**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI TP. HỒ CHÍ MINH**

**VIỆN CÔNG NGHỆ THÔNG TIN VÀ ĐIỆN, ĐIỆN TỬ**



**BÁO CÁO BÀI TẬP LỚN**

TP. Hồ Chi Minh, ngày 21 tháng 06 năm 2025

**STOCK TRACKER LAB**

**ỨNG DỤNG gRPC XÂY DỰNG HỆ THỐNG THEO DÕI**

*Môn:* Lập trình mạng

*Lớp học phần:* 010112301302

*Giảng viên hướng dẫn:* **Bùi Dương Thế**

*Sinh viên thực hiện:* **Nguyễn Thanh Tân** *– 089205000960*

**Lê Hoàng Chiến** *– 089205010292*

**Nguyễn Huỳnh Sang** *– 087205009789*

**CHỨNG KHOÁN THỜI GIAN THỰC**

**LỜI CẢM ƠN**

Nhóm 3 chúng em xin chân thành cảm ơn thầy **Bùi Dương Thế** đã tận tình giảng dạy và hướng dẫn trong suốt quá trình học tập và thực hiện đồ án môn học này.

Trong quá trình làm việc, nhóm đã học hỏi được rất nhiều kiến thức chuyên sâu về giao thức gRPC, kiến trúc hệ thống phân tán, cũng như kỹ năng tích hợp giữa các nền tảng công nghệ khác nhau (Node.js và Java). Những bài giảng và góp ý từ thầy là nguồn động lực quan trọng giúp nhóm hoàn thiện dự án **"Stock Tracker Lab"** một cách tốt nhất có thể.

Mặc dù nhóm đã cố gắng hoàn thiện dự án một cách nghiêm túc và đầy đủ, nhưng chắc chắn vẫn không tránh khỏi những thiếu sót. Nhóm rất mong nhận được sự góp ý từ thầy để có thể cải thiện trong các dự án sau này.

Trân trọng cảm ơn!

# MỤC LỤC

[MỤC LỤC 1](#_Toc201498415)

[DANH MỤC HÌNH ẢNH 4](#_Toc201498416)

[DANH MỤC BẢNG BIỂU 5](#_Toc201498417)

[CHƯƠNG 1: GIỚI THIỆU DỰ ÁN 6](#_Toc201498418)

[1.1. Tổng quan 6](#_Toc201498419)

[1.2. Mục tiêu 6](#_Toc201498420)

[1.3. Đối tượng 7](#_Toc201498421)

[1.4. Kế hoạch thực hiện (Workchart) 7](#_Toc201498422)

[CHƯƠNG 2: CƠ SỞ LÝ THUYẾT VÀ PHÂN TÍCH HỆ THỐNG 7](#_Toc201498423)

[2.1. Phân tích và Lựa chọn Công nghệ Lõi: gRPC 7](#_Toc201498424)

[2.2. Các khái niệm lý thuyết liên quan 9](#_Toc201498425)

[2.2.1. Kiến trúc Client-Server và Hệ thống phân tán 9](#_Toc201498426)

[2.2.2. Nền tảng Node.js và Mô hình Event-Driven 9](#_Toc201498427)

[2.2.3. Lập trình giao diện với Java Swing và Event Dispatch Thread (EDT) 9](#_Toc201498428)

[2.3. Phân tích yêu cầu chức năng (Use Case) 10](#_Toc201498429)

[2.3.1. Xác định tác nhân 10](#_Toc201498430)

[2.3.2. Biểu đồ Use Case 11](#_Toc201498431)

[2.3.3. Yêu cầu chức năng 12](#_Toc201498432)

[CHƯƠNG 3: THIẾT KẾ HỆ THỐNG 14](#_Toc201498433)

[3.1. Thiết kế kiến trúc tổng quan 14](#_Toc201498434)

[3.1.1. Backend Server (Node.js) 14](#_Toc201498435)

[3.1.2. Desktop Client (Java) 14](#_Toc201498436)

[3.1.3. Web Admin Dashboard (HTML/CSS/JS) 14](#_Toc201498437)

[3.2. Thiết kế luồng hoạt động (Workflow / Flowchart) 16](#_Toc201498438)

[3.3. Thiết kế thuật toán (Pseudo Code) 16](#_Toc201498439)

[3.3.1. Mã giả 3.1: Thuật toán mô phỏng giá cổ phiếu trong DataSimulator 17](#_Toc201498440)

[3.3.2. Mã giả 3.2: Logic xử lý stream hai chiều tại gRPC service 17](#_Toc201498441)

[CHƯƠNG 4: THỰC HÀNH TRIỂN KHAI VÀ KẾT QUẢ 18](#_Toc201498442)

[4.1. Thực hành: Cài đặt và Hướng dẫn sử dụng 18](#_Toc201498443)

[4.1.1. Yêu cầu môi trường 18](#_Toc201498444)

[4.1.2. Cài đặt và Khởi chạy Backend Server & Dashboard 19](#_Toc201498445)

[4.1.3. Cài đặt và Khởi chạy Desktop Client 19](#_Toc201498446)

[4.2. Thực nghiệm: Hình ảnh sản phẩm và kết quả 19](#_Toc201498447)

[4.2.1. Giao diện Quản trị viên (Web Dashboard) 20](#_Toc201498448)

[4.2.2. Giao diện Người dùng (Desktop Client) 21](#_Toc201498449)

[CHƯƠNG 5: TỔNG KẾT VÀ ĐÁNH GIÁ 26](#_Toc201498450)

[5.1. Tổng kết kết quả đạt được 26](#_Toc201498451)

[5.2. Kết quả chưa đạt được và Hạn chế 27](#_Toc201498452)

[5.3. Kinh nghiệm rút ra 27](#_Toc201498453)

[5.3.1. Về kỹ thuật 27](#_Toc201498454)

