# Aider Assignment 2: Building a Real-Time Financial Dashboard

## Overview

In this assignment, you will develop a real-time financial dashboard application using Elixir for the backend and Svelte for the frontend. Your application will connect to the Finnhub WebSocket API to display live stock market data for a selected portfolio of technology, finance, and consumer companies. This project is designed to demonstrate the real-time capabilities of Elixir and Phoenix for managing WebSocket connections and the reactivity of Svelte for building responsive user interfaces.

## Learning Objectives

By completing this assignment, you will: - Gain experience building a modern, real-time web application - Learn how to integrate third-party APIs into a full-stack application - Practice developing with Elixir and Svelte in a monorepo structure - Develop skills in using the Aider code assistant effectively - Implement progressive feature development with integrated testing

## Problem Statement

Many financial applications provide delayed data or require expensive subscriptions for real-time information. Your task is to build an accessible financial dashboard that leverages free, real-time market data to provide users with current stock information. The dashboard should display live price updates, daily percentage changes, and simple visualizations for the selected portfolio.

## Required Stocks to Display

Your dashboard must include real-time data for the following stocks:

1. **Technology**:
   - AAPL (Apple Inc.)
   - MSFT (Microsoft Corporation)
   - NVDA (NVIDIA Corporation)
   - GOOGL (Alphabet Inc.)
2. **Finance**:
   - JPM (JPMorgan Chase & Co.)
   - BAC (Bank of America Corporation)
   - V (Visa Inc.)
3. **Consumer**:
   - AMZN (Amazon.com Inc.)
   - WMT (Walmart Inc.)
   - MCD (McDonald's Corporation)

## Technical Requirements

### Backend (Elixir/Phoenix)

- Set up a Phoenix application that manages WebSocket connections to the Finnhub API
- Create GenServers to maintain connections and handle data processing
- Implement Phoenix Channels to broadcast updates to the frontend
- Use ETS (Erlang Term Storage) for short-term data caching
- Implement proper error handling and reconnection logic

### Frontend (Svelte)

- Build a responsive dashboard using Svelte components
- Display real-time stock prices and percentage changes
- Implement a simple line chart for each stock showing recent price movements
- Create a summary view for the entire portfolio
- Show visual indicators for price movements (up/down)

### Integration

- Connect the frontend to the backend using Phoenix Channels
- Ensure smooth real-time updates with minimal latency
- Implement proper loading states and error handling

## Project Milestones and Development Approach

This is an intensive weekend project that you will complete in a short timeframe. You will build this project incrementally, adding features and tests as you develop. Complete the following milestones in sequence:

**Milestone 1: Project Setup** - Initialize the monorepo structure - Set up the Phoenix backend - Create the Svelte frontend - Establish basic connectivity between the two

**Milestone 2: API Integration** - Implement Finnhub API authentication - Create GenServers for WebSocket management - Test basic data retrieval - Implement ETS storage for stock data

**Milestone 3: Data Flow** - Set up Phoenix Channels for data broadcasting - Create data transformation utilities - Begin basic frontend components

**Milestone 4: Frontend Development** - Build the dashboard UI components - Implement real-time data display - Create stock charts and indicators

**Milestone 5: Testing and Refinement** - Add tests for backend and frontend - Implement error handling and recovery mechanisms - Optimize performance and responsiveness

**Milestone 6: Documentation and Submission** - Complete project documentation - Prepare final submission with all required components

## Working with AI Assistants

For this assignment, you'll be using two AI assistants to help you complete the project efficiently within the tight weekend timeframe:

### Using Aider

Aider is an AI-powered code assistant that can help you with this project. Here's how to use it effectively:

1. **Project Planning**: Use Aider to help plan your application architecture. Ask it to suggest file structures, component breakdowns, and data flow diagrams.
2. **Code Generation**: When implementing specific features, ask Aider to generate code snippets. For example:
    - "Help me create a GenServer to manage the Finnhub WebSocket connection"
    - "Show me how to implement a Phoenix Channel for broadcasting stock updates"
    - "Write a Svelte component for displaying stock price changes with color indicators"
3. **Debugging**: When you encounter issues, share error messages with Aider for troubleshooting. Be specific about the context and what you've already tried.
4. **Learning**: Ask Aider to explain unfamiliar concepts or patterns you encounter during development.
5. **Testing**: Request help writing effective tests for your components and modules.
6. **Refactoring**: As your project grows, ask Aider for suggestions on code organization and performance improvements.

