Faster Stock Option Pricing

Bachelorarbeit

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Frühjahrssemester 2022

Autoren:	Jonas Knupp, Thomas Hindermann
Betreuerin:	Prof. Dr. Mitra Purandare
Experte:	Dr. Raphael Polig
Gegenleserin:	Prof. Dr. Nathalie Weiler
Co-Betreuer:	Philipp Kramer

Abstract

Die numerische Bewertung von Optionen ist rechenintensiv, da der Optionswert vom zukünftigen Preis des Basiswertes abhängt. Europäische Optionen können mittels Monte-Carlo-Simulation bewertet werden, wobei der Rechenaufwand linear steigt, während amerikanische Optionen durch das Binomialmodell mit quadratischer Laufzeit bewertet werden können. Es existieren zahlreiche Webapplikationen zur Optionsbewertung, die es dem Nutzer allerdings nicht erlauben, den Optionswert parallelisiert mittels Multithreading auf der CPU oder der GPU berechnen zu lassen. Ausserdem müssen die Optionen oft manuell erfasst werden und es ist häufig nicht möglich, mehrere Optionen auf einmal bewerten zu lassen.

In einem ersten Schritt wurde eine Domänenanalyse durchgeführt, um die Monte-Carlo-Simulation und das Binomialmodell in Pseudocode-Algorithmen zu formalisieren. Anschliessend wurden die Anforderungen an die Webapplikation ermittelt. Aus den Anforderungen hat sich ergeben, dass die Applikation als verteiltes System konzipiert wird, da je nach Parallelisierungsansatz unterschiedliche Anforderungen an die Hardware bestehen. So wurde für die parallelisierten Berechnungen auf der CPU und GPU jeweils ein Microservice in C++ erstellt. Die Programmierung der GPU erfolgte mit dem CUDA Toolkit. Um Funktionen wie unter anderem Benutzerverwaltung, Input-Validierung, Persistenz und Kommunikation mit Yahoo Finance zu realisieren, wurde ein API-Gateway mittels NestJS implementiert. Das Frontend wurde mit React umgesetzt. Die Kommunikation zwischen den Backend-Diensten findet über einen RabbitMQ Message Broker statt, während das Frontend mit dem API-Gateway über REST kommuniziert. Um bei den parallelisierten Modellen einen möglichst grossen Speedup zu erreichen, wurden jeweils verschiedene Ansätze ausprobiert.

Das Ergebnis dieser Bachelorarbeit ist eine Webapplikation die eine parallelisierte Bewertung von europäischen und amerikanischen Optionen ermöglicht. Dem Nutzer stehen zahlreiche Konfigurationsmöglichkeiten zur Verfügung. Zusätzlich zur Möglichkeit einzelne Optionen bewerten zu lassen, gibt es mit dem “Profit-Spotter“ die Möglichkeit alle Optionen einer Aktie, die auf Yahoo Finance verfügbar ist, bewerten zu lassen. Beim “Profit-Spotter“ werden ausschliesslich amerikanische Optionen bewertet, da alle Optionen auf Yahoo Finance amerikanisch sind. Um einen maximalen Speedup zu erreichen, wird nicht jede Option einzeln bewertet, sondern unter dem Begriff “Multibinomialmodell“ Varianten erstellt, die auf die gleichzeitige Bewertung von mehreren Optionen optimiert sind. Da die parallelisierten Modelle in Software-Bibliotheken ausgelagert wurden, können diese auch in anderen Applikationen verwendet werden. Die Benchmarks für die CPU Varianten wurden auf einem Intel Core i7-11370H Prozessor mit vier echten Kernen und die Benchmarks für die GPU Varianten auf einer NVIDIA Tesla V100 PCIe durchgeführt. Für die Monte-Carlo-Simulation mit 10’000’000 Preispfaden wurde mit Multithreading und Vector Extensions ein Speedup von 40 erreicht, während auf der GPU ein Speedup von 179 erreicht wurde. Beim Binomialmodell mit 1’023 Zeitabschnitten wurde auf der CPU kein nennenswerter und auf der GPU ein Speedup von 9 erreicht. Beim Multibinomialmodell mit 1’023 Zeitabschnitten konnte ein Speedup von 4 auf der CPU und 900 auf der GPU erreicht werden.

Lay Summary

An den weltweiten Finanzmärkten hat sich der Handel mit Optionen etabliert. Der Wert einer Option hängt stark vom zukünftigen Kurs der Aktie, auf die sich die Option bezieht, ab. Da der zukünftige Aktienkurs unbekannt ist, stellt das Finden eines fairen Preises für Optionen eine komplexe Angelegenheit dar. Es existieren verschiedene Modelle um Optionen zu bewerten. In dieser Bachelorarbeit wurde Monte-Carlo-Simulation zur Bewertung von europäischen und das Binomialmodell zur Bewertung von amerikanischen Optionen verwendet. Beide Modelle sind sehr rechenintensiv. Da die Börse sehr schnelllebig ist, ist es notwendig, dass Optionswerte möglichst schnell berechnet werden.

Im Rahmen dieser Bachelorarbeit wurde die Webapplikation StOP (für Stock Option Pricing) entwickelt, die es Nutzern ermöglicht, Optionen mit Monte-Carlo-Simulation und dem Binomialmodell zu bewerten. Damit die Details einer Option nicht manuell vom Nutzer eingegeben werden müssen, bietet StOP die Möglichkeit, dass Optionen von Yahoo Finance importiert werden können. Ferner kann der Nutzer praktisch alle Aspekte der Optionsbewertung konfigurieren. So ist es beispielsweise möglich, eine eigene Volatilität einzugeben oder Volatilität automatisch durch StOP schätzen zu lassen.

Damit der Nutzer das Resultat möglichst zeitnah erhält, wurden die Modelle parallelisiert. Neben der Funktion, einzelne Optionen bewerten zu lassen, gibt es den "Profit-Spotter". Dabei wählt der Nutzer ein Aktiensymbol und StOP bewertet alle Optionen zu dieser Aktie, die auf Yahoo Finance verfügbar sind, und zeigt dem Nutzer, bei welchen Optionen der Marktpreis unter dem Modellpreis liegt. Da hier je nach Aktie die Werte mehrerer tausend Optionen berechnet werden müssen, wurde ein eigener Algorithmus dafür entwickelt, anstatt dass jede Option einzeln bewertet wird.

Es wurden verschiedene Ansätze zur Parallelisierung ausprobiert. Moderne CPUs haben mehrere Prozessorkerne, welche unabhängig voneinander Berechnungen ausführen können. Der erste Ansatz nutzt diese Eigenschaft aus, indem die Optionsbewertung mittels Multithreading parallelisiert wurde. Ein weiterer Ansatz war die Parallelisierung durch die Nutzung einer Grafikkarte. Im Vergleich zu CPUs, die in gegenwärtigen Computern verbaut sind und normalerweise etwa vier Kerne haben, bieten Grafikkarten mehrere tausend Kerne.

Die Benchmarks für die CPU Varianten wurden auf einem Intel Core i7-11370H Prozessor mit vier echten Kernen und die Benchmarks für die GPU Varianten auf einer NVIDIA Tesla V100 PCIe durchgeführt. Bei der Monte-Carlo-Simulation mit 10'000'000 Preispfaden konnten die Berechnungen mit Multithreading 40 mal schneller und auf der Grafikkarte 179 mal schneller durchgeführt werden. Beim Binomialmodell mit 1'023 Zeitabschnitten konnten die Berechnungen auf der CPU nicht nennenswert und auf der Grafikkarte um neun beschleunigt werden. Bei der Bewertung von 1'571 Optionen mittels des Multibinomialmodells mit 1'023 Zeitabschnitten konnte die Performance auf der CPU um vier und auf der Grafikkarte um 900 verbessert werden.

StOP ermöglicht es Nutzern selbst auszuwählen, ob die Parallelisierung auf der CPU oder der Grafikkarte stattfinden soll. Auch die Anzahl Preispfade bzw. Zeitabschnitte kann vom Nutzer bestimmt werden. Neben dem Optionspreis zeigt StOP auch die Zeit an, die für die Berechnung des Optionspreises aufgewendet wurde, was es Nutzern erlaubt, selbst zu experimentieren.

In dieser Bachelorarbeit wurde gezeigt, dass die Bewertung von Optionen mittels Monte-Carlo-Simulation und Binomialmodell durch Parallelisierung massiv beschleunigt werden kann. Mit StOP wurde eine Webapplikation entwickelt, die diese parallelisierten Algorithmen in einer benutzerfreundlichen Weise zugänglich macht.

Danksagung

Wir möchten allen danken, die uns bei der Umsetzung dieser Bachelorarbeit unterstützt haben. Insbesondere gebührt unser Dank Prof. Dr. Mitra Purandare, die unsere Fragen schnell und unkompliziert beantwortet und uns den Zugriff auf den OST-Cluster ermöglicht hat. Ausserdem möchten wir Dr. Raphael Polig danken, der uns auf die Vorteile einer message-basierten Kommunikation hingewiesen hat. Adrian Rohner danken wir für die Einführung in den OST-Cluster.

Rapperswil, 16. Juni 2022
Jonas Knupp und Thomas Hindermann

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Problembeschreibung

An den Finanzmärkten hat sich der Handel mit Optionen etabliert. Um eine vorteilhafte Investitionsentscheidung zu treffen, ist es für Anleger wichtig zu erfahren, ob der Preis einer angebotenen Option angemessen ist. Im Rahmen dieser Bachelorarbeit wurden ausschliesslich europäische und amerikanische Aktienoptionen berücksichtigt. Es existieren zahlreiche Webangebote zur Optionsbewertung. Diese bieten jedoch keine Funktionalität an, um Optionswerte parallelisiert berechnen zu lassen. Im Rahmen dieser Bachelorarbeit sollte nun mit StOP (für Stock Option Pricing) eine Webapplikation entwickelt werden, die die parallelisierte Bewertung von Optionen ermöglicht. Es soll möglich sein, einzelne Optionen sowie alle Optionen einer Aktie gemeinsam zu bewerten. Die Parallelisierung soll auf der CPU mittels Multithreading und auf der GPU mit dem CUDA-Toolkit erfolgen.

Dieses Kapitel befasst sich mit der Analyse der Problemstellung. In einem ersten Schritt wurde die Domänenanalyse durchgeführt, in Folge derer ein Überblick über die Thematik der Optionsbewertung gewonnen wurde. Durch die Domänenanalyse konnten zentrale Begriffe geklärt werden. Ausserdem wurden die Variablen, die den Optionspreis beeinflussen identifiziert. Mit der Black-Scholes-Formel und Monte-Carlo-Simulation konnten geeignete Modelle zur Bewertung von europäischen Optionen bestimmt werden. Für amerikanische Optionen wird das Binomialmodell nach Cox, Ross und Rubinstein verwendet.

In einem nächsten Schritt wurden verwandte Arbeiten betrachtet. Dies beinhaltet eine kurze Sichtung von bereits existierenden Werkzeugen zur Optionsbewertung. Ausserdem wurde eine Literaturrecherche durchgeführt, um herauszufinden, welche Ansätze zur Parallelisierung der in der Domänenanalyse evaluierten Modelle bereits ausprobiert wurden. Es hat sich gezeigt, dass eine Fülle an Literatur zur parallelisierten Bewertung von europäischen und amerikanischen Optionen existiert.

Anschliessend wurde die Anforderungsanalyse durchgeführt. Für Nutzer ist es sinnvoll, wenn sie Bewertungsanfragen sowie Bewertungsresultate speichern können, sodass Vergleiche zwischen den verschiedenen Parallelisierungsansätzen einfacher möglich sind. Insgesamt wurden fünf funktionale sowie 13 nicht-funktionale Anforderungen identifiziert.

1.1 Domänenanalyse

Alle Angaben, bei welchen nicht explizit eine andere Quelle angegeben wude, stammen aus “The Concepts and Practice of Mathematical Finance“ von Joshi [1].

Eine Option ist ein Finanzinstrument, das dem Halter das Recht, aber nicht die Pflicht, zum Kauf oder Verkauf einer bestimmten Menge eines Basiswertes zu einem bestimmten Preis, dem Strike Preis, in einem festgelegten Zeitraum gibt. Eine Call-Option verbrieft das Recht auf den Kauf des Basiswertes, während eine Put-Option das Recht auf den Verkauf des Basiswertes verbrieft.

Optionen können nach den Ausübungsrechten unterschieden werden. Europäische Optionen können ausschliesslich am Fälligkeitsdatum ausgeübt werden. Amerikanische Optionen können an jedem Handelstag vor dem und am Fälligkeitsdatum ausgeübt werden. Optionen, bei deren Ausübung auschliesslich die Differenz zwischen dem Kurs des Basiswertes und dem Strike Preis ausgezahlt wird, werden Vanilla Optionen genannt. Beeinflussen auch andere Faktoren den Auszahlungspreis, spricht man von exotischen Optionen. So wird beispielsweise bei asiatischen Optionen nicht der Preis des Basiswertes bei der Ausübung betrachtet, sondern der durchschnittliche Preis des Basiswertes über einen bestimmten Zeitraum.

1.1.1 Qualitative Bewertung von Optionen

Wie bei anderen Derivaten muss auch bei Optionen der Zeitwert des Geldes beachtet werden. Generell wird in der Finanzmathematik davon ausgegangen, dass Geld risikolos verzinst werden kann. Dies hat zur Folge, dass Geld welches man in der Gegenwart besitzt einen grösseren Wert hat, als Geld welches man erst in der Zukunft besitzen wird. Die Gleichung

$$PV = e^{-rT} FV \tag{1.1}$$

beschreibt den Zeitwert des Geldes mathematisch. PV steht für den gegenwärtigen Wert, r für den risikofreien Zinssatz, T für die Zeit, die vergeht bis man das Geld in der Zukunft erhalten wird und FV für den zukünftigen Wert [2].

Der Wert einer Option setzt sich aus zwei Komponenten zusammen. Der intrinsische Wert ist positive Differenz zwischen Strike Preis und gegenwärtigem Marktpreis des Basiswertes. Für Call-Optionen ist der intrinsische Wert IV durch

$$IVC = max(S_T - K, 0) \tag{1.2}$$

gegeben. Für Put-Optionen ergibt sich der intrinsische Wert gemäss

$$IVP = max(K - S_T, 0) \tag{1.3}$$

Daneben haben Optionen einen Zeitwert. Der Zeitwert ist in der Hoffnung des Investors begründet, dass sich der Preis des Basiswertes zu seinem Vorteil entwickelt. Zum Fälligkeitsdatum ist der Zeitwert null [3].

Zwei wichtige Variablen, die den intrinsischen Wert einer Option bestimmen ist der Preis des Basiswertes und der Strike Preis. Bei Call-Optionen steigt der Wert einer Option mit steigendem Preis des Basiswertes, da der Inhaber die Option zum fixen Strike Preis kaufen kann und sich somit sein Gewinn vergrössert. Bei einer Put-Option sinkt hingegen der Wert einer Option mit steigendem Wert des Basispreises, da der Eigentümer der Option den Basiswert zu einem niedrigeren Preis, dem Strike Preis, verkaufen kann.

Eine weitere Variable ist die Zeit bis zum Fälligkeitsdatum. Je grösser die Dauer bis zum Fälligkeitsdatum, desto grösser ist der Wert der Option, da die Option in diesem Zeitraum mehr Möglichkeiten hat in den Gewinnbereich einzutreten.

Auch die Volatilität des Basiswerts beeinflusst den Wert einer Option. Mit zunehmender Volatilität nimmt die Wahrscheinlichkeit zu, dass der Basiswert sinkt oder steigt. Ein Basiswert mit einer Volatilität von 0 würde sich niemals verändern, was bedeutet, dass man mit einer Option niemals einen Gewinn erzielen könnte, sofern die Option beim Kauf nicht schon im Gewinnbereich liegt. Obschon sich der Preis des Basiswertes durch eine hohe Volatiltät auch in eine für den Optionsinhaber schlechte Richtung entwickeln kann, ist der maximale Verlust des Optionsinhabers durch den Preis der Option begrenzt.

Ein steigender risikofreier Zinssatz hat unterschiedliche Auswirkungen auf Call und Put-Optionen. Call-Optionen gewinnen mit steigendem risikofreien Zinssatz an Wert, da der Optionseigentümer das Geld, das er sonst für den Basiswert ausgegeben hätte zu einem höheren Zinssatz verzinsen könnte. Bei Put-Optionen ist der Effekt umgekehrt, der Wert sinkt mit steigendem risikofreien Zinssatz. Da der Optionseigentümer den Basiswert erst in der Zukunft verkauft, entgehen ihm die potentiellen Zinseinnahmen auf das durch den sofortigen Verkauf des Basiswertes eingenommene Geld.

Ein letzter Faktor sind Dividendenzahlungen, die während der Laufzeit einer Option anfallen. In dieser Arbeit wird diese Faktor allerdings nicht berücksichtigt, da das Black-Scholes-Modell davon ausgeht, dass die begutachteten Aktien keine Dividenden ausschütten.

Die Tabelle 1.1 fasst die Auswirkungen auf die Veränderung einer Variable auf den Preis einer Option zusammen. Ein + bedeutet, dass eine Erhöhung der Variable zu einer Erhöhung des Optionswertes führt oder der Optionswert konstant bleibt. Ein − bedeutet, dass eine Erhöhung der Variable zu einer Verminderung des Optionswertes führt oder der Optionswert konstant bleibt [4]. Diese Angaben sind insbesondere nützlich für Plausibilitätstests.

Tabelle 1.1: Auswirkungen bei Veränderung einer Variable

Variable	**Call**	**Put**
Gegenwärtiger Preis des Basiswertes	+	−
Strike Preis	−	+
Zeit bis zum Ausübungszeitpunkt	+	+
Volatilität	+	+
Risikofreier Zinssatz	+	−

1.1.2 Quantitative Bewertung von europäischen Optionen

Die Black-Scholes-Gleichung

$$\frac{\partial C}{\partial t} + rS\frac{\partial C}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} = rC \tag{1.4}$$

ist eine partielle Differentialgleichung, die von allen Preisfunktionen sämtlicher Derivate auf eine Aktie ohne Dividenden erfüllt werden muss. Da Optionen ein Derivat sind, gilt die Black-Scholes-Gleichung auch für Optionen. $C(S,T)$ ist die Preisfunktion des Derivats basierend auf dem Preis des Basiswerts S und der Zeit t und dem risikofreien Zinssatz r.

Dabei gelten die folgenden Annahmen [4]:

- Der Preis des Basiswertes folgt einer geometrischen brownschen Bewegung.
- Der Leerverkauf von Finanzinstrumenten ist gestattet.
- Es gibt keine Steuern oder Transaktionskosten.
- Vom Abschluss bis zum Ende des Derivats werden keine Dividenden ausgezahlt.
- Es gibt keine Möglichkeit zur risikolosen Arbitrage.
- Die Finanzinstrumente werden kontinuierlich gehandelt.
- Der risikofreie Zinssatz ist konstant und für alle Laufzeiten gleich.
- Die Option kann nicht frühzeitig ausgeübt werden

Analytische Lösung der Black-Scholes-Gleichung

Für den Preis von europäischen Optionen existiert eine analytische Lösung der Black-Scholes-Gleichung. Die Formel

$$C = S\phi(d_1) - Ke^{-rT}\phi(d_2) \tag{1.5}$$

kann zur Bestimmung des Wertes einer Call-Option verwendet werden, wobei S der Preis des Basiswertes zum Bewertungszeitpunkt, K der Strike Preis, σ die Volatilität des Basiswertes, r der risikofreier Zinssatz, T die Zeit in Jahren bis zum Fälligkeitsdatum und ϕ die kumulative Normalverteilung ist.
Analog dazu kann die Formel

$$P = -S\phi(-d_1) + Ke^{-rT}\phi(-d_2) \tag{1.6}$$

für die Berechnung des Wertes von Put-Optionen hinzugezogen werden. Für die Gleichungen 1.5 und 1.6 gilt

$$d_j = \frac{log(\frac{S}{K}) + (r + (-1)^{j-1}\frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}} \tag{1.7}$$

Formeln 1.5 und 1.6 werden in der Literatur stellenweise als Black-Scholes-Formel bezeichnet.

Monte-Carlo-Simulation

Monte-Carlo-Simulation ist ein stochastisches Verfahren zur numerischen Lösung von analytisch nicht oder nur schwer lösbaren Problemen durch das Ziehen von Stichproben. Sie basiert auf dem Gesetz der grossen Zahl, welches besagt, wenn Y_n eine Folge von unabhängigen, identisch verteilten Zufallsvariablen ist, der Erwartungswert, welcher für alle Zufallsvariablen identisch ist, wie folgt berechnet werden kann

$$\lim_{N\to\infty} \frac{1}{N}\sum_{j=1}^{N} Y_j = E[Y_1] \tag{1.8}$$

Im Black-Scholes-Modell folgt S_T, also der Preis des Basiswertes, einer geometrischen brownschen Bewegung. Der Preis des Basiswertes ergibt sich deshalb zu

$$S_T = S_0 e^{(r-\frac{1}{2}\sigma^2)T + \sigma\sqrt{T}\mathcal{N}(0,1)} \tag{1.9}$$

Damit ist der Preis für Call-Optionen durch

$$C = e^{-rT} E[max(S_T - K, 0)] \tag{1.10}$$

und für Put-Optionen durch

$$P = e^{-rT} E[max(K - S_T, 0)] \tag{1.11}$$

gegeben. Das Gesetz der grossen Zahl erlaubt nun die Approximation des Erwartungswertes durch das Ziehen von Stichproben. Der Standardfehler ist

$$SE = \sqrt{\frac{V}{N}} \tag{1.12}$$

mit der Varianz der Stichprobe V und der Anzahl Stichproben N. Das Ergebnis der Monte-Carlo-Simulation konvergiert mit wachsendem N zum Ergebnis der Black-Scholes-Formel.

1.1.3 Quantitative Bewertung von amerikanischen Optionen

Die Informationen dieses Abschnitts stammen von Hull [4]. Da amerikanische Optionen nicht mit dem Black-Scholes-Modell bewertet werden können, gibt es dafür alternative Verfahren. Ein verbreitetes und zugleich intuitives Modell ist das Binomialmodell. Dabei wird ein Binomialbaum erstellt, der verschiedene Pfade des Basiswertpreises repräsentiert. Bei jedem Zeitschritt steigt der Preis des Basiswertes um eine gewisse Grösse mit einer bestimmten Wahrscheinlichkeit. Gleichfalls gibt es die Möglichkeit, dass der Preis des Basiswertes um eine gewisse Grösse mit einer bestimmten Wahrscheinlichkeit fällt. In dieser Arbeit werden Binomialbäume nach Cox, Ross und Rubinstein [5] verwendet.

In einem ersten Schritt wird der Basiswertpreis zu allen Zeitpunkten bestimmt. Die Abbildung 1.1 zeigt die Formeln am Beispiel eines dreistufigen Binomialbaumes.

Anschliessend wird der Wert der Option von rechts nach links bestimmt. Dies hat den Grund, dass zum Fälligkeitsdatum der Wert einer Option einfach bestimmt werden kann, es ist nämlich der intrinsische Wert der Option, da die Option keinen Zeitwert mehr innehat. Sind die Optionswerte für alle Blattknoten bestimmt, kann der binomiale Wert BV für die Elternknoten der Blattknoten gemäss

$$BV_{t-1} = (pC_{tu} + (1-p)C_{td})e^{-r\Delta t} \tag{1.13}$$

bestimmt werden. Der binomiale Wert ist der Wert der Option unter der Annahme, dass die Option zu diesem Zeitpunkt gehalten und somit nicht ausgeübt wird. Da hier amerikanische Optionen betrachtet werden, kann die Option zu jedem Zeitpunkt ausgeübt werden. Entsprechend ergibt sich der Wert einer Option wie folgt

$$V_{t-1} = max(IV, BV) \tag{1.14}$$

Diese Prozedur wird fortgesetzt, bis der Wurzelknoten erreicht ist. Der Wert einer Call-Option beim j. Knoten (von unten nach oben indexiert) zum Zeitpunkt ist $i\Delta t$ durch

$$C_{i,j} = max(max(S_0 u^j d^{i-j} - K, 0), e^{-r\Delta t}(pC_{i+1,j+1} + (1-p)C_{i+1,j})) \tag{1.15}$$

und für eine Put-Option durch

$$P_{i,j} = max(max(K - S_0 u^j d^{i-j}, 0), e^{-r\Delta t}(pP_{i+1,j+1} + (1-p)P_{i+1,j})) \tag{1.16}$$

gegeben.

