LASTENHEFT / PFLICHTENHEFT

Abschlussprojekt an der HBFSWI Saarbrücken

Thema: Orderbuch-Simulation
Name: Moritz Kolb
Klasse: WI24ZI
Abschlussjahr: 2026

Inhaltsverzeichnis

1. Einführung

In jedem Finanzmarkt auf der Welt wird ein sogenanntes Orderbuch verwendet. Dieses Orderbuch listet die Kauf- und Verkaufsangebote der Marktteilnehmer auf und verrechnet diese miteinander. Zu jedem Zeitpunkt befindet sich das Orderbuch auf dem aktuellen Stand der Interessen von Käufern und Verkäufern.

Jede Order besteht aus einer bestimmten Menge des gehandelten Wertes sowie aus einem festen Preis. Bei Verkäufern beschreibt dieser Preis den minimalen Preis, zu dem sie bereit sind zu verkaufen, während es sich bei Käufern um den maximalen Preis handelt, den sie bereit sind zu zahlen. Die im Orderbuch enthaltenen Ein- und Verkaufsanweisungen werden systematisch miteinander verrechnet. Die besten Angebote bestimmen dabei den Preis, zu dem das jeweilige Gut aktuell gehandelt wird.

Im System gibt es zwei Möglichkeiten, eine Order im Markt zu platzieren:

Art der Order:	**Limit-Order**	**Market-Order**
Beschreibung:	Wird mit einem festen Preis ins Orderbuch eingetragen und wartet dort auf ein entsprechendes Gegenangebot. Die Order wird erst ausgeführt, sobald ein passender Handelspartner die Gegenorder platziert, sodass der festgelegte Preis erreicht wird.	Wird nicht ins Orderbuch eingetragen, sondern sofort zum aktuellen Marktpreis ausgeführt.
Vorteil:	Preisgenauigkeit – die Order wird nur zu dem gewünschten Preis ausgeführt.	Direkte Ausführung – die Order wird sofort abgewickelt.
Nachteil:	Die Order kann möglicherweise nicht sofort ausgeführt werden, sondern muss warten, bis ein passendes Gegenangebot im Markt vorliegt.	Der Preis, zu dem die Order ausgeführt wird, ist nicht vorher festgelegt, sondern richtet sich nach dem aktuellen Markt. Bei geringer Marktliquidität (wenige passende Angebote oder Gegenangebote) kann der ausgeführte Preis vom gewünschten Preis abweichen.

Eine Software, welche diesen Prozess simuliert, ist besonders interessant für Lehrtätigkeiten sowie für Forschungs- und Testzwecke. Mit einer solchen Simulation lassen sich die zugrundeliegenden Prozesse der Preisentstehung besser verstehen, analysieren und erklären. Da es sich um ein internes Projekt handelt, wird eine fiktive, aber realitätsnahe Projektsituation angenommen.

2. Ist-Situation

Eine Bank, die die Geldanlagen ihrer Kunden möglichst effektiv anlegen möchte, nutzt eine Software, welche verschiedene Finanzmärkte auf Investitionsmöglichkeiten untersucht. Dabei investiert die Bank bevorzugt in Märkte, deren aktueller Preis über dem Durchschnittspreis der letzten 50 Tage liegt.

Um sicherzustellen, dass diese Software auch im späteren Live-Betrieb zuverlässig und fehlerfrei funktioniert, müssen umfangreiche Unit-Tests durchgeführt werden. Diese Tests benötigen eine große Menge an Marktdaten, um möglichst viele unterschiedliche Testszenarien abdecken zu können.

Reale digitale Finanzmarktdaten sind jedoch in ihrer Anzahl begrenzt und häufig kostenpflichtig. Außerdem lassen sie sich nur eingeschränkt anpassen. Aus diesem Grund besteht die Notwendigkeit, künstliche Preisdaten zu erzeugen. Eine Software, die einen Markt mithilfe eines Orderbuchs simuliert, stellt hierfür eine geeignete Lösung dar.

