Trading Bot

Contents

Einleitung 1
Vorgaben 1
Funktionen: 1
Nice to have: 1
Planen 2
Zeitplan 2
Use-Case Diagramm 3
Klassen Diagramm 4
Kontrollieren 5
Testbedingungen 5
Testprotokoll 5
Auswerten 5
Erreichen der Ziele 5
Teamarbeit 6
Probleme 6
Zeitplan 7

Einleitung

In einer Zweierarbeit soll ein Projekt realisiert werden, welches unsere Fähigkeiten erweitert und wir unser Wissen praktisch anwenden können. Als Projekt wählten wir einen Tradingbot, dieser sollte anhand der realen Marktdaten und einer Strategie die man ihm übergeben kann, mit Aktien handeln können. Uns war es dabei wichtig das wir sehr viel Profit machen können, zu Beginn sagten wir uns; "Wir gehen hier nicht unter 200 Millionen raus!".

Vorgaben

Funktionen:

- Aktuelle Daten vom Markt holen
- Daten anhand einer ihm zuweisbaren Strategie verwerten
- Orders öffnen und schliessen können
- UI mit dem man den Bot kontrollieren kann (Einstellungen ändern, Bot starten/stoppen)
- Paperaccount um Bot zu testen

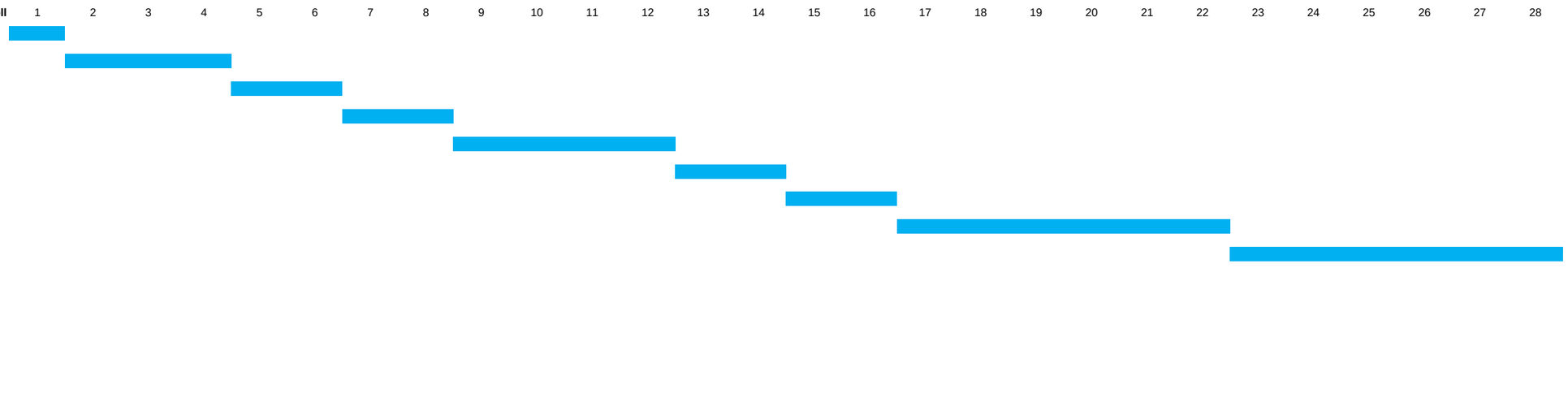
Nice to have:

- UI um die Übersicht über die Aktivitäten des Bots zu beobachten
- Möglichst geringe Verzögerung beim Datenfeed (< 5min)
- Der Bot soll anhand des Markts die beste Strategie selber wählen
- Viele Strategieoptionen
- Business Trading Account

