Stockit - Securities Trading of Concepts Implementierung

Studienarbeit

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Herbstsemester 2021

Autor(en): Abinas Kuganathan, Jonas Knupp
Betreuer: Prof. Dr. Daniel Patrick Politze

OST
Ostschweizer
Fachhochschule

Stockit - Securities Trading of Concepts Implementierung

Abinas Kugnathan und Jonas Knupp

22. Dezember 2021

Danksagung

An dieser Stelle möchten wir all jenen danken, die uns während der Studienarbeit unterstützt haben.

Unser Dank gebührt vor allem Prof. Dr. Daniel Patrick Politze, welcher uns die Durchführung der Arbeit ermöglicht hat. Für die konstruktive Kritik und Freiheiten während der Arbeit sind wir besonders dankbar.

Des Weiteren geht unser Dank an Joel H. und Thomas H., die sich Zeit genommen haben, an unserem Usability Test teilzunehmen.

Rapperswil, 22. Dezember 2021
Abinas Kuganathan und Jonas Knupp

Inhaltsverzeichnis

Abstract

Die Identifikation von neuen, profitablen Produktkonzepten ist eine aufwändige Angelegenheit. Bestehende Methoden der Präferenzmessung wie beispielsweise Umfragen oder Conjoint-Analysen sind für das durchführende Unternehmen mit erheblichem finanziellem Aufwand verbunden und für die Probanden eine trockene Angelegenheit. Dahan et al. beschreiben in ihrem 2010 im Journal of Marketing Research erschienen Paper eine kostengünstige und skalierbare, alternative Methode der Präferenzmessung namens Securities Trading of Concepts (STOC), nach der Produktkonzepte von den Probanden wie Wertschriften an der Börse gehandelt werden. Am Ende der Handelsphase erfolgt die Auswertung, wobei das Konzept mit dem grössten VWAP bzw. Medianpreis gewinnt. Dahan et al. führten ihre STOC-Spiele mit dem MIT Web Market Simulator aus. Diese Applikation wurde zur Durchführung von verschiedensten Marktexperimenten entwickelt und ist daher nicht speziell auf die Bedürfnisse zur Durchführung von STOC-Spielen zugeschnitten. Im Rahmen dieser Studienarbeit sollte eine moderne Web-Applikation zur Erstellung, Durchführung und Auswertung von STOC-Spielen entwickelt werden.

In einem ersten Schritt musste im Kontext der Domänenanalyse ein geeignetes Marktmodell evaluiert werden. Die Wahl fiel auf einen Continuous Double Auction Market. Für das Marktmodell wurde ein Interface definiert, sodass in Zukunft mit geringem Aufwand zusätzliche Marktmodelle implementiert werden können. Im nächsten Schritt wurde eine Anforderungsanalyse durchgeführt. Die Anforderungen wurden aus dem STOC-Paper gewonnen und im Gespräch mit dem Betreuer präzisiert. Aufgrund der Ergebnisse der Anforderungsanalyse konnten schliesslich Wireframes für das User Interface erstellt werden. Anschliessend erfolgte die Evaluation der Technologien, wobei die Wahl für das Frontend auf Vue.js und für das Backend auf NestJS fiel. Als Datenbank wird PostgreSQL eingesetzt. Die Kommunikation zwischen Front- und Backend findet mittels einer REST und einer Socket.io Schnittstelle statt. Zur Validierung der Applikation wurden automatisierte Unit- und Integrationstests erstellt. Am Ende der Entwicklungsmeilensteine wurden zudem manuelle Systemtests durchgeführt.

Das Ergebnis dieser Arbeit ist eine Web-Applikation, die die Erstellung, Durchführung und Auswertung von STOC-Spielen ermöglicht. Bei der Erstellung eines STOC-Spieles stehen zahlreiche Konfigurationsmöglichkeiten zur Verfügung. So kann der Organisator beispielsweise die Transaktionsgebühren oder das Startkapital der Trader festlegen. Während einem STOC-Spiel haben die Trader in Echtzeit Zugriff auf Marktinformationen und erhalten einen detaillierten Überblick über ihr Portfolio sowie ihre Performance. Am Ende eines STOC-Spieles wird für den Organisator ein Bericht generiert, mit dem er entscheiden kann, welches Produktkonzept weiterverfolgt werden sollte. In zukünftigen Arbeiten sollten ausführliche Testdurchläufe mit ca. 50 Tradern durchgeführt werden, um Stockit konzeptionell und technisch zu validieren. Ferner könnten neue Marktmodelle implementiert oder der bestehende Continuous Double Auction Market ausgebaut werden.

Management Summary

Ausgangslage

Die Identifizierung von neuen, profitablen Produktkonzepten ist eine aufwändige Angelegenheit. Bestehende Methoden der Präferenzmessung wie Umfragen oder die Conjoint Analyse sind mit hohen Kosten und mangelhafter Skalierbarkeit verbunden. Ausserdem ist der Prozess für die Probanden trocken, was deren Anwerbung schwierig gestaltet. Securities Trading of Concepts (STOC) ist eine alternative Methode der Präferenzmessung, die von Dahan et al. im gleichnamigen Paper, dass 2010 im Journal of Marketing Research erschienen ist, vorgestellt wurde. Nach STOC werden Produktkonzepte wie Wertschriften an der Börse gehandelt. STOC basiert auf der Feststellung, dass Märkte ein effizientes Werkzeug zur Ermittlung des Wertes eines Gutes sind. Die Verwendung von STOC ist mit folgenden Vorteilen verbunden:

1. Da die Trader ihre Meinung mehrmals während eines STOC-Spieles in Form von Orders kundtun können, wird die Genauigkeit der Messung erhöht.
2. Die Trader lernen mit der Zeit wie andere Trader ein Produktkonzept bewerten und passen ihre Bewertung entsprechend an.
3. In Umfragen kann nur eine begrenzte Anzahl an Fragen gestellt werden. Die Effizienz des Marktes steigt mit der Anzahl Tradern, was STOC inhärent skalierbar macht.
4. STOC stellt keine formalen Anforderungen an die Beschreibung der Produktkonzepte. Alles was nötig ist, ist eine präzise Beschreibung in beliebiger Form.
5. STOC ist häufig kosteneffektiver als alternative Methoden der Präferenzmessung.

Ein STOC-Spiel läuft folgendermassen ab: In einem ersten Schritt werden die handelbaren Produktkonzepte beschrieben. Anschliessend muss die Semantik des Preises definiert werden. Die Trader müssen die Produktkonzepte sowie den Markt verstehen. In einem nächsten Schritt folgt dann die Durchführung des STOC-Spieles. Dabei generieren die Trader Transaktionen, die persistiert werden. Nach dem Ende eines STOC-Spieles erfolgt die Auswertung. Während der Auswertung wird anhand der Transaktionsdaten entschieden, welches Produktkonzept das profitabelste ist [5].

Das Ziel dieser Arbeit war die Konzeptualisierung und Entwicklung einer Applikation zur Erstellung, Durchführung und Auswertung von STOC-Spielen. Die Ausarbeitung der genauen Funktionalitäten der Applikation war Bestandteil der Arbeit und erfolgte aufgrund des Papers Securities Trading of Concepts (STOC) und im Gespräch mit dem Betreuer.

Vorgehen und Technologien

Mit Scrum+ wurde ein agiles Vorgehen gewählt, dass jedoch durch die Definition von Meilensteinen eine langfristige Planung ermöglichte. Im Zuge der Domänenanalyse wurde ein Domänenmodell erstellt. Die funktionalen Anforderungen wurden als Use Cases definiert, während die nichtfunktionalen Anforderungen gemäss ISO 25010 erstellt wurden. Basierend auf der Domain- und Anforderungsanalyse wurde die Evaluation der Technologien durchgeführt.

Das Frontend wurde mittels Vue.js umgesetzt. Das Backend wurde mittels NestJS realisiert. Als Datenbank kommt PostgreSQL zum Einsatz. Es wurden zwei Schnittstellen für die Kommunikation zwischen dem Front- und Backend definiert. Die synchrone Kommunikation findet mittels REST statt, während die asynchrone Echtzeitkommunikation mittels Socket.io stattfindet. Zur einfacheren Handhabung von HTTPS wird NGINX als Reverse Proxy eingesetzt.

Ergebnisse

Innerhalb von 14 Wochen wurde mit Stockit eine Applikation entwickelt, die die Erstellung, Durchführung und Auswertung von STOC-Spielen ermöglicht. Während der Erstellung stehen dem Organisator zahlreiche Konfigurationsmöglichkeiten zur Verfügung. So können Dauer, Währung, Startkapital, Transaktionskosten und das verwendete Marktmodell festgelegt werden. Am Ende des Erstellungsprozesses hat der Organisator die Möglichkeit die Produktkonzepte zu erfassen. Pro Produktkonzept können Name, Symbol, Beschreibung, Link, Kontakt Email-Adresse sowie ein Bild erfasst werden. Nach der Erstellung kann das STOC-Spiel zu einem beliebigen Zeitpunkt vom Organisator gestartet werden. Mittels eines Game Codes können die Trader einem STOC-Spiel beitreten. Während der Handelsphase kann der Organisator das Marktgeschehen beobachten. Die Trader haben die Möglichkeit, Anteile an Produktkonzepte zu kaufen und zu verkaufen. Dabei wird der Markt in Echtzeit aktualisiert. Eine Rangliste motiviert die Trader möglichst profitabel zu handeln. Das STOC-Spiel kann durch den Organisator vorzeitig beendet werden. Mit dem Ende eines STOC-Spiels wird ein Bericht für den Organisator generiert, der es erlaubt, das gewinnbringendste Produktkonzept zu identifizieren.

Ausblick

Mit Stockit sollten umfangreiche Praxistests mit ca. 50 Tradern durchgeführt werden. Dabei sollte geprüft werden, inwieweit Stockit technisch und konzeptionell den Anforderungen an eine STOC-Applikation genügt. Ferner könnte Stockit angepasst werden, sodass es als verteiltes System betrieben werden kann. Ausserdem könnten weitere Marktmodelle implementiert werden. Es wäre ebenfalls denkbar, einen Zahlungsprovider an Stockit anzubinden, um erfolgreiche Trader am Ende eines Spieles zu belohnen.

Kapitel 1

Problembeschreibung

Dieses Kapitel befasst sich mit der Analyse der Problemstellung. Die Themenausschreibung wurde sehr offen formuliert, die einzige Vorgabe war es, eine Web-Applikation zur Durchführung von STOC-Spielen zu entwickeln. Zur Konkretisierung der Problemstellung wurde eine ausführliche Domänenanalyse durchgeführt. Dazu wurde zunächst die Literatur zum Thema STOC gesichtet. Während der Sichtung der STOC-Literatur wurde festgestellt, dass ein Marktmodell evaluiert werden musste. Durch eine Literaturrecherche zum Thema Finanzmärkte konnte mit dem Continuous Double Auction Market ein geeignetes Marktmodell identifiziert und ausgearbeitet werden. Aus den Ergebnissen der Domänenanalyse und im Gespräch mit dem Betreuer konnte schliesslich die Anforderungsanalyse durchgeführt werden, in deren Verlauf die funktionalen und nicht-funktionalen Anforderungen definiert wurden.

1.1 Domänenanalyse

Im Zuge der Domänenanalyse wurde zuerst eine Analyse der STOC-Theorie durchgeführt. Anschliessend wurde ein geeignetes Marktmodell evaluiert.

1.1.1 Securities Trading of Concepts

Die Identifizierung neuer, profitabler Produktkonzepte erfordert einen Einblick in die Präferenzen der Konsumenten. Dahan et al. [5] entwickelten eine neue Methode zur Messung der Präferenzen von Konsumenten namens Securities Trading of Concepts (STOC). Nach STOC werden Produktkonzepte wie Wertschriften an der Börse gehandelt. STOC basiert auf der Erkenntnis, dass Märkte den Wert von Gütern über den Preis effizient bestimmen können [7].

Nach Dahan et al. [5] sollte ein STOC-Experiment wie folgt durchgeführt werden: In einem ersten Schritt müssen die zu evaluierenden Produktkonzepte ausgewählt und beschrieben werden. Anschliessend muss die Semantik des Preises der Anteile geklärt werden. Beispielsweise könnte den Tradern mitgeteilt werden, dass jedes Produktkonzept von einem Unternehmen vertrieben wird und mit dem STOC-Spiel der Preis der Initial Public Offering bestimmt werden soll. In einem nächsten Schritt müssen die Trader mit der Software, mit der das STOC-Spiel durchgeführt wir vertraut gemacht werden. Dann kann mit dem Start des Tradings begonnen werden. Alle Transaktionen müssen gespeichert werden. Nach dem das STOC-Spiel beendet wurde, wird aufgrund von Metriken, die auf den gespeicherten Transaktionen basieren, entschieden, welches Produktkonzept von den Tradern bevorzugt wurde. Dahan et. stellten fest, dass der volume-weighted-average-price (VWAP) und der Medianpreis sich dazu am besten eigenen.

STOC hat einige Vorteile gegenüber alternativen Methoden der Präferenzmessung. So weist STOC eine erhöhte Genauigkeit, Skalierbarkeit, Kosteneffektivität auf und es werden keine formalen Anforderungen an die Beschreibung der Produktkonzepte gestellt.

Dahan et al. validierten STOC, indem sie vier STOC-Spiele durchführten. Zur Durchführung der STOC-Spiele wurde der MIT Web Market Simulator, eine Börsensoftware, die eine Plattform für verschiedene elektronische Märkte anbietet, genutzt. Der MIT Web Market Simulator wurde zur Durchführung von ökonomischen Simulationen und Experimenten entwickelt und ist daher nicht speziell für die Durchführung von STOC-Spielen optimiert [10]. Elemente, wie zum Beispiel, dass die Beschreibungen der Produktkonzepte direkt in der STOC-Applikation verfügbar sind oder ein Ranking, fehlen deswegen. Ferner führt das etwas "technisch" anmutende Design dazu, dass der "Gamification" Aspekt verloren geht.

Obschon die STOC-Methode bereits etwa 20 Jahre alt ist, existiert gegenwärtig keine spezialisierte Software, welche die Erstellung, Durchführung und Auswertung von STOC-Spielen unterstützt. Im Rahmen dieser Studienarbeit soll eine Web-Applikation entwickelt werden, die alle Funktionalitäten zur Anwendung der STOC-Methode anbietet. Die Software soll gemäss den Funktionen des MIT Web Market Simulator und den Informationen aus [5] konzipiert werden. Damit soll es Organisationen mit geringem Aufwand ermöglicht werden, die STOC-Methode in der Praxis anzuwenden.

1.1.2 Marktmodell

Die Durchführung von STOC-Spielen erfordert einen Markt, auf dem die Produktkonzepte gehandelt werden können. Da dem Projektteam nicht bekannt ist, was es für verschiedene Ausprägungen von Finanzmärkten gibt, und wie diese aufgebaut sind, wurde eine Literaturrecherche durchgeführt. Nach der Literaturrecherche wurde entschieden, dass ein Continuous Double Auction Market verwendet wird.

Taxonomie von Finanzmärkten

Finanzmärkte lassen sich nach De Jong und Rindi [9] in die Kategorien Order-driven, Quote-driven und Hybrid Market einteilen. Die verschiedenen Marktmodelle sind relevant, da die STOC-Applikation mit beliebigen Marktmodellen arbeiten können muss.

Order-driven Market In einem Order-driven Market werden Buy- und Sell Orders ohne einen Intermediären (ausser dem Broker) gematcht. Order-driven Markets benötigen Regeln, die besagen, welche Orders Vorrang haben und wie der Preis zustande kommt. Es ist möglich Limit und Market Orders zu platzieren. Ein Beispiel eines Order-driven Market ist die Continuous Double Auction mit einem Orderbuch.

Quote-driven Market In einem Quote-driven Market bestimmen Liquiditätsgeber (Market Makers oder Dealers) den Preis einer Wertschrift. Ausserdem besitzen die Liquiditätsgeber ein Monopol auf den Handel, was bedeutet, dass jeder, der handeln möchte mit einem Liquiditätsgeber handeln muss. In einem reinen Quote-driven Market gibt es keine Limit-Orders.

Hybrid Market In einem Hybrid Market werden Elemente des Order-driven und Quote-driven Market kombiniert. Zum Beispiel können sich Market Makers und eine Continuous Double Auction konkurrenzieren.

Marktmodell Continuous Double Auction

Das Projektteam hat sich dazu entschieden, den Continuous Double Auction Market als Marktmodell zu implementieren, da dieser auch von Dahan et al. [5] verwendet wird. Auch die Einschränkung, dass keine Market Orders unterstützt werden, wird von Dahan et al. übernommen. Da von Dahan et al. keine Details zum Regelwerk des Marktes (Order-Matching und Preisstrategie) beschrieben werden, wurde in der Literatur nach einer Beschreibung für die detaillierte Marktstruktur gesucht.

Beschreibung der Marktstruktur Ein einfacher Continuous Double Auction Market kann folgendermassen modelliert werden: Für jedes Wertpapier wird ein eigenes Orderbuch erstellt. Wenn eine Limit Order eintrifft, wird zunächst geprüft, ob diese ausführbar ist. Handelt es sich bei der eingetroffenen Order um eine Buy Limit Order, ist dies möglich, wenn die Limite der eintreffenden Order $\geq$ Ask. Handelt es sich hingegen um eine Sell Limit Order ist die Order ausführbar, wenn die Limite der eintreffenden Order $\leq$ Bid. Ist die Order nicht ausführbar, wird sie in das zur Wertschrift gehörende Orderbuch eingetragen. Falls die Order ausführbar ist, muss ermittelt werden, mit welcher Order im Orderbuch sie zusammengeführt wird. Handelt es sich bei der hereinkommenden Order um eine Buy Limit Order wird die Order mit der Sell Limit Order mit der tiefsten Limite zusammengeführt. Eine hereinkommende Sell Limit Order wird mit der Buy Limit Order mit der grössten Limite zusammengeführt. Falls im Orderbuch mehrere Orders zum selben Preis aufzufinden sind, entscheidet die Order-Matching Strategie über das weitere vorgehen. Der Preis, zudem ein Handel stattfindet, wird von der Preisstrategie festgelegt. Falls die Order nicht vollständig ausgeführt werden kann, wird die Order partiell ausgeführt und ihr Volumen entsprechend angepasst [18].

Die Abbildung 1.1 gibt eine Übersicht über die Abwicklung einer ankommenden Limit Order.

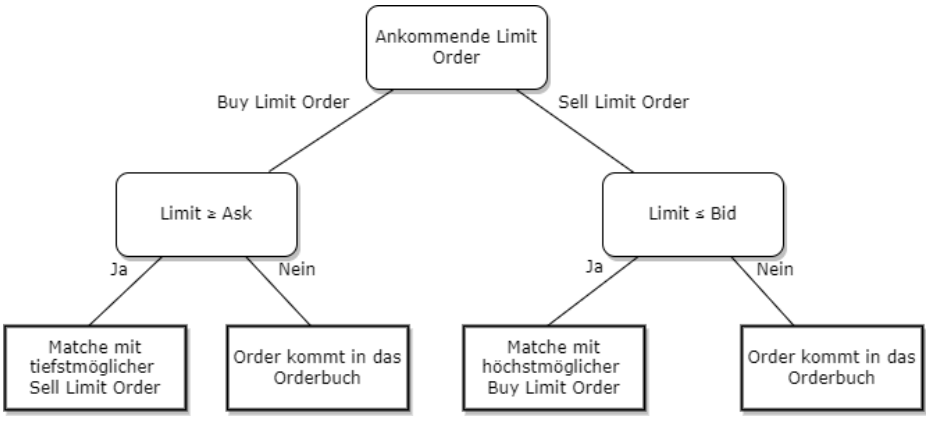


Abbildung 1.1: Bearbeitung einer ankommenden Limit Order

Die nachfolgenden Abschnitte beschreiben einige mögliche Order-Matching und Preisstrategien. Es wurden Strategien gewählt, die den Strategien der Xetra, der grössten Deutschen Börse, nahekommen.

Order-Matching Strategie Es gibt zwei verbreitete Verfahren: Price-Time-Priority, auch FIFO genannt und Pro-Rata. Gemäss Price-Time-Priority werden die Orders nach dem zeitlichen eintreffen priorisiert, wobei die zuerst eingetroffene Order Priorität hat und soweit möglich vollständig ausgeführt wird. Dem Pro-Rata Ansatz nach wird kein Vorrang definiert, sondern alle in Frage kommenden Orders werden anteilsmässig am Volumen ausgeführt [15].

Da Xetra Price-Time-Priority verwendet, wird diese Strategie auch von uns implementiert [16].

Preisstrategie Die Preisstrategie bestimmt, zu welchem Preis ein Handel zustande kommt, nachdem zwei Order zusammengeführt wurden. Die k-Preisstrategie erlaubt es einen Parameter k zu wählen, der bestimmt wie die Rente zwischen Käufer und Verkäufer aufgeteilt wird. Die Formel für den Preis ist: $p = b + (1 - k)a$, wobei a der Ask und b der Bid ist. Würde man k auf 0.5 setzen wäre der Preis das arithmetische Mittel zwischen der Buy Limit und Sell Limit Order [13]. Xetra verwendet nicht die k-Preisstrategie sondern eine eigene Preisstrategie [16]. Die für uns relevanten Regeln wurden im Folgendem Zusammengefasst:

- Trifft eine hereinkommende Buy Limit Order auf eine bestehende Sell Limit Order, gilt der Preis der gematchten Sell Limit Order als Preis.
- Trifft eine hereinkommende Sell Limit Order auf eine bestehende Buy Limit Order, gilt der Preis der gematchten Buy Limit Order als Preis.

Da in der Literatur keine Bezeichnung für diese Preisstrategie zu finden ist, verwenden wir dafür den von uns gewählten Begriff Ask/Bid Strategie.

1.1.3 Domänenmodell

Mit den gewonnen Informationen aus der Analyse der STOC-Theorie und der Recherche über die Marktmodelle konnte das Domänenmodell, dass in Abbildung 1.2 zu sehen ist, entwickelt werden.

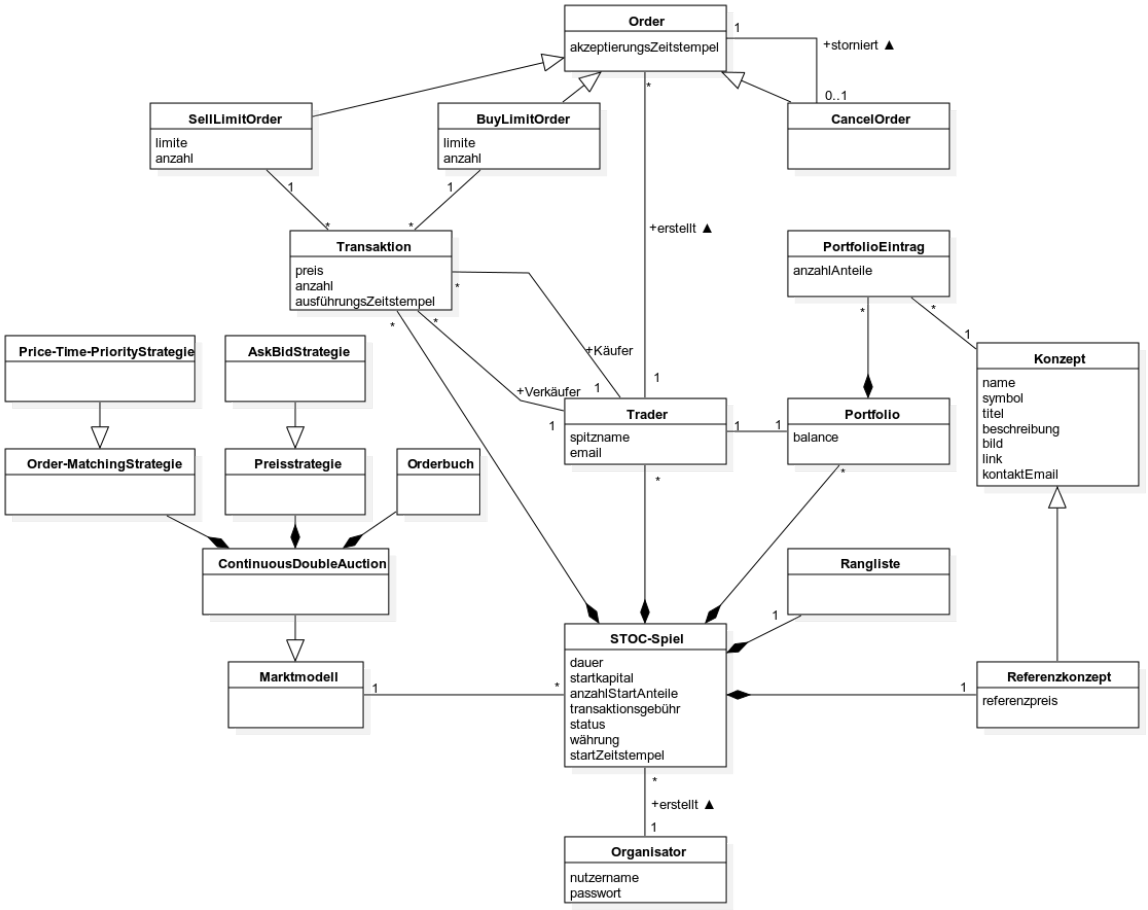


Abbildung 1.2: Domänenmodell

1.2 Anforderungsanalyse

Die Anforderungsanalyse umfasst die Identifizierung und Beschreibung der funktionalen- sowie nichtfunktionalen Anforderungen.