3. Soll-Situation

Aus Sicht des Auftraggebers soll eine Software entstehen, die es ermöglicht, einen Finanzmarkt realitätsnah zu simulieren. Dabei sollen die grundlegenden Mechanismen eines Orderbuchs abgebildet werden, um daraus Preisverläufe zu generieren, die für Tests, Analysen und Lehrzwecke genutzt werden können.

3.1. Soll-Zustand

Nach Abschluss des Projekts steht eine lauffähige Anwendung zur Verfügung, die einen Markt mithilfe eines Orderbuchs simuliert. Der Benutzer kann verschiedene Parameter des Marktes konfigurieren und den daraus entstehenden Preisverlauf beobachten.

Die Software erzeugt dabei Preisdaten, die gespeichert und später weiterverwendet werden können. Der entstehende Mehrwert liegt darin, dass Test- und Schulungsdaten unabhängig von realen Finanzmärkten generiert werden können und somit keine externen Marktdaten benötigt werden.

3.2 Funktionale Anforderungen

Die Software soll es ermöglichen, einen Finanzmarkt mithilfe eines Orderbuchs realitätsnah zu simulieren. Der Benutzer kann verschiedene Parameter des Marktes konfigurieren und den daraus entstehenden Preisverlauf beobachten. Dabei sollen sowohl Limit-Orders als auch Market-Orders unterstützt werden. Alle Transaktionen, Orders und der aktuelle Simulationszustand werden persistent in einer Datenbank gespeichert, sodass ein Neustart oder Absturz der Anwendung den letzten Zustand korrekt wiederherstellt. Das Orderbuch sowie der daraus entstehende Preisverlauf sollen visuell in Echtzeit dargestellt werden, wobei der Benutzer die Darstellung interaktiv steuern kann. Die grafische Benutzeroberfläche muss benutzerfreundlich gestaltet sein und fehlerhafte Eingaben korrekt abfangen.

Die funktionalen Anforderungen im Detail:

1. **Marktparameter konfigurieren (14 Std.)**
   Der Benutzer kann die Anzahl der Käufer und Verkäufer, deren Risikoprofil für den Handel einstellen. Alle Parameter müssen korrekt im System übernommen werden.

2. **Simulation starten (10 Std.)**
   Der Benutzer kann die Handelssimulation starten. Das System initialisiert die Simulation aufgrund der angegebenen Parameter, das Orderbuch und die Bots.

3. **Limit-Order platzieren (12 Std.)**
   Benutzer oder Bots können Kauf- oder Verkaufsaufträge mit festem Preis platzieren. Orders werden vom System validiert, ins Orderbuch eingetragen und bei passendem Gegenangebot verrechnet.

4. **Market-Order ausführen (10 Std.)**
   Benutzer oder Bots können Kauf- oder Verkaufsaufträge zum aktuellen Marktpreis ausführen. Orders werden nur ausgeführt, wenn ein passendes Gegenangebot vorhanden ist, andernfalls gibt das System eine Fehlermeldung aus und die Order wird nicht ausgeführt.

5. **Orderbuch visualisieren (16 Std.)**
   Das System zeigt den aktuellen Zustand des Orderbuchs in Echtzeit an, inklusive Mengen und Preise.

6. **Preisgraph visualisieren (14 Std.)**
   Der aus dem Orderbuch generierte Preisverlauf wird als Graph dargestellt. X- und Y-Achse können verschoben und per Mausrad skaliert werden.

7. **Persistente Speicherung (14 Std.)**
   Alle Transaktionen, Orders und der Simulationszustand werden in einer Datenbank gespeichert. Bei Neustart oder Absturz wird der letzte Zustand korrekt wiederhergestellt.

8. **GUI-Bedienbarkeit (20 Std.)**
   Alle Funktionen müssen über die grafische Benutzeroberfläche benutzerfreundlich verfügbar sein. Fehlerhafte Eingaben werden korrekt abgefangen.

3.3 Nicht-funktionale Anforderungen

1) Persistente Speicherung von Transaktionen (16.Std.)
Das System muss alle im Rahmen der Simulation ausgeführten Transaktionen in einer zentralen Datenbank dauerhaft speichern und sicherstellen, dass diese bei Bedarf abgerufen oder analysiert werden können.