Planen

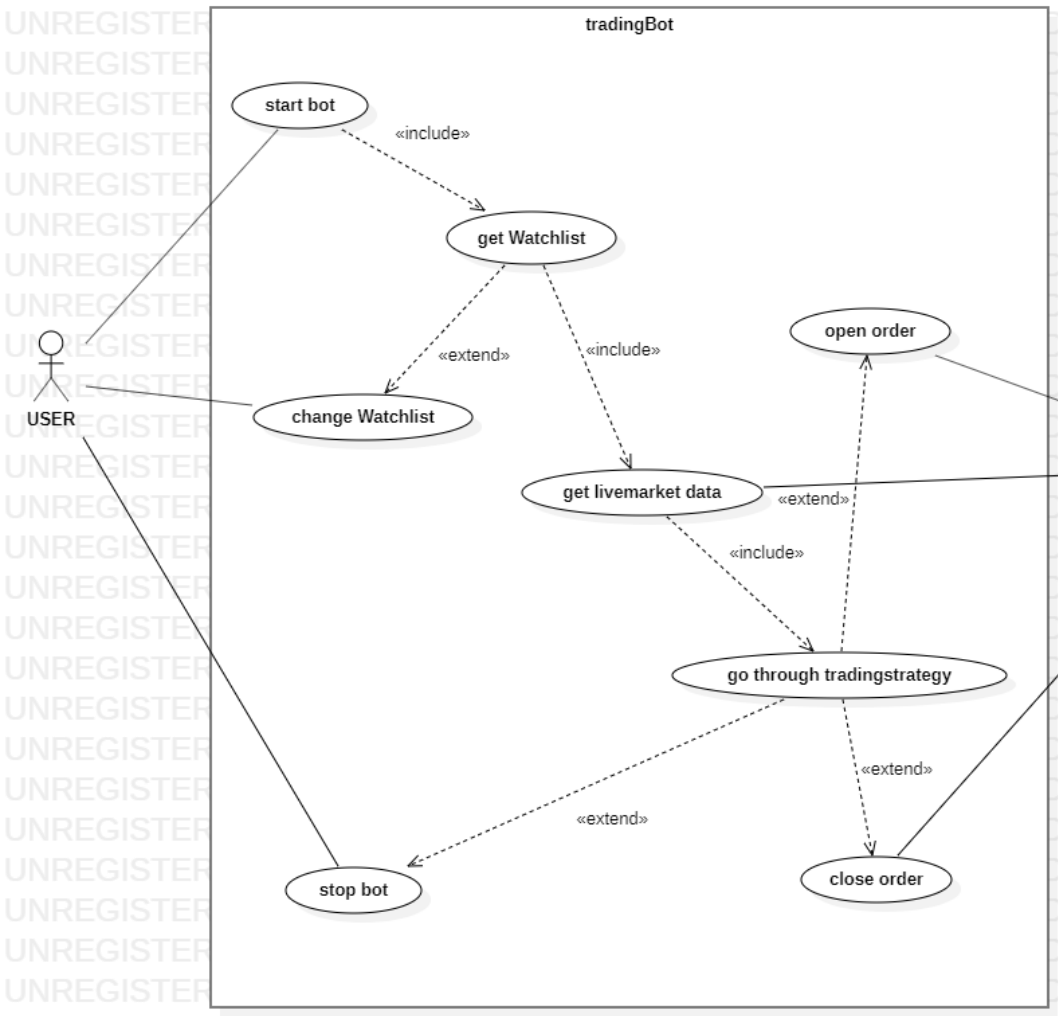
Zeitplan

IPERKA	Nr.	Arbeitspaket	Dauer (h)	Ist/Soll
I	1	zum Auftrag informieren	1	Soll
				Ist
	2	Idee finden und Umsetzbarkeit überprüfen	3	Soll
				Ist
P	3	Zeitplan schreiben	2	Soll
				Ist
	4	Use Case Diagramm erstellen	2	Soll
				Ist
	6	Klassendiagramm erstellen	4	Soll
				Ist
	8	API finden und implementieren	2	Soll
				Ist
	9	Grundstruktur erstellen	2	Soll
				Ist
	10	TradingAPI und getMarketData realisieren	6	Soll
				Ist
R	11	Tradingbot realisieren	8	Soll
				Ist
	12	Strategie realisieren	4	Soll
				Ist
	13	Edgecases die Fehler verursachen abfangen	4	Soll
				Ist
	14	Maincontroller realisieren	8	Soll
				Ist
K	15	Tests realisieren	6	Soll
				Ist
A	16	Dokumentation beenden	4	Soll
				Ist

