Trading Bot

mit Devisen- & News-Trading und Signalen

Projektdokumentation

Allgemein

Forex (FX) steht für Foreign Exchange (=Devisenhandel). Beim Trading mit Forex werden zwei Währungen gegeneinander gehandelt: Der Kauf einer Währung resultiert im gleichzeitigen Verkauf einer anderen.

Währungen werden immer paarweise gehandelt, weshalb man auch von Währungspaaren spricht. Der FX-Trading-Markt – der größte Finanzmarkt der Welt – wird häufig auch als Devisenmarkt oder Währungsmarkt bezeichnet.

Motivation:

Anfänglich verfolgten wir als Gruppe lediglich das Ziel den Anforderungen der Aufgabenstellung gerecht zu werden. Als nach einiger mühsamer Arbeit der erste “Feldversuch” gestartet wurde fiel jedoch die Ernüchterung groß aus.
Das Team hat einen großen Aufwand betrieben und alles was wir vorlegen konnten war ein Trading-Bot welcher nichts weiter konnte außer zuverlässig Verluste einzufahren.
Nun stellte sich uns die Frage, welche der von uns Benutzten Methode zum automatisierten Trading am besten funktioniert und ob unsere gewählten Methoden überhaupt sinnvoll sind, oder ob wir genauso gut einfach zufällig traden können. Also entschieden wir dazu einen weiteren Trader einzubauen, welcher zufällig traden sollte, und eine Logging Methode mit welcher man sich einen leichten Überblick darüber verschaffen kann, welche Methode am meisten Gewinn erwirtschaftet.

Quick Start Guide:

Voraussetzungen:

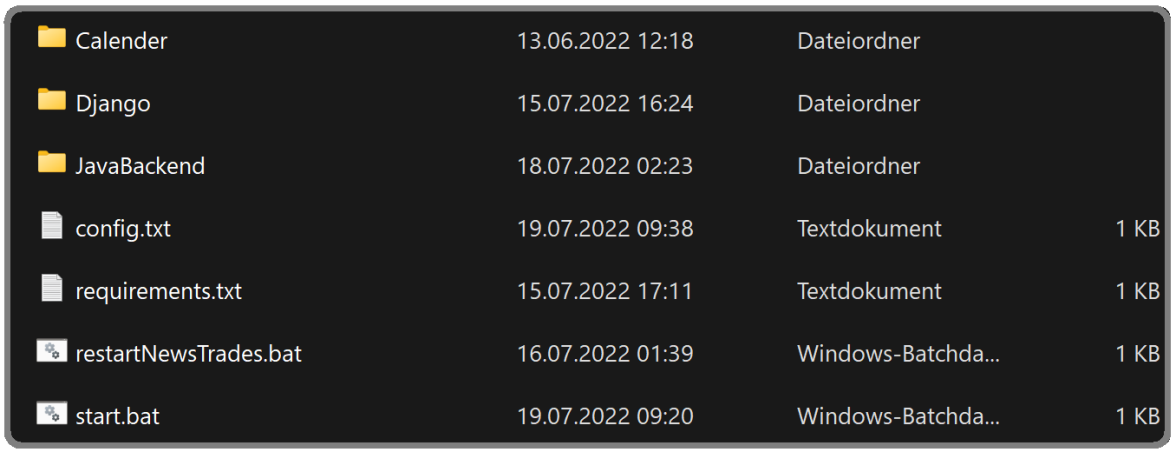
-python Interpreter mit support für python 3.10.5 oder höher
-java runtime environment Version 52 oder höher
-Um den Django Teil zum laufen zu bekommen gibt es dort noch eine extra Anleitung

Inbetriebnahme:

Bitte beachten sie zunächst dass die Ports 12000 und 12001 auf Ihrem PC nicht belegt sein dürfen.
Innerhalb des Projektordners befindet sich eine Bat Datei namens “start.bat”, mit welcher der Trading Bot gestartet werden kann. Es ist wichtig zu beachten, dass sich 3 Konsolen Fenster öffnen werden. Eines in welchem der der Hauptteil der Software in Java ausgeführt wird und ein weiteres in welchem eine Python Software läuft die für die beschaffung der Nachrichten für das News Based Trading zur Verfügung stellt. In der dritten Konsole läuft die Software, welche für das Loggen der Daten und der GUI zuständig ist.
Die eben genannte GUI lässt sich über localhost:8000 im Browser erreichen.

Des Weiteren befindet sich innerhalb des Projektordners eines Konfigurationsdatei mit folgenden Parametern:

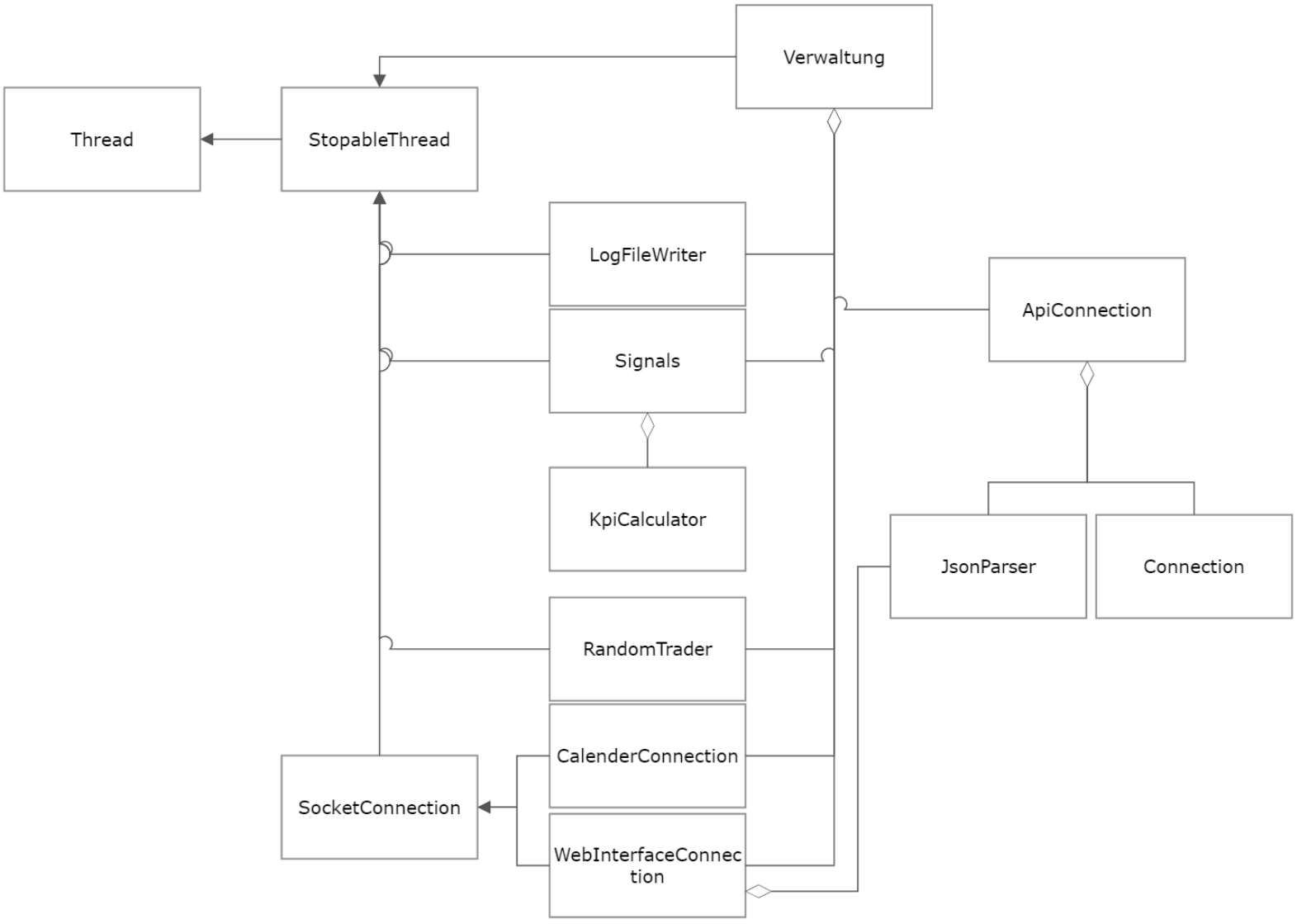
signal0	**=(true/false)**	aktiviert Hauptstrategie mit MACD, EMA und Parabolic SAR
signal1	**=(true/false)**	aktiviert Strategie mit MACD und Parabolic SAR
signal2	**=(true/false)**	aktiviert Strategie mit EMA und ATR
signal3	**=(true/false)**	aktiviert Strategie mit MACD und SMA
eventBased	**=(true/false)**	aktiviert das eventbasierte Trading
random	**=(true/false)**	aktiviert das zufällige Trading



