Stock Tracker con Rust e Macroquad

Guida Completa dal Codice Base alla Versione Finale

Creato il 07/02/2026

Indice

1. Introduzione

Questo documento descrive lo sviluppo progressivo di un'applicazione di tracking finanziario realizzata con Rust e Macroquad. Il progetto permette di monitorare in tempo reale l'andamento di titoli azionari, ETF e criptovalute, visualizzando grafici aggiornati ogni minuto.

Obiettivi del Progetto

• Creare un'interfaccia grafica con Macroquad
• Recuperare dati finanziari in tempo reale da Yahoo Finance
• Visualizzare grafici aggiornati automaticamente
• Permettere la selezione multipla di titoli
• Supportare stock, ETF e criptovalute

Tecnologie Utilizzate

• **Rust**: Linguaggio di programmazione principale
• **Macroquad**: Framework per grafica e interfaccia
• **reqwest**: Client HTTP per chiamate API
• **chrono**: Gestione date e timestamp
• **Yahoo Finance API**: Fonte dati finanziari gratuita

2. Versione 1: Applicazione Base

La prima versione implementa le funzionalità base: una finestra Macroquad che mostra un singolo grafico statico. Questo stabilisce la struttura fondamentale del progetto.

Cargo.toml

```
[package]
name = "stock_tracker"
version = "0.1.0"
edition = "2021"

[dependencies]
macroquad = "0.4"
```

Codice Principale (main.rs)

```
use macroquad::prelude::*;

#[macroquad::main("Stock Tracker")]
async fn main() {
    loop {
        clear_background(BLACK);

        // Disegna un grafico semplice
        draw_text("Stock Tracker v1", 20.0, 40.0, 30.0, WHITE);
        draw_line(100.0, 200.0, 700.0, 200.0, 2.0, WHITE);
        draw_line(100.0, 400.0, 700.0, 150.0, 3.0, GREEN);

        next_frame().await
    }
}
```

Caratteristiche Versione 1

• Finestra grafica base con Macroquad
• Grafico statico disegnato con linee
• Nessun dato reale, solo visualizzazione di test
• Ciclo di rendering principale (game loop)

3. Versione 2: Recupero Dati Reali

Aggiungiamo la capacità di recuperare dati reali da Yahoo Finance e memorizzarli in strutture dati appropriate. Introduciamo anche le strutture StockData per organizzare le informazioni.

Cargo.toml Aggiornato

```
[dependencies]
macroquad = "0.4"
reqwest = { version = "0.11", features = ["json"] }
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"
chrono = "0.4"
tokio = { version = "1", features = ["full"] }
```

Struttura Dati

```
#[derive(Debug, Clone)]
struct StockData {
    symbol: String,
    prices: Vec<f32>,
    timestamps: Vec<DateTime<Utc>>,
    current_price: f32,
    change_percent: f32,
}
```

Funzione Fetch

```
async fn fetch_stock_data(symbol: &str)
    -> Result<(f32, f32), Box<dyn std::error::Error>> {
    let url = format!(
        "https://query1.finance.yahoo.com/v8/finance/chart/{}",
        symbol
    );

    let response = reqwest::get(&url).await?;
    let json: serde_json::Value = response.json().await?;

    // Estrai prezzo e variazione percentuale
    let current_price = json["chart"]["result"][0]["meta"]
        ["regularMarketPrice"].as_f64().unwrap_or(0.0) as f32;

    Ok((current_price, change_percent))
}
```

• Chiamate HTTP asincrone con reqwest
• Parsing JSON dei dati Yahoo Finance
• Struttura dati organizzata per stock
• Gestione errori con Result

4. Versione 3: Threading e Aggiornamenti

Problema critico: Macroquad non usa Tokio runtime, causando panic quando tentiamo chiamate async. Soluzione: usare thread separati con reqwest::blocking e Arc> per condividere i dati.

Problema Risolto

Errore Precedente:
"there is no reactor running, must be called from the context of a Tokio runtime"

Soluzione:
Usare reqwest::blocking e thread::spawn invece di async/await

Implementazione Threading

```
use std::sync::{Arc, Mutex};
use std::thread;
use std::time::Duration;

struct App {
    stocks: Arc<Mutex<HashMap<String, StockData>>>,
    last_update: Arc<Mutex<DateTime<Utc>>>,
}

fn start_update_worker(
    stocks: Arc<Mutex<HashMap<String, StockData>>>,
    symbols: Vec<String>,
) {
    thread::spawn(move || {
        loop {
            thread::sleep(Duration::from_secs(60));

            for symbol in &symbols {
                if let Ok((price, change)) = fetch_stock_data(symbol) {
                    if let Ok(mut stocks_lock) = stocks.lock() {
                        if let Some(stock) = stocks_lock.get_mut(symbol) {
                            stock.prices.push(price);
                            stock.current_price = price;
                        }
                    }
                }
            }
        }
    });
}
```

• Thread separato per aggiornamenti periodici
• Arc> per sincronizzazione thread-safe
• Aggiornamento automatico ogni 60 secondi
• reqwest::blocking invece di async
• Nessun panic al runtime

5. Versione 4: Interfaccia a Due Pannelli

Ridisegniamo completamente l'interfaccia dividendola in due sezioni: un pannello lista a sinistra per selezionare i titoli, e un pannello grafico dettagliato a destra per visualizzare il titolo selezionato.

Layout Interfaccia

Pannello Sinistro (300px):
• Lista verticale di tutti i titoli disponibili
• Mostra simbolo, prezzo corrente, variazione %
• Evidenziazione del titolo selezionato
• Effetto hover interattivo

Pannello Destro (resto schermo):
• Grafico grande e dettagliato
• Header con informazioni complete
• Griglia con etichette prezzo
• Area riempita sotto la linea del grafico
• Statistiche min/max

Gestione Click

```
fn draw_list_item(
    stock: &StockData,
    x: f32, y: f32, width: f32, height: f32,
    is_selected: bool,
    mouse_pos: (f32, f32),
) -> bool {
    let (mx, my) = mouse_pos;
    let is_hovered = mx >= x && mx <= x + width
        && my >= y && my <= y + height;
    let is_clicked = is_hovered
        && is_mouse_button_pressed(MouseButton::Left);

    // Colori dinamici in base allo stato
    let bg_color = if is_selected {
        Color::from_rgba(50, 100, 200, 255)
    } else if is_hovered {
        Color::from_rgba(40, 40, 50, 255)
    } else {
        Color::from_rgba(30, 30, 40, 255)
    };

    draw_rectangle(x, y, width, height, bg_color);
    // ... rendering contenuto ...

    is_clicked
}
```

6. Versione 5: Selezione Multipla

La versione finale permette di selezionare più titoli contemporaneamente tramite checkbox. I grafici selezionati vengono visualizzati in una griglia dinamica nel pannello destro.

HashSet per Selezioni

```
use std::collections::HashSet;

struct App {
    stocks: Arc<Mutex<HashMap<String, StockData>>>,
    selected_symbols: HashSet<String>,  // Multi-selezione
}

// Nel loop principale:
if let Some(clicked_symbol) = clicked {
    if app.selected_symbols.contains(&clicked_symbol) {
        app.selected_symbols.remove(&clicked_symbol);
    } else {
        app.selected_symbols.insert(clicked_symbol);
    }
}
```

Checkbox Personalizzati

```
// Disegna checkbox
let checkbox_size = 20.0;
let checkbox_x = x + 10.0;
let checkbox_y = y + (height - checkbox_size) / 2.0;

let checkbox_color = if is_selected {
    Color::from_rgba(50, 100, 200, 255)
} else {
    Color::from_rgba(40, 40, 50, 255)
};

draw_rectangle(checkbox_x, checkbox_y,
               checkbox_size, checkbox_size, checkbox_color);

// Checkmark se selezionato
if is_selected {
    draw_line(checkbox_x + 4.0, checkbox_y + 10.0,
              checkbox_x + 8.0, checkbox_y + 16.0,
              2.0, WHITE);
    draw_line(checkbox_x + 8.0, checkbox_y + 16.0,
              checkbox_x + 16.0, checkbox_y + 4.0,
              2.0, WHITE);
}
```

Layout Griglia Dinamico

```
fn draw_charts_panel(
    stocks_snapshot: &HashMap<String, StockData>,
    selected_symbols: &HashSet<String>,
    x: f32, y: f32, width: f32, height: f32,
) {
    let selected_vec: Vec<&String> =
        selected_symbols.iter().collect();
    let count = selected_vec.len();

    // Calcola layout ottimale
    let cols = if count == 1 { 1 }
               else if count <= 4 { 2 }
               else { 3 };
    let rows = (count + cols - 1) / cols;

    let chart_width = (width - padding * (cols + 1)) / cols;
```

```
    let chart_height = (height - padding * (rows + 1)) / rows;

    // Disegna ogni grafico nella griglia
    for (i, symbol) in selected_vec.iter().enumerate() {
        let col = i % cols;
        let row = i / cols;
        let chart_x = x + padding + col * (chart_width + padding);
        let chart_y = y + padding + row * (chart_height + padding);
        draw_mini_chart(stock, chart_x, chart_y,
                        chart_width, chart_height);
    }
}
```

Scroll nella Lista

Per gestire liste lunghe, implementiamo lo scroll con la rotella del mouse:

```
let (_wheel_x, wheel_y) = mouse_wheel();
if mouse_in_list_area {
    scroll_offset += wheel_y * 20.0;
    scroll_offset = scroll_offset.max(0.0);
}

// Disegna solo elementi visibili
let item_y = start_y + (i * item_height) - scroll_offset;
if item_y + item_height >= visible_area_y
   && item_y <= visible_area_y + visible_area_height {
    draw_list_item(...);
}
```

7. Versione Finale: Supporto Criptovalute

L'ultima evoluzione aggiunge il supporto per le criptovalute. Yahoo Finance fornisce dati crypto con il suffisso -USD (es: BTC-USD per Bitcoin).

Lista Simboli Completa

```
let symbols = vec![
    // ETF
    "SPY".to_string(),
    "QQQ".to_string(),
    "DIA".to_string(),
    "IWM".to_string(),

    // Tech Stocks
    "AAPL".to_string(),
    "MSFT".to_string(),
    "GOOGL".to_string(),
    "AMZN".to_string(),
    "NVDA".to_string(),

    // Criptovalute (suffisso -USD)
    "BTC-USD".to_string(),    // Bitcoin
    "ETH-USD".to_string(),    // Ethereum
    "BNB-USD".to_string(),    // Binance Coin
    "SOL-USD".to_string(),    // Solana
    "ADA-USD".to_string(),    // Cardano
    "DOGE-USD".to_string(),   // Dogecoin
    "XRP-USD".to_string(),    // Ripple
    "MATIC-USD".to_string(),  // Polygon
];
```

Differenze Crypto vs Stock

Caratteristica	Azioni/ETF	Criptovalute
Orari trading	9:30-16:00 EST	24/7
Volatilità	Moderata	Molto alta
Suffisso Yahoo	Nessuno	-USD
Aggiornamenti	Durante ore mercato	Continui
Esempio simbolo	AAPL	BTC-USD

Nota importante: Le criptovalute su Yahoo Finance usano sempre il suffisso -USD. Altri suffissi esistono (-EUR, -GBP) ma -USD è il più comune e liquido. Non funzionano i simboli senza suffisso.

8. Dipendenze e Setup

Cargo.toml Finale

```
[package]
name = "stock_tracker"
version = "0.1.0"
edition = "2021"

[dependencies]
macroquad = "0.4"
reqwest = { version = "0.11", features = ["blocking", "json"] }
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"
chrono = "0.4"
```

Compilazione ed Esecuzione

```
# Compila in modalità release (più veloce)
cargo build --release

# Esegui l'applicazione
cargo run --release

# Per il debug (più lento ma con messaggi di errore)
cargo run
```

Requisiti Sistema

• **Rust**: Versione 1.70 o superiore
• **Sistema Operativo**: Linux, macOS, Windows
• **RAM**: Minimo 4GB (consigliati 8GB)
• **GPU**: Qualsiasi GPU moderna con supporto OpenGL 3.3+
• **Connessione Internet**: Necessaria per dati in tempo reale

Installazione Rust

```
# Linux / macOS
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

# Windows
# Scarica rustup-init.exe da https://rustup.rs/

# Verifica installazione
rustc --version
cargo --version
```

9. Conclusioni e Sviluppi Futuri

Abbiamo sviluppato un'applicazione completa di tracking finanziario partendo da zero, affrontando e risolvendo diverse sfide tecniche lungo il percorso. L'applicazione finale offre:

• Interfaccia grafica moderna e reattiva
• Aggiornamenti automatici ogni minuto
• Supporto per stock, ETF e criptovalute
• Selezione multipla con visualizzazione a griglia
• Gestione thread-safe dei dati
• Scroll e interattività completa
• Grafici con area riempita e statistiche

Possibili Miglioramenti Futuri

1. Funzionalità Avanzate:
• Zoom e pan sui grafici
• Indicatori tecnici (MA, RSI, MACD)
• Notifiche per variazioni significative
• Export dati in CSV/Excel
• Watchlist personalizzate salvate

2. Interfaccia:
• Dark/Light mode
• Temi colore personalizzabili
• Drag-and-drop per riordinare
• Finestre ridimensionabili
• Layout salvabili

3. Dati:
• Intervalli temporali variabili (1m, 5m, 1h, 1d)
• Storico più lungo
• Più provider di dati
• Notizie correlate ai titoli
• Analisi fondamentale

4. Performance:
• Cache locale dei dati
• Compressione storico
• Ottimizzazione rendering
• Richieste parallele
• WebSocket per dati real-time

Risorse Utili

• **Documentazione Macroquad**: https://macroquad.rs/
• **Rust Book**: https://doc.rust-lang.org/book/

• **Yahoo Finance API**: Non ufficiale, usare con cautela
• **Alternative API**: Alpha Vantage, Twelve Data, IEX Cloud
• **Repo GitHub Macroquad**: https://github.com/not-fl3/macroquad

Questo progetto dimostra come Rust e Macroquad possano essere utilizzati per creare applicazioni desktop performanti e moderne. La gestione della concorrenza con thread e Arc> è un pattern fondamentale in Rust che garantisce sicurezza e prestazioni.

Buon coding! ■