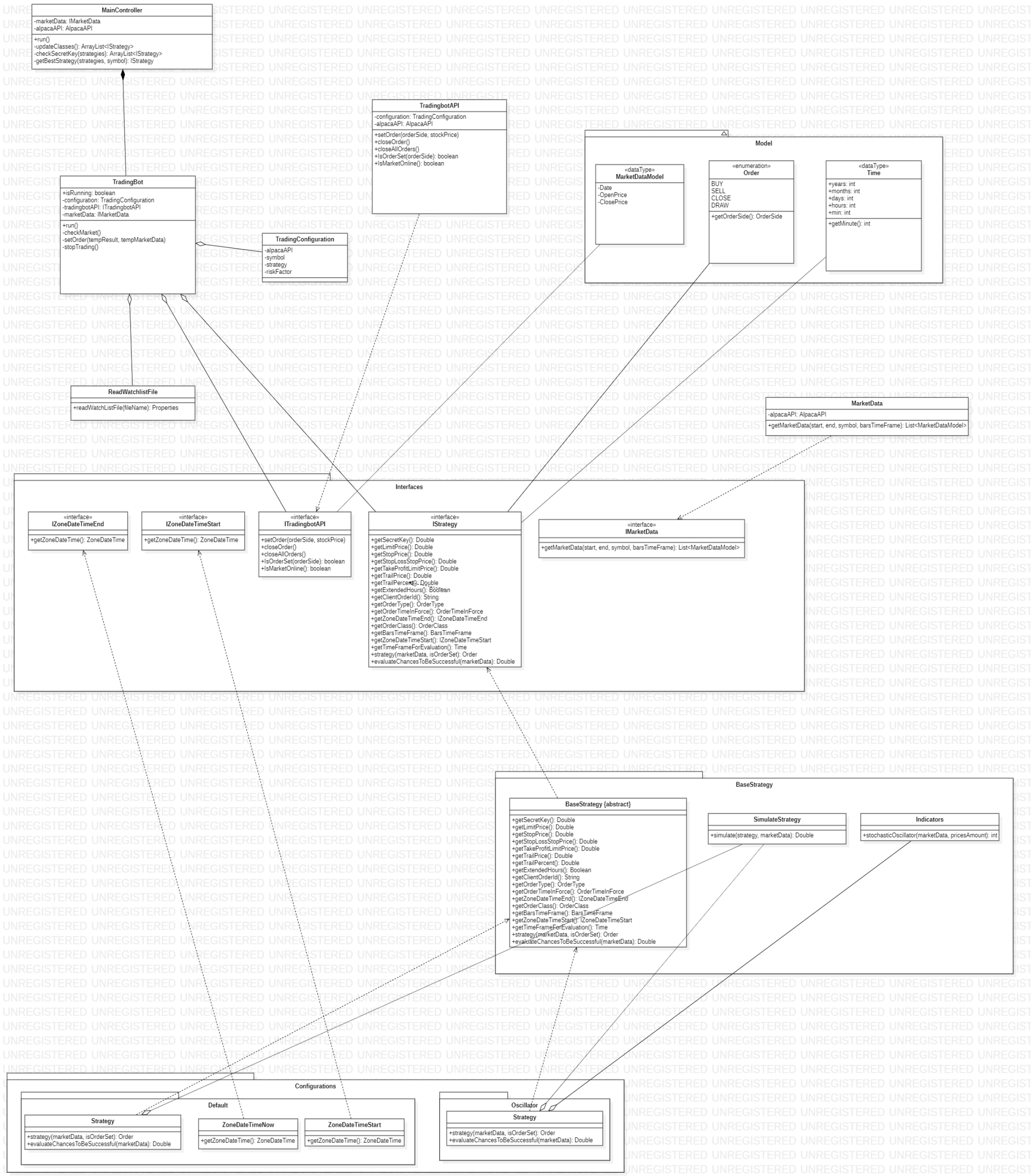


IPERKA	Nr.	Arbeitspaket	Dauer (h)	Ist/Soll
I	1	zum Auftrag informieren	1	Soll
				Ist
	2	Idee finden und Umsetzbarkeit überprüfen	3	Soll
				Ist
P	3	Zeitplan schreiben	2	Soll
				Ist
	4	Use Case Diagramm erstellen	2	Soll
				Ist
	6	Klassendiagramm erstellen	4	Soll
				Ist
	8	API finden und implementieren	2	Soll
				Ist
	9	Grundstruktur erstellen	2	Soll
				Ist
	10	TradingAPI und getMarketData realisieren	6	Soll
				Ist
R	11	Tradingbot realisieren	8	Soll
				Ist
	12	Strategie realisieren	4	Soll
				Ist
	13	Edgecases die Fehler verursachen abfangen	4	Soll
				Ist
	14	Maincontroller realisieren	8	Soll
				Ist
K	15	Tests realisieren	6	Soll
				Ist
A	16	Dokumentation beenden	4	Soll
				Ist

Use-Case Diagramm



Klassen Diagramm



Kontrollieren

Testbedingungen

Tester	David Oberkalmsteiner
Datum	24. Jan. 2021
Testgerät	OST – NB-IT-500610
Dateiversion von TradingBotGradle	javaTrading Git Master vom 22. Jan. 2021
Programm	IntelliJ IDEA Community Edition 202.6397.94

Testprotokoll

Methode	**Kriterien**	**Test-Methode**	☐	**Bemerkung**
Tradingbot.closeAllOrders()	Soll alpacaAPI.closeAllPositions aufrufen	Run TradingbotAPITest. closeAllOrdersTest()	☒	
Tradingbot.closeOrder()	Soll, wenn ein Order offen ist, diesen schliessen (von Symbol)	Run TradingbotAPITest.closeOrderTest_with_Order_set() und . closeOrderTest_without_Order_set()	☒	
Tradingbot.IsMarketOnline ()	Soll alpacaAPI.getClock() .getIsOpen() aufrufen	Run TradingbotAPITest.IsMarektOnlineTest()	☒	
Tradingbot.IsOrderSet(orderside)	Soll zurückgeben ob ein Order auf dem Symbol und in die übergebene Richtung offen ist. Bei Null spielt die Richtung keine Rolle	Run TradingbotAPITest.IsOrderSetTest()	☒	