### Using opto-gpt

In addition to Aider, you should leverage our company's LLM server, opto-gpt, for conceptual guidance and high-level planning:

1. **Architectural Planning**: Use opto-gpt for broader architectural discussions and system design considerations before implementing in code.
2. **Conceptual Understanding**: When you need deeper explanations of Elixir concurrency patterns, Phoenix Channels, or Svelte reactivity, consult opto-gpt.

3. **Algorithm Development**: For any complex data processing or transformation logic, get suggestions from opto-gpt before implementation.
4. **Research**: Use opto-gpt to explore best practices and industry standards for financial dashboards.
5. **Project Management**: Get advice on organizing your work schedule to complete all requirements within the weekend timeframe.

Remember that opto-gpt should be used primarily for conceptual understanding and planning, while Aider is better suited for direct code implementation and debugging.

## Minimum Knowledge Requirements

To complete this assignment successfully, you should have:

### Elixir/Phoenix

- Basic Elixir syntax and pattern matching
- Understanding of the actor model and GenServers
- Familiarity with Phoenix framework basics
- Knowledge of how WebSockets and Phoenix Channels work

### Svelte

- Component structure and lifecycle
- Reactivity principles and state management
- Event handling
- Basic knowledge of props and stores

### General

- Git version control fundamentals
- Understanding of WebSockets and real-time data
- Basic knowledge of API integration
- Testing principles

## Grading Criteria

Your project will be evaluated based on the following criteria:

| Criterion | Weight | Description |
|---|---|---|
| Milestone Completion | 40% | Successful implementation of each project milestone, with all required functionality working correctly |

| Criterion | Weight | Description |
|---|---|---|
| Code Quality | 20% | Well-organized, properly commented code following best practices for Elixir and Svelte |
| Architecture | 15% | Appropriate separation of concerns and effective use of Elixir concurrency features |
| Testing | 15% | Comprehensive tests for both backend and frontend components |
| Documentation | 10% | Clear project documentation and milestone achievement paragraphs |

## Submission Requirements

1. A GitHub repository containing your monorepo with complete source code, including:
   - All `.aider` files and interaction history
   - Comprehensive tests for each component
   - Full documentation
2. A README.md file with:
   - Project description
   - Setup instructions
   - Description of architecture
   - Screenshots of the working application
3. A Milestone.md report containing:
   - A paragraph describing the accomplishment of each milestone
   - Challenges encountered during each milestone and how you resolved them
   - How you utilized both Aider and opto-gpt to complete each milestone
   - Lessons learned from each phase of the project

## Resources

- Finnhub API Documentation: https://finnhub.io/docs/api
- Elixir Documentation: https://elixir-lang.org/docs.html
- Phoenix Framework: https://hexdocs.pm/phoenix/overview.html
- Svelte Documentation: https://svelte.dev/docs
- Aider Documentation: https://aider.chat/docs/

## Notes for Success

1. **Start Small**: Begin with a single stock and ensure the data flow works end-to-end before adding more complexity.
2. **Leverage Aider's Strengths**: Use Aider for generating boilerplate code, explaining concepts, and suggesting solutions to specific problems.
3. **Incremental Testing**: Write tests as you develop features, not after completing the entire application.
4. **Learn from Errors**: When you encounter issues with the Finnhub API, use these as learning opportunities to improve your error handling.
5. **Documentation**: Document your code and architecture decisions as you go, not just at the end of the project.

Good luck with your assignment! Remember that the goal is not just to complete a working application, but to learn effective development practices and how to leverage AI assistance in your software engineering workflow.