[5.3.2. Về quy trình làm việc 28](#_Toc201498455)

[5.4. Hướng phát triển 28](#_Toc201498456)

[5.4.1. Tăng cường bảo mật 28](#_Toc201498457)

[5.4.2. Nâng cấp Client 28](#_Toc201498458)

[5.4.3. Hoàn thiện Backend 28](#_Toc201498459)

[5.4.4. Triển khai và Vận hành 29](#_Toc201498460)

[CHƯƠNG 6: KẾT LUẬN 29](#_Toc201498461)

[TÀI LIỆU THAM KHẢO 30](#_Toc201498462)

[PHỤ LỤC A: API CONTRACT 30](#_Toc201498463)

[A.1. Định nghĩa chung (common.proto) 30](#_Toc201498464)

[A.2. Dịch vụ Theo dõi Cổ phiếu (stock\_tracker.proto) 30](#_Toc201498465)

[A.3. Dịch vụ Theo dõi Chỉ số (index\_tracker.proto) 32](#_Toc201498466)

[A.4. File tổng hợp Proto (stl.proto) 32](#_Toc201498467)

# DANH MỤC HÌNH ẢNH

[Hình 1.1. Sơ đồ Gantt kế hoạch thực hiện dự án 6](#_Toc201498390)

[Hình 2.1. Sơ đồ phân định vai trò và tương tác của các tác nhân trong hệ thống 9](#_Toc201498391)

[Hình 2.2. Biểu đồ use case yêu cầu chức năng 10](#_Toc201498392)

[Hình 3.1.Kiến trúc hệ thống với Backend tích hợp Dashboard 14](#_Toc201498393)

[Hình 3.2. Luồng hoạt động của Client khi kết nối và nhận dữ liệu 15](#_Toc201498394)

[Hình 4.1. Giao diện Web Dashboard ở trạng thái ban đầu 19](#_Toc201498395)

[Hình 4.2. Giao diện Web Dashboard khi dịch vụ gRPC đang hoạt động 20](#_Toc201498396)

[Hình 4.3. Cửa sổ cấu hình lần đầu cho Client (Onboarding) 21](#_Toc201498397)

[Hình 4.4. Giao diện chính của Client sau khi kết nối thành công 21](#_Toc201498398)

[Hình 4.5. Giao diện chi tiết khi theo dõi một mã cổ phiếu (BCM) 22](#_Toc201498399)

[Hình 4.6. Chuyển đổi theo dõi sang mã cổ phiếu khác (BID) 23](#_Toc201498400)

[Hình 4.7. Biểu đồ giá real-time của mã cổ phiếu FPT 24](#_Toc201498401)

[Hình 4.8 Menu chức năng của ứng dụng 24](#_Toc201498402)

[Hình 4.9 Hộp thoại "Cài đặt" cho phép thay đổi cấu hình 25](#_Toc201498403)

# DANH MỤC BẢNG BIỂU

[Bảng 2.1 Chức năng dành cho Người dùng Client 11](#_Toc201498353)

[Bảng 2.2 Chức năng dành cho quản trị viên 12](#_Toc201498354)

# GIỚI THIỆU DỰ ÁN

## Tổng quan

Trong bối cảnh thị trường tài chính và chứng khoán ngày càng phát triển, nhu cầu tiếp cận thông tin giá cả biến động theo thời gian thực đã trở thành một yêu cầu thiết yếu đối với các nhà đầu tư, nhà phân tích và các ứng dụng tài chính. Các kiến trúc truyền thống dựa trên REST API và cơ chế polling định kỳ thường gặp phải các vấn đề về độ trễ và hiệu suất, không đáp ứng được yêu cầu tức thời của các ứng dụng tài chính chuyên nghiệp. Chính vì vậy, dự án **"Stock Tracker Lab"** ra đời với mong muốn nghiên cứu và xây dựng một hệ thống theo dõi chứng khoán hiệu năng cao, ứng dụng các công nghệ giao tiếp hiện đại để giải quyết các thách thức trên.

## Mục tiêu

Mục tiêu của dự án là xây dựng một hệ thống Client-Server hoàn chỉnh có khả năng mô phỏng và truyền tải dữ liệu giá chứng khoán theo thời gian thực. Hệ thống cho phép người dùng theo dõi biến động của các mã cổ phiếu và chỉ số thị trường thông qua một ứng dụng desktop trực quan. Đồng thời, dự án cũng cung cấp một Web Dashboard để quản trị viên có thể điều khiển và cấu hình hoạt động của server. Dự án được triển khai bằng kiến trúc đa ngôn ngữ, với **Backend** được xây dựng bằng **Node.js/TypeScript** và Client được xây dựng bằng **Java/Swing**. Giao thức **gRPC** với tính năng streaming hai chiều được sử dụng làm phương thức giao tiếp chính, và **Protocol Buffers** được dùng để định nghĩa cấu trúc dữ liệu.

## Đối tượng

Hệ thống hướng đến hai đối tượng người dùng chính với các vai trò và chức năng riêng biệt:

* **Người dùng Client:** Là những người có nhu cầu theo dõi dữ liệu thị trường, ví dụ như nhà đầu tư, nhà phân tích, hoặc sinh viên đang nghiên cứu về lĩnh vực tài chính. Họ sẽ sử dụng ứng dụng desktop để xem biểu đồ và các chỉ số giá.
* **Quản trị viên:** Là những người chịu trách nhiệm vận hành và bảo trì hệ thống. Họ sử dụng Web Dashboard để khởi động, dừng, cấu hình server và giám sát hoạt động của hệ thống.

## Kế hoạch thực hiện (Workchart)

Kế hoạch thực hiện dự án được vạch ra qua 5 giai đoạn chính, được minh họa bằng biểu đồ Gantt dưới đây.

A screenshot of a computer

AI-generated content may be incorrect.