Tradingbot.setOrder(orderside, stockPrice)	Soll, wenn es den Order nicht schon gibt, einen Order mit dem Übergebenen orderside und stockPrice setzten. Wenn ein Order in die andere Richtung gesetzt ist soll dieser geschlossen werden	Run TradingbotAPITest.setOrderTest_other_OrderSide_saved() und .setOrderTest_same_OrderSide_saved()	☒	
ReadWatchlist	ReadWatchlist soll korrekt trading.Watchlist.properties	Breakpoint auf MainController Zeile 41 setzen und nach ausführen var watchlist mit Datei vergleichen	☒	

Auswerten

Erreichen der Ziele

Wir denken, dass wir grundsätzlich viel in der uns gegebenen Zeit erreicht haben. Wir konnten auch viele unserer Ziele erreichen, nämlich die Folgenden. Wir haben einen funktionierenden Tradingbot, der aktuelle Daten vom Markt in einem Zeitintervall von einer Minute holt. Das Programm hat ausserdem einen Maincontroller der jedem Markt die wahrscheinlich beste Strategie zuweisen kann. Wenn die Strategie sich als schlecht erweist kann der Maincontroller eine neue Strategie zuweisen. Die Orders werden mit der Alpaca API verwaltet auf dem wir momentan zwar nur einen Paperaccount besitzen aber es möglich wäre mit echtem Geld zu handeln. Die Strategien können auf Indikatoren zugreifen und man kann sehr einfach eine neue Strategie hinzufügen, momentan besitzt er nur zwei Strategien.

Was wir leider nicht umsetzen konnten, ist das UI. Wir merkten mit der Zeit wie viel Zeit unser Projekt beanspruchen wird und entschieden uns, dass es besser ist unseren Fokus auf die Programmlogik selber zu legen und anstatt dem UI die Konsole und das UI von Alpaca API zu verwenden.