2) Stabile und performante Software (12.Std.)
Die Software muss auch bei einer hohen Anzahl von Marktteilnehmern (mindestens 100 gleichzeitig aktive Teilnehmer) flüssig arbeiten, ohne dass es zu merklichen Verzögerungen (>200 ms pro Transaktion) kommt oder Abstürze auftreten.
Zeitabschätzung: 12 Std.

3) Intuitive und leicht verständliche Benutzeroberfläche (6.Std.)
Die Benutzeroberfläche muss so gestaltet sein, dass neue Benutzer die grundlegenden Funktionen innerhalb von 5 Minuten ohne zusätzliche Anleitung bedienen können.

3.4 Schnittstellen

Zur Speicherung und zum Laden der Transaktionen sowie relevanter Simulationsdaten wird eine PostgreSQL-Datenbank eingesetzt. Diese dient als zentrale Schnittstelle zwischen der Anwendung und der persistenten Datenhaltung.

3.5 Risiken

Risiko:
Ein simuliertes Orderbuch allein reicht möglicherweise nicht aus, um alle für reale Finanzmärkte typischen Muster und Preisbewegungen abzubilden, da nicht alle externen Einflussfaktoren berücksichtigt werden können.

Gegenmaßnahme:
Die Software bietet die Möglichkeit, historische oder aktuelle Marktdaten eines realen Orderbuchs über eine API-Schnittstelle einzubinden, um die Simulation zu validieren und zu ergänzen. Somit werden alle Einflussfaktoren berücksichtigt und die Software spiegelt die Dynamik eines echten Marktes wieder.

4. Abnahmekriterien

1) Marktparameter einstellen:
Alle vom Benutzer eingestellten Parameter (Anzahl Käufer/Verkäufer, Risikoprofil, Mindest- und Höchstpreise) müssen korrekt übernommen und im System angewendet werden.
Überprüfung:
Ändern der Parameter und Kontrolle der Wirkung im Orderbuch und Simulationsergebnis.

2) Simulation starten:
Nach Klick auf „Simulation starten“ initialisiert das System alle Bots, legt das Orderbuch korrekt an und zeigt den Startzustand der Simulation in der GUI an.
Überprüfung:
Start der Simulation ohne Fehlermeldungen; alle Bots und das Orderbuch werden korrekt angezeigt.

3) **Limit-Orders:**
Limit-Orders werden validiert, korrekt ins Orderbuch eingetragen und bei passendem Gegenangebot verrechnet. Orders ohne passendes Gegenangebot verbleiben im Orderbuch.
Überprüfung:
- Test mit passenden Orders → korrekte Verrechnung, Mengen und Preise stimmen.
- Test ohne passende Orders → Order bleibt im Orderbuch, unverändert.

4) Market-Orders:
- Market-Orders werden nur ausgeführt, wenn ein passendes Gegenangebot vorhanden ist.
- Bei fehlendem Gegenangebot gibt das System eine Fehlermeldung aus und das Orderbuch bleibt unverändert.
Überprüfung:
- Mit Gegenangebot → Order wird ausgeführt, Transaktionen in DB gespeichert.
- Ohne Gegenangebot → Fehlermeldung wird angezeigt, Orderbuch unverändert.