Nach Start der Software wird auf User Input gewartet. Nun kann die Software über folgende Konsolenbefehle bedient werden:

1. “start” : Die Software startet das automatisierte Trading
2. “pause” : Die Software pausiert das automatisierte Trading
3. “log” : Aktiviert/Deaktiviert die Darstellung der Software Aktivität.
4. “trades” : Gibt einmalig alle Positionen und Trades in der Konsole aus
5. “close” : Hierdurch wird die Software beendet

Überblick über die Klassen Struktur:



Verwaltung (Daniel Harant)

Die Verwaltungsklasse gewährleistet das fehlerfreie Zusammenarbeiten der einzelnen Komponenten, aus welcher die Software besteht.

Verwaltung

Komponentenverwaltung	Kommunikationsverwaltung	Positionsverwaltung
Klassen: Signals WebInterfaceConnection CalendarConnection LogFileWriter RandomTrader Verwaltung	Aufgaben: kaufen, loggen und Informationsbereitstellung	Lokale überblick über die Positionen

Komponentenverwaltung

Die Verwaltung erbt die StoppableThread Klasse und überschreibt die onTick Methode um die Bedienung zu ermöglichen.

Um die Verwaltung zu aktivieren muss zwingend die runbot() Methode ausgeführt werden.

Zunächst fügt sie alle Komponenten in eine Liste(“threads”) ein. Hierbei können die Bestandteile der Liste jederzeit gestartet oder pausiert werden. Dies erfolgt durch die Methoden “startThreads” und “stopThreats”.Anschließend werden die Threads der Komponenten gestartet, welche dauerhaft aktiviert sein müssen.

Kommunikationsverwaltung

In diesem Teilgebiet der Verwaltung geht es um die Kommunikation zwischen den Signalen, dem Logging-System und der Connection zu den Oanda Servern.

Kaufen:

Hierbei unterscheidet man vier verschiedene Arten von Bestellungen, welche Klassen aufgeben können.

pushSignal()
Mit der Methode “pushSignal” kann eine Kpi* an die Verwaltungsklasse übergeben werden.

Zunächst wird überprüft ob das Instrument im Zusammenhang mit dem Signal bereits geblockt worden ist. Anschließend wird überprüft, ob noch genügend finanzielle Mittel auf dem Konto vorhanden sind. Wenn der Kontostand unter 100 Euro gefallen ist, werden keine neuen Trades zugelassen. Anhand der in der Kpi hinterlegten Werte, wird entschieden ob es sich um eine “long” oder “short” Position handelt, und wie viele Einheiten gekauft werden. Es wird an die API der Kaufbefehl mit den ermittelten Werten gesendet. Im folgenden wird überprüft ob dieser erfolgreich war.

Wenn erfolgreich: Wird dieses Instrument mit dem übergebenen Signal für weitere Käufe geblockt und das Signal wird geloggt.

Wenn gescheitert: In der Konsole wird der Grund des Scheiterns ausgegeben.

Zuletzt werden die lokale Liste der Positionen aktualisiert.

pushCalenderOrder()
Mit der Methode “pushCalenderOrder” kann eine Calender Order an die Verwaltungsklasse übergeben werden.

Der Ablauf ist beinahe identisch zur “pushSignal” Methode, jedoch werden hier noch die “take profit” und “stop loss” berechnet. Zusätzlich gibt es den Volatility Wert, welcher den Einsatz, “take profit und “stop loss” beeinflusst.

pushUpcomingEvent()

Der Ablauf ist ebenfalls sehr ähnlich zu der Methode “pushSignal”. Der signifikante Unterschied ist jedoch, dass hier zwei “limit orders” gesetzt werden, anstelle einer “market order”. Die eine leicht oberhalb, die andere leicht unterhalb des Kurses.

Blocken von Signalen

In der erklärung zu “pushSignals” wurde das Blocken von Signalen erwähnt. Um zu verhindern, dass der Bot zu häufig ein bestimmtes Instrument kauft, wird beim Trade durch ein Signal das entsprechende Instrument für dieses Signal geblockt. Erst wenn der aufgegebene Trade wieder verkauft worden ist, kann das Signal dieses Instrument wieder erwerben.

Bereitstellung von Informationen

Für alle Informationen, die andere Komponenten benötigen (Bspw. Instrumente, Candles, Kurs, etc.) stellt die Klasse Verwaltung die entsprechenden Methoden zur Verfügung.

Positionsmanagement

Für den lokalen Überblick über alle aktuellen Trades und Positionen werden diese in einer Liste gespeichert, welche über die Methode “aktualisierePositionen” aktualisiert werden kann.
Des Weiteren stellt die Verwaltung Methoden zur Verfügung um zu Überprüfen, ob wir ein bestimmtes Instrument besitzen und um die Positionen in der Konsole auszugeben.