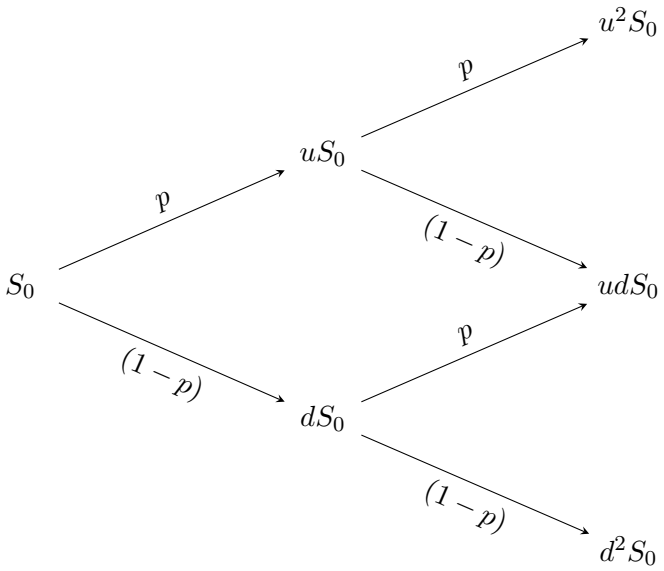


Abbildung 1.1: Dreistufiger Binomialbaum

Die Wahrscheinlichkeit p für eine Reduktion des Basiswertpreises ergibt sich aus der Formel

$$p = \frac{e^{r\Delta t} - d}{u - d} \tag{1.17}$$

Der Faktor u bestimmt, wie stark der Basiswert in einem Zeitschritt steigt und ist gegeben durch

$$u = e^{\sigma\sqrt{\Delta t}} \tag{1.18}$$

während der Faktor d bestimmt, wie stark der Basiswert in einem Zeitschritt schrumpft mit der Formel

$$d = e^{-\sigma\sqrt{\Delta t}} \tag{1.19}$$

bestimmt wird.

Die Anzahl Zeitabschnitte n bestimmt die Präzision mit der der Optionspreis bestimmt wird. Mit zunehmender Anzahl Zeitabschnitten nimmt die Präzision zu, da die Anzahl simulierter Preispfade erhöht wird. Die Anzahl Blattknoten ist durch $n+1$ und die Anzahl simulierter Preispfade ist durch 2^n gegeben. Von diesen 2^n Preispfaden sind allerdings viele redundant, da der Aktienpreis gleich ist, wenn der Preis zuerst steigt und dann fällt als wie wenn der Preis zuerst fällt und dann steigt. Deshalb werden in dieser Arbeit rekombinierte Binomialbäume verwendet, bei dem identische Knoten zusammengefasst werden. Die Anzahl Knoten kann so auf $\frac{n^2}{2} + \frac{3n}{2} + 1$ reduziert werden. Nach Hull [4] erhält man mit $n = 30$ eine für die Praxis ausreichend gute Approximation des Optionspreises.

1.1.4 Multibinomialmodell

Als Multibinomialmodell bezeichnen wir batchweise Optionsbewertung mittels Binomialmodell. Im Gegensatz zu den anderen Modellen wird dabei nicht eine einzelne Option bewertet, sondern mehrere Optionen auf einmal. In unserer Arbeit beschränken wir uns darauf, mehrere Optionen, die zum gleichen Basiswert gehören, gleichzeitig zu bewerten.

Dies bedeutet, dass der Preis des Basiswertes zum Bewertungszeitpunkt und der risikofreie Zinssatz für alle Optionen identisch sind. Eigentlich könnte auch angenommen werden, dass die Volatilität für alle Optionen identisch sei, allerdings möchten wir die von Yahoo Finance zur Verfügung gestellte implizite Volatilität nutzen. Diese ist pro Option unterschiedlich, selbst wenn diese zum gleichen Basiswert gehören. Deshalb erlauben wir in diesem Modell die Volatilität pro Option festzulegen.

1.1.5 Ermittlung der Variablenwerte für die Formeln

Die Black-Scholes-Formel, Monte-Carlo-Simulation und das Binomialmodell benötigen mehrere Parameter zur Ermittlung des Optionspreises. Dieser Abschnitt beschreibt, wie Werte für die Parameter ermittelt werden können.

Basiswertpreis zum Bewertungszeitpunkt

Der Basiswertpreis zum Bewertungszeitpunkt kann von der Yahoo Finance API abgefragt werden.

Strike Preis

Der Strike Preis einer Option kann ebenfalls von der Yahoo Finance API abgefragt werden.

Volatilität

Es existieren verschiedene Verfahren um die zukünftige Volatilität zu schätzen.

Historische Volatilität Die erwartete Volatilität des Basiswertes während der Zeit bis die Option ausgeübt wird, kann mit der historischen Volatilität, welche über die Yahoo Finance API ermittelt werden kann, approximiert werden. Mit

$$u_i = ln(\frac{S_i}{S_{i-1}}), i = 1, 2, ..., n \tag{1.20}$$

wobei S_i der Preis des Basiswertes zum Zeitpunkt i ist. Die Standardabweichung s von u_i ist gegeben durch

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(u_i - \overline{u})^2} \tag{1.21}$$

Die Volatilität kann nun durch

$$\hat{\sigma} = \frac{s}{\sqrt{\tau}} \tag{1.22}$$

geschätzt werden. τ ist die Länge der Intervalle in Jahren zwischen den einzelnen Zeitpunkten bei denen der Preis des Basiswertes betrachtet wird.

Die Wahl eines geeigneten n ist schwierig, da mehr historische Daten grundsätzlich ein präziseres Resultat liefern, aber alte Daten eventuell nicht mehr relevant sind um die zukünftige Volatilität vorherzusagen. Als eine Faustregel um ein vernünftiges Resultat zu erhalten, kann n gleich der Anzahl Tage gesetzt werden bis die Option ausläuft. Würde beispielsweise eine Option betrachtet werden, die in 100 Tagen ausläuft, würde man n=100 setzen und die Tagesschlusskurse des Basiswertes für die vorherigen 100 Tage betrachten. Alternativ kann ein konstanter Wert zwischen 90 und 180 Tagen verwendet werden.

Der Standardfehler kann durch

$$SE = \frac{\hat{\sigma}}{\sqrt{2n}} \tag{1.23}$$

approximiert werden [4].

Implizite Volatilität Bei diesem Verfahren wird angenommen, dass die Volatilität durch den beobachteten Marktpreis einer Option impliziert wird. Dazu wird angenommen, dass alle übrigen Parameter sowie der Optionspreis bekannt sind. Anschliessend wird ein Wert für die Volatilität gesucht, so dass das verwendete Preismodell die Option zum Marktpreis bewertet [4]. Die implizite Volatilität kann von der Yahoo Finance API abgefragt werden.

Zinssatz

Der risikofreie Zinssatz kann durch den Zinssatz einer Treasury Bill mit einer Laufzeit, die ähnlich der Laufzeit der Option ist, geschätzt werden [6]. Da über die Yahoo Finance API nur Treasury Bills mit einer Laufzeit von 13 Wochen sowie 5, 10 und 30 Jahren verfügbar sind, wird für alle Optionslaufzeiten der Wert einer Treasury Bill mit 13 Wochen Laufzeit verwendet.

Zeit in Jahren bis zum Fälligkeitsdatum

Auch dieser Wert kann von der Yahoo Finance API abgefragt werden. Da Yahoo Finance das Fälligkeitsdatum zurückliefert muss die Differenz in Jahren selbst ausgerechnet werden.

1.1.6 Domänenmodell

Aus den gewonnen Informationen durch die Domänenanalyse konnte das in Abbildung 1.2 zu sehende Domänenmodell erstellt werden.

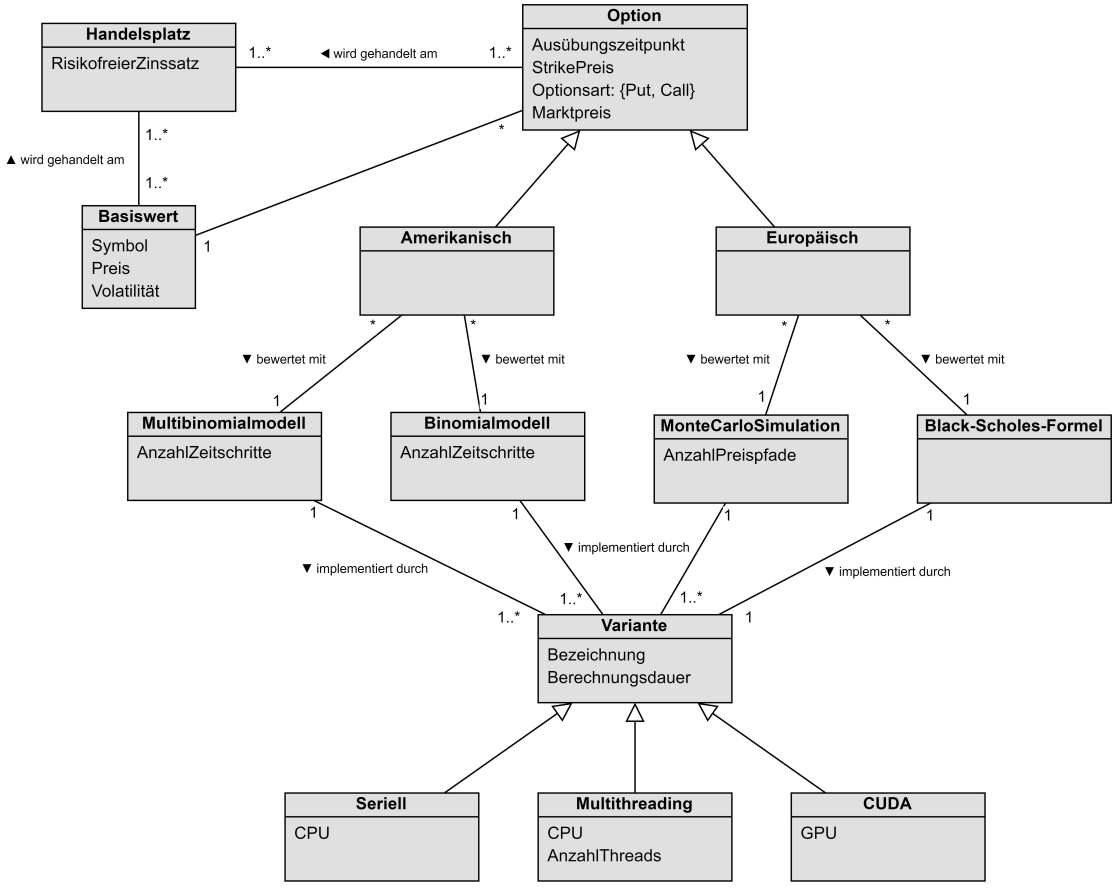


Abbildung 1.2: Domänenmodell

1.2 Verwandte Arbeiten

In diesem Abschnitt werden zunächst Applikationen, die einen ähnlichen Funktionsumfang wie StOP bieten vorgestellt. Anschliessend wird über Arbeiten, die sich ebenfalls mit der Parallelisierung von Monte-Carlo-Simulation und des Binomialmodells beschäftigen berichtet.

1.2.1 Verwandte Applikationen

Im Web existieren zahlreiche Dienste zur Bewertung von europäischen und amerikanischen Optionen. Aufgrund der grossen Anzahl beschränkt sich die Diskussion an dieser Stelle auf optionseducation.org[1], hoadley.net[2], cboe.com[3] und interactivebrokers.com[4].

Alle Dienste ausser interactivebrokers.com, bei dem nicht klar ist um welchen Optionstyp es geht, unterstützen europäische und amerikanische Optionen.

Tabelle 1.2: Eigenschaften verschiedener Optionsbewertungs-Dienste

Dienst	**Europäische Optionen**	**Amerikanische Optionen**	**Besonderheiten**
optionseducation.org	Black-Scholes	Binomialmodell (100)	Nutzer kann Auswahl von Optionen zu einer Aktie aufrufen
hoadley.net	Black-Scholes, Binomialmodell (1-150), Trinomialmodell (1-150)	Black-Scholes, Binomialmodell (1-150), Trinomialmodell (1-150)	Binomial- und Trinomialbaum werden als Abbildung dargestellt
cboe.com	Nicht ermittelbar	Nicht ermittelbar	-
interactivebrokers.com	Nicht ermittelbar	Nicht ermittelbar	-

Wie aus Tabelle 1.2 hervorgeht gibt es grosse Unterschiede zwischen den Diensten und kein Dienst unterstützt Monte-Carlo-Simulation. Dies ist vermutlich darauf zurückzuführen, dass es weniger aufwändig ist, europäische Optionen mit dem Black-Scholes-Modell zu bewerten, als mit Monte-Carlo-Simulation und die Implementierung von Monte-Carlo-Simulation für amerikanische Optionen im Vergleich zum Binomialmodell komplexer und damit fehleranfälliger ist.

Die Dienste richten sich primär an Personen, die ausschliesslich am Optionswert interessiert sind, was unter anderem daran zu erkennen ist, dass das verwendete Modell teilweise unterschlagen wird. Keiner der Dienste erwähnt, ob die Optionswerte parallelisiert berechnet werden. Es ist jedoch davon auszugehen, dass dies nicht der Fall ist, da Binomialbäume in der Grösse wie sie von den Diensten angeboten werden innerhalb kurzer Zeit seriell berechnet werden können und die Bewertung mit dem Black-Scholes-Modell für einzelne Optionen nicht sinnvoll parallelisiert werden kann.

[1] Siehe: https://www.optionseducation.org/toolsoptionquotes/optionscalculator
[2] Siehe: https://www.hoadley.net/options/calculators.htm
[3] Siehe: https://www.cboe.com/education/tools/options-calculator/
[4] Siehe: https://www.interactivebrokers.com/en/trading/options-calculator.php

1.2.2 Verwandte Ansätze zur Parallelisierung

In der Literatur finden sich einige Ansätze um europäische Optionen mittels Monte-Carlo-Simulation auf der GPU zu bewerten. Die offizielle Sammlung von Beispielen, die von NVIDIA zur Verfügung gestellt wird, enthält eine Monte-Carlo-Simulation-Implementierung zur gleichzeitigen Bewertung mehrerer Optionen [7], welche von Podlozhnyuk und Harris [8] genauer beschrieben wird. Dabei werden zwei Ansätze verfolgt: Mehrere Blöcke und ein Block pro Option, wobei ersterer optimal für die Bewertung einer grossen Anzahl an Preispfaden und letzterer optimal für die Bewertung von einer kleiner Anzahl Preispfaden geeignet ist.

Grauer-Gray et al. [9] haben unter anderem die Black-Scholes-Formel und die Quasi-Monte-Carlo-Simulation der QuantLib Bibliothek mittels GPU und Multicore-Programmierung parallelisiert. Bei beiden Modellen wird mit einfacher Genauigkeit gerechnet. Bei der Batch-Bewertung von 20'000 Optionen konnte bei der Black-Scholes-Formel mit der Parallelisierung durch CUDA ein Speedup von über 100 erreicht werden, während mit OpenMP ein Speedup von sieben erreicht wurde. Die Monte-Carlo-Simulation wurde unter anderen mit CUDA parallelisiert. Dabei erfolgte die Generierung der Zufallszahlen mittels cuRAND und die Summierung der Optionswerte mit Thrust. Bei der Bewertung einer einzelnen Option mit 10'000 Preispfaden wurde ein Speedup von etwa 1'000 im Vergleich zur seriellen Implementierung erreicht. Mit OpenMP wurde ein Speedup von 40 festgestellt. Die Summierung und Berechnung des Durchschnitts der Optionswerte wurde nicht in die Berechnung des Speedups miteinbezogen. Bei den OpenMP Varianten wurde festgestellt, dass der Compiler Vektorinstruktionen generiert hat.

Abbas-Turki et al. [10] verwenden Monte-Carlo-Simulation zur Bewertung von europäischen und amerikanischen Optionen. Für eine europäische Option mit einer Million Preispfaden wurde ein Speedup von nahezu 200 im Vergleich zur seriellen Implementierung erreicht.

Auch zur Optionsbewertung mittels Binomialmodell auf der GPU existiert eine Fülle an Literatur. Greinsmark und Lindström [11] berichten, dass bei der Berechnung des Binomialmodells für eine einzelne Option die GPU eine naive CPU-Implementierung ab 200 Zeitschritten übertrifft. Die Aufteilung der Arbeit auf die Threads erfolgte über die Preisachse. Ferner spekulieren sie, dass durch die gleichzeitige Berechnungen von mehreren Optionen ein grösserer Speedup erreicht werden kann.

Zhang et al. [12] schlagen einen hybriden Algorithmus vor, bei dem der Binomialbaum in mehrere kleinere Bäume unterteilt wird, die stufenweise jeweils von der CPU und GPU gemeinsam berechnet werden.

Einen anderen Weg gehen Popuri et al. [13]. Sie schlagen vor, das Binomialmodell in ein "embarassingly parallel" Problem zu transformieren und zeigen die Durchführung dieser Methode anhand von europäischen Optionen mittels Parallelisierung durch MPI.

Zur parallelisierten Optionsbewerten mittels Multithreading existiert weniger Literatur. Zhang und Man [14] berichten von ihren Erfahrungen bei der Nutzung von Multithreading bei der Implementierung von Monte-Carlo-Simulation und dem Binomialmodell. Bei der Monte-Carlo-Simulation sei es wünschenswert die Zufallszahlen parallelisiert zu generieren, wozu ein "skip ahaed" Zufallsgenerator benötigt werde. Bei der Implementierung des Binomialmodells schlagen sie vor, anstelle einer Matrix, ein eindimensionales Array zu benutzen, da dies schneller sei. Eine weitere Optimierung ist die Vermeidung der repetitiven Berechnung von gemeinsamen Teilausdrücken, die im Binomialmodell vorhanden sind.

1.3 Anforderungsanalyse

Dieser Abschnitt beschreibt das Ergebnis der Anforderungsanalyse.

1.3.1 Funktionale Anforderungen

Die Abbildung 1.3 bietet eine Übersicht über die Anwendungsfälle. Detaillierte Beschreibungen im Fully Dressed Format folgen weiter unten.

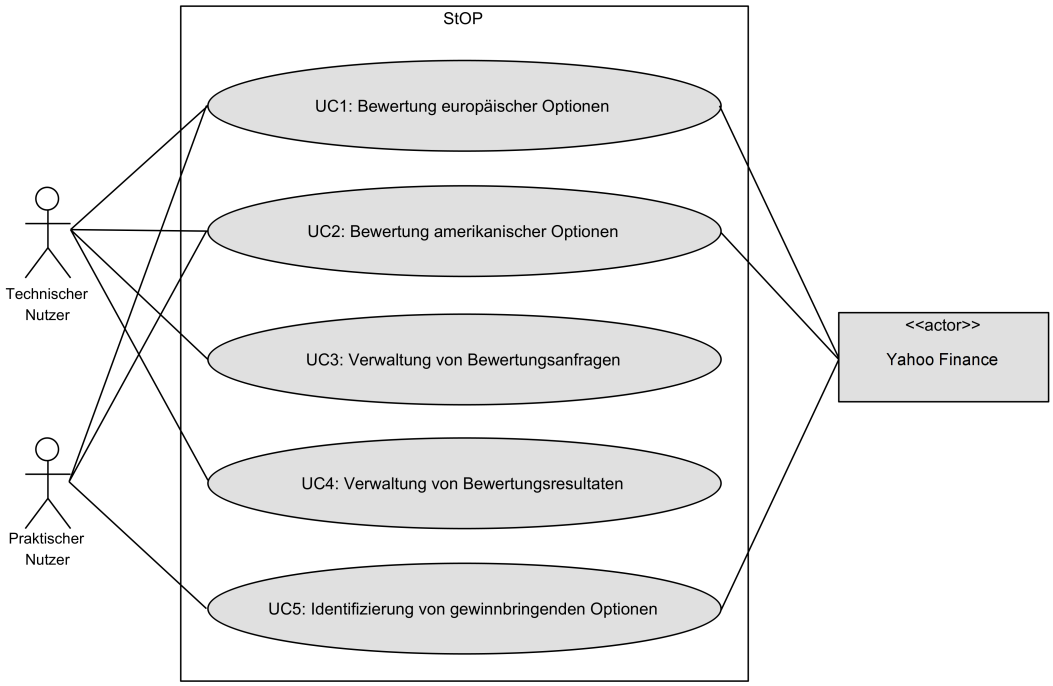


Abbildung 1.3: Diagramm der Anwendungsfälle

Aktoren

Die drei Aktoren, die mit StOP interagieren sind in der Tabelle 1.3 aufgeführt.

Tabelle 1.3: Aktoren

Aktoren	
Technischer Nutzer	Dieser Nutzer ist eher an der technischen Umsetzung der Applikation interessiert und möchte die Geschwindigkeiten der verschiedenen Parallelisierungstechniken vergleichen.
Praktischer Nutzer	Dieser Nutzer ist an den von StOP berechneten Optionswerten interessiert. Die Resultate sollen den Nutzer bei seinen Investitionsentscheidungen unterstützen.
Yahoo Finance	Yahoo Finance dient als Datenquelle von amerikanische Optionen. Ausserdem werden Informationen zu Aktien und dem risikofreien Zinssatz von Yahoo Finance abgefragt.

