**Tài Liệu Báo Cáo Về API Crypto Compare**

1. **Giới thiệu**
2. **Tổng quan về API**

Dữ liệu thị trường đa dạng của Crypto Compare bao gồm dữ liệu giao dịch tiền điện tử, dữ liệu sổ đặt hàng, dữ liệu blockchain, dữ liệu xã hội và dữ liệu lịch sử. Đảm bảo tính toàn vẹn và chính xác của dữ liệu của mình thông qua các quy trình nghiêm ngặt như kiểm tra API thường xuyên, đánh giá trao đổi hàng tháng và báo cáo bất thường.

API của CryptoCompare cung cấp các điểm cuối có độ tin cậy cao và có khả năng mở rộng, đạt 180 triệu yêu cầu mỗi giờ vào thời điểm cao điểm (25 tỷ mỗi tháng). Cơ sở hạ tầng chạy trên nhiều máy chủ trên nhiều trung tâm dữ liệu để đảm bảo phân phối dữ liệu nhanh nhất và độ trễ thấp nhất có thể. Môi trường lưu trữ dự phòng và cân bằng tải để có độ tin cậy tối đa.

Hàng tháng, hơn 30 triệu người nhận được thông tin về giá tiền điện tử! Trung bình, có đến 30.000 khách hàng được kết nối với dữ liệu phát trực tiếp và hơn 100 máy chủ WebSocket và API để hỗ trợ tải. Các dịch vụ vi mô tích hợp phụ trợ thu thập và xử lý hơn 8.000 giao dịch và 300.000 cập nhật sổ đặt hàng mỗi giây.

Điểm cuối API sử dụng HTTPS (mã hóa TLS) để lưu lượng truy cập của người dùng được bảo mật khi yêu cầu và nhận dữ liệu. Để đảm bảo tính bảo mật tối đa, bạn cũng có thể yêu cầu Crypto Compare ký vào các phản hồi API của mình. Một số điểm cuối yêu cầu đăng ký và tạo Khóa API.

1. **Ứng Dụng của API**

Price (Giá): API của chúng tôi trả về giá hiện tại của bất kỳ loại tiền điện tử nào và tất cả thông tin giao dịch (giá, khối lượng, mở, cao, thấp, v.v.) cho các cặp được yêu cầu. Chúng tôi cung cấp dữ liệu về giá cho hơn 5.300 đồng xu và hơn 240.000 cặp tiền tệ.

Historical Data (Lịch sử dữ liệu): Nhận dữ liệu lịch sử hàng ngày, hàng giờ và phút, dữ liệu hàng ngày tại bất kỳ dấu thời gian nhất định nào, giá trung bình hàng ngày dựa trên vwap hàng giờ và tổng khối lượng trao đổi hàng ngày và hàng giờ.

Top Lists (Danh sách hàng đầu): Các điểm cuối của danh sách hàng đầu truy xuất dữ liệu khối lượng cho các sàn giao dịch hàng đầu cho một cặp tiền tệ, dữ liệu CryptoCompare Index (CCCAGG) đầy đủ, danh sách các đồng tiền hàng đầu, danh sách các cặp giao dịch hàng đầu và các đồng tiền hàng đầu theo tổng khối lượng trên tất cả các thị trường trong 24 giờ qua.

Streaming (Truyền phát): API phát trực tuyến trả về tất cả các tổ hợp thông tin đăng ký và giá cả để biết nội dung nào cần phát trực tuyến và cách kết nối với người truyền phát. Bạn cũng có thể nhận được tất cả các kênh đăng ký bộ truyền phát có sẵn cho các cặp được yêu cầu.

News (Tin tức): Nhận nguồn cấp tin tức và bài viết từ tất cả các nhà cung cấp tin tức tiền điện tử lớn mà CryptoCompare đã tích hợp. Bạn cũng có thể truy xuất danh sách đầy đủ các danh mục, một cách tuyệt vời để lọc qua các bài báo.

More Information (Thông tin khác): Bạn có thể làm được nhiều điều hơn nữa với API và CryptoCompare liên tục bổ sung các điểm cuối có giá trị. Những đề cập đáng trân trọng: danh sách tất cả các sàn giao dịch và cặp giao dịch, các sàn giao dịch cấu thành cho CryptoCompare Index (CCCAGG) và tất cả các đồng tiền trên trang web CryptoCompare.

1. **Hướng Dẫn Sử Dụng Các Tính Năng Chính Của API**

**Bước 1:** Truy cập vào trang web [Crypto Compare](https://min-api.cryptocompare.com) sauđónhấp vào Login rồi đăng ký tài khoản ở phần Sign Up rồi đăng nhập

A screenshot of a login form

Description automatically generated

**Bước 2:** Nhấp vào [API Key](https://www.cryptocompare.com/cryptopian/api-keys) và tạo API Key để theo dõi, quản lý số lần gọi,... của API Key đó

A screenshot of a computer

Description automatically generated

**Bước 3:** Truy cập vào đường link [document](https://min-api.cryptocompare.com/documentation) để vào trang tài liệu API

A screenshot of a computer

Description automatically generated

**Lưu ý**: Gán API Key vào sau Endpoint &api\_key={your\_api\_key}

\*Nếu sử dụng dụng Postman chỉ cần gán Endpoint vào và get data về

\*API chỉ hỗ trợ theo dõi thị trường và thông tin của các đồng tiền ảo trên sàn chứ không hỗ trợ trading trực tiếp lên sàn

1. **Price (Giá):**

Single Symbol Price (Giá ký hiệu đơn): Tính toán thông tin giao dịch hiện tại (giá, khối lượng, mở cửa, cao, thấp, v.v.) của cặp được yêu cầu dưới dạng trung bình trọng số khối lượng dựa trên các sàn giao dịch được yêu cầu.

Endpoint:

<https://min-api.cryptocompare.com/data/price?fsym=BTC&tsyms=USD,JPY,EUR>

Kết quả trả về:

USD: 61634.1

JPY: 9249769.54

EUR: 56962.33

Multiple Symbols Price (Giá nhiều biểu tượng): Tương tự như đường dẫn API đơn nhưng có nhiều đường dẫn từ ký hiệu.

Endpoint:

<https://min-api.cryptocompare.com/data/pricemulti?fsyms=BTC,ETH&tsyms=USD,EUR>

Kết quả trả về:

 Info: Object

USD: 61462.56

EUR: 56777.17

 ETH: Object

USD: 3374.99

EUR: 3118.27

Multiple Symbols Full Data (Nhiều ký hiệu dữ liệu đầy đủ): Nhận tất cả thông tin giao dịch hiện tại (giá, khối lượng, mở, cao, thấp, v.v.) của bất kỳ danh sách tiền điện tử nào bằng bất kỳ loại tiền tệ nào khác mà bạn cần. Nếu tiền điện tử không giao dịch trực tiếp với toSymbol được yêu cầu thì BTC sẽ được sử dụng để chuyển đổi.