1.2.1 Funktionale Anforderungen

Die funktionalen Anforderungen wurden als Use Cases beschrieben. Ausserdem wurden die beteiligten Akteure identifiziert.

Use Case Diagramm

Das Use Case Diagramm in Abbildung 1.3 gibt eine Übersicht über die Use Cases und Akteure.

Abbildung 1.3: Use Case Diagramm

Akteure

Die Tabelle 1.1 zeigt eine Übersicht über die involvierten Akteure.

Akteur	Bezeichnung
Organisator	Der Organisator möchte ein STOC-Spiel durchführen, um das profitabelste Produktkonzept zu identifizieren.
Trader	Der Trader nimmt am STOC-Spiel im Auftrag des Organisators bei. Er handelt mit Anteilen an Produktkonzepten um den Wert seines Portfolios zu maximieren.

Tabelle 1.1: Übersicht über die Akteure

Use Cases

Die Use Cases wurden nach Larman beschrieben [11]. Komplexe Use Cases wurden im fully-dressed Format beschrieben, während einfache Use Cases im brief Format spezifiziert wurden.

UC01: Erstellung und Anmeldung Organisator Account

- **Main Success Scenario:** Der Organisator kann einen Account erstellen. Wenn ein Account vorhanden ist, kann sich der Organisator anmelden.

UC02: STOC-Spiel erstellen und konfigurieren

- **Primary Actor:** Organisator
- **Stakeholders and Interests:**
  - Organisator: Möchte eine einfache Möglichkeit ein Spiel zu erstellen und zu konfigurieren, sowie Produktkonzepte zu erfassen und Trader einzuladen.
- **Preconditions:** UC01
- **Postconditions:**
  - Die Produktkonzepte sind in der Datenbank gespeichert.
  - Die Konfiguration für das STOC-Spiel wurde in der Datenbank gespeichert.
  - Es wurde ein Einladungs-Code generiert, den der Organisator an die Trader senden kann.
  - Der Organisator kann das STOC-Spiel starten.
- **Main Success Scenario:**
  1. Der Organisator erstellt ein STOC-Spiel.
  2. Der Organisator definiert die Dauer des STOC-Spiels.
  3. Der Organisator definiert die Höhe der Transaktionsgebühren.
  4. Der Organisator bestimmt wie viel "Bargeld" die Trader zu Beginn erhalten.
  5. Der Organisator wie viele Anteile an Produktkonzepten jeder Trader zu Beginn erhält.
  6. Der Organisator bestimmt die Währung.
  7. Der Organisator wählt ein Marktmodell.

8. Der Organisator definiert die handelbaren Produktkonzepte.
9. Der Organisator definiert ein Referenzkonzept.

- **Extensions:**

4. a)
    1. Der Organisator wählt einen Continuous Double Auction Market.
    2. Der Organisator wählt die Order-Matching Strategie Price-Time Priority.
    3. Der Organisator wählt die Preisstrategie Ask/Bid Strategie.

- **Frequency of Occurrence:** Häufig

UC03: STOC-Spiel beitreten

- **Preconditions:** UC02, Einladungscode ist vorhanden
- **Main Success Scenario:** Der Trader kann einem STOC-Spiel beitreten. Vor dem Start des Spieles kann der Trader detaillierte Beschreibungen zu den Produktkonzepten aufrufen.

UC04: Echtzeitübersicht für Organisator

- **Preconditions:** UC02
- **Main Success Scenario:** Während des STOC-Spieles erhält der Organisator eine Echtzeitübersicht über das Marktgeschehen. Ausserdem kann der Organisator die Rangliste über das Vermögen der Spieler einsehen.

UC05: Am Handel teilnehmen

- **Preconditions:** UC03
- **Main Success Scenario:** Der Trader kann am Handel teilnehmen. Dies beinhaltet UC06, UC07, UC08 und UC09.

UC06: Order aufgeben

- **Primary Actor:** Trader
- **Stakeholders and Interests:**
    - Trader: Der Trader möchte Anteile eines Produktkonzeptes verkaufen, das er für überbewertet hält oder Anteile eines Produktkonzeptes kaufen, das er für unterbewertet hält.
- **Preconditions:** UC03
- **Postconditions:**
    - Falls die Order gültig ist, wurde sie im Orderbuch eingetragen.
- **Main Success Scenario:**

1. Der Trader erstellt eine Limit Order

- **Extensions:**

    1. a) Buy Limit Order
        1. Das System prüft, ob der Trader genug "Bargeld" hat um die Buy Limit Order im Worst Case auszuführen. Da es keine Schulden gibt, prüft das System die allfällig anderen offenen Orders des Traders, inklusive deren potentiellen Transaktionsgebühren.
        2. Falls genug "Bargeld" vorhanden ist wird die Order dem Markt übergeben. Ansonsten wird der Trader benachrichtigt.

    1. b) Sell Limit Order
        1. Das System prüft, ob der Trader genug Anteile an den Produktkonzepten hat um die Sell Limit Order auszuführen. Dazu prüft das System die allfällig ausstehenden anderen offenen Order des Traders.
        2. Falls genug Anteile an den Produktkonzepten vorhanden sind, wird die Order ins Orderbuch eingetragen. Ansonsten wird der Trader benachrichtigt.

- **Frequency of Occurrence:** Häufig

UC07: Order stornieren

- **Preconditions:** UC03
- **Main Success Scenario:** Der Trader kann Buy Limit und Sell Limit Orders stornieren, solange diese noch nicht ausgeführt wurden.

UC08: Marktinformationen abrufen

- **Preconditions:** UC03
- **Main Success Scenario:** Der Trader kann die folgenden Marktinformationen abrufen: Last, Low, High, Bid (mit Size), Ask (mit Size) und das gehandelte Volumen. Ausserdem erhält der Trader einen Überblick über sein Portfolio ("Bargeld", Anzahl Anteile pro Produktkonzept im Besitz und deren Marktwert). Der Trader kann auch die Rangliste über das Vermögen aller Trader einsehen.

UC09: Produktkonzepte ansehen

- **Preconditions:** UC03
- **Main Success Scenario:** Der Trader kann während des Spiels detaillierte Beschreibungen der Produktkonzepte aufrufen.

UC10 STOC-Spiel beenden

- **Preconditions:** UC02
- **Main Success Scenario:** Der Organisator kann das STOC-Spiel vor Ablauf der Zeit manuell beenden.

UC11: Auswertung des STOC-Spieles ansehen

- **Preconditions:** UC10 oder Beendigung nach Zeit
- **Main Success Scenario:** Der Organisator sieht eine Übersicht über die zustande gekommenen Transaktionen, den VWAP, die Rangliste sowie pro handelbarem Produktkonzept eine Liste der Trader, die darin investiert haben, inklusive dem jeweiligen Volumen. Die Liste ist absteigend nach dem Volumen geordnet.
- **Optionale Erweiterung:** Der Organisator kann die Auswertung des Spieles als CSV exportieren

1.2.2 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen wurden nach ISO 25010 definiert [8].

Reliability

NFR01: Verhalten bei Systemabsturz Bei einem Systemabsturz müssen abgeschlossene Spiele erhalten bleiben. Unterbrochene Spiele müssen nicht wiederhergestellt werden können.

NFR02: Verfügbarkeit Die Applikation garantiert eine Verfügbarkeit von 95%. Um zu ermitteln ob dies erreicht wurde, wird die Software über die Dauer von einem Tag beobachtet und die Verfügbarkeit für diesen Zeitraum gemessen.

Performance Efficiency

NFR03: Anzahl Konzepte pro STOC-Spiel Es werden maximal 15 Konzepte pro STOC-Spiel unterstützt.

NFR04: Maximale Anzahl parallel durchführbare STOC-Spiele Es müssen zwei STOC-Spiele parallel durchgeführt werden können. Die Erreichung dieses Zieles wird experimentell ermittelt.

NFR05: Maximale Anzahl Trader pro STOC-Spiel Es müssen maximal 100 Trader an einem STOC-Spiel teilnehmen können. Die Erreichung dieses Zieles wird experimentell ermittelt.

NFR06: Maximale Zeit bis aktualisierte Marktinformationen beim Trader ankommen Die Zeit vom Abschluss einer Transaktion bis die Trader die aktualisierten Marktinformationen sehen darf zwei Sekunden nicht überschreiten. Die Erreichung dieses Zieles wird experimentell ermittelt.

Security

NFR07: Organisator Accounts Die Authentifizierung von Organisatoren findet mittels Nutzername und Passwort statt. Passwörter müssen mindestens acht Zeichen lang sein und dürfen nicht im Klartext gespeichert werden.

NFR08: Zutritt zu STOC-Spielen Der Zutritt zu STOC-Spielen ist nur durch die Kenntnis des Einladungscodes geschützt. Jeder, der den Einladungscode besitzt, kann dem Spiel beitreten.

Compatibility

NFR09: Co-Existenz mit anderen Systemen Das Frontend darf nicht alle Browser-Ressourcen konsumieren. Während die Frontend Applikation läuft muss es daher jederzeit möglich sein, im Browser beliebige andere Applikationen, ohne erkennbare Performance-Probleme, auszuführen.

Usability

NFR10: Lernbarkeit Das User-Interface der Applikation muss selbsterklärend sein. Es ist daher keine externe Anleitung nötig, um die Applikation als Organisator oder Trader zu bedienen.

NFR11: Schutz vor Fehlern Organisatoren und Trader können die Applikation nicht in einen fehlerhaften Zustand versetzen. Falls sie ungültige Eingaben tätigen, lehnt die Applikation die Eingaben ab und informiert die Benutzer über die Fehler.

NFR12: Wiedereinstieg in das STOC-Spiel für Trader Trader müssen einem STOC-Spiel erneut beitreten können, falls sie während des STOC-Spiels die Applikation verlassen haben.

NFR13: Lokalisierung Das Frontend ist für die Deutschschweiz lokalisiert. GUI-Texte werden in einer externen Datei definiert, die beim Start der Applikation geladen wird, sodass in Zukunft mit geringem Aufwand weitere Sprachen hinzugefügt werden können.

Maintainability

NFR14: Austauschbarkeit des Marktmechanismus Der Marktmechanismus (standardmässig ein Continuous Double Auction Market) soll durch einen anderen Marktmechanismus austauschbar sein.

NFR15: Dokumentation der Schnittstellen Damit die Applikation zu einem späteren Zeitpunkt ausgebaut werden kann bzw. einzelne Komponenten ausgewechselt werden können, müssen sämtliche Schnittstellen detailliert dokumentiert sein.

NFR16: Testbarkeit Jeder Anwendungsfall wird im Rahmen eines Systemtests verifiziert.

Portability

NFR17: Browserunterstützung Das Frontend ist lauffähig im Web-Browser Chrome Version 94, Firefox Version 93 und Edge Version 94.

NFR18: Installierbarkeit Die Applikation muss mit Hilfe einer Installationsanleitung innerhalb von maximal zwei Stunden installiert werden können.

1.2.3 Minimum Viable Product

Das Minimum Viable Product umfasst die UC01 bis UC08 und UC10, wobei einige dieser Use Cases auch Funktionen enthalten, die nicht zum Minimum Viable Product gehören:

- UC04: Rangliste
- UC08: Rangliste

Kapitel 2

Lösungskonzept

Dieses Kapitel beschäftigt sich mit der Erarbeitung des Lösungskonzeptes. Aufgrund der Problembeschreibung wurden zunächst geeignete Technologien evaluiert, wobei die Wahl für das Frontend auf Vue.js und für das Backend auf NestJS fiel, während eine PostgreSQL Datenbank zum Einsatz kommt. Mit den gewählten Technologien wurde anschliessend ein Prototyp als Proof-of-Concept erstellt. Die erfolgreiche Umsetzung des Prototyps erlaubte die Erstellung der Softwarearchitektur. Ausserdem wurde das Designkonzept von Stockit erarbeitet. Mit der Fertigstellung der Softwarearchitektur und des Designkonzepts konnte die Construction-Phase begonnen werden. Da während der Implementierung festgestellt wurde, dass die Initialisierung eines STOC-Spieles möglicherweise mangelhaft war, wurde eine Analyse über mögliche Initialisierungsstrategien durchgeführt. Die Analyse brachte zum Vorschein, dass die bis anhin genutzte Initialisierungsstrategie, bei der alle Konzepte zum gleichen, vom Organisator festgelegten Preis initialisiert werden, mangelhaft ist. Deshalb wurde ein alternatives Verfahren implementiert, bei dem zum Spielbeginn ein Referenzkonzept erfasst werden muss, dass als Numéraire fungiert.

2.1 Evaluation der Technologien

In diesem Abschnitt wird festgelegt, mit welchen Technologien gearbeitet wird. Es werden nur die wichtigsten Technologien im Voraus evaluiert. Kleinere Libraries werden während des Entwicklungsprozesses spontan und ohne Vermerk in der Dokumentation evaluiert. Die Angaben zu den evaluierten Frameworks wurden im Oktober 2021 ermittelt.

2.1.1 Programmiersprachen

Das Frontend wird mittels Typescript programmiert. Als Alternative würde sich lediglich WebAssembly anbieten. WebAssembly ist allerdings eine eher neue Technologie und das Projektteam möchte keine zusätzlichen Risiken eingehen, da bereits die Echtzeitverbindung eine Herausforderung darstellt. Ausserdem kann dank Typescript auch ohne WebAssembly sauberer, objekt-orientierter und stark typisierter Code produziert werden. Aus Qualitätsmassnahme werden bei jedem Merge Request gegenseitige Code Reviews durchgeführt. Deshalb wurde beschlossen, dass das Backend ebenfalls in Typescript geschrieben wird.

2.1.2 Frontend

Für das Frontend muss ein Framework, eine Design- und ein State-Management Library ausgewählt werden.

Frontend Framework

Das Projektteam hat die drei sehr weit verbreiteten SPA-Frameworks Vue.js, React und Angular evaluiert, wobei zu beachten ist, dass React streng genommen kein Framework, sondern eine Library ist.

Framework	Github Stars	Letzter Release	Initialer Release
Vue.js	189k	07.06.2021	02.2014
React	176k	22.03.2021	29.05.2013
Angular	77k	07.10.2021	14.09.2016

Tabelle 2.1: Übersicht der evaluierten Frontend Frameworks

Vue.js Vue.js ist ein Library, die aber mit offiziellen, optionalen Paketen zu einem vollständigen Framework ausgebaut werden kann. Komponenten bestehen aus einem Vue-File, das jeweils separate Sektionen für das Template (HTML), Skripte (Javascript) und Style (CSS) beinhaltet.

Vorteile

- Flexibel installierbare, optionale Pakete
- Pro Komponenten gibt es ein File, dass allerdings in Template, Skript und Style unterteilt wird
- Unterstützung von JSX

Nachteile

- Ökosystem noch nicht so gross wie bei Angular und React

React Im Gegensatz zu Vue.js mit installierten offiziellen Paketen und Angular ist React eine Library und kein Framework, weshalb es sehr leichtgewichtig ist. React nutzt JSX um Komponenten zu definieren. Mit JSX ist es möglich, die Logik und die View in einem File zu vereinen. Über die Organisation von Komponenten trifft React keine Aussagen, was dem Entwickler viele Freiheiten ermöglicht.

Vorteile

- Architektur kann frei gewählt werden
- Sehr schnelle Entwicklung möglich

Nachteile

- Logik und View im selben File, was bei sehr grossen Projekten unübersichtlich werden kann
- Da React kein Framework ist, müssen weitere Libraries gewählt werden, um gewisse Features zu implementieren (z.B. Dependency Injection, HTTP-Calls)

Angular Angular ist ein relativ schwergewichtiges Framework, dass bereits eine grobe Architektur vorgibt. Die Applikation wird in mehrere Module unterteilt, die jeweils eine gewisse Funktionalität zur Verfügung stellen. Innerhalb von Modulen ist es möglich, Komponenten zu definieren. Komponenten sollten wiederverwendbar sein, um die Code Duplikation zu verringern. Eine Komponente besteht aus einem Typescript-File für die Logik, einem HTML-File für die Präsentation, einem CSS-File für das Styling und einem Spec-File für das Testing. Durch diese Struktur entsteht eine sehr starke Separation of Concern. Ausserdem bietet Angular einen Dependency Injection Mechanismus an, mit dem Services mit geringem Aufwand injected werden können.

Vorteile

- Grosse Funktionalität (z.B. Dependency Injection)
- Sehr ausgeprägte Separation of Concern

Nachteile

- Komplexe Architektur mit steiler Lernkurve

Entscheidung Als Frontend Framework wird Vue.js verwendet. Angular ist zu schwergewichtig für das Projekt, da die Modularität aufgrund des geringen Umfangs des STOC Frontends nicht benötigt wird. React wäre ebenfalls eine gute Wahl gewesen, allerdings wurde das Projektteam dadurch von Vue.js überzeugt, dass es möglich ist, Template, Script und Style zu trennen, aber trotzdem in einem File unterzubringen.

State-Management Library

Um eine gute Maintainability zu gewährleisten, ist es wichtig, dass eine zentrale State-Verwaltung im Frontend vorhanden ist. Ansonsten ist der State über das gesamte Frontend verteilt, was sehr schnell unübersichtlich wird. Für das State-Management wird Vuex verwendet. Im Gegensatz zu anderen State-Management Libraries, wie beispielsweise Redux wurde Vuex speziell dazu entwickelt mit Vue.js verwendet zu werden, was eine bessere Integration mit der restlichen Applikation ermöglicht.

Design Library

Da das Team sich für Vue.js im Abschnitt Frontend Framework entschieden hat, wird für die Design Library Naive UI und Primevue evaluiert.

Naive UI Naive UI beschreibt sich selbst als "A Vue 3 Component Library. Fairly Complete. Customizable Themes. Uses TypeScript. Not too Slow.".

Vorteile

- In Typescript geschrieben
- Mehr als 70 Komponenten

Nachteile

- Einige nicht intuitive Komponenten
- Englische Übersetzung nicht perfekt

Primevue Primevue ist eine Design Library von Primefaces. Primevue gibt es auch für Angular, React und Java.

Vorteile

- Stellt Typescript Typ-Interfaces zur Verfügung
- 80+ Komponenten
- Theme einfach anpassbar
- Wird von grossen Unternehmen wie bspw. Nvidia und SAP verwendet

Nachteile

- Wenige Github-stars (1.7k)
- Hat kostenpflichtige Themes

Entscheidung Das Projektteam hat sich für Primevue entschieden, da es anpassbar ist und mehr Komponenten zur Verfügung stellt. Primevue hat auch einen Wrapper über charts.js, welcher nützlich für die Visualisierung der Preise sein wird.

2.1.3 Backend

Auf der Seite des Backends musste ebenfalls ein geeignetes Framework ermittelt werden. Ausserdem musste ein Datenbankmanagementsystem eruiert werden.

Backend Framework

Für das Backend wurden Express.js, meteor und NestJS evaluiert. Die folgende Tabelle gibt eine Übersicht über die wichtigsten Kennzahlen der evaluierten Frameworks:

Framework	Github Stars	Letzter Release	Initialer Release
Express.js	54.6k	26.05.2019	16.11.2010
meteor	42.6k	15.09.2021	20.01.2012
NestJS	41.3k	06.10.2021	2017

Tabelle 2.2: Übersicht der evaluierten Backend Frameworks

Express.js Express.js ist ein minimales Framework. Grundsätzlich ist die einzige Vorgabe von Express.js, dass Router definiert werden, die wiederum einen oder mehrere Requesthandler beinhalten. Der Entwickler hat die vollständige Freiheit, wie er das Backend aufbauen möchte.

Vorteile

- Sehr grosse Verbreitung (grosses Ökosystem, grosse Community)
- Hat sich über die Jahre bewährt

Nachteile

- Viele grundsätzliche Dinge wie bspw. Authentisierung, Autorisierung, Testing, Services und Controller müssen selber geplant und implementiert werden
- Letzter Release liegt über zwei Jahre zurück

meteor Im Gegensatz zu Express.js und NestJS ist meteor ein Fullstack Framework. Als Frontend kann z.B. Angular oder React verwendet werden. Ein Alleinstellungsmerkmal von meteor ist, dass das Distributed Data Protocol für die Kommunikation zwischen Client und Server verwendet wird. Dieses basiert auf dem Websocket Protokoll und erlaubt eine Echtzeitverbindung zwischen Client und Server. Meteor gibt ein vergleichsweise starres Gerüst für die Architektur der Applikation vor.

Vorteile

- Gutes Tooling
- Hat sich über die Jahre bewährt

Nachteile

- Wenig Flexibilität (z.B. bei der Wahl der Datenbank)

NestJS NestJS ist ein relativ junges Framework, dass aber ein starkes Wachstum aufweist. So war es 2019 das am stärksten wachsende Javascript Web-Backend Framework, gemessen an der Anzahl Github Stars [2]. NestJS baut auf Express.js (kann durch Fastify ersetzt werden) auf und orientiert sich an der Architektur von Angular. Die Applikation wird in Module aufgeteilt. NestJS bietet einen Dependency Injection Mechanismus an, mit dem Services in Controller oder andere Services injected werden können. Eine grosse Anzahl an Modulen und Services wird bereits von NestJS zur Verfügung gestellt. So gibt es beispielsweise Module für Websockets, den Datenbankzugriff, die Authentisierung und Autorisierung. NestJS folgt dem Convention over Configuration Ansatz und ermöglicht es dem Architekten, die bereitgestellten Module anzupassen (bspw. den Authentifizierungsmechanismus oder die Datenbankanbindung). Die von NestJS zur Verfügung gestellte Architektur ist so organisiert, dass das Testing leicht fällt.

Vorteile

- Framework wurde in Typescript geschrieben
- Grobe Architektur wird bereits vorgegeben, inklusive benötigtem Tooling (z.B. für das Testing)
- Ausgezeichnete Dokumentation mit Verweisen auf vollständige Beispiele in einem separatem Github-Repository

Nachteile

- Vergleichsweises junges Framework

Entscheidung Das Projektteam hat sich für NestJS entschieden. NestJS bietet eine grobe Architektur und ein gutes Tooling, erlaubt aber trotzdem viel Freiraum für den Entwickler. Express.js wurde primär nicht gewählt, weil sehr viel "Gerüst-Code" geschrieben hätte werden müssen. Da die zur Verfügung stehende Projektzeit relativ eng begrenzt ist, möchte sich das Projektteam auf die Implementierung der Business-Logik konzentrieren können. Meteor wurde insbesondere aufgrund der starren Architekturvorgabe nicht gewählt. Ausserdem wäre für meteor einiges an Einarbeitungszeit nötig, da es sich erheblich von Express.js und NestJS unterscheidet. Der Einstieg in NestJS gestaltet sich für das Projektteam einfacher, da es auf Express.js basiert, mit dem das Projektteam bereits Erfahrung hat.

Datenbankmanagementsystem

Sämtliche Daten, die persistent gespeichert werden müssen, sind stark strukturiert. Deshalb bietet sich ein relationales Datenbankmanagementsystem an. Die Menge an zu speichernden Daten und die Zugriffe darauf sind bescheiden, weshalb alle grossen relationalen Datenbankmanagementsysteme in Frage kommen. Das Projektteam hat sich für PostgreSQL entschieden, da es ein ausgezeichnetes Tooling (z.B. pgAdmin, pg_dump), eine grosse Community und eine umfangreiche Dokumentation besitzt. Ausserdem ist Postgres Open Source und somit sind mit der Nutzung keine Lizenzgebühren verbunden.

2.1.4 Schnittstellen

Dieser Abschnitt befasst sich damit, auf welche Technologien für die Schnittstelle zwischen dem Front- und dem Backend zurückgegriffen wird.

Kommunikation ausserhalb von STOC-Spielen

Kommunikation, die ausserhalb eines STOC-Spieles stattfindet, soll über eine synchrone Schnittstelle erfolgen. Im Web-Umfeld hat sich REST für synchrone Schnittstellen durchgesetzt [4]. Aufgrund der grossen Verbreitung gibt es zahlreiche Ressourcen über REST und es stehen verschiedene Tools für die Dokumentation von REST Schnittstellen zur Verfügung. Ausserdem hat das Projektteam bereits Erfahrung im Umgang mit REST. Aufgrund dieser Argumente wird die synchrone Schnittstelle mittels REST umgesetzt. Die Dokumentation der REST Schnittstelle erfolgt mittels OpenAPI.

Kommunikation innerhalb von STOC-Spielen

Die Kommunikation, die innerhalb eines STOC-Spieles stattfindet wird über eine asynchrone Schnittstelle realisiert, da die Kommunikation innerhalb eines STOC-Spieles inhärent asynchron ist. Beispielsweise ist nicht garantiert, dass wenn ein Client eine Order an den Server sendet, diese sofort ausgeführt wird. Im Gegenteil, der Server kann die Order zu einem beliebigen Zeitpunkt ausführen und muss dann den Client benachrichtigen. Zusätzlich muss beachtet werden, dass Marktinformationen bei Änderungen zeitnah dem Client ausgeliefert werden. Die obigen Anforderungen können durch vier Technologien umgesetzt werden.