UC1: Bewertung europäischer Optionen

Tabelle 1.4: UC1: Bewertung europäischer Optionen

Merkmal	Beschreibung
Level	User-goal
Primary Actor	Technischer Nutzer, Praktischer Nutzer
Stakeholders and Interests	Der Nutzer möchte den Wert einer europäischen Option bestimmen.
Preconditions	-
Success Guarantee	Der Nutzer erhält alle relevanten Daten über die gewählte Option.
Main Success Scenario	1. Der Nutzer gibt die geforderten Informationen zur Berechnung eines Optionenwerts an. (Die nötigen Informationen erhält er beispielsweise bei Swissquote[5]) 2. Der Nutzer wählt, ob er den Preis mit der Black-Scholes-Formel oder mit Monte-Carlo-Simulation bestimmen möchte. Bei der Auswahl von Monte-Carlo-Simulation wird die gewünschte Anzahl simulierter Preispfade angegeben. 3. Der Nutzer wählt, wie die Monte-Carlo-Simulation parallelisiert werden soll. Zur Auswahl stehen CUDA und Multithreading. Bei der Auswahl von Multithreading wird die Anzahl zu verwendende Threads angegeben. 4. Der Nutzer wählt, ob er die Volatilität selbst eingeben oder automatisch schätzen lassen möchte. Im zweiten Fall hat der Nutzer die Auswahl zwischen der historischen Volatilität, bei der die Volatilität aufgrund der Laufzeit der Option bestimmt wird und der historischen Volatilität mit einer benutzerdefinierten Anzahl Tage zurück. 5. Der Nutzer bestimmt, ob er einen eigenen risikofreien Zinssatz eingeben, oder den risikofreien Zinssatz automatisch schätzen lassen möchte. 6. Sobald die Berechnungen beendet sind, kann der Nutzer das Resultat begutachten. Neben dem berechneten Preis der Option wird der risikofreie Zinssatz, sowie die Volatilität des Basiswertes dargestellt. Ausserdem sieht der Nutzer, wie lange die parallelisierte Berechnung gedauert hat und wie lange es insgesamt vom Absenden der Bewertungsanfrage bis zum Empfang des Resultats gedauert hat.
Frequency of Occurrence	Regelmässig

[5]Siehe: https://www.swissquote.ch

UC2: Bewertung amerikanischer Optionen

Tabelle 1.5: UC2: Bewertung amerikanischer Optionen

Merkmal	Beschreibung
Level	User-goal
Primary Actor	Technischer Nutzer, Praktischer Nutzer
Stakeholders and Interests	Der Nutzer möchte den Wert einer amerikanischen Option bestimmen.
Preconditions	-
Success Guarantee	Der Nutzer erhält alle relevanten Daten über die gewählte Option.
Main Success Scenario	1. Der Nutzer sucht gezielt nach einer Option oder gibt die geforderten Informationen manuell ein. 2. Der Nutzer lässt seine Resultate mittels Binomialmodell berechnen. Dazu wählt der Nutzer die Anzahl Zeitabschnitte. 3. Der Nutzer wählt, wie das Binomialmodell parallelisiert werden soll. Zur Auswahl stehen CUDA und Multithreading. Bei der Auswahl von Multithreading wird die Anzahl zu verwendende Threads angegeben. 4. Der Nutzer wählt, ob er die Volatilität selbst eingeben oder automatisch schätzen lassen möchte. Im zweiten Fall hat der Nutzer die Auswahl zwischen der historischen Volatilität, bei der die Volatilität aufgrund der Laufzeit der Option bestimmt wird, der historischen Volatilität mit einer benutzerdefinierten Anzahl Tage zurück und der impliziten Volatilität. Die implizite Volatilität kann ausschliesslich dann verwendet werden, wenn die Optionseingabe nicht manuell erfolgte. 5. Der Nutzer bestimmt, ob er einen eigenen risikofreien Zinssatz eingeben, oder den risikofreien Zinssatz automatisch schätzen lassen möchte. 6. Sobald die Berechnungen beendet sind, kann der Nutzer das Resultat begutachten. Neben dem berechneten Preis der Option wird der risikofreie Zinssatz, sowie die Volatilität des Basiswertes angezeigt. Ausserdem sieht der Nutzer, wie lange die parallelisierte Berechnung gedauert hat und wie lange es insgesamt vom Absenden der Bewertungsanfrage bis zum Empfang des Resultats gedauert hat.
Frequency of Occurrence	Regelmässig

UC3: Verwaltung von Bewertungsanfragen

Tabelle 1.6: UC3: Verwaltung von Bewertungsanfragen

Merkmal	**Beschreibung**
Level	User-goal
Primary Actor	Technischer Nutzer
Stakeholders and Interests	Der technische Nutzer möchte Bewertungsanfragen speichern und laden können.
Preconditions	Der technische Nutzer muss mit einem Account angemeldet sein.
Success Guarantee	Der technische Nutzer kann Bewertungsanfragen speichern und an einem späteren Zeitpunkt wieder aufrufen.
Main Success Scenario	1. Der Nutzer speichert eine Bewertungsanfrage ab. 2. Dem Nutzer steht eine Übersicht aller gespeicherten Bewertungsanfragen zur Verfügung. 3. Der Nutzer kann Bewertungsanfragen laden um erneut eine Optionsbewertung mit den gleichen Einstellungen durchführen zu lassen.
Extensions	4. Der Nutzer kann Bewertungsanfragen löschen.
Frequency of Occurrence	Regelmässig

UC4: Verwaltung von Bewertungsresultaten

Tabelle 1.7: UC4: Verwaltung von Bewertungsresultaten

Merkmal	**Beschreibung**
Level	User-goal
Primary Actor	Technischer Nutzer
Stakeholders and Interests	Der technische Nutzer möchte Bewertungsresultate speichern und ansehen können.
Preconditions	Der technische Nutzer muss mit einem Account angemeldet sein.
Success Guarantee	Der technische Nutzer kann Bewertungsresultate speichern und zu einem späteren Zeitpunkt wieder aufrufen.
Main Success Scenario	1. Der Nutzer speichert ein erhaltenes Bewertungsresultat ab. 2. Dem Nutzer steht eine Übersicht aller gespeicherten Bewertungsresultate zur Verfügung. 3. Der Nutzer kann die Details eines gespeicherten Bewertungsresultates aufrufen.
Extensions	4. Der Nutzer kann Bewertungsresultate löschen.
Frequency of Occurrence	Regelmässig

UC5: Identifizierung von gewinnbringenden Optionen

Tabelle 1.8: UC5: Identifizierung von gewinnbringenden Optionen

Merkmal	**Beschreibung**
Level	User-goal
Primary Actor	Praktischer Nutzer
Stakeholders and Interests	Der praktische Nutzer möchte eine Liste aller gewinnbringenden Optionen für eine Aktie erhalten.
Preconditions	-
Success Guarantee	Der praktische Nutzer erhält eine Liste aller gewinnbringenden Optionen für eine Aktie.
Main Success Scenario	1. Der Nutzer sucht im "Profit-Spotter" nach einer Aktie. 2. Der Nutzer wählt, ob die Bewertung mit einem Binomialbaum mit 31 oder 63 Zeitabschnitten erfolgen soll. 3. Der Nutzer wählt, wie das Multibinomialmodell parallelisiert werden soll. Zur Auswahl stehen CUDA und Multithreading. Bei der Auswahl von Multithreading wird die Anzahl zu verwendende Threads angegeben. 4. Der Nutzer wählt, ob er die Volatilität selbst eingeben oder automatisch schätzen lassen möchte. Im zweiten Fall hat der Nutzer die Auswahl zwischen der historischen Volatilität, bei der die Volatilität aufgrund der Laufzeit der Option bestimmt wird, der historischen Volatilität mit einer benutzerdefinierten Anzahl Tage zurück und der impliziten Volatilität. 5. Der Nutzer bestimmt, ob er einen eigenen risikofreien Zinssatz eingeben, oder den risikofreien Zinssatz automatisch schätzen lassen möchte. 6. Der Nutzer erhält eine Tabelle, die alle Optionen von Yahoo Finance zu der eingegeben Aktie enthält. In dieser Tabelle ist der mit dem Binomialmodell ermittelte Preis, der Marktpreis gemäss Yahoo Finance, sowie die Differenz zwischen diesen Preisen sichtbar. Der Nutzer kann die Tabelle sortieren. Ausserdem wird dem Nutzer angezeigt, wie lange die parallelisierte Berechnung gedauert hat und wie lange es insgesamt vom Absenden der Anfrage bis zum Erhalt des Resultats gedauert hat.
Frequency of Occurrence	Regelmässig

Stakeholders

In Abbildung 1.9 sind die Stakeholder ersichtlich.

Tabelle 1.9: Stakeholder

Stakeholder	
Entwickler	Die Entwickler sind verantwortlich für die rechtzeitige Lieferung der Software.
OST	Die OST stellt den SHPC0003 Cluster sowie einen vServer für das Projekt zur Verfügung.

1.3.2 Nicht-funktionale Anforderungen

In diesem Abschnitt werden die Qualitätsanforderungen an StOP gemäss ISO/IEC 25010 spezifiziert [15].

Usability

Tabelle 1.10: NFR: Usability

Nr	Subkategorie	Beschreibung
01	Learnability	Das User-Interface der Applikation muss selbsterklärend sein. Es ist daher keine externe Anleitung nötig, um die Applikation zu bedienen.
02	User Error Protection	Nutzer können die Applikation nicht in einen fehlerhaften Zustand versetzen. Falls ungültige Eingaben getätigt werden, lehnt die Applikation die Eingaben ab und informiert den Nutzer über die Fehler.

Reliability

Tabelle 1.11: NFR: Reliability

Nr	Subkategorie	Beschreibung
03	Fault Tolerance	Falls einzelne Dienste zur Optionsbewertung ausfallen, muss die Applikation trotzdem mit den noch verfügbaren Verfahren zur Optionsbewertung funktionieren.
04	Recoverability	Ein Absturz des Systems darf nicht zum Verlust von persistenten Daten führen.

Security

Tabelle 1.12: NFR: Security

Nr	Subkategorie	Beschreibung
05	Authenticity	Es dürfen keine Passwörter im Klartext gespeichert werden. Die Passwörter müssen mit einem Salt gehashed werden.
06	Authenticity	Passwörter müssen mindestens 8 Zeichen lang sein.

Maintainability

Tabelle 1.13: NFR: Maintainability

Nr	Subkategorie	Beschreibung
07	Modularity	Die einzelnen Verfahren zur Bestimmung des Wertes einer Option können auf unterschiedlicher Hardware unabhängig voneinander deployt werden.
08	Analysability	Alle Schnittstellen zwischen den Microservices müssen detailliert nach einem verbreiteten Standard dokumentiert werden.
09	Analysability	Der Webserver muss sämtliche Anfragen in einem standardisierten Format (bspw. Common Log Format) loggen.
10	Modifiability	Die Texte für das User-Interface müssen so spezifiziert werden, dass in Zukunft mit geringem Aufwand zusätzliche Sprachen unterstützt werden können.
11	Testability	Jeder implementierte Anwendungsfall muss im Rahmen eines Systemtestes verifiziert werden.

Portability

Tabelle 1.14: NFR: Portability

Nr	Subkategorie	Beschreibung
12	Adaptability	StOP ist Kompatibel mit Chrome Version 99 und Mozilla Firefox Version 97.
13	Installability	StOP kann innerhalb eines Arbeitstages installiert werden.

Kapitel 2

Lösungskonzept

Die im vorherigen Kapitel durchgeführte Analyse der Problemstellung ermöglichte die Erarbeitung eines Lösungskonzeptes. Geeignete Technologien wurden durch die Technologieevaluation ermittelt. Es wurde entschieden, das Backend in mehrere Microservices aufzuteilen, wobei die Benutzerverwaltung, Kommunikation mit Yahoo Finance, Persistenz und weitere Funktionen in einem API-Gateway, das auf dem NestJS Framework basiert, untergebracht wurden. Zur Kommunikation zwischen den Backend-Diensten kommt ein RabbitMQ Message Broker zum Einsatz. Es wurde ein Microservice für die Optionsbewertung mittels Multithreading und für die Optionsbewertung mittels CUDA vorgesehen. Das Frontend kommuniziert mit der NestJS REST API über REST.

Im Anschluss an die Technologieevaluation wurde die Softwarearchitektur erstellt. Diese zeigt den inneren Aufbau der einzelnen Microservices, die Interaktion der Microservices untereinander, sowie das Datenmodell. Für die durch Multithreading und CUDA parallelisierten Algorithmen wurde jeweils eine separate Bibliothek vorgesehen, sodass sie auch in anderen Applikationen verwendet werden können. Im Designkonzept sind die wichtigsten Grundsätze für die Gestaltung der Benutzeroberfläche festgehalten.

Um eine solide Basis für die Implementierung der Monte-Carlo-Simulation und des Binomialmodells zu schaffen, wurden für beide Modelle Pseudocode-Algorithmen, die die serielle Implementierung zeigen, erstellt. Ausgehend von diesen Pseudocode-Algorithmen wurde pro Modell jeweils eine serielle und mehrere parallelisierte Varianten implementiert. Für das Multibinomialmodell wurde auf die Erstellung eines Pseudocode-Algorithmus verzichtet, da ein solcher Algorithmus seriell durch die Anwendung des Binomialmodells für jede einzelne Option implementiert werden kann. Für jedes Modell wurde mindestens eine Variante, die Multithreading nutzt und mindestens eine Variante, die die GPU nutzt erstellt.

2.1 Evaluation der Technologien

In diesem Abschnitt werden geeignete Technologien zur Umsetzung der Bachelorarbeit ermittelt. Obschon jeder Microservices einen vollständig unterschiedlichen Technologiestack haben könnte, wird das Ziel, die Anzahl verwendeter Programmiersprachen möglichst gering zu halten, verfolgt, da jede Programmiersprache mit einem eigenen Ökosystem und einer eigenen Toolchain verbunden ist.

Damit StOP schnell und komfortabel installiert werden kann, werden sämtliche Microservices als Docker-Container[1] ausgeliefert.

2.1.1 Frontend Microservice

Da das Frontend umfangreiche Logik, wie beispielsweise die Validierung der Formulare, enthält, bietet sich die Entwicklung einer Single-Page-Webanwendung an. Die populärsten Frameworks dafür sind React[2], Vue.js[3] und Angular[4] [16]. Angular kommt nicht in Frage, da es eher für umfangreiche Frontend-Applikationen geeignet ist und unser Frontend überschaubar ausfallen wird. Da das Projektteam mit Vue.js keine Erfahrung hat und React zudem weiter verbreitet ist, wird React als Frontend-Framework verwendet. Um ein ansprechendes Design zu erreichen, wird die Komponentenbibliothek MUI[5] eingesetzt.

2.1.2 API-Gateway Microservice

Das API-Gateway bietet dem Frontend eine REST-Schnittstelle an. Es leitet Multithreading und GPU Bewertungsanfragen an die entsprechenden Microservices weiter und ist unter anderem für die Input-Validierung, Benutzerverwaltung, Kommunikation mit Yahoo Finance und Persistenz zuständig. Das Projektteam hat sich dazu entschieden, vier Frameworks für die Implementierung des Backends zu evaluieren. Die Auswahl fiel auf die folgenden vier Frameworks, aufgrund deren Popularität und der Vertrautheit des Projektteams mit den darunterliegenden Programmiersprachen.

Django

Django[6] ist ein auf Python basierendes Webframework, dass dem Model View Controller Ansatz folgt. Es ist aber auch möglich eine REST-Schnittstelle zu implementieren. Django wurde 2005 veröffentlicht. Der letzte Release ist vom 01.05.2022.

Express.js

Express.js[7] ist ein minimales Webframework, dass auf Javascript basiert. Die Verwendung von Typescript ist allerdings auch möglich. Erstmals wurde Express.js im Jahr 2010 veröffentlicht. Der letzte Release ist vom 26.05.2019.

[1] Siehe: https://www.docker.com/
[2] Siehe: https://reactjs.org/
[3] Siehe: https://vuejs.org/
[4] Siehe: https://angular.io/
[5] Siehe: https://mui.com/
[6] Siehe: https://www.djangoproject.com/
[7] Siehe: https://expressjs.com/

Flask

Auch für Python existiert mit Flask[8] ein minimales Webframework. Es wurde erstmals 2010 veröffentlicht. Die aktuellste Version ist am 19.02.2022 erschienen.

NestJS

NestJS[9] ist ein auf Express.js basierendes Webframework. Da NestJS selbst in Typescript implementiert wurde, bietet es eine hervorragende Unterstützung dafür an.

Kriterienkatalog

Der Kriterienkatalog beschreibt die Kriterien, nach denen die Frameworks beurteilt werden. Die Bewertung erfolgt von 0 (sehr schlecht) bis 4 (ausgezeichnet).

K1 Dokumentation Eine ausführliche und umfassende Dokumentation reduziert die Entwicklungszeit und erhöht die Robustheit der entwickelten Software. Ausführliche Code-Beispiele helfen beim Verständnis des Frameworks.

K2 Community Eine grosse Nutzerbasis weist darauf hin, dass das Framework eine hohe Qualität aufweist, sonst würden die Entwickler wohl auf andere Technologien zurückgreifen. Ferner ist die Wahrscheinlichkeit bei Problemen Hilfe im Internet zu finden bei populären Frameworks grösser. Als Indikatoren für die Beliebtheit dienen die Anzahl Github Stars und Anzahl Fragen auf Stackoverflow.

K3 Testbarkeit Da automatisierte Unit-Tests sowie Integrationstests nicht-funktionale Anforderungen an die Applikation sind, muss das Framework die Erstellung von Tests unterstützen.

K4 Angemessenheit des Funktionsumfangs Einerseits muss das Framework alle benötigten Anforderungen wie beispielsweise Authentisierung oder Anbindung an eine Datenbank unterstützen, andererseits sollte das Framework nicht mit unnötigen Funktionen überladen sein.

K5 Vertrautheit Da der Zeitrahmen für die Bachelorarbeit begrenzt ist und das Projektteam sich unter anderem bereits in C++ einarbeiten muss, wird ein Webframework mit kleiner Einarbeitungszeit gesucht.

Gewichtung Die Tabelle 2.1 beschreibt die Gewichtung der einzelnen Kriterien.

Tabelle 2.1: Gewichtung der Kriterien

Kriterium	Gewichtung
K1 Dokumentation	4
K2 Community	2
K3 Testbarkeit	4
K4 Angemessenheit des Funktionsumfanges	6
K5 Vertrautheit	4

[8] Siehe: https://flask.palletsprojects.com/
[9] Siehe: https://nestjs.com/

Bewertung

In diesem Abschnitt werden die Frameworks nach den definierten Kriterien bewertet. Die Tabellen 2.2 bis 2.6 zeigen, wie viele Punkte die Frameworks nach den verschiedenen Kriterien erreicht haben.

Tabelle 2.2: Bewertung K1 Dokumentation

Framework	Einschätzung	Punktzahl
Django	Die Dokumentation von Django ist ausgezeichnet und beinhaltet zahlreiche Code-Beispiele.	3
Express.js	Die Dokumentation von Express.js ist ausgezeichnet und beinhaltet zahlreiche Code-Beispiele. Daneben existiert ein Github Repository, dass jeweils eine vollständige Express.js Applikation zeigt, in der ein spezifisches Feature genutzt wird.	4
Flask	Die Dokumentation von Flask ist ausgezeichnet und beinhaltet zahlreiche Code-Beispiele.	3
NestJS	Die Dokumentation von NestJS ist ausgezeichnet. Neben einer ausführlichen Erläuterung der Konzepte sind in der Dokumentation zahlreiche Code-Beispiele zu finden. Daneben existiert ein Github Repository, dass jeweils eine vollständige NestJS Applikation zeigt, in der ein spezifisches Feature genutzt wird.	4

Tabelle 2.3: Bewertung K2 Community

Framework	Einschätzung	Punktzahl
Django	Django hat 62'900 Github Stars. Auf Stackoverflow existieren 286'571 Fragen mit dem Tag "django".	4
Express.js	Express.js hat 56'300 Github Stars. Auf Stackoverflow existieren 84'186 Fragen mit dem Tag "express".	3
Flask	Flak hat 58'300 Github Stars. Auf Stackoverflow existieren 49'237 Fragen mit dem Tag "flask".	3
NestJS	NestJS hat 45'300 Github Stars. Auf Stackoverflow existieren 7'164 Fragen mit dem Tag "nestjs".	1

Tabelle 2.4: Bewertung K3 Testbarkeit

Framework	Einschätzung	Punktzahl
Django	Das Framework stellt Mechanismen zur Erstellung von Unit- sowie Integrationstests zur Verfügung.	4
Express.js	Express.js stellt keine Unterstützung für das Testing zur Verfügung.	0
Flask	Das Framework stellt Mechanismen zur Erstellung von Unit- sowie Integrationstests zur Verfügung.	4
NestJS	Das Framework stellt Mechanismen zur Erstellung von Unit- sowie Integrationstests zur Verfügung.	4

Tabelle 2.5: Bewertung K4 Angemessenheit des Funktionsumfangs

Framework	Einschätzung	Punktzahl
Django	Django erlaubt die Modularisierung einer Applikation in Packages. Ausserdem bietet es eine sehr umfangreiche API, die unter anderem Authentifizierung, Logging und den Zugriff auf die Datenbank unterstützt. Ferner stehen zahlreiche, von der Community zur Verfügung gestellte, Packages zur Verfügung. Obschon Django ein Model View Controller Framework ist, können damit auch REST-Schnittstellen entwickelt werden. Da die anderen evaluierten Frameworks offener sind, mit welcher Frontend Technologie gearbeitet wird, wird Django in dieser Kategorie schlecht bewertet.	1
Express.js	Express.js stellt nur minimale Funktionen wie einen Router zur Verfügung. Werden zusätzliche Funktionalitäten benötigt, muss manuell eine passende Bibliothek gefunden und in die Applikation integriert werden.	2
Flask	Flask stellt nur minimale Funktionen wie einen Router zur Verfügung und wurde so entworfen, dass es mittels Extensions erweiterbar ist. Es existieren zahlreiche Extensions wie beispielsweise für die Authentifizierung oder für das einfachere Erstellen einer REST-Schnittstelle.	3
NestJS	NestJS ist modular aufgebaut. Dem Anwender ist es möglich eigene Module zu erstellen, es stehen jedoch auch zahlreiche offizielle und inoffizielle, fertige Module zur Verfügung, die beispielsweise das Handling der Authentifizierung vereinfachen. Der Funktionsumfang von NestJS kann also präzise den Anforderungen angepasst werden.	4

Tabelle 2.6: Bewertung K5 Vertrautheit

Framework	Einschätzung	Punktzahl
Django	Beide Projektmitglieder haben bereits im Machine Learning Bereich mit Python gearbeitet, aber niemand hat Erfahrung mit Django.	1
Express.js	Beide Projektmitglieder haben bereits im Engineering Projekt und im Modul “Web Engineering 2” mit Express.js gearbeitet.	4
Flask	Beide Projektmitglieder haben bereits im Machine Learning Bereich mit Python gearbeitet, aber niemand hat Erfahrung mit Flask.	1
NestJS	Beide Projektmitglieder sind mit Typescript vertraut. Ein Projektmitglied hat für die Studienarbeit bereits mit NestJS gearbeitet, während das andere Projektmitglied keine Erfahrung mit NestJS hat.	3

Resultat

Tabelle 2.7 zeigt die erreichten Punkte pro Framework. Da NestJS in Summe am meisten Punkte erreicht hat, wird das Backend mit NestJS implementiert.

Tabelle 2.7: Auswertung der Technologieevaluation für das API-Gateway

Kriterium (Gewichtung)	Django	Express.js	Flask	NestJS
K1 (4)	4*3=12	4*4=16	4*3=12	4*4=16
K2 (2)	2*4=8	2*3=6	2*3=6	2*1=2
K3 (4)	4*4=16	4*0=0	4*4=16	4*4=16
K4 (6)	6*1=6	6*2=12	6*3=18	6*4=24
K5 (4)	4*1=4	4*4=16	4*1=4	4*3=12
Total	**46**	**50**	**56**	**70**

2.1.3 Datenbank

Für die Persistierung der Accountdaten, Bewertungsanfragen und Bewertungsresultate wird eine PostgreSQL[10] Datenbank verwendet. PostgreSQL ist Open Source, hat ein umfangreiches Ökosystem, eine gute Integration in NestJS und ist dem Projektteam aus früheren Arbeiten bekannt.

2.1.4 Multithreading Microservice

Die C++ Standardbibliothek unterstützt Multithreading, allerdings wird kein Thread Pool zur Verfügung gestellt. Deshalb wird der Thread Pool von Shoshany [17], welcher mit der MIT Lizenz verfügbar ist, verwendet. Es wird C++20 verwendet, da diese Version des Standards Synchronisationsprimitiven (Latch, Barriere) enthält, die in früheren Versionen nicht verfügbar sind [18].

2.1.5 GPU Microservice

Von der OST werden NVIDIA Grafikkarten zur Verfügung gestellt, was die Verwendung von CUDA nahelegt. CUDA bietet eine API für C++ und Fortran an. Ferner existieren Wrapper für Java und Python. Auf die Verwendung der Wrapper wird verzichtet, da keine ausführliche, offizielle Dokumentationen von NVIDIA existieren und die Kernel trotzdem in C++ geschrieben werden müssten. Für C++ und Fortran steht jeweils eine ausführliche Dokumention von NVIDIA zur Verfügung. Da mehr Ressourcen im Internet zu CUDA C++ als zu CUDA Fortran zu finden sind und das Projektteam keinerlei Erfahrung mit Fortran hat, wird CUDA C++ verwendet. Die neuste Version des C++-Standards, die von NVCC unterstützt wird, ist C++17 [18].

2.1.6 C++ Build-Tools

Um ein möglichst effizientes Arbeiten zu ermöglichen, werden in allen auf C++ basierenden Microservices die gleichen Build Tools eingesetzt. Als Build-System wird CMake[11] verwendet, da es das Builden auf mehren Plattformen (unter anderem Linux und Windows) unterstützt. Auf Windows wird MSVC eingesetzt, während auf Linux GCC verwendet wird.