Hình 1.1. Sơ đồ Gantt kế hoạch thực hiện dự án

# CƠ SỞ LÝ THUYẾT VÀ PHÂN TÍCH HỆ THỐNG

## Phân tích và Lựa chọn Công nghệ Lõi: gRPC

Nền tảng của hệ thống được xây dựng dựa trên quyết định chiến lược về công nghệ giao tiếp. Sau khi xem xét các phương án, gRPC đã được lựa chọn làm giải pháp cốt lõi dựa trên các phân tích chi tiết về hiệu năng, kiến trúc và quy trình phát triển.

Lý do hàng đầu cho sự lựa chọn này đến từ hiệu năng vượt trội mà **gRPC** mang lại, vốn được xây dựng trên nền tảng **HTTP/2 tiên tiến**. Cụ thể, **HTTP/2** hỗ trợ Multiplexing (ghép kênh), cho phép gửi và nhận nhiều yêu cầu và phản hồi đồng thời trên một kết nối TCP duy nhất, loại bỏ hoàn toàn vấn đề **“head-of-line blocking”** (chặn do tắc nghẽn ở đầu hàng) của **HTTP/1.1**. Đặc biệt, tính năng **Bidirectional Streaming** (luồng hai chiều) tạo ra một kênh giao tiếp song công, liên tục, cho phép server chủ động đẩy dữ liệu về client ngay khi có sự kiện, đây là yếu tố then chốt cho các ứng dụng thời gian thực như dự án này. Thêm vào đó, việc sử dụng **Binary Protocol** (giao thức nhị phân) với **Protocol Buffers** để tuần tự hóa dữ liệu giúp thông điệp truyền đi có kích thước nhỏ gọn và được xử lý nhanh hơn nhiều so với định dạng văn bản như JSON, đồng thời cơ chế **Header Compression** (nén header) cũng góp phần giảm thiểu băng thông mạng.

Bên cạnh hiệu năng, một ưu điểm chiến lược khác là khả năng tương tác đa ngôn ngữ **(Cross-Language Interoperability)**. Thế mạnh này được thể hiện rõ trong dự án khi tích hợp thành công giữa Backend viết bằng Node.js và Client viết bằng Java. Cơ chế này hoạt động dựa trên việc định nghĩa một “hợp đồng**”** API chung và trung lập về ngôn ngữ trong file .proto, từ đó tự động sinh ra mã nguồn tương thích, đảm bảo hai hệ thống có thể giao tiếp một cách liền mạch.

Về mặt quy trình phát triển, **gRPC** khuyến khích phương pháp thiết kế "Hợp đồng trước" **(Contract-First)**. Việc phải định nghĩa các service (dịch vụ) và message (tin nhắn) một cách chặt chẽ ngay từ đầu tạo ra một bản thiết kế API rõ ràng, giúp giảm thiểu lỗi tích hợp và cho phép các thành phần Client và Server được phát triển song song một cách hiệu quả.

## Các khái niệm lý thuyết liên quan

### Kiến trúc Client-Server và Hệ thống phân tán

Kiến trúc Client-Server là mô hình cơ bản của hệ thống, trong đó Client (ứng dụng Java) chịu trách nhiệm về giao diện và tương tác người dùng, trong khi Server (ứng dụng Node.js) quản lý logic nghiệp vụ và dữ liệu. Mô hình này giúp tách biệt các mối quan tâm (separation of concerns), cho phép phát triển và bảo trì độc lập từng thành phần.

### Nền tảng Node.js và Mô hình Event-Driven

Node.js là một môi trường runtime JavaScript phía server, được xây dựng trên engine V8 của Chrome. Với kiến trúc hướng sự kiện, không chặn (non-blocking I/O), Node.js đặc biệt phù hợp để xây dựng các ứng dụng mạng có khả năng mở rộng, xử lý hàng ngàn kết nối đồng thời một cách hiệu quả.

### Lập trình giao diện với Java Swing và Event Dispatch Thread (EDT)

Java Swing là một bộ công cụ GUI cho Java để xây dựng các ứng dụng desktop. Một khái niệm quan trọng trong Swing là Event Dispatch Thread (EDT). Mọi tương tác và cập nhật trên giao diện Swing phải được thực hiện trên luồng duy nhất này để đảm bảo an toàn luồng (thread-safety).

## Phân tích yêu cầu chức năng (Use Case)

### Xác định tác nhân

Hệ thống có hai tác nhân chính, mỗi tác nhân có một vai trò và mục đích sử dụng hệ thống riêng biệt:

* **Người dùng Client:** Đây là tác nhân chính sử dụng sản phẩm cuối, tương tác trực tiếp với ứng dụng Desktop Client (Java Swing). Mục tiêu chính của họ là theo dõi, phân tích và trực quan hóa dữ liệu thị trường chứng khoán được cung cấp bởi hệ thống. Họ thực hiện các thao tác như xem danh sách cổ phiếu, chọn một mã cụ thể để xem biểu đồ giá chi tiết, và cấu hình các thông số kết nối tới server.
* **Quản trị viên:** Đây là tác nhân có vai trò kỹ thuật, chịu trách nhiệm quản lý, vận hành và giám sát hoạt động của Backend Server. Họ tương tác với hệ thống thông qua giao diện Web Dashboard tích hợp, thực hiện các công việc như khởi động hoặc dừng dịch vụ gRPC, thay đổi các tham số cấu hình hệ thống, và theo dõi logs hoạt động.

A diagram of a user

AI-generated content may be incorrect.

Hình 2.1. Sơ đồ phân định vai trò và tương tác của các tác nhân trong hệ thống

### Biểu đồ Use Case

A diagram of a customer service

AI-generated content may be incorrect.