Short Polling Mittels Polling könnte man die Kommunikation innerhalb von STOC-Spielen synchron umsetzen. Der Client würde in regelmässigen Intervallen Anfragen an die REST Schnittstelle senden. Beispielsweise könnte der Client alle fünf Sekunden fragen, ob eine Order bereits ausgeführt wurde.

Vorteile

- Einheitliche, synchrone Schnittstelle
- Geringe Komplexität des Backends
- Einfache Skalierbarkeit

Nachteile

- Hohe Netzwerklast
- Negativer Einfluss auf die Performanz des Backends

Long Polling Beim Long Polling sendet der Client eine Anfrage an den Server. Der Server beantwortet die Anfrage nicht, bis er eine Antwort bereit hat bzw. ein bestimmtes Ereignis eingetreten ist. Sobald der Server geantwortet hat, verarbeitet der Client die Antwort und sendet eine erneute Anfrage, die wiederum unbeantwortet bleibt, bis der Server nach einiger Zeit antwortet.

Vorteile

- Einheitliche, synchrone Schnittstelle
- Geringe Komplexität des Backends
- Einfache Skalierbarkeit

Nachteile

- Hohe Netzwerklast
- Negativer Einfluss auf die Performanz des Backends

Server-Sent Events Server-Sent Events ermöglichen einen unidirektionalen Stream vom Server zum Client. Die Kommunikation von Client zu Server würde weiterhin über eine traditionelle REST Schnittstelle stattfinden.

Vorteile

- Transport über HTTP
- Einfache Skalierbarkeit

Nachteile

- Nur unidirektionale Kommunikation von Server zu Client möglich
- Eingeschränkte API (kompliziertes Uni- und Multicast)

Websockets Websockets ermöglichen eine bidirektionale Verbindung zwischen Client und Server. Da das Websocket Protokoll keinen automatischen Reconnect und kein Fallback zum Polling unterstützt, gibt es Libraries die diese Features und weitere Komfort-Features implementieren. Beispielsweise können mit Socket.io sogenannte Rooms erstellt werden. Clients können sich in einen Raum begeben und der Server kann Nachrichten an ausgewählte "Rooms" senden.

Vorteile

- Bidirektionale Verbindung zwischen Client und Server
- Umfangreiche Features wie bspw. Fallback zu Polling, Rooms, etc.

Nachteile

- Komplexe Skalierbarkeit
- Komplexität des Backends steigt, da das Websocket Protokoll unabhängig von HTTP ist

Entscheidung Obwohl ein unidirektionaler Event-Stream vom Server zum Client ausreichen würde, werden keine Server-Sent Events sondern Websockets mit Socket.io verwendet. Dies hat den Grund, dass Server-Sent Events nur ein sehr bescheidenes API zur Verfügung stellen. Beispielsweise müsste man mit Server-Sent Events pro Client einen Event-Stream erstellen, um nicht unnötigerweise alle Clients in einem STOC-Spiel zu benachrichtigen. Ausserdem sind Websockets bzw. Socket.io verbreiteter und es ist somit einfacher, Hilfe bei Problemen zu finden. Polling kommt nicht in Frage, da dabei Ressourcen (Bandbreite und CPU) verschwendet werden.

2.2 Prototyp

Um zu validieren, dass die Software mittels der gewählten Technologien umgesetzt werden kann, wurde ein Prototyp erstellt. Der Prototyp ist für den Betrieb auf dem Port 3000 ausgerichtet.

2.2.1 Funktionalität

Die Funktionen wurden so gewählt, dass insgesamt ein technischer Durchstich durchgeführt wird. Der Prototyp geht davon aus, dass nur ein STOC-Spiel gleichzeitig betrieben wird. Zu Beginn muss der Client dem Server mitteilen, dass er dem Raum "game-1" beitreten möchte. Anschliessend sendet der Client Orders mit dem Symbol "AAP" zum Server. Der Server bestätigt den Empfang der Order und markiert die Order nach einer zufälligen Zeit zwischen 5 und 20 Sekunden zum Preis der Limite der Order als ausgeführt. Die Ausführung wird in der Datenbank als Transaktion vermerkt und der Client wird über die Ausführung benachrichtigt. Ausserdem erhält der Client aktualisierte Marktinformationen, wenn eine Order ausgeführt wurde. Die Nachricht, dass eine Order akzeptiert bzw. ausgeführt wurde wird nur an den Trader gesendet, der die Order aufgegeben hat. Über die geänderten Marktinformationen werden alle Trader informiert.

Der Client bietet einen Button an, mit dem neue Orders erstellt werden können. Ausserdem listet er die nicht abgeschlossenen Orders tabellarisch auf. Wenn eine Order abgeschlossen wurde, wird sie aus der Tabelle entfernt. Der Last-Preis wird in einem Liniendiagramm dargestellt.

2.2.2 Schnittstelle Client/Server

Der Server bietet eine synchrone REST sowie eine asynchrone Socket.io Schnittstelle an.

REST

Dieser Abschnitt beschreibt die REST Schnittstelle.

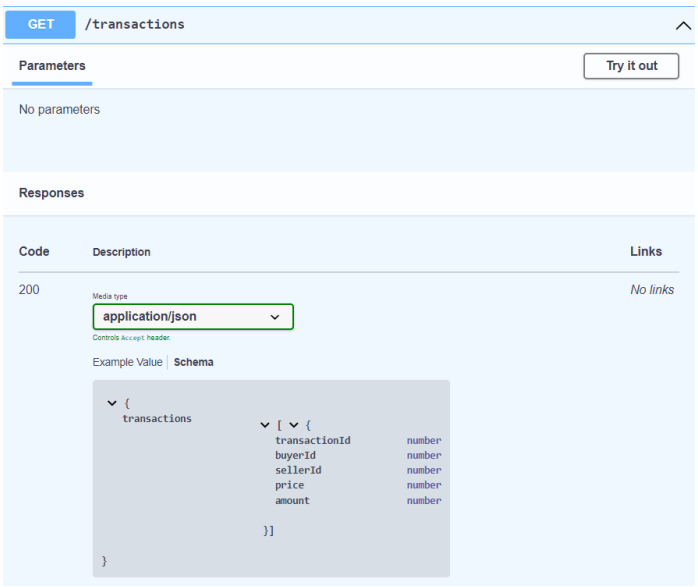


Abbildung 2.1: Schnittstelle zum Laden der Transaktionen

Socket.io

Dieser Abschnitt beschreibt die Socket.io Schnittstelle.

Publish Dieser Abschnitt beinhaltet Nachrichten, die vom Client zum Server gesendet werden.

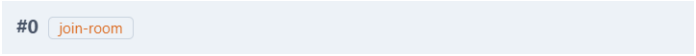


Abbildung 2.2: Schnittstelle für den Raumbeitritt

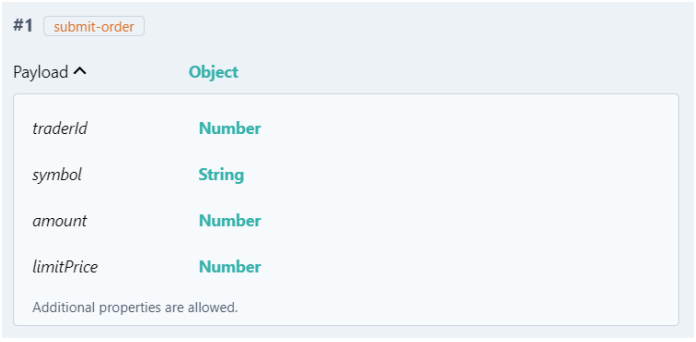


Abbildung 2.3: Schnittstelle für das Aufgeben von Orders

Subscribe Dieser Abschnitt beinhaltet Nachrichten, die vom Server an den Client gesendet werden.

Abbildung 2.4: Schnittstelle, um den Client über die Akzeptanz der Order zu informieren

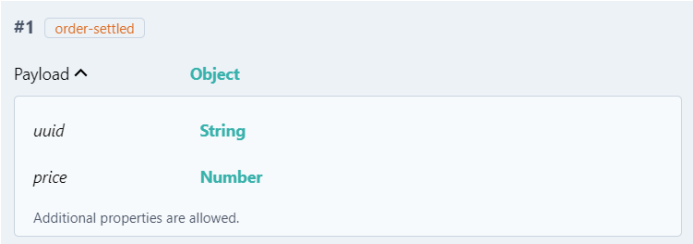


Abbildung 2.5: Schnittstelle, um den Client über die Ausführung der Order zu informieren

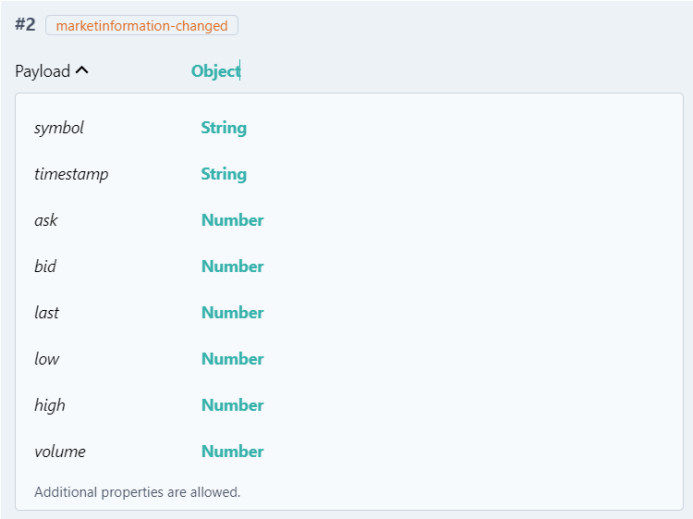


Abbildung 2.6: Schnittstelle, um alle Clients über die Änderung der Marktinformationen zu informieren

2.2.3 Ergebnis

Der Test des Prototyps verlief erfolgreich. Damit ist sichergestellt, dass die evaluierten Technologien geeignet zur Lösung der Problemstellung sind und es kann mit der Planung der Architektur begonnen werden.

2.3 Softwarearchitektur

Dieses Kapitel befasst sich mit der Architektur der Stockit Software. Zu Beginn wurde die Stockit Software gemäss der arc42 Building Block View beschrieben [1]. Des Weiteren werden die Schnittstellen, wichtige Abläufe, die CI/CD, das Deployment, die Authentication, das Logging, die Persistenz sowie der Speicherverbrauch beschrieben.

2.3.1 Kontextabgrenzung

Die Whitebox Systemübersicht in Abbildung 2.7 zeigt die zwei Akteure, Organisator und Trader, die mit Stockit interagieren können.

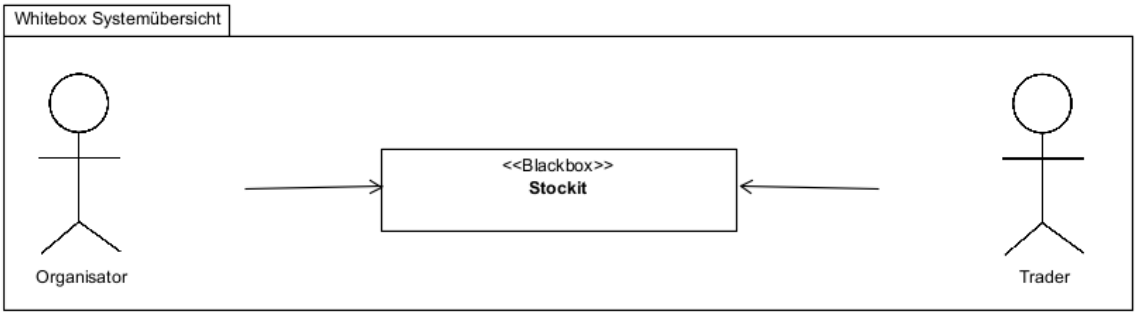


Abbildung 2.7: Kontextabgrenzung

Blackboxes

Name	Verantwortung
Stockit	Registrierung und Anmeldung von Organisatoren, Erstellung und Durchführung von STOC-Spielen, Präsentation von Berichten

Tabelle 2.3: Blackboxes der Kontextabgrenzung

2.3.2 Ebene 1

Stockit ist in drei Teile gegliedert: das Frontend besteht aus einer Vue.js Single-Page Application, während das Backend aus einem NestJS Server und einem PostgreSQL Datenbankmanagementsystem besteht. Die Kommunikation zwischen Organisator resp. Trader und Stockit findet über Vue.js statt. Vue.js kommuniziert mittels REST und Socket.io mit dem NestJS Server. PostgreSQL ist mittels TypeORM mit dem NestJS Server verbunden.

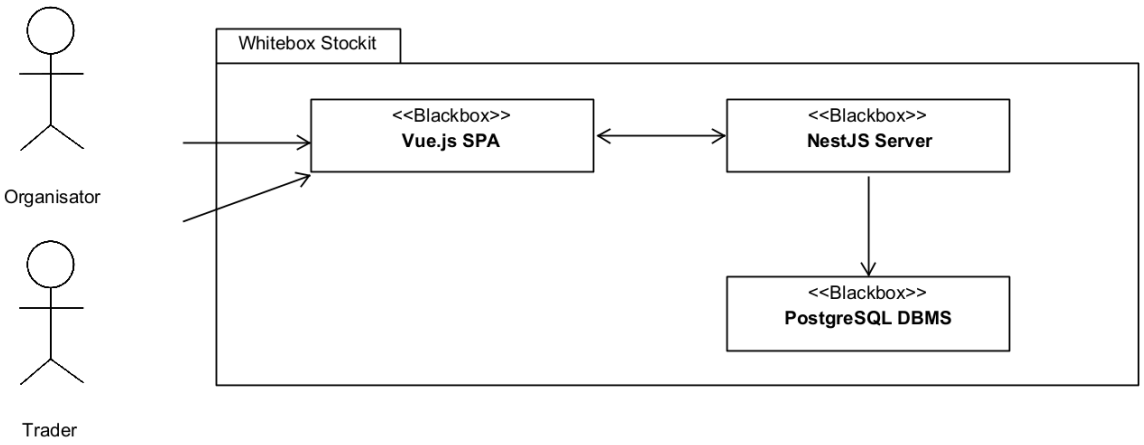


Abbildung 2.8: Whitebox Stockit

Blackboxes

Name	Verantwortung
Vue.js SPA	Entgegennahme von Benutzereingaben, Darstellung von Informationen
NestJS Server	Koordination von STOC-Spielen
PostgreSQL DBMS	Persistierung der Organisatoren-, Trader-, Konzept- und Spieldaten

Tabelle 2.4: Blackboxes von Stockit

2.3.3 Ebene 2

Whitebox Vue.js SPA

Die Blackbox Router leitet Anfragen an die Whitebox Präsentation weiter. Die Whitebox Präsentation kann auf die Whitebox Business Logic zurückgreifen, um Daten zu laden, zu senden oder Daten lokal zu speichern. Die Whitebox Business Logic kommuniziert mit der Blackbox NestJS Server über die Blackbox Data Access.

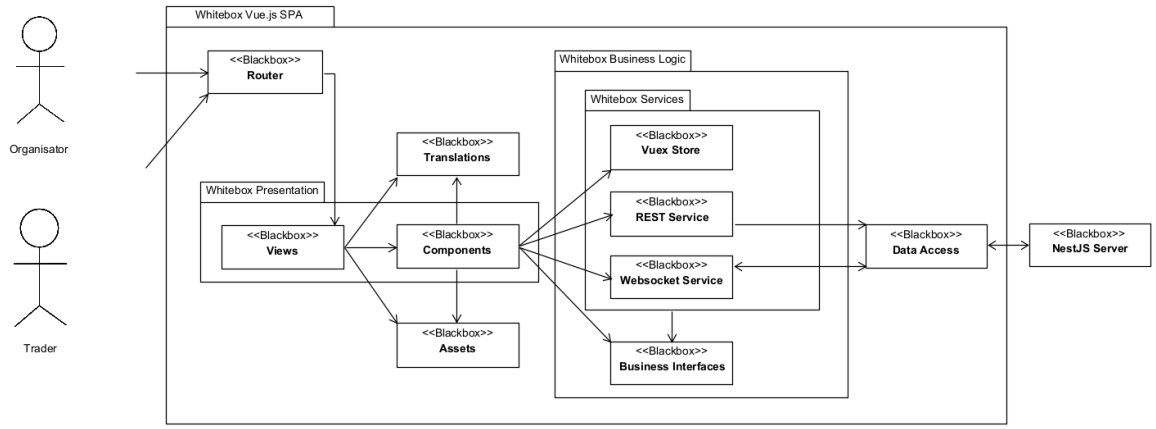


Abbildung 2.9: Whitebox Vue.js SPA

Whiteboxes

Name	Verantwortung
Präsentation	Entgegennahme von Benutzereingaben, Darstellung von Informationen
Business Logic	Implementierung der Business Logic
Services	State Mangement und Kommunikation mit dem NestJS Server

Tabelle 2.5: Whiteboxes der Vue.js SPA

Blackboxes

Name	Verantwortung
Router	Weiterleitung der Anfrage an die richtige View
View	Darstellung von Informationen mittels Components
Components	Entgegennahme von Inputdaten und Anzeige von Informationen
Translations	Anbieten von Übersetzungen
Assets	Anbieten von Bilder und Styles
Vuex Store	Daten speichern und bereitstellen
REST Service	Kommunikation über REST mit dem Server
Websocket Service	Kommunikation über Websocket mit dem Server
Business Interfaces	Bereitstellen von Interfaces
Data Access	Protokollspezifische Kommunikation

Tabelle 2.6: Blackboxes der Vue.js SPA

Whitebox NestJS Server

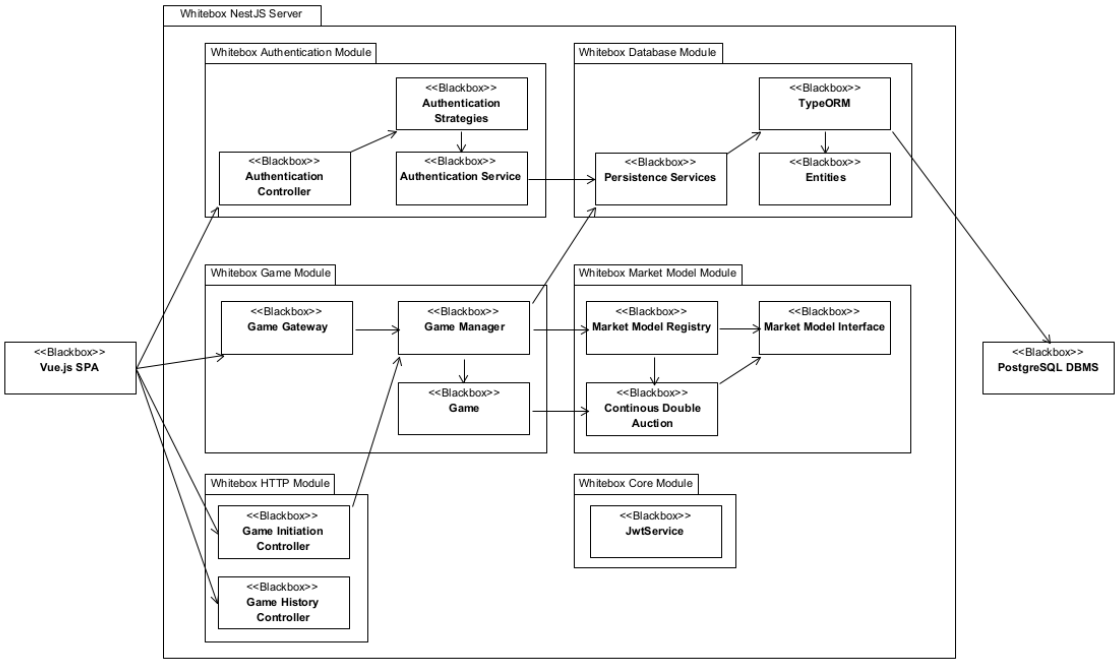


Abbildung 2.10: Whitebox NestJS Server

Whiteboxes

Name	Verantwortung
Authentication Module	Das Authentication Module ist für die Registrierung und Anmeldung von Organisatoren zuständig. Ausserdem erfolgt die Identifizierung der Trader über dieses Modul.
Game Module	Das Game Module kapselt die Durchführung der Spiele.
HTTP Module	Dieses Modul ist für die Erstellung von STOC-Spielen und Abfrage von Game Reports zuständig.
Database Module	Das Database Module kapselt den Zugriff auf die Datenbank. Des Weiteren enthält es die Entities, die das Datenbankschema definieren.
Market Model Module	Das Market Model Module enthält das Market Model Interface sowie alle damit verbundenen Interfaces. Ausserdem enthält es konkrete Implementierungen von Market Models.
Core Module	Das Core Module ist für die Bereitstellung von globalen Services verantwortlich. Es importiert das von NestJS zur Verfügung gestellte JwtModule und stellt dessen Services global zur Verfügung.

Tabelle 2.7: Whiteboxes des NestJS Servers

Blackboxes

Name	Verantwortung
Authentication Controller	Der Authentication Controller ist für die Kommunikation über HTTP zuständig, die die Authentication betrifft.
Authentication Strategies	Kapselt die verwendeten Authentifizierungsstrategien.
Authentication Service	Der Authentication Service ist für die Validierung von Nutzernamen und Passwörtern der Organisatoren sowie die Identifizierung von Tradern zuständig.
Game Gateway	Das Game Gateway ist für die gesamte Kommunikation, die über Socket.io läuft zuständig.
Game Manager	Der Game Manger ist ein Singleton Service, der die Erstellung, Durchführung und Beendigung von STOC-Spielen organisiert.
Game	Die Game Klasse kapselt ein einzelnes STOC-Spiel. Es enthält eine Instanz des verwendeten Market Models, die Transaktionen, die Trader, die Portfolios der Trader und stellt das Ranking zur Verfügung.
Game Initiation Controller	Der Game Initiation Controller bietet eine HTTP-Schnittstelle für das Erstellen und Starten von STOC-Spielen an.
Game History Controller	Der Game History Controller bietet eine HTTP-Schnittstelle für Erstellung von Berichten für vergangene STOC-Spiele an.
Persistence Services	Die Persistence Services vermitteln zwischen den Modulen, die auf die Datenbank zugreifen und dem Datenbankmodul. Die Persistence Services nutzen Repositories, um Anfragen zu beantworten. Für komplexere Datenbankabfragen, für die die Repositories nicht ausreichen, verwenden die Persistence Services den Entity Manager.
TypeORM	TypeORM wird genutzt, um auf das PostgreSQL DBMS zuzugreifen.
Entities	Für jede Tabelle in der Datenbank wird eine Entity erstellt. Diese werden mit Annotation versehen um das Datenbankschema zu definieren. TypeORM erstellt pro Entity zur Laufzeit ein Repository, über dass CRUD Abfragen für die entsprechende Entity getätigt werden können.
Market Model Registry	Die Market Model Registry ist für die Registrierung und das Abrufen von Informationen zu den implementierten Market Models zuständig.
Market Model Interface	Diese Blackbox enthält das Market Model Interface und alle damit verbundenen Interfaces. Wenn in Zukunft ein weiteres Market Model implementiert wird, kann von diesen Interfaces abgeleitet werden.
Continous Double Auction	In dieser Blackbox sind alle Klassen enthalten, die das Market Model Continous Double Auction betreffen. Dies umfasst insbesondere auch eine Order Book Klasse.
JwtService	Erlaubt das Signieren und die Verifizierung von JWT-Token.

Tabelle 2.8: Blackboxes des NestJS Servers

2.3.4 Ebene 3

Whitebox Market Model Module

Für das Market Model Module wurde ein UML Klassendiagramm, welches in Abbildung 2.11 ersichtlich ist, erstellt, weil ein stabiles Interface benötigt wird, damit es in Zukunft möglich ist, eigene Market Models zu erstellen. Ausserdem wurden die Klassen, die für die Registrierung von Marktmodellen nötig sind modelliert. Das Continuous Double Auction Market Model wurde als Blackbox modelliert.

Abbildung 2.11: Whitebox Market Model Module

OrderMatchingStrategy Die "OrderMatchingStrategy" bestimmt mit welchen Orders eine ankommende Order zusammengeführt wird. Bei der "PriceTimePriorityStrategy" wird nur eine Order zurückgegeben. Die "OrderMatchingStrategy" muss die gematchten Orders aus dem Orderbuch entfernen. Wichtig ist, dass die "OrderMatchingStrategy" nur zur Anwendung kommt, wenn es mehrere Orders zum Ask bzw. Bid Preis gibt. Bei der Rückgabe muss angegeben werden, welches Volumen der jeweiligen ausgewählten Orders ausgeführt werden ("quantityToExecute"). Die Summe der "quantityToExecute"-Werte muss $\leq$ der Quantity, der ankommenden Order sein.

PriceStrategy Die Preisstrategie legt fest, zu welchem Preis eine Transaktion zustande kommt. Dabei ist zu beachten, dass der 1. Parameter die ankommende Order ist und der 2. Parameter die Order, die bereits im Orderbuch vorhanden war. Der zurückgegebene Preis muss $\geq$ der Limite der Sell Limit Order und $\leq$ der Limite der Buy Limit Order sein.