Ausserdem konnten wir nicht alle Edgecases überprüfen und mögliche Fehler abfangen. Wenn wir mit unserem Bot mit echtem Geld handeln möchten wäre dies das Wichtigste.

Teamarbeit

Dies war nicht unser erstes gemeinsames Projekt, wodurch wir wussten, dass wir gut harmonieren werden. Ausserdem haben wir ähnliche Interessen wodurch wir schnell eine Projektidee hatten die uns beide sehr interessierte. Wir hatten aber einen sehr grossen Wissensunterschied, David Oberkalmsteiner arbeitet schon das ganz Jahr mit C# was ähnlich wie Java ist und wodurch er schon gut im Objekt Orientiert Programmieren (OOP) war. Luca Hofstetter jedoch arbeitete im Geschäft mit Webseiten und hatte bis jetzt nicht sehr viel mit OOP und Programmlogik zu tun. Luca Hofstetter hatte sich aber schon ein grosses Wissen im Bereich Trading angeeignet, was David Oberkalmsteiner fehlte. Also konnten wir uns sehr gut ergänzen. Da die Schwierigkeit bei unserem Projekt nicht der Code selber war, sondern die Struktur und der Aufbau des Programmes, entschieden wir uns die Aufgaben nicht aufzuteilen. Deshalb trafen wir die meisten Entscheidungen zusammen, um unser gemeinsames Wissen einfliessen lassen zu können. Diese Taktik war zwar grundsätzlich langsamer aber wir denken, dass es uns sehr viele Korrekturen und Umstellungen in unserem Projekt ersparte und wir am Ende ein besseres Produkt haben.

Probleme

Die grössten Probleme hatten wir hauptsächlich zu Beginn des Projektes. Nämlich war uns IntelliJ und Java noch nicht vertraut und wir hatten grosse Probleme auf die verschiedenen Java Libraries und Imports klarzukommen. Daher hatten wir sehr grosse Mühe AlpacaAPI in unser Gradleprojekt zu importieren. Wir fanden leider kaum Hilfen, da kaum jemand einen Tradingbot auf Java programmiert und wir keine wirkliche Anleitung oder Dokumentation zu AlpacaAPI hatte.

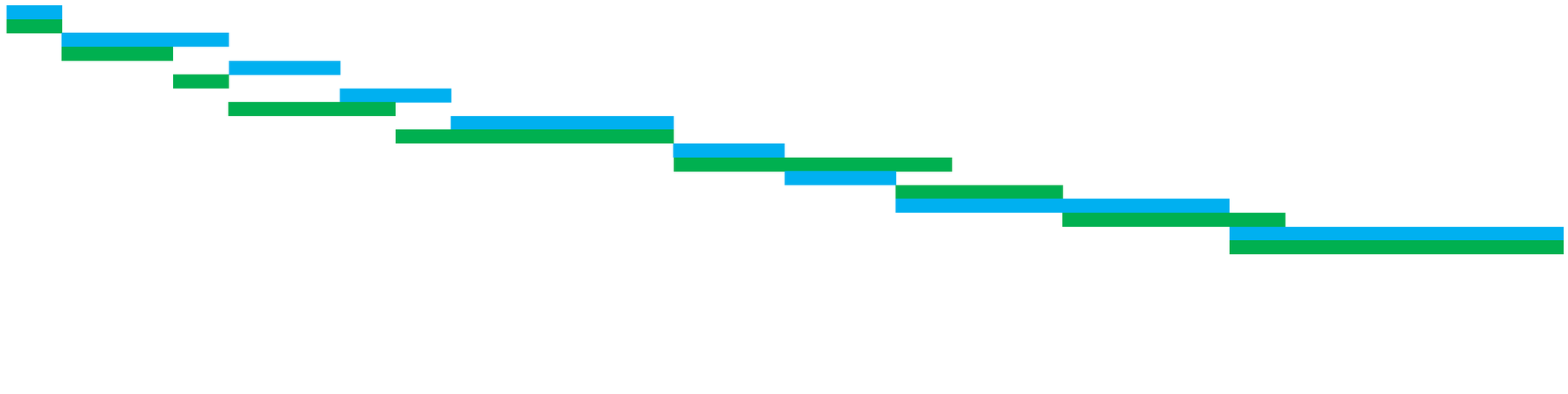
Ein anderes grosses Problem war, dass wir zu Beginn nicht genug Erfahrung in OOP hatten und viele Fehler in der Struktur und des Aufbaues programmierten. Da David Oberkalmsteiner parallel zu dem Projekt im Geschäft ein grosses praktisches Projekt umsetzte, entdeckte er immer mehr Verbesserungsmöglichkeiten und musste dann das ganze Projekt neu strukturieren. Dies kostete uns sehr viel Zeit. In Zukunft hoffen wir, dass wir zu Beginn des Projektes schon genug Wissen und Erfahrung sammeln konnten.