[10] Siehe: https://www.postgresql.org/
[11] Siehe: https://cmake.org/

2.1.7 Message Broker

Die Backend-Dienste werden über einen Message Broker kommunizieren. Vom Experten wurde die Verwendung von RabbitMQ vorgeschlagen. RabbitMQ hat die Vorteile, dass es erstens Open Source ist und zweitens AMQP verwendet, welches ebenfalls Open Source ist. Ferner weist RabbitMQ eine relativ hohe Verbreitung auf und es existieren zahlreiche Client-Bibliotheken[12] dafür. Als Client-Bibliothek wird der RabbitMQ C AMQP Client[13] verwendet, da dieser eine schnelle Entwicklung ermöglichen sollte.

2.1.8 Cache

Da das Frontend nicht an den Message Broker angebunden ist, muss die NestJS REST API die Antworten der Option Pricer zwischenspeichern, bis sie vom Frontend abgefragt werden. Neben den Bewertungsresultaten muss auch der Kontext der Anfrage (z.B. beim "Profit-Spotter" die von der Yahoo Finance API heruntergeladenen Optionseigenschaften) gespeichert werden. Grundsätzlich könnten diese Informationen in der NestJS REST API in einem selbst-implementierten Cache-Service, der beispielsweise auf einer Map basiert, gespeichert werden. Dies würde die NestJS REST API allerdings "stateful" machen, was nicht erwünscht ist. Deshalb wird eine externe In-Memory Datenbank verwendet. Das Projektteam hat sich für Redis[14] entschieden da es Open Source, sehr verbreitet, leichtgewichtig und einfach zu bedienen ist.

[12] Siehe: https://www.rabbitmq.com/devtools.html
[13] Siehe: https://github.com/alanxz/rabbitmq-c
[14] Siehe: https://redis.io/

2.2 Softwarearchitektur

In diesem Kapitel wird die Architektur der StOP Software detailliert beschrieben. Zu Beginn wird mit der Bausteinansicht die Struktur dokumentiert. Anschliessend wird die Organisastion des Deployments erörtert.

2.2.1 Bausteinansicht

Die Bausteinansicht wurde gemäss Arc42 [19] beschrieben. Die Grundidee dahinter ist die Struktur der Software auf verschiedenen Abstraktionsstufen darzustellen. Die niedrigen Ebenen verschaffen einen groben Überblick über das System, während der Detailgrad mit steigender Ebene zunimmt.

Ebene 0: Kontextabgrenzung

Die Whitebox Kontextabgrenzung in Abbildung 2.1 zeigt den Aktor "User", der mit StOP den Preis einer Option ermitteln möchte. Um den Preis einer Option zu berechnen, greift StOP auf Daten von Yahoo Finance zurück, auf die über die Yahoo Finance API zugegriffen werden kann.

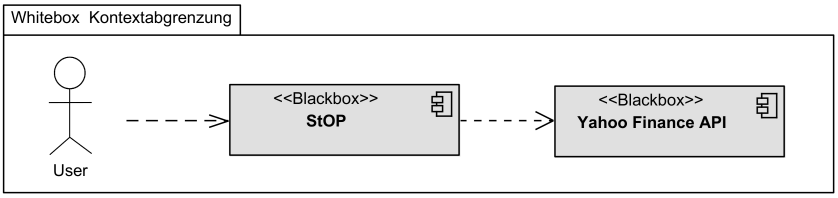


Abbildung 2.1: Whitebox Kontextabgrenzung

In der Tabelle 2.8 sind alle Blackboxes der Whitebox Kontextabgrenzung beschrieben.

Tabelle 2.8: Blackboxes der Kontextabgrenzung

Komponente	Verantwortung
StOP	Berechnung & Anzeigen der Werte von europäischen und amerikanischen Optionen
Yahoo Finance API	Stellt amerikanischen Optionsdaten sowie Daten zu Aktien zur Verfügung

Ebene 1

Die in Abbildung 2.2 zu sehende Whitebox-Ansicht zeigt, aus welchen Komponenten die StOP-Applikation besteht. Alle Blackboxes können unabhängig voneinander deployt werden und könnten somit als eigene Microservices agieren. Dies ist insbesondere von Vorteil, da die verschiedenen Option Pricer sehr unterschiedliche Hardwareanforderungen aufweisen. So benötigt der Multithreading Option Pricer Hardware mit einer möglichst grossen Anzahl Threads und der CUDA Option Pricer eine möglichst Leistungsstarke GPU.

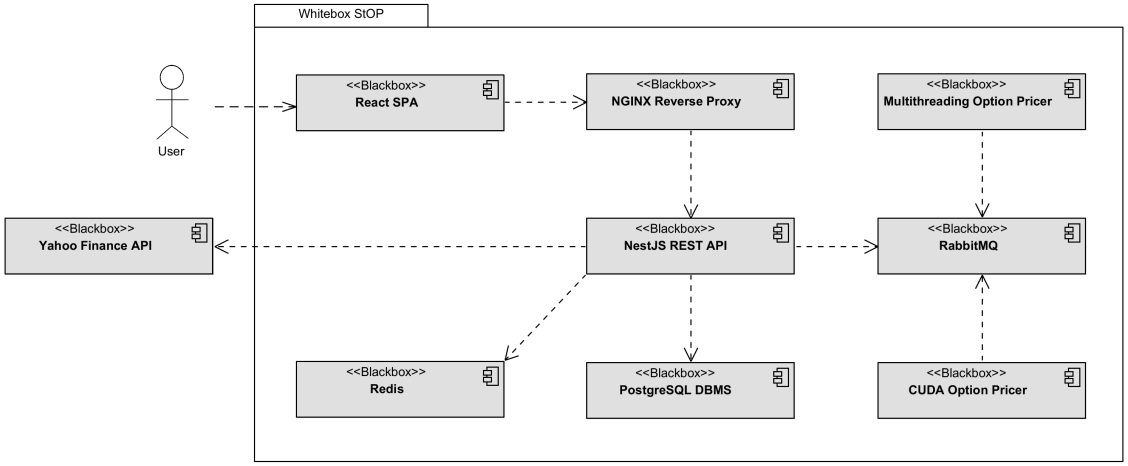


Abbildung 2.2: Whitebox StOP

Die Tabelle 2.9 bietet eine Übersicht über die Verantwortlichkeiten der in der StOP-Applikation vorhandenen Blackboxes.

Tabelle 2.9: Blackboxes der StOP-Applikation

Komponente	Verantwortung
React SPA	Die React SPA ist die Schnittstelle zwischen dem User und der StOP-Applikation. Sie nimmt Anfragen entgegen, leitet sie an die NestJS REST API weiter und stellt die empfangene Antwort in einer benutzerfreundlichen Form dar.
NestJS REST API	Die NestJS REST API ist für die Beantwortung von Anfragen der React SPA zuständig. Die NestJS REST API kommuniziert mit verschiedenen Diensten um alle Informationen zu sammeln, die zur Beantwortung einer Anfrage nötig sind. Für die Berechnung von Optionswerten sendet die NestJS REST API Nachrichten an RabbitMQ. Ausserdem konsumiert die NestJS REST API die Bewertungsresultate von RabbitMQ. Für Options- und Aktieninformationen steht eine Verbindung zur Yahoo Finance API zur Verfügung. Für die Persistierung von Nutzerdaten, Bewertungsanfragen und Bewertungsresultaten steht eine PostgreSQL Datenbank bereit. Kurzlebige Informationen werden Redis Cache abgelegt.
PostgreSQL DBMS	Das PostgreSQL DBMS ist für die persistente Speicherung der Nutzerdaten, Bewertungsanfragen und Bewertungsresultaten zuständig.
Redis	Daten wie die Volatilität oder der risikofreie Zinssatz einer Option, die die Option Pricer zwar zur Berechnung des Optionswertes benötigen, jedoch nicht mit dem Resultat zurückgeben, werden im Redis abgelegt, damit die NestJS REST API sie dem React SPA mit dem Resultat der Option Pricer zurückgeben kann.
RabbitMQ	Der RabbitMQ Message Broker agiert als Vermittler zwischen der NestJS REST API und dem CUDA sowie Multithreading Option Pricer. Er speichert Bewertungsanfragen und Resultate bis sie von interessierten Diensten abgeholt werden.
NGINX Reverse Proxy	NGINX wird als Reverse Proxy verwendet. Sämtliche Anfragen an StOP werden über den Reverse Proxy entweder an den Static File Server der React SPA oder an die NestJS REST API weitergeleitet.
Multithreading Option Pricer	Der Multithreading Option Pricer ist für die Berechnung von amerikanischen und europäischen Optionswerten mit einer konfigurierbaren Anzahl Threads zuständig.
CUDA Option Pricer	Der CUDA Option Pricer ist für die Berechnung von amerikanischen und europäischen Optionswerten unter Nutzung einer GPU zuständig.

Ebene 2

In der Ebene 2 werden die einzelnen Komponenten der StOP-Applikation als Whitebox betrachet.

Whitebox React SPA Der Aufbau der React SPA ist in Abbildung 2.3 ersichtlich.

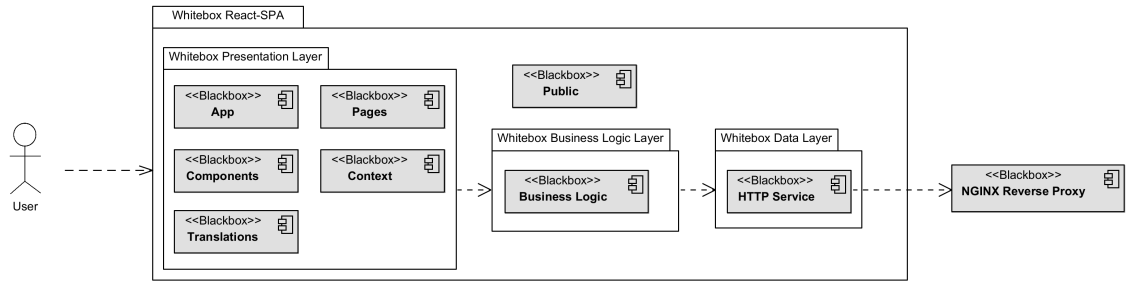


Abbildung 2.3: Whitebox React SPA

Die Tabelle 2.10 beschreibt die Blackboxes der React SPA.

Tabelle 2.10: Blackboxes der React SPA

Komponente	**Verantwortung**
Pages	Die Pages Blackbox enthält alle Pages, welche wiederum Komponenten enthalten.
Components	Enthält alle React-Komponenten.
Context	Zur Speicherung von globalen Variablen wie Nutzereingaben oder Resultaten.
Public	Die Public Blackbox umfasst statische Assets wie Bilder und Styles.
Translations	Dient der Speicherung der Übersetzungsdateien im JSON Format.
Business Logic	Diese Blackbox umfasst die Business Logik. Dazu gehört unter anderem die Messung der Zeit für die Beantwortung einer Optionbewertungsanfrage sowie die Formularvalidierung.
HTTP Service	Dient dem Absenden von HTTP Anfragen.

Whitebox NestJS REST API Eine Übersicht über den Aufbau der NestJS REST API bietet das Komponentendiagramm in Abbildung 2.4.

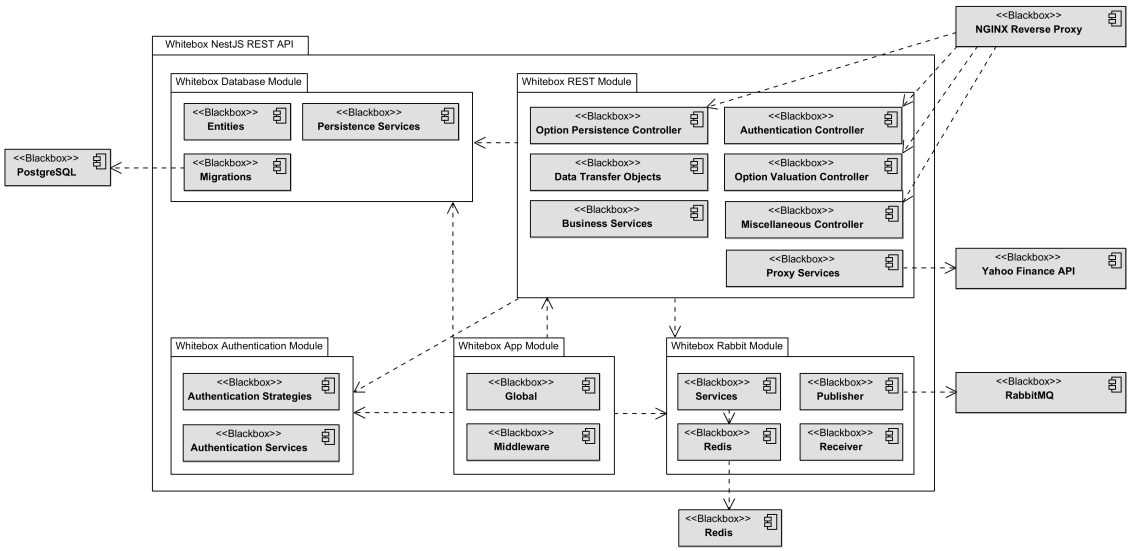


Abbildung 2.4: Whitebox NestJS REST API

Die Blackboxes der NestJS REST API können der Tabelle 2.11 entnommen werden.

Tabelle 2.11: Blackboxes der NestJS REST API

Komponente	Verantwortung
Authentication Strategies	Mittels Passport.js können verschiedene Authentfizierungsstrategien implementiert werden. In diesem Projekt wird die LocalStrategy sowie die JWTStrategy verwendet.
Authentication Service	Der Authentication Service wird von der LocalStrategy genutzt um mit den Persistence Services zu kommunizieren. Ausserdem bietet er weitere Services wie das Hashen von Passwörtern an.
Global	Beinhaltet globale Konstanten und den Taskservice, welcher veraltete Optionsbewertungs-Jobs oder unreferenzierte Optionen in der Datenbank löscht.
Persistence Services	Nutzt die TypeORM Repositories um Datenbankabfragen zu ermöglichen.
Migrations	Die Kommunikation zwischen der NestJS REST API mit dem PostgreSQL DBMS findet über TypeORM statt.
Entities	Das Datenbankschema wird über Entities mittels Annotationen generiert.
Authentication Controller	Der Authentication Controller bearbeitet HTTP-Anfragen, die die Registrierung, Anmeldung und Abmeldung betreffen.
Option Valuation Controller	Der Option Valuation Controller bearbeitet alle Anfragen, die die Bewertung von Optionen betreffen.
Option Persistence Controller	Der Option Persistence Controller bearbeitet alle Anfragen, die die Persistierung von Bewertungsanfragen und Bewertungsresultaten betreffen.
Miscellaneous Controller	Der Miscellaneous Controller beantwortet alle Anfragen betreffend der Hardware-Eigenschaften.
Data Transfer Objects	Für jede REST-Route existiert ein Data Transfer Object über das automatisch eine Anfragevalidierung durchgeführt wird.
Business Services	Die Business Services enthalten die Domänenlogik. Insbesondere enthält es die Logik zur Berechnung der analytischen Lösung der Black-Scholes-Gleichung sowie die Logik zur Schätzung der Volatilität und dem risikofreien Zinssatz.
Proxy Services	Die Proxy Services kapseln den Zugriff auf externe Services wie die Yahoo Finance API. Für jeden externen Microservice existiert ein dezidierter Proxy Service.
Rabbit-Services	Services für RabbitMQ Receiver & Publisher.
Rabbit-Publisher	Ist für das Senden von Bewertungsanfragen an die Option Pricer zuständig.
Rabbit-Receiver	Ist für das Empfangen von Bewertungsresultaten & Hardwareinformationen der Option Pricer zuständig.
Redis	Stellt die Verbindung zum Redis Cache sicher und legt dort wieder benötigte Daten ab.

Für das Hashing der Passwörter wird die bcrypt Hashfunktion mit einem Kostenfaktor von 10 verwendet.

Whitebox PostgreSQL DBMS Da der Zugriff auf das PostgreSQL DBMS über TypeORM[15] erfolgt, wird das Datenbankschema aus den Entities im Source Code automatisch generiert. Das Speichermodell als Entity-Relationship-Diagramm ist in Abbildung 2.5 zu sehen.

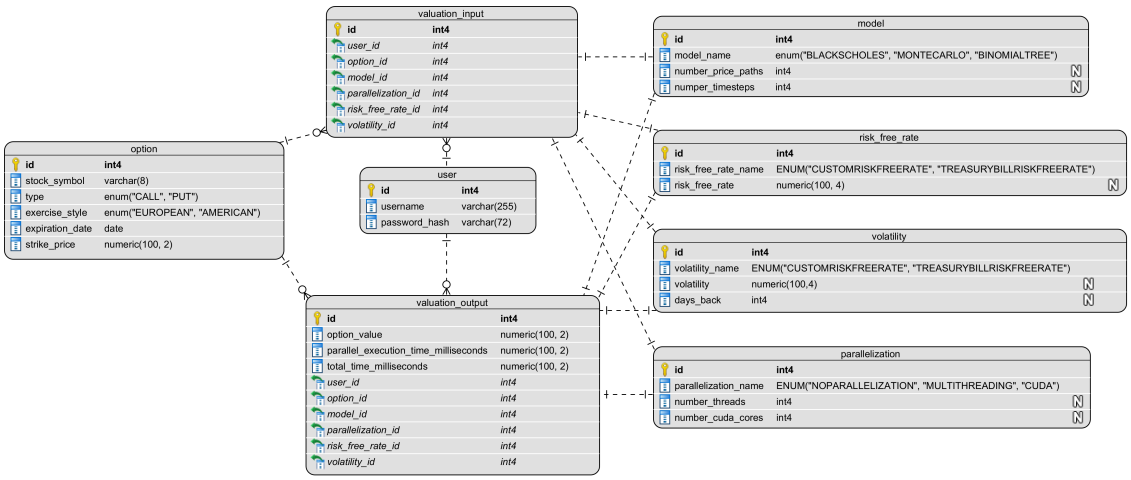


Abbildung 2.5: Entity-Relationship-Modell

Whitebox CUDA Option Pricer Der CUDA Option Pricer in ein Executable worker und eine Bibliothek lib-cuda-option-pricing unterteilt, wobei sämtlicher Programmcode, der in CUDA C++ geschrieben wurde in der statischen Bibliothek lib-cuda-option-pricing enthalten ist. Somit muss ausschliesslich die Bibliothek lib-cuda-option-pricing mit dem NVCC kompiliert werden. Weitere Informationen können der Abbildung 2.6 entnommen werden.

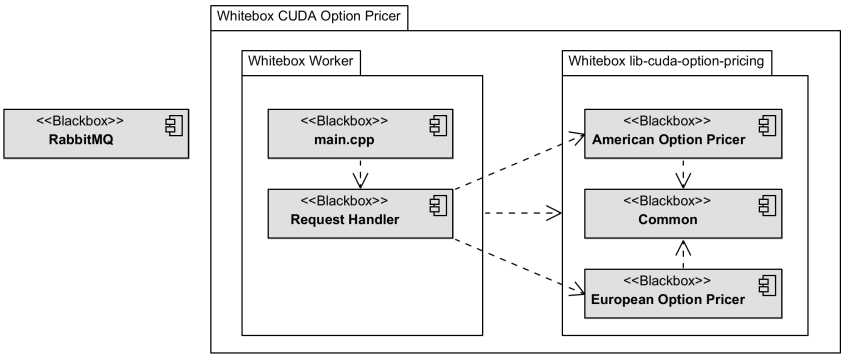


Abbildung 2.6: Whitebox CUDA Option Pricer

Eine Übersicht über die Blackboxes des CUDA Option Pricers bietet die Tabelle 2.12.

[15] Siehe: https://typeorm.io/

Tabelle 2.12: Blackboxes des CUDA Option Pricers

Komponente	Verantwortung
main.cpp	Nimmt Optionsbewertungsanfragen vom RabbitMQ entgegen und übergibt sie an den Request Handler
Request Handler	Der Request Handler ist für die Entscheidet aufgrund der Anfrage, welcher Option Pricer für die Berechnung verwendet wird und lässt die Option(en) berechnen.
Common	Enthält diverse Hilfsklassen für die Pricer.
Request Handler	Der Request Handler ist für die Entgegennahme und Weiterleitung der (Berechnungs-) Aufträge zuständig
European Option Pricer	Der European Option Pricer ist für die Berechnung der Werte von europäischen Optionen zuständig.
American Option Pricer	Der American Option Pricer ist für die Berechnung der Werte von amerikanischen Optionen zuständig.

Whitebox Multithreading Option Pricer Analog zum CUDA Option Pricer wird der Multithreading Option Pricer ebenfalls in ein Executable worker und eine Bibliothek lib-multi-option-pricing unterteilt. Der Multithreading Option Pricer folgt dem in Abbildung 2.7 beschriebenen Aufbau.

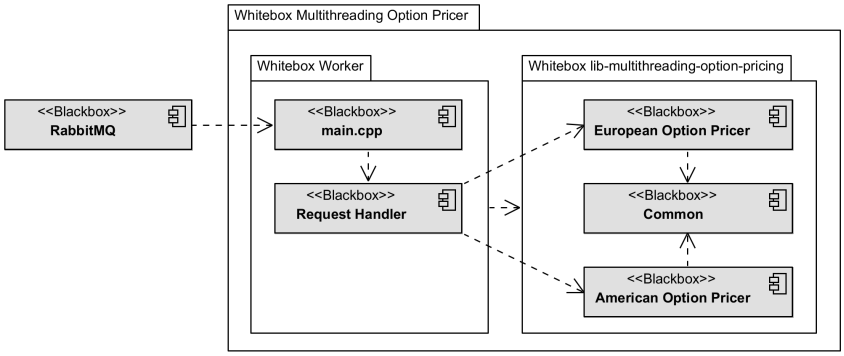


Abbildung 2.7: Whitebox Multithreading Option Pricer

Die Blackboxes des Multithreading Option Pricers können der Tabelle 2.13 entnommen werden.

Tabelle 2.13: Blackboxes des Multithreading Option Pricers

Komponente	Verantwortung
main.cpp	Nimmt Optionsbewertungsanfragen vom RabbitMQ entgegen und übergibt sie an den Request Handler
Request Handler	Der Request Handler parst den Body der Anfragen und ruft Funktionen der lib-option-pricing auf, um die Anfragen zu beantworten.
Common	Enthält diverse Hilfsklassen für die Pricer.
European Option Pricer	Der European Option Pricer ist für die Berechnung der Werte von europäischen Optionen zuständig.
American Option Pricer	Der American Option Pricer ist für die Berechnung der Werte von amerikanischen Optionen zuständig.

Whitebox RabbitMQ Abbildung 2.8 zeigt den Aufbau des RabbitMQ Message Brokers. Es existieren drei Queues: In der `cuda` Queue sind alle Optionsbewertungsanfragen an den CUDA Option Pricer enthalten. Die Queue `multithreading` enthält die Optionsbewertungsanfragen an den Multithreading Option Pricer. Die CUDA und Multithreading Option Pricer übermitteln ihre Antworten in die `response` Queue. Die `global-exchange` verteilt ankommende Nachrichten anhand des Nachrichtenschlüssels auf die Queues.

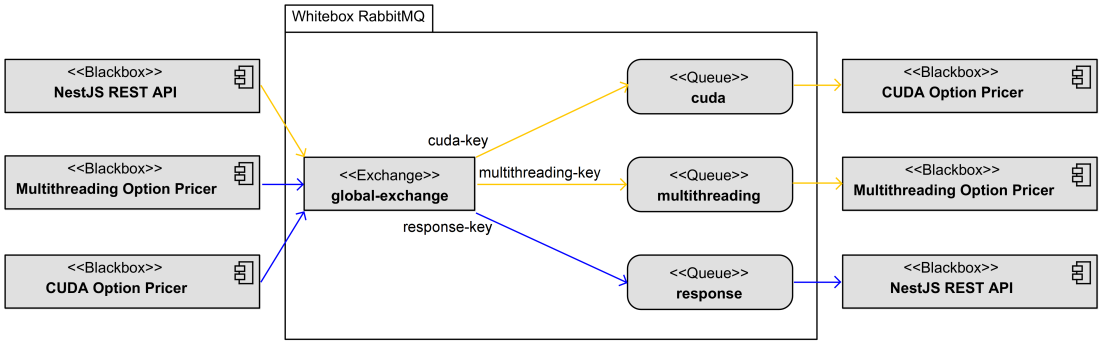


Abbildung 2.8: Whitebox RabbitMQ

2.2.2 Deployment

Dieser Abschnitt beschreibt das Deployment der StOP-Applikation.