2.3.5 Schnittstellen

Die REST Schnittstelle wurde nach OpenAPI 3.0.3 spezifiziert. Die Socket.io Schnittstelle wurde nach AsyncAPI 2.2.0 erstellt. Beide Schnittstellenbeschreibungen sind im Anhang G als PDF verfügbar.

2.3.6 Abläufe

Dieser Abschnitt beschreibt den Spielbeitritt sowohl für die Trader wie auch für den Organisator. Der Vorteil an den definierten Abläufen ist, dass sobald ein Trader bzw. ein Organisator ein JWT-Token für ein STOC-Spiel besitzt, er beliebig die Stockit-Seite neuladen bzw. verlassen und wieder betreten kann und er automatisch wieder zum STOC-Spiel weitergeleitet werden kann.

Spielbeitritt Trader

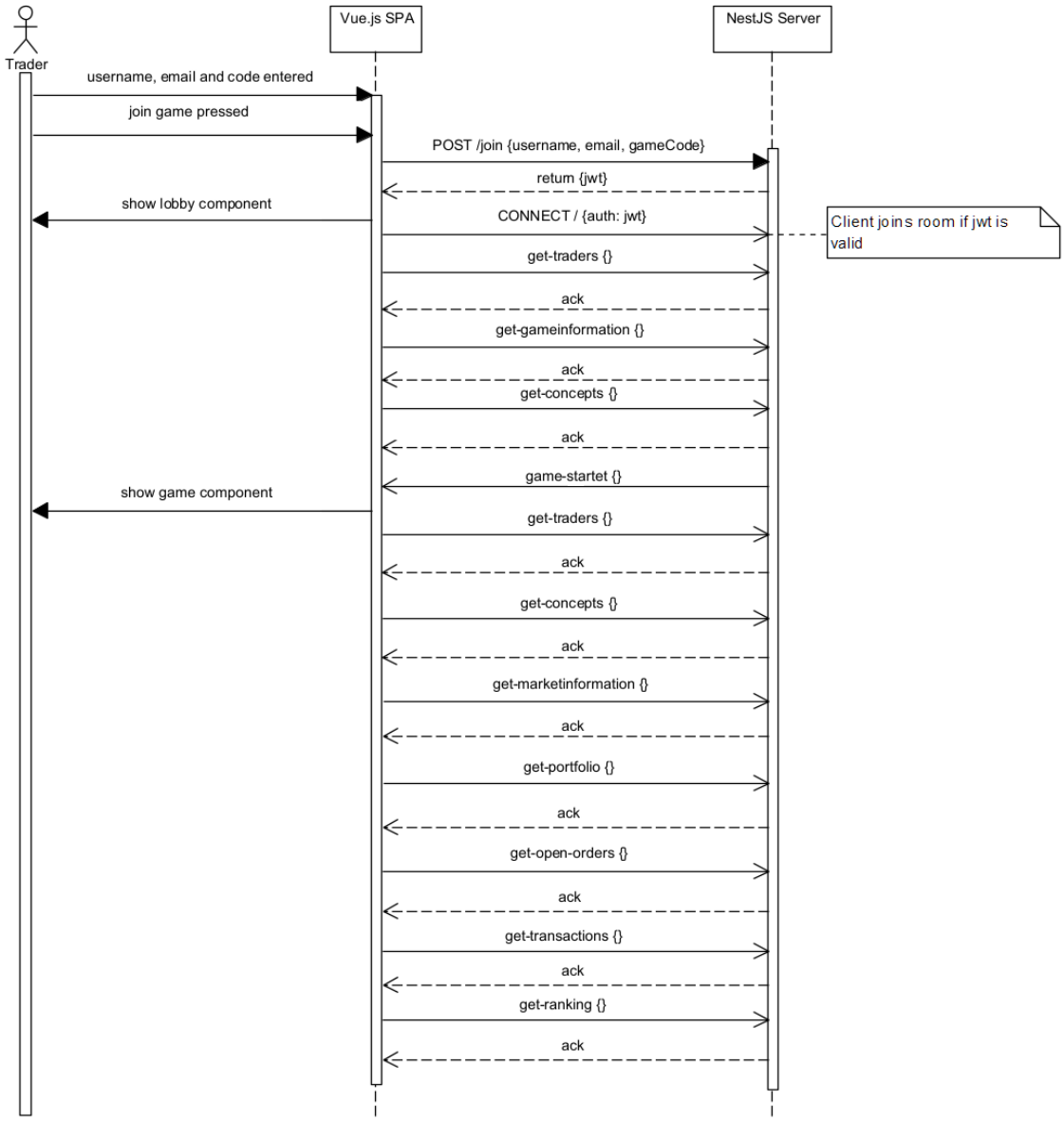


Abbildung 2.12: Sequenzdiagramm für den Spielbeitritt eines Traders

Spielbeitritt Organisator

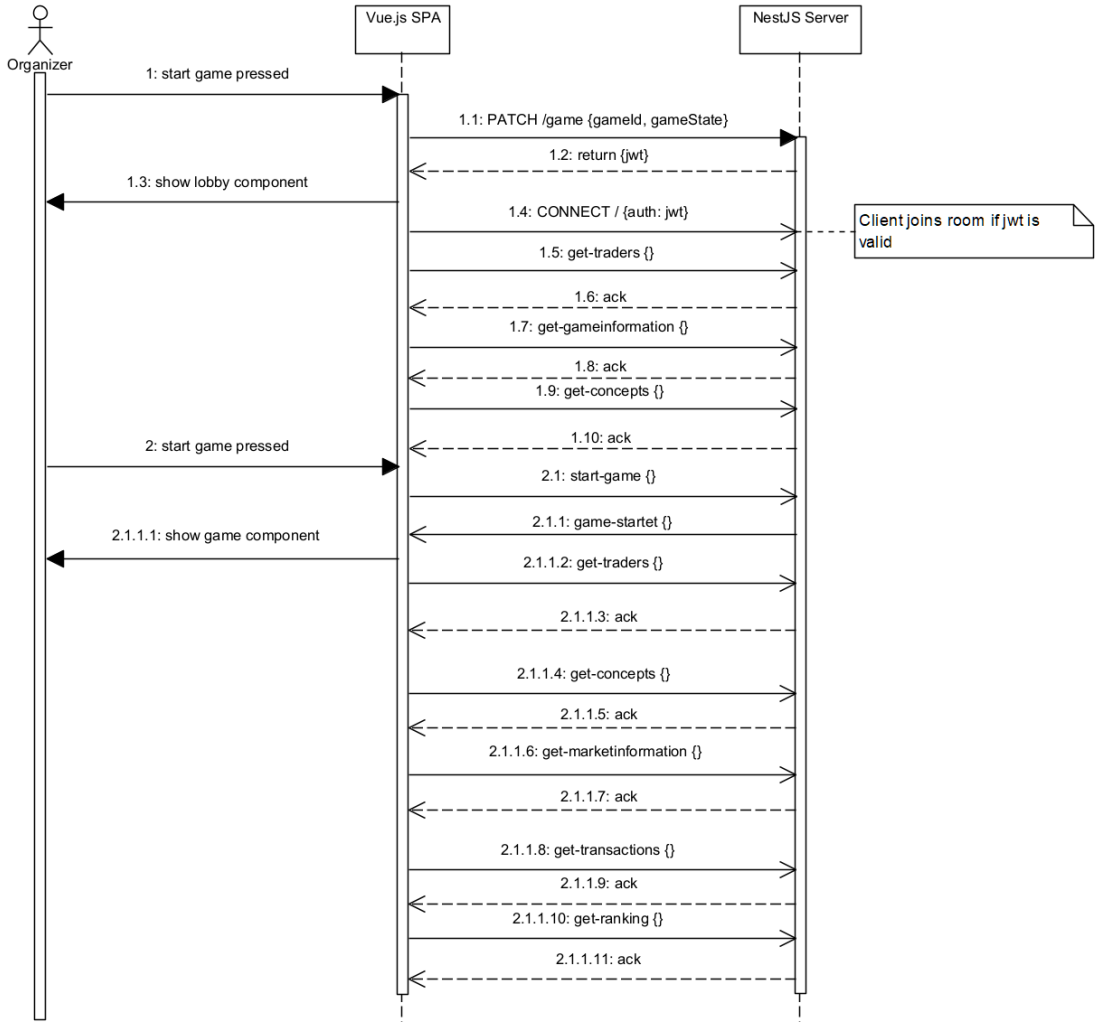


Abbildung 2.13: Sequenzdiagramm für den Spielbeitritt eines Organisators

2.3.7 CI/CD

Feature Branch Pipeline

Die Feature Branch Pipeline wird bei jedem push in einen *Feature* Branch, im Gitlab Repository, ausgeführt.

Ablauf Front- und Backend

1. Build Stage
    - Build
2. Test Stage
    - Unit Tests
3. Analyze Stage
    - Lint

Master Branch Pipeline Die *Master* Branch Pipeline wird ausgeführt, wenn *Feature* Branches in den *Master* Branch gemerged werden. In der Analyze Stage wird zusätzlich eine SonarQube Analyse durchgeführt.

- sonarqube-check

Release Branch Pipeline

Ein Deployment wird bei jedem push in den *Release* Branch ausgelöst.

Es wird die vollständige *Feature* Branch Pipeline durchgeführt und am Ende zusätzlich die folgende Release Stage:

- Release

2.3.8 Deployment

Die Software wird containerisiert deployed. Für das Front- und Backend wird jeweils ein eigenes Docker Image erstellt. Vor dem Front- und Backend wird ein Reverse-Proxy zwischengeschaltet, um CORS Fehler zu vermeiden und die Handhabung von HTTPS zu vereinfachen. Das Deployment-Diagramm in Abbildung 2.14 zeigt, wie die Applikation während der Entwicklung deployed wird.

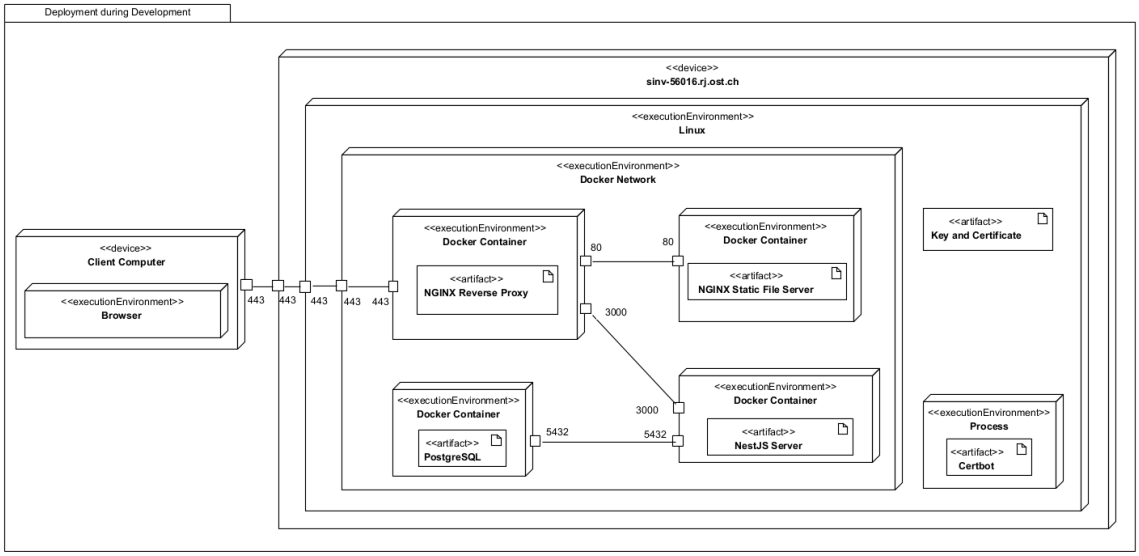


Abbildung 2.14: Deployment Diagramm

2.3.9 Authentication

Die Authentisierung von Organisatoren wird mittels JSON Web Tokens (JWT) realisiert. Nach einem erfolgreichem Login wird dem Organisator das HTTPOnly-Cookie "organizer-jwt" gesetzt. Das Cookie ist ein JWT Token und enthält die vier Key-Value-Pairs: id, username, iat und exp. Als Signaturalgorithmus wird HS256 (HMAC mit SHA256) verwendet.

Das Hashing der Organisatoren Passwörter erfolgt mittels der bcrypt-Hashfunktion mit einem Kostenfaktor von acht (entspricht 256 Hashing-Iterationen).

Beim Start eines STOC-Spieles erhält der Organisator das Cookie "organizer-game-jwt". Das Cookie ist ein JWT-Token und beinhaltet id, gameId, gameCode, username, iat und exp.

Bei Tradern, die einem Spiel beitreten findet keine Authentisierung statt. Damit die Trader identifiziert werden können, erhalten sie ein JWT-Token im HTTP-Cookie "trader-game-jwt" mit folgendem Inhalt: id, gameId, gameCode, nickname, email, iat, exp.

Die Verwendung von Cookies erlaubt es, dass Trader und Organisatoren nach dem Verlassen und erneutem Besuchen von Stockit wieder in das beigetretene Spiel hinzugefügt werden können, solange das Cookie vorhanden ist und das Spiel noch nicht abgeschlossen wurde.

2.3.10 Logging

Das Logging erfolgt in der Konsole. Der Vue.js Static File Server loggt alle HTTP-Anfragen. Der NestJS Server loggt alle HTTPS-Anfragen sowie ausgewählte Socket.io Events und allfällige Fehlermeldungen.

2.3.11 Persistenz

Der NestJS Server kommuniziert über TypeORM mit der PostgreSQL Datenbank. Das Datenbankschema wird mittels des Code-First Ansatzes generiert. Zur Verhinderung eines Datenverlustes in der Zukunft werden Migrationen verwendet.

Datenmodell

Das physische Datenmodell wurde als Entity-Relationship-Modell, welches in der Abbildung 2.15 ersichtlich ist, umgesetzt. Der Primärschlüssel der Tabelle Trader ist eine UUID, da Kommunikation zwischen der Datenbank und der Applikation während der Durchführung eines STOC-Spieles auf ein Minimum beschränkt werden soll. Deshalb sollte die ID der Trader nicht von der Datenbank vergeben werden. Da gleichzeitig mehrere STOC-Spiele durchgeführt werden können, muss aber garantiert werden, dass die Trader über alle zu einem Zeitpunkt stattfindenden Spiele eine eindeutige ID haben. Dies kann mittels einer UUID erreicht werden.

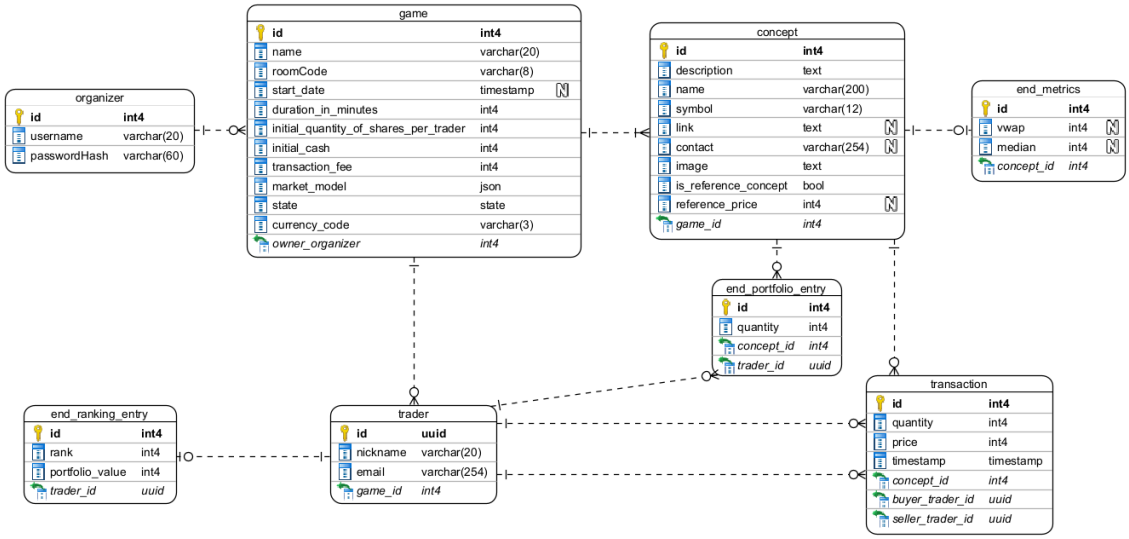


Abbildung 2.15: Physisches Datenmodell als Entity-Relationship-Modell

Speicherverbrauch

Dieser Abschnitt befasst sich mit dem geschätzten Speicherbedarf der Applikation. Die Tabelle 2.9 gibt eine Übersicht über den Speicherverbrauch pro Datentyp. Die Angaben sind der PostgreSQL Dokumentation entnommen [14].

Datentyp	Speicherverbrauch (Byte)
boolean	1
int4	4
text und varchar() der Länge n bis 126 Bytes	$1 + n$
text und varchar() der Länge n ab 127 Bytes	$4 + n$
timestamp	8
UUID	16

Tabelle 2.9: Speicherverbrauch der verschiedenen Datentypen

Ausgehend davon wurde der Speicherverbrauch pro Zeile pro Tabelle berechnet. Für die Länge der text und varchar() Attribute wurden jeweils durchschnittliche Werte eingesetzt. Es wurde expe-

rimentell ermittelt, dass das JSON um das Marktmodell Continuous Double Auction zu speichern 184 Byte Speicher benötigt. Die Reihenfolge der Terme in der Speicherverbrauch-Spalte entspricht der Reihenfolge der Attribute im Entity-Relationship-Modell.

Tabelle	**Speicherverbrauch (Byte)**
organizer	$4 + 21 + 61 = 182$
game	$4 + 21 + 8 + 8 + 4 + 4 + 4 + 4 + 184 + 21 + 4 + 4 = 270$
concept	$4 + 604 + 204 + 12 + 204 + 258 + 204 + 1 + 4 + 4 = 1499$
end_metrics	$4 + 4 + 4 + 4 = 16$
end_portfolio_entry	$4 + 4 + 4 + 16 = 28$
end_ranking_entry	$4 + 4 + 4 + 4 = 16$
trader	$16 + 21 + 258 + 4 = 299$
transaction	$4 + 4 + 4 + 8 + 4 + 16 + 16 = 56$

Tabelle 2.10: Durchschnittlicher Speicherverbrauch pro Zeile pro Tabelle

2.4 Designdokumentation

Für das Frontend wurde die Design Library Primevue gewählt, die Begründung für diesen Entscheid ist im Kapitel 2.1.2 zu finden. Für die Icons wird Primeicons verwendet, da Primevue Komponenten diese benötigen und diese auch direkt in die Designsprache passen. Die Wireframes sind im Anhang H zu finden.

Farbe

Die Farbe Blau (#0A65DB) wurde als Primärfarbe gewählt, da Stockit einen finanziellen Aspekt mit sich bringt. Blau wird oft mit Vertrauenswürdigkeit verbunden und hat eine beruhigende Wirkung. Damit sich die Sekundärfarbe von der Primärfarbe abheben kann wurde die Farbe Orange (#FA9600) gewählt. Diese und weitere verwendete Farben sind in der Abbildung 2.16 zu sehen.

Abbildung 2.16: Farbpalette

Schriftart

Für die Schrift wurde Inter verwendet. Inter ist eine serifenlose Schrift und ist unter der Open Font License lizenziert. Die Lizenz erlaubt die kommerzielle und anderweitige Nutzung der Schriftart für Produkte und Projekte im Print wie auch Digital.

Inter

Abbildung 2.17: Schriftart Inter

Logo

Für das Logo wurden verschiedene Designs erarbeitet. In der Abbildung 2.18 sind die Logo Design ersichtlich welche in Betracht gezogen wurden. Es wurde sich geeinigt nur die Wortmarke als Logo zu verwenden. Das Logo welches verwendet wurde ist in der Abbilung 2.19 zu sehen.

Abbildung 2.18: Alternative Logo Designs

Abbildung 2.19: Logo

Favicon

Für das Favicon wurden ebenfalls verschiedene Designs erarbeitet. In der Abbildung 2.20 sind die erarbeiteten Favicon Designs ersichtlich. Die Entscheidung fiel auf ein schlichtes, zweifarbiges “S”, welches in der Abbildung 2.21 zu sehen ist.

Abbildung 2.20: Alternative Favicon Designs

Abbildung 2.21: Favicon

2.5 Spiel Initialisierung

Während des 5. Review Meetings kam die Diskussion auf, ob die Trader zu Beginn handeln würden, da jeder Trader initial bereits ein Portfolio an Konzepten besitzt. Dieses Kapitel beschäftigt sich damit, ob bzw. wie stark diese Problematik vorhanden ist und welche Massnahmen getroffen werden können, um die Trader zu Beginn des Spieles zum Handeln zu motivieren.

2.5.1 Analyse des Problems

Wie von Chan et al. [3] beschrieben wird zu Beginn kein Startpreis festgesetzt. Stattdessen erfasst der Organisator bei der Erstellung des STOC-Spiels ein Referenzkonzept, dass als Numéraire fungiert.

Da die Trader zu Beginn keine Preise sehen, könnte es vorkommen, dass die Trader zu Beginn abwarten und beobachten was die anderen tun und folglich kein Handel stattfindet. Deshalb werden im folgenden Abschnitt einige Ansätze diskutiert, die den Handel zu Beginn des Spieles fördern sollen.

2.5.2 Vorschläge zur Mitigation des Problems

Dieser Abschnitt beschäftigt sich mit möglichen Ansätzen, die den Handel zu Beginn eines STOC-Spieles fördern.

Gleicher Startpreis für alle Konzepte

Es wird kein Referenzkonzept erstellt, dafür gibt der Organisator einen initialen Preis an, der für alle Konzepte gilt. Diese Variante wurde in einer früheren Version der Software verwendet. Der Vorteil dieser Variante ist es, dass es für die Trader verständlicher ist, wenn sie einen initialen Preis anstelle eines Referenzkonzeptes sehen. Allerdings ist diese Variante nicht praktikabel, da durch den initialen Preis zu Beginn die Konzepte mit einem Wert (in diesem Fall dem gleichen Wert für alle Konzepte) assoziiert werden. Der Ankereffekt [6] führt dazu, dass die Trader nahe am initialen Preis handeln und somit den tatsächlichen Wert der Konzepte nur bedingt berücksichtigen.

Zufällige Startpreise pro Konzept

Die Initialisierung mit zufälligen Startpreisen ist im Grunde genommen identisch mit der Initialisierung mit gleichen Startpreisen für alle Konzepte, da die Distanz zwischen dem konstanten Startpreis und und den jeweiligen Gleichgewichtspreisen als zufällig betrachtet werden kann, da der Startpreis frei vom Organisator gewählt werden kann.

Preise basierend auf Voting

Vor dem Beginn des Handels wird eine Umfrage durchgeführt, in der die Trader eine Rangliste der Konzepte erstellen. Der initiale Preis soll dann aufgrund des Ergebnisses der Rangliste berechnet werden (Konzepte, oben auf der Rangliste erhalten einen hohen Startpreis). Auch dieser Ansatz ist nicht zielführend, denn wenn die Initialpreise basierend auf einer Rangliste berechnet werden, ist der Initialpreis bereits recht nahe am Gleichgewichtspreis, was dazu führt, dass weniger gehandelt wird. Ausserdem stellt sich die Frage, wie sinnvoll dieser Ansatz ist, wenn man bedenkt, dass STOC eigentlich eine Alternative zu existierenden Methoden der Präferenzmessung sein soll.

Bank

Jeder Trader startet mit einem Portfolio, dass ausschliesslich Geld aber keine Anteile an Konzepten enthält. Die Ausgabe der Konzeptanteile erfolgt analog zur Emission von Aktien: über eine Bank können die Trader Anteile erwerben. Dieses Verfahren würde die Initialisierung interaktiver gestalten, da sich die Trader nun aktiv einbringen müssen, um überhaupt Konzeptanteile zu besitzen. Die Implementierung würde sich allerdings aufwändig gestalten. So müsste ein Mechanismus für die Festsetzung der Preise für die Konzeptanteile, die bei der Bank erworben werden können, gefunden werden. Des Weiteren müsste abgeklärt werden, ob die Anzahl der totalen Anteile pro Konzept über das ganze Spiel konstant ist bzw. wenn nicht, wie dies realisiert werden könnte.

2.5.3 Ergebnis

Die einzige alternative Initialisierungsstrategie die nicht massive Nachteile nach sich zieht, ist die Bank, da alle anderen Strategien von einem festgelegten Initialpreis ausgehen. Da die Festsetzung eines sinnvollen Emissionspreises für die Konzeptanteile, die bei der Bank erworben werden können, allerdings eine Auktion oder einen ähnlichen Mechanismus erfordert, würde man im Grunde genommen einen zweiten Markt implementieren. Die Implementierung eines solchen Verfahrens geht über den Rahmen dieser Studienarbeit hinaus.