Neben vielen kleineren Problemen mit Threads und andren kleineren Funktionen, hatten wir sehr grosse Problemen mit Mocking. David Oberkalmsteiner hatte Mocking im Geschäft sehr häufig in C# benutzt. Wir nahmen also an, dass sich das Mocking in Java nicht gross von C# unterscheidet. Grundsätzlich stimmte das auch, aber in Java gab es noch viele Probleme die erst in C# oder mit gewissen Addons gelöst wurden. Ein Beispiel dafür ist das Mocken von finalen und/oder privaten Methoden und Klassen. Mit dem normalen Mockito ist dies nicht möglich. Deshalb mussten wir Powermockito installieren, dort dann noch ein Addon hinzufügen. Da dies komplettes Neuland für uns war fanden wir sehr spät heraus was wir genau tun mussten.

Zeitplan

Wir konnten den Zeitplan leider nicht ganz einhalten da die oben erwähnten Probleme sehr lange gebraucht haben. Wir denken aber, dass wir es im Endeffekt gut gelöst habe indem wir nicht Schritte die für die Bewertung nicht allzu relevant sind wegliessen.

IPERKA	Nr.	Arbeitspaket	Dauer (h)	Ist/Soll
I	1	zum Auftrag informieren	1	Soll
				Ist
	2	Idee finden und Umsetzbarkeit überprüfen	3	Soll
				Ist
P	3	Zeitplan schreiben	2	Soll
				Ist
	4	Use Case Diagramm erstellen	2	Soll
				Ist
	6	Klassendiagramm erstellen	4	Soll
				Ist
	8	API finden und implementieren	2	Soll
				Ist
	9	Grundstruktur erstellen	2	Soll
				Ist
	10	TradingAPI und getMarketData realisieren	6	Soll
				Ist
R	11	Tradingbot realisieren	8	Soll
				Ist
	12	Strategie realisieren	4	Soll
				Ist
	13	Edgecases die Fehler verursachen abfangen	4	Soll
				Ist
	14	Maincontroller realisieren	8	Soll
				Ist
K	15	Tests realisieren	6	Soll
				Ist
A	16	Dokumentation beenden	4	Soll
				Ist



IPERKA	Nr.	Arbeitspaket	Dauer (h)	Ist/Soll
I	1	zum Auftrag informieren	1	Soll
				Ist
	2	Idee finden und Umsetzbarkeit überprüfen	3	Soll
				Ist
P	3	Zeitplan schreiben	2	Soll
				Ist
	4	Use Case Diagramm erstellen	2	Soll
				Ist
	6	Klassendiagramm erstellen	4	Soll
				Ist
	8	API finden und implementieren	2	Soll
				Ist
	9	Grundstruktur erstellen	2	Soll
				Ist
	10	TradingAPI und getMarketData realisieren	6	Soll
				Ist
R	11	Tradingbot realisieren	8	Soll
				Ist
	12	Strategie realisieren	4	Soll
				Ist
	13	Edgecases die Fehler verursachen abfangen	4	Soll
				Ist
	14	Maincontroller realisieren	8	Soll
				Ist
K	15	Tests realisieren	6	Soll
				Ist
A	16	Dokumentation beenden	4	Soll
				Ist