Hình 2.2. Biểu đồ use case yêu cầu chức năng

### Yêu cầu chức năng

#### **Mô tả Use Case cho tác nhân Người dùng Client**

Bảng 2.1 Chức năng dành cho Người dùng Client

|  |  |  |
| --- | --- | --- |
| Mã số | Tên Use Case | Mô tả chi tiết |
| UC1 | Cấu hình kết nối lần đầu | Trong lần chạy đầu tiên, ứng dụng hiển thị cửa sổ “Onboarding” yêu cầu người dùng nhập Địa chỉ và Cổng kết nối của gRPC Server để bắt đầu. |
| UC2 | Theo dõi thị trường tổng quan | Giao diện chính hiển thị danh sách các mã cổ phiếu và chỉ số thị trường (VN-INDEX) với giá và % thay đổi được cập nhật liên tục. |
| UC3 | Xem chi tiết cổ phiếu | Khi người dùng nhấp vào một mã cổ phiếu, vùng hiển thị chi tiết sẽ cập nhật thông tin của mã đó, bao gồm biểu đồ giá, tổng khối lượng/giá trị giao dịch và các thông tin giá bổ sung (sàn, trần, tham chiếu…). |
| UC4 | Thay đổi cấu hình kết nối | Người dùng có thể vào menu Tùy chọn → Cài đặt để mở hộp thoại cho phép thay đổi lại địa chỉ và cổng kết nối của Server. |
| UC5 | Khởi động lại ứng dụng | Người dùng có thể chọn Tùy chọn → Khởi động lại để đóng và mở lại ứng dụng một cách nhanh chóng. |

#### **Mô tả Use Case cho tác nhân Quản trị viên**

Bảng 2.2 Chức năng dành cho quản trị viên

|  |  |  |
| --- | --- | --- |
| Mã số | Tên Use Case | Mô tả chi tiết |
| UC6 | Quản lý trạng thái Server | Quản trị viên sử dụng các nút bấm trên Dashboard để Bắt đầu, Dừng, hoặc Khởi động lại tiến trình Backend Server. |
| UC7 | Giám sát Server | Quản trị viên theo dõi các chỉ số hoạt động của Server theo thời gian thực, bao gồm Trạng thái (Running/Stopped), Port gRPC, số lượng Client đang kết nối, và thời gian hoạt động (Uptime). |
| UC8 | Quản lý Logs | Quản trị viên xem các log hệ thống được ghi lại (server started, dashboard started…) và có thể xóa toàn bộ log khỏi giao diện. |
| UC9 | Cấu hình hoạt động Server | Quản trị viên có thể thay đổi các tham số cấu hình của server như Port, tần suất cập nhật, khối lượng giao dịch mô phỏng, và biên độ thay đổi giá tối đa. |
| UC10 | Quản lý danh mục cổ phiếu | Quản trị viên có thể xem, thêm mới hoặc xóa các mã cổ phiếu được sử dụng trong bộ dữ liệu mô phỏng. |

# THIẾT KẾ HỆ THỐNG

## Thiết kế kiến trúc tổng quan

Hệ thống được thiết kế theo kiến trúc Client-Server, trong đó thành phần Backend đóng vai trò trung tâm và có một cấu trúc đặc biệt, đảm nhiệm đồng thời hai chức năng riêng biệt trên các kênh giao tiếp khác nhau.

### Backend Server (Node.js)

Là một ứng dụng thống nhất, chịu trách nhiệm cho toàn bộ logic phía máy chủ. Ứng dụng này lắng nghe trên hai cổng (ports) mạng riêng biệt:

Kênh gRPC (ví dụ: Port 50051): Dành riêng cho Desktop Client kết nối và nhận luồng dữ liệu chứng khoán

Kênh HTTP/WebSocket (ví dụ: Port 3000): Do framework Hono quản lý, dùng để phục vụ giao diện Web Admin Dashboard cho Quản trị viên

### Desktop Client (Java)

Là một ứng dụng độc lập, chỉ giao tiếp với Backend Server thông qua kênh gRPC.

### Web Admin Dashboard (HTML/CSS/JS)

Là một tập hợp các tài nguyên tĩnh được tích hợp và phục vụ trực tiếp bởi Backend Server qua kênh HTTP.

A diagram of a software system

AI-generated content may be incorrect.

Hình 3.1.Kiến trúc hệ thống với Backend tích hợp Dashboard

## Thiết kế luồng hoạt động (Workflow / Flowchart)

Biểu đồ tuần tự dưới đây mô tả các bước tương tác chính khi một Client kết nối và nhận dữ liệu từ Server.

**A diagram with text and colorful squares

AI-generated content may be incorrect.**

Hình 3.2. Luồng hoạt động của Client khi kết nối và nhận dữ liệu

## Thiết kế thuật toán (Pseudo Code)

Để làm rõ logic xử lý cốt lõi, phần này trình bày mã giả cho các thuật toán quan trọng của hệ thống.

### Mã giả 3.1: Thuật toán mô phỏng giá cổ phiếu trong DataSimulator

Function updateStockPrices():

FOR EACH stock IN allStocks:

max\_change = stock.referencePrice \* config.maxPriceChangePercent

price\_change = random(-0.1, 0.1) \* max\_change

new\_price = stock.currentPrice + price\_change

// Ensure price is within floor and ceiling

new\_price = max(stock.floorPrice, min(stock.ceilingPrice, new\_price))

stock.currentPrice = new\_price

stock.lastUpdateTime = now()

ENDFOR

EndFunction

### Mã giả 3.2: Logic xử lý stream hai chiều tại gRPC service

Function SubscribeToUpdates(call\_stream):

// Listen for messages from client

call\_stream.on('data', request => {

IF request.action == START THEN

add\_stream\_to\_active\_list(call\_stream)

start\_broadcasting\_if\_not\_started()

ELSE IF request.action == STOP THEN

remove\_stream\_from\_active\_list(call\_stream)

stop\_broadcasting\_if\_list\_is\_empty()

ENDIF

})

