Project Specification: Academic Planner SaaS

Course: CS-UY 4513 - Software Engineering

Professor: Dr. DePasquale

Project Title: Investra (Stock Trading Analyzer)

Group Member: Chris Brasil (leader), Dhruv Gupta, Hanqi Liu, Hongyi Wu, Michael Bian, Sania K. Awale

Date: September 27th, 2025

1.0 Project Overview

The goal for our project is to develop a stock trading platform, a multi-module Software-as-a-Service (SaaS) system using Ruby on Rails, for both personal and company use. The system, “Investra ”, allows both investment managers and individuals to track their progress in the stock market at a given time. This service can help managers evaluate their execution traders monthly, weekly, etc. The intended users of this service are managers, associates, admins, and individual investors.

The investing system will require a management system for users and companies

- roles and permissions within a given company
- compartmentalized architecture, for users and stocks

Each module must expose a well-defined API for inter-module communication

2.0 Core Requirements

The project must adhere to the following mandatory characteristics:

2.1 User-Based System

The application must be a user-based system that requires a secure user authentication and authorization framework. User accounts will be managed via standard email/password, linked to their company/organization registration and login.

2.2 User Roles

The system will support three distinct user roles, each with specific permissions:

- Trader: Responsible for their own trades, can sign up using their own personal email/phone number.
- Associate Trader: Each associate trader works under a portfolio manager, and under the same domain/company. They make individual trades daily.
- Portfolio Manager: Responsible for establishing a separate organizational domain that is used for mass management of their associates. They are able to see the status of their associates and can modify details/plugins for their website.
- System Administrator: Has full access to the system. Can manage trading accounts and all system settings.

2.3 Persistent Storage

The application must use a relational database to persist all data. The database schema must be well-designed and contain no more than 10 tables. Proposed Database Schema (max 10 tables):

● users: Stores user credentials and profile information.
● roles: Defines user roles. (Trader, Associate, Manager, Admin)
● user_roles: Join table for users and roles.
● stock: Stores stock details (name, description, etc.).
● stock_price_history: Stores past stock histories.
● portfolios: Stores details on what stocks users had chosen.
● company: Stores company details
● company_associate: Connects the Associate and the Manager.
● trade: Stores trade details (tradeID, userID, action, datetime, quantity, Price, etc.)

2.4 Modular Architecture (3-5 Modules)

The system must be logically divided into a minimum of 3 and a maximum of 5 modules. These modules must demonstrate clear dependencies and rely on each other for full functionality.

Proposed Modules:

1. **User & Identity Management:**
    ○ Manages user registration, login, and profile information.
    ○ Manages roles and permissions (Trader, Associate, Manager, Admin)
    ○ **Dependency**: This module is a core dependency for all other modules.
2. **Trading & Portfolio Management:**
    ○ Allow traders and associates to place buy/sell orders for stocks
    ○ Tracking individual portfolios (stock holding, week/month performance, transaction history)
    ○ Provides real-time stock price updates
    ○ **Dependency**: depends on User & Identity Management for authentication, the Market Data module to get information to help the trade
3. **Market Data:**

    - ○ Supplies real-time and historical data on stock
    - ○ Display company information (seasonal financial reports, industry data, recent news)
    - ○ Historical price trends of each stock
    - ○ Runs AI/ML models to generate future stock price predictions for the trader to reference
    - ○ **Dependency**: Providing data to the Trading & Portfolio Management
4. **Associate Analytics:**
    - ○ Generates reports on associate trades and overall performance
    - ○ Provides manager dashboards, associate trade summary, ranking, profits, etc.
    - ○ Offer system-wide analytics for admin
    - ○ **Dependency**: Is dependent on the Trading/Portfolio Management module to analyze and rank associate profits. And Market Data & AI Prediction to help the analytics.

2.5 API Interfaces

Each module must expose a RESTful API interface. This is crucial for enabling the modules to communicate with each other and for potential future integrations. The API should follow standard REST conventions.

API Endpoints:

- User & Identity Management API:
    - ○ POST /api/users/register
    - ○ POST /api/users/login
    - ○ GET /api/users/me
    - ○ DELETE /api/users/:id
- Stock Management API:
    - ○ POST /api/orders
    - ○ GET /api/orders/:id
    - ○ GET /api/orders?user_id=123
    - ○ POST /api/stock/sell/:id
- Associate Management API:
    - ○ GET /api/associates
    - ○ POST /api/associates
    - ○ GET /api/associates/:id
    - ○ PATCH /api/associates/:id
    - ○ DELETE /api/associates/:id
- Portfolio API:
    - ○ GET /api/portfolios/:userId
    - ○ GET /api/portfolios/:id/holdings
    - ○ POST /api/portfolios
    - ○ PATCH /api/portfolios/:id
    - ○ DELETE /api/portfolios/:id

- Trade/Order APII:

- POST /api/orders
- GET /api/orders/:id
- GET /api/orders?userId=123
- PATCH /api/orders/:id/cancel

3.0 Technical Stack

- Language: Ruby
- Framework: Ruby on Rails (latest stable version)
- Database: MySQL
- Testing: RSpec for unit and integration testing.

4.0 Deliverables

The final submission must include:
1. Complete Source Code: A well-structured and commented codebase, including all necessary configuration files.
2. Project Documentation: A README.md file that explains the system's architecture, setup instructions, and how to run the application.
3. API Documentation: A separate document or file detailing all API endpoints, their expected parameters, and response formats.
4. Presentation: A brief presentation and demonstration of the application's core feature