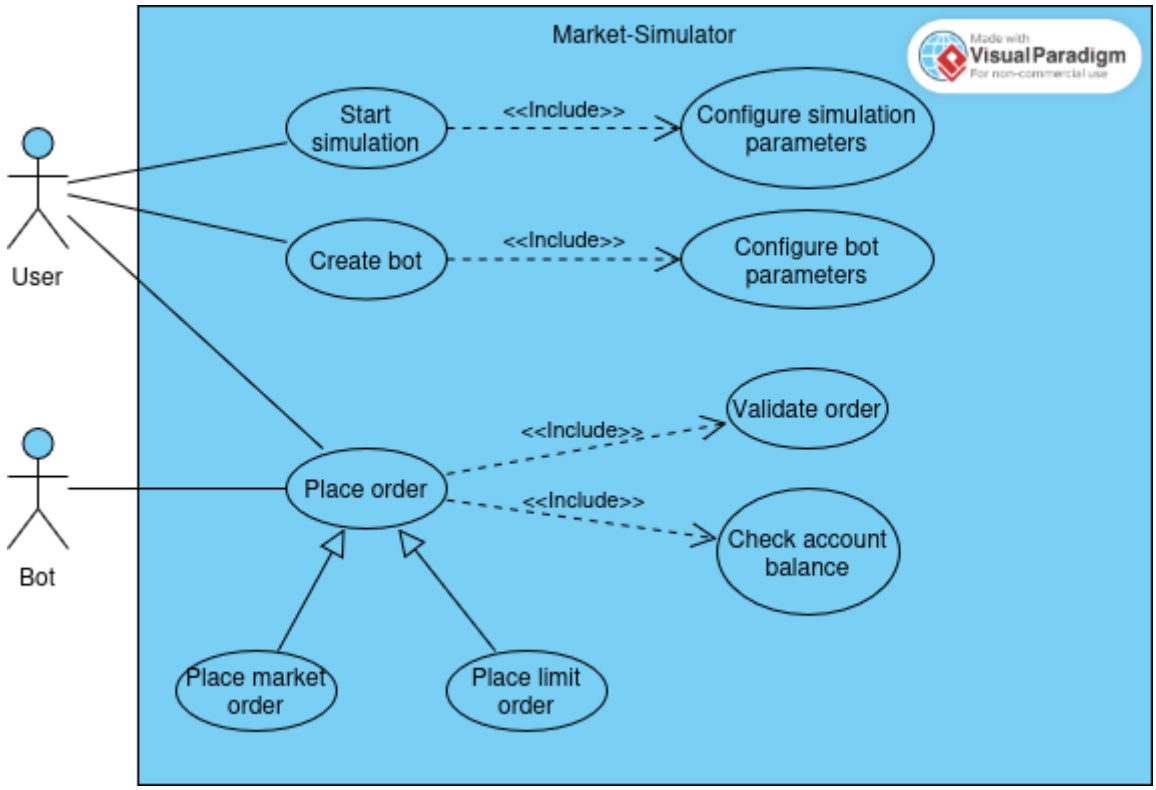
5) Orderbuch visualisieren:
Das Orderbuch muss in Echtzeit den aktuellen Zustand anzeigen, inklusive Mengen und Preisen.
Überprüfung:
Simulation aktiv → Vergleich der Anzeige mit den tatsächlichen Orders.

6) Preisgraph visualisieren:
Preisverlauf wird korrekt dargestellt; X- und Y-Achse müssen verschiebbar und per Mausrad skalierbar sein.
Überprüfung:
Interaktion mit Graphen testen (Zoom, Scroll) → Preisverlauf korrekt und nachvollziehbar.

7) Persistente Speicherung:
- Alle Transaktionen, Orders und der Simulationszustand müssen in der Datenbank gespeichert werden.
- Nach Neustart oder Absturz muss der letzte Zustand korrekt wiederhergestellt werden.
Überprüfung:
Simulation speichern → Software neu starten → Orderbuch und Transaktionen prüfen.

8) GUI-Bedienbarkeit
- Alle Funktionen müssen über die grafische Oberfläche bedienbar sein.
- Fehlerhafte Eingaben werden korrekt abgefangen, die Software stürzt nicht ab.
Überprüfung:
Alle Funktionen durchgehen → ungültige Eingaben testen → System reagiert korrekt**.**

5. Use-Case-Diagramm:



USE-CASE 1: Die Simulation starten:

Use Case Name	Simulation starten
Akteur(e)	Benutzer
Ziel	Benutzer startet eine Handelssimulation
Vorbedingungen	Benutzer ist angemeldet und die Simulation kann gestartet werden
Nachbedingungen	Simulation läuft, Bot kann erstellt und Aktionen ausgeführt werden
Hauptablauf	1. Benutzer klickt „Simulation starten“ 2. System initialisiert die Simulation
Alternativabläufe / Ausnahmen	1. System kann Simulation nicht starten → Fehlermeldung

USE-CASE 2: Einen Bot erstellen:

Use Case Name	Bot erstellen
Akteur(e)	Benutzer
Ziel	Benutzer erstellt einen Handelsbot
Vorbedingungen	Simulation läuft
Nachbedingungen	Bot ist erstellt und kann Aufträge platzieren
Hauptablauf	1. Benutzer wählt „Bot erstellen“ 2. System legt Bot an
Alternativabläufe / Ausnahmen	1. Bot kann nicht erstellt werden → Fehlermeldung

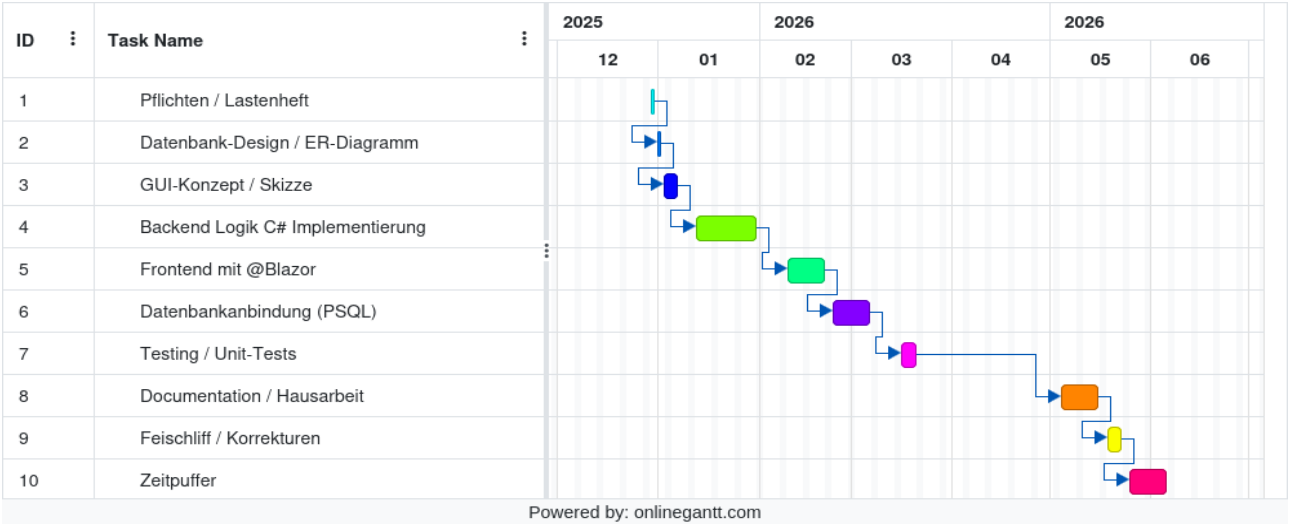
USE-CASE 3: Limit-Order erstellen:

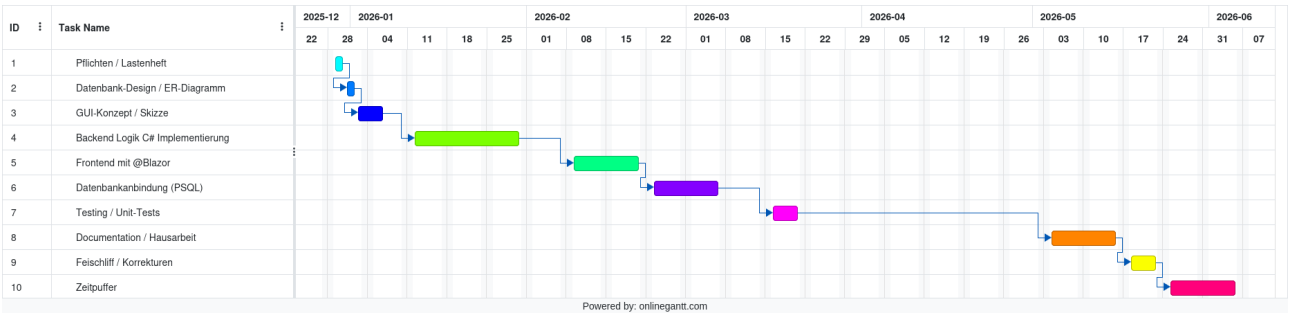
Use Case Name	Limit-Order im Orderbuch platzieren
Akteur(e)	Bot oder Benutzer
Ziel	Bot platziert einen Kauf- oder Verkaufsauftrag mit festem Preis
Vorbedingungen	Bot existiert, Simulation läuft, Benutzerkonto ist aktiv
Nachbedingungen	1. Order wird erstellt, validiert und ins Orderbuch eingetragen. 2. Falls kein entsprechendes Gegenangebot vorhanden ist, bleibt die Order im Orderbuch, bis ein passendes Gegenangebot eingeht. 3. Verrechnete Orders werden aus dem Orderbuch entfernt.
Hauptablauf	1. Bot wählt „Order platzieren“ 2. System **prüft Kontostand** (`include`) 3. System **validiert Order** (`include`) 4. Order wird ins Orderbuch eingetragen 5. Ausgeführte Orders werden als Transaktionen in einer Datenbank gespeichert
Alternativabläufe / Ausnahmen	1. Kontostand unzureichend → Order abgelehnt 2. Order ungültig → Fehlermeldung
Include	Include: Kontostand prüfen, Order validieren