API này cũng trả về giá trị Hiển thị cho tất cả các trường. Nếu cặp đối diện giao dịch, chúng tôi sẽ đảo ngược nó (ví dụ BTC-XMR)

Endpoint:

<https://min-api.cryptocompare.com/data/pricemultifull?fsyms=BTC&tsyms=USD,EUR>

Kết quả trả về: Test trên web để có kết quả đầy đủ

FROMSYMBOL: "BTC" (Ký hiệu đồng tiền)

TOSYMBOL: "USD" (Mệnh giá quy đổi)

PRICE: 60982.8421148541 (Giá đồng tiền hiện tại)

VOLUMEHOUR: 6.01203307 (Khối lượng giao dịch trong giờ gần nhất)

VOLUMEHOURTO: 366658.112019214 (Giá trị tổng cộng của các giao dịch trong giờ gần nhất)

OPENHOUR: 60992.7461124188 (Giá mở cửa của đồng tiền trong giờ gần nhất)

HIGHHOUR: 60995.6376386993 (Giá cao nhất của đồng tiền trong giờ gần nhất)

LOWHOUR: 60977.6230506756 (Giá thấp nhất của đồng tiền trong giờ gần nhất)

VOLUMEDAY: 6891.55870534 (Khối lượng giao dịch trong ngày hiện tại)

VOLUMEDAYTO: 422299606.805097(Giá trị tổng cộng của các giao dịch trong ngày hiện tại)

OPENDAY: 61179.5847839226 (Giá mở cửa của đồng tiền trong ngày hiện tại)

HIGHDAY: 61777.0827526541 (Giá cao nhất của đồng tiền trong ngày hiện tại)

LOWDAY: 60802.1270717901 (Giá thấp nhất của đồng tiền trong ngày hiện tại)

VOLUME24HOUR: 82050.20924688(Khối lượng giao dịch trong 24 giờ gần nhất)

VOLUME24HOURTO: 5093603317.60184 (Giá trị tổng cộng của các giao dịch trong 24 giờ gần nhất)

OPEN24HOUR: 61581.508790741 (Giá mở cửa của đồng tiền 24 giờ trước)

HIGH24HOUR: 63652.7592923188 (Giá cao nhất của đồng tiền trong 24 giờ gần nhất)

LOW24HOUR: 60370.977074658 (Giá thấp nhất của đồng tiền trong 24 giờ gần nhất)

Generate Custom Average (Tạo mức trung bình tùy chỉnh): Tính toán thông tin giao dịch hiện tại (giá, khối lượng, mở cửa, cao, thấp, v.v.) của cặp được yêu cầu dưới dạng trung bình trọng số khối lượng dựa trên các sàn giao dịch được yêu cầu.

Endpoint:

<https://min-api.cryptocompare.com/data/generateAvg?fsym=BTC&tsym=USD&e=Kraken>

Kết quả trả về: Test trên web để có kết quả

MARKET: "CUSTOMAGG" (Thị trường giao dịch)

FROMSYMBOL: "BTC" (Đồng tiền gốc)

TOSYMBOL: "USD" (Đồng tiền quy đổi)

PRICE: 60952 (Giá hiện tại của đồng tiền)

LASTUPDATE: 1709262921 (Thời gian cập nhật dữ liệu cuối cùng)

LASTVOLUME: 0.000985 (Khối lượng giao dịch gần nhất)

LASTVOLUMETO: 60.03772 (Giá trị của giao dịch gần nhất được chuyển đổi sang đồng tiền quy đổi)

LASTTRADEID: "67561968" (ID của giao dịch cuối cùng)

VOLUME24HOUR: 6371.93937478 (Khối lượng giao dịch trong 24 giờ qua)

VOLUME24HOURTO: 394534140.983912 (Giá trị tổng cộng của các giao dịch trong 24 giờ qua)

OPEN24HOUR: 61554.6 (Giá mở cửa của đồng tiền 24 giờ trước)

HIGH24HOUR: 63583.5 (Giá cao nhất của đồng tiền trong 24 giờ qua)

LOW24HOUR: 60108.8 (Giá thấp nhất của đồng tiền trong 24 giờ qua)

LASTMARKET: "Kraken" (Sàn giao dịch cuối cùng)

TOPTIERVOLUME24HOUR: 6371.93937478 (Khối lượng giao dịch trong 24 giờ qua tính trên các sàn hàng đầu)

TOPTIERVOLUME24HOURTO: 394534140.983912 (Giá trị tổng cộng của các giao dịch trong 24 giờ qua tính trên các sàn hàng đầu)

1. **Historial (Lịch sử):**

Daily Pair OHLCV (Cặp hàng ngày OHLCV): Nhận giá mở cửa, cao nhất, thấp nhất, đóng cửa, khối lượng từ và khối lượng tới từ dữ liệu lịch sử hàng ngày. Các giá trị được dựa trên thời gian 00:00 GMT. Nếu e=CCCAGG và tryConversion=true, nó sẽ cố gắng chuyển đổi thông qua BTC hoặc ETH để xác định đường dẫn tốt nhất có thể. Loại chuyển đổi và ký hiệu được sử dụng sẽ được thêm vào theo điểm lịch sử. Nếu bạn muốn lấy tất cả dữ liệu lịch sử có sẵn, bạn có thể sử dụng limit=2000 và tiếp tục quay ngược thời gian bằng cách sử dụng thông số toTs. Sau đó, bạn có thể tiếp tục yêu cầu các đợt bằng cách sử dụng: &limit=2000&toTs={the earliest timestamp received}.

Endpoint:

<https://min-api.cryptocompare.com/data/v2/histoday?fsym=BTC&tsym=USD&limit=10>

Kết quả trả về: Test trên web để có kết quả đầy đủ

TimeFrom: 1708387200 (Thời gian bắt đầu của khoảng thời gian)

TimeTo: 1709251200 (Thời gian kết thúc của khoảng thời gian)

 Data: Array [11] (Mảng dữ liệu giá trong khoảng thời gian đã cho)

 0: Object

time: 1708387200 (Thời gian của mỗi mẫu dữ liệu)

high: 52985.95 (Giá cao nhất trong khoảng thời gian đó)

low: 50766.77 (Giá thấp nhất trong khoảng thời gian đó)

open: 51781.22 (Giá mở cửa của khoảng thời gian đó)

volumefrom: 39504.32 (Khối lượng giao dịch từ đồng tiền gốc)

volumeto: 2050493239.79 (Giá trị của các giao dịch được chuyển đổi sang đồng tiền quy đổi)

close: 52269.59 (Giá đóng cửa của khoảng thời gian đó)

conversionType: "direct" (Loại quy đổi)

conversionSymbol: "" (Biểu tượng quy đổi)

Hourly Pair OHLCV (Cặp theo giờ OHLCV): Nhận giá mở cửa, cao nhất, thấp nhất, đóng cửa, khối lượng từ và khối lượng tới từ dữ liệu lịch sử hàng giờ. Nếu e=CCCAGG và tryConversion=true, nó sẽ cố gắng chuyển đổi thông qua BTC hoặc ETH để xác định đường dẫn tốt nhất có thể. Loại chuyển đổi và ký hiệu được sử dụng sẽ được thêm vào theo điểm lịch sử. Nếu bạn muốn lấy tất cả dữ liệu lịch sử có sẵn, bạn có thể sử dụng limit=2000 và tiếp tục quay ngược thời gian bằng tham số toTs. Sau đó, bạn có thể tiếp tục yêu cầu các đợt bằng cách sử dụng: &limit=2000&toTs={the earliest timestamp received}.