Produktion

Eine grobe Übersicht über das produktive Deployment während der Projektdauer bietet das Deployment-Diagramm in Abbildung 2.9. Der NGINX Reverse Proxy vereinfacht die Handhabung von HTTPS. Ausschliesslich der NGINX Reverse Proxy muss Kenntnis vom SSL-Zertifikat haben, da die übrigen Microservices nicht gegen Aussen kommunizieren können diese eine unverschlüsselte Schnittstelle anbieten. Ausserdem werden CORS Fehler vermieden. Das HTTPS-Zertifikat wird von Let's Encrypt[16] bezogen und mittels CertBot[17] verwaltet.

[16] Siehe: https://letsencrypt.org/
[17] Siehe: https://certbot.eff.org/

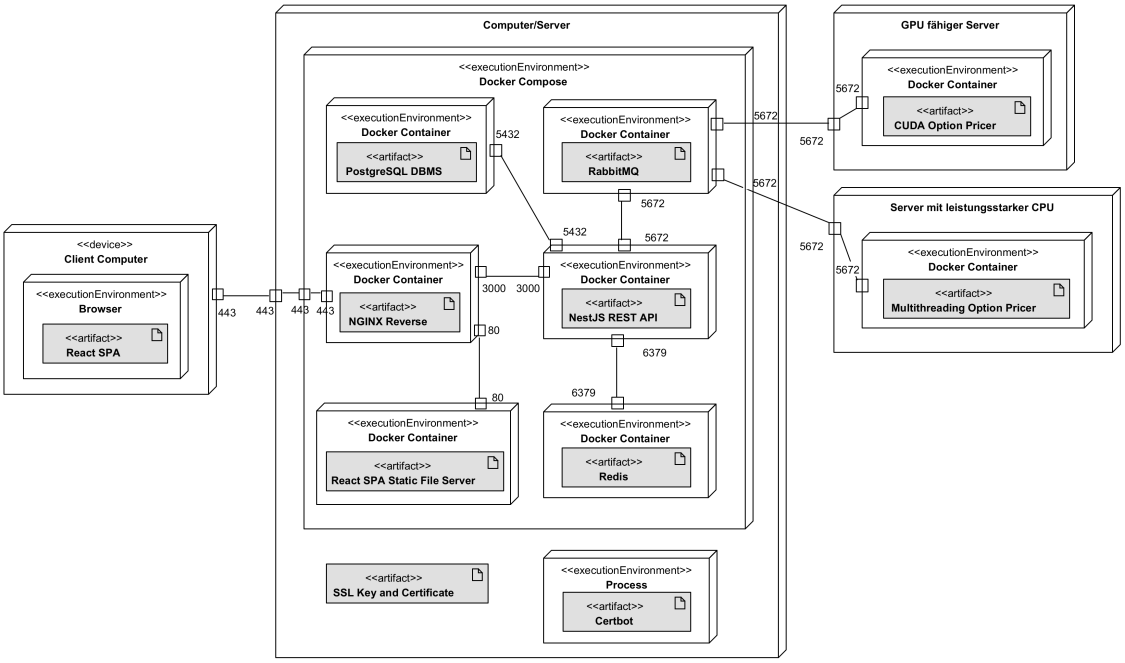


Abbildung 2.9: Deployment während der Entwicklung

Entwicklung

Während der Entwicklung sollte auf den Einsatz von HTTPS verzichtet werden. Das Deployment während der Entwicklung ist sehr individuell, basierend auf der zur Verfügung stehenden Hardware. Aufgrund der Microservice-Architektur können die einzelnen Komponenten unabhängig voneinander entwickelt werden.

2.3 UI Design

2.3.1 Designdokumentation

Die Software wird als Desktopanwendung umgesetzt, da die Anwendung auf einem mobilen Gerät unpraktisch erscheint.

Farben

Das Farbdesign wird mit der 60/30/10 Methode[18] umgesetzt. Dabei wird versucht, die Grundfarbe zu 60% zu verwenden, die Sekundärfarbe zu 30% und zuletzt eine markante Farbe nur zu 10%.

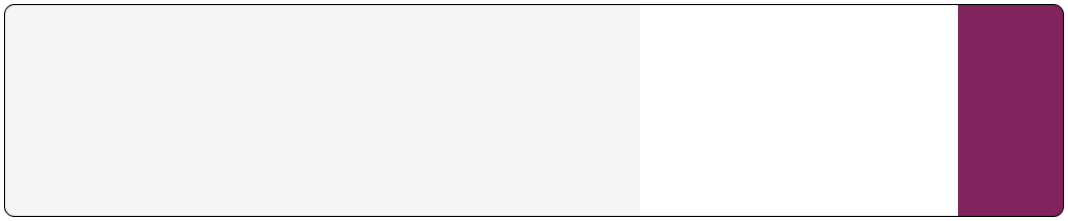


Abbildung 2.10: 60/30/10 Methode

- Grundfarbe: #F5F5F5
- Sekundärfarbe: #FFFFFF
- Drittfarbe: #81245D
- Text: #000000

Logo

Abbildung 2.11: StOP Logo

Layout

Zwischen einem Container und seinem Content existiert immer ein Margin/Padding.

[18] Siehe: https://www.saralynnbrennan.com/blog/the-60-30-10-design-rule

Design Library

Es wird Material UI[19] für React verwendet. Soweit möglich werden dessen vordefinierten Komponenten und Icons übernommen.

Font

Als Standardschriftart wird *Mulish* verwendet

Regular

Abbildung 2.12: StOP Schriftzug

Bold

Abbildung 2.13: StOP Schriftzug (Bold)

2.3.2 Wireframes

Die Wireframes wurden mit Figma[20] erstellt. Es steht eine interaktive Präsentation[21] bereit. Im Anhang Wireframes sind einige Ausschnitte zu sehen.

[19] Siehe: https://mui.com
[20] Siehe: https://www.figma.com
[21] Siehe: https://www.figma.com/proto/1h7TEtvBesuY2aJdyx7tpB/UI-Design

2.4 Entwurf der Optionsbewertungs-Algorithmen

In diesem Abschnitt werden die Algorithmen für die Monte-Carlo-Simulation und das Binomialmodell in Pseudocode beschrieben. Diese Beschreibungen stellte die Grundlage für die Implementierung in C++ dar.

2.4.1 Monte-Carlo-Simulation

Der Algorithmus 1 hat eine Laufzeit von $\mathcal{O}(n)$, wobei n die Anzahl simulierter Preispfade ist. Wenn P die Anzahl paralleler Einheiten ist, kann die Laufzeit durch Parallelisierung auf $\mathcal{O}(\frac{n}{P})$ reduziert werden.

Algorithm 1 Optionsbewertung mit Monte-Carlo-Simulation für europäische Optionen

Require: n {Anzahl Preispfade}
Require: S_0 {Basiswertpreis zum Bewertungszeitpunkt}
Require: σ {Volatilität}
Require: r {Risikofreier Zinssatz}
Require: T {Anzahl Tage in Jahren bis zur Fälligkeit der Option}
Require: K {Strike Preis}
Require: *type* {Call oder Put}

1: $sum \leftarrow 0$
2: **for** $i \leftarrow 0$ to n **do**
3: $\quad S_T \leftarrow S_0 e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}\mathcal{N}(0,1)}$
4: $\quad$ **if** *type* = Call **then**
5: $\quad\quad sum \leftarrow \max(S_T - K, 0)$
6: $\quad$ **else**
7: $\quad\quad sum \leftarrow \max(K - S_T, 0)$
8: $\quad$ **end if**
9: **end for**
10: **return** $e^{-rT} sum/n$

2.4.2 Binomialmodell

Der Algorithmus 2 hat eine Laufzeit von $\mathcal{O}(n^2)$. Durch Parallelisierung mit P parallelen Einheiten kann die Laufzeit auf $\mathcal{O}(\frac{n^2}{P})$ reduziert werden. Die in der Domänenanalyse im Abschnitt 1.1.3 ermittelten Formeln zur Berechnung der Knoten sind für die Implementierung suboptimal, da die Knoten auf der vertikalen von unten nach oben indexiert werden. Dieser Umstand würde die Implementierung komplexer und somit fehleranfälliger machen. Um dieser Problematik zu umgehen, wurden die Formeln im Pseudocode-Algorithmus so angepasst, dass die Indexierung auf der vertikalen von oben nach unten erfolgt.

Algorithm 2 Optionsbewertung mit Binomialmodell für amerikanische Optionen

Require: n {Anzahl Zeitabschnitte}
Require: S_0 {Basiswertpreis zum Bewertungszeitpunkt}
Require: σ {Volatilität}
Require: r {Risikofreier Zinssatz}
Require: T {Anzahl Tage in Jahren bis zur Fälligkeit der Option}
Require: K {Strike Preis}
Require: *type* {Call oder Put}

$\Delta t \leftarrow T/n$
$d \leftarrow e^{-\sigma\sqrt{\Delta t}}$
$u \leftarrow e^{\sigma\sqrt{\Delta t}}$
$p \leftarrow \frac{e^{r\Delta t}-d}{u-d}$
C $\leftarrow$ leere Matrix $\in \mathbb{R}^{n+1\times n+1}$
for $i \leftarrow n \geq 0$ **do**
  for $j \leftarrow 0 \leq i$ **do**
    if $i = n$ **then**
      if *type* = Call **then**
        $C_{j,i} \leftarrow \max(S_0 u^{i-j} d^j - K, 0)$
      else
        $C_{j,i} \leftarrow \max(K - S_0 u^{i-j} d^j, 0,)$
      end if
    else
      if *type* = Call **then**
        $Cj, i \leftarrow \max(\max(S_0 u^{i-j} d^j - K, 0), e^{-r\Delta t}(pC_{j,i+1} + (1-p)C_{j+1,i+1}))$
      else
        $Cj, i \leftarrow \max(\max(K - S_0 u^{i-j} d^j, 0), e^{-r\Delta t}(pC_{j,i+1} + (1-p)C_{j+1,i+1}))$
      end if
    end if
  end for
end for
return $C_{0,0}$

2.4.3 Multibinomialmodell

Das Multibinomialmodell wird für die Batch-Verarbeitung von amerikanischen Optionen verwendet. Da bei der seriellen Implementierung lediglich jede Option mit dem Binomialmodell bewertet wird, wurde auf die Erstellung eines Pseudocode-Algorithmus verzichtet.

2.5 Implementierung der Optionsbewertungs-Algorithmen

Dieser Abschnitt beschreibt die Implementierung der parallelisierten Monte-Carlo-Simulation, Binomialmodell und Multibinomialmodell Varianten. Die Erläuterungen werden durch Abbildungen ergänzt.

2.5.1 CUDA

Die im Kapitel Entwurf der Optionsbewertungs-Algorithmen beschriebenen Algorithmen wurden jeweils mehrmals implementiert um verschiedene Ansätze der Parallelisierung zu testen. In den nachfolgenden Abschnitten wird auf die Details der Implementierung mit CUDA eingegangen.

Bei den Berechnungen auf der GPU wurde lediglich mit einfacher Genauigkeit (float) gerechnet, da die meisten NVIDIA GPUs eine viel grössere Anzahl an FP32 Kernen als FP64 Kerne beinhalten [20]. Die verschiedenen Varianten verwenden unterschiedliche Zufallsgeneratoren, da diese jeweils unterschiedlich schnell sind.

Monte-Carlo-Simulation

Zur Generierung von standardnormalverteilten Zahlen wurde die von NVIDIA entwickelten Bibliotheken cuRAND und Thrust verwendet, mit welcher die Zahlen direkt auf der GPU generiert werden können.

Variante Host API Von cuRAND werden zwei APIs zur Generierung von Zufallszahlen zur Verfügung gestellt. Zum einen die Host API, welche es vom Host-Code aus ermöglicht ein Array an Zufallszahlen zu generieren. In dieser Variante wird der `MTGP32` aus der Mersenne Twister Familie[22] als Zufallsgenerator verwendet.

Variante Device API Die von cuRAND zur Verfügung gestellte Device API ermöglicht die Generierung von Zufallszahlen im Device-Code mit dem `XORWOW`[23] Zufallsgenerator.

Variante Thrust In dieser Variante wird der Kongruenzgenerator[24] von Thrust verwendet, um eine Sequenz von Zufallszahlen zu generieren.

Variante Unified Memory Für diese Variante wurde mit Unified Memory gearbeitet. Die Generierung der Zufallszahlen erfolgte gleich wie bei der Variante Host API. Der einzige Unterschied besteht darin, dass der Speicher nicht manuell verwaltet wird [21].

Binomialmodell

Die Implementierung des Binomialbaums gestaltet sich komplexer, da die Threads nach der Berechnung eines Zeitabschnitts mittels einer Barriere, die durch `__syncthreads()` realisiert wird, synchronisiert werden müssen. Diese ist allerdings nur für die Threads eines Blocks gültig. Da für n Zeitabschnitte $n+1$ Threads benötigt werden ist die Anzahl Zeitschritte, die in einem Block berechnet werden können auf die Anzahl Threads pro Block - 1 begrenzt. Auf gegenwärtigen NVIDIA GPUs beinhaltet ein Block typischerweise bis zu

[22] Siehe: https://docs.nvidia.com/cuda/curand/host-api-overview.html
[23] Siehe: https://docs.nvidia.com/cuda/curand/device-api-overview.html
[24] Siehe: https://bit.ly/3b4vlzU

1'024 Threads [20]. In der Domänenanalyse wurde bereits erwähnt, dass für das Binomialmodell etwa 30 Zeitabschnitte ausreichen, um für die Praxis genügend genaue Resultate zu erhalten. Ein Block reicht deshalb komfortabel aus, um genügend präzise Resultate zu erzielen, weshalb auf die Implementierung einer Variante mit mehreren Blöcken verzichtet wurde. Die Matrizen zur Speicherung der Binomialbäume wurden in allen Varianten linearisiert, als 1D-Array, gespeichert.

Variante Global Memory Bei dieser Variante wird die Matrix, die den Binomialbaum speichert, vollständig im globalen Speicher abgelegt. Der Speicherverbrauch ist mit `sizeof(float)*(n+1)*(n+1)` gegeben, da der Binomialbaum konzeptionell durch eine quadratische Matrix repräsentiert wird.

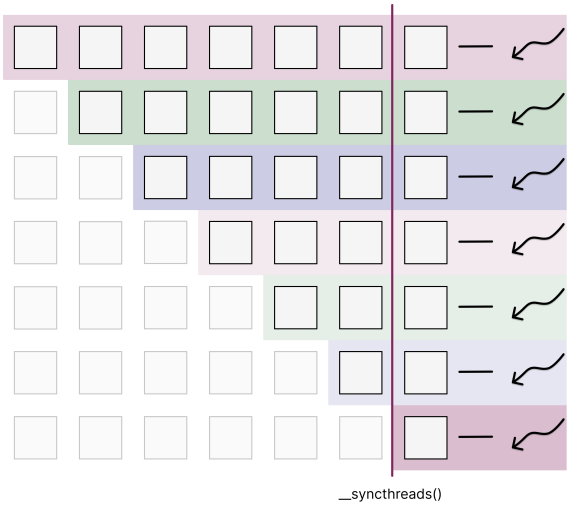


Abbildung 2.14: CUDA - Variante Global Memory

Wie in Abbildung 2.14 farblich dargestellt, berechnet jeder Thread alle Knoten auf seiner "Ebene", was dazu führt, dass mit jedem berechneten Zeitschritt ein weiterer Thread keine Knoten mehr zu berechnen hat.

Variante Shared Memory Diese Variante macht Gebrauch vom Shared Memory. Da das Shared Memory einen begrenzten Speicherplatz aufweist, kann nicht mehr der vollständige Binomialbaum gespeichert werden. Da in unserem Anwendungsfall nur das Schlussresultat relevant ist, können die Zwischenschritte in der Matrix jeweils überschrieben werden. In diesem Fall ist der Speicherverbrauch mit `sizeof(float)*(n+1)*2` gegeben.

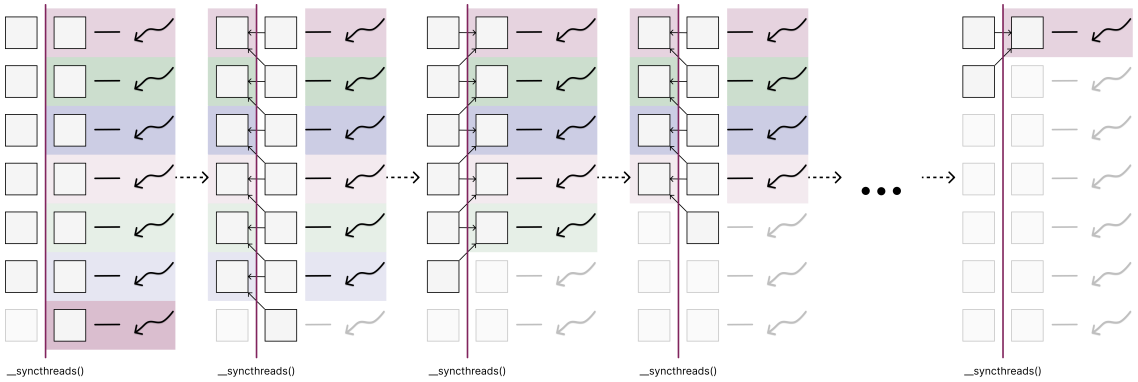


Abbildung 2.15: CUDA - Variante Shared Memory

In der Abbildung 2.15 erkennt man, wie bereits in der Variante Global Memory, wie jeder berechnete Zeitschritt dazu führt, dass jeweils ein Thread keine Aufträge mehr erhält. Es besteht die Problematik, dass der Optionswert in der Wurzel des Baumes entweder in der linken oder rechten Spalte der Matrix steht. Dies ist abhängig davon, ob der Binomialbaum eine gerade oder ungerade Grösse hat. Wenn n gerade ist, befindet sich das Endresultat in der linken, ansonsten in der rechten Spalte.

Multibinomialmodell

Für die Berechnung von mehreren amerikanischen Optionen wurden ebenfalls zwei Varianten basierend auf Global und Shared Memory implementiert. Bei der Variante Global Memory wird der selbe Trick, dass lediglich zwei Spalten an Speicher alloziert werden, wie bei der Shared Memory Variante angewendet, womit der quadratisch steigende Speicherplatzbedarf umgangen wird. Dies ist nötig, da der Speicherbedarf sonst zu gross ist, wenn viele Optionen gleichzeitig bewertet werden. Ansonsten funktionieren die beiden Varianten identisch.

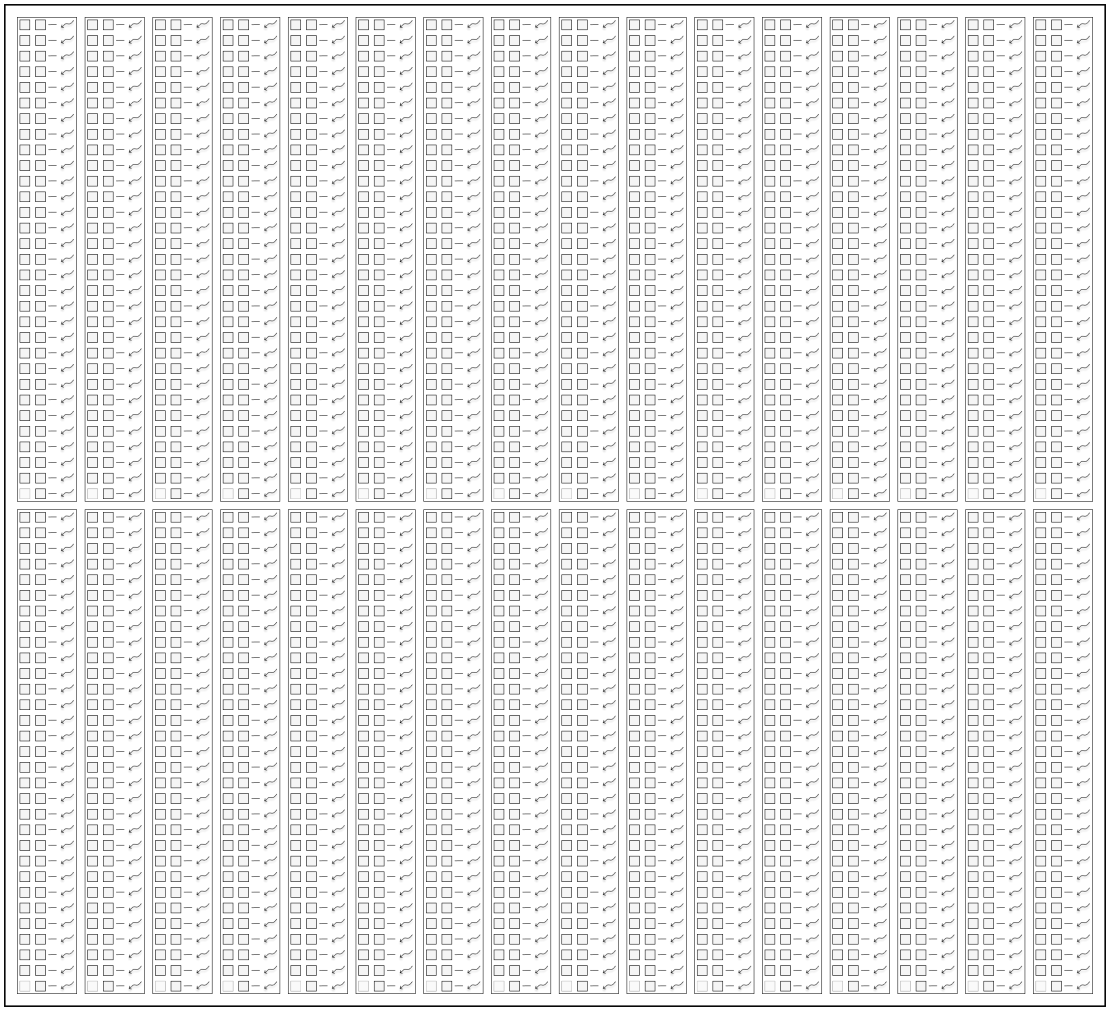


Abbildung 2.16: CUDA - Multibinomialbaum

Beim Multibinomialmodell werden nur Binomialbäume der Grösse $x^2 - 1$ akzeptiert. Diese Einschränkung erlaubt eine überlappungsfreie Aufteilung von mehreren Optionen auf die CUDA-Blöcke. Als Beispiel wie in Abbildung 2.16 dargestellt, stehen 1'024 Threads pro Block zur Verfügung. Wenn jede Option mit 31 $(5^2 - 1)$ Zeitabschnitten berechnet werden soll, benötigt man für jede Option lediglich 32 Threads. Somit können von einem Block parallel 32 Optionen berechnet werden. Der Grund, wieso eine überlappungsfreie Aufteilung in die Blöcke angestrebt wurde, war, dass man über die Blöcke hinaus die Threads nicht mit `__syncthreads()` synchronisieren kann.

2.5.2 Multithreading

In den nachfolgenden Abschnitten wird auf die Details der Implementierung mit Multithreading eingegangen.

Obschon die CPU problemlos mit doppelter Genauigkeit (double) umgehen könnte, wurden alle Algorithmen zur Gewährleistung der Vergleichbarkeit mit der CUDA Implementierung mit einfacher Genauigkeit (float) implementiert.

Monte Carlo Simulation

Variante Singlethreaded Zufallszahlen-Generierung Bei dieser Variante werden die Zufallszahlen mit einem Mersenne Twister Generator aus der C++-Standardbibliothek generiert. Die Generierung der Zufallszahlen erfolgt auf einem Thread. Nachdem die Zufallszahlen generiert wurden, werden die verschiedenen Preispfade berechnet. Nachdem alle Threads die Preispfade berechnet haben, erfolgt die Summierung und Durchschnittsbildung.