Kapitel 3

Validierung

In diesem Kapitel wird die Validierung der Software beschrieben. Die durchgeführten Validierungsmethoden sollen die Robustheit des Systems und die Erfüllung der Anforderungen testen. Die Überprüfung der Anforderungen wurde mittels Systemtests bewerkstelligt. Um die Benutzerfreundlichkeit der Software zu testen, wurden Usability-Tests durchgeführt. Mittels eines Security Audits wurde Stockit auf die gängigsten Sicherheitslücken überprüft. Die Applikation sollte fähig sein, mit mehreren Tradern in einem Spiel umzugehen, hierfür wurde eine Livedurchführung mit 6 Trader durchgeführt. Es ist anzumerken, dass mehr als 6 Trader einem Spiel beitreten können, jedoch ist dieses Szenario nicht getestet worden, da der zeitliche Rahmen einer Studienarbeit grössere Tests nicht zugelassen hat.

3.1 Anforderungen

Die Tabellen 3.1 und 3.2 zeigen welche funktionalen (Use Cases)- und nicht-funktionalen Anforderungen erfüllt wurden.

Funtionale Anforderungen	Status
UC01: Erstellung und Anmeldung Organisator Account	erfüllt
UC02: STOC-Spiel erstellen und konfigurieren	erfüllt
UC03: STOC-Spiel beitreten	erfüllt
UC04: Echtzeitübersicht für Organisator	erfüllt
UC05: Am Handel teilnehmen	erfüllt
UC06: Order aufgeben	erfüllt
UC07: Order stornieren	erfüllt
UC08: Marktinformationen abrufen	erfüllt
UC09: Produktkonzepte ansehen	erfüllt
UC10: STOC-Spiel beenden	erfüllt
UC11: Auswertung des STOC-Spieles ansehen	erfüllt

Tabelle 3.1: Validation der funktionalen Anforderungen

Nicht-funktionale Anforderungen	Status
NFR01: Verhalten bei Systemabsturz	erfüllt
NFR02: Verfügbarkeit	erfüllt
NFR03: Anzahl Konzepte pro STOC-Spiel	erfüllt
NFR04: Maximale Anzahl parallel durchführbare STOC-Spiele	erfüllt
NFR05: Maximale Anzahl Trader pro STOC-Spiel	erfüllt
NFR06: Maximale Zeit bis aktualisierte Marktinformationen beim Trader ankommen	erfüllt
NFR07: Organisator Accounts	erfüllt
NFR08: Zutritt zu STOC-Spielen	erfüllt
NFR09: Co-Existenz mit anderen Systemen	erfüllt
NFR10: Lernbarkeit	nicht erfüllt
NFR11: Schutz vor Fehlern	erfüllt
NFR12: Wiedereinstieg in das STOC-Spiel für Trader	erfüllt
NFR13: Lokalisierung	erfüllt
NFR14: Austauschbarkeit des Marktmechanismus	erfüllt
NFR15: Dokumentation der Schnittstellen	erfüllt
NFR16: Testbarkeit	erfüllt
NFR17: Browserunterstützung	erfüllt
NFR18: Installierbarkeit	erfüllt

Tabelle 3.2: Validation der nicht-funktionalen Anforderungen

Wie in der Tabelle 3.2 ersichtlich ist, konnte die Anforderung “Lernbarkeit” nicht umgesetzt werden, da die Erklärung von komplexen Marktmechanismen mittels des User Interfaces über den zeitlichen Rahmen einer Studienarbeit hinaus gegangen wäre.

3.2 Systemtests

Während dieser Arbeit wurden zwei Systemtests durchgeführt. Der erste entstand am Ende des Meilensteins M4. Der zweite wurde am Ende des Meilensteins M5 durchgeführt. Aus den beiden Systemtests ist ersichtlich, dass die Anforderungen erfüllt wurden. Die ausführlichen Systemtests sind im Anhang C zu finden.

3.2.1 M4 Systemtest

Die Tabelle 3.3 zeigt, welche Tests durchgeführt wurden.

Test	Status	Kommentar
Valide Nutzername/Passwort Kombination	erfolgreich	-
Valide Logindaten	erfolgreich	-
Valide Konfigurationseingaben	erfolgreich	-
Lobby sehen (Organisator)	erfolgreich	-
Verbleibende Zeit, Anzahl Trader, Game Code und Sprache ansehen	erfolgreich	-
Aktuelle Marktinformationen ansehen	erfolgreich	-
Anzeige der Transaktionshistorie	erfolgreich	-
Vorzeitige Beendigung eines Spiels	erfolgreich	-
Lobby sehen (Trader)	erfolgreich	-
Ansehen eines Produktkonzepts	erfolgreich	-
Verbleibende Zeit, Anzahl Trader, Game Code und Sprache ansehen Handlung (Trader)	erfolgreich	-
Übersicht über Portfolio	erfolgreich	-
Übersicht über die offenen Orders	erfolgreich	-
Aktuelle Marktinformationen ansehen	erfolgreich	-
Ungenügende Balance	erfolgreich	-
Genügende Balance	erfolgreich	-
Ungenügende Anzahl Produktkonzepte im Portfolio	erfolgreich	-
Genügende Anzahl Produktkonzepte im Portfolio	erfolgreich	-
Order canceln	erfolgreich	-
Genügende Anzahl Produktkonzepte im Portfolio	erfolgreich	-
Genügende Anzahl Produktkonzepte im Portfolio	erfolgreich	-

Tabelle 3.3: Übersicht durchgeführter Systemtests M4

3.2.2 M5 Systemtest

Die Systemtests von M4 wurden nochmals durchgeführt. Die Auswertung ist in der Tabelle 3.4 ersichtlich.

Test	Status	Kommentar
Valide Nutzername/Passwort Kombination	erfolgreich	-
Valide Logindaten	erfolgreich	-
Valide Konfigurationseingaben	fehlgeschlagen	Die Konfigurationsmöglichkeiten wurden geändert
Lobby sehen (Organisator)	erfolgreich	-
Verbleibende Zeit, Anzahl Trader, Game Code und Sprache ansehen	erfolgreich	-
Aktuelle Marktinformationen ansehen	erfolgreich	-
Anzeige der Transaktionshistorie	erfolgreich	-
Vorzeitige Beendigung eines Spiels	erfolgreich	-
Lobby sehen (Trader)	erfolgreich	-
Ansehen eines Produktkonzepts	erfolgreich	-
Verbleibende Zeit, Anzahl Trader, Game Code und Sprache ansehen Handlung (Trader)	erfolgreich	-
Übersicht über Portfolio	erfolgreich	-
Übersicht über die offenen Orders	erfolgreich	-
Aktuelle Marktinformationen ansehen	erfolgreich	-
Ungenügende Balance	erfolgreich	-
Genügende Balance	erfolgreich	-
Ungenügende Anzahl Produktkonzepte im Portfolio	erfolgreich	-
Genügende Anzahl Produktkonzepte im Portfolio	erfolgreich	-
Order canceln	erfolgreich	-
Genügende Anzahl Produktkonzepte im Portfolio	erfolgreich	-
Genügende Anzahl Produktkonzepte im Portfolio	erfolgreich	-

Tabelle 3.4: Übersicht durchgeführter Systemtests M4 in M5

Zusätzlich wurden folgende Systemtests 3.5 durchgeführt:

Test	Status	Kommentar
Als Organisator möchte ich das STOC-Spiel zu Beginn konfigurieren können	erfolgreich	-
VWAP und Median sehen	erfolgreich	-
Rangliste sehen	erfolgreich	-
Transaktionen sehen	erfolgreich	-
Transaktionen exportieren	erfolgreich	-

Tabelle 3.5: Übersicht durchgeführter Systemtests M5

3.3 Usability Test

Der Usability Test wurde am 04.12.2021 mit zwei Probanden durchgeführt. Der Usability Test berücksichtigt nur die Perspektive des Traders. Mit diesem Test soll ermittelt werden, ob Stockit benutzerfreundlich ist. Die Schlussbefragung ergab, dass einige Marktbegriffe unklar sind und einige Begriffe angepasst werden müssen. Die Anwendung wurde teilweise nach den Befunden des Usability Tests angepasst. Weitere Informationen zum Usability Test befinden sich im Anhang D.
Die Aufgaben und Resultate sind in der Tabelle 3.6 gelistet.

Aufgabe	**Bemerkung**
Treten Sie dem Spiel mit dem Room Code 92371932 bei	-
Wie viele andere Trader sind in der Lobby?	Proband war etwas überrascht, dass Lobby nicht als solche erkennbar war
Beschreiben Sie kurz welche Produktkonzepte zur Auswahl stehen	-
Wie viel Geld steht Ihnen zur Verfügung?	-
Wie viele Anteile am Konzept PLFM besitzen Sie?	Proband hat im Markt nach dem Anteil gesucht und dachte, dass gehandelte Volumen entspreche der Anzahl Anteile im Besitz
Zu welchem Preis wurde der SMS zuletzt gehandelt?	Proband hat nicht verstanden was Ask und Bid sind
Steht gegenwärtig ein Anteil von SMS zum Verkauf?	Proband hat Kauf Order aufgegeben, um herauszufinden, ob SMS zum Verkauf steht
Verkaufen Sie 5 Anteile von APPARCH für 150	Proband war etwas unsicher beim Verkauf der Anteile, ob es sich um den totalen Preis oder den Preis pro Anteil handelt
Kaufen Sie 7 Anteile von WED1 für 50	-
Welche Orders haben Sie momentan offen?	Proband hat offene Orders in der Portfolio-Info-Bar gesucht und war verwirrt, das seine Sell Order dort nicht aufzufinden war und Proband hat Orders Tab im Portfolio nicht gefunden
Annullieren Sie die erstellte Verkaufs-Order	-
Wie stehen Sie im Vergleich zu anderen Tradern da?	-

Tabelle 3.6: Resultate des Usability Tests

3.4 Security Audit

Um Sicherheitslücken aufzudecken, wurde ein kurzer Security Audit durchgeführt.

3.4.1 Cross-Site-Scripting

Vorgehen

1. Erstellung eines Spieles
2. Ein Konzept wird mit dem Namen "<script>alert('XSS')</alert>" erfasst
3. Das Spiel wird erstellt und geladen

Ergebnis Sowohl für Trader wie auch für den Organisator ist der Name des Konzeptes "<script>alert('XSS')</alert>". Bei den Tradern und dem Organisator wurde der Javascript Code nicht ausgeführt. Dieses Ergebnis ist zu erwarten, da Vue.js Templates automatisch escaped.

3.4.2 SQL-Injection

Vorgehen

1. Registrierung mit dem Nutzernamen "testnutzer; DROP TABLE game"

Ergebnis Ein Organisator mit dem Nutzername "testnutzer; DROP TABLE game" wurde registriert. Die Tabelle "game" wurde nicht gelöscht. Das Ergebnis ist zu erwarten, da im Back-End sämtliche Abfragen über TypeORM getätigt werden, welches (solange keine Raw-SQL Queries gemacht werden) SQL-Injection verhindert.

3.4.3 Bekannte Schwachstellen

- Kein Schutz vor Denial of Service Angriffen
- Unbegrenzte Anzahl Login-Versuche ermöglichen Brute-Force und Dictionary Angriffe
- Kein Schutz vor Cross Site Request Forgery

3.5 Testdurchführung

Die Testdurchführung wurde am 07.12.2021 mit 6 Tradern durchgeführt. Dieser Test dient dazu, die Stabilität und Integrität des Marktes zu prüfen. Ausserdem wurde getestet, ob mit Stockit ein STOC-Spiel erstellt und durchgeführt werden kann. Das Resultat der Durchführung war, dass einige Probleme aufgetreten sind, jedoch wurden diese adressiert und sind in dieser Sektion aufgeführt. Im Anhang E wird die Durchführung detaillierter beschrieben.

3.5.1 Setup

Das Spiel wurde gemäss Tabelle 3.7 konfiguriert.

Konfiguration	Wert
Startkapital	1000.– CHF
Anfangsmenge der Konzepte pro Händler	10
Anfangspreis	100.– CHF
Transaktionsgebühr	5%
Marktmodell	Continuous Double Auction mit Preis-Zeit-Priorität und der Ask/Bid Strategie
Dauer	40 Minuten

Tabelle 3.7: Konfiguration der Testdurchführung

Konzepte:

- WV EC2000
- WV HComfort
- WV RoadStar 2
- WV Punto
- WV Nucleon
- WV Forest Master

3.5.2 Auswertung

Technische Probleme

- Falsche Berechnung des verfügbaren Kapitals im Backend
- Transaktionen in der Transaktionshistorie falsch sortiert
- Trader blieben im Portfolio, da im Markt nur gekauft werden kann.
- Im Portfolio sind keine näheren Informationen zu den Konzepten ersichtlich

Adressierung Das Backend wurde angepasst, so dass die Berechnung stimmt. Im Frontend wurden die Transaktionen richtig sortiert und das Problem, dass die Trader ausschliesslich aus ihrem Portfolio handeln, wurde dadurch gelöst, dass im Markt auch verkauft werden kann. Das Problem, dass keine näheren Informationen zu den Konzepten im Portfolio ersichtlich sind, wurde nicht adressiert, damit die Trader motiviert sind, diese im Markt zu suchen und auch gleich dort zu handeln.

Probleme während der Spiel Durchführung

- Das Spiel kam ins Stocken, da einige Trader das Geld angehäuft hatten und dieses nicht ausgeben wollten
- Preismanipulation am Ende des Spiels

Adressierung Um das Problem, dass das Spiel ins Stocken gerät zu lösen, ist es nicht empfehlenswert das Startkapital zu erhöhen, da ein enges Budget die Trader ermutigt, nicht benötigte Anteile zu verkaufen. Gute Werte für das Startkapital und Anzahl initiale Anteile pro Trader müssten experimentell ermittelt werden. Um Preismanipulationen am Ende eines Spieles zu verhindern, sollte der Organisator das Spiel manuell beenden, bevor die vollständige Spielzeit abgelaufen ist.

3.6 Code Metriken

Die Code Metriken wurden aus Sonarqube ausgelesen. Die grosse Anzahl an Klassen im Backend ist auf die Anwendung des DTO Patterns zurückzuführen. Die Code Duplikation im Backend ist auf die Tests beschränkt. Die Unit-Tests wurden mit Jest durchgeführt und für die Aufbereitung für Sonarqube wurde jest-sonar-reporter benutzt.

Metrik	Wert
Lines of Code	4853
Funktionen	497
Klassen	135
Files	97
Duplications	1.75%
Code Smells	0

Tabelle 3.8: Backend Code Metriken

Metrik	Wert
Lines of Code	3556
Funktionen	444
Klassen	11
Files	59
Duplications	0.0%
Code Smells	0

Tabelle 3.9: Frontend Code Metriken

3.6.1 Test Coverage

Da in dieser Arbeit die Features im Vordergund standen wurden im Backend nur die wichtigsten Funktionen mit Unit-Tests getestet und im Frontend wurde getested, ob die Seiten auch die richtigen Elemente darstellen.

(a) Backend Code Coverage

(b) Frontend Code Coverage

Abbildung 3.1: Code Coverage

Kapitel 4

Schlussfolgerungen

4.1 Bewertung der Ergebnisse

Innerhalb von 14 Wochen konnte mit Stockit eine Applikation entwickelt werden, die die wichtigsten Aspekte der Aufgabenstellung erfüllt. Wie aus dem im Anhang F hervorgeht, ist Stockit Open Source. Die sehr offen formulierte Aufgabenstellung erforderte eine ausführliche Anforderungsanalyse. Um geeignete Anforderungen zu identifizieren wurde eine gründliche Domänenanalyse durchgeführt. Diese beinhaltete primär das Lesen der Paper, die zu STOC verfasst wurden, es wurden aber auch Recherchen zur Ermittlung eines geeigneten Marktmodells durchgeführt, die in der Wahl eines Continuous Double Auction Market endeten. Aufbauend darauf konnten mit Vue.js, NestJS und PostgreSQL geeignete Technologien zur Umsetzung von Stockit eruiert werden. Mit der Softwarearchitektur wurde ein solides Fundament für die Construction-Phase geschaffen. Stockit wurde während der Construction-Phase laufend in Form von Unit- und Integrationstests validiert. Gegen Ende der Arbeit wurde zudem ein Usability-Test sowie eine Testdurchführung eines STOC-Spieles durchgeführt.

Im Verlaufe des Projektes sind einige Herausforderungen aufgetreten, auf die dank des agilen Projektmanagements flexibel reagiert werden konnte.

Die Anforderung, dass das Marktmodell austauschbar sein muss, hat die Komplexität der Software stark erhöht und zu einigen Problemen geführt, so dass das Marktmodell-Interface im Verlauf der Construction-Phase mehrmals angepasst werden musste. Das entwickelte Marktmodell-Interface ist mit der Einschränkung verbunden, dass Transaktionen immer zwischen zwei Tradern stattfinden, was es nicht möglich macht einen Markt mit einem Market Maker zu implementieren. Auch die Anforderung, dass die einzelne Marktmodelle eigene Konfigurationsmöglichkeiten haben können (z.B. die Preisstrategie beim Continuous Double Auction Market), hat erheblichen Mehraufwand verursacht. Retrospektiv betrachtet hätte der Markt wohl besser nicht austauschbar implementiert werden sollen, da die einzige Aufgabe des Marktes die Aggregierung der Wertvorstellungen der Trader in Form von Preisen ist. Durch welches Marktmodell bzw. welche Konfiguration eines Marktmodells die Preise zustande kommen ist wohl von sekundärer Bedeutung für STOC, zumindest sind Dahan et al. in [5] nicht näher darauf eingegangen.

Marktmodelle und deren Konfigurationsmöglichkeiten müssen über die Klasse “MarketModellRegistry” registriert werden. Dies ist hat zur Folge, dass der Registrierungsprozess für neue Marktmodelle unübersichtlich ist, da zahlreiche Klassen involviert sind. Ein Umbau hin zur Nutzung von Reflection und Decorators würde den Registrierungsprozess vereinfachen und die Komplexität reduzieren. Bei der Erstellung der Softwarearchitektur wurde von der Verwendung von Reflection

und Decorators abgesehen, da Decorators nur experimentell von Typescript unterstützt werden [17]. Während der Implementierung des Backends wurde allerdings festgestellt, dass Decorators sehr oft von NestJS verwendet werden und dies keinerlei Probleme verursacht. Aus Mangel an Zeit wurde von einem Refactoring hin zu Decorators und Reflection abgesehen.

Ferner hat sich das Testen der Socket.io Schnittstelle als sehr aufwändig erwiesen. Deshalb wurde ein grosser Teil der Event-Handler im Server nur manuell mittels Postman getestet. Allerdings konnte diese Problematik durch eine solide Kapselung mitigiert werden. Die Socket.io Event-Handler beinhalten nämlich keine Business Logik. Die Business Logik wurde im Marktmodell bzw. in der Game Klasse untergebracht. Da diese Klassen nicht von Socket.io abhängig sind, konnten diese ausgiebig mittels automatisierter Unit-Tests validiert werden.

Trotz den existierenden Problemen sind ist das Projektteam mit Stockit insgesamt zufrieden. Für die aufgetretenen Herausforderungen konnte jeweils eine passende, wenn auch nicht immer optimale Lösung gefunden werden. Die gewählten Technologien haben sich bewährt. Das im Frontend im Einsatz stehende Vue.js ermöglichte es, dass User Interface aus wiederverwendbaren Komponenten zusammenzusetzen, was einen positiven Einfluss auf die Kohärenz und Wartbarkeit hat. Auch NestJS konnte gewinnbringend eingesetzt werden. Durch die Modularisierung wurde eine klare Struktur geschaffen, was die Testbarkeit erhöht. Ausserdem konnte durch die Nutzung der von NestJS zur Verfügung gestellten Module (bspw. JwtModule und TypeOrmModule) einiges an Zeit eingespart werden. Mit dem gegen Ende des Projektes durchgeführten Testdurchlauf konnte gezeigt werden, dass Stockit in der Praxis zur Anwendung der STOC-Methode, zumindest in einem kleinen Rahmen, verwendet werden kann.

4.2 Ausblick

Die ausführlich dokumentierten Schnittstellenbeschreibungen sowie die Modularität von Stockit bieten eine solide Grundlage für die Entwicklung weiterer Funktionalitäten.

Besonders hervorzuheben ist die in der Aufgabenstellung (optional) vorgesehene Anforderung, dass Stockit als verteiltes System betrieben werden kann. Dies konnte nicht umgesetzt werden, doch könnte dies beispielsweise unter anderem mittels des Redis-Adapter für Socket.io realisiert werden.

Ausserdem wäre es sinnvoll ausführlichere Tests in einem grösseren Rahmen durchzuführen. In einem ersten Schritt könnte dies im Kontext des OST Unterrichts erfolgen. Im Falle einer erfolgreichen Testdurchführung könnten weitere Tests unter realistischen Bedingungen, beispielsweise in Zusammenarbeit mit einem Unternehmen, durchgeführt werden.

Die User Experience könnte verbessert werden, indem Stockit besser auf verschiedene Bildschirmgrössen angepasst wird. Weitere Usability Tests, insbesondere mit einer heterogeneren Auswahl an Testpersonen würde möglicherweise zusätzliche Problemstellen offenlegen.

Momentan wird der Wert der Portfolios der Trader aus dem "Cash" und den gehaltenen Anteilen bewertet zum Last-Preis der jeweiligen Produktkonzepte ermittelt. Nach Dahan et al. [5] könnte die Bewertung der Anteile auch zum VWAP oder einer anderen Kennzahl erfolgen. Wie die Anteile bewertet werden, könnte konfigurierbar gemacht werden, so dass der Organisator zu Beginn des Spieles aus verschiedenen Möglichkeiten auswählen kann.

Dahan et al. [5] schlagen vor, dass die STOC-Spiele zu einem zufälligen Zeitpunkt enden. Mit Stockit hat der Organisator die Möglichkeit die Dauer eines STOC-Spieles bei der Erstellung festzulegen. Der Organisator hat aber die Möglichkeit ein Spiel vorzeitig zu beenden. In der Testdurchführung hat sich herausgestellt, dass wenn die Trader den exakten Zeitpunkt des Endes kennen, es zu Marktmanipulationen kommen kann. Um diesen Missstand zu beheben, sollte Stockit dahingehend angepasst werden, dass die Trader den exakten Zeitpunkt des Spielendes nicht kennen.

Um die Motivation der Trader zu erhöhen ihre wahren Präferenzen zu offenbaren, könnte statt fiktives Geld echtes Geld verwendet werden [5]. Am Ende eines STOC-Spieles könnte das Geld automatisiert über einen Zahlungsprovider an die Trader überwiesen werden.

Das Projektteam hält STOC für eine vielversprechende Methode zur Messung von Konsumentenpräferenzen und hofft mit dieser Studienarbeit dazu beizutragen, dass STOC den Weg von der Theorie in die Praxis findet.

Literatur

[1] *arc42*. 2021. URL: https://arc42.org/ (besucht am 11. 11. 2021).

[2] bestofjs. *2019 JavaScript Rising Stars*. 2020. URL: https://risingstars.js.org/2019/en#section-nodejs-framework (besucht am 12. 10. 2021).

[3] Nicholas T. Chan u. a. *Experimental Markets for Product Concepts*. Techn. Ber. Massachusetts Institute of Technology, Juli 2001.

[4] Tyler Charboneau. *Breaking Down SmartBear's 2020 State of API Report*. 2020. URL: https://nordicapis.com/breaking-down-smartbears-2020-state-of-api-report (besucht am 11. 10. 2021).

[5] Ely Dahan u. a. "Securities Trading of Concepts (STOC)". In: *Journal of Marketing Research* 48 (2011). DOI: 10.2139/ssrn.1163442.

[6] *Dorsch Lexikon der Psychologie. Ankereffekt*. URL: https://dorsch.hogrefe.com/stichwort/ankereffekt (besucht am 11. 12. 2021).

[7] Friedrich A. Hayek. "The Use of Knowledge in Society". In: *American Economic Review* 35 (1945), S. 519–530.

[8] *ISO/IEC 25010*. URL: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010 (besucht am 13. 10. 2021).

[9] Frank de Jong und Barbara Rindi. *The Microstructure of Financial Markets*. Cambridge University Press, 2009. DOI: 10.1017/CBO9780511818547.

[10] Adlar Jeewook Kim. "An Order Flow Model and a Liquidity Measure of Financial Markets". Diss. Massachusetts Institute of Technology, Sep. 2008.

[11] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. John Wait, 2004. ISBN: 013148906-2.

[12] *Open Source Licenses by Category*. URL: https://opensource.org/licenses/category (besucht am 21. 12. 2021).

[13] Simon Parsons u. a. *Everything you wanted to know about double auctions, but were afraid to (bid or) ask*. Techn. Ber. City University of New York, University of Liverpool, Jan. 2006.

[14] *PostgreSQL. Documentation*. URL: https://www.postgresql.org/docs/14/index.html (besucht am 16. 12. 2021).

[15] Gordon Scott. *Matching Orders*. 2021. URL: https://www.investopedia.com/terms/m/matchingorders.asp (besucht am 07. 10. 2021).

[16] *T7 Release 9.1. Market Model for the Trading Venue Xetra®*. Deutsche Börse AG, 2021.

[17] *Typescript. Reference: Decorators.* URL: https://www.typescriptlang.org/docs/handbook/decorators.html (besucht am 18. 12. 2021).

