Stock Trading Dashboard Specification for Code Generation

1. Purpose & Overview

This document defines a cross-platform native stock trading dashboard application, outlining all features, workflows, and technical details necessary for automatic code generation.

- **Primary Views:**
  - Risk Management Dashboard (psychological & emotional risk metrics)
  - Stock-Rating Dashboard (market, sector, security ratings with opportunity visualization)
  - Live Calendar & Trade Tracker (weekly grid slots, historical review)
- **User Workflow:**
  1. Calibrate risk using daily psychological factors
  2. Rate market, sectors, and individual securities
  3. Input and manage trade positions in a rotating basket strategy
  4. Review historical trades via calendar and searchable log

2. Tech Stack & Languages

- **Core Language:** Rust (business logic, data models)
- **Framework:** Tauri (cross-platform native wrapper)
- **Frontend:** Svelte or React (embedded web UI in Tauri)
- **Persistence:** SQLite (primary), JSON files (fallback)
- **Charting:** Plotters (Rust) or Chart.js via Tauri for opportunity visualizations

3. Code Generator Input Guidelines

When feeding into a code generator, ensure scaffolding for:

1. Rust project layout with `src-tauri` and frontend code directories
2. Data model definitions (Rust structs/enums)
3. Business logic functions (state calculations, persistence)
4. UI component stubs corresponding to Gherkin scenarios
5. Cucumber test harness and step definitions in Rust or JS
6. Database schema and migration scripts for SQLite

Cucumber/Gherkin and Pseudo-code Plan for Stock Trading Dashboard

---

1. Psychological & Risk Metrics Extension

Feature: Psychological State and Total Risk Score

```
Scenario: User inputs psychological factors for risk assessment
Given the RM dashboard is displayed
When the user adjusts inputs for:
  | Factor                      | Value Range |   |
  | Gain/Loss Yesterday         | numeric     | (float) |
  | Emotional State             | slider      | (-3…+3) |
  | FOMO                        | slider      | (-3…+3) |
  | Market Bias                 | slider      | (-3…+3) |
  | Hunger                      | slider      | (0…+3)  |
  | Headache/Pain               | slider      | (0…+3)  |
  | <Additional Factor>         | slider/text | custom  |
And the user clicks "Compute Risk Score"
Then the Total Risk Score is calculated
And displayed prominently on the RM dashboard
```

Pseudo-code:

```
struct PsychologicalState {
    gain_loss_yesterday: f64,
    emotional_state: i32,    // -3 to +3
    fomo: i32,               // -3 to +3
    market_bias: i32,        // -3 to +3
    hunger: i32,             // 0 to +3
    headache_pain: i32,      // 0 to +3
    extra_factors: HashMap<String, i32>,
}

fn calculate_total_risk(state: &PsychologicalState) -> f64 {
    let base = state.emotional_state + state.fomo + state.market_bias;
    let physical = state.hunger + state.headache_pain;
    let pnl_effect = (state.gain_loss_yesterday / 100.0).clamp(-1.0, 1.0) * 3.0;
    let extras: i32 = state.extra_factors.values().sum();
    (base as f64 + physical as f64 + pnl_effect + extras as f64) / 3.0
}

fn save_psych_state(state: PsychologicalState) {
    persist_to_db("psych_state", &state);
    let risk_score = calculate_total_risk(&state);
    update_ui_risk_score(risk_score);
}
```

2. Enhanced Opportunity Visualization

Feature: Opportunity Chart and Graphics

```
Scenario: Display opportunity visualization for a rated stock
Given the stock-rating dashboard shows a list of rated securities
When the user selects a security
Then an opportunity chart appears showing:
  - Enthusiasm Score overlaid on rarity distribution
  - Bar or gauge indicating strength of pattern bonus
  - Historical sentiment trend line
```

Pseudo-code & Notes:

```
// Render with Plotters or web SVG via Tauri
fn render_opportunity_chart(rating: &StockRating, history: &[StockRating]) {
    // X-axis: past N days/week ratings
    // Y-axis: sentiment score (-3 to +3)
    // Overlay: histogram of how often this pattern occurs market-wide
}
```