Variante Multithreaded Zufallszahlen-Generierung Bei der zweiten Variante generiert jeder Thread seine eigenen Zufallszahlen mit einem individuellen Seed. Es hat sich gezeigt, dass die Zufallszahlen eine höhere Qualität aufweisen, wenn sie von nur einem Thread generiert werden, als wenn jeder Thread seine eigenen Zufallszahlen mit eigenem Seed generiert. Ansonsten ist diese Variante identisch mit der Variante Singlethreaded Zufallszahlen-Generierung.

Variante Vector Extensions Die dritte Variante wurde mit SIMD und somit Vektorisierung implementiert. Als Zufallsgenerator wurde der `Vmt19937_64` von EigenRand[25], der eine vektorisierte Variante des Mersenne Twister Generators ist, verwendet. Zuerst werden die Preispfade auf die Threads aufgeteilt. Jeder Thread generiert seine eigenen Zufallszahlen. Anschliessend werden die Preispfade mittels 512-Bit Vektorinstruktionen berechnet. So können pro Thread jeweils 16 Preispfade gleichzeitig berechnet werden. Die Summierung und Durchschnittsbildung ist identisch wie bei den anderen Multithreading Varianten.

Binomialmodell

Variante Thread Pool Für diese Variante wird ein Thread Pool benutzt. Der Vorteil liegt darin, dass erstellte Threads mehrmals verwendet werden können und nicht ständig neue erstellt werden müssen. In einem ersten Schritt werden die Blattknoten gleichmässig auf die vorhandenen Threads aufgeteilt. Der erste Thread berechnet zusätzlich die übrigen Knoten, sollte die Aufteilung nicht aufgehen.

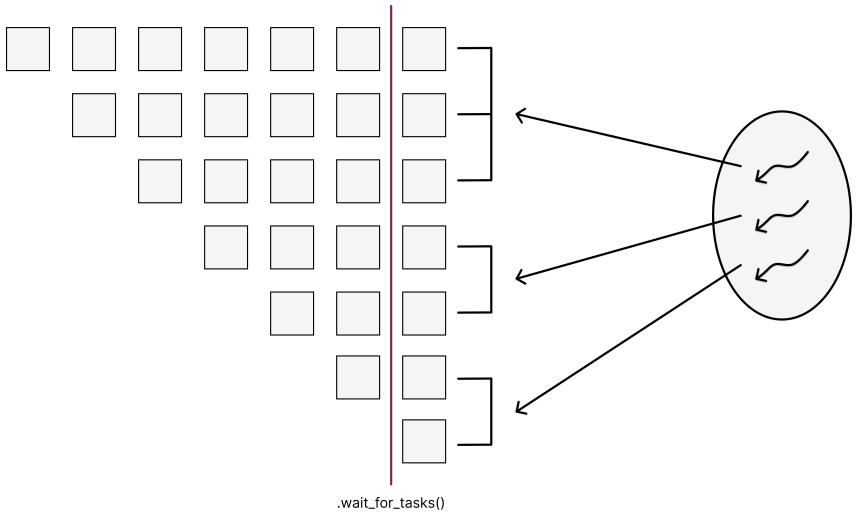


Abbildung 2.17: Multithreading - Variante Thread Pool (Schritt 1)

[25] Siehe: https://bab2min.github.io/eigenrand

Der Thread Pool wartet mit dem Befehl `wait_for_tasks()` bis alle Threads zurück in den Pool gelangen. Sobald alle Threads zurück sind und somit alle Knoten eines Zeitabschnitts berechnet sind werden die Knoten des nächsten Zeitschrittes wiederum gleichmässig auf die Threads verteilt. In der Abbildung (2.18) kann man einen Extremfall erkennen, in dem der erste Thread dreimal so viel Berechnungen zu erledigen hat, wie die anderen. Dieser Fall ist jedoch nur bei kleinen Anzahl Knoten so extrem und liegt immer im Bereich von 0 bis (t-1) zusätzlich zu berechnenden Knoten.

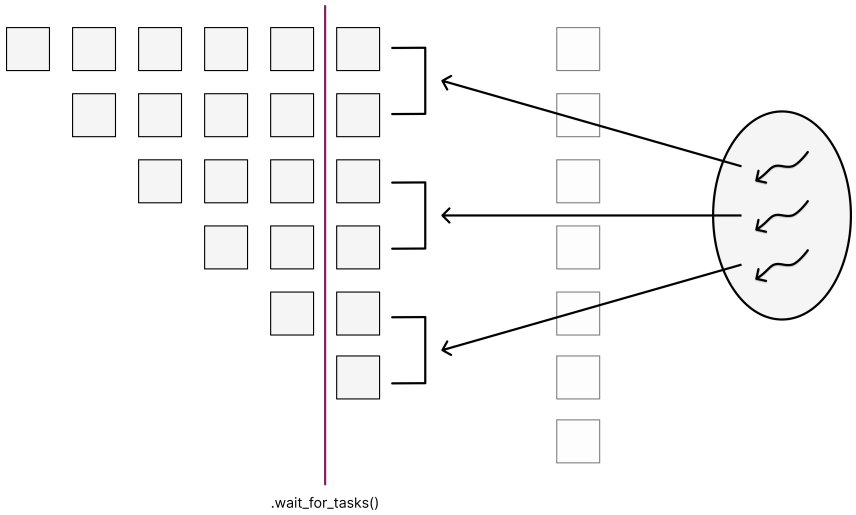


Abbildung 2.18: Multithreading - Variante Thread Pool (Schritt 2)

Diese Prozedur wiederholt sich, bis der Optionswert am Bewertungszeitpunkt berechnet wurde.

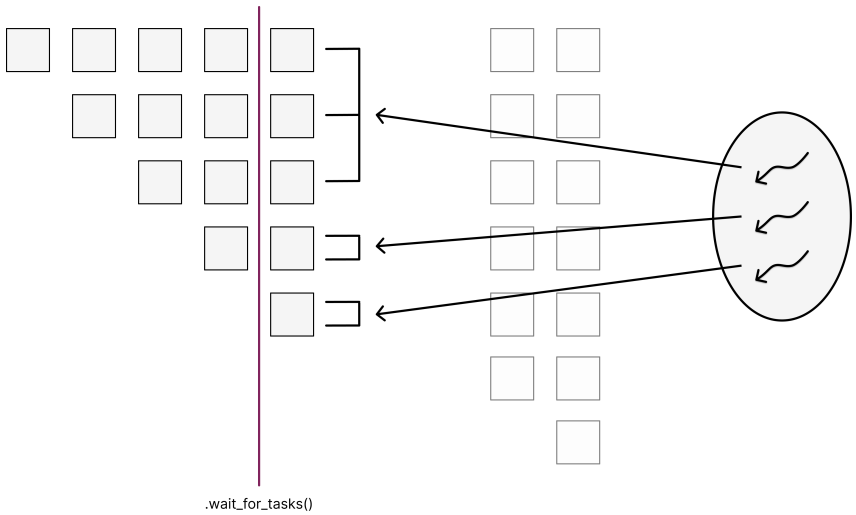


Abbildung 2.19: Multithreading - Variante Thread Pool (Schritt 3)

Variante Barriere In dieser Variante werden wie in der Variante mit Thread Pool ebenfalls nur zu Beginn die Anzahl gewünschter Threads erstellt. Bei jedem Zeitabschnitt gibt es hier aber eine Barriere aus der C++ Standardbibliothek, die die Threads aufeinander warten lässt und erst durchlässt, wenn alle angekommen sind. Die Zuteilung der Knoten auf die Threads bleibt gleich wie bei der Variante Thread Pool.

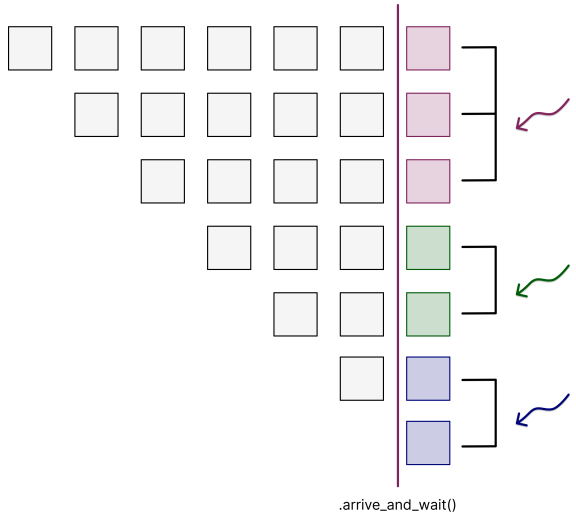


Abbildung 2.20: Multithreading - Variante Barriere (Schritt 1)

Wenn ein Thread alle seine Knoten berechnet hat, registriert er sich mit `arrive_and_wait()` bei der Barriere, um zu signalisieren, dass er bereit für den nächsten Zeitschritt ist. Damit ist der Thread solange blockiert, bis alle Threads bei der Barriere angekommen sind. Diese Prozedur wiederholt sich, bis man bei der Wurzel angelangt und der Optionswert definiert ist.

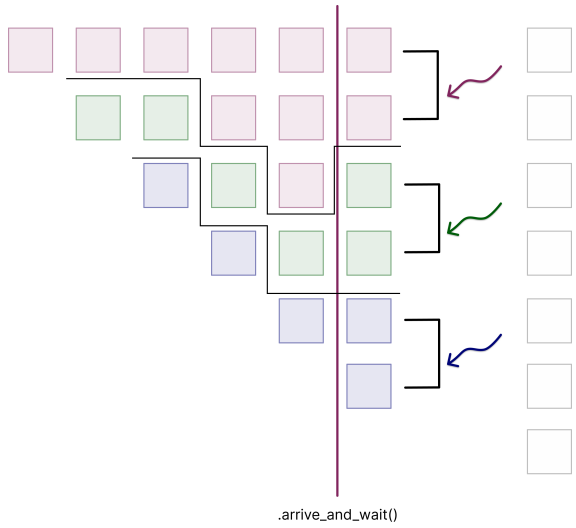


Abbildung 2.21: Multithreading - Variante Barriere (Schritt 2)

Sobald die Anzahl Knoten kleiner gleich der Anzahl Threads ist, muss mit jedem Zeitschritt die Anzahl Threads um eins reduziert werden. Dies geschieht dadurch, dass der nicht mehr benötigte Thread `arrive_and_drop()` und `return` aufruft.

Variante Atomaren Variablen Zu Beginn dieser Variante werden zwei gleich grosse Bäume erstellt. Der eine Baum ist für die effektive Berechnung der Optionswerte da, der Andere enthält bei jedem Knoten eine atomare "Boolean"-Variable. Die Knoten werden horizontal gruppiert und abwechslungsweise auf die vorhandenen Threads aufgeteilt. Nun können die Knoten Schrittweise (wie in 2.22 nummeriert) von den Threads berechnet werden.

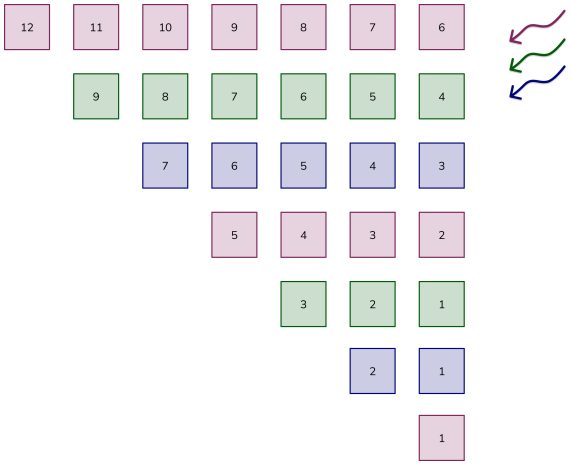


Abbildung 2.22: Multithreading - Variante Atomare Variablen

Sobald ein Knoten berechnet wurde, überschreibt der Thread im zweiten Baum den Knoten mit demselben Index mit "true". Dies ist der Indikator, dass der Knoten nun berechnet ist und die von diesem Knoten abhängigen Knoten damit berechnet werden können. Im Fall, dass der Knoten noch mit "false" gekennzeichnet ist, wartet der davon abhängige Thread in einer "while-Schlaufe", darauf, bis sich dies ändert. Die Variablen im zweiten Baum sind atomar, damit das Schreiben auf diese in einem einzelnen Schritt geschieht und somit Race Conditions vermieden werden können.

Multibinomialmodell

Für die Berechnung von mehreren amerikanischen Optionen stehen zwei Ansätze zur Verfügung. Bei beiden wird ein Baum von jeweils einem und nicht mehreren Threads berechnet.

Variante Thread Pool In dieser Variante wird zu Beginn ein Thread Pool mit der gewünschten Anzahl Threads erstellt. Für jede zu berechnende Option wird dann ein Task erstellt, der von einem einzelnen Thread ausgeführt wird. Nachdem ein Thread eine Option bewertet hat und das Ergebnis zurückgegeben hat, kehrt er zum Thread Pool zurück, um sogleich einem neuen Task zugewiesen zu werden. Dies wird fortgeführt, bis alle Optionen bewertet sind.

Variante Self-Managed Threads Beim Ansatz mit selbstverwaltenden Threads wird zu Beginn die gewünschte Anzahl Threads erstellt, jedoch gibt es diesmal keinen Thread Pool. Allen Threads werden die zu berechnenden Optionen gleichmässig zugeteilt, wobei der Erste die restlichen Optionen zusätzlich erhält. Der erwartete Vorteil dieser Variante war, dass nicht für jede Option ein eigener Baum alloziert werden muss, sondern jeder Thread seinen Binomialbaum wiederverwenden kann. Daher muss nicht für jede Option der Speicher für den Binomialbaum alloziert und dealloziert werden. Jeder Thread muss dies nur einmal machen.

Kapitel 3

Validierung

In diesem Kapitel werden die getroffenen Massnahmen zur Validierung der Software beschrieben. Um zu überprüfen, ob die funktionalen und nicht-funktionalen Anforderungen erfüllt sind, wurden Systemtests durchgeführt. Bei diesen Systemtests wurde jeweils die Funktionalität der gesamten StOP-Applikation getestet - es wurde also auch die Zusammenarbeit zwischen den verschiedenen Microservices getestet.

Um sicherzustellen, dass die Optionsbewertungs-Algorithmen korrekt funktionieren, wurden Unit-Tests erstellt. Dabei wird der relative Fehler zur seriellen Implementierung ermittelt, so dass allfällige Fehler erkannt werden können. Die Korrektheit der seriellen Implementierungen wurde dadurch garantiert, dass verschiedene Optionen bewertet wurden und das Ergebnis mit dem Resultat von Optionsbewertungs-Tools aus dem Web verglichen wurde.

Zusätzlich wurden Benchmarks erstellt, um den Speedup der verschiedenen Varianten im Vergleich zur seriellen Implementierung zu ermitteln. Die Benchmarks für die CPU Varianten wurden auf einem Intel Core i7-11370H Prozessor und die Benchmarks für die GPU Varianten auf einer NVIDIA Tesla V100 PCIe durchgeführt. Für die Monte-Carlo-Simulation mit 10'000'000 Preispfaden wurde mit Multithreading und Vector Extensions ein Speedup von 40 erreicht, während auf der GPU ein Speedup von 179 erreicht wurde. Beim Binomialmodell mit n=1'023 wurde auf der CPU kein nennenswerter und auf der GPU ein Speedup von 9 erreicht. Beim Multibinomialmodell mit n=1'023 konnte ein Speedup von 4 auf der CPU und von 900 auf der GPU erreicht werden. Insgesamt konnte also bei allen Modellen ein signifikanter Speedup erreicht werden, wobei beim Binomialmodell der Geschwindigkeitsvorteil durch die Parallelisierung am geringsten ausfällt.

Am Ende des Kapitels werden die Code Metriken erläutert. Unter anderem sind darin die Anzahl Codezeilen pro Microservice sowie die Testabdeckung pro Microservice ersichtlich.

3.1 Test der Optionsbewertungs-Algorithmen

In diesem Kapitel wird die Validierung der parallelisierten Algorithmen, die im CUDA und Multithreading Option Pricer zum Einsatz kommen, beschrieben. Die Tests wurden mit der GoogleTest[1] Bibliothek erstellt. Zur Gewährleistung der Reproduzierbarkeit der Tests wurden von Yahoo Finance am 22.05.2022 alle 1'571 Optionen auf die Apple Aktie heruntergeladen und in der Datei `test-options.json` gespeichert. Ausserdem wurde eine serielle Implementierung der Monte-Carlo, Binomialmodell und Multibinomialmodell Algorithmen erstellt. Für jede parallelisierte Implementierung wurde eine Test Suite erstellt. In jeder Test-Suite wird für alle 1'571 Optionen der relative Fehler der parallelisierten Implementierung zur seriellen Implementierung bestimmt. Da der Optionswert null sein kann, musste, um Divisonen durch null zu verhindern, die Formel zur Bestimmung des relativen Fehlers minimal angepasst werden:

$$\Delta_{rel} = \frac{|R_S - R_P|}{R_S + 0.01} \tag{3.1}$$

Dabei ist R_S das Resultat der seriellen Implementierung und R_P das Resultat der parallelisierten Implementierung.

Damit sichergestellt ist, dass die seriellen Implementationen korrekt sind, wurden diese Implementation ebenfalls getestet. Da das Resultat des Monte-Carlo-Simulation mit steigendem n zur analytischen Lösung der Black-Scholes-Formel konvergiert, wurde das Resultat des seriellen Monte-Carlo Algorithmus mit dem Resultat der Black-Scholes-Formel mit einer Toleranz von 0.05 verglichen ($n = 100,000,000$). Für die Berechnung der Black-Scholes-Formel wurde goodcalculators.com[2] verwendet. Die serielle Binomialbaum-Implementierung wurde mit dem Resultat des Binomialbaum-Rechners von janroman.dhis.org[3] mit einer Toleranz von 0.01 verglichen. Beim seriellen Multibinomialmodell wurde getestet, ob alle Optionen in der Datei `test-options.json` das gleiche Resultat ergeben, wie wenn sie mit der seriellen Binomialmodell-Implementierung bewertet werden. Alle Test der Monte-Carlo-Simulation Varianten wurden mit $n = 200'000$ durchgeführt. Damit kann der Umstand, dass die verschiedenen Implementationen unterschiedliche Zufallsgeneratoren verwenden etwas abgeschwächt werden. Falls im parallelisierten Algorithmus der gleiche Zufallsgenerator wie im seriellen Algorithmus verwendet wird, wurde der gleiche Seed genutzt.

Bei den Binomialbaum-Implementation würde man grundsätzlich erwarten, dass die Ergebnisse der seriellen Implementierung mit den Ergebnissen der parallelisierten Implementierung exakt übereinstimmen, da es sich im Gegensatz zur Monte-Carlo-Simulation um einen deterministischen Algorithmus handelt. Bei den Varianten, die auf der GPU ausgeführt werden entsteht jedoch eine gewisse Abweichung. Diese ist jedoch im Kontext unserer Applikation vernachlässigbar.

Die Tests der Binomialbaum-Implementation wurden mit $n = 1'000$ durchgeführt. Die Test der Multibinomialbaum-Implementationen wurden mit $n = 63$ durchgeführt.

3.1.1 Monte-Carlo-Simulation

Bei der Monte-Carlo-Simulation ist damit zu rechnen, dass die Ergebnisse eine gewisse Abweichung aufweisen, da verschiedene Zufallsgeneratoren unterschiedliche Zufallszahlen generieren. Um zu bestätigen, dass die Abweichung auf unterschiedliche Zufallszahlen zurück

[1] Siehe: https://github.com/google/googletest
[2] Siehe: https://goodcalculators.com/black-scholes-calculator/
[3] Siehe: http://janroman.dhis.org/calc/Binomial2.php

geht, wurden mehrere Testdurchläufe mit verschiedenen n durchgeführt. Mit steigendem n sollten die generierten Zufallszahlen immer akkurater einer Standardnormalverteilung folgen, womit der Fehler, der durch unterschiedliche Zufallssequenzen entsteht reduziert werden sollte.

Singlethreaded Zufallszahlen-Generierung

Die parallel bestimmten Optionswerte stimmten exakt mit den seriell bestimmten Optionswerten überein.

Multithreaded Zufallszahlen-Generierung

Die parallel bestimmten Optionswerte stimmen nicht exakt mit den seriell bestimmten Optionswerten überein. Dies ist darauf zurückzuführen, dass diese Variante zwar den selben Zufallsgenerator wie die serielle Version verwendet, aber mehrere Instanzen davon erstellt, die offensichtlich mit unterschiedlichen Seeds initialisiert werden müssen (sonst würden alle Instanzen die selbe Zufallsfolge generieren).

Tabelle 3.1: Relative Abweichungen der Variante Multithreaded Zufallszahlen-Generierung

n	Relative Abweichung
1'000	0.07794
10'000	0.01706
100'000	0.01194
500'000	0.00508
1'000'000	0.00227

Vector Extensions

Wie bei der Variante Multithreaded Zufallszahlen-Generierung kommen mehrere Zufallsgeneratoren zum Einsatz, was bedeutet, dass mit einem Fehler zu rechnen ist, da mit unterschiedlichen Zufallszahlen gearbeitet wird.

Tabelle 3.2: Relative Abweichungen der Variante Vector Extensions

n	Relative Abweichung
1'000	0.101501
10'000	0.039972
100'000	0.012856
500'000	0.002782
1'000'000	0.001512

Device API Zufallszahlen-Generierung

Wie erwartet stimmen die Optionswerte nicht exakt überein. Analog zum Vorgehen bei der Variante Multithreaded Zufallszahlen-Generierung wurden verschiedene Werte für n getestet, um zu überprüfen, ob der Fehler mit steigendem n abnimmt. Der Fehler nimmt nicht ab, was daran liegen könnte, dass die Thread Ids als Seed verwendet werden, statt dass das Subsequenze-Feature der cuRAND Device API verwendet wird. Falls die Thread Id als Subsequenz verwendet wird, dauern die Berechnungen aber sehr viel länger, da der Rechenaufwand für grössere Subsequenzen stark steigt. Trotzdem ist der Umstand,

dass der Fehler nicht abnimmt, überraschend, da bei den Multithreading Monte-Carlo-Simulation Varianten auch jeder Thread ein Zufallsgenerator mit eigenem Seed verwendet und die NVIDIA cuRAND Dokumentation explizit vorschlägt pro Zufallsgenerator einen eigenen Seed zu verwenden [22]. Es wäre wohl sinnvoll, die Funktion anzupassen, so dass sich beispielsweise immer 1000 Threads einen Seed teilen und jeder Thread eine bei 0 beginnende Subsequenz wählt. So wäre garantiert, dass die Subsequenzen nicht zu grosse werden und gleichzeitig könnten die statistischen Eigenschaften der Zufallszahlen verbessert werden. Dieser Ansatz konnte aufgrund der fortgeschrittenen Zeit aber nicht mehr verfolgt werden.

Tabelle 3.3: Relative Abweichungen der Variante Device API Zufallszahlen-Generierung

n	Relative Abweichung
1'000	0.20986
10'000	0.0336997
100'000	0.0444611
500'000	0.0342327
1'000'000	0.0319309

Host API Zufallszahlen-Generierung

Wie erwartet stimmen die Optionswerte nicht exakt überein. Analog zum Vorgehen bei der Variante Multithreaded Zufallszahlen-Generierung wurden verschiedene Werte für n getestet, um zu überprüfen, ob der Fehler mit steigendem n abnimmt.

Tabelle 3.4: Relative Abweichungen der Variante Host API Zufallszahlen-Generierung

n	Relative Abweichung
1'000	0.138016
10'000	0.0384346
100'000	0.0142912
500'000	0.0034944
1'000'000	0.0031182

Unified Memory

Wie erwartet stimmen die Optionswerte nicht exakt überein. Analog zum Vorgehen bei der Variante Multithreaded Zufallszahlen-Generierung wurden verschiedene Werte für n getestet, um zu überprüfen, ob der Fehler mit steigendem n abnimmt.

Tabelle 3.5: Relative Abweichungen der Variante Unified Memory

n	Relative Abweichung
1'000	0.138016
10'000	0.0384346
100'000	0.0142912
500'000	0.0034944
1'000'000	0.0031182

Wie in der Tabelle 3.5 ersichtlich ist, stimmen die Abweichungen mit den Abweichungen der Variante Host API Zufallszahlen-Generierung überein, was zu erwarten ist, da lediglich die Speicherverwaltung unterschiedlich ist.