[18] Perukrishnen Vytelingum. "The Structure and Behaviour of the Continuous Double Auction". Diss. University of Southampton, 2006. URL: https://eprints.soton.ac.uk/263234/.

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

Ask Minimaler Preis, zudem ein Verkäufer bereit ist eine Wertschrift zu verkaufen. 3, 4, 9, 30

Bid Maximaler Preis, zudem ein Käufer bereit ist eine Wertschrift zu kaufen. 3, 4, 9, 30

Buy Limit Order dt. Kauforder mit einem Limit; Eine Kauforder, die nur zu einem Preis ausgeführt wird, der kleiner oder gleich dem in der Kauforder spezifizierten Preis ist.. 3, 4, 9

Continuous Double Auction dt. Markt, der auf einer fortlaufenden Doppelauktion basiert; Bei einer Doppelauktion können Käufer und Verkäufer Gebote abgeben.. 1–3, 36, 51

CSV Comma-separated values. 10

DTO Data transfer object. 50

FIFO First-In, First-Out. 4

JSX Javascript Syntax Extension. 13, 14

REST Representational state transfer. 18, 20, 23, 25, 30

Sell Limit Order dt. Verkaufsorder mit einem Limit; Eine Verkauforder, die nur zu einem Preis ausgeführt wird, der grösser oder gleich dem in der Verkauforder spezifizierten Preis ist.. 3, 4, 9

STOC Securities Trading of Concepts; Verfahren zur Identifizierung von profitablen Produktkonzepten. iv

Trader dt. Händler; Proband, der im Auftrag einer Organisation an einem STOC-Spiel teilnimmt.. iv

UUID Universally Unique Identifier. 35

VWAP Volume-weighted average price; Metrik zur Beurteilung der Performance eines Produktkonzeptes, die den Preis und das gehandelte Volumen berücksichtigt.. iv, 2, 10, 52, 78, 86

Anhang A

Aufgabenstellung

Aufgabenstellung Innovation-Trading App

September 24, 2021

1 Problembeschrieb

Das Erkennen erfolgreicher neuer Produktkonzepte kann ein herausfordernder Prozess sein, der Einblick in die Präferenzen privater Verbraucher erfordert. Um die Präferenzen der Verbraucher für neue Produktkonzepte zu messen, kann der STOC-Ansatz (Securities Trading of Concepts) angewendet werden, bei dem neue Produktkonzepte wie Finanztitel gehandelt werden.

Auf einem STOC-Markt handeln verschiedene Teilnehmer virtuelle Aktien, die jeweils im Zusammenhang mit einem Produkt oder Dienstleistungskonzept stehen. Jeder Teilnehmer erhält ein anfängliches Portfolio aus virtuellem Bargeld und virtuellen Aktien. Das Ziel des STOC-Spiels besteht darin, dass jeder Teilnehmer den Wert seines Portfolios zu maximieren versucht.

Um ein STOC-Spiel durchzuführen, ist eine spezialisierte Trading-Software notwendig, welche im Rahmen dieser Arbeit entwickelt werden soll.[1]

2 Auftrag

Die Software soll als Single-Page Web-Applikation umgesetzt werden. Das User-Interface soll nach dem Prinzip "Desktop-First" entworfen werden.

Die Innovation-Trading App wird als Software as a Service realisiert. Daher soll es für die Verantwortlichen eines Unternehmens möglich sein, ein STOC-Spiel zu erstellen, durchzuführen und die Resultate anzusehen ohne selbst Software aufsetzen zu müssen.

Vor der Durchführung eines STOC-Spiels soll der Veranstalter die Möglichkeit haben, die handelbaren Produktkonzepte zu definieren, sowie einige Einstellungen wie beispielsweise das Startkapital festzulegen.

Die Funktionalitäten, die bei einer Durchführung eines STOC-Spiels zur Verfügung stehen, sollen analog zum vom MIT im Rahmen des Papers *Securities Trading of Concepts (STOC), 2011, Ely Dahan et al.* entwickelten Prototypen sein. Insbesondere soll es Tradern möglich sein, Orders zu platzieren und zu stornieren, Informationen zum Markt zu erhalten und das eigene Portfolio einzusehen.

[1] Dem Arbeitsverwaltungstool entnommen

Am Ende soll der Veranstalter eine Übersicht über die Resultate erhalten. Diese soll persistent zur Verfügung stehen, damit auch die Resultate früherer STOC-Spiele einsehbar sind.

Die Applikation wird so entworfen, dass die optionale Erweiterungsmöglichkeit "Horizontale Skalierung" mit geringem Aufwand hinzugefügt werden kann.

Daneben besteht die Arbeit aus einer kurzen Erläuterung der STOC-Theorie.

2.1 Optionale Erweiterungsmöglichkeiten

- Horizontale Skalierung implementieren
- Export der Resultate eines STOC-Spiels z.B. in CSV
- User-Interface für Smartphones

3 Beteiligte Personen

Person	**Funktion**
Prof. Dr. Daniel Patrick Politze	Betreuer
Abinas Kuganathan	Studierender
Jonas Knupp	Studierender

4 Termine

Bezeichnung	**Termin**
Anfang	21.09.2021
Ende	24.12.2021

Prof. Dr. Daniel Patrick Politze

Anhang B

Benutzungsanweisung

B.1 Installationsanleitung

B.1.1 Voraussetzungen

Die folgenden Programme müssen zur Installation und Ausführung von Stockit installiert sein:

- Server mit offenen Ports 80 und 443 sowie Ubuntu 20 LTS
- Docker, mind. Version 20
- Docker Compose, mind. Version 1
- Entpackter "source"-Ordner der abgegeben Archivdatei (ZIP-File)

B.1.2 Vorgehen

Der "source"-Ordner muss auf den in den Voraussetzungen erwähnten Server verschoben werden. Im Folgenden wird davon ausgegangen, dass alle Befehle mit einer Konsole, die im "source"-Ordner ist, ausgeführt werden. Alle erwähnten Dateien befinden sich im "source"-Order, oder einem Unterordner davon.
Als erstes muss die *docker-compose.yml* Datei bearbeitet werden.

Umgebungsvariablen NestJs
Für den Service "nestjs-server" stehen die in Tabelle B.1 aufgelisteten Umgebungsvariablen zur Verfügung. Falls die mitgelieferte Datenbank verwendet wird, sollte ausschliesslich die Umgebungsvariable "JWTSECRET" einmalig geändert werden.

Umgebungsvariable	Bedeutung	Default
ENVIRONMENTMODE	Deploymentmodus	production
DBHOST	(Docker-) Host der Datenbank	postgres
DBPORT	Port der Datenbank	5432
DBUSERNAME	Username über den auf die Datenbank zugegriffen wird	admin
DBPASSWORD	Dass zum Username gehörende Passwort	Hh9bflPGx
DBNAME	Name der Datenbank, in der die Daten gespeichert werden	stockitdb
JWTSECRET	Secret, dass zur Generierung von JWT-Tokens verwendet wird	9XrfpOo

Tabelle B.1: Umgebungsvariablen nestjs-server

Umgebungsvariablen Postgres
Für den Service "postgres" stehen die in Tabelle B.2 definierten Umgebungsvariablen zur Verfügung. Falls die mitgelieferte Datenbank verwendet wird, sind keine Anpassungen nötig.

Umgebungsvariable	Bedeutung	Default
POSTGRES_DB	Name, der von PostgreSQL zu erstellenden Datenbank	stockitdb
POSTGERS_USER	Name, des von PostgreSQL zu erstellenden Users	admin
POSTGERS_PASSWORD	Password, für den von PostgreSQL erstellten User	Hh9bflPGx

Tabelle B.2: Umgebungsvariablen postgres

Volumes
Im Volume "stockit-postgres-data" werden die Daten der Datenbank gespeichert.
Das Volume "stockit-image-data" enthält die Bilder der gespeicherten Produktkonzepte.

Frontend Anpassungen (Optional)
Die Farben für das Frontend können in der Datei *./stoc_front-end/src/assets/tailwind-light/theme.css* angepasst werden. Um beispielsweise die Primärfarbe zu ändern, müssen alle Vorkommen von "#0a65db" durch den Hex-Wert der gewünschten Primärfarbe ersetzt werden.
Die Anpassungen an den Texten der Applikation kann im Ordner *./stoc_front-end/src/translations* vorgenommen werden. Die Texte sind in der jeweiligen Datei für die Sprache als JSON abgelegt. In der gewünschten Sprache können nun die Key-Value Paare angepasst werden.

Zertifikat
Jetzt kann Stockit gestartet werden. Der erste Start dauert eine Weile, da sämtliche NPM-Pakete installiert werden müssen.

```
$ sudo docker-compose up -d
```

Nun muss über Certbot ein Zertifikat von Let's Encrypt bezogen werden.
Der "source"-Ordner wurde vorbereitet, so dass die Konfiguration mittels Certbot ein wenig komfortabler ist.

Zuerst muss Certbot gemäss der Anleitung `https://certbot.eff.org/instructions?ws=other&os=ubuntufocal` (Dezember 2021) installiert werden. Falls bei Punkt 6 die Fehlermeldung "ln: failed to create symbolic link '/usr/bin/certbot': File exists" auftaucht, kann diese ignoriert werden. Bei Punkt 7 muss gemäss der Option "No, I need to keep my web server running." vorgegangen werden. Nachdem der Befehl bei Punkt 7 ausgeführt wurde, müssen diverse Eingaben getätigt werden. Wenn nach dem webroot-Pfad gefragt wird, muss "./nginx-webroot" angegeben werden. Nun muss Die Datei *docker-compose.yml* angepasst werden. Beim Service "nginx" bei den "volumes" muss die Zeile "- ./nginx.http.conf:/etc/nginx/nginx.conf" durch die Zeile "- ./nginx.https.conf:/etc/nginx/nginx.conf" ersetzt werden. Ausserdem muss die Zeile "- /etc/letsencrypt:/etc/nginx" bei den "volumes" im Service "nginx" hinzugefügt werden.
Nach diesen Änderungen sollte der Service "nginx" in der Datei *docker-compose.yml* wie folgt aussehen:

```
nginx:
       image: nginx
       container_name: nginx
       volumes:
              - ./nginx.https.conf:/etc/nginx/nginx.conf
              - ./nginx-webroot:/usr/share/nginx/html
              - /etc/letsencrypt:/etc/nginx
       ports:
              - "80:80"
              - "443:443"
       depends_on:
              - nestjs-server
              - vuejs-spa
```

Im nächsten Schritt muss in der Datei *nginx.https.conf* in der Konfigurationsoption "ssl_certificate" der Platzhalter "HOSTNAME" durch den Hostname, für den das Zertifikat ausgestellt wurde ersetzt werden. In der Konfigurationsoption "ssl_certificate_key" muss dasselbe getan werden.

Stockit starten
Damit die Änderungen an der HTTPS-Konfiguration gültig werden, muss Stockit wie folgt neugestartet werden:

```
$ sudo docker-compose down
$ sudo docker-compose up -d
```

Die Installation ist damit abgeschlossen.

B.2 Entwicklungshandbuch

Dieser Abschnitt gibt Hinweise zum Aufsetzen der Entwicklungsumgebungen für das Front- und Backend und zeigt die wichtigsten Funktionen der installierten Tools.

B.2.1 Voraussetzungen

Das Frontend wurde mit Visual Studio Code und das Backend mit Webstorm entwickelt. Es können aber auch andere IDEs verwendet werden. Zusätzlich ist die Installation der folgenden Software nötig:

- Docker, mind. Version 20
- Docker Compose, mind. Version 1
- NPM, mind. Version 6
- Node, mind. Version 14
- Entpackter "source"-Ordner der abgegeben Archivdatei (ZIP-File)

Während der Entwicklung wurde Stockit für den Betrieb mit HTTP auf Port 80 ausgelegt.

B.2.2 Frontend

Für die Entwicklung des Frontendes ist zu empfehlen die Browsererweiterung Vue.js devtools zu installieren. Die folgenden Befehle sind im Ordner "stoc_front-end" auszuführen.

Zuerst müssen die Abhängigkeiten mittels NPM installiert werden.

```
$ npm install
```

Start

Das Frontend benötigt den NGINX Reverse Proxy, das Backend und die Datenbank. Diese können über das zur Verfügung gestellte *docker-compose.yml* gestartet werden:

```
$ docker-compose up -d
```

Anschliessend kann das Frontend gestartet werden. Bei Änderungen am Code wird das Frontend automatisch neu erstellt.

```
$ npm run serve
```

Das Frontend ist unter http://localhost:80 über den NGINX Reverse Proxy erreichbar.

Ausführung der automatisierten Tests

Die Unit Tests können wie folgt ausgeführt werden:

```
$ npm run test:unit
```

Ausführung des Linters

Eslint kann wie folgt ausgeführt werden:

```
$ npm run lint
```

B.2.3 Backend

Die folgenden Befehle sind im Ordner "stoc_back-end" auszuführen.
Zuerst müssen die Abhängigkeiten mittels NPM installiert werden.

```
$ npm install
```

Start

Das Backend benötigt den NGINX Reverse Proxy, das Frontend und die Datenbank. Diese können über das zur Verfügung gestellte *docker-compose.yml* gestartet werden:

```
$ docker-compose up -d
```

Anschliessend kann das Backend gestartet werden. Bei Änderungen am Code wird das Backend automatisch neu gestartet.

```
$ npm run start:dev
```

Während der Entwicklung ist es manchmal vorgekommen, dass NestJS die Applikation nicht richtig gebaut hat, wenn mit dem obigen Befehl gearbeitet wurde. In solchen fällen muss die Applikation einmalig manuell neu gebaut werden.

```
$ npm run build
```

Ausführung der automatisierten Tests

Die Unit Tests können wie folgt ausgeführt werden:

```
$ npm run test
```

Für die Integrationstests wird automatisch mittels Docker eine PostgreSQL Instanz hochgefahren. Diese Tests nehmen deshalb etwas mehr Zeit in Anspruch als die Unit Tests.

```
$ npm run test:e2e
```

Ausführung des Linters

Eslint kann wie folgt ausgeführt werden:

```
$ npm run lint
```

Falls gewünscht ist, dass ein Error-Code zurückgegeben wird, falls gegen Regeln von Eslint verstossen wird, steht ein anderer Befehl zur Verfügung. Dieser Befehl eignet sich vor allem für die Einbindung in die CI/CD Pipeline.

```
$ npm run lint-fail
```

TypeORM Migrationen

TypeORM wurde so konfiguriert, dass Migrationen, die sich im Ordner "src/database/migrations" befinden automatisch beim Start des Backends ausgeführt werden.
Neue Migrationen werden automatisch im bereits erwähnten Ordner abgelegt. Eine neue Migration kann wie folgt erstellt werden:

```
$ npm run typeorm -- migration:generate -n <name>
```

Anhang C

Systemtests

C.1 M4 Systemtest

Dieser Systemtest wurde vor dem Ende des Meilenstein M4 durchgeführt.

C.1.1 Als Organisator möchte ich mich registrieren können

C.1.1.1 Valide Nutzername/Passwort Kombination

Handlung

1. Drücke den “Sign up” Button
2. Gebe Nutzername “joel” ein
3. Gebe Passwort “12345678” ein
4. Gebe Passwort “12345678” in das Bestätigungsfeld ein
5. Drücke den “Sign up” Button

Erwartetes Resultat

- Der Organisator ist in der Datenbank gespeichert
- Das “organizer-jwt” Cookie wird gesetzt
- Der Organisator wird weitergeleitet auf das Dashboard

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.2 Als Organisator möchte ich mich anmelden können

C.1.2.1 Valide Logindaten

Handlung

1. Drücke den “Log in” Button

2. Gebe Nutzername “joel” ein
3. Gebe Passwort “12345678” ein
4. Drücke den “Log in” Button

Erwartetes Resultat

- Das “organizer-jwt” Cookie wird gesetzt
- Der Organisator wird weitergeleitet auf das Dashboard

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.3 Als Organisator möchte ich das STOC-Spiel zu Beginn konfigurieren können

C.1.3.1 Valide Konfigurationseingaben

Handlung

1. Drücke den “Spiel erstellen” Button
2. Gebe Spielname “Spiel 1” ein
3. Gebe Dauer “60” ein
4. Wähle Währung “Schweizer Franken” aus
5. Gebe Anfangsgeld “1000” ein
6. Gebe Anfangsmenge der Konzepte pro Händler “10” ein
7. Gebe Anfangspreis einer Aktie “100” ein
8. Gebe Transaktionsgebühr “5” ein
9. Wähle Marktmodell “Continuous Double Auction” aus
10. Drücke den “Weiter” Button
11. Wähle Price Strategy “Ask Bid” aus
12. Wähle Order Matching Strategy “Price Time Priority” aus
13. Drücke den “Weiter” Button
14. Drücke den “+” Button
15. Gebe Name “Angular” ein
16. Gebe Symbol “ANG” ein
17. Gebe Beschreibung “Ein modulares Frontend Framework” ein
18. Gebe Link “http://www.angular.io” ein

19. Drücke den “Hochladen” Button
20. Wähle ein valides Bild aus
21. Drücke den “Hinzufügen” Button
22. Drücke den “+” Button
23. Gebe Name “React” ein
24. Gebe Symbol “REA” ein
25. Gebe Beschreibung “Ein schlankes Frontend Framework” ein
26. Drücke den “Hochladen” Button
27. Wähle ein valides Bild aus
28. Drücke den “Hinzufügen” Button
29. Drück den “Erstellen” Button

Erwartetes Resultat

- Das Spiel inklusive der Konzepte ist in der Datenbank gespeichert
- Das “organizer-game-jwt” Cookie wird gesetzt
- Der Organisator wird weitergeleitet auf die Lobby

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.4 Als Organisator möchte ich die Lobby sehen

C.1.4.1 Lobby sehen

Handlung

1. Warte bis die Seite geladen ist

Erwartetes Resultat

- Eine Socket.io Verbindung wurde hergestellt
- Die Lobby wird angezeigt

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.5 Als Organisator möchte ich Informationen zum Spiel ansehen können

C.1.5.1 Verbleibende Zeit, Anzahl Trader, Game Code und Sprache ansehen

Handlung

1. Keine

Erwartetes Resultat

- Kurz nachdem das Spiel begonnen wurde, beträgt die verbleibende Zeit 60 Minuten
- Die Anzahl Trader entspricht der Anzahl Trader im Spiel
- Der Game Code entspricht dem bei der Erstellung des Spiels generiertem Game Code
- Englisch und Deutsch stehen als Sprachen zur Auswahl

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.6 Als Organisator möchte ich die aktuellen Marktinformationen erhalten

C.1.6.1 Aktuelle Marktinformationen ansehen

Handlung

1. Drücke den "Markt" Button

Erwartetes Resultat

- Anzeige von Last, Hoch, Tief, Ask, Bid und Volumen für jedes Produktkonzept

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.7 Als Organisator möchte ich alle vergangenen Transaktionen ansehen können

C.1.7.1 Anzeige der Transaktionshistorie

Handlung

1. Drücke den "Historie" Button

Erwartetes Resultat

- Für jede stattgefundene Transaktion wird Symbol, Käufer, Verkäufer, Anzahl, Preis und Datum angezeigt

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.8 Als Organisator möchte ich ein Spiel vorzeitig beenden können

C.1.8.1 Vorzeitige Beendigung eines Spiels

Handlung

1. Drücke den "Stop" Button
2. Drücke den "Ja" Button

Erwartetes Resultat

- Die Socket.io Verbindung zu allen Tradern und dem Organisator wird beendet
- Die Trader bzw. der Organisator werden auf eine Resultateseite weitergeleitet
- Den Tradern wird das Cookie "trader-game-jwt" gelöscht
- Dem Organisator wird das Cookie "organizer-game-jwt" gelöscht
- Die Transaktionen, die Endportfolios, die Rangliste sowie die Metriken für die Produktkonzepte werden in der Datenbank gespeichert

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.9 Als Trader möchte ich einem Spiel beitreten

C.1.9.1 Lobby sehen

Handlung

1. Gebe den vom Organisator erhaltenen Game Code ein
2. Drücke den "Beitreten" Button
3. Gebe Spitzname "profi-trader" ein
4. Gebe E-Mail "trader@mail.com" ein
5. Drücke den "Los geht's" Button

Erwartetes Resultat

- Der Trader wurde in der Datenbank gespeichert
- Das "trader-game-jwt" Cookie wurde gesetzt
- Der Trader wird zur Lobby weitergeleitet

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.10 Als Trader möchte ich die Lobby sehen

C.1.10.1 Lobby sehen

Handlung

1. Warte bis die Seite geladen ist

Erwartetes Resultat

- Eine Socket.io Verbindung wurde hergestellt
- Die Lobby wird angezeigt

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.11 Als Trader möchte ich die Beschreibung der virtuellen Produktideen ansehen

C.1.11.1 Ansehen eines Produktkonzepts

Handlung

1. Drücke den "Markt" Button
2. Wähle das Produktkonzept "ANG" aus der Liste aus
3. Drücke auf die Card rechts oben in der Ecke

Erwartetes Resultat

- Die angezeigte Produktbeschreibung entspricht der Produktbeschreibung, die bei der Erstellung des Spiels erfasst wurde

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.12 Als Trader möchte ich Informationen zum Spiel ansehen können

C.1.12.1 Verbleibende Zeit, Anzahl Trader, Game Code und Sprache ansehen

Handlung

1. Keine

Erwartetes Resultat

- Kurz nachdem das Spiel begonnen wurde, beträgt die verbleibende Zeit 60 Minuten
- Die Anzahl Trader entspricht der Anzahl Trader im Spiel
- Der Game Code entspricht dem bei der Erstellung des Spiels generiertem Game Code
- Englisch und Deutsch stehen als Sprachen zur Auswahl

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.13 Als Trader möchte ich eine Übersicht über mein Portfolio

C.1.13.1 Übersicht über Portfolio

Handlung

1. Drücke den "Portfolio" Button

Erwartetes Resultat

- Für jede Produktidee von der der Trader Anteile besitzt wird Symbol, Anzahl, Investiert, G / V, G / V (%), Wert und ein Kauf- und Verkauf Button angezeigt

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.13.2 Übersicht über die offenen Orders

Handlung

1. Drücke den “Portfolio” Button
2. Drücke den “Orders” Button

Erwartetes Resultat

- Für jede offene Order wird Symbol, Typ, Anzahl, Limit Preis, Total, Total (inkl. Gebühren) und ein Stornieren-Button angezeigt

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.14 Als Trader möchte ich aktuelle Marktinformationen erhalten

C.1.14.1 Aktuelle Marktinformationen ansehen

Handlung

1. Drücke den “Markt” Button

Erwartetes Resultat

- Anzeige von Last, Hoch, Tief, Ask, Bid und Volumen für jedes Produktkonzept

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.15 Als Trader möchte ich eine Buy Limit Order in Auftrag geben

C.1.15.1 Ungenügende Balance

Voraussetzung Der Trader hat eine Balance von 1000.00.

Handlung

1. Drücke den “Markt” Button
2. Drücke den “Kaufen” Button für das Produktkonzept ANG
3. Gebe Preis (Pro Anteil) “100” ein
4. Gebe Anzahl “10” ein
5. Drücke den “Kaufen” Button

Erwartetes Resultat

- Fehlermeldung: “Nicht genügend Geld vorhanden” wird dargestellt und die Order wird nicht an den Markt übermittelt

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.15.2 Genügende Balance

Voraussetzung Der Trader hat eine Balance von 1000.00.

Handlung

1. Drücke den "Portfolio" Button
2. Drücke den "Kaufen" Button für das Produktkonzept ANG
3. Gebe Preis (Pro Anteil) "100" ein
4. Gebe Anzahl "1" ein
5. Drücke den "Kaufen" Button

Erwartetes Resultat

- In der Liste der offenen Order wir die erstellte Order dargestellt
- Es ist nicht möglich eine Sell Limit Order für ANG zu erstellen
- Wenn die Order ausgeführt wurde, wird die Order vom Markt entfernt, der Trader informiert und eine Transaktion erstellt

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.16 Als Trader möchte ich eine Sell Limit Order in Auftrag geben

C.1.16.1 Ungenügende Anzahl Produktkonzepte im Portfolio

Voraussetzung Der Trader hat 10 Produktkonzepte von ANG im Portfolio und keine offene Buy Limit Orders für ANG.

Handlung

1. Drücke den "Portfolio" Button
2. Drücke den "Verkaufen" Button für das Produktkonzept ANG
3. Gebe Preis (Pro Anteil) "100" ein
4. Gebe Anzahl "11" ein
5. Drücke den "Verkaufen" Button

Erwartetes Resultat

- Fehlermeldung: "Nicht genügend Anteile im Besitz" wird dargestellt und die Order nicht an den Markt übermittelt

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.16.2 Ungenügende Anzahl Produktkonzepte im Portfolio

Voraussetzung Der hat 10 Produktkonzepte von ANG im Portfolio und keine offene Buy Limit Orders für ANG.