Endpoint:

<https://min-api.cryptocompare.com/data/v2/histohour?fsym=BTC&tsym=USD&limit=10>

Kết quả trả về: Test trên web để có kết quả

Minute Pair OHLCV (Cặp phút OHLCV): Nhận giá mở cửa, cao nhất, thấp nhất, đóng cửa, khối lượng từ và khối lượng đến từ dữ liệu lịch sử từng phút. Dữ liệu này chỉ được lưu trữ trong 7 ngày, nếu bạn cần nhiều hơn, hãy sử dụng đường dẫn hàng giờ hoặc hàng ngày. Nếu e=CCCAGG và tryConversion=true, nó sẽ cố gắng chuyển đổi thông qua BTC hoặc ETH để xác định đường dẫn tốt nhất có thể. Loại chuyển đổi và ký hiệu được sử dụng sẽ được thêm vào theo điểm lịch sử. Nếu bạn muốn lấy tất cả dữ liệu lịch sử có sẵn, bạn có thể sử dụng limit=2000 và tiếp tục quay ngược thời gian bằng cách sử dụng thông số toTs. Sau đó, bạn có thể tiếp tục yêu cầu các đợt bằng cách sử dụng: &limit=2000&toTs={the earliest timestamp received}.

Endpoint:

https://min-api.cryptocompare.com/data/v2/histominute?fsym=BTC&tsym=GBP&limit=10

Kết quả trả về: Test trên web để có kết quả

Daily Exchange Vol (Số lượng trao đổi hàng ngày): Nhận tổng khối lượng từ dữ liệu trao đổi lịch sử hàng ngày. Các giá trị được dựa trên thời gian 00:00 GMT. Chúng tôi lưu trữ dữ liệu bằng BTC và nhân với giá trị BTC-tsym. Nếu bạn muốn lấy tất cả dữ liệu lịch sử có sẵn, bạn có thể sử dụng limit=2000 và tiếp tục quay ngược thời gian bằng cách sử dụng thông số toTs. Sau đó, bạn có thể tiếp tục yêu cầu các đợt bằng cách sử dụng: &limit=2000&toTs={the earliest timestamp received}.

Endpoint: <https://min-api.cryptocompare.com/data/exchange/histoday?tsym=USD&limit=10>

Kết quả trả về:

Info: "Press Execute Call to see what the endpoint returns."

Message: "Got the data"

 Data: Array [11]

 0: Object

time: 1708387200

volume: 37464096390.44

 1: Object

time: 1708473600

volume: 33226762851.95

 2: Object

time: 1708560000

volume: 31957304865.97

Hourly Exchange Vol (Số lượng trao đổi hàng giờ): Nhận tổng khối lượng từ dữ liệu trao đổi lịch sử hàng giờ. Chúng tôi lưu trữ dữ liệu bằng BTC và nhân với giá trị BTC-tsym. Nếu bạn muốn lấy tất cả dữ liệu lịch sử có sẵn, bạn có thể sử dụng limit=2000 và tiếp tục quay ngược thời gian bằng cách sử dụng thông số toTs. Sau đó, bạn có thể tiếp tục yêu cầu các đợt bằng cách sử dụng: &limit=2000&toTs={the earliest timestamp received}.

Endpoint:

<https://min-api.cryptocompare.com/data/exchange/histohour?tsym=BTC&limit=10>

Kết quả trả về:

Info: "Press Execute Call to see what the endpoint returns."

Message: "Got the data"

 Data: Array [11]

 0: Object

time: 1709218800

volume: 38134.67

 1: Object

time: 1709222400

volume: 65782.79

 2: Object

time: 1709226000

volume: 65986.82

Daily Symbol Vol (Biểu tượng hàng ngày Vol): Nhận tổng khối lượng từ dữ liệu khối lượng biểu tượng lịch sử hàng ngày. Các tập được lưu trữ trong fsym (cơ sở) và được chuyển đổi bằng cách sử dụng tỷ lệ chuyển đổi fsym - tsym lịch sử phù hợp. Ví dụ: nếu bạn đang yêu cầu fsym BTC và tsym USD, khối lượng được trả về là tổng số BTC là một phần của tất cả các giao dịch (cơ sở và báo giá) được chuyển đổi bằng tỷ giá hối đoái BTC-USD

Giá trị dựa trên 00:00 giờ GMT và được tính bằng tổng của tất cả các giao dịch trong đó ký hiệu/tài sản có liên quan dưới dạng từ/cơ sở hoặc dưới dạng đến/báo giá/bộ đếm. Top\_tier\_volume\_quote và base là tổng số dựa trên các sàn giao dịch có trong điểm chuẩn trao đổi của chúng tôi được xếp hạng AA, A hoặc B. cccagg\_volume\_quote và \_base dựa trên các sàn giao dịch có trong mức trung bình CCCAGG của chúng tôi. Total\_volume\_quote và \_base là tổng của tất cả các giao dịch trên tất cả các thị trường trong đó tài sản/đồng xu là fsym/base hoặc tsym/quote/counter

Nếu bạn muốn lấy tất cả dữ liệu lịch sử có sẵn, bạn có thể sử dụng limit=2000 và tiếp tục quay ngược thời gian bằng cách sử dụng thông số toTs. Sau đó, bạn có thể tiếp tục yêu cầu các đợt bằng cách sử dụng: &limit=2000&toTs={the earliest timestamp received}.

Endpoint:

<https://min-api.cryptocompare.com/data/symbol/histoday?fsym=BTC&tsym=USD&limit=10>

Kết quả trả về: Test trên web để có kết quả đầy đủ

Data: Array [11]

 0: Object

time: 1708387200

top\_tier\_volume\_quote: 849043175.95 (Khối lượng giao dịch hàng đầu được tính bằng đồng tiền quy đổi)

top\_tier\_volume\_base: 13459995958.58 (Khối lượng giao dịch hàng đầu được tính bằng đồng tiền gốc)

top\_tier\_volume\_total: 14309038611.83 (Tổng khối lượng giao dịch hàng đầu)

cccagg\_volume\_quote: 842234016.46 (Khối lượng giao dịch trên CCCAGG được tính bằng đồng tiền quy đổi)

cccagg\_volume\_base: 9251441446.56 (Khối lượng giao dịch trên CCCAGG được tính bằng đồng tiền gốc)

cccagg\_volume\_total: 10093675463.02 (Tổng khối lượng giao dịch trên CCCAGG)

total\_volume\_quote: 17945700563.04 (Tổng khối lượng giao dịch được tính bằng đồng tiền quy đổi)

total\_volume\_base: 34684366141.7 (Tổng khối lượng giao dịch được tính bằng đồng tiền gốc)

total\_volume\_total: 52630066704.74 (Tổng khối lượng giao dịch)

Hourly Symbol Vol (Biểu tượng hàng giờ Vol): Nhận tổng khối lượng từ dữ liệu khối lượng ký hiệu lịch sử hàng giờ. Các tập được lưu trữ trong fsym (cơ sở) và được chuyển đổi bằng cách sử dụng tỷ lệ chuyển đổi fsym - tsym lịch sử phù hợp. Ví dụ: nếu bạn đang yêu cầu fsym BTC và tsym USD, khối lượng được trả về là tổng số BTC là một phần của tất cả các giao dịch (cơ sở và báo giá) được chuyển đổi bằng tỷ giá hối đoái BTC-USD

Giá trị dựa trên thời gian hàng giờ và được tính bằng tổng của tất cả các giao dịch trong đó ký hiệu/tài sản có liên quan dưới dạng từ/cơ sở hoặc dưới dạng đến/báo giá/bộ đếm. Top\_tier\_volume\_quote và base là tổng số dựa trên các sàn giao dịch có trong điểm chuẩn trao đổi của chúng tôi được xếp hạng AA, A hoặc B. cccagg\_volume\_quote và \_base dựa trên các sàn giao dịch có trong mức trung bình CCCAGG của chúng tôi. Total\_volume\_quote và \_base là tổng của tất cả các giao dịch trên tất cả các thị trường trong đó tài sản/đồng xu là fsym/base hoặc tsym/quote/counter

Nếu bạn muốn lấy tất cả dữ liệu lịch sử có sẵn, bạn có thể sử dụng limit=2000 và tiếp tục quay ngược thời gian bằng cách sử dụng thông số toTs. Sau đó, bạn có thể tiếp tục yêu cầu các đợt bằng cách sử dụng: &limit=2000&toTs={the earliest timestamp received}.