USE-CASE 4: Market-Order ausführen:

Use Case Name	Market-Order ausführen
Akteur(e)	Bot oder Benutzer
Ziel	Bot oder Benutzer kauft/verkauft einen Vermögenswert zum Aktuellen Marktpreis
Vorbedingungen	Simulation läuft, Kontostand ist ausreichend, mindestens ein passendes Gegenangebot im Orderbuch vorhanden, sodass die Order ausgeführt werden kann
Nachbedingungen	Market-Order wird ausgeführt und Orders die für die Ausführung verrechnet wurden, werden aus dem Orderbuch entfernt.
Hauptablauf	1. Marktteilnehmer wählt Kaufoption 2. System prüft Kontostand und validiert Order 3. System prüft auf verfügbares Gegenangebot im Orderbuch 4. Order wird direkt zum aktuellen Marktpreis ausgeführt und Transaktionen werden erzeugt 5. Die Transaktionen werden in einer Datenbank gespeichert
Alternativabläufe / Ausnahmen	Sollte keine entsprechendes Gegenangebot im Orderbuch vorhanden sein, wird die Order nicht ausgeführt und der Vorgang wird abgebrochen

6. Projektplan





7. Produktumgebung

1) Blazor Server

Blazor Server ist ein **Web-Framework von Microsoft**, mit dem man **interaktive Webanwendungen komplett in C#** erstellen kann, ohne JavaScript zu verwenden.