Handlung

1. Drücke den "Portfolio" Button
2. Drücke den "Verkaufen" Button für das Produktkonzept ANG
3. Gebe Preis (Pro Anteil) "100" ein
4. Gebe Anzahl "10" ein
5. Drücke den "Verkaufen" Button

Erwartetes Resultat

- In der Liste der offenen Order wir die erstellte Order dargestellt
- Es ist nicht möglich eine Buy Limit Order für ANG zu erstellen
- Wenn die Order ausgeführt wurde, wird die Order vom Markt entfernt, der Trader informiert und eine Transaktion erstellt

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.1.17 Als Trader möchte ich eine Order canceln können

C.1.17.1 Order canceln

Voraussetzung Der Trader hat eine offene Order.

Handlung

1. Drücke den "Portfolio" Button
2. Drücke den "Orders" Button
3. Drücke den "Stornieren" Button für die zu stornierende Order

Erwartetes Resultat

- Der Trader wird informiert, dass die Order gecancelt wurde
- Die Order wird vom Markt entfernt

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.2 M5 Systemtest

Dieser Systemtest wurde vor dem Ende des Meilenstein M5 durchgeführt.

C.2.1 Als Organisator möchte ich das STOC-Spiel zu Beginn konfigurieren können

C.2.1.1 Valide Konfigurationseingaben

Handlung

1. Drücke den “Spiel erstellen” Button
2. Gebe Spielname “Spiel 1” ein
3. Gebe Dauer “60” ein
4. Wähle Währung “Schweizer Franken” aus
5. Gebe Startkapital “1000” ein
6. Gebe Anfangsmenge der Konzepte pro Händler “10” ein
7. Gebe Transaktionsgebühr “5” ein
8. Wähle Marktmodell “Continuous Double Auction” aus
9. Drücke den “Weiter” Button
10. Wähle Price Strategy “Ask Bid” aus
11. Wähle Order Matching Strategy “Price Time Priority” aus
12. Drücke den “Weiter” Button
13. Drücke den “+” Button
14. Gebe Name “Angular” ein
15. Gebe Symbol “ANG” ein
16. Gebe Beschreibung “Ein modulares Frontend Framework” ein
17. Gebe Link “http://www.angular.io” ein
18. Drücke den “Hochladen” Button
19. Wähle ein valides Bild aus
20. Drücke den “Hinzufügen” Button
21. Drücke den “+” Button
22. Gebe Name “React” ein
23. Gebe Symbol “REA” ein

24. Wähle Referenzprodukt aus
25. Gebe Referentpreis “10” ein
26. Gebe Beschreibung “Ein schlankes Frontend Framework” ein
27. Drücke den “Hochladen” Button
28. Wähle ein valides Bild aus
29. Drücke den “Hinzufügen” Button
30. Drück den “Erstellen” Button

Erwartetes Resultat

- Das Spiel inklusive der Konzepte ist in der Datenbank gespeichert
- Das “organizer-game-jwt” Cookie wird gesetzt
- Der Organisator wird weitergeleitet auf das Dashboard

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.2.2 Als Organisator möchte ich den Bericht eines STOC-Spiels sehen

C.2.2.1 VWAP und Median sehen

Handlung

1. Drücke auf ein Spiel in der Card Berichte

Erwartetes Resultat

- Der Organisator wird weitergeleitet auf den Bericht

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.2.2.2 Rangliste sehen

Voraussetzung Der Organisator ist in einem Bericht.

Handlung

1. Drücke auf den Tab Rangfolge.

Erwartetes Resultat

- Der Organisator wird sieht das Rangliste.

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.2.2.3 Transaktionen sehen

Voraussetzung Der Organisator ist in einem Bericht.

Handlung

1. Drücke auf den Tab Transaktionen.

Erwartetes Resultat

- Der Organisator wird sieht das Transaktionen.

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

C.2.2.4 Transaktionen exportieren

Voraussetzung Der Organisator ist in einem Bericht.

Handlung

1. Drücke auf den Tab Transaktionen.
2. Drücke auf den Button Export.

Erwartetes Resultat

- Die Transaktionen werden heruntergeladen.

Effektives Resultat Das effektive Resultat entspricht dem erwartetem Resultat.

Anhang D

Usability Test

Im Sprint 5 wurde am Samstag, dem 04.12.2021 ein Usability Test mit den zwei Probanden Joel H. und Thomas H. durchgeführt. Beim Usability Test wurde ausschliesslich die Trader-Perspektive berücksichtigt.

D.1 Zielsetzung

Mittels des Usability Tests soll ermittelt werden, ob Stockit aus der Perspektive eines Traders intuitiv bedienbar und selbsterklärend ist. Durch die gewonnen Erkenntnisse können Änderungen an Stockit vorgenommen werden, um die Usability zu verbessern, oder falls die erforderlichen Änderungen nicht mehr im Rahmen der Studienarbeit vorgenommen werden könnten, würde die Problematik immerhin bekannt sein.

D.2 Software

Der Usability Test wurde mit der zum Zeitpunkt aktuellsten Entwicklungsversion durchgeführt, bei der bereits alle User Stories bis und mit des 5. Sprints implementiert wurden.

D.3 Intro

Es wurde ein STOC-Spiel erstellt, bei dem herausgefunden werden soll, welches Auto von den Probanden am besten bewertet wird.

D.3.1 Ablauf

Den Probanden wurde der Ablauf des Usability Tests erläutert.

D.3.2 Regeln für die Durchführung

Den Probanden wurden die folgenden Regeln mitgeteilt:

- Aus Gründen der Genauigkeit dieses Experiments können Fragen erst nach der Sitzung beantwortet werden

- Das Ziel ist es Stockit zu testen, nicht Sie. Es gibt daher keine falschen Antworten
- Bitte denken Sie laut über alles nach, was Sie tun

D.3.3 Mini-Interview

Die zwei Probanden wurden kurz interviewed. Das Ergebnis des Interviews ist in der folgenden Tabelle festgehalten:

Probant	Alter	Beruf	Börsen-Vorkenntnisse
Joel H.	24	Student Informatik	Keine
Thomas H.	25	Student Informatik	Keine

Tabelle D.1: Ergebnisse der Mini-Interviews

D.4 Aufgaben

Den Probanden wurden mündlich Aufgaben gestellt. Während dem Lösen der Aufgaben wurden die Probanden beobachtet und auftretende Schwierigkeiten notiert.

D.4.1 Spielbeitritt

1. Treten Sie dem Spiel mit dem Room Code 92371932 bei

Ergebnis Joel H.

Keine Auffälligkeiten.

Ergebnis Thomas H.

Keine Auffälligkeiten.

D.4.2 Lobby

1. Wie viele andere Trader sind in der Lobby?
2. Beschreiben Sie kurz welche Produktkonzepte zur Auswahl stehen

Ergebnis Joel H.

Keine Auffälligkeiten.

Ergebnis Thomas H.

- Proband war etwas überrascht, dass Lobby nicht als solche erkennbar war (keine Beschriftung)

D.4.3 Portfolio

1. Wie viel Geld steht Ihnen zur Verfügung?
2. Wie viele Anteile am Konzept PLFM besitzen Sie?

Ergebnis Joel H.

- Proband hat im Markt nach dem Anteil gesucht und dachte, dass gehandelte Volumen entspreche der Anzahl Anteile im Besitz

Ergebnis Thomas H.

Keine Auffälligkeiten.

D.4.4 Markt

1. Zu welchem Preis wurde der SMS zuletzt gehandelt?
2. Steht gegenwärtig ein Anteil von SMS zum Verkauf?

Ergebnis Joel H.

- Proband hat nicht verstanden was Ask und Bid sind

Ergebnis Thomas H.

- Proband hat Kauf Order aufgegeben, um herauszufinden, ob SMS zum Verkauf steht

D.4.5 Handel

1. Verkaufen Sie 5 Anteile von APPARCH für 150
2. Kaufen Sie 7 Anteile von WED1 für 50
3. Welche Orders haben Sie momentan offen?
4. Annullieren Sie die erstellte Verkaufs-Order

Ergebnis Joel H.

- Proband hat offene Orders in der Portfolio-Info-Bar gesucht und war verwirrt, das seine Sell Order dort nicht aufzufinden war

Ergebnis Thomas H.

- Proband war etwas unsicher beim Verkauf der Anteile, ob es sich um den totalen Preis oder den Preis pro Anteil handelt
- Proband hat Orders Tab im Portfolio nicht gefunden

D.4.6 Ranking

1. Wie stehen Sie im Vergleich zu anderen Tradern da?

Ergebnis Joel H.

Keine Auffälligkeiten.

Ergebnis Thomas H.

Keine Auffälligkeiten.

D.5 Schlussbefragung

Am Ende des Usability Tests wurden die Probanden gefragt, was ihnen Schwierigkeiten bereitet hat bzw. ob sie generelle Anmerkungen oder Verbesserungsvorschläge haben.

D.5.1 Joel H.

- Marktbegriffe (bspw. Ask, Bid) in einem Tooltip erklären
- Unklar, was unter investiert zu verstehen ist

D.5.2 Thomas H.

Proband hatte keine Anmerkungen.

D.6 Auswertung

Bei der Portfolio-Info-Bar wird “Offene Orders” durch “Offene Buy Orders” ersetzt. Bei der Portfolio-Info-Bar wird “Investiert” durch “Total investiert” ersetzt.

Anhang E

Testdurchführung

Am 4.12.2021 um 10:00 Uhr wurde ein STOC-Spiel mit 6 Tradern durchgeführt.

E.1 Setup

Das Startgeld wurde auf 1000 CHF festgelegt. Jeder Trader erhielt 10 Aktien, zu einem Anfangspreis von 100 CHF. Die Transaktionsgebühren betrugen 5%. Als Marktmodell wurde der Continuous Double Auction mit Preis-Zeit-Priorität und der Ask/Bid Strategie ausgewählt. Die Spieldauer betrug 40 Minuten. Die Tabelle E.2 beschreibt die gehandelten Konzepte:

Symbol	Name	Beschreibung
EC2000	WV EC2000	Der WV EC2000 wird durch einen Elektromotor mit 204 PS angetrieben. Eine Batterieladung reicht für 500 km. Der VW EC2000 ist zu einem Preis von 50'000 CHF erhältlich.
HCOMFORT	WV HComfort	Der WV HComfort wird durch eine Brennstoffzelle (Wasserstoff) mit 163 PS angetrieben. Eine Tankladung reicht für 666 km. Der WV HComfort ist zu einem Preis von 80'000 CHF erhältlich.
ROADSTAR2	WV RoadStar 2	Der WV RoadStar 2 wird durch einen Hybridmotor (Benzin und Elektro) mit 184 PS angetrieben. Die Reichweite beträgt 795 km. Der WV RoadStar 2 ist zu einem Preis von 40'000 CHF erhältlich.
PUNTO	WV Punto	Der WV Punto wird durch einen Otto-Motor (Benzin) mit 110 PS angetrieben. Eine Batterieladung reicht für 800 km. Der WV Punto ist zu einem Preis von 30'000 CHF erhältlich.
NUCLEON	WV Nucleon	Der WV Nucleon wird durch einen kleinen Kernreaktor mit 400 PS angetrieben. Die Reichweite beträgt 8.000 km. Der WV Nucleon ist zu einem Preis von 100'000 CHF erhältlich.
FORESTMASTER	WV Forest Master	Der WV Forest Master wird durch einen Holzvergaser mit 50 PS angetrieben. Die Reichweite beträgt 50 km. Der WV Forest Master ist zu einem Preis von 20'000 CHF erhältlich.

Tabelle E.1: Gehandelte Konzepte

E.2 Technische Auswertung

Während dem Spiel sind einige Fehler bzw. Unschönheiten aufgetreten.

E.2.1 Probleme

Die Berechnung, ob ein Trader über ausreichend verfügbares Kapital verfügt, um eine Kauf Order auszuführen, wurden vom Backend falsch ausgeführt.

Die Transaktionen in der Historie sollten so angezeigt werden, dass die neueste Transaktion zuoberst auf der ersten Seite in der paginierten Tabelle anzutreffen ist.

Weil im Markt nur gekauft und nicht verkauft werden kann, neigen einige Traders dazu, fast ausschliesslich über das Portfolio zu handeln, da dort gekauft und verkauft werden kann. Da Konzepte, von denen ein Trader keine Anteile besitzt nicht im Portfolio dargestellt werden, haben einige Trader "vergessen", dass diese Konzepte ebenfalls gehandelt werden können.

Im Portfolio ist nur das Symbol der Konzepte ersichtlich. Nähere Informationen zu einem Konzept sind nur über den Markt ersichtlich.

E.2.2 Adressierung der Probleme

Das Backend wird angepasst damit die Berechnung, ob ein Trader über ausreichend Kapitel verfügt um eine Kauf Order auszuführen korrekt ist. Im Frontend wird die Reihenfolge der Transaktionen angepasst. Das Frontend wird angepasst, so dass in der Marktansicht Konzepte verkauft werden können. Die Spalten "Ask" und "Bid" werden aus der Tabelle gelöscht, da diese im Kauf bzw. Verkauf Button ersichtlich sind. Die Detailansicht für ein Konzept wird geöffnet falls auf das Symbol eines Konzeptes im Portfolio gedrückt wird.

E.3 STOC Theorie Auswertung

Nach dem Start des Spieles hat der Handel relativ schnell eingesetzt. Am Ende des Spiels wurden die Konzepte folgendermassen bewertet:

Konzept	VWAP	Median
EC2000	129.51	148.00
NUCLEON	123.38	120.00
FORESTMASTER	109.23	122.50
HCOMFORT	100.08	79.00
ROADSTAR2	74.82	90.00
PUNTO	72.80	88.00

Tabelle E.2: Konzepte mit den dazugehörigen Metriken

E.3.1 Probleme

Zu einigen Zeitpunkten ist das Spiel ins stocken geraten, da einige Trader grosse Teile des Geldes angehäuft hatten und es nicht ausgeben wollten.

Wenn am Ende des Spieles Preise "manipuliert" werden (z.B. einen Anteil für sehr wenig Geld verkaufen um den Portfoliowert eines Konkurrenten zu reduzieren) hat der Markt keine Zeit sich davon zu erholen und das Ranking verliert stark an Aussagekraft.

E.3.2 Adressierung der Probleme

Es ist nicht zu empfehlen, einfach das Startkapital zu erhöhen um Blockierungen zu vermeiden, da ein enges Budget die Trader ermutigt, nicht benötigte Anteile zu verkaufen, was zu einer höheren Dynamik führt. Gute Werte für Startkapital, Anzahl initiale Anteile pro Trader und initialer Preis pro Anteil müssten experimentell ermittelt werden.

Um Preismanipulationen am Ende eines Spieles zu verhindern, sollte der Organisator das Spiel manuell beenden, bevor die vollständige Spielzeit abgelaufen ist.

Anhang F

Lizenzbericht

Dieses Dokument listet die Lizenzen der verwendeten NPM-Pakete auf. Da die verwendeten NPM-Pakete alle auf Open Source Lizenzen basieren [12], kann Stockit frei weiterentwickelt und genutzt werden.

F.1 Backend

Die Tabelle F.1 zeigt die Lizenzen für die NPM-Pakete, welche in der Produktion benötigt werden.

NPM-Paketname	Lizenz
@nestjs/cli	MIT
@nestjs/common	MIT
@nestjs/config	MIT
@nestjs/core	MIT
@nestjs/jwt	MIT
@nestjs/passport	MIT
@nestjs/platform-express	MIT
@nestjs/platform-socket.io	MIT
@nestjs/schedule	MIT
@nestjs/typeorm	MIT
@nestjs/websockets	MIT
bcrypt	MIT
class-transformer	MIT
class-validator	MIT
cookie	MIT
cookie-parser	MIT
fast-glob	MIT
file-type	MIT
mnemonist	MIT
passport	MIT
passport-jwt	MIT
passport-local	MIT
pg	MIT
reflect-metadata	Apache-2.0
rimraf	ISC
rxjs	Apache-2.0
typeorm	MIT
typeorm-naming-strategies	MIT
simple-statistics	ISC
hare-niemeyer	MIT
uuid	MIT

Tabelle F.1: NPM-Pakete für die Produktion und deren Lizenzen

Die Tabelle F.2 zeigt die Lizenzen für die NPM-Pakete welche während der Entwicklung benötigt werden.

NPM-Paketname	Lizenz
@nestjs/schematics	MIT
@nestjs/testing	MIT
@types/bcrypt	MIT
@types/cookie-parser	MIT
@types/express	MIT
@types/jest	MIT
@types/node	MIT
@types/passport-jwt	MIT
@types/passport-local	MIT
@types/pg	MIT
@types/supertest	MIT
@types/cookie	MIT
@types/lodash	MIT
@types/uuid	MIT
@typescript-eslint/eslint-plugin	MIT
@typescript-eslint/parser	BSD-2-Clause
env-cmd	MIT
eslint	MIT
eslint-config-prettier	MIT
eslint-plugin-prettier	MIT
eslint-plugin-sonarjs	LGPL-3.0
jest	MIT
jest-sonar-reporter	MIT
node-mocks-http	MIT
prettier	MIT
supertest	MIT
ts-jest	MIT
ts-loader	MIT
ts-node	MIT
tsconfig-paths	MIT
typescript	Apache-2.0
docker-compose	MIT
set-cookie-parser	MIT
socket.io-client	MIT
lodash	MIT

Tabelle F.2: NPM-Pakete für die Entwicklung und deren Lizenzen

F.2 Frontend

Die Tabelle F.3 zeigt die Lizenzen für die NPM-Pakete, welche in der Produktion und Entwicklung benötigt werden. Im Docker Image für das Frontend werden auch die NPM-Pakete für die Entwicklung installiert, deshalb sind alle NPM-Pakete in einer Tabelle dargestellt.

NPM-Paketname	Lizenz
@vue/cli-service	MIT
@vue/compiler-sfc	MIT
@vue/eslint-config-prettier	MIT
@vue/eslint-config-typescript	MIT
@vue/test-utils	MIT
@vue/vue3-jest	MIT
@vue/cli-plugin-vuex	MIT
@vue/cli-plugin-unit-jest	MIT
@vue/cli-plugin-typescript	MIT
@vue/cli-plugin-router	MIT
@vue/cli-plugin-eslint	MIT
@vue/cli-plugin-e2e-cypress	MIT
@vue/cli-plugin-babel	MIT
@typescript-eslint/parser	BSD-2-Clause
@typescript-eslint/eslint-plugin	MIT
@types/uuid	MIT
eslint	MIT
eslint-plugin-prettier	MIT
eslint-plugin-vue	MIT
jest	MIT
jest-sonar-reporter	MIT
lint-staged	MIT
prettier	MIT
ts-jest	MIT
typescript	Apache-2.0
@vuelidate/core	MIT
@vuelidate/validators	MIT
chart.js	MIT
chartjs-adapter-date-fns	MIT
chartjs-plugin-streaming	MIT
chartjs-plugin-zoom	MIT
core-js	MIT
date-fns	MIT
primeicons	MIT
primevue	MIT
socket.io-client	MIT
uuid	MIT
vue	MIT
vue-i18n	MIT
vue-router	MIT
vuex	MIT

Tabelle F.3: NPM-Pakete für die Produktion und Entwicklung und deren Lizenzen

Anhang G

Schnittstellenbeschreibungen

Dieser Anhang enthält die Schnittstellenbeschreibungen als PDF. Die PDF-Dateien wurden aus der OpenAPI bzw. AsyncAPI Spezifikation generiert.

API Reference

Stockit REST API

API Version: 1.0.0

This API describes the synchronous API for the Stockit application.

INDEX

Security and Authentication

SECURITY SCHEMES

KEY	TYPE	DESCRIPTION
bearerAuth	http, bearer, JWT	

API

1. AUTHENTICATION

1.1 POST /register

Registers a new organisator

REQUEST

REQUEST BODY - application/json

```
{
  username string max:20 chars
  password string min:8 chars
}
```

RESPONSE

STATUS CODE - 200: Successfully registered. Automatically signs in. Returns jwt in a cookie named "jwt"

RESPONSE MODEL - application/json

```
{
  id       integer
  username string
}
```

STATUS CODE - 400: Username already exists or password does not meet requirements

1.2 POST /login

Login as organisator

REQUEST

REQUEST BODY - application/json

```
{
  username string max:20 chars
  password string min:8 chars
}
```

RESPONSE

STATUS CODE - 200: Successfully authenticated. Returns jwt in a cookie named "jwt"

RESPONSE MODEL - application/json

```
{
  id       integer
  username string
}
```

STATUS CODE - 401: Unsuccessfully authenticated

1.3 POST /logout

Logs the organisator out

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200: Successfully logged out

STATUS CODE - 401: Requesting organizer was not logged in

1.4 POST /join

Request to join a game

REQUEST

REQUEST BODY - application/json

```
{
  nickname  string  max:20 chars
  email     string  max:254 chars
  gameCode  string  8 to 8 chars
                    PATTERN: ^[0-9]{8}$
}
```

RESPONSE

STATUS CODE - 200: Successfully joined. Returns jwt in a cookie named "trader-jwt"

STATUS CODE - 400: Can not join the game. Possibly because the provided nickname is already taken

1.5 GET /gameCodeState/{gameCode}

Tells if a game code is valid

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*gameCode	string 8 to 8 chars	

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

```
{
  isValid  boolean
}
```

1.6 GET /gameState/{gameId}

Returns the state of a game

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*gameId	number	

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

```
{
  gameState  enum  ALLOWED:CREATED, LOADED, STARTED, FINISHED
}
```

STATUS CODE - 400: The provided gameId is invalid

1.7 GET /loginState

Checks whether the client is logged in

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

```
{
  loggedIn  boolean
}
```

2. GAME-CREATION

2.1 GET /game

Returns all games for the authenticated organizer

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*gameState	enum **ALLOWED:** CREATED, LOADED, STARTED, FINISHED	All games with this gameState are returned.

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

```
{
  games [{
  Array of object:
    id                                integer
    roomCode                          integer
    name                              string
    durationInMinutes                 integer
    initialCash                       integer
    initialQuantityOfSharesPerTrader  integer
    transactionFee                    integer
    currencyCode                      string
    marketModel {
      name     string
      config [{
      Array of object:
        configName   string
        configValue  string
      }]
    }
  }]
}
```

STATUS CODE - 400: Invalid request

STATUS CODE - 401: Unauthorized access

2.2 POST /game

Creates a new game with the provided details

REQUEST

REQUEST BODY - application/json

```
{
```

```
  name                              string   max:20 chars
  durationInMinutes                 integer  between 10 and  300
  initialCash                       integer  between 0 and  2000000000
  initialQuantityOfSharesPerTrader  integer  between 0 and  2000000000
  transactionFee                    integer  between 0 and  100
  currencyCode                      string   3 to 3 chars
  marketModel {
    name     string
    config [{
    Array of object:
      configName   string
      configValue  string
    }]
  }
  concepts {
    concepts [{
    Array of object:
      name*               string   max:200 chars
      symbol*             string   3 to 12 chars
      description*        string
      image*              string
      isReferenceConcept* boolean
      referencePrice      number   between 0 and  2000000000
      link                string
      contact             string   max:254 chars
    }]
  }
}
```

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

```
{
  id                                integer
  roomCode                          integer
  name                              string
  durationInMinutes                 integer
  initialCash                       integer
  initialQuantityOfSharesPerTrader  integer
  transactionFee                    integer
  currencyCode                      string
  marketModel {
    name     string
    config [{
    Array of object:
      configName   string
      configValue  string
    }]
  }
}
```

STATUS CODE - 400: Invalid request

STATUS CODE - 401: Unauthorized access

2.3 PATCH /game

Creates a new game with the provided details

REQUEST

REQUEST BODY - application/json

```
{
  gameId     number
  gameState  enum    ALLOWED:LOADED
}
```

RESPONSE

STATUS CODE - 200: Successfully loaded game in memory.

STATUS CODE - 400: Invalid request

STATUS CODE - 401: Unauthorized access

2.4 GET /marketModels

Returns all available market models with their configuration options

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

```
[{
Array of object:
  name      string
  config [{
  Array of object:
    configName    string
    configValues  [string]
  }]
}]
```

3. GAME-REPORT

3.1 GET /reports

Returns a list of game reports for a specific organisatorId

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

```
{
  reports [{
  Array of object:
    gameId             integer
    name               string
    startDate          string
    numberOfConcepts   integer  returns the number of non-reference concepts
    durationInMinutes  integer
    numberOfTraders    integer
  }]
}
```

STATUS CODE - 401: Unauthorized access

3.2 GET /conceptReports

Returns all conceptReports for a given gameId

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*gameId	integer >=0	The ID of the game

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

```
{
  conceptReports [{
  Array of object:
    conceptId  integer
    vwap       integer
    median     integer
    shareholders [{
```

Array of object:

```
      traderId  integer
      amount    integer
    }]
  }]
}
```

STATUS CODE - 400: Invalid gameId or game is not finished

STATUS CODE - 401: Unauthorized access

3.3 GET /transactions

Returns all transactions for a given gameId

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*gameId	integer >=0	The ID of the game

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

```
{
  transactions [{
  Array of object:
    id              integer
    conceptId       integer
    buyerTraderId   integer
    sellerTraderId  integer
    price           integer
    amount          integer
    timestamp       string
  }]
}
```

STATUS CODE - 400: Invalid gameId or game is not finished

STATUS CODE - 401: Unauthorized access

3.4 GET /traders

Returns all traders for a given gameId

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*gameId	integer >=0	The ID of the game

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

```
{
  traders [{
  Array of object:
    id     integer
    name   string
    email  string
  }]
}
```

STATUS CODE - 400: Invalid gameId or game is not finished

STATUS CODE - 401: Unauthorized access

3.5 GET /ranking

Returns the ranking for a given gameId

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*gameId	integer >=0	The ID of the game

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

```
{
  rankingEntries [{
  Array of object:
    traderId       integer
    rank           integer
    portfolioValue integer
  }]
}
```

STATUS CODE - 400: Invalid gameId or game is not finished

STATUS CODE - 401: Unauthorized access

3.6 GET /concepts

Returns the concepts for a given gameId

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*gameId	integer >=0	The ID of the game

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

```
[{
Array of object:
  id*                  integer
  name*                string
  symbol*              string
  description*         string
  image*               string
  isReferenceconcept*  boolean
  referencePrice       number
  link                 string
  contact              string
}]
```

STATUS CODE - 400: Invalid gameId

STATUS CODE - 401: Unauthorized access

Stockit Socket.io API 1.0.0 documentation

This API is responsible for the asynchronous communication with the client during STOC-games. The x-return-value correspond to acknowledgments in Socket.io.