Endpoint:

<https://min-api.cryptocompare.com/data/symbol/histohour?fsym=BTC&tsym=USD&limit=10>

Kết quả trả về: Test trên web để có kết quả

All the Coins (Tất cả các đồng tiền ảo): Trả về tất cả số tiền mà CryptoCompare đã thêm vào trang web. Đây không phải là danh sách đầy đủ các đồng tiền chúng tôi có trong hệ thống, đây chỉ là danh sách các đồng tiền chúng tôi đã thực hiện một số nghiên cứu.

Endpoint:

<https://min-api.cryptocompare.com/data/all/coinlist>

Kết quả trả về: Test trên web để có kết quả

All Exchanges General Info (Tất cả các sàn giao dịch Thông tin chung): Bạn chỉ có thể sử dụng điểm cuối này với api\_key hợp lệ . Trả về thông tin chung và khối lượng 24 giờ cho tất cả các sàn giao dịch mà chúng tôi đã tích hợp.

Endpoint:

<https://min-api.cryptocompare.com/data/exchanges/general>

Kết quả trả về: Test trên web để có kết quả

Latest News Articles (Tin tức mới nhất): Trả về các bài viết tin tức từ các nhà cung cấp mà CryptoCompare đã tích hợp.

Endpoint:

<https://min-api.cryptocompare.com/data/v2/news/?lang=EN>

Kết quả trả về: Test trên web để có kết quả

List News Feeds (Liệt kê nguồn cấp tin tức): Trả về tất cả các nguồn cấp tin tức (nhà cung cấp) mà CryptoCompare đã tích hợp.

Endpoint:

<https://min-api.cryptocompare.com/data/news/feeds>

Kết quả trả về: Test trên web để có kết quả

1. **Chi Phí nâng cấp tài khoản**

**A screenshot of a website

Description automatically generated**

1. **Chi tiết về Code**
2. **Mô tả về Project**

Đoạn mã trên là một Component Angular được sử dụng để hiển thị biểu đồ giao dịch (candlestick chart) của một cặp tiền mã hóa (BTC/USD) sử dụng thư viện AMCharts 5. Dưới đây là mô tả của từng phần trong component:

Import Statements: Phần import các module và thư viện cần thiết cho component, bao gồm ChangeDetectorRef, Component, OnInit từ @angular/core, các module và class từ thư viện AMCharts, HttpClient từ @angular/common/http, và interval từ rxjs.

@Component Decorator: Decorator được sử dụng để cung cấp metadata cho Angular về cách tạo và sử dụng component. Trong đoạn mã này, có định nghĩa selector, templateUrl, styleUrls, và standalone (được đặt thành true).

BASE\_URL và API\_KEY: Các biến được sử dụng để lưu trữ URL cơ sở và khóa API được sử dụng để truy vấn dữ liệu giao dịch từ CryptoCompare API.

fetchOHLC(): Phương thức này được sử dụng để truy vấn dữ liệu giao dịch từ CryptoCompare API bằng cách sử dụng HttpClient của Angular.

ngOnInit(): Phương thức này được gọi khi Angular khởi tạo component. Trong phương thức này, biểu đồ candlestick được tạo và cấu hình bằng cách sử dụng thư viện AMCharts.

generateChartData(): Phương thức này được sử dụng để tạo dữ liệu mẫu cho biểu đồ giao dịch. Dữ liệu này bao gồm giá đóng cửa, giá mở cửa, giá thấp nhất, giá cao nhất, và khối lượng giao dịch của mỗi thời điểm.

Các phương thức và cấu hình khác như tạo biểu đồ, tạo các trục (axis), tạo series, và cấu hình các control như scrollbar, series type switcher, toolbar cho biểu đồ giao dịch.

Sử dụng interval từ RxJS để cập nhật dữ liệu giao dịch mỗi giây và hiển thị dữ liệu mới trên biểu đồ.

1. **Plugin, Link GitHub và vị trí Project ở máy công ty**

Biểu đồ sử dụng thư viện của amcharts (Tham khảo tài liệu <https://www.amcharts.com/docs/v5/charts/stock/>)

Link GitHub: [Chart](https://github.com/ThaiAn31/chart.git)

Project đặt ở E:\2.Project\chart\src\app\components\amchart\

1. **Cấu trúc Project**

A screenshot of a computer

Description automatically generated

\*Các thư mục quản lý UI

**A screenshot of a computer

Description automatically generated**

amchart.component.html

A screenshot of a computer program

Description automatically generated

amchart.component.ts

A screenshot of a computer program

Description automatically generated

private BASE\_URL: string = 'https://min-api.cryptocompare.com/data': Đây là khai báo biến BASE\_URL là một biến private (chỉ có thể truy cập trong class này) kiểu string, và gán giá trị là URL cơ sở của API được sử dụng để truy vấn dữ liệu giao dịch.

private API\_KEY: string = 'e26ca12fcad2c55de5cd9620fb875261c509ead99ecfeb2cfd595f1340d39e48': Tương tự như trên, đây là khai báo biến API\_KEY là một biến private kiểu string, và gán giá trị là khóa API được sử dụng để xác thực và truy cập vào API của CryptoCompare.

dataSubscription: any;: Đây là khai báo một biến dataSubscription, kiểu dữ liệu của biến này là any, có thể chứa bất kỳ loại dữ liệu nào. Có thể đây là một biến được sử dụng để lưu trữ kết quả của việc đăng ký subscription cho việc cập nhật dữ liệu.

firstDate: any = new Date();: Khai báo biến firstDate là một biến kiểu Date và gán giá trị ban đầu là thời điểm hiện tại. Biến này có thể được sử dụng để theo dõi thời điểm đầu tiên của dữ liệu hoặc thực hiện các phép toán liên quan đến thời gian.

value: number = 1200;: Đây là khai báo biến value là một biến kiểu number và gán giá trị ban đầu là 1200. Biến này có thể đại diện cho một giá trị nào đó trong dữ liệu giao dịch.

constructor(private http: HttpClient) { }: Đây là hàm khởi tạo của class. Nó nhận một đối tượng http kiểu HttpClient, được Angular cung cấp để thực hiện các yêu cầu HTTP như truy vấn API từ máy chủ. Trong trường hợp này, nó được sử dụng để gửi yêu cầu HTTP đến API của CryptoCompare.