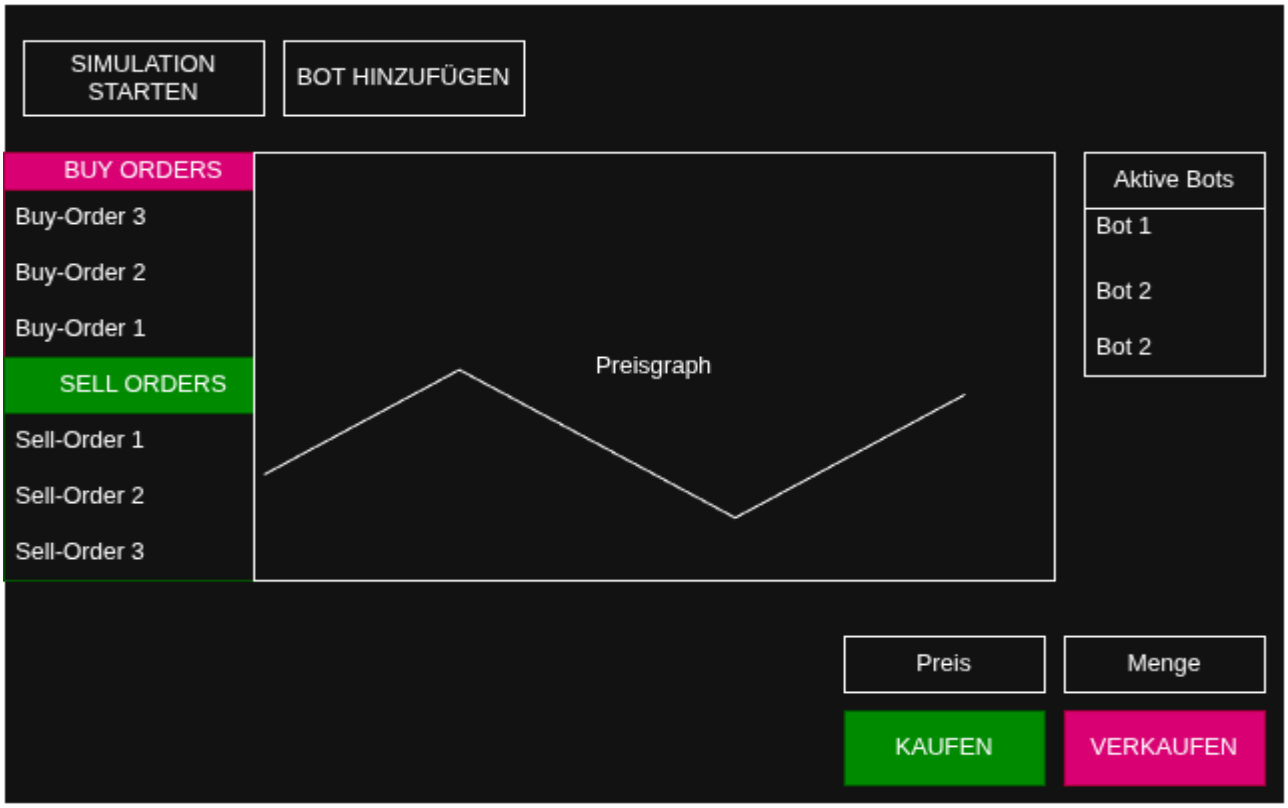
- **Frontend (UI)**: Geschrieben in C# über `.razor`-Komponenten.
- **Backend (Logik & Daten)**: Ebenfalls C# mit ASP.NET Core.
- **Kommunikation**: Die Anwendung läuft auf dem **Server**, während der Browser nur die UI rendert. Änderungen werden in **Echtzeit über SignalR** zwischen Server und Client synchronisiert.

2) PostgreSQL (PSQL)