// Handle client disconnection

call\_stream.on('cancelled', () => {

remove\_stream\_from\_active\_list(call\_stream)

})

EndFunction

# THỰC HÀNH TRIỂN KHAI VÀ KẾT QUẢ

## Thực hành: Cài đặt và Hướng dẫn sử dụng

### Yêu cầu môi trường

Backend Server:

* Hệ điều hành: Windows, macOS, hoặc Linux
* Runtime: Bun v1.0+ (hoặc Node.js v20+)

Desktop Client:

* Java Development Kit (JDK) phiên bản 21 hoặc mới hơn
* Hệ thống build Gradle v8.0+

Quản trị viên:

* Trình duyệt web hiện đại (Chrome, Firefox, Edge)

### Cài đặt và Khởi chạy Backend Server & Dashboard

1. Di chuyển vào thư mục gốc của backend: cd server-ts
2. Thực thi lệnh bun install để cài đặt các thư viện cần thiết
3. Thực thi lệnh bun dev để khởi động server
4. Sau khi khởi động thành công, server sẽ lắng nghe trên hai port: port gRPC (ví dụ: 50051) cho Client Java và port HTTP (ví dụ: 3000) cho Dashboard
5. Truy cập Bảng điều khiển (Dashboard): Mở trình duyệt web và truy cập vào địa chỉ http://localhost:3000 để xem giao diện quản trị

### Cài đặt và Khởi chạy Desktop Client

1. Mở dự án client-java trong một Môi trường Phát triển Tích hợp (IDE) có hỗ trợ Gradle như IntelliJ IDEA
2. Thực thi tác vụ build của Gradle để biên dịch dự án và tải các thư viện
3. Thực thi tác vụ run của Gradle để khởi chạy ứng dụng Client

## Thực nghiệm: Hình ảnh sản phẩm và kết quả

Kết quả của giai đoạn triển khai là một sản phẩm phần mềm hoàn chỉnh, có khả năng hoạt động và phản ánh các mục tiêu thiết kế đã được đề ra. Hoạt động của hệ thống được thể hiện một cách trực quan thông qua hai giao diện chính:

* Giao diện **Web Dashboard** dành cho Quản trị viên, phục vụ công tác quản lý và giám sát Backend Server.
* Giao diện **Desktop Client** dành cho Người dùng cuối, phục vụ việc kết nối, nhận và hiển thị dữ liệu chứng khoán theo thời gian thực.

Các hình ảnh và phân tích dưới đây sẽ minh họa chi tiết cho các chức năng này.

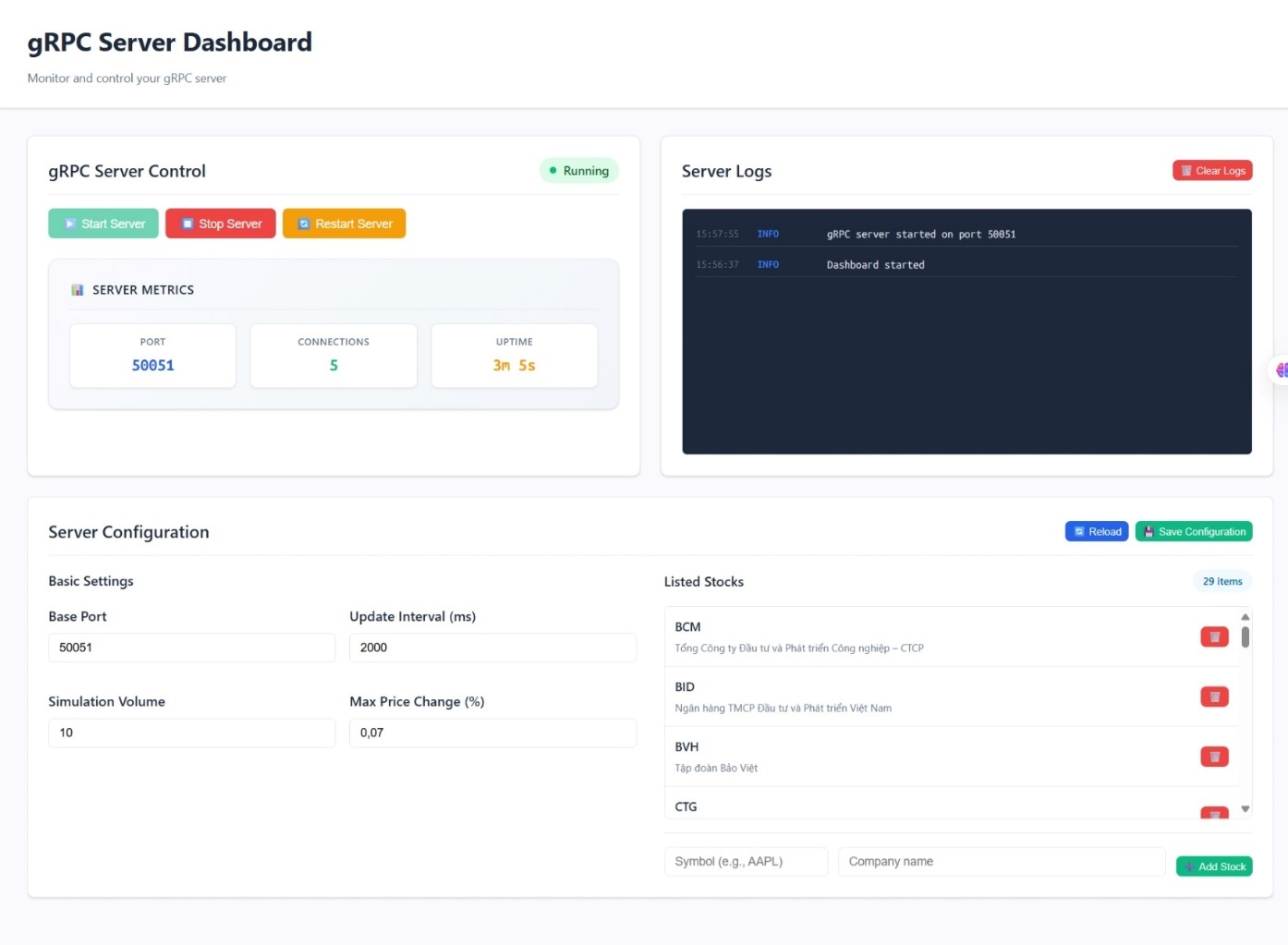
### Giao diện Quản trị viên (Web Dashboard)