A screen shot of a computer

Description automatically generated

const url = ${this.BASE\_URL}/histohlc?fsym=BTC&tsym=USD&limit=1000&api\_key=${this.API\_KEY}``: Dòng này tạo một biến url để lưu trữ URL hoàn chỉnh cho việc truy vấn dữ liệu OHLC (Open, High, Low, Close) từ API. Biến này sẽ được sử dụng trong yêu cầu HTTP để lấy dữ liệu giao dịch.

this.http.get(url): Sử dụng đối tượng http (kiểu HttpClient) để gửi một yêu cầu HTTP GET đến URL được chỉ định.

.subscribe({ ... }): Phương thức subscribe() được gọi trên đối tượng Observable để đăng ký một hàm xử lý cho dữ liệu được trả về từ yêu cầu HTTP.

next: (data: any) => { console.log('data', data) }: Đây là một hàm xử lý được truyền vào như một đối số cho phương thức subscribe(). Trong trường hợp này, nó được sử dụng để xử lý dữ liệu nhận được từ yêu cầu HTTP. Dữ liệu được in ra console để kiểm tra.

error: err => { console.log(err) }: Đây là một hàm xử lý lỗi được truyền vào như một đối số cho phương thức subscribe(). Nếu có lỗi xảy ra trong quá trình gửi yêu cầu HTTP, thông tin về lỗi sẽ được in ra console để gỡ lỗi.

A computer screen shot of text

Description automatically generated

let root = am5.Root.new("chartdiv");: Dòng này tạo một đối tượng root từ lớp am5.Root, đại diện cho gốc của cây DOM của biểu đồ. Tham số "chartdiv" chỉ định rằng biểu đồ sẽ được render trong một phần tử HTML có id là "chartdiv".

const myTheme = am5.Theme.new(root);: Dòng này tạo một theme mới (myTheme) từ lớp am5.Theme. Theme này sẽ được áp dụng cho biểu đồ.

myTheme.rule("Grid", ["scrollbar", "minor"]).setAll({ visible: false });: Dòng này đặt một quy tắc cho theme myTheme. Quy tắc này áp dụng cho các thành phần có tên là "Grid" trong danh sách các thành phần ["scrollbar", "minor"], và đặt thuộc tính visible thành false. Điều này có thể được hiểu là ẩn lưới cho các thành phần scrollbar và minor.

root.setThemes([ am5themes\_Animated.new(root), myTheme ]);: Dòng này áp dụng các theme cho đối tượng root. Theme am5themes\_Animated được thêm vào đầu tiên, sau đó là theme myTheme được tạo ở trên. Điều này đảm bảo rằng các cài đặt từ myTheme sẽ được áp dụng sau theme am5themes\_Animated.

let stockChart = root.container.children.push(am5stock.StockChart.new(root, { paddingRight: 0 }));: Dòng này tạo một biểu đồ giao dịch mới từ lớp am5stock.StockChart. Biểu đồ này được thêm vào trong đối tượng container của root. Thuộc tính paddingRight được đặt thành 0, có thể chỉ định khoảng cách giữa biểu đồ và lề phải của phần tử chứa nó.

root.numberFormatter.set("numberFormat", "#,###.00");: Dòng này đặt định dạng số toàn cục cho biểu đồ. Trong trường hợp này, định dạng số được đặt thành "#,###.00", với hai chữ số sau dấu thập phân và sử dụng dấu phân cách hàng nghìn. Điều này sẽ ảnh hưởng đến cách mà các số được hiển thị trên biểu đồ.

A screen shot of a computer program

Description automatically generated

let mainPanel = stockChart.panels.push(am5stock.StockPanel.new(root, { wheelY: "zoomX", panX: true, panY: true }));: Dòng này tạo một panel chính (hoặc chart) trong biểu đồ giao dịch. Panel này là nơi chứa các dữ liệu giao dịch chính và các thành phần khác của biểu đồ. Các thuộc tính như wheelY, panX, và panY được đặt để kích hoạt chức năng zoom và pan trên trục Y và trục X.

let valueAxis = mainPanel.yAxes.push(am5xy.ValueAxis.new(root, { renderer: am5xy.AxisRendererY.new(root, { pan: "zoom" }), extraMin: 0.5, tooltip: am5.Tooltip.new(root, {}), numberFormat: "#,###.00", extraTooltipPrecision: 2 }));: Dòng này tạo một trục giá trị (value axis) cho panel chính. Trục này được sử dụng để hiển thị các giá trị của dữ liệu giao dịch. Các thuộc tính được đặt để tùy chỉnh giao diện của trục, bao gồm định dạng số, và cung cấp thông tin chi tiết khi di chuột qua.

let dateAxis = mainPanel.xAxes.push(am5xy.GaplessDateAxis.new(root, { baseInterval: { timeUnit: "minute", count: 1 }, renderer: am5xy.AxisRendererX.new(root, { pan: "zoom", minorGridEnabled: true }), tooltip: am5.Tooltip.new(root, {}) }));: Dòng này tạo một trục thời gian (date axis) cho panel chính. Trục này được sử dụng để hiển thị thời gian trên trục x của biểu đồ. Thuộc tính được thiết lập để cấu hình trục thời gian, bao gồm đơn vị thời gian cơ bản (baseInterval), kiểu renderer và hiển thị lưới phụ (minor grid).

A screen shot of a computer code

Description automatically generated

let currentValueDataItem = valueAxis.createAxisRange(valueAxis.makeDataItem({ value: 0 }));: Dòng này tạo một đối tượng currentValueDataItem từ trục giá trị (valueAxis). Đối tượng này được tạo ra từ một phạm vi trên trục giá trị, được tạo từ một dữ liệu (makeDataItem) với giá trị ban đầu là 0.

let currentLabel = currentValueDataItem.get("label");: Dòng này lấy ra đối tượng nhãn (label) từ currentValueDataItem. Đối tượng nhãn này được sử dụng để hiển thị giá trị của dữ liệu trên biểu đồ.

if (currentLabel) { currentLabel.setAll({ fill: am5.color(0xffffff), background: am5.Rectangle.new(root, { fill: am5.color(0x000000) }) }) }: Dòng này kiểm tra xem có tồn tại nhãn (currentLabel) hay không. Nếu có, nó sẽ đặt thuộc tính fill và background của nhãn. Fill được đặt thành màu trắng (#ffffff), và background được đặt thành một hình chữ nhật đen (#000000) để làm nổi bật giá trị.

let currentGrid = currentValueDataItem.get("grid");: Dòng này lấy ra đối tượng lưới (grid) từ currentValueDataItem. Đối tượng lưới này thường được sử dụng để vẽ lưới trên trục giá trị.

if (currentGrid) { currentGrid.setAll({ strokeOpacity: 0.5, strokeDasharray: [2, 5] }) }: Dòng này kiểm tra xem có tồn tại lưới (currentGrid) hay không. Nếu có, nó sẽ đặt thuộc tính của lưới, trong đó có strokeOpacity (độ mờ của đường vẽ) và strokeDasharray (mảng độ đan xen giữa đường vẽ và khoảng trống).

let valueSeries = mainPanel.series.push(am5xy.CandlestickSeries.new(root, { ... }));: Dòng này tạo một loạt dữ liệu mới (series) cho panel chính. Trong trường hợp này, loại loạt dữ liệu là CandlestickSeries, được sử dụng để hiển thị dữ liệu giao dịch dưới dạng cây nến. Các thuộc tính được đặt để cấu hình loạt dữ liệu, bao gồm các trường dữ liệu (valueXField, valueYField, highValueYField, lowValueYField, openValueYField), và các cài đặt khác như xAxis, yAxis, legendValueText và legendRangeValueText.