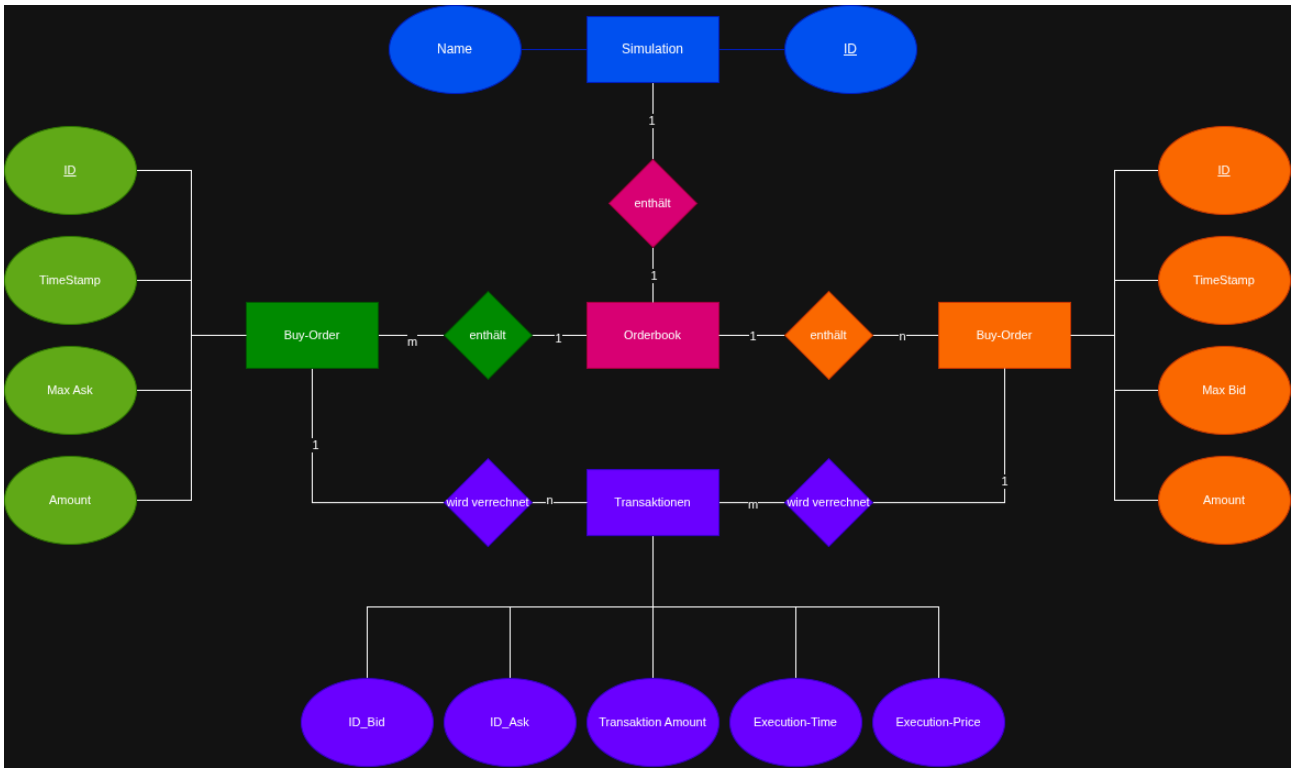
PostgreSQL (häufig PSQL genannt) ist ein **objekt-relationales Open-Source-Datenbanksystem**. Es wird für die Verwaltung von strukturierten Daten in relationalen Datenbanken verwendet und bietet viele moderne Features wie Transaktionen, Erweiterbarkeit und SQL-Konformität.

- **Relational**: Daten werden in Tabellen mit Zeilen und Spalten gespeichert.
- **Open-Source**: Kostenlos nutzbar und erweiterbar.

8. GUI-Skizze



9. DB-Entwurf



10. Testplan

Testplan für das Programm Orderbook-Simulation:

Testfall	Beschreibung	Vorbedingungen	Test-Schritte	Erwartetes Resultat
Test 1	Korrekte Verrechnung des Orderbooks	Orderbook initialisiert, mindestens 2 Orders vorhanden	Neue Orders anlegen, Verrechnung starten	Orders korrekt verrechnet, Mengen und Preise stimmen
Test 2	Richtige Ausführung der Orders	Orderbook korrekt, Testorders vorhanden	Orders eingeben, Ausführung auslösen	Orders werden in richtiger Reihenfolge und zu korrekten Preisen ausgeführt
Test 3	Speicherung in Datenbank	Datenbank erreichbar, Testorders angelegt	Orders ausführen, Datenbank prüfen	Alle Orders korrekt in DB gespeichert
Test 4	Korrekte visuelle Darstellung des Preises	Frontend aktiv, Orderbook aktuell	Preisänderungen erzeugen, Frontend prüfen	Preise korrekt und aktuell angezeigt
Test 5	Parameter sind einstellbar	System gestartet, Parameter änderbar	Parameter ändern, speichern, System prüfen	Parameter werden übernommen und wirken korrekt
Test 6	Market-Order ohne verfügbares Gegenangebot	Simulation läuft, Kontostand ausreichend, Orderbuch enthält kein passendes Gegenangebot	1. Benutzer/Bot wählt Market-Order aus 2. System prüft Gegenangebot 3. Versuch, Order auszuführen	System lehnt die Order ab und gibt eine Fehlermeldung aus. Orderbuch bleibt unverändert.
Test 7	Persistenz nach Neustart	Orders ausgeführt und gespeichert	System neu starten, Orders laden	Alle zuvor gespeicherten Orders und Simulationszustand werden korrekt wiederhergestellt

Unit-Tests:
Test 1, Test 2 und Test 3 sollen mit Unit-Tests überprüft werden.

11. Git-Repository Link

https://github.com/M0K097/MARKET_SIMULATOR