Giao diện Web Dashboard là công cụ quản trị hệ thống, được xây dựng dưới dạng một trang web và phục vụ trực tiếp từ Backend Server. Giao diện này cung cấp cho Quản trị viên một trung tâm điều khiển tập trung, cho phép thực hiện các tác vụ quan trọng như: khởi động, dừng hoặc khởi động lại dịch vụ gRPC; giám sát các chỉ số hoạt động theo thời gian thực như trạng thái, port, số lượng kết nối và thời gian hoạt động; đồng thời cho phép thay đổi các tham số cấu hình của hệ thống một cách linh hoạt

A screenshot of a computer dashboard

AI-generated content may be incorrect.

Hình 4.1. Giao diện Web Dashboard ở trạng thái ban đầu



Hình 4.2. Giao diện Web Dashboard khi dịch vụ gRPC đang hoạt động

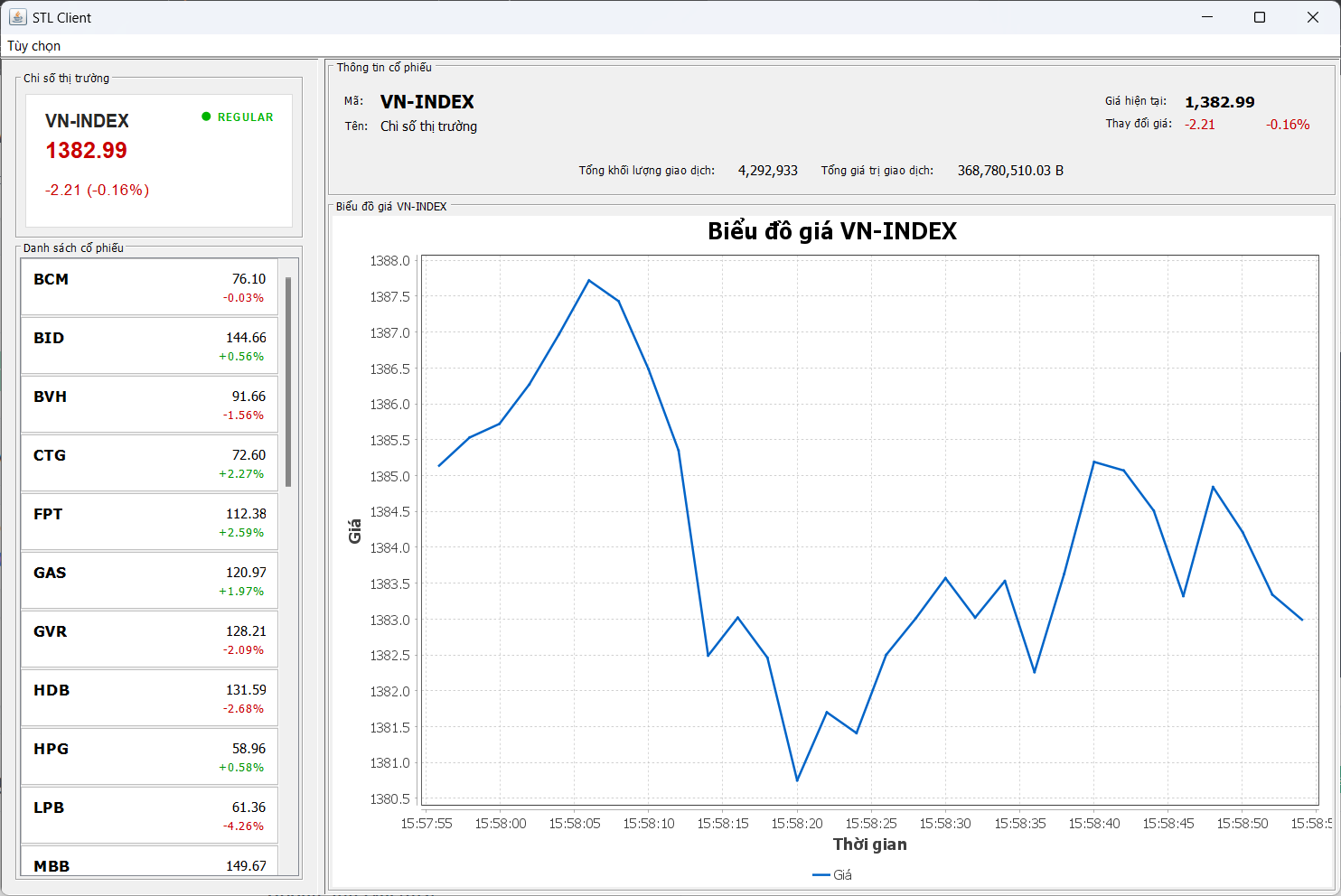
### Giao diện Người dùng (Desktop Client)

Thành phần Desktop Client là một ứng dụng độc lập được phát triển bằng Java Swing, đóng vai trò là giao diện người dùng cuối. Chức năng chính của Client là kết nối đến gRPC server để nhận và hiển thị dữ liệu chứng khoán một cách trực quan. Các tính năng nổi bật bao gồm một danh sách các mã cổ phiếu được cập nhật liên tục, một khu vực hiển thị chi tiết với biểu đồ giá động cho từng mã được chọn, và các hộp thoại cho phép người dùng tự cấu hình thông tin kết nối đến server.