Trong Candlestick chart (biểu đồ cây nến), các trường dữ liệu được đặt như sau:

valueXField: "Date": Đây là trường dữ liệu đại diện cho thời gian của mỗi cây nến trên trục x. Nó thường là thời gian của dữ liệu giao dịch.

valueYField: "Close": Trường này đại diện cho giá đóng cửa của cây nến, tức là giá cuối cùng của mỗi chu kỳ giao dịch.

highValueYField: "High": Đây là trường dữ liệu cho giá cao nhất trong chu kỳ giao dịch.

lowValueYField: "Low": Trường này đại diện cho giá thấp nhất trong chu kỳ giao dịch.

openValueYField: "Open": Trường này đại diện cho giá mở cửa của chu kỳ giao dịch.

Nếu giải thích theo kiến thức tiền ảo, thì:

Giá trị valueYField đại diện cho thân nến, tức là phần giữa của cây nến. Đây là giá cuối cùng mà thị trường giao dịch trong chu kỳ.

Các trường highValueYField, lowValueYField, và openValueYField tương ứng với đỉnh, đáy, và mở cửa của thân nến.

Đuôi nến (wick) là phần của cây nến mở rộng từ thân nến lên và xuống. Nó được biểu thị bởi sự khác biệt giữa giá cao nhất (highValueYField) và giá thấp nhất (lowValueYField) trong chu kỳ giao dịch.

A screenshot of a graph

Description automatically generated

A screen shot of a computer program

Description automatically generated

stockChart.set("stockSeries", valueSeries): Đây là cách để đặt loại series chính (main value series) cho biểu đồ stock. Trong trường hợp này, valueSeries là loại series chứa dữ liệu giá cổ phiếu (candlestick data).

let valueLegend = mainPanel.plotContainer.children.push(am5stock.StockLegend.new(root, { stockChart: stockChart })): Dòng này tạo một hình chú thích cho biểu đồ stock. Chú thích này hiển thị thông tin về các loại series trên biểu đồ, cho phép người dùng dễ dàng nhận biết mỗi series tương ứng với loại dữ liệu nào.

let volumeAxisRenderer = am5xy.AxisRendererY.new(root, {}): Tạo một renderer cho trục giá trị của dữ liệu khối lượng giao dịch (volume).

volumeAxisRenderer.labels.template.set("forceHidden", true): Ẩn các nhãn trên trục giá trị của khối lượng giao dịch.

volumeAxisRenderer.grid.template.set("forceHidden", true): Ẩn lưới trên trục giá trị của khối lượng giao dịch.

let volumeValueAxis = mainPanel.yAxes.push(am5xy.ValueAxis.new(root, { ... })): Tạo một trục giá trị mới cho dữ liệu khối lượng giao dịch và đưa nó vào panel chính của biểu đồ. Trục này sẽ hiển thị giá trị khối lượng giao dịch trên biểu đồ.

A screen shot of a computer program

Description automatically generated

A green and red rectangles

Description automatically generated

let volumeSeries = mainPanel.series.push(am5xy.ColumnSeries.new(root, { ... })): Dòng này tạo một loại series mới để hiển thị dữ liệu khối lượng giao dịch dưới dạng cột trên biểu đồ. Các cột này sẽ được đặt trên trục giá trị của khối lượng giao dịch.

volumeSeries.columns.template.setAll({ strokeOpacity: 0, fillOpacity: 0.5 }): Đây là cách để đặt các thuộc tính chung cho tất cả các cột trong loạt dữ liệu. Trong trường hợp này, các cột sẽ không có viền (strokeOpacity: 0) và có độ trong suốt là 0.5 (fillOpacity: 0.5).

volumeSeries.columns.template.adapters.add("fill", function (fill, target) { ... }): Dòng này thêm một adapter vào thuộc tính fill của mỗi cột trong loạt dữ liệu. Adapter này được sử dụng để tùy chỉnh màu sắc của cột dựa trên quy tắc của biểu đồ stock. Màu sắc được xác định bằng phương thức getVolumeColor của stockChart.

stockChart.set("volumeSeries", volumeSeries): Đặt loại series dữ liệu khối lượng giao dịch là volumeSeries cho biểu đồ stock.

valueLegend.data.setAll([valueSeries, volumeSeries]): Đặt dữ liệu cho hình chú thích (valueLegend). Trong trường hợp này, chú thích sẽ hiển thị thông tin về cả loạt dữ liệu giá và khối lượng giao dịch.

A screen shot of a computer program

Description automatically generated

mainPanel.set("cursor", am5xy.XYCursor.new(root, { ... })): Dòng này tạo một con trỏ trên biểu đồ, cho phép người dùng xem giá trị của dữ liệu khi di chuột qua. Con trỏ này được cài đặt để tự động chuyển đến dữ liệu của valueSeries gần nhất theo trục y.

let scrollbar = mainPanel.set("scrollbarX", am5xy.XYChartScrollbar.new(root, { ... })): Dòng này tạo một thanh cuộn cho biểu đồ, cho phép người dùng di chuyển trên biểu đồ một cách dễ dàng hơn. Trong trường hợp này, thanh cuộn sẽ được đặt ở phía dưới biểu đồ và có chiều cao là 50 pixels.

let sbDateAxis = scrollbar.chart.xAxes.push(am5xy.GaplessDateAxis.new(root, { ... })): Tạo một trục x cho biểu đồ thanh cuộn. Trục này sẽ hiển thị thời gian và được đặt theo các khoảng thời gian liên tục mà không có khoảng trống giữa chúng.

let sbValueAxis = scrollbar.chart.yAxes.push(am5xy.ValueAxis.new(root, { ... })): Tạo một trục y cho biểu đồ thanh cuộn. Trục này sẽ hiển thị giá trị số và được đặt theo các giá trị tuyến tính.

let sbSeries = scrollbar.chart.series.push(am5xy.LineSeries.new(root, { ... })): Thêm một loạt dữ liệu kiểu đường vào biểu đồ thanh cuộn. Trong trường hợp này, dữ liệu được liên kết với trục x và trục y tương ứng của thanh cuộn.

A screen shot of a computer program

Description automatically generated

sbSeries.fills.template.setAll({ ... }): Dòng này cài đặt các thuộc tính cho mẫu fill (màu nền) của các cột trong loạt dữ liệu sbSeries trên biểu đồ thanh cuộn. Trong trường hợp này, fill sẽ được hiển thị và có độ mờ là 0.3.

let seriesSwitcher = am5stock.SeriesTypeControl.new(root, { stockChart: stockChart }): Tạo một điều khiển loại loạt dữ liệu (series type control) cho biểu đồ. Điều khiển này cho phép người dùng chọn loại dữ liệu cụ thể mà họ muốn hiển thị trên biểu đồ. Đối số stockChart được chuyển vào để chỉ ra biểu đồ mà điều khiển này sẽ được áp dụng.

seriesSwitcher.events.on("selected", function (ev) { ... }): Dòng này xác định một sự kiện để lắng nghe việc chọn một loại dữ liệu cụ thể từ điều khiển loại loạt dữ liệu. Khi người dùng chọn một loại dữ liệu, hàm được truyền vào sẽ được gọi với đối số là sự kiện ev, trong đó có thông tin về loại dữ liệu được chọn.

if (ev.item instanceof Object && 'id' in ev.item) { ... }: Dòng này kiểm tra xem đối tượng được chọn có phải là một đối tượng và có thuộc tính id không. Nếu đúng, đoạn mã bên trong sẽ được thực thi. Điều này giúp đảm bảo rằng chỉ xử lý các đối tượng có thuộc tính id.