Thrust Zufallszahlen-Generierung

Da diese Variante einen Kongruenzgenerator verwendet und die serielle Variante den Mersenne-Twister Zufallsgenerator weichen die Resultate bei kleinen n erheblich voneinander ab.

Tabelle 3.6: Relative Abweichungen der Variante Thrust Zufallszahlen-Generierung

n	Relative Abweichung
1'000	2.42475
10'000	1.01977
100'000	0.0939201
500'000	0.0201311
1'000'000	0.0138122

3.1.2 Binomialmodell

Barriere

Die parallel bestimmten Optionswerte stimmten exakt mit den seriell bestimmten Optionswerten überein.

Atomare Variablen

Die parallel bestimmten Optionswerte stimmten exakt mit den seriell bestimmten Optionswerten überein.

Thread Pool

Die parallel bestimmten Optionswerte stimmten exakt mit den seriell bestimmten Optionswerten überein.

Global Memory

Die mittlere relative Abweichung beträgt 0.0000847943.

Shared Memory

Die mittlere relative Abweichung beträgt 0.0000847943.

3.1.3 Multibinomialmodell

Thread Pool

Die parallel bestimmten Optionswerte stimmten exakt mit den seriell bestimmten Optionswerten überein.

Self-Managed Threads

Die parallel bestimmten Optionswerte stimmten exakt mit den seriell bestimmten Optionswerten überein.

Global Memory

Die mittlere relative Abweichung beträgt 0.00000314267.

Shared Memory

Die mittlere relative Abweichung beträgt 0.00000314267.

3.2 Benchmarks der Optionsbewertungs-Algorithmen

In diesem Abschnitt werden die Benchmarks für die parallelisierten Algorithmen beschrieben. Damit die Benchmarks möglichst einfach zu reproduzieren sind, wurde die Google Benchmark[4] Bibliothek verwendet. Es wurden fünf Iterationen aller Benchmarks ausgeführt. Der Wert, der als Laufzeit angegeben wurde, ist das arithmetische Mittel der Laufzeit über alle Iterationen. Alle Laufzeiten sind in Millisekunden.

Als Testdaten für die Benchmarks der Monte-Carlo-Simulation und des Binomialmodells dienten die in Tabelle 3.7 ersichtlichen Daten.

Tabelle 3.7: Referenzdaten

Eigenschaft	**Wert**
Zeit bis zur Fälligkeit	1 Jahr
Basiswert	ABB
Typ	Call
Strike Preis	32
Preis des Basiswertes	35
Volatilität des Basiswertes	45%
Risikofreier Zinssatz	2%

Für das Multibinomialmodell wurde als Input die selben Optionen (`test-options.json`) verwendet, die bereits im Abschnitt Test der Optionsbewertungs-Algorithmen verwendet wurden.

Die Benchmarks für die Serielle und Multithreading Implementierungen wurden auf einem Intel Core i7-11370H Prozessor[5] auf Windows 11 durchgeführt. Als Compiler wurde MSVC mit der Version 19.30.30705 verwendet. Da der Prozessor vier Kerne mit Hyperthreading hat, wurden bei den Multithreading-Varianten jeweils eins bis acht Threads getestet.

Die CUDA Benchmarks wurden auf dem SHPC0003 Cluster der OST durchgeführt. Auf dem Cluster ist eine NVIDIA Tesla V100 PCIe[6] mit 16GB Grafikspeicher installiert. Die Benchmarks wurden in einem Docker-Container ausgeführt, was nach Xu et al. [23] keinen signifikanten Einfluss auf die Performance hat.

Um Fehler beim Kopieren der Resultate der Benchmarks zu vermeiden, wurde das Python-Skript `benchmark-processor.py` erstellt. Es parst den JSON-Output der Benchmarks in Pandas-Dataframes. Aus diesen Dataframes wurden die in diesem Abschnitt ersichtlichen Tabellen und Abbildungen generiert.

[4] Siehe: https://github.com/google/benchmark
[5] Siehe: https://intel.ly/3Hmqk1H
[6] Siehe: https://www.nvidia.com/de-de/data-center/tesla-v100/

3.2.1 Monte-Carlo-Simulation

Serielle Implementierung

Wie Tabelle 3.8 zeigt, weist die serielle Implementierung erwartungsgemäss ungefähr eine lineare Laufzeit zu n auf. Die Linearität ist in Abbildung 3.1 nicht auf den ersten Blick zu sehen, da die x-Achse logarithmisch ist.

Tabelle 3.8: Laufzeiten der seriellen Monte-Carlo-Simulation-Implementierung

n	Laufzeit (ms)
10^3	0.05
10^4	0.52
10^5	5.06
10^6	50.73
10^7	506.68

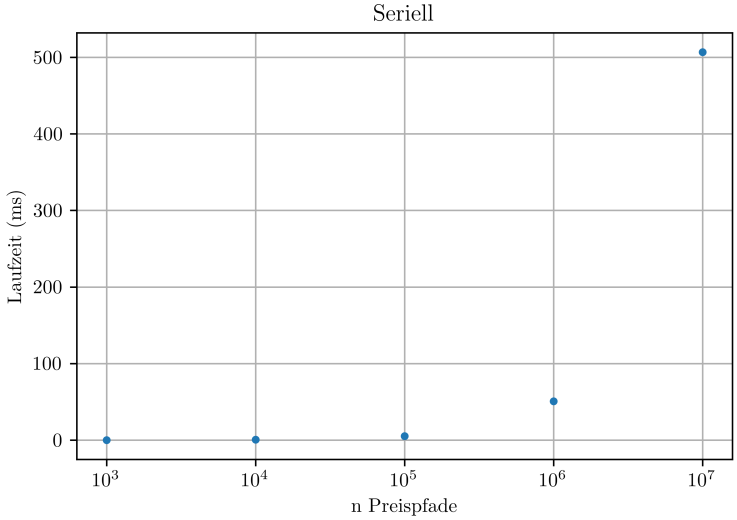


Abbildung 3.1: Laufzeiten der seriellen Monte-Carlo-Simulation-Implementierung

Multithreaded Zufallszahlen-Generierung

Tabelle 3.9 und Abbildung 3.2 zeigen, dass der Speedup bei hinreichend grossem n ungefähr der Anzahl verwendeter Threads entspricht, so lange die Anzahl Threads kleiner gleich der Anzahl echter Kerne ist.

Tabelle 3.9: Speedups der Variante Multithreaded Zufallszahlen-Generierung

n	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**	**t=6**	**t=7**	**t=8**
10^3	0.43	0.4	0.34	0.27	0.22	0.19	0.17	0.15
10^4	0.56	1.45	1.81	1.89	1.7	1.65	1.55	1.46
10^5	0.85	1.76	1.82	2.72	3.22	3.68	4.01	3.59
10^6	1.07	2.1	2.46	3.36	3.61	4.32	4.85	4.37
10^7	1.12	2.2	2.9	3.49	3.84	4.46	4.98	4.59

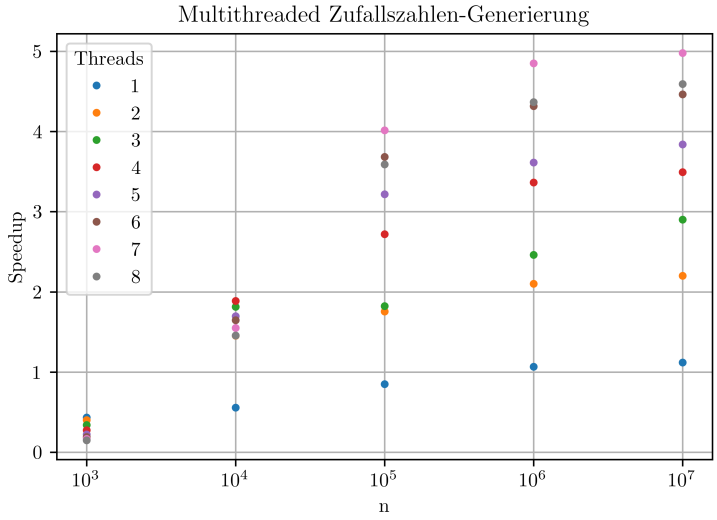


Abbildung 3.2: Speedups der Variante Multithreaded Zufallszahlen-Generierung

Singlethreaded Zufallszahlen-Generierung

Wie Tabelle 3.10 und Abbildung 3.3 zeigen, ist der Speedup auch bei mehr als einem Thread minimal. Dies ist vermutlich darauf zurückzuführen, dass der Speedup durch die Generierung der Zufallszahlen mit nur einem Thread begrenzt wird.

Tabelle 3.10: Speedups der Variante Singlethreaded Zufallszahlen-Generierung

n	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
10^3	0.41	0.31	0.26	0.22	0.19	0.17	0.15	0.13
10^4	0.51	0.78	0.78	0.79	0.75	0.73	0.69	0.65
10^5	0.81	0.95	1.01	0.99	1.01	1.0	0.99	0.86
10^6	0.91	1.03	1.1	1.13	1.13	1.15	1.17	1.16
10^7	0.97	1.1	1.15	1.17	1.19	1.21	1.22	1.22

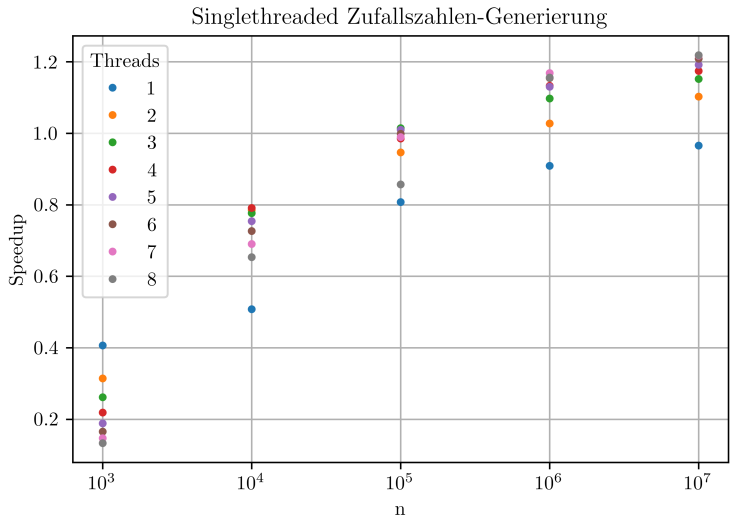


Abbildung 3.3: Speedups der Variante Singlethreaded Zufallszahlen-Generierung

Vector Extensions

Durch die 512 Bit Vektorinstruktionen kann mit 16 floats gleichzeitig gerechnet werden. Insgesamt könnte also ein Speedup von maximal 4 echte Kerne * 16 = 64 erreicht werden. Allerdings werden die Zufallszahlen lediglich mit 256 Bit Vektorinstruktionen generiert, was erklären könnte, wieso der in Tabelle 3.11 und Abbildung 3.4 ersichtliche, gemessene Speedup bei maximal 41 liegt.

Tabelle 3.11: Speedups der Variante Vector Extensions

n	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**	**t=6**	**t=7**	**t=8**
10^3	0.67	0.44	0.34	0.28	0.23	0.2	0.17	0.16
10^4	4.76	4.36	3.55	2.89	2.36	2.05	1.8	1.61
10^5	6.09	16.57	18.71	18.61	17.32	16.5	15.38	14.29
10^6	8.6	16.39	18.45	31.42	33.8	39.42	41.42	37.27
10^7	11.25	20.8	26.01	33.12	33.84	38.57	40.74	37.63

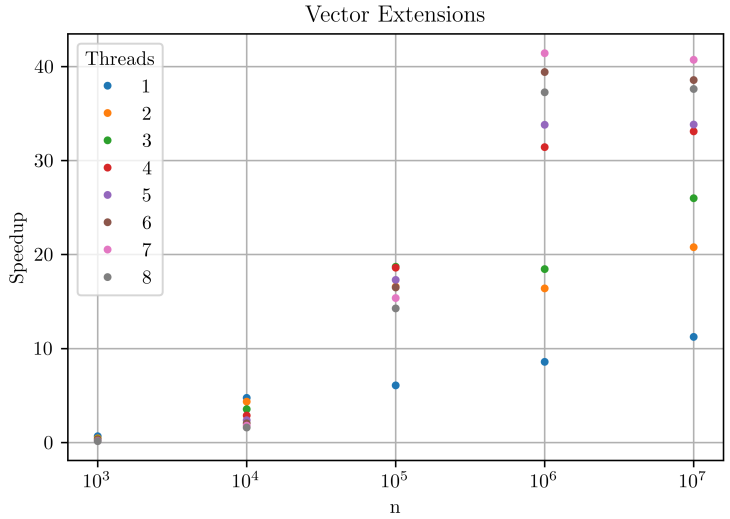


Abbildung 3.4: Speedups der Variante Vector Extensions

Device API Zufallszahlen-Generierung

Die Tabelle 3.12 zeigt, dass ab $n = 10'000$ die Variante Device API Zufallszahlen-Generierung schneller ist, als die serielle Implementierung.

Tabelle 3.12: Speedups der Variante Device API Zufallszahlen-Generierung

n	**Speedup**
10^3	0.18
10^4	1.9
10^5	8.36
10^6	20.42
10^7	38.23

Host API Zufallszahlen-Generierung

Die Tabelle 3.13 zeigt, dass ab n=100'000 die Variante Host API Zufallszahlen-Generierung schnell ist als die serielle Implementierung.

Tabelle 3.13: Speedups der Variante Host API Zufallszahlen-Generierung

n	Speedup
10^3	0.05
10^4	0.5
10^5	4.96
10^6	28.69
10^7	136.51

Unified Memory

Der Tabelle 3.14 kann entnommen werden, dass ab n=100'000 die Variante Unified Memory schneller ist als die serielle Implementierung.

Tabelle 3.14: Speedups der Variante Unified Memory

n	Speedup
10^3	0.03
10^4	0.33
10^5	2.48
10^6	16.09
10^7	37.23

Thrust Zufallszahlen-Generierung

Tabelle 3.15 zeigt, dass ab n=10'000 die Variante Thrust Zufallszahlen-Generierung schneller als die serielle Implementierung ist.

Tabelle 3.15: Speedups der Variante Thrust Zufallszahlen-Generierung

n	Speedup
10^3	0.18
10^4	1.92
10^5	17.85
10^6	69.04
10^7	179.43

3.2.2 Binomialmodell

Serielle Implementierung

Der Binomialmodell-Algorithmus hat in der Theorie eine quadratische Laufzeit. Tabelle 3.16 und Abbildung 3.5 zeigen, dass der Algorithmus auch in der Praxis ungefähr eine quadratische Laufzeit hat.

Tabelle 3.16: Laufzeiten der seriellen Binomialmodell-Implementierung

n	Laufzeit (ms)
31	0.01
63	0.06
127	0.22
255	0.88
511	3.92
1,023	16.64

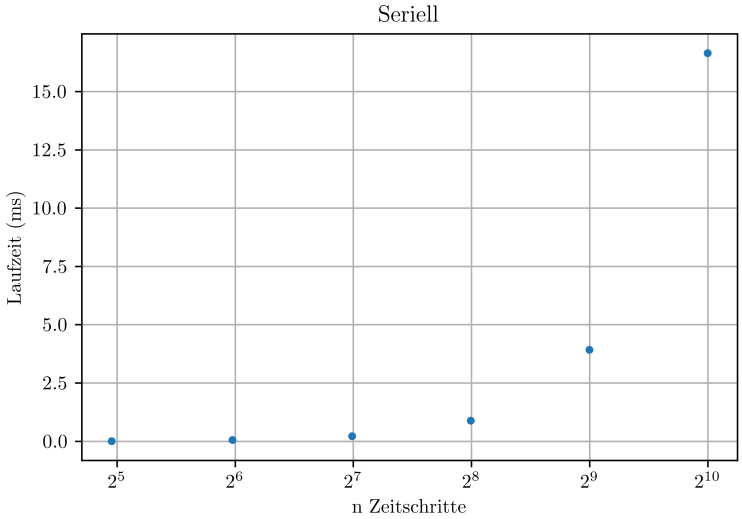


Abbildung 3.5: Laufzeiten der seriellen Binomialmodell-Implementierung

Barriere

Selbst bei n=1'023 konnte kein nennenswerter Speedup erreicht werden, wie Tabelle 3.17 und Abbildung 3.8 zeigen.

Tabelle 3.17: Speedups der Variante Barriere

n	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
31	0.14	0.07	0.04	0.03	0.03	0.02	0.02	0.02
63	0.43	0.28	0.11	0.1	0.1	0.08	0.06	0.05
127	0.52	0.66	0.26	0.22	0.23	0.18	0.16	0.13
255	0.66	1.09	0.56	0.47	0.46	0.4	0.34	0.28
511	0.74	1.04	1.23	0.94	0.82	0.74	0.71	0.61
1,023	0.9	1.1	1.73	1.6	1.25	1.21	1.2	1.1

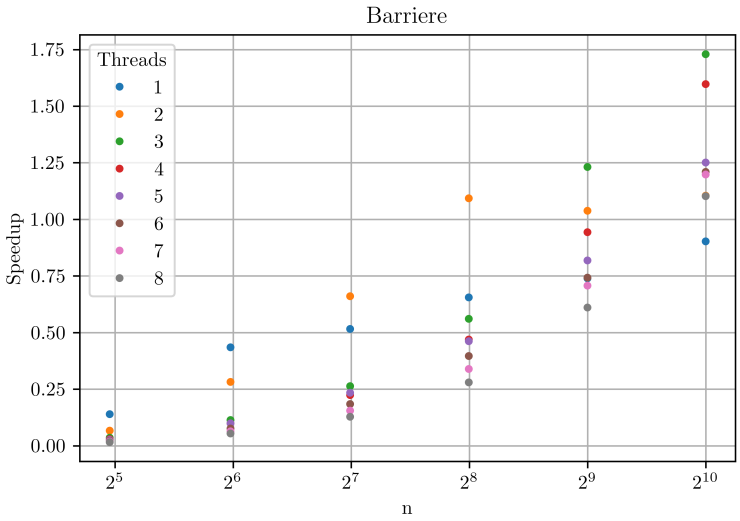


Abbildung 3.6: Speedups der Variante Barriere

Atomare Variablen

Wie bei der Variante Barriere konnte auch mit dieser Variante kein nennenswerter Speedup erreicht werden.

Tabelle 3.18: Speedups der Variante Atomare Variablen

n	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
31	0.14	0.08	0.07	0.05	0.04	0.04	0.03	0.01
63	0.43	0.29	0.26	0.22	0.19	0.17	0.14	0.06
127	0.5	0.56	0.6	0.55	0.5	0.46	0.41	0.18
255	0.62	0.62	0.49	0.86	0.8	0.8	0.8	0.36
511	0.73	0.96	1.07	1.16	1.19	1.14	1.21	0.58
1,023	0.84	1.19	1.44	1.3	1.34	1.35	1.43	0.97

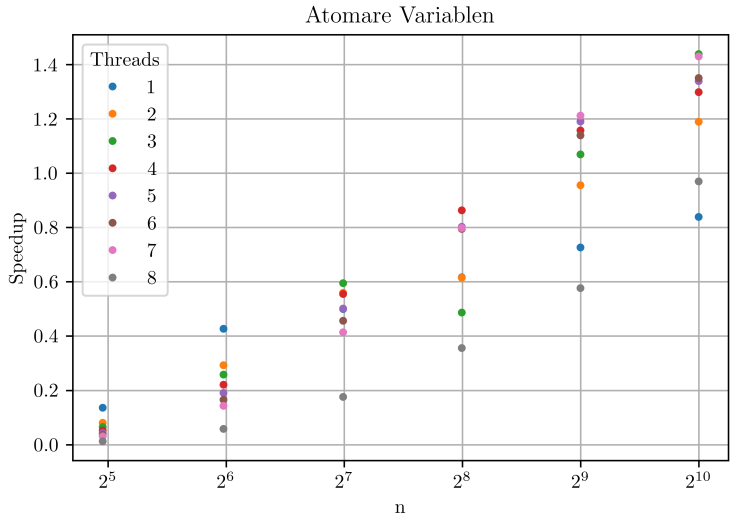


Abbildung 3.7: Speedups der Variante Atomare Variablen

Thread Pool

Wie bei der Variante Barriere konnte auch mit dieser Variante kein nennenswerter Speedup erreicht werden.

Tabelle 3.19: Speedups der Variante Thread Pool

n	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
31	0.1	0.07	0.02	0.01	0.0	0.0	0.0	0.0
63	0.28	0.2	0.08	0.04	0.03	0.02	0.01	0.01
127	0.31	0.36	0.22	0.11	0.07	0.04	0.03	0.02
255	0.55	0.57	0.48	0.24	0.14	0.08	0.07	0.07
511	0.78	0.94	0.83	0.51	0.38	0.32	0.32	0.35
1,023	0.87	1.22	1.22	1.04	0.97	0.86	0.87	0.86

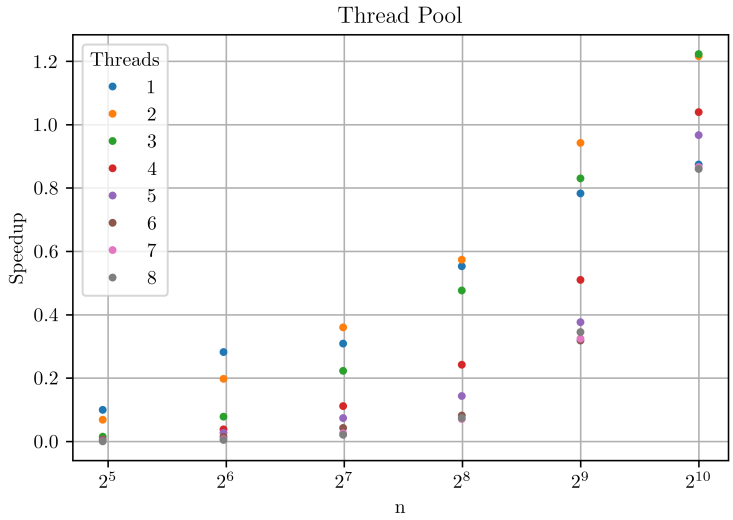


Abbildung 3.8: Speedups der Variante Thread Pool

Global Memory

In Tabelle 3.20 ist ersichtlich, dass ab n=255 ein Speedup von grösser als eins erreicht wurde. Dies deckt sich mit den Resultaten von Greinsmark und Lindström [11], die ab 200 Zeitschritten einen Speedup von grösser als eins beobachtet haben.

Tabelle 3.20: Speedups der Variante Global Memory

n	Speedup
31	0.04
63	0.19
127	0.62
255	1.73
511	3.84
1,023	5.2

Shared Memory

Wie bei der Variante Global Memory ist auch bei dieser Variante ab n=255 der Speedup grösser gleich eins.

Tabelle 3.21: Speedups der Variante Shared Memory

n	Speedup
31	0.04
63	0.2
127	0.64
255	1.85
511	4.98
1,023	9.4

3.2.3 Multibinomialmodell

Serielle Implementierung

Wie Tabelle 3.22 und Abbildung 3.9 zeigen, weist die serielle Implementierung eine quadratische Laufzeit auf, wie es gemäss theoretischer Analyse zu erwarten gewesen ist.