---

3. Sector Selection & Multiple Entries

Feature: Dynamic Sector List

```
Scenario: User selects sectors for screening
Given the stock-rating dashboard is displayed
When the user adds sectors from the list:
  | Basic Materials | Communication Services | Consumer Cyclical |
  | Consumer Defensive | Energy | Financial |
  | Healthcare | Industrials | Real Estate |
  | Technology | Utilities | ... |
And the user may add duplicates with rationale
Then the dashboard reflects each sector-selection slot
```

Pseudo-code:

```
let mut sectors: Vec<String> = Vec::new();
fn add_sector(sector: String) {
    sectors.push(sector);
    update_ui_sector_list(&sectors);
}
```

---

4. Detailed Stock Analysis Page

Feature: Deep-Dive Stock Rating and Trade Log

```
Scenario: User performs detailed analysis on a single security
Given the detailed-analysis page is displayed
When the user fills in fields:
  | Field                    | Type     | Notes                          |
  | Bull/Bear                | toggle   | Bull = +1, Bear = -1           |
  | Confidence               | slider   | (0…100%)                       |
  | Market Trend             | dropdown | e.g. Uptrend, Downtrend        |
  | Chart Pattern            | dropdown | High Base, Asc Triangle, etc.  |
  | Strategy                 | text     | e.g. "Breakout", "Pullback" |
  | Overall Score            | computed | internal algorithm             |
  | Market, Sector           | sliders  | (-3…+3)                        |
  | Security Symbol          | text     | e.g. AAPL                      |
  | Bought?                  | checkbox |                                |
  | Entry Reason             | text     |                                |
  | Time, Entry, Stop, Target | datetime, float, float, float   |
  | Short Leg, Long Leg      | text     | options legs                   |
  | Debit/Credit             | float    | positive = debit               |
  | Quantity                 | integer  |                                |
  | Risk (max), Reward       | float    | calculated based on entry      |
  | Max Gain, % Profit       | float    | post-trade computation         |
  | Greeks: Delta, Theta,    | float    | appended Greek values          |
  | Gamma, Vega, Alerts?     | text/list | e.g. price triggers           |
  | Exit Reason, Skip Reason | text     |                                |
And the user clicks "Save Analysis"
Then all fields are validated, persisted, and chart/UI updates
```

Pseudo-code:

```
struct DetailedAnalysis {
    bull_bear: i8,
    confidence: u8,
    market_trend: String,
    chart_pattern: ChartPattern,
    strategy: String,
    overall_score: i32,
    market_sentiment: i32,
    sector_sentiment: i32,
    security: String,
    bought: bool,
    entry_reason: String,
    time: DateTime<Utc>,
    entry_price: f64,
    stop_loss: f64,
    target_price: f64,
    short_leg: Option<String>,
    long_leg: Option<String>,
    debit_credit: f64,
    quantity: u32,
    risk_max: f64,
    reward: f64,
    max_gain: f64,
    percent_profit: f64,
    delta: f64,
    theta: f64,
    gamma: f64,
    vega: f64,
    alerts: Vec<String>,
    exit_reason: String,
    skip_reason: String,
}

fn save_detailed_analysis(a: DetailedAnalysis) {
    validate(&a)?;
    persist_to_db("analysis", &a);
    update_ui_analysis(&a);
}
```

5. Integration & Next Steps

1. **Extend data models** to include psychological and detailed-analysis structs.
2. **Build UI components** in Svelte/React (Tauri) or directly in iced for each new metric.
3. **Implement charts** using Plotters or an embedded JS library (e.g. Chart.js).
4. **Write Cucumber tests** for each scenario to drive development.
5. **Persist and migrate** database schema (SQLite) to support new tables and fields.

This expansion ensures you capture both the psychological risk factors and the granular details needed for deep stock analysis. Let me know if you'd like further elaboration or adjustments!