A black background with colorful text

Description automatically generated

function getNewSettings(series) { ... }: Đây là khai báo của một hàm JavaScript có tên là getNewSettings. Hàm này nhận một tham số là series, đại diện cho một loạt dữ liệu (series) trên biểu đồ.

let newSettings = [];: Dòng này khai báo một mảng rỗng có tên là newSettings. Mảng này sẽ chứa các cài đặt mới cho loạt dữ liệu.

am5.array.each([...], function (setting) { ... }): Đây là một vòng lặp qua mỗi phần tử trong mảng các chuỗi được định nghĩa trước đó. Mỗi phần tử trong mảng này đại diện cho một cài đặt cụ thể của loạt dữ liệu.

newSettings[setting] = series.get(setting);: Dòng này gán giá trị của cài đặt setting của loạt dữ liệu series vào một phần tử của mảng newSettings.

return newSettings;: Hàm này trả về mảng newSettings sau khi đã cập nhật các cài đặt mới của loạt dữ liệu.

A screen shot of a computer program

Description automatically generated

function setSeriesType(seriesType) { ... }: Đây là khai báo của một hàm JavaScript có tên là setSeriesType. Hàm này nhận một tham số là seriesType, đại diện cho loại của loạt dữ liệu mà người dùng muốn chuyển đổi.

let currentSeries = stockChart.get("stockSeries");: Dòng này lấy loạt dữ liệu hiện tại từ biểu đồ, sử dụng phương thức get của đối tượng stockChart.

let newSettings = getNewSettings(currentSeries);: Dòng này gọi hàm getNewSettings để lấy các cài đặt mới cho loạt dữ liệu, dựa trên các cài đặt của loạt dữ liệu hiện tại.

let data = currentSeries.data.values;: Dòng này lấy dữ liệu của loạt dữ liệu hiện tại để sử dụng lại sau khi loạt dữ liệu được thay đổi.

mainPanel.series.removeValue(currentSeries);: Dòng này loại bỏ loạt dữ liệu hiện tại khỏi bảng điều khiển chính của biểu đồ.

Trong khối switch (seriesType) { ... }, hàm xử lý tùy thuộc vào giá trị của seriesType, có thể là "line", "candlestick", "procandlestick", hoặc "ohlc". Dựa vào giá trị này, một loạt dữ liệu mới tương ứng sẽ được tạo và thêm vào bảng điều khiển chính của biểu đồ.

if (series) { ... }: Đây là một điều kiện kiểm tra xem loạt dữ liệu mới đã được tạo thành công hay không. Nếu có, loạt dữ liệu hiện tại sẽ được thay thế bằng loạt dữ liệu mới, dữ liệu cũ sẽ được cập nhật và bảng điều khiển chính cũng sẽ được cập nhật với loạt dữ liệu mới.

valueLegend.data.insertIndex(0, series);: Dòng này chèn loạt dữ liệu mới vào đầu danh sách của biểu đồ để đảm bảo nó hiển thị đầu tiên trên bảng điều khiển chính.

if ('clustered' in series) { series.clustered = false; }: Dòng này kiểm tra xem loạt dữ liệu mới có thuộc tính 'clustered' hay không. Nếu có, thuộc tính này sẽ được đặt thành false để loạt dữ liệu không được nhóm lại.

A screen shot of a computer program

Description automatically generated



let toolbar = am5stock.StockToolbar.new(root, { ... }): Dòng này tạo một thanh công cụ cho biểu đồ dạng cổ phiếu. Đối tượng toolbar được khởi tạo từ lớp am5stock.StockToolbar, và được liên kết với đối tượng root, là đối tượng cơ sở của biểu đồ.

{ container: document.getElementById("chartcontrols"), stockChart: stockChart, controls: [ ... ] }: Đây là đối số của hàm am5stock.StockToolbar.new(). Trong đó:

container: document.getElementById("chartcontrols"): Đây là phần tử HTML trong DOM mà thanh công cụ sẽ được gắn vào. Phần tử này có id là "chartcontrols".

stockChart: stockChart: Tham chiếu đến biểu đồ cổ phiếu được hiển thị bởi thanh công cụ.

controls: [ ... ]: Một mảng các điều khiển (controls) sẽ được thêm vào thanh công cụ.

Các điều khiển (controls) bao gồm:

am5stock.IndicatorControl.new(root, { ... }): Điều khiển chỉ số, cho phép người dùng chọn và hiển thị các chỉ số kỹ thuật trên biểu đồ. Nó được liên kết với biểu đồ và bảng điều khiển giá trị (valueLegend).

am5stock.DateRangeSelector.new(root, { ... }): Điều khiển lựa chọn phạm vi ngày, cho phép người dùng chọn khoảng thời gian hiển thị trên biểu đồ.

am5stock.PeriodSelector.new(root, { ... }): Điều khiển lựa chọn chu kỳ, cho phép người dùng chọn khoảng thời gian để hiển thị dữ liệu trong các chu kỳ khác nhau (ví dụ: 1 ngày, 1 tuần, 1 tháng).

seriesSwitcher: Điều khiển chuyển đổi loại loạt dữ liệu, cho phép người dùng chuyển đổi giữa các loại biểu đồ như đường, nến, hoặc ohlc.

am5stock.DrawingControl.new(root, { ... }): Điều khiển vẽ, cho phép người dùng vẽ các đường, hình, hoặc ký hiệu trên biểu đồ.

am5stock.ResetControl.new(root, { ... }): Điều khiển đặt lại, cho phép người dùng đặt lại phạm vi hiển thị và thu phóng biểu đồ về trạng thái ban đầu.

am5stock.SettingsControl.new(root, { ... }): Điều khiển cài đặt, cho phép người dùng tùy chỉnh các cài đặt của biểu đồ như màu sắc, kích thước và các cài đặt khác.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

let data = this.generateChartData(this.firstDate, this.value);: Dòng này tạo ra dữ liệu cho biểu đồ bằng cách gọi hàm generateChartData với tham số là this.firstDate (ngày đầu tiên) và this.value (giá trị ban đầu).

valueSeries.data.setAll(data);: Dòng này thiết lập dữ liệu cho loạt dữ liệu chính (valueSeries) trên biểu đồ từ dữ liệu đã được tạo.

volumeSeries.data.setAll(data);: Dòng này thiết lập dữ liệu cho loạt dữ liệu thể hiện khối lượng giao dịch (volumeSeries) trên biểu đồ từ dữ liệu đã được tạo.

sbSeries.data.setAll(data);: Dòng này thiết lập dữ liệu cho loạt dữ liệu sử dụng trong thanh cuộn ngang (sbSeries) trên biểu đồ từ dữ liệu đã được tạo.

this.dataSubscription = interval(1000).subscribe(() => { ... }: Dòng này thiết lập một đăng ký để thực hiện một hàm mỗi giây. Trong hàm này, dữ liệu mới được tạo và cập nhật để mô phỏng sự thay đổi giá của tiền ảo.

