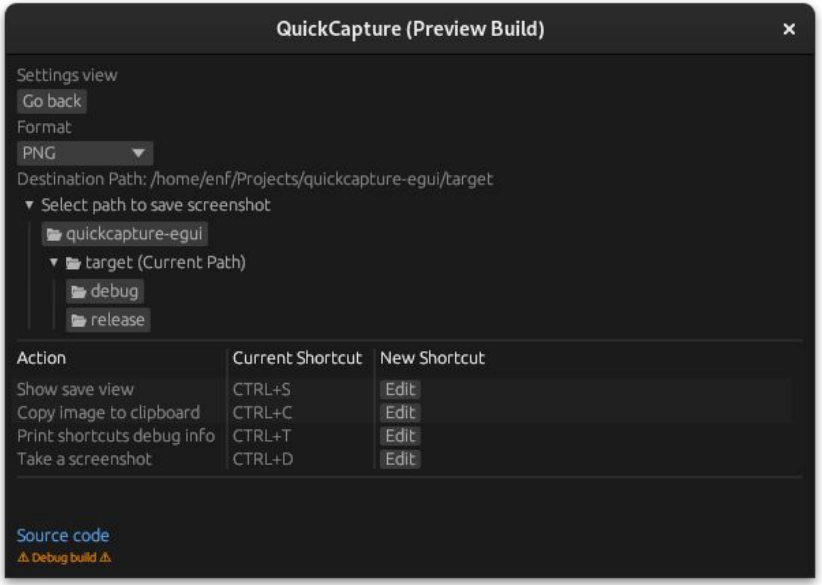
- Catturare uno screenshot
- Disegno nell’immagine
- Salvataggio
- Copia nella clipboard



View: Settings (I)

Permette l’accesso alle funzionalità

- Visualizza e modifica scorciatoie
- Scelta percorso di default
- Scelta formato di default

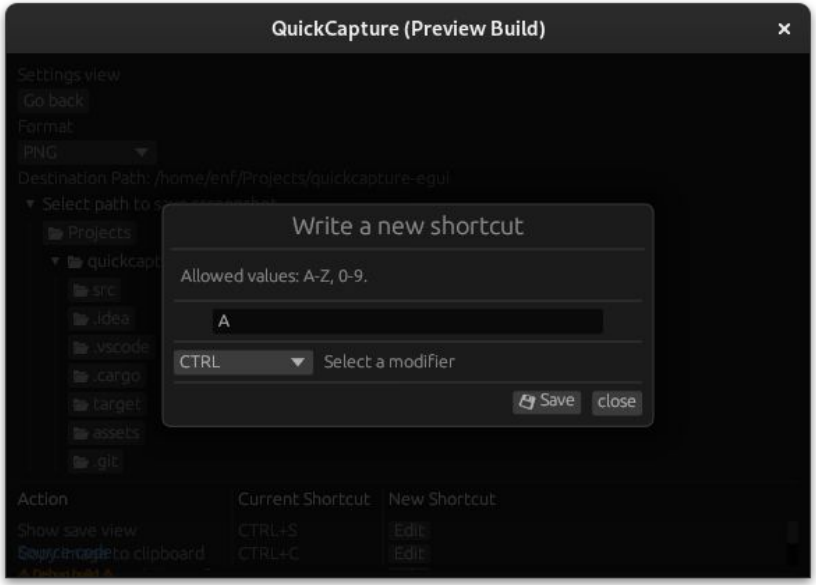


View: Settings (II)

Permette l’accesso alle funzionalità

- Visualizza e modifica scorciatoie
- Scelta percorso di default
- Scelta formato di default

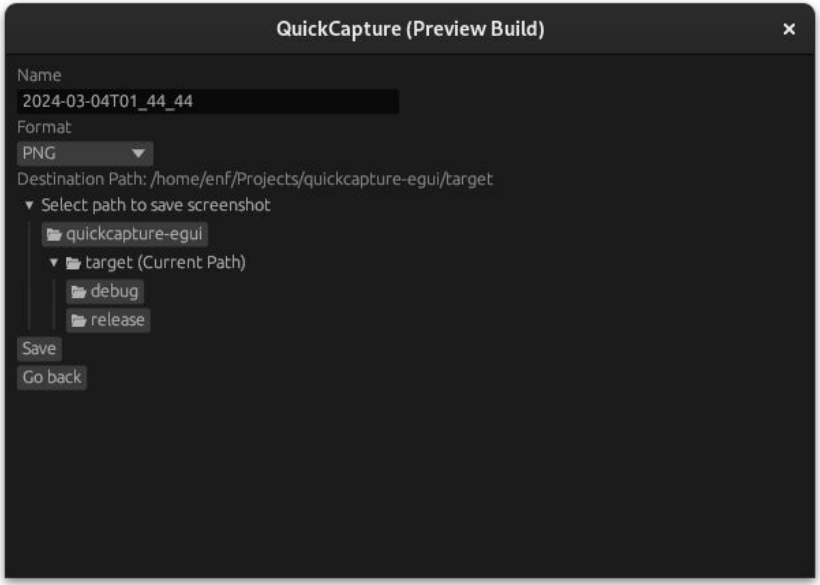
L’esito è notificato tramite toast



View: Save

Permette l’accesso alle funzionalità

- Scelta percorso di salvataggio
- Scelta formato di salvataggio



Ritaglio

È inoltre possibile fare aggiustamenti dell'area selezionata dallo screenshot tramite l'apposito strumento di ritaglio.

Il codice del ritaglio è implementato in crop_utils.rs

Disegno

È possibile editare lo screenshot appena catturato tramite due strumenti:

- Disegno a mano libera
- Linee rette

È possibile selezionare la shape richiesta tra le due implementate con un enum chiamato DrawingShape. Ogni annotazione è salvata in una struct chiamata DrawObj, al cui interno troviamo un Vec<Pos2>, ovvero la serie di punti coperti dall'annotazione appena inserita, e un egui::Stroke, che contiene informazioni riguardanti lo spessore della linea e il colore selezionato.

```
enum DrawingShape {
    Line,
    StraightLine,
}
```

```
3 implementations
pub struct DrawObj {
    points: Vec<Pos2>,
    stroke: egui::Stroke,
}

impl DrawObj {
    fn new(points: Vec<Pos2>, stroke: egui::Stroke) -> Self {
        Self {
            points: points,
            stroke: stroke,
        }
    }
}
```

Gestione percorsi

- La gestione percorsi, per selezionare un percorso di salvataggio dello screenshot, è implementata in path_utils.rs
- Al suo interno troviamo la funzione ui che aggiunge la vista ad albero del file system
- Questo aggiornerà il path di salvataggio all'interno della struct SavePath, definita all'interno di save_utils.rs e facente parte a sua volta della struct QuickCaptureApp, definita in app.rs

path_utils.rs

```rust
use crate::app::save_utils::check_filename;
use crate::app::save_utils::SavePath;
use crate::app::ImgFormats;
use egui::{CollapsingHeader, Color32, ComboBox, ScrollArea, Ui};
use std::fs;

pub fn ui(ui: &mut Ui, path: &mut SavePath) {
    ui.label(text: "Name");
    let response: Response = ui.text_edit_singleline(text: &mut path.name);
    if !check_filename(&path.name) {
        ui.colored_label(
            color: Color32::LIGHT_RED,
            text: "Filename is not valid! Forbidden characters: \\ / : * ? \" < > |",
        );
    }

    if response.lost_focus() {
        println!("Name: {}", path.name);
        path.user_mod_name = true;
    }
    ui.end_row();
    ui.label(text: "Format");
    ComboBox::from_label("") ComboBox
        .selected_text(format!("{:?}", path.format)) ComboBox
        .show_ui(ui, menu_contents: |ui: &mut Ui| {
            ui.style_mut().wrap = Some(false);
            ui.set_min_width(60.0);
            ui.selectable_value(current_value: &mut path.format, selected_value: ImgFormats::PNG, text:
            ui.selectable_value(current_value: &mut path.format, selected_value: ImgFormats::JPEG, text
            ui.selectable_value(current_value: &mut path.format, selected_value: ImgFormats::GIF, text:
```

save_utils.rs

```rust
use image::ImageBuffer;
use crate::app::ImgFormats;
use std::path::PathBuf;
use chrono::{DateTime, Local};

#[derive(Debug, Clone, PartialEq, Eq)]
5 implementations
pub struct SavePath{
    pub path: PathBuf,
    pub name: String,
    pub format: ImgFormats,
    pub user_mod_name: bool,
}

impl SavePath {
    pub fn new(path: PathBuf, format: ImgFormats) -> Self {
        let date: DateTime<Local> = Local::now();
        let formatted: DelayedFormat<StrftimeItems<'_>> = date.format(fmt: "%Y-%m-%dT%H:%M:%S");
        let name: String = formatted.to_string();
        Self {
            path,
            format,
            name,
            user_mod_name: false,
        }
    }

}

pub fn save_image(save_path: &SavePath, picture: ImageBuffer<image::Rgba<u8>, Vec<u8>>){
```

Salvataggio

Codice scritto in save_utils.rs, richiamato in app.rs

- La struct SavePath contiene le variabili per la definizione del percorso, il nome e il formato del file
- La funzione save_image, richiamata opportunamente in app.rs, è un wrapper di image::save_buffer che prende come argomento image::ImageBuffer
- Funzioni di corredo funzionamento della funzionalità
- Il nome del file di default è determinato dal timestamp della cattura

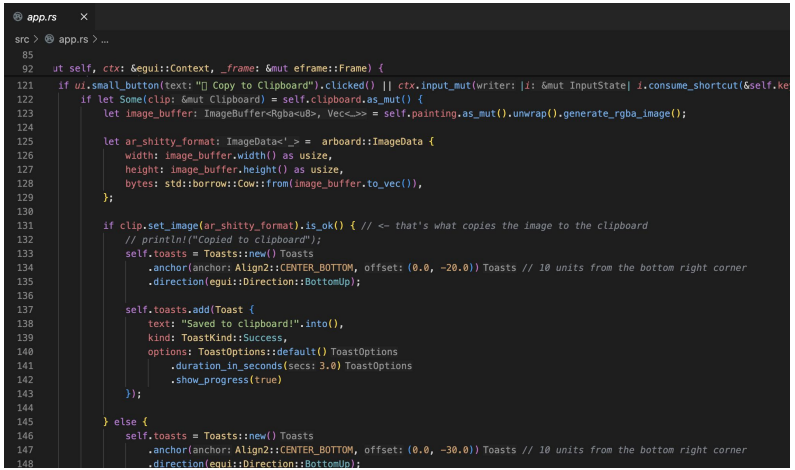
save_utils.rs

src > app > save_utils.rs > ...

```rust
use image::ImageBuffer;
use crate::app::ImgFormats;
use std::path::PathBuf;
use chrono::{DateTime, Local};

#[derive(Debug, Clone, PartialEq, Eq)]
5 implementations
pub struct SavePath{
    pub path: PathBuf,
    pub name: String,
    pub format: ImgFormats,
    pub user_mod_name: bool,
}

impl SavePath {
    pub fn new(path: PathBuf, format: ImgFormats) -> Self {
        let date: DateTime<Local> = Local::now();
        let formatted: DelayedFormat<StrftimeItems<'_>> = date.format(fmt: "%Y-%m-%dT%H:%M:%S");
        let name: String = formatted.to_string();
        Self {
            path,
            format,
            name,
            user_mod_name: false,
        }
    }

}

```

Copia nella clipboard

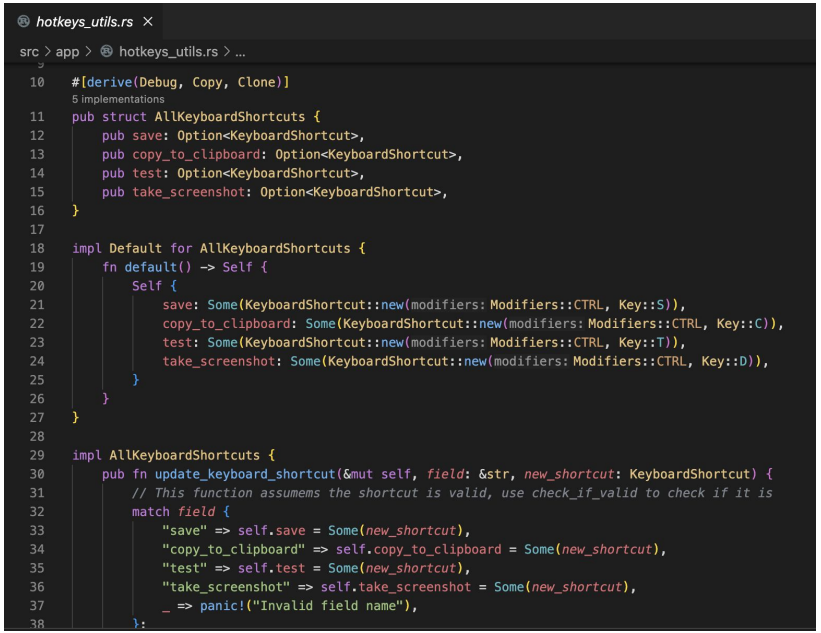
- Il crate arbord fornisce l’interfaccia alla clipboard di sistema
- L’immagine con i disegni è clonata e convertita in arboard::ImageData, lasciandola inalterata
- Un toast confermerà l’avvenuto successo o fallimento dell’opzione



Hotkeys

Codice scritto in hotkeys_utils.rs, e richiamato in app.rs

- egui fornisce l'implementazione dei tipi Key, KeyboardShortcut e Modifier
- struct AllKeyboardShortcut contiene tutte le funzionalità richiamabili tramite scorciatoie
- Funzioni di corredo funzionamento della funzionalità



Fine

:)