A screenshot of a computer

AI-generated content may be incorrect.

Hình 4.3. Cửa sổ cấu hình lần đầu cho Client (Onboarding)



Hình 4.4. Giao diện chính của Client sau khi kết nối thành công

A screenshot of a graph

AI-generated content may be incorrect.

Hình 4.5. Giao diện chi tiết khi theo dõi một mã cổ phiếu (BCM)

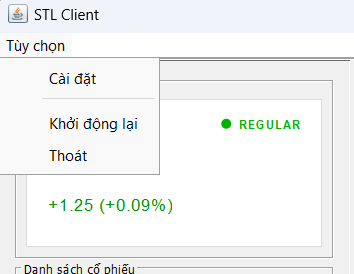


Hình 4.6. Chuyển đổi theo dõi sang mã cổ phiếu khác (BID)

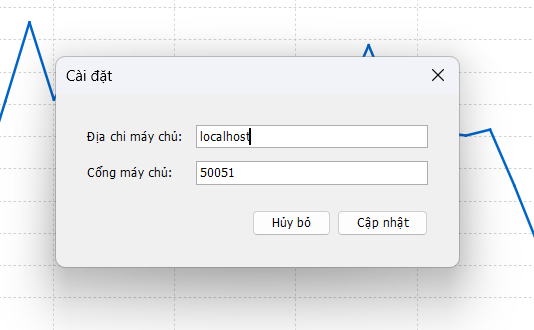
A screen shot of a graph

AI-generated content may be incorrect.

Hình 4.7. Biểu đồ giá real-time của mã cổ phiếu FPT



Hình 4.8 Menu chức năng của ứng dụng



Hình 4.9 Hộp thoại "Cài đặt" cho phép thay đổi cấu hình

g

# TỔNG KẾT VÀ ĐÁNH GIÁ

## Tổng kết kết quả đạt được

Dự án đã hoàn thành các mục tiêu chính đã đề ra, cụ thể:

* **Xây dựng thành công Backend Server (Node.js)**: Server có khả năng mô phỏng dữ liệu chứng khoán, quản lý và phát luồng dữ liệu thông qua các dịch vụ gRPC. Server cũng tích hợp một Web
* Dashboard quản trị hoạt động ổn định.
* **Triển khai thành công giao thức gRPC**: Đã ứng dụng hiệu quả giao thức gRPC, đặc biệt là tính năng Bidirectional Streaming, để tạo ra kênh giao tiếp hai chiều, hiệu năng cao giữa server và client.
* **Phát triển thành công ứng dụng Client (Java Swing)**: Ứng dụng client có giao diện người dùng trực quan, có khả năng kết nối, nhận, và hiển thị dữ liệu chứng khoán theo thời gian thực một cách mượt mà.
* **Hoàn thiện kiến trúc đa ngôn ngữ**: Hệ thống đã chứng minh được khả năng tương tác giữa hai nền tảng công nghệ khác nhau (Node.js và Java) thông qua một hợp đồng API được định nghĩa bằng Protocol Buffers.

## Kết quả chưa đạt được và Hạn chế

Bên cạnh những kết quả đã đạt được, dự án vẫn còn một số hạn chế cần được cải thiện:

* **Cơ chế xử lý lỗi**: Việc xử lý lỗi khi kết nối bị ngắt đột ngột ở phía client còn đơn giản, chưa có cơ chế tự động kết nối lại (retry) một cách thông minh.
* **Tính thực tế của dữ liệu mô phỏng**: Thuật toán mô phỏng dữ liệu còn đơn giản, chưa phản ánh hết được sự phức tạp và các quy luật biến động của thị trường chứng khoán thực tế.
* **Bảo mật**: Hệ thống hiện chưa triển khai các cơ chế bảo mật cho kênh truyền gRPC (như TLS) và chưa có cơ chế xác thực cho các client kết nối.
* **Kiểm thử**: Dự án chưa có các kịch bản kiểm thử tự động (unit test, integration test), việc kiểm thử chủ yếu được thực hiện thủ công.

## Kinh nghiệm rút ra

Quá trình thực hiện dự án đã mang lại nhiều kiến thức và kinh nghiệm quý báu:

### Về kỹ thuật

* Hiểu biết sâu sắc về nguyên lý hoạt động, ưu điểm và cách triển khai gRPC, đặc biệt là các mô hình streaming
* Nắm vững quy trình làm việc với Protocol Buffers để định nghĩa API và tự động sinh mã nguồn (code generation)
* Kinh nghiệm thực tế trong việc xây dựng và tổ chức một ứng dụng Java Swing theo kiến trúc module hóa, dễ bảo trì

### Về quy trình làm việc

* Kinh nghiệm trong việc thiết kế hệ thống theo hướng “Hợp đồng trước” (Contract-First), giúp giảm thiểu rủi ro và tăng hiệu quả khi phát triển song song các thành phần
* Kỹ năng gỡ lỗi (debug) trong một hệ thống phân tán đa ngôn ngữ

## Hướng phát triển

Dựa trên các hạn chế hiện tại, dự án có thể được phát triển và mở rộng theo các hướng sau:

### Tăng cường bảo mật

* Tích hợp TLS (Transport Layer Security) để mã hóa kênh truyền gRPC
* Xây dựng cơ chế xác thực client dựa trên token (ví dụ: JWT) được gửi qua metadata của gRPC

### Nâng cấp Client

* Cải thiện biểu đồ giá với các chỉ báo kỹ thuật (technical indicators) như MA, RSI
* Cho phép người dùng tùy chỉnh danh sách các mã cổ phiếu muốn theo dõi

### Hoàn thiện Backend

* Xây dựng thuật toán mô phỏng dữ liệu phức tạp hơn
* Mở rộng API để cung cấp thêm các thông tin lịch sử và thống kê chi tiết

### Triển khai và Vận hành

* Đóng gói các thành phần của hệ thống bằng công nghệ container hóa (ví dụ: Docker) để đơn giản hóa quá trình triển khai và tăng tính di động

# KẾT LUẬN

Trong thời gian thực hiện đồ án, nhóm em đã hoàn thành được các mục tiêu cốt lõi, xây dựng thành công một hệ thống theo dõi chứng khoán thời gian thực bằng việc nghiên cứu và ứng dụng giao thức gRPC. Thông qua việc tích hợp thành công giữa Backend Node.js và Client Java, đề tài đã giải quyết được bài toán về truyền tải dữ liệu hiệu năng cao, đồng thời chứng minh được tính linh hoạt và ưu việt của kiến trúc đa ngôn ngữ dựa trên gRPC.