Trong hàm subscribe:

Lấy ra dữ liệu cuối cùng từ loạt dữ liệu chính (valueSeries) để cập nhật dữ liệu mới.

Kiểm tra xem thời gian hiện tại có khác so với thời gian trước đó ở mức phút hay không, nếu có, tạo một dữ liệu mới.

Nếu không, cập nhật dữ liệu cuối cùng với giá trị mới và cập nhật các giá trị khác như cao nhất, thấp nhất và khối lượng giao dịch.

Thực hiện các cập nhật liên quan đến hiển thị trên biểu đồ như cập nhật nhãn và màu sắc.

if (lastDataObject) { ... }: Kiểm tra xem có dữ liệu cuối cùng hay không. Nếu có, thực hiện các bước cập nhật.

let previousDate = lastDataObject.Date;: Lấy thời điểm của cây nến trước đó.

let previousValue = lastDataObject.Close;: Lấy giá đóng cửa của cây nến trước đó.

this.value = am5.math.round(previousValue + (Math.random() < 0.5 ? 1 : -1) \* Math.random() \* 2, 2);: Tạo một giá trị mới cho cây nến hiện tại bằng cách tăng hoặc giảm giá trị đóng cửa của cây nến trước đó một lượng ngẫu nhiên.

let high = lastDataObject.High;, let low = lastDataObject.Low;, let open = lastDataObject.Open;, let volume = lastDataObject.Volume;: Lấy các giá trị cao nhất, thấp nhất, mở cửa và khối lượng giao dịch của cây nến trước đó.

if (am5.time.checkChange(Date.now(), previousDate, "minute")) { ... }: Kiểm tra xem đã sang phút mới chưa. Nếu có, tạo một cây nến mới và thêm nó vào dữ liệu.

Trong trường hợp không, cập nhật các giá trị cao nhất, thấp nhất, mở cửa và đóng cửa của cây nến hiện tại.

**Quan trọng:**

A screen shot of a computer code

Description automatically generated

if (currentLabel) { ... }: Nếu có nhãn hiện tại, thực hiện các bước để cập nhật nhãn và màu nền của nhãn dựa trên giá trị mới.

currentValueDataItem.animate({ key: "value", to: this.value, duration: 500, easing: am5.ease.out(am5.ease.cubic) });: Thực hiện một hiệu ứng hoạt hình để cập nhật giá trị của nhãn tới giá trị mới của cây nến. Hiệu ứng này kéo dài 500 miligiây và sử dụng một hàm mượt mà để thực hiện chuyển đổi.

currentLabel.set("text", stockChart.getNumberFormatter().format(this.value));: Đặt văn bản của nhãn thành giá trị mới của cây nến, sau khi đã được định dạng bằng định dạng số của biểu đồ.

let bg = currentLabel.get("background");: Lấy nền của nhãn hiện tại.

if (bg) { ... }: Kiểm tra xem có nền không trước khi thực hiện các bước tiếp theo.

if (this.value < open) { bg.set("fill", root.interfaceColors.get("negative")); } else { bg.set("fill", root.interfaceColors.get("positive")); }: Nếu giá trị mới của cây nến nhỏ hơn giá mở cửa, đặt màu nền của nhãn thành màu tiêu cực (âm), ngược lại, đặt màu nền thành màu tích cực (dương).

+ Nếu giá đóng cửa của cây nến nhỏ hơn giá mở cửa, điều này có thể chỉ ra một xu hướng giảm giá. Trong trường hợp này, việc đặt màu nền của nhãn thành màu tiêu cực (âm) có thể được hiểu là một tín hiệu bán ra, tức là người dùng có thể muốn bán tài sản của họ hoặc không mua vào tại thời điểm đó.

+ Ngược lại, nếu giá đóng cửa của cây nến lớn hơn giá mở cửa, điều này có thể chỉ ra một xu hướng tăng giá. Trong trường hợp này, việc đặt màu nền của nhãn thành màu tích cực (dương) có thể được hiểu là một tín hiệu mua vào, tức là người dùng có thể muốn mua tài sản hoặc không bán ra tại thời điểm đó.

A graph with different colored squares

Description automatically generated

Các hàm như am5.math.round và am5.time.checkChange được sử dụng để làm tròn giá trị và kiểm tra thay đổi thời gian. Điều này không trực tiếp liên quan đến tiền ảo mà là cách xử lý dữ liệu trong biểu đồ.

A computer screen shot of a code

Description automatically generated

let chartData = [];: Khởi tạo một mảng rỗng để lưu trữ dữ liệu của các cây nến.

for (var i = 0; i < 50; i++) {: Bắt đầu vòng lặp for để tạo dữ liệu cho 50 cây nến.

let newDate = new Date(firstDate);: Tạo một đối tượng Date mới từ ngày đầu tiên được cung cấp.

newDate.setMinutes(newDate.getMinutes() - i);: Đặt thời điểm của newDate trở về thời điểm trước một số phút tương ứng với chỉ số vòng lặp i. Điều này tạo ra một chuỗi các thời điểm từ hiện tại đến quá khứ, mỗi cây nến cách nhau 1 phút.

value += Math.round((Math.random() < 0.49 ? 1 : -1) \* Math.random() \* 10);: Tăng giá trị value theo một số ngẫu nhiên từ -10 đến 10, để mô phỏng sự biến động giá.

let open = value + Math.round(Math.random() \* 16 - 8);: Tính giá mở cửa của cây nến, là giá trị hiện tại cộng với một số ngẫu nhiên từ -8 đến 8.

let low = Math.min(value, open) - Math.round(Math.random() \* 5);: Tính giá thấp nhất của cây nến, là giá trị thấp nhất giữa giá hiện tại và giá mở cửa, trừ đi một số ngẫu nhiên từ 0 đến 5.

let high = Math.max(value, open) + Math.round(Math.random() \* 5);: Tính giá cao nhất của cây nến, là giá trị cao nhất giữa giá hiện tại và giá mở cửa, cộng thêm một số ngẫu nhiên từ 0 đến 5.

let volume = Math.round(Math.random() \* 10000);: Tạo một giá trị ngẫu nhiên cho khối lượng giao dịch của cây nến, từ 0 đến 10000.

chartData.unshift({ ... });: Tạo một đối tượng đại diện cho cây nến hiện tại với các thông số về thời gian, giá đóng cửa, giá mở cửa, giá thấp nhất, giá cao nhất và khối lượng giao dịch. Sau đó, đối tượng này được đẩy vào đầu mảng chartData, để các cây nến được sắp xếp theo thứ tự thời gian từ mới nhất đến cũ nhất.

console.log(chartData);: Hiển thị dữ liệu của mảng chartData trong console, cho mục đích gỡ lỗi hoặc theo dõi.

return chartData;: Trả về mảng chartData chứa dữ liệu của các cây nến đã tạo.

**Biểu đồ hiển thị trên Web:**

**A screenshot of a graph

Description automatically generated**

**Biểu đồ hiển thị trên màn hình Mobile:**

**A screenshot of a screen

Description automatically generated**

1. **Các điểm chưa được giải quyết của Project**

Vấn đề còn lại chưa giải quyết được là đo được độ dài nến để từ đó lọc ra xu hướng phát cảnh báo cho người dùng

Data gọi từ api khi gán vào biểu đồ bị lỗi undefine

**Hết**