Tabelle 3.22: Laufzeiten der seriellen Multibinomialmodell-Implementierung

n	Laufzeit (ms)
31	19.7
63	80.98
127	331.88
255	1,384.04
511	6,210.45
1,023	28,048.97

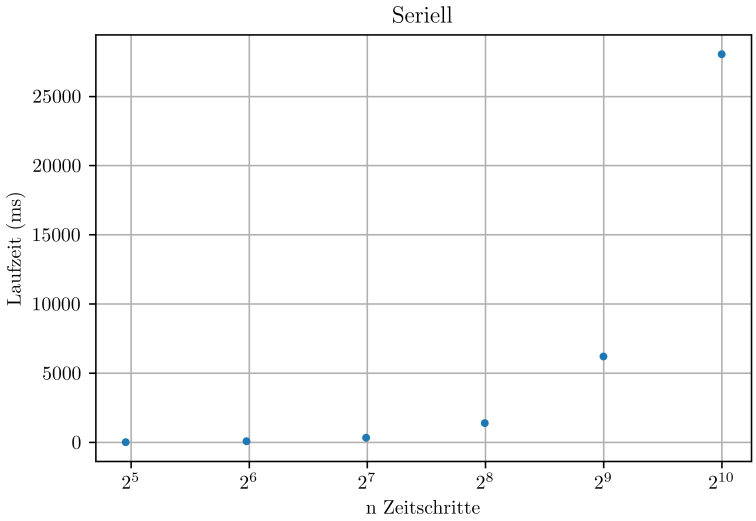


Abbildung 3.9: Laufzeiten der seriellen Multibinomialmodell-Implementierung

Thread Pool

Aus Tabelle 3.23 und Abbildung 3.10 geht hervor, dass der Speedup bei hinreichend grossem n etwa bei der Anzahl echter Kerne liegt.

Tabelle 3.23: Speedups der Variante Thread Pool

n	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
31	0.82	1.51	2.2	2.15	1.98	1.86	1.87	1.87
63	0.94	1.68	2.27	2.78	3.03	3.27	3.44	3.55
127	0.97	1.7	2.42	2.88	3.22	3.48	3.63	3.76
255	1.0	1.89	2.53	2.98	3.35	3.56	3.73	3.79
511	1.0	1.92	2.56	3.07	3.41	3.69	3.89	3.97
1,023	1.01	1.97	2.66	3.2	3.48	3.76	3.97	4.08

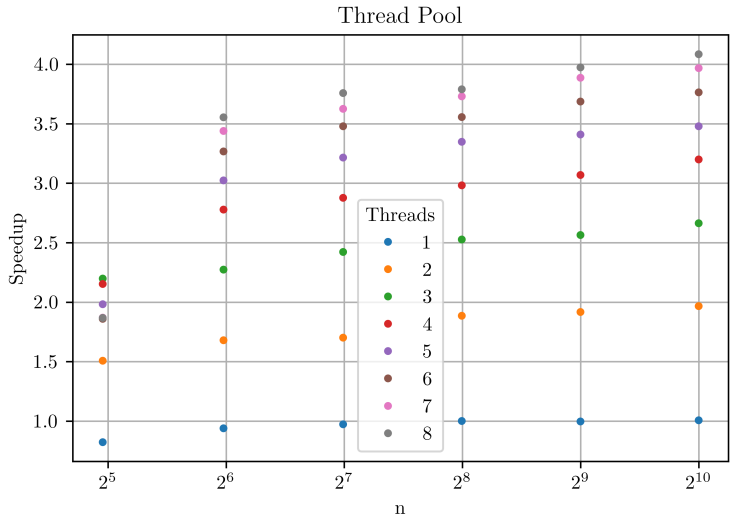


Abbildung 3.10: Speedups der Variante Thread Pool

Self-Managed Threads

Die in Tabelle 3.24 und Abbildung 3.11 ersichtlichen Ergebnisse entsprechen etwa dem Ergebnis der Variante Thread Pool.

Tabelle 3.24: Speedups der Variante Self-Managed Threads

n	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
31	0.9	1.77	2.03	2.76	2.59	3.04	3.38	3.18
63	0.97	1.79	2.18	2.69	2.62	3.1	3.46	3.31
127	0.99	1.85	2.38	2.85	2.88	3.26	3.53	3.43
255	1.0	1.9	2.48	2.96	3.13	3.41	3.65	3.66
511	1.06	2.04	2.7	3.22	3.46	3.78	3.98	4.02
1,023	1.05	2.06	2.78	3.3	3.56	3.84	4.03	4.06

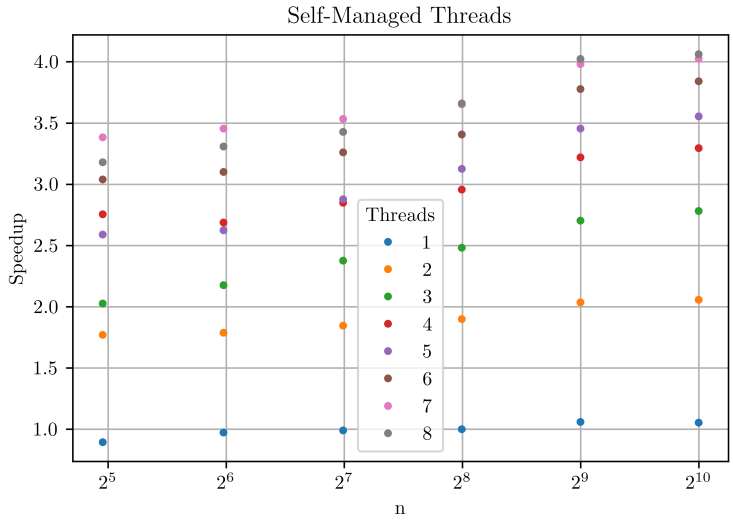


Abbildung 3.11: Speedups der Variante Self-Managed Threads

Global Memory

Mit der Variante Global Memory kann, wie in Tabelle 3.25 ersichtlich ist, ein Speedup von 127 bei n=1'023 erreicht werden. Allerdings wird bei kleineren, in der Praxis verwendeten n von 31 und 63 noch kein so grosser Speedup erreicht.

Tabelle 3.25: Speedups der Variante Global Memory

n	Speedup
31	1.29
63	5.08
127	17.39
255	45.46
511	82.35
1,023	127.41

Shared Memory

Tabelle 3.26 zeigt, dass diese Variante bereits ab n=31 einen grossen Speedup aufweist.

Tabelle 3.26: Speedups der Variante Shared Memory

n	Speedup
31	62.31
63	151.59
127	303.32
255	543.3
511	722.32
1,023	900.95

3.2.4 Vergleich

In diesem Abschnitt werden die implementierten Varianten verglichen. Da für die Multithreading-Varianten Benchmarks mit unterschiedlichen Anzahl Threads durchgeführt wurden, wurde bestimmt, dass für den Vergleich immer die Resultate mit sieben Threads verwendet werden, da die meisten Multithreading-Varianten damit die kleinste Laufzeit aufweisen. Ferner scheint es eine logische Wahl, da die Algorithmen, zumindest bei grossen n, theoretisch alle von einer grösseren Anzahl Threads profitieren sollten.

Bei den Multithreading Varianten hat sich gezeigt, dass kaum ein Speedup erreicht werden konnte, der grösser als die Anzahl echter Kerne ist. Das Hyperthreading scheint nur einen kleinen, positiven Einfluss auf die Performance zu haben. Dies liegt vermutlich darin begründet, dass beim Hyperthreading sich zwei Threads ein Rechenwerk teilen. Hyperthreading bietet den Vorteil, dass wenn ein logischer Kern blockiert ist, der andere logische Kern das Rechenwerk beanspruchen kann. Dies kann kann unter anderem bei Cache Misses oder Branch Misspredictions vorkommen [24]. Da alle implementierten Modelle und deren Varianten sehr rechenintensiv sind, ist der Nutzen des Hyperthreadings sehr gering.

Monte-Carlo-Simulation

Die Abbildung 3.12 bietet eine Übersicht über die Performance der verschiedenen Monte-Carlo-Simulation-Varianten. Es fällt auf, dass die Multithreading Varianten mit Ausnahme der Vector Extensions Variante einen kleineren Speedup als die GPU Varianten aufweisen. Der Grund dafür ist, dass der Speedup dieser Multithreading Varianten durch die Anzahl Prozessorkerne begrenzt ist. Etwas überraschend ist, dass die Vector Extensions Variante selbst bei n=10'000'000 noch mit einigen GPU Varianten mithalten kann. Ferner zeigt sich, dass die Zufallszahlen-Generierung mittels Thrust offenbar schneller ist als die Generierung von Zufallszahlen mittels cuRAND. Auffällig ist auch, dass die Unified Memory Variante massiv langsamer als die Host API Zufallszahlen-Generierung Variante ist, denn die Generierung der Zufallszahlen findet bei beiden Varianten gleich statt, was bedeuten würde, dass die automatisierte Speicherverwaltung, die bei der Unified Memory Variante zum Einsatz kommt, einen grossen, negativen Einfluss auf die Performance hat.

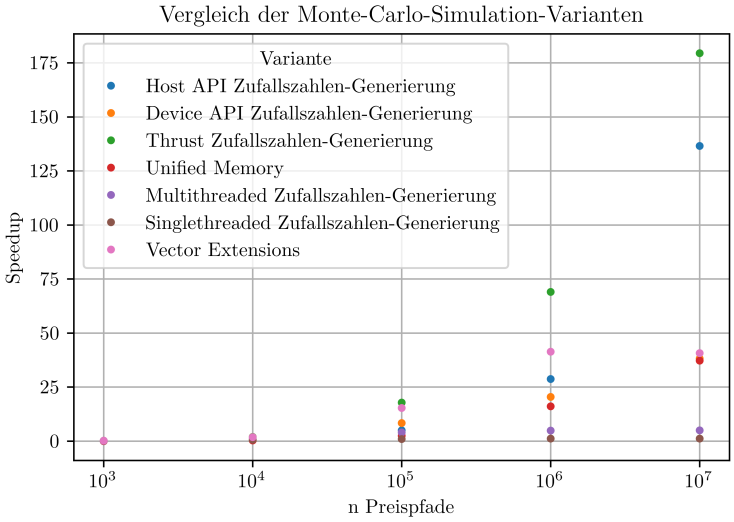


Abbildung 3.12: Vergleich der verschiedenen Monte-Carlo-Simulation-Varianten

Binomialmodell

Eine Übersicht über die Performance der verschiedenen Binomialmodell-Varianten ist in Abbildung 3.13 ersichtlich. Im Vergleich zur Monte-Carlo-Simulation sind die Speedups deutlich kleiner. Dies war zu erwarten, da erstens das grösste n, das getestet wurde 1'023 war und zweitens viel mehr Synchronisation nötig ist. Ferner ist zu erkennen, dass das Shared Memory einen messbaren, positiven Einfluss auf die Performance hat.

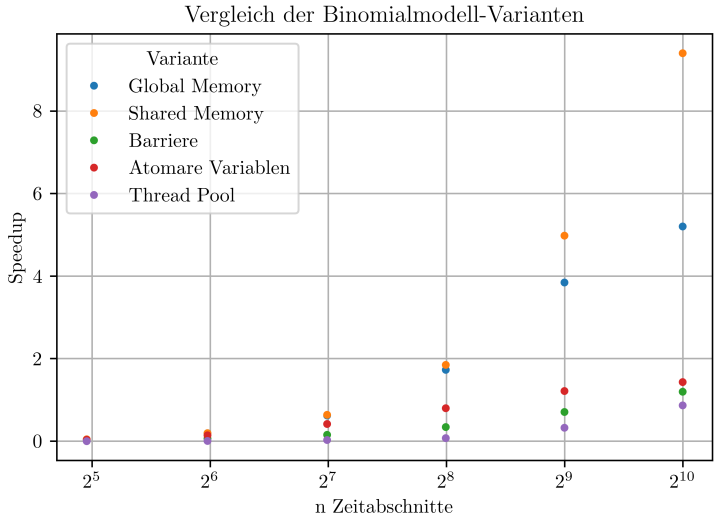


Abbildung 3.13: Vergleich der verschiedenen Binomialmodell-Varianten

Multibinomialmodell

Die Performance der verschiedenen Multibinomialmodell-Varianten ist in Abbildung 3.14 ersichtlich. Beim Multibinomialmodell zeigte sich analog zur Monte-Carlo-Simulation, dass der Speedup der Multithreading Varianten durch die Anzahl Prozessorkerne begrenzt ist. Dass die Shared Memory Variante schneller als die Global Memory Variante ist, kann dadurch begründet werden, dass es bei n Zeitschritten $\frac{n^2}{2} + \frac{3n}{2} + 1$ Knoten zu berechnen gibt. Bei jedem Knoten muss der Speicher einmal beschrieben und zweimal gelesen werden (ausser beim letzten Zeitpunkt wo nur ein einmaliges Beschreiben notwendig ist). Dadurch, dass das Shared Memory im Gegensatz zum Global Memory auf dem Chip verbaut ist, kann eine viel niedrigere Latenz und höhere Bandbreite erreicht werden [25].

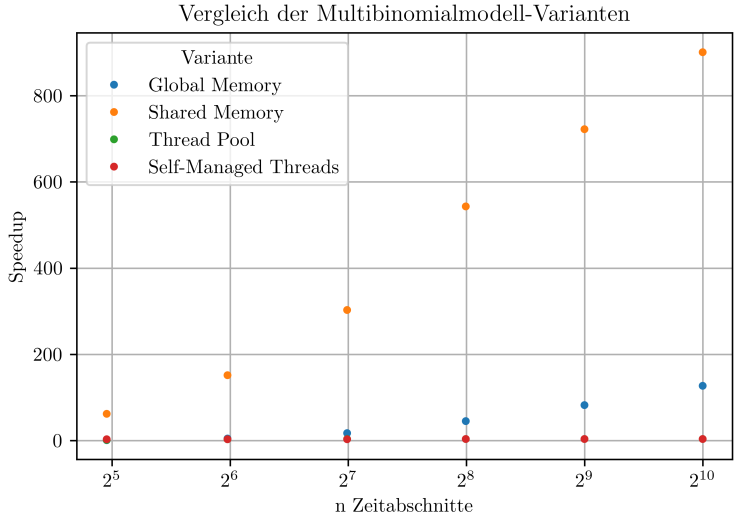


Abbildung 3.14: Vergleich der verschiedenen Multibinomialmodell-Varianten

3.3 Erfüllung der Anforderungen

Dieser Abschnitt beschreibt, inwiefern die funktionalen und nicht-funktionalen Anforderungen erfüllt wurden.

3.3.1 Funktionale Anforderungen

Es wurden alle funktionalen Anforderungen erfüllt.

UC01: Bewertung europäischer Optionen

Während den Systemtests wurde festgestellt, dass es bei gewissen Bewertungs-Parametern zu einem Overflow des Optionspreises kommt. Da das Problem bei der Bewertung mit dem Binomialmodell gravierender ist, wird es im Abschnitt zur Bewertung von amerikanischen Optionen näher beschrieben. Abgesehen von diesem Umstand waren die Systemtests erfolgreich. Das Ergebnis der Systemtests ist im Abschnitt B.2.2 ersichtlich.

UC02: Bewertung amerikanischer Optionen

Damit die Bewertung von europäischen und amerikanischen Optionen einheitlich gehandhabt wird, wurde entschieden die Overflow-Problematik gleich zu lösen. Es wurden Preconditions für die Bewertungs-Parameter definiert. So wurde beispielsweise die Volatilität, die eingegeben werden kann, auf das Intervall [0;1] beschränkt. Eine weiteres Problem ist, dass die Volatilität und der risikofreier Zinssatz nicht unbedingt vom Nutzer eingegeben werden müssen, sondern dass er diesen automatisch ermitteln lassen kann. In diesem Fall gibt es keine Garantie, dass die automatisch ermittelten Werte in den definierten Intervallen liegt, weshalb wir diese manuell in das auch für die Eingabe geltende Intervall normalisieren. Würde also beispielsweise die implizite Volatilität von Yahoo Finance grösser als eins sein, würde sie auf eins gesetzt werden. Die erlaubten Inputs sind in der NestJS REST API Beschreibung, die im Anhang C verfügbar ist, ersichtlich.

Nach diesen Anpassungen wurden die entsprechenden Systemtests erneut und dieses mal erfolgreich durchgeführt. Das Ergebnis ist im Abschnitt B.2.2 ersichtlich. Es ist jedoch davon auszugehen, dass die gewählten Preconditions nicht in jedem Fall einen Overflow verhindern, also dass es Kombinationen von Eingaben gibt, so dass es zu einem Overflow kommt. Das Problem ist, dass es sehr viele Variablen gibt (unter anderem Volatilität, risikofreier Zinssatz, Anzahl Zeitabschnitte), die bestimmen ob ein Overflow auftreten wird. Je nach Optionstyp (Call oder Put) haben die Variablen zudem einen entgegengesetzten Einfluss auf den Preis.

UC03: Verwaltung von Bewertungsanfragen

Wie die Systemtests B.2.3 zeigen, konnte diese Anforderung umgesetzt werden.

UC04: Verwaltung von Bewertungsresultaten

Wie die Systemtests B.2.3 zeigen, konnte diese Anforderung umgesetzt werden.

UC05: Identifizierung von gewinnbringenden Optionen

Diese Anforderung wurde mit dem “Profit-Spotter” umgesetzt. Ursprünglich war geplant dem Nutzer ausschliesslich gewinnbringende Optionen anzuzeigen. Da sowieso alle Opti-

onswerte berechnet werden müssen, wurde entschieden, dem Nutzer alle Optionen anzuzeigen. Da der Nutzer die Möglichkeit hat, die Optionen nach der Preisdifferenz zu sortieren, kann er gewinnbringende Optionen trotzdem identifizieren. Die Systemtests B.2.6 zeigen, dass die Anforderung erfolgreich umgesetzt werden konnte.

Während den Systemtests wurde das Problem erkannt, dass Yahoo Finance bei Optionen mit einem sehr hohen Strike Preis manchmal unplausible Optionswerte zurückgibt. Die Problematik kann am im Abbildung 3.15 ersichtlichen Beispiel verdeutlicht werden. Es handelt sich um eine Call Option auf die Apple Aktie mit einem Strike Preis von $700, die am 17.06.2022 ausläuft. Die Abbildung wurde am 13.06.2022 erstellt, es dauerte also noch vier Tage bis die Option ausläuft. Zum Zeitpunkt der Erstellung des Screenshots war der Aktienkurs von Apple bei $137.13. Der von Yahoo Finance gemeldete Optionspreis von $1'125 macht keinen Sinn, da es sehr unwahrscheinlich ist, dass die Apple Aktie innerhalb von vier Tagen ihren Kurs vervielfacht. Dementsprechend liegt der wahre Optionswert nahe null.

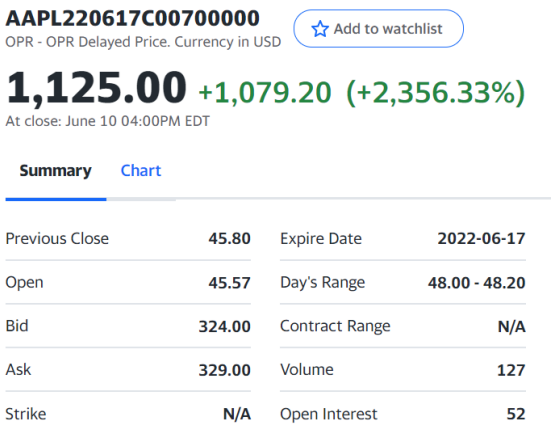


Abbildung 3.15: Auszug aus Yahoo Finance

Da wir keinen Einfluss auf die Daten von Yahoo Finance haben und die Zeit zu weit fortgeschritten war, um Yahoo Finance durch einen anderen Datenprovider zu ersetzen, konnte dieses Problem nicht behoben werden.

3.3.2 Nicht-funktionale Anforderungen

Dieser Abschnitt beschreibt die Zielerreichung der nicht-funktionalen Anforderungen.

NFR 01: Learnability

Es kann nicht abschliessend beurteilt werden, inwiefern diese Anforderung erfüllt wurde, da keine Usability-Tests durchgeführt wurden. Während der Entwicklung wurde allerdings darauf geachtet, eine möglichst intuitive Benutzeroberfläche zu erstellen.

NFR 02: User Error Protection

Diese Anforderungen wurde erfüllt. Das Frontend verhindert das Absenden von fehlerhaften Angaben. Falls der Benutzer fehlerhafte Eingaben tätigt, macht ihn das Frontend darauf aufmerksam und teilt ihm mit, welche Eingaben möglich sind.

NFR03: Fault Tolerance

Diese Anforderung wurde erfüllt. Falls der CUDA oder Multithreading Option Pricer ausfällt, funktioniert die Bewertung mit dem jeweils anderen Option Pricer noch. Die Verwendung des RabbitMQ Message Brokers verhindert zudem den Verlust von Bewertungsanfragen. Falls ein Option Pricer während der Optionsbewertung abstürzt, wird die Bewertungsanfrage zurück in die Queue gelegt. Sobald der Option Pricer wieder verfügbar ist, wird er die Bewertungsanfrage erneut ausführen. Dies wurde dadurch realisiert, dass die Option Pricer den Empfang einer Bewertungsanfrage erst bestätigen, nachdem sie das Resultat in die `response` Queue gelegt haben.

NFR04: Recoverability

Diese Anforderung wurde erfüllt.

NFR05: Authenticity (Passwortlänge)

Die Anforderung wurde erfüllt. Passwörter die weniger als acht Zeichen lang sind, werden abgelehnt.

NFR06: Authenticity (Hashing von Passwörtern)

Die Anforderung wurde erfüllt. Die Passwörter werden mit Bcrypt[7] gehashed.

NFR07: Modularity

Diese Anforderung wurde erfüllt. Für die Optionsbewertung mittels CUDA und Multithreading wurde jeweils ein eigener Microservice erstellt, der unabhängig von der restlichen Applikation deployt werden kann.

NFR08: Analysability (Schnittstellenbeschreibung)

Diese Anforderung wurde erfüllt. Die Schnittstelle der NestJS REST API wurde nach OpenAPI und die Schnittstelle des RabbitMQ Message Broker nach AsyncAPI spezifiziert.

NFR09: Analysability (Logging-Format)

Diese Anforderung wurde erfüllt. Die NestJS REST API loggt HTTP-Anfragen nach dem Common Log Format. Um dies zu erreichen, wurde eine eigene Middleware geschrieben.

NFR10: Modifiability

Diese Anforderung wurde erfüllt. Durch die Verwendung von i18n[8] konnten alle Texte in einer separaten JSON-Datei definiert werden. Zudem wurde die Applikation bereits zweisprachig (Deutsch und Englisch) implementiert.

NFR11: Testability

Diese Anforderung wurde erfüllt. Jeder Anwendungsfall ist durch mindestens einen Systemtest abgedeckt.

[7] Siehe: https://www.npmjs.com/package/bcrypt
[8] Siehe: https://www.npmjs.com/package/react-i18next

NFR12: Adaptability

Diese Anforderungen wurde erfüllt. StOP wurde ausgiebig mit Chrome Version 99 und Firefox Version 97 getestet.

NFR13: Installability

Diese Anforderung wurde erfüllt. Es wird aber angenommen, dass die installierende Person Erfahrung im Umgang mit Docker und Docker-Compose hat.

3.4 Code Metriken

Die Code Metriken wurden mittels SonarQube ermittelt. In der NestJS REST API existiert eine grosse Anzahl Klassen, da das DTO Pattern verwendet wurde. Die in den CUDA und Multithreading Option Pricer vorhandene, hohe Code Duplikation lässt sich damit erklären, dass die verschiedenen Optionsbewertungs-Varianten eine ähnliche Struktur aufweisen. Beispielsweise ist die Formel, wie ein einzelner Knoten beim Binomialbaum ausgerechnet wird immer exakt dieselbe, egal wie der Binomialbaum parallelisiert wird.

Tabelle 3.27: React SPA Code Metriken

Metrik	**Wert**
Lines of Code	4'254
Funktionen	359
Klassen	15
Dateien	61
Duplikation	2.3%
Code Smells	0

Tabelle 3.28: NestJS REST API Code Metriken

Metrik	**Wert**
Lines of Code	3'639
Funktionen	259
Klassen	107
Dateien	66
Duplikation	3.2%
Code Smells	0

Tabelle 3.29: CUDA Option Pricer Code Metriken

Metrik	**Wert**
Lines of Code	1'201
Funktionen	56
Klassen	11
Dateien	32
Duplikation	8.2%
Code Smells	0

Tabelle 3.30: Multithreading Option Pricer Code Metriken

Metrik	**Wert**
Lines of Code	1'114
Funktionen	45
Klassen	9
Dateien	26
Duplikation	19.4%
Code Smells	0

3.4.1 Test-Coverage

Tabelle 3.31 bietet eine Übersicht über die Test-Coverage in den einzelnen Microