



QuickCapture-egui

Francesco Paolo Carmone
Daniele De Rossi

s308126
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

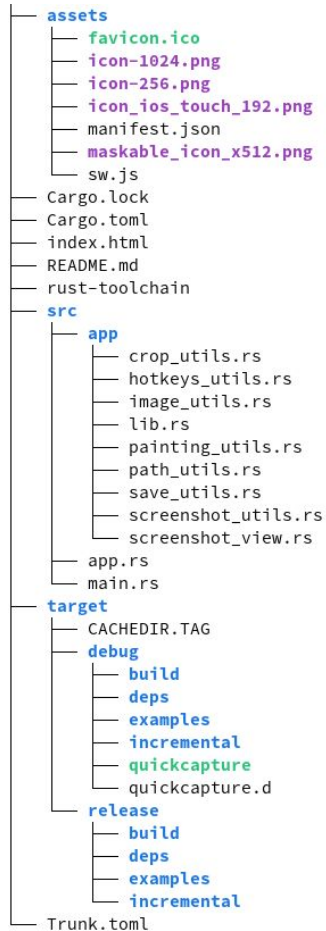
```
egui = "0.22.0"  
egui-extras = "0.22.0"  
egui-toast = "0.8.1"  
egui-modal = "=0.2.4"  
egui::epaint = "0.22.0"
```

```
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

```

41 pub struct QuickCaptureApp {
42     pub view: Views,
43     screenshot_image_buffer: Option<RgbImage>, // The screenshot data
44     screenshot_type: Option<ScreenshotType>,
45     painting: Option<painting_utils::Painting>, // UI and methods to draw o
46     painted_screenshot: Option<egui::TextureHandle>, // egui wants TextureH
47     pub save_path: SavePath,
48     screenshot_view: ScreenshotView::ScreenshotView,
49     update_counter: u8, // Serve per chiamare _frame.set_visible(). Una
50     keyboard_shortcuts: hotkeys_utils::AllKeyboardShortcuts,
51     clipboard: Option<Clipboard>,
52     toasts: Toasts,
53     which_shortcut_field: String,
54     modifier: Modifiers,
55     key_var: String,
56 }

```



main.rs

Contiene

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

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

```
30 // Called each time the UI needs repainting, which may be many times per second.
31 fn update(&mut self, ctx: &egui::Context, _frame: &mut eframe::Frame) {
32
33     match self.view {
34         Views::Home => {
35             _frame.set_visible(true);
36             _frame.set_decorations(decorated: true);
37             self.home_view(ctx, _frame: _frame);
38         },
39         Views::Screenshot => {
40             self.screenshot_view(ctx, _frame: _frame);
41         },
42         Views::Settings => {
43             self.settings_view(ctx, _frame: _frame);
44         },
45         Views::Save => {
46             self.save_view(ctx, _frame: _frame);
47         },
48     }
49
50 }
```

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

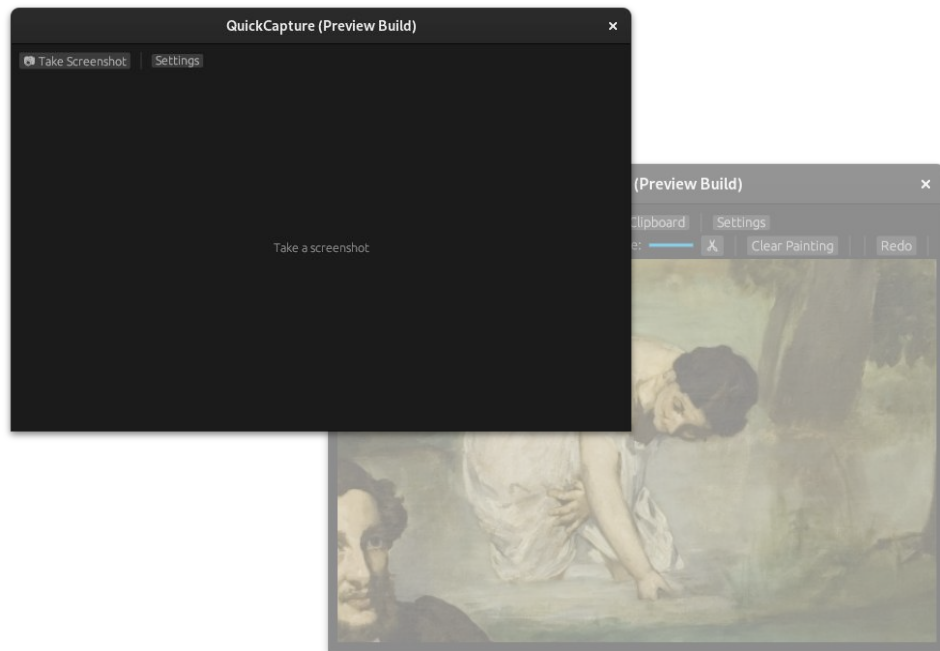
save_utils.rs

```
1 use image::ImageBuffer;
2 use crate::app::ImgFormats;
3 use std::path::PathBuf;
4 use chrono::{DateTime, Local};
5
6 #[derive(Debug, Clone, PartialEq, Eq)]
7 5 implementations
8 pub struct SavePath{
9     pub path: PathBuf,
10    pub name: String,
11    pub format: ImgFormats,
12    pub user_mod_name: bool,
13 }
14
15 impl SavePath {
16     pub fn new(path: PathBuf, format: ImgFormats) -> Self {
17         let date: DateTime<Local> = Local::now();
18         let formatted: DelayedFormat<StrftimeItems<'_>> = date.format(fmt: "%Y-%m-%dT%H:%M:%S");
19         let name: String = formatted.to_string();
20         Self {
21             path,
22             format,
23             name,
24             user_mod_name: false,
25         }
26     }
27 }
28
29 > pub fn save_image(save_path: &SavePath, picture: ImageBuffer<image::Rgba<u8>, Vec<u8>>){...
30
31 > pub fn generate_filename() -> String {...
32
33 > pub fn check_filename(name: &str) -> bool {...
34
35 >
36
37 >
38
39 >
40
41 >
42
43 >
44
45 >
46
47 >
48
49 >
50
51 >
52
53 >
54
55 >
56
57 >
58
59 >
60
61 >
62
63 >
64
65 >
66 |
```

View: Home (I)

Permette l'accesso alle funzionalità

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

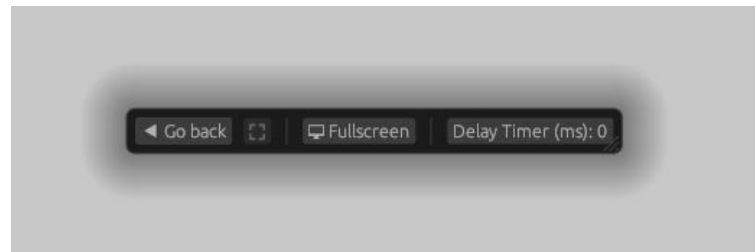




View: Screenshot

Permette l'accesso alle funzionalità

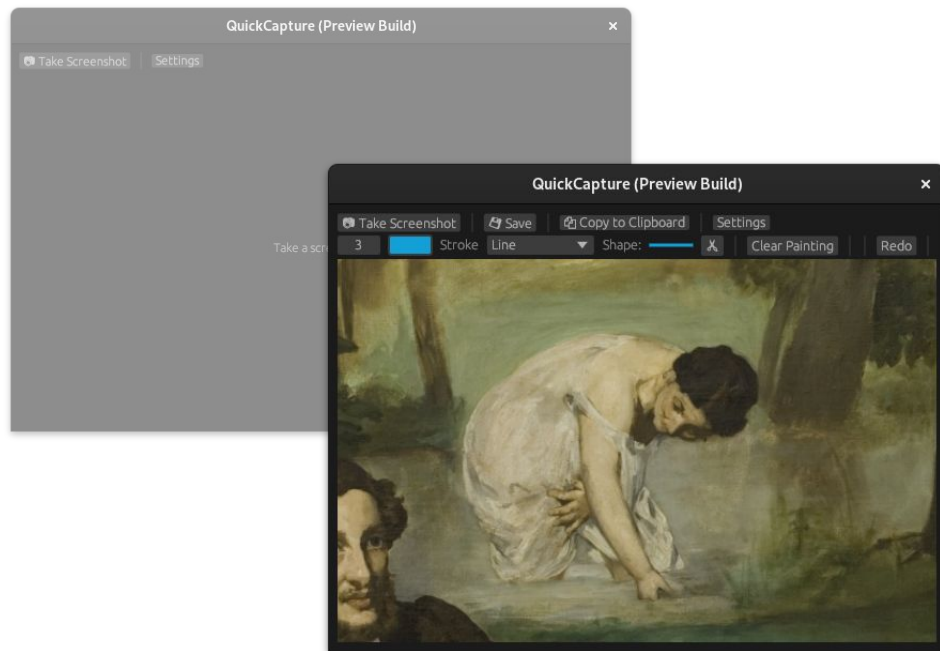
- Screenshot parziale (multischermo)
- Screenshot totale (multischermo)
- Ritardo



View: Home (II)

Permette l'accesso alle funzionalità

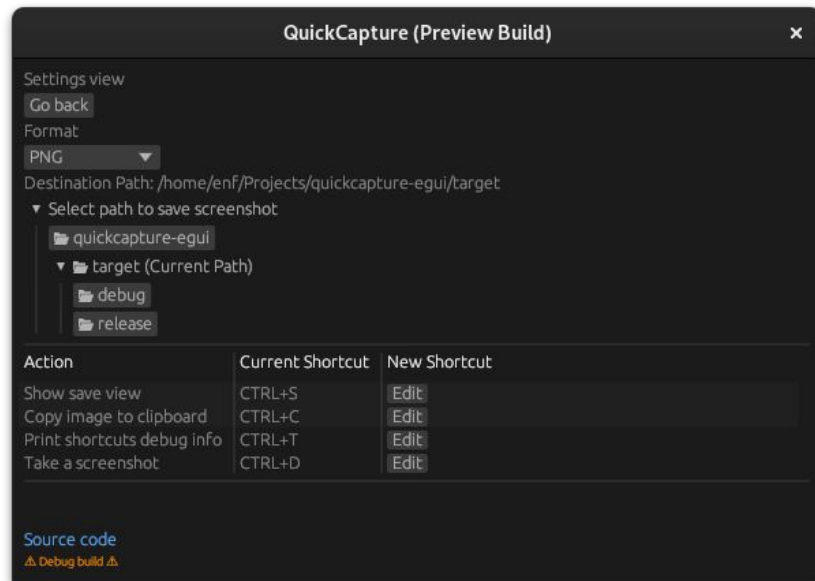
- Catturare uno screenshot
- Disegno nell'immagine
- Salvataggio
- Copia nella clipboard



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

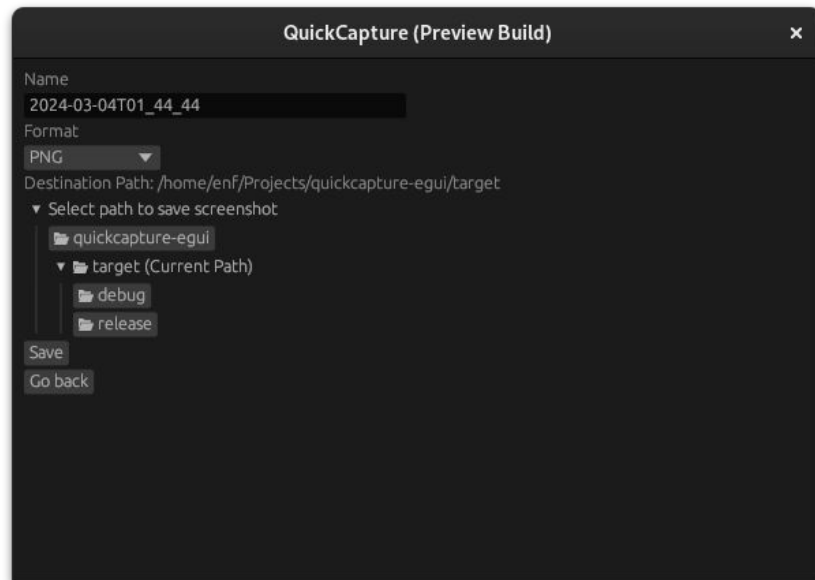




View: Save

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`

```
crop_utils.rs x
src > app > crop_utils.rs > ...
1 use egui::*;
2 use egui::epaint::RectShape;
3
4 #[derive(Clone)]
5 1 implementation
6 enum Side {
7     None,
8     Left,
9     Right,
10    Up,
11    Down,
12    Center
13 }
14
15 #[derive(Clone)]
16 3 implementations
17 pub struct Crop {
18     cut_rect: Rect,
19     scaled_rect: Rect,
20     pub offset_x_right: f32,
21     pub offset_x_left: f32,
22     pub offset_y_up: f32,
23     pub offset_y_down: f32,
24     last_click: Pos2,
25     limit_reached: bool,
26     side: Side,
27 }
```



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.

```
3 implementations
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 X
src > app > @ path_utils.rs > ...
1 use crate::app::save_utils::check_filename;
2 use crate::app::save_utils::SavePath;
3 use crate::app::ImgFormats;
4 use egui::CollapsingHeader, Color32, ComboBox, ScrollArea, Ui;
5 use std::fs;
6
7 pub fn ui(ui: &mut Ui, path: &mut SavePath) {
8     ui.label(text: "Name");
9     let response = ui.text_edit_singleline(text: &mut path.name);
10    if !check_filename(&path.name) {
11        ui.colored_label(
12            color: Color32::LIGHT_RED,
13            text: "Filename is not valid! Forbidden characters: \\ / : * ? \" < > | ",
14        );
15    }
16
17    if response.lost_focus() {
18        println!("Name: {}", path.name);
19        path.user_mod_name = true;
20    }
21    ui.end_row();
22    ui.label(text: "Format");
23    ComboBox::from_label("") ComboBox
24        .selected_text(format!("{}", path.format)) ComboBox
25        .show_ui(ui, menu_contents: |ui: &mut Ui| {
26            ui.style_mut().wrap = Some(false);
27            ui.set_min_width(60.0);
28            ui.selectable_value(current_value: &mut path.format, selected_value: ImgFormats::PNG, text:
29            ui.selectable_value(current_value: &mut path.format, selected_value: ImgFormats::JPEG, text:
30            ui.selectable_value(current_value: &mut path.format, selected_value: ImgFormats::GIF, text:
```

```
@ save_utils.rs X
src > app > @ save_utils.rs > ...
1 use Image::ImageBuffer;
2 use crate::app::ImgFormats;
3 use std::path::PathBuf;
4 use chrono::(DateTime, Local);
5
6 #[derive(Debug, Clone, PartialEq, Eq)]
7 5 implementations
8 pub struct SavePath {
9     pub path: PathBuf,
10    pub name: String,
11    pub format: ImgFormats,
12    pub user_mod_name: bool,
13 }
14
15 impl SavePath {
16     pub fn new(path: PathBuf, format: ImgFormats) -> Self {
17         let date: DateTime<Local> = Local::now();
18         let formatted: DelayedFormat<StrftimeItem> = date.format(fmt: "%Y-%m-%dT%H:%M:%S");
19         let name: String = formatted.to_string();
20         Self {
21             path,
22             format,
23             name,
24             user_mod_name: false,
25         }
26     }
27 }
28
29 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 X
src > app > save_utils.rs > ...
1 use image::ImageBuffer;
2 use crate::app::ImgFormats;
3 use std::path::PathBuf;
4 use chrono::{DateTime, Local};
5
6 #[derive(Debug, Clone, PartialEq, Eq)]
7 pub struct SavePath {
8     pub path: PathBuf,
9     pub name: String,
10    pub format: ImgFormats,
11    pub user_mod_name: bool,
12 }
13
14 impl SavePath {
15     pub fn new(path: PathBuf, format: ImgFormats) -> Self {
16         let date: DateTime<Local> = Local::now();
17         let formatted: DelayedFormat<StrftimeItems<'_>> = date.format(fmt: "%Y-%m-%dT%H:%M:%S");
18         let name: String = formatted.to_string();
19         Self {
20             path,
21             format,
22             name,
23             user_mod_name: false,
24         }
25     }
26 }
27
28 }
```

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

```

@ app.rs x
src > @ app.rs > ...
85
92 ut self, ctx: &gui::Context, _frame: &mut eframe::Frame {
121
122 if ui.small_button(text: "[ Copy to Clipboard ").clicked() { | | ctx.input_mut(writer: | | &mut InputState| |.consume_shortcut(&self.ke
123 if let Some(clip: &mut Clipboard) = self.clipboard.as_mut() {
124     let image_buffer: ImageBuffer<Rgba<u8>, Vec<_>> = self.painting.as_mut().unwrap().generate_rgba_image();
125
126     let ar_shitty_format: ImageData<'_> = arboard::ImageData {
127         width: image_buffer.width() as usize,
128         height: image_buffer.height() as usize,
129         bytes: std::borrow::Cow::from(image_buffer.to_vec()),
130     };
131
132 if clip.set_image(ar_shitty_format).is_ok() { // <- that's what copies the image to the clipboard
133     // println!("Copied to clipboard");
134     self.toasts = Toasts::new() Toasts
135         .anchor(anchor: Align2::CENTER_BOTTOM, offset: (0.0, -20.0)) Toasts // 10 units from the bottom right corner
136         .direction(egui::Direction::BottomUp);
137
138     self.toasts.add(Toast {
139         text: "Saved to clipboard!".into(),
140         kind: ToastKind::Success,
141         options: ToastOptions::default() ToastOptions
142             .duration_in_seconds(secs: 3.0) ToastOptions
143             .show_progress(true)
144     });
145 } else {
146     self.toasts = Toasts::new() Toasts
147         .anchor(anchor: Align2::CENTER_BOTTOM, offset: (0.0, -30.0)) Toasts // 10 units from the bottom right corner
148         .direction(egui::Direction::BottomUp);

```

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 controllo funzionamento della funzionalità

```
hotkeys_utils.rs X
src > app > hotkeys_utils.rs > ...

10 #[derive(Debug, Copy, Clone)]
11 5 implementations
12 pub struct AllKeyboardShortcuts {
13     pub save: Option<KeyboardShortcut>,
14     pub copy_to_clipboard: Option<KeyboardShortcut>,
15     pub test: Option<KeyboardShortcut>,
16     pub take_screenshot: Option<KeyboardShortcut>,
17 }
18
19 impl Default for AllKeyboardShortcuts {
20     fn default() -> Self {
21         Self {
22             save: Some(KeyboardShortcut::new(modifiers: Modifiers::CTRL, Key::S)),
23             copy_to_clipboard: Some(KeyboardShortcut::new(modifiers: Modifiers::CTRL, Key::C)),
24             test: Some(KeyboardShortcut::new(modifiers: Modifiers::CTRL, Key::T)),
25             take_screenshot: Some(KeyboardShortcut::new(modifiers: Modifiers::CTRL, Key::D)),
26         }
27     }
28 }
29
30 impl AllKeyboardShortcuts {
31     pub fn update_keyboard_shortcut(&mut self, field: &str, new_shortcut: KeyboardShortcut) {
32         // This function assumes the shortcut is valid, use check_if_valid to check if it is
33         match field {
34             "save" => self.save = Some(new_shortcut),
35             "copy_to_clipboard" => self.copy_to_clipboard = Some(new_shortcut),
36             "test" => self.test = Some(new_shortcut),
37             "take_screenshot" => self.take_screenshot = Some(new_shortcut),
38             _ => panic!("Invalid field name"),
39         }
40     }
41 }
```



Fine

:)