Table of Contents

Servers

`` Server

- URL: `wss://sinv-56016.rj.ost.ch/`
- Protocol: `Socket.io`

Path: api/socket.io

Security

Security Requirement 1

- Type: `HTTP`
  - Scheme: bearer
  - Bearer format: JWT

  Trader (trader-game-jwt): {id: number; gameId: number; gameCode: number; nickname: string; email: string; iat: Date; exp: Date;}
  Organizer (organizer-game-jwt): {id: number; gameId: number; gameCode: number; username: string; iat: Date; exp: Date;}

Operations

PUB `/` Operation

Accepts **one of** the following messages:

Message `start-game`

Can only be emitted by an organizer

Message `get-traders`

Message `get-concepts`

Message `get-gameinformation`

Message `get-portfolio`

Can only be emitted by a trader

Message `get-marketinformation`

Message `get-transactions`

If all is set, all transactions are returned. Otherwise only the transactions where the requesting trader is involved are returned.

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
all	boolean	-	-	-	-

> Examples of payload *(generated)*

```
{
  "all": true
}
```

Message `get-open-orders`

Can only be emitted by a trader

Message `get-ranking`

Message `place-order`

Can only be emitted by a trader

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
clientOrderId	string	-	-	-	-
type	string	-	allowed (`"BUY"`, `"SELL"`)	-	**additional properties are allowed**
limitPrice	integer	-	-	-	-
quantity	integer	-	-	-	-
conceptId	integer	-	-	-	-

> Examples of payload *(generated)*

```
{
  "clientOrderId": "string",
  "type": "BUY",
  "limitPrice": 0,
  "quantity": 0,
  "conceptId": 0
}
```

Message `cancel-order`

Can only be emitted by a trader

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
orderToCancel	integer	orderId of the order to cancel	-	-	-

> Examples of payload *(generated)*

```
{
  "orderToCancel": 0
}
```

Message `stop-game`

Can only be emitted by an organizer

SUB `/` Operation

Accepts **one of** the following messages:

Message `traders-updated`

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
traders	array	-	-	-	-
traders.traderId	integer	-	-	-	-
traders.nickname	string	-	-	-	-
traders.email	string	-	-	-	-
traders.connected	boolean	-	-	-	-

> Examples of payload *(generated)*

```
{
  "traders": [
    {
      "traderId": 0,
      "nickname": "string",
      "email": "string",
      "connected": true
    }
  ]
}
```

Message `game-started`

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	any	-	-	-	**additional properties are allowed**

> Examples of payload *(generated)*

```
""
```

Message `portfolio-updated`

Emitted when the portfolio of a trader changed

Payload

Name	Type	Description	Value	Constraints	Notes

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
balance	integer	-	-	-	-
portfolioEntries	array	-	-	-	-
portfolioEntries.conceptId	integer	-	-	-	-
portfolioEntries.quanity	integer	-	-	-	-
portfolioEntries.invested	integer	-	-	-	-

> Examples of payload *(generated)*

```
{
  "balance": 0,
  "portfolioEntries": [
    {
      "conceptId": 0,
      "quanity": 0,
      "invested": 0
    }
  ]
}
```

Message `marketinformation-updated`

Emitted to all traders and the organizer when the marketinformation changed

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
conceptId	integer	-	-	-	-
ask	integer	-	-	-	-
askSize	integer	-	-	-	-
bid	integer	-	-	-	-
bidSize	integer	-	-	-	-
last	integer	-	-	-	-
high	integer	-	-	-	-
low	integer	-	-	-	-
volume	integer	-	-	-	-

> Examples of payload *(generated)*

```
{
  "conceptId": 0,
  "ask": 0,
  "askSize": 0,
  "bid": 0,
  "bidSize": 0,
  "last": 0,
  "high": 0,
  "low": 0,
  "volume": 0
}
```

Message `transaction-occurred`

Emitted to the two traders involved and the organizer

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
quantity	integer	-	-	-	-
price	integer	-	-	-	-
timestamp	string	-	-	format (`date-time`)	-
conceptId	integer	-	-	-	-
buyerTraderId	integer	-	-	-	-
sellerTraderId	integer	-	-	-	-

> Examples of payload *(generated)*

```
{
  "quantity": 0,
  "price": 0,
  "timestamp": "2019-08-24T14:15:22Z",
  "conceptId": 0,
  "buyerTraderId": 0,
  "sellerTraderId": 0
}
```

Message `ranking-updated`

Emitted to all traders and the organizer when the ranking changed

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**

Name	Type	Description	Value	Constraints	Notes
rankingEntries	array	-	-	-	-
rankingEntries.rank	integer	-	-	-	-
rankingEntries.traderId	integer	-	-	-	-
rankingEntries.portfolioValue	integer	-	-	-	-

Examples of payload *(generated)*

```
{
  "rankingEntries": [
    {
      "rank": 0,
      "traderId": 0,
      "portfolioValue": 0
    }
  ]
}
```

Message `order-received`

Emitted to the trader that placed the order when the server received the order

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	additional properties are allowed
clientOrderId	string	-	-	-	-
orderId	number	-	-	-	-

Examples of payload *(generated)*

```
{
  "clientOrderId": "string",
  "orderId": 0
}
```

Message `order-validated`

Emitted to the trader that placed the order when the server validated the order

Payload

Name	Type	Description	Value	Constraints	Notes

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
orderId	integer	-	-	-	-
state	string	-	allowed (`"ACCEPTED"`, `"REJECTED"`)	-	**additional properties are allowed**
rejectionReason	string	This property is optional	allowed (`"MARKET"`, `"INSUFFICIENT_BALANCE"`, `"INSUFFICIENT_QUANTITY"`, `"POTENTIAL_SELF_MATCH"`)	-	**additional properties are allowed**

Examples of payload *(generated)*

```
{
  "orderId": 0,
  "state": "ACCEPTED",
  "rejectionReason": "MARKET"
}
```

Message `order-settled`

Emitted to the trader that placed the order when the order was entirely settled

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
orderId	integer	-	-	-	-

Examples of payload *(generated)*

```
{
  "orderId": 0
}
```

Message `order-partially-settled`

Emitted to the trader that placed the order when the order was partially settled

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
orderId	integer	-	-	-	-
newQuantity	integer	-	-	-	-

Examples of payload *(generated)*

```
{
  "orderId": 0,
  "newQuantity": 0
}
```

Message `game-stopped-initialized`

Game does not accept requests anymore

Message `game-stopped`

Game has been saved in the database

Anhang H

Wireframes

Stockit

Join a Game! What is Stockit? About us

SIGN UP LOG IN

Get into a Game!

Game Code JOIN

What is Stockit?

About us

Nickname:

nickname

E-Mail:

E-Mail

LETS GO!

Room: 789563

Stockit

J Jonas Knupp

Overview

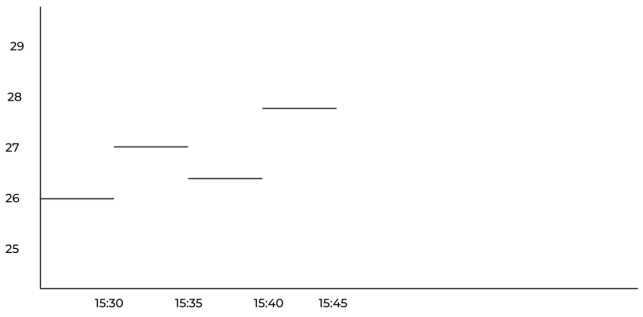
History

Ranking

No. of player
10

time remaining
20:00

STOP



AirStik

Description: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent congue ipsum et tellus

Security	Change	Last	High	Low	Ask	Bid
AirStik	+1					
GearHead	-5					

Stockit

J **Jonas Knupp**

Overview

History

Ranking

No. of player
10

time remaining
20:00

STOP

Security	Buyer	Seller	Amount	Price	Date
AirStik					
GearHead					

Stockit

J **Jonas Knupp**

Overview

History

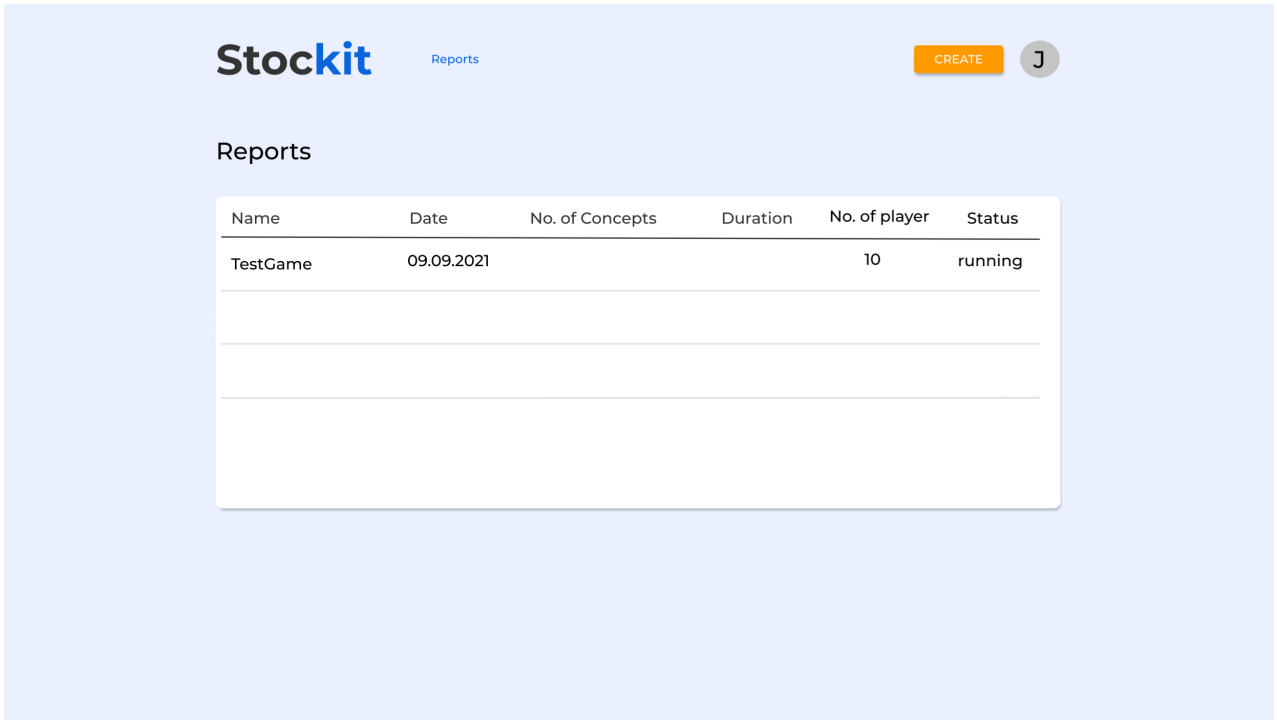
Ranking

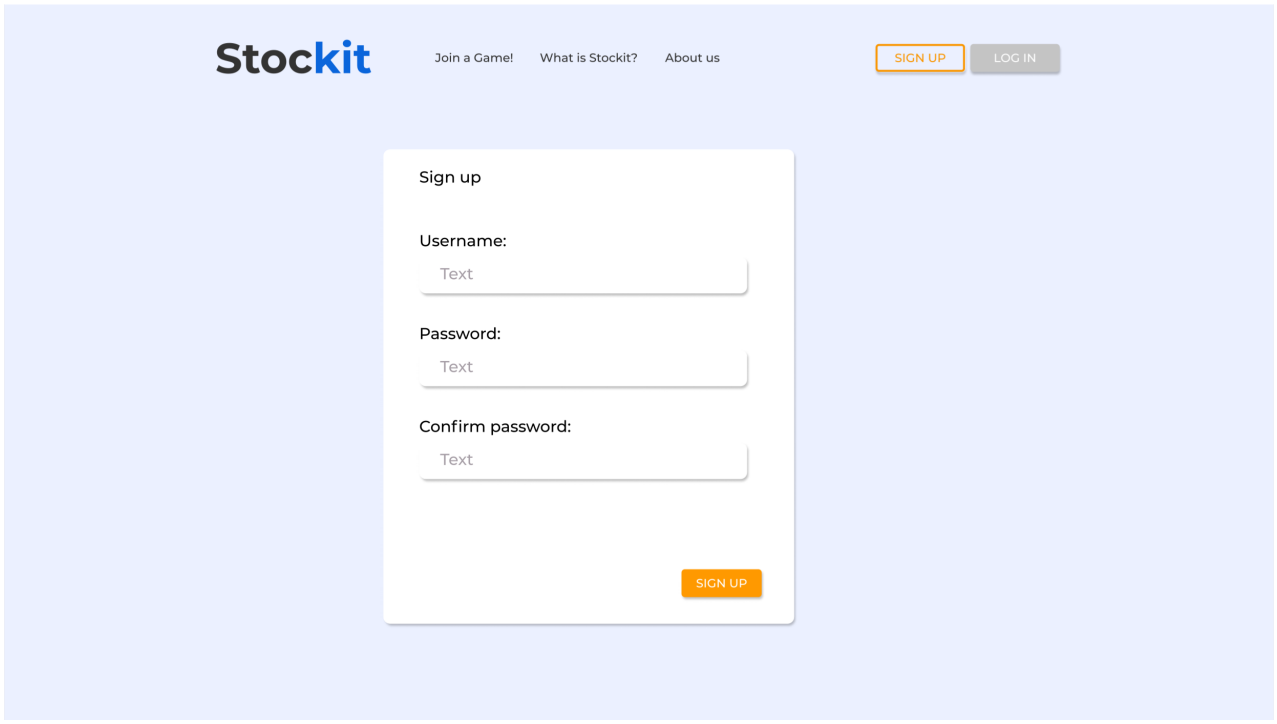
No. of player
10

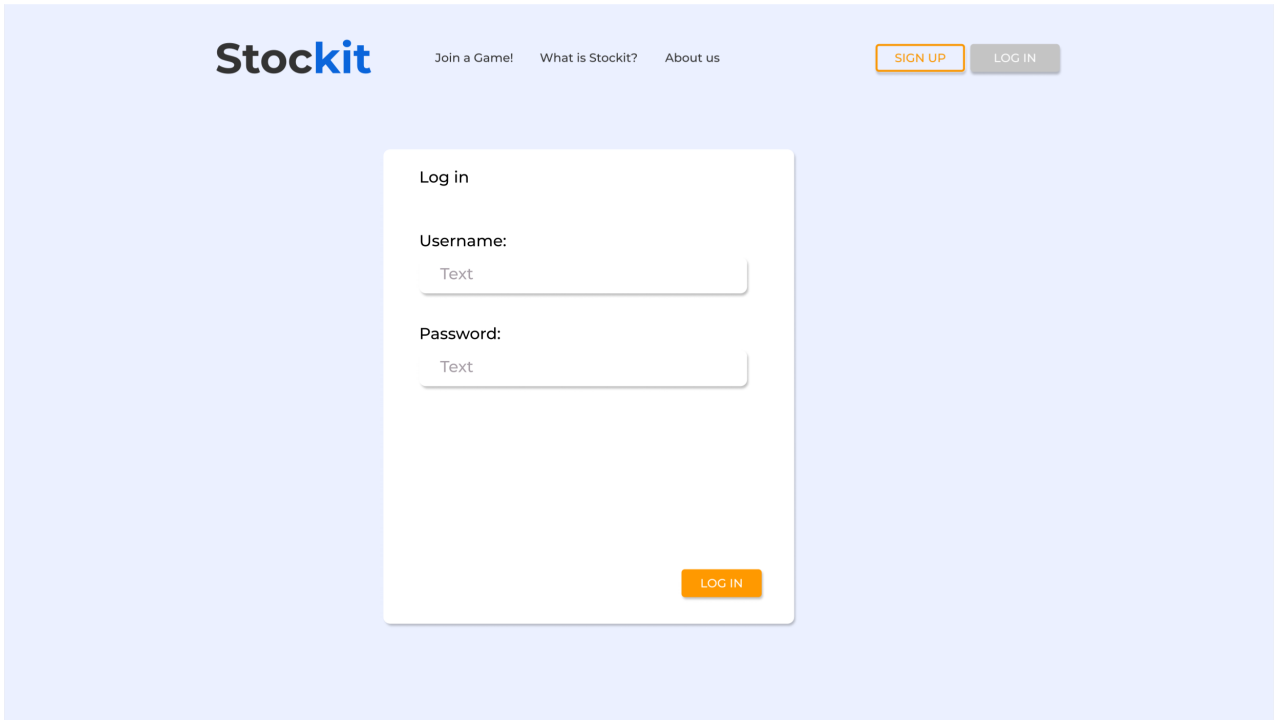
time remaining
20:00

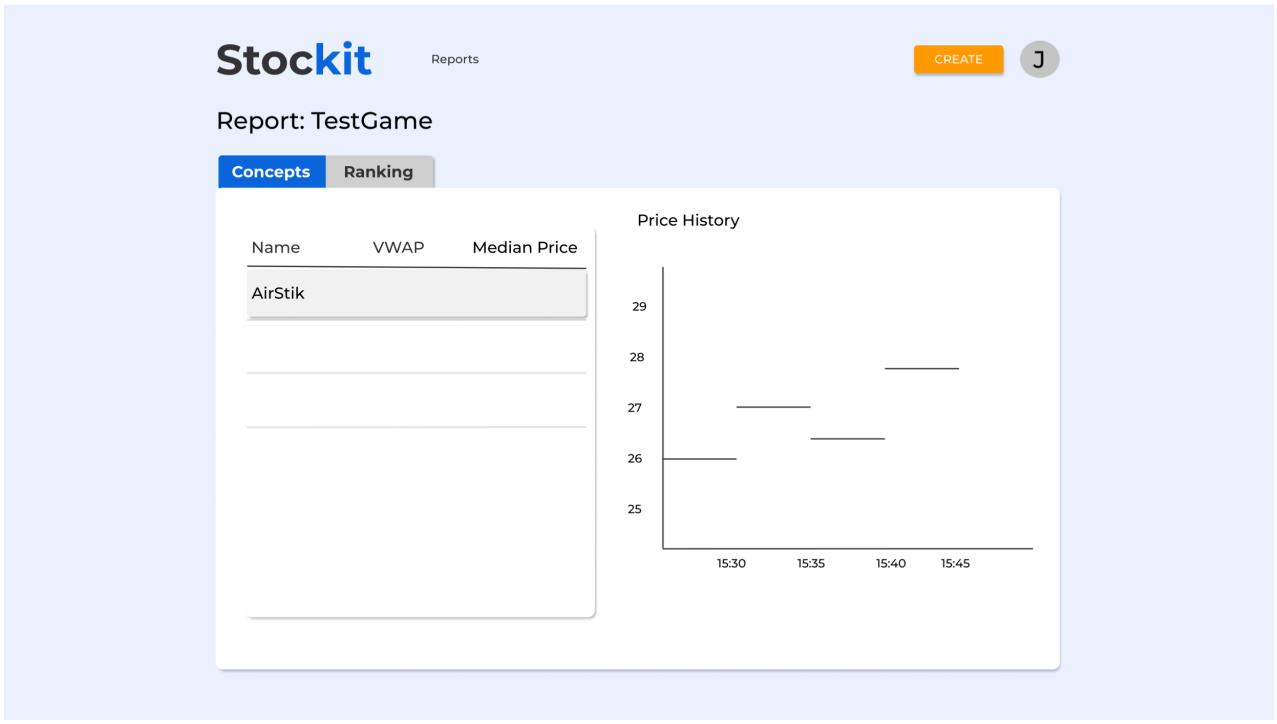
STOP

Rank	Name	E-Mail	Portfolio value
1	Thomas		1000.-
2	Abi		999.-


Stockit
Reports
CREATE
J
Reports
Name
Date
No. of Concepts
Duration
No. of player
Status
TestGame
09.09.2021
10
running


Stockit
Join a Game!
What is Stockit?
About us
SIGN UP
LOG IN
Sign up
Username:
Text
Password:
Text
Confirm password:
Text
SIGN UP


Stockit
Join a Game!
What is Stockit?
About us
SIGN UP
LOG IN
Log in
Username:
Text
Password:
Text
LOG IN

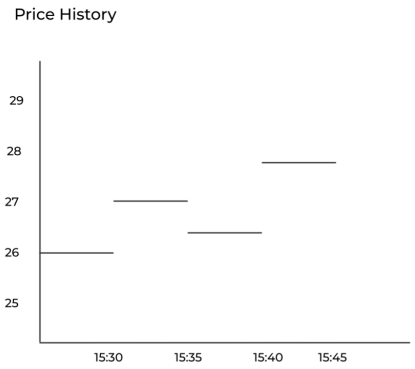

Stockit
Reports
CREATE
J
Report: TestGame
Concepts
Ranking
Name
VWAP
Median Price
AirStik
Price History
29
28
27
26
25
15:30
15:35
15:40
15:45

Stockit Reports CREATE J

Report: TestGame

Concepts Ranking

Name	VWAP	Median Price
AirStik		



Stockit Reports CREATE J

Report: TestGame

Concepts Ranking

Rank	Name	E-Mail	Portfolio value

Stockit

J

X

Set-up — market model options — product concept

Game name: Text

Duration: Text

Initial cash: Text

Initial quantity of shares per trader: Text

Initial price of a share: Text

Transaction fee: Text

Currency EUR V

Market model: Double auction V

PREVIOUS NEXT

Stockit

J

X

Set-up — market model options — product concept

Price strategie: Arithmetic mean V

Order matching strategie: Price-Time-Priority V

PREVIOUS NEXT

Stockit

J

X

Set-up — market model options — product concept

AirStik

PREVIOUS CREATE

Stockit

J

Room Code: 132456

Abi

Thomas

Joel

START

Stockit

Room Code: 132456

Abi

Thomas

Joel

Waiting ...

Stockit

Market

Portfolio

History

Ranking

time remaining
20:00

Room: 789563

8'000.-
Available

2'000.-
Total Allocated

-9.-
Profit:

9'991.-
Total Equity

Portfolio | Orders

Symbol	Type	Amount	Limit price	Total	Total (inkl Fees)	
AirStik	Limit buy				1000	✕
GearHead	Limit sell				500	✕

Stockit

- Market
- Portfolio
- History
- Ranking

time remaining
20:00

Room: 789563

Symbol	Buyer	Seller	Amount	Price	Date
AirStik					
GearHead					

Stockit

- Market
- Portfolio
- History
- Ranking

time remaining
20:00

Room: 789563

Rank	Name	Portfolio value
1	Anonymous	1000.-
2	Abi	999.-

Stockit

- Market
- Portfolio
- History
- Ranking

time remaining
20:00

Room: 789563

8'000.- Available + 1'000.- Total Invested + 1'000.- Open Orders (inkl. Fees) + -9.- Profit: = 9'991.- Portfolio value

Portfolio | Orders

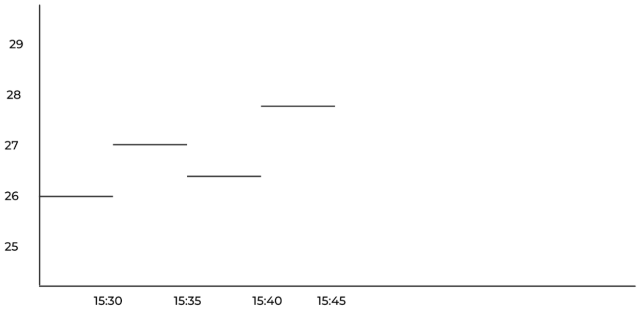
Symbol	Volume	Invested	P/L (chf)	P/L (%)	Value	Sell	Buy
AirStik	45	500.-	1.-	0.2% 2%	501.-	28.-	27.50
GearHead	10	500.-	-10.-		490.-	28.-	27.50

Stockit

- Market
- Portfolio
- History
- Ranking

time remaining
20:00

Room: 789563



AirStik

Description: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent congue ipsum et tellus

Symbol	Change	Last	High	Low	Ask	Bid	Volume	
AirStik	+1							BUY
GearHead	-5							BUY