Quá trình thực hiện dự án đã giúp nhóm học hỏi và tích lũy được nhiều kiến thức và kinh nghiệm thực tiễn quý báu, từ việc thiết kế hệ thống phân tán, làm chủ giao thức gRPC và mô hình streaming, cho đến kỹ năng tích hợp hai nền tảng công nghệ khác biệt là Node.js và Java. Bên cạnh đó, nhóm cũng đã có những trải nghiệm giá trị về kỹ năng làm việc nhóm và quản lý dự án để hoàn thành sản phẩm.

Sản phẩm hoàn thiện không chỉ đáp ứng được các mục tiêu đề ra mà còn mở ra nhiều hướng phát triển tiềm năng. Nhóm hy vọng rằng hệ thống này sẽ là một nền tảng vững chắc cho các cải tiến sau này và là một minh chứng rõ ràng cho những kiến thức đã được học và áp dụng trong suốt môn học.

# TÀI LIỆU THAM KHẢO

* [gRPC](https://grpc.io/docs/)
* [Bun — A fast all-in-one JavaScript runtime](https://bun.sh/)
* [Hono - Web framework built on Web Standards](https://hono.dev/)
* [JFreeChart](https://www.jfree.org/jfreechart/)

# PHỤ LỤC A: API CONTRACT

Toàn bộ định nghĩa API (API Contract) được sử dụng trong dự án, định nghĩa bằng Protocol Buffers.

## A.1. Định nghĩa chung (common.proto)

syntax = "proto3";

package stl.proto;

option java\_package = "io.github.withtozy.stl.proto";

option java\_multiple\_files = false;

option java\_outer\_classname = "CommonProto";

enum ActionType {

START = 0;

STOP = 1;

}

message StockSubscriptionRequest {

ActionType action = 1;

}

message HistoricalPricePoint {

double price = 1;

int64 timestamp = 2;

}

## A.2. Dịch vụ Theo dõi Cổ phiếu (stock\_tracker.proto)

syntax = "proto3";

import "common.proto";

package stl.proto;

option java\_package = "io.github.withtozy.stl.proto";

option java\_multiple\_files = true;

option java\_outer\_classname = "StockerTrackerProto";

service StockTracker {

rpc SubscribeToStockListUpdates(stream stl.proto.StockSubscriptionRequest)

returns (stream StockUpdateResponse);

rpc SubscribeToDetailedStockUpdates(stream StockDetailSubscriptionRequest)

returns (stream StockDetailedUpdateResponse);

rpc GetStockPriceHistory(StockHistoryRequest)

returns (StockHistoryResponse);

}

message StockDetailSubscriptionRequest {

string symbol = 1;

}

message StockHistoryRequest {

string symbol = 1;

}

message StockUpdateResponse {

string symbol = 1;

double current\_price = 2;

double percentage\_change = 3;

int64 last\_update\_time = 4;

}

message StockDetailedUpdateResponse {

string symbol = 1;

string name = 2;

double current\_price = 3;

double lowest\_price = 4;

double highest\_price = 5;

double ceiling\_price = 6;

double floor\_price = 7;

double average\_price = 8;

int64 total\_volume = 9;

int64 last\_update\_time = 10;

double percentage\_change = 11;

double reference\_price = 12;

double total\_value = 13;

}

message StockHistoryResponse {

string symbol = 1;

repeated HistoricalPricePoint history = 2;

}

## A.3. Dịch vụ Theo dõi Chỉ số (index\_tracker.proto)

syntax = "proto3";

import "common.proto";

package stl.proto;

option java\_package = "io.github.withtozy.stl.proto";

option java\_multiple\_files = true;

option java\_outer\_classname = "MarketIndexTrackerProto";

service MarketIndexTracker {

rpc SubscribeToMarketIndexUpdates(stream stl.proto.StockSubscriptionRequest)

returns (stream MarketIndexUpdateResponse);

rpc SubscribeToDetailedMarketIndexUpdates(stream stl.proto.StockSubscriptionRequest)

returns (stream MarketIndexDetailedUpdateResponse);

rpc GetMarketIndexHistory(MarketIndexHistoryRequest)

returns (MarketIndexHistoryResponse);

}

message MarketIndexUpdateResponse {

string index\_symbol = 1;

double current\_index\_value = 2;

int64 last\_update\_time = 3;

double percentage\_change = 4;

}

message MarketIndexDetailedUpdateResponse {

string index\_symbol = 1;

double current\_index\_value = 2;

int64 total\_volume = 3;

double total\_value = 4;

int64 last\_update\_time = 5;

double percentage\_change = 6;

}

message MarketIndexHistoryResponse {

string index\_symbol = 1;

repeated HistoricalPricePoint history = 2;

}

message MarketIndexHistoryRequest {}

## A.4. File tổng hợp Proto (stl.proto)

syntax = "proto3";

import public "common.proto";

import public "index\_tracker.proto";

import public "stock\_tracker.proto";

package stl.proto;

option java\_package = "io.github.withtozy.stl.proto";

option java\_multiple\_files = true;

option java\_outer\_classname = "StlProto";