Verwaltung
ApiConnection LogFileWriter Signals CalenderConnection WebInterfaceConnection randomTrader
+ void onTick() + Verwaltung(connection, randomConnection,granularity, einsatz) + JsonInstrumentsRoot + toggleLog() + startTraiding() + runBot() + startThreads() + stopThreads() + addThread(StopableThread st) + public boolean eneoughBalance() + pushUpcommingEvent(upcomingEvent) + pushCalenderOrder(calenderOrder) + pushOrder(order) + pushSignal(kpi) + pushRanomOrder(randomOrder) + placeOrder(instrument, units, takeProfit, stopLoss) - blockSignal(instrument, signal, id) - aktualisiereBlockedSignals() - signalIsBlocked(instrument,signal) :boolean - blockedSignalContainsSignal(String instrument, int signal) :boolean + getJsonCandlesRoot(count, instrument, from, to, price, granularity) + getKurs(instrument) + addManualPosition(instrument) - addPosition(a) + aktualisierePosition() + zusammenschluss() +containsPosition(String instrument): boolean +enthalten(trade t): boolean + printPositions() + toString()

Das Gerüst (Vartan Artyunyan)

Für die einwandfreie Operation des Trading Bots ist es für bestimmte Klassen notwendig bestimmte Methoden wiederholt aufzurufen. Diese Funktionalität stellt die Klasse “StopableThread” zur Verfügung.

StopableThread extends Thread
+ StopableThread() + StopableThread(tickrate) + start() + onStart() + onTick() + onTimer() + onClose() + setTimer(period) + timerIsSet() : boolean + stopThread()

Implementationen von Klassen die von “StopableThread” erben sollten die Methoden “onStart()”, “onTick()”, “onTimer()” und “onClose()” überschreiben um ihre Funktionalität zu realisieren.

Wird in einem Objekt der erbenden Klasse nun die Methode “start” aufgerufen so wird zunächst ein neuer Thread geöffnet.

Im genannten Thread wird als erstes die Methode “onStart” ausgeführt.

Anschließend wird wiederholt die Methode “onTick()” aufgerufen. Die Rate in welcher “onTick” aufgerufen wird, kann im Konstruktor in Form einer Integer welche die gewünschte Frequenz (Hz) repräsentiert festgelegt werden (Frequenz kann mindestens 10Hz und maximal 240Hz betragen). Standardmäßig beträgt die Tickrate 240Hz. (Es ist dann sinnvoll eine niedrigere Tickrate zu wählen wenn beispielsweise CPU Runtime gespart werden soll)

In einem Objekt der erbenden Klasse kann die Methode “setTimer(period)” aufgerufen werden. Dieser wird eine Zeitperiode in Millisekunden in Form eines Longs mitgegeben. Wurde die Methode in einem Objekt aufgerufen so wird dieses nach aufrufen der “start()” Methode, zusätzlich zu der “onTick()” Methode, die “onTimer” Methode regelmäßig aufrufen und zwar mit dem in “setTimer(period)” mitgegebene Zeitabstand zwischen den Aufrufen.

Wird in einem Objekt in welchem die “start()” Methode aufgerufen wurde, die “stopThread()” Methode aufgerufen, so wird zunächst das wiederholte aufrufen der “onTick()” und “onTimer()” Methoden beendet. Anschließen wird die “onClose()” Methode aufgerufen.

Nachdem die “onClose()” Methode vollständig ausgeführt wurde, wird der Thread beendet.

Kommunikation mit Oanda

ApiConnection
+ jsonParser: JsonParser + connection: Connection + availableInstruments: JsonInstrumentsRoot + canldeCache: CandleCache
+ ApiConnection(connection) + getTrades(): ArrayList<trade> + getTrade(id): trade + getJsonCandlesRoot(count,instrument,from,to,price,granularity): JsonCandlesRoot + getJsonInstrumentsRoot() + getKurs(intrument): double + placeOrder(instrument, units, takeProfit, stopLoss): OrderResponse + placeOrder(instrument, units): OrderResponse + placeLimitOrder(instrument, cancleTime, units, limit, takeProfit, stopLoss): OrderResponse + getBalance(): double + closePosition(instrument, units) + closeWholePosition(instrument)

Alle Methoden holen sich die, für die Methode notwendigen Informationen über eine Instanz der Connection Klasse, in Form eines Strings in Json Format. Anschließend nutzen sie den JsonParser um die gewünschten Informationenen aus dem Json String auszulesen, tragen die Information gegebenenfalls in ein entsprechendes Objekt ein und geben dies schließlich zurück.

Candles und Instrumente werden gecached um Zugriffszeiten zu verkürzen. (Vor allem im Falle der Candles konnten dank des Caches die Zugriffszeiten signifikant verkürzt werden, da die Notwendigkeit entfällt diese immer wieder neu zu parsen)

Die Connection mit den Oanda Servern

Connection
+ Connection() + Connection(token) - setAccountID() + getAccountIds(): String + getTrades(): String + getTrade(id): String + getInstruments(): String + getCandleStickData(count,instrument,from,to,price,granularity): String + placeOrder(requestJson) - GET(urlString) - POST(urlString, requestJson) - print(input) - getResponse(connection,debug)

Die oben genannten Methoden erstellen einen entsprechende REST Ressource und geben diese an die GET bzw POST Methode weiter. Diese erstellen zunächst eine Connection zu den Oanda Servern und liest anschließend mit der “getResponse(connection,debug)” Methode die Antwort vom Server aus.

Falls die Anfrage erfolgreich war wird wird die Antwort vom Server in Form eines Strings zurück gegeben.

Bei einer fehlerhaften Anfrage wird zunächst die Anfrage in die Konsole ausgegeben welche den Fehler verursacht hat, gefolgt vom Fehlercode und der Fehlermeldung welche vom Server zurückgegeben worden sind. Abschließend wird der String “” von der Methode zurückgegeben.

Die Kommunikation mit dem News Trader und dem Webinterface

Da ein Teil einige Mitglieder des Teams Ihren Teil der Software bevorzugt in Python implementieren wollten, war es notwendig eine Möglichkeit zu finden Informationen zwischen dem in Java geschriebenen Hauptteil der Software und den in Python geschriebenen News Trader und Webinterface auszutauschen.

Für die Umsetzung dieser Aufgabe haben wir uns für einen sehr simplen Austausch der Informationen über einen WebSocket entschieden. Die Klasse CalenderConnection horcht auf den Port 12000 auf Nachrichten vom NewsReader, parst die ankommenden Nachrichten und leitet sie an die Verwaltung weiter. Während die Klasse WebInterfaceConnection Nachrichten auf den Port 12001 für das WebInterface hinterlässt.

Der LogFileWriter

Ursprünglich nutzten wir den LogFileWriter um ausgehende Trades in einer CSV Datei zu loggen, da wir uns jedoch im laufe der Entwicklung dazu entschieden alle Aktivitäten lieber in einer Datenbank zu loggen und anschließend auf dem Webinterface anzuzeigen, musste der LogFileWriter für seine neue Aufgabe umfunktioniert werden.

Bei jeder ausgehenden Order leitet der LogFileWriter sowohl alle Informationen über die Order als auch die Ursache für die Order an das WebInterface weiter. Des Weiteren notiert sich der LogFileWriter die ID des durch die Order aufgemachten Trades. Dies hat den Hintergrund dass wir tracken wollen welche Trades bereits verkauft worden sind. Alle 2 Sekunden vergleicht der LogFileWriter die Liste der notierten TradeIDs mit der Liste der aktuell aktiven TradeIDs. Im Falle dass sich die IDs nicht decken wissen wir dass ein oder mehrer Trades verkauft worden sind. Nun fordert der LogFileWriter den realisierten Gewinn/Verlust aller verkauften Trades von der ApiConnection an und leitet diese Information an das WebInterface weiter damit diese Information in der Datenbank nachgetragen werden kann.

Die anwendungsspezifische Implementierung eines JsonParsers

Da wir mit der Performance des JsonParsers welchen wir ursprünglich für das Projekt verwenden wollten unzufrieden waren, haben wir uns dazu entscheiden einen eigenen JsonParser zu implementieren welcher in seiner Funktionalität zwar sehr stark eingeschränkt ist, jedoch eine signifikant bessere Performance aufweist. Je nach Systemkonfiguration konnte dank der custom Implementierung die Zeit welche für das Parsen der Candles notwendig war, von 1-2 Minuten auf wenige Sekunden, bzw von ein paar Sekunden auf den Bruchteil einer Sekunde reduziert werden. Aufgrund der Optimierungen ist der JsonParser jedoch ausschließlich für das parsen von Json Objekten die dem von Oanda gewählten Format entsprechen geeignet. Der vollständige Json Standard wird nicht unterstützt.

JsonParser
+convertApiStringToTradesArray(String json): ArrayList<trade> +makeOrderRequestJson(instrument, units, takePofit, stopLoss): String +makeOrderRequestJson(instrument, units): String +makeLimitOrderRequestJson(instrument, cancelTime, units, limit, takeProfit, stopLoss): String -truncate(input, places): double +getBalanceFromAccountJson(json): double +convertAPIStringToCandlesRootModel(json): JsonCandlesRoot +parseLastCandleFromAPIString(json): JsonCandlesCandle +convertAPIStringToInstrumentsRootModel(json): JsonUnstrumentsRoot +convertAPIStringToTrade(json): trade +makeOrderResponseFromJson(input): OrderResponse

Alle “convert” Methoden erstellen zunächst ein Objekt der Klasse JsonObject und geben im Konstruktor den mitgegebenen Json String mit. Das JsonObject erstellt intern aus der Json eine Dictionary aus dem dann die benötigten Werte ausgelesen werden können.

Alle “make” Methoden benutzen den JsonBuilder um aus den mitgegebenen Werten ein Json im passenden Format zu bauen.

JsonObject
+JsonObject() +JsonObject(Json) +contains(input): boolean +getObject(input): JsonObject +getArray(input): JsonArray +getValue(input): String +toString(): String

Gibt man im Konstruktor ein Json String mit so wird diese in ein Dictionary geparsed. Aus diesem können dann mittels der “get” Methoden die gewünschten Informationenen ausgelesen werden.

JsonBuilder
+JsonBuilder() +addValue(name, (String) value) +addValue(name, (double) value) +addValue(name, (int) value) +addValue(name, (boolean) value) -pushObject() -pushArray() -popObject(): boolean -popArray(): boolean +openObject(name) +closeObject() +openArray(name) +closeArray() +inArray() -removeLastChar() +build()

Erstellt man ein Objekt der Klasse JsonBuilder so können in das Object mit der Methode “addValue(name,value)” Variablen hinzugefügt werden. Ebenso können mit “openObject(name)” und “openArray(name)” Objekte und Arrays geöffnet werden.

Mit der Methode “build()”, wird ein String im Json Format zurückgegeben welcher alle eingefügten Variablen, Objekte und Arrays enthält. Es gilt zu beachten dass alle Objekte und Arrays wieder geschlossen werden müssen, andernfalls wird beim Aufruf der “build()” Methode eine invalidJsonOperation Exception geworfen. Der Json Builder ist in seiner Funktionalität ebenso stark eingeschränkt und beschränkt sich auf die Erstellung von Jsons in einem Format wie es unsere Software bzw für die Kommunikation mit Oanda nötig ist.

Kennzahlen (Niklas Meyer)

Das Programm benötigt Kennzahlen, um die gehandelten Finanzprodukte zu beurteilen. Der aktuelle Kauf- oder Verkaufspreis kann damit vom Programm als tendenziell günstig oder teuer eingestuft werden. Diese Kennzahlen werden in Kombination verwendet, um daraus Preistrends und Kaufsignale abzuleiten. Gleichzeitig werden Verkaufslimits nach unten und nach oben errechnet und geliefert. Diese werden, falls es zu einem Kauf kommt, gleich der Order als Stop-Loss und Take-Profit übergeben.

Die Kennzahlen berechnen sich durch mathematische Formeln. Die API des verwendeten Datenlieferanten Oanda liefert lediglich Daten zu Preisen in vorgegebenen Zeiträumen in einer vorgegebenen Granularität (Candles). Fertige Kennzahlen werden nicht geliefert, sondern müssen aus dem Zahlenmaterial errechnet werden. Es werden die nachfolgenden, in der Finanzwelt üblichen, Kennzahlen ermittelt.

Die Klasse KpiCalculator

Der Bereich Kennzahlen arbeitet vor allem mit der Klasse KpiCalculator, die Methoden zur Kennzahlenermittlung bereitstellt. Die meisten Methoden verwenden als Rückgabewert die Klasse Kpi, in der die Ergebniswerte gesammelt werden. Daneben gibt es Hilfsklassen, in denen die API-Daten, die im JSON-Format geliefert werden, in ein passendes Objekt aufnehmen werden können (https://freecodegenerators.com/code-converters/json-to-pojo).

KpiCalculator
- verwaltung: Verwaltung
+ KpiCalculator(Verwaltung) + getAll(String, String , int, Object…): Kpi + getBasicKpi(String, int, String, JsonCandlesRoot): Kpi + getParabolicSAR(String, String, int, double, double, double, JsonCandlesRoot): Kpi + getMACD(String, String, int, int, int, JsonCandlesRoot): Kpi + getAtr(String, int, String, JsonCandlesRoot): Kpi + getRSI(String, int, String, JsonCandlesRoot): Kpi + getSMA(String, int, String, JsonCandlesRoot): Kpi + getInitialKpi(String, Int, String, JsonCandlesRoot): Kpi + getCandles(String, String): JsonCandlesRoot + getInstruments(): JsonInstrumentsRoot - startDate(String): String

Wichtige Kennzahlen-Variablen aus der Klasse Kpi

Neben allgemeinen Kennzahlen wie höchster, niedrigster, durchschnittlicher und aktueller Preis (Methode getBasicKpi) werden die nachfolgenden spezielleren Kennzahlen errechnet:

Kennzahl	Beschreibung	Variablen
EMA Methode getBasicKpi	Der **E**xponential **M**oving **A**verage ist eine Art gleitender Durchschnitt, der den jüngsten Kursen in seiner Berechnung mehr Gewicht gibt.	**ema**: Double aktueller Wert **emas**: ArrayList<Double> alle Werte im Zeitraum
PSAR Methode getParabolicSAR	Der **P**arabolic **SAR** (Stop- and Reverse) ermöglicht eine schnelle Reaktion auf Kurstrends, indem er angibt, ob wir uns gerade in einen Bullenmarkt (Aufwärtstrend) oder einen Bärenmarkt (Abwärtstrend) befinden.	**parabolicSAR**: Double aktueller Wert **parabolicSARs**:ArrayList<Double> alle Werte im Zeitraum **trend**: String Entweder „bull“ oder „bear“ **trendWechsel**: boolean Bei Wechsel von bull auf bear oder umgekehrt true
MACD Methode getMACD	Der **M**oving **A**verage **C**onvergence-**D**ivergence ist eine wichtige Größe der technischen Analyse. Er bedient sich eines Diagramms zum Erkennen von Kauf- und Verkaufssignalen im Kursverlauf von Börsentiteln. Der getriggerte MACD ist die Signal-linie und wird aus dem arithmetischen Mittel der letzten MACD’s in der Periode berechnet. Die Signalintensität ist eine Eigen-kreation mit Werten von -1 (starker Abwärtstrend) bis 1 (starker Aufwärtstrend).	**macd**: Double aktueller Wert **macds**: ArrayList<Double> alle Werte im Zeitraum **macdTriggert**: Double aktueller Wert **macdsTriggert**: ArrayList<Double> alle Werte im Zeitraum **macdIntensity**: Double aktueller Wert **macdIntensitys**: ArrayList<Double> alle Werte im EMA-Zeitraum
RSI Methode getRSI	Der **R**elative **S**trength **I**ndex wird auch als Momentum Indikator bezeichnet und gibt das Verhältnis der Aufwärts- und Abwärtskraft an.	**rsi:** Double aktueller Wert **rsiListe**: ArrayList<Double> alle Werte im Zeitraum
ATR Methode getATR	Der **A**verage **T**rue **R**ange dient dazu die Volatilität eines Instruments zu messen. Je höher der Wert ausfällt desto wahrscheinlicher ist ein Trendwechsel. Der IntegerAtr wird zur Berechnung für den Stop Loss und Take Profit benötigt.	**atr:** Double aktueller Werte **atrListe:** ArrayList<Double> alle Werte im Zeitraum **IntegerAtr:** Integer aktueller Wert **IntegerAtrListe:** ArrayList<Integer> alle Werte im Zeitraum
SMA Methode getSMA	**S**imple **M**oving **A**verage Der SMA ist ein trend folgender Indikator und gibt das arithmetische Mittel in der letzten Periode an	**sma:** double aktueller Wert **smaListe:**ArrayList<Double> alle Werte im Zeitraum

Einstiegsmethode getAll()

Der Traiding-Bot greift auf Kennzahlen zu, indem die Methode getAll gerufen wird. In dieser werden zunächst die Candels gelesen und danach mit diesen die untergeordneten KPI-Methoden parallelisiert für die einzelnen Indikatoren aufgerufen. Wenn alle Ergebnisse vorhanden sind, werden sie in einem Kpi-Objekt vereinigt und an die Aufrufstelle returniert.
Alle KPI-Methoden sind public, da sie auch separat von außen gerufen werden können. Meist wird aber nur die Haupt-Methode getAll gerufen, die folgende Aufrufparameter kennt:

Parameter	**Verwendung**
instrument: String	Betrachtetes Finanzprodukt; Beispiel: EUR_USD
granularity: String	Kürzel der Periodenart. Gibt die Länge einer Candle an; Beispiele: W1 = 1 Woche, D = 1 Tag, H1 = 1 Stunde, M15 = 15 Minuten, M1 = 1 Minute, S5 = 5 Sekunden
emaperiods: int	Perioden in die Vergangenheit nur für die EMA-Kennzahl (Periodenlänge gemäß Granularität); Beispiel: 200 getBasicKpi Methode muss immer aufgerufen werden
signale: Object…	Mit diesem Parameter können beliebig viele Indikatoren mit unterschiedlichen Perioden aufgerufen werden Zunächst muss ein Kürzel des gewünschten Indikatoren eingeben werden (”macd”,”parabolicSAR”,“ema”,”rsi”,”atr”,”sma”). Gefolgt von den Eingabeparametern: Bei “ema”,”rsi”,”atr”,”sma”: Kürzel,periods, z.B. “ema”,14 Bei “macd”: Kürzel,periods,startBF, inkrementBF, maxBF z.B. “macd”,14,0.02,0.02,0.2 Bei parabolic SAR: Kürzel,periods,maxBF,macdX,macdY,macdZ z.B. ”parabolicSAR”,20,12,26,9
periods: int	Perioden in die Vergangenheit für jeweilige Kennzahlen (Periodenlänge gemäß Granularität); Beispiel: 14
startBF: double	Anfangswert des Beschleunigungsfaktors des Parabolic SARs; Beispiel: 0.02
inkrementBF: double	Schrittweite für den Beschleunigungsfaktor des Parabolic SARs; Beispiel: 0.02
maxBF: double	Maximalwert für den Beschleunigungsfaktor des Parabolic SARs; Beispiel: 0.2
macdX: int	Erste EMA-Periode des MACDs; Beispiel 12
macdY: int	Zweite EMA-Periode des MACDs; Beispiel 26
macdZ: int	Dritte EMA-Periode des MACDs; Beispiel 9

```
Kpi kpi=e.getAll(instrument.name, "M15", 200, "sma",20,"sma",50,"atr",14,"parabolicSAR",0.02,0.02,0.2,"macd",12,26,9);
```

Methode startDate

Die startDate Methode dient der Ermittlung eines Zeitpunkts, für den die erste Candle abgerufen wird. Hintergrund: Die Oanda-API erlaubt maximal 5000 Candles bei einem Abruf.

Trading Signale (Tom Schneider)

Anhand der Trading Signale wird entschieden, mit welchen Positionen gehandelt werden soll. Dabei haben wir uns entschieden eine Tradingstrategie als Hauptstrategie zu verwenden. Es handelt sich dabei um das Kombinieren dreier Trading Signale, dem MACD, EMA und Parabolic SAR. Anhand dieser Strategie wird die Funktionsweise im Folgenden erklärt.

Klasse Signals

Signals
+ Signals(Verwaltung verwaltung, LogFileWriter logFileWriter, String granularity) + onTick() + runSignals(String granularity) + ausgabe(String emaName, Kpi kpi, JsonInstrumentsInstrument instrument) + kombiniereMACDSMA(Kpi kpi): int + kombiniereMACDPSAR(Kpi kpi): int + kombiniereEMA200ATR(Kpi kpi): int + kombiniereMACDEMAPSAR(Kpi kpi): int + pruefeMACD(Kpi kpi): int + pruefeRSI(Kpi kpi): int + pruefeSMA(Kpi kpi): int + pruefeEMA200(Kpi kpi): int + pruefePSAR(Kpi kpi): int + pruefePerioden(Kpi kpi, String entscheideSignal, int anzahlVorperioden): int + pruefeATR(Kpi kpi: int + pruefeSMACrossover(Kpi kpi, int anzahlVorperioden): int

Die Prüfe-Methoden verwenden die Rohdaten aus der Kpi und berechnen die Vergleichswerte für die übergebenen Instrumente. Die Kombination der Signale aus der Hauptstrategie verwenden die pruefePerioden, pruefeEMA200 und pruefePSAR Methoden. Allgemein werden die Rohdaten mit den aktuellen Kursdaten in Relation gestellt und somit entschieden, ob das Signal positiv oder negativ ist, indem die Rückgabewerte angepasst werden.

Die Kombiniere-Methoden benutzen dann die Rückgabewerte der aufgerufenen Methoden und interpretieren, ob das Signal für die Verwendung in der Strategie geeignet ist.

Aufgerufen werden die Kombiniere-Methoden mit der Methode runSignals. Diese interpretiert die aufgerufenen Kombiniere-Methoden und entscheidet, ob eine Short oder Long Position gewählt wird. Außerdem kann unter anderem eine Signalstärke gewählt werden, die die Höhe des eingesetzten Kapitals definiert und ein Boolean gesetzt werden, das den ATR als Stoploss bzw. Takeprofit festlegt. Diese und weitere Werte werden dann der Verwaltung übergeben, die den Kauf abwickelt.

Prüfe- und Kombiniere-Methoden der Hauptstrategie

Methodenname	Beschreibung	Rückgabewert mit Interpretation
pruefePerioden	Durch Angabe des Strings “MACD” und der Anzahl an Perioden, die in der Vergangenheit geprüft werden sollen, wird der Verlauf von der MACD-Linie und der Trigger-Linie überprüft. Die MACD-Linie soll in den Vorperioden immer über bzw. unter der Trigger-Linie sein und in der aktuellen Periode die Trigger-Linie kreuzen oder gleichstellen.	**Fall 1:** die MACD-Linie ist in den Vorperioden unter der Trigger-Linie und der aktuelle MACD-Wert ist über bzw. gleich dem Trigger-Wert. Rückgabewert = -1 **Fall 2:** die MACD-Linie ist in den Vorperioden über der Triggerlinie und der aktuelle MACD-Wert ist unter bzw. gleich dem Trigger-Wert. Rückgabewert = 1 **Fall 3:** Die MACD- und Trigger-Linie kreuzen sich innerhalb der Vorperioden. Rückgabewert = 0 **Restliche Fälle:** Bei Fehlern ist der Rückgabewert = 99
pruefeEMA200	Überprüfung, ob der aktuelle Kurs über oder unter dem EMA der letzten 200 Perioden (auch Langzeittrend genannt) liegt.	**Fall 1:** Der aktuelle Kurs liegt über dem Langzeittrend. Rückgabewert = 1 **Fall 2:** Der aktuelle Kurs liegt unter dem Langzeittrend. Rückgabewert = -1 **Fall 3:** Der aktuelle Kurs ist gleich dem Langzeittrend. Rückgabewert = 0
pruefePSAR	Überprüfung, ob der aktuelle Kurs über oder unter dem Parabolic SAR liegt.	**Fall 1:** Der aktuelle Kurs liegt über dem Parabolic SAR. Rückgabewert = 1 **Fall 2:** Der aktuelle Kurs liegt unter dem Parabolic SAR. Rückgabewert = -1 **Fall 3:** Der aktuelle Kurs ist gleich dem Parabolic SAR. Rückgabewert = 0
kombiniereMACDEMAPSAR	Anhand der Rückgabewerte der Prüfe-Methoden soll entschieden werden, ob eine Short oder Long Position oder gar keine Kaufentscheidung gewählt werden soll. **Fall 1**, Short Position: Rückgabewert von pruefeEMA200 = -1, pruefePerioden = 1, pruefePSAR = 1. **Fall 2**, Long Position: Rückgabewert von pruefeEMA200 = 1, pruefePerioden = -1, pruefePSAR = -1. **Restliche Fälle**: Abweichungen in oben beschriebenen Rückgabewerten	**zu Fall 1:** Rückgabewert = -1 **zu Fall 2:** Rückgabewert = 1 **zu den Restlichen Fällen:** Rückgabewert = 0

Klasse Kpi

Kpi
+ Kennzahlenattribute (siehe Wichtige Kennzahlen-Variablen aus der Klasse Kpi) + signalTyp: int + ATRMultiplierSL: int = 2 + ATRMultiplierTP: int = 3 + logShort: boolean + signalStrenght: double = 1 + root: JsonCandlesRoot + useATRAsSLTP: boolean
+ Getter & Setter der Kennzahlenattribute + Kpi(String instrument, String granularity, int periods) + Kpi(String instrument, String granularity) + hashCode(): int + equals(Object o): boolean + resetKpiElements(Kpi kpi, String… werte): Kpi + runden(double wert, int n): double + aufrunden(double wert, int n):double + abrunden(dobule wert, int n): double + getLimitPrice(): double + getStopLoss(): double + getTakeProfit(): double + getLongTakeProfit(): double + getLongStopLoss(): double + getShortStopLoss(): double + getShortTakeProfit(): double + convertIntegerATRInDouble(boolean plusMinus): double + getLongStopLossATR(): double + getLongTakeProfitATR(): double + getShortStopLossATR(): double + getShortTakeProfitATR(): dobule + checkPrecision(double wert, boolean aufrunden): double + getUnitPrice(KpiCalculator calculator): double + compareTo(Kpi wert2): int

In der Klasse Kpi werden unter anderem die kalkulierten Werte aus der Klasse KpiCalculator gespeichert, damit diese dann in der Signals-Klasse verwendet werden können. Außerdem werden hier die Stoploss- bzw. Takeprofit-Werte berechnet. Dies geschieht, in dem der aktuelle Preis mit einem Multiplikator multipliziert wird. Falls gewünscht und in von der Signals-Klasse übergeben, kann der ATR auch als Stoploss bzw. Takeprofit gewählt werden. Je nachdem mit welcher Währung getradet werden soll werden mit der checkPrecision Methode die Werte auf 3 oder 5 Stellen auf- oder abgerundet, um immer eine ausreichende Genauigkeit für die übergebenen Ausstiegswerte zu bekommen.

JUnit-Tests in der Klasse KpiTesting (Niklas Meyer & Tom Schneider)

In der Klasse gibt es Testmethoden für die einzelnen Kennzahlen und zu den Kombiniere- und Prüfe-Methoden. Um alle Tests mit den gleichen Candles und für alle Instrumente vom Typ “CURRENCY” durchzuführen, wird in der Methode beforeAll parallelisiert die getAll Methode aufgerufen. Dies bewirkt insbesondere eine deutliche Laufzeitverbesserung. Um in der Testmethode getKpiCheck beweisen zu können, dass die Werte einer Untermethode (am Beispiel getBasicKpi) identisch mit der Methode getAll sind, wird auch die Methode getBasicKpi zum gleichen Zeitpunkt aufgerufen. Einerseits wird getestet ob die Kennzahlen-Variablen realistische Werte annehmen. Andererseits werden die Ausgabewerte der Kombiniere- und Prüfe-Methoden auf ihre Gültigkeit getestet um sicherzustellen, dass keine falschen Parameter zurückgegeben werden.

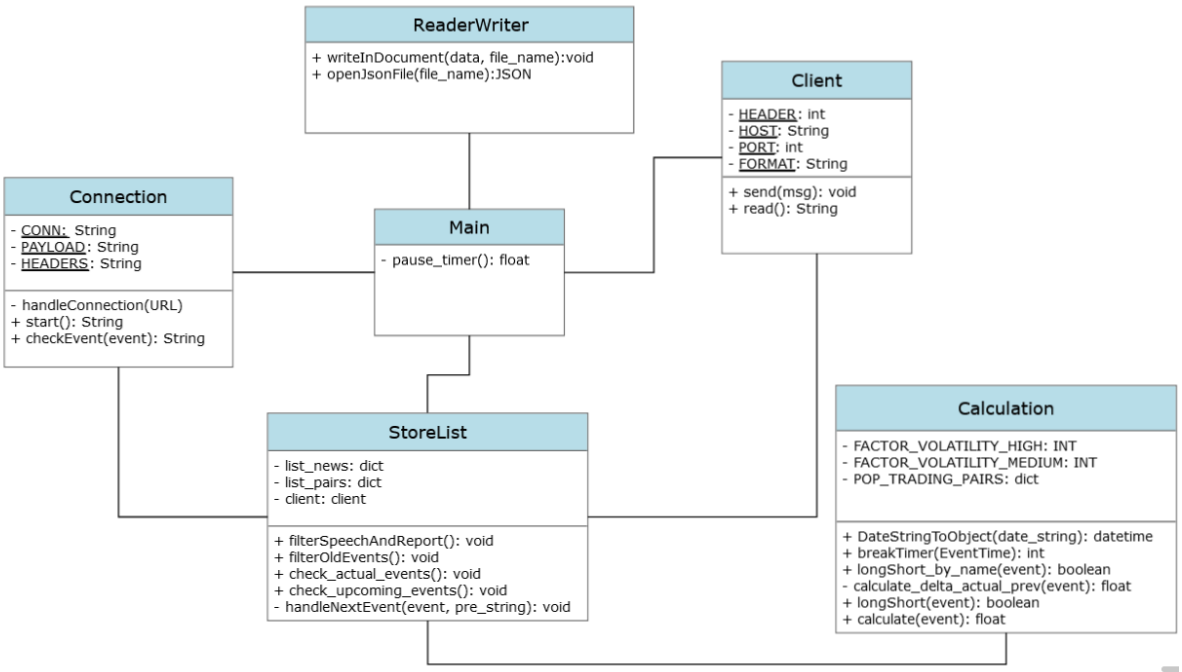
```
currenciesString.parallelStream().sorted().forEach(k->{currencies.add(werte.getAll(k, 200, 14, "M15", 0.02, 0.02, 0.2, 12, 26, 9));
basicKpiList.add(werte.getBasisKpi(k, 200, "M15",werte.getCandles(k, "M15")));});
```

Wirtschaftskalender / News Trading (Tim Heyder)

Das News Trading ist eine essenzielle Investmentstrategie beim Handeln mit Währungspaaren (Devisen). Im Gegensatz zur signalbasierten Chartanalyse werden Trading - Entscheidungen aufgrund aktueller oder erwartender Nachrichten getroffen. Dabei hängt es von der eigenen Strategie ab, ob man Signale mit einbeziehen möchte.

Eine Sammlung der zukünftigen und vergangenen News findet sich dabei in sogenannten Wirtschaftskalendern (engl. Economic Calendar). Diese sind in der Regel frei zugänglich zu finden und stellen die Informationen übersichtlich da. In diesen finden sich sowohl die Namen der Events, Prognosen, vergangene Werte als auch Konsens (Erwartungen). Aktuelle Werte werden bei Veröffentlichung den restlichen Infos hinzugefügt.

Ziel des Moduls ist es die Aktualität der Nachrichten zu überprüfen und sie, in Form von verarbeiteten Trades, automatisch an das Hauptprogramm weiterzugeben.



Main

Die Main-Methode wird zum Starten des Moduls ausgeführt und sollte erst nach dem Server (innerhalb der Verwaltung) erfolgen, um so eine problemlose Verbindung zu gewährleisten.

Ihr Zweck ist

1. Aufbau einer Verbindung mit der Verwaltung über den Port 12000
2. Entgegennehmen der verfügbaren Handelspaare über aktive Verbindung
3. Auslesen aller News des aktuellen Tages + Abspeichern als lokale Datei
4. Handelspaare, News und Client in StoreList speichern
5. Dauerhaft aktiv bis Ende aller Events:
    a. filtern nach alten (vergangenen) Nachrichten
    b. Check: aktualisierte, aktuellen Werte
    c. Check: Upcoming News innerhalb der nächsten 10min
6. Pausieren bis zum nächsten Tag 0:00 Uhr

Die Datei mit allen Events des aktuellen Tages wird im JSON-Format gespeichert. So können die Trades besser nachvollzogen und die Gewichtung der Trades angepasst werden.

StoreList

Beim Initialisieren der Klasse werden die Liste der aktuellen News und der angebotenen Handelspaare, zusammen mit dem Client-Objekt, zwischengespeichert. Es kann so eine einfachere Handhabung und Aufruf der Methoden gewährleistet werden.

filterSpeechAndReport(): Da Berichte und Reden über keine aktuellen Werte verfügen mussten sie ausgefiltert werden. Konnte durch filterOldEvents() abgelöst werden, um sie auch bei check_upcoming_events() zu berücksichtigen.

filterOldEvents(): Prüfen ob die Zeit der Events abgelaufen ist. Noch nicht aktualisierte Werte sollen erhalten bleiben, da sich die Veröffentlichung teilweise über Stunden verzögern kann. Zudem werden abgelaufene Berichte und Reden herausgefiltert.

check_actual_events(): Prüfen nach aktualisierten, aktuellen Werten über die Connection-Klasse

check_upcoming_events(): Prüfen nach Nachrichten innerhalb der nächsten 10min

Connection

Die Connection handhabt alle Abfragen des Wirtschaftskalenders auf FXStreet („https://www.fxstreet.com“). Die angebotene API ist nur über ein kostenpflichtiges Token aufrufbar. Um die Paywall zu umgehen war ein Workaround notwendig. Als Header simulieren wir so eine Anfrage als Browser, in diesem Fall „Firefox“, um uns eine GET-Request an den Server zu senden. Über die Manipulation der URL lassen sich Filter erzielen. (Filtermöglichkeiten: „https://calendar-api.fxstreet.com/swagger/index.html“)
Die Response erfolgt dann als JSON-String.

start(): Über die Manipulation der URL wird das heute Datum und der Zeitraum gesendet, um die News des kompletten Tages zu erhalten. Zudem wird lediglich nach hohen und mittelmäßigen Auswirkungen gefiltert, da diese den höchsten Einfluss auf den Forex-Markt haben.

checkEvent(): Abfrage einzelnen Events anhand der ID. Die Antwort sind die kompletten Informationen (z.b Ist-Wert, Prognose, Auswirkung), die später nach einem aktuellen Wert überprüft werden.

Client

Der Aufbau einer Verbindung mit der Verwaltung erfolgt über “localhost” und den Port 12000. Dieser darf für einen reibungslosen Ablauf nicht belegt sein.

send(message): Senden einer vorformatierten Nachricht an den Server im richtigen Encoding. Es wird das JSON - Format zur Übermittlung verwendet. Zulässig sind nur:

- {“upcoming” : {"name" : eventName, "countryCode" :country_code, "instrument" : Instrument, "volatility ": volatility, "time" : time}
- {“order” : {"name" : eventName, "countryCode" : country_code, "instrument" : Instrument , "volatility" :volatility, "factor" : factor, "longShort" : longShort}

Placeholder werden zuvor in StoreList ergänzt. Das Auslesen erfolgt dann im eigen-implementierten JSON-Parser.

read(): auslesen der im Puffer befindlichen Nachricht. Wird benutzt, um die Handelspaare bei Start auszulesen.

Calculation

Die Klasse „Calculation“ ist das Herzstück für die Bestimmung des Faktors. Dieser soll die Gewichtung der Nachricht festlegen und die damit eingesetzte Geldmenge für den späteren Trade. Zudem findet hier die Bestimmung von Long oder Short statt. Die Berechnung ist nach persönlichen Präferenzen anpassbar und kann ausgetauscht werden. Möglich ist auch den Überraschungswert mit einzubeziehen oder bei Handel mit populären Handelspaaren, die mit einem höheren Volumen getradet werden, den Faktor zu verstärken. Außerdem sind hier auch noch weitere Methoden, die für die Zeitbestimmung benötigt werden

DateStringToObect(date_string): Rückgabe des formatierten Date-Strings als datetime-Objekt

breakTimer(EventTime): Rückgabe der Differenz von Zeit des Events und aktueller Zeit

longShort_by_name(event): Kann noch für die Bestimmung für Long/Short oder des Faktors benötigt werden. Es zeigte sich bei der Analyse teilweise atypisches Kursverhalten, dieses kann genauer betrachten werden umso noch besser Vorhersagen treffen zu können.

calculate_delta_actual_prev(event): Rückgabe der Vorher-Nachher-Differenz

longShort(event): Bestimmen ob die Nachrichten einen negativen oder positiven Einfluss auf den Kurs hat. Rückgabewerte:

- True: Bei positivem Einfluss auf die Währung soll eine Long-Position getradet werden
- False: Bei negativem Einfluss resultiert das in einer Short-Position.

calculate(event): Liefert als Return-Wert den Faktor zur Gewichtung. Daraus resultiert die eingesetzte Geldmenge im Trade

ReaderWriter

writeInDocument(data, file_name): Schreiben von Daten in File und speichern als übergebenen Dateinamen. Dient zum Abspeichern der täglichen, relevanten Nachrichten

openJsonFile(file_name): Öffnen einer File und Rückgabe als JSON-Objekt

Django (Samuel Rajabi)

Überblick:

Um die getradeten Elemente visualisieren zu können, die Daten in einer Datenbank abspeichern zu können und einen Überblick über das Projekt im Allgemeinen zu geben haben wir uns entschieden eine Website zu machen. Hierzu haben wir das Framework Django genutzt.

Um den Django Teil des Projekts zum Laufen zu bringen, installieren Sie bitte die in der Datei „requirements.txt“ enthaltenen Dependencies. Dies können Sie mit Hilfe des folgenden Befehls tun:

```
pip install -r requirements.txt
```

Um die Website zu starten, wechseln Sie in den Folder „Django/TradingBot“ und schreiben Sie den folgenden Command in das Terminal:

```
python manage.py runserver
```

Nun geben Sie folgende URL in einen Browser ihrer Wahl ein, um die Seite aufzurufen:

```
http://127.0.0.1:8000/
```

Aufbau:

- Website
  - Generelles
  - Daten Visualisieren
- Datenbank
  - Generelles
  - Oanda Anbindung
  - Anbindung an Java

Webseite

Generelles

Die Webseite besteht aus der Homepage, in der das Projekt kurz beschrieben wird, ein Überblick über die allgemeine Performance des Bots gegeben wird und der Arbeitsbereich der Beteiligten kurz geschildert wird.

Neben der Hauptseite gibt es zu den vier Teilen, nach denen getradet wird, jeweils einen eigenen Abschnitt, in dem sie kurz beschrieben werden und die offenen Positionen abgebildet werden. Der Link zu den Abschnitten befindet sich im Demo Bereich der Homepage.

Aufbau

- Homepage
    - Home
    - Bot Beschreibung
    - Demo
    - Beteiligte Personen
- Signal Seite
    - Offene Positionen
- Calendar Seite
    - Offene Positionen
- Upcoming Seite
    - Offene Positionen
- Random Seite
    - Offene Positionen

Daten Visualisieren:

Die Daten, welche für die Visualisierung benötigt werden, befinden sich in der „SQLite“ Datenbank. Die Visualisierung erfolgt mithilfe der Python Bibliothek „Matplotlib“.

Datenbank:

Generelles:

Bei der Datenbank handelt es sich um „SQLite“.
Die Datenbank ist im Framework „Django” automatisch integriert und muss nicht separat installiert werden.

Daten in der Datenbank:

- Aktueller Geldbestand (Daten von Oanda)
- Signale (Daten von Java)
- Calendar (Daten von Java)
- Random (Daten von Java)
- Upcoming (Daten von Java)

Oanda Anbindung:

Um den aktuellen Kontostand zu bekommen werden regelmäßig get requests zur Oanda API erzeugt. Dies erfolgt durch die Eingabe des Commands:

```
python manage.py updateDatabase
```

Nach der Ausführung wird der Kontostand in die Datenbank abgespeichert und auf der Homepage in der Grafik mit abgebildet.

Anbindung an Java:

Um die aktuellen getradeten Daten zu bekommen wartet die Datei „client.py” im Ordner „Django/TradingBot/trading_B” permanent darauf Werte von der Java Seite zu bekommen.
Die Werte können normale Signale, getradete News und Random Signale sein.
Die Werte werden dann in der Textdatei „basedata.txt” zwischengespeichert.
Durch die Eingabe des Commands

```
python manage.py updateDatabase
```

werden die Daten aus der Textdatei gelesen, in der Datenbank abgespeichert und die Daten in der Textdatei anschließend gelöscht.

How to get rich quick!!!
“This is totally not a pyramid sceme” - Tim Heyder

Tom Schneider - “Trust me bro, you have to buy Bitcoin bro, it’s gonna be big bro”

“NFTs are the Future” - Niklas Meyer

Samuel Rajabi - “You have the wrong mindset”

BANKS HATE THIS TRICK!!!

Learn how to make 100k+ a week with our Trading Bot
“If u are not rich yet u are making it wrong” - Daniel Harant

Vartan Artyunyan - “Just earn more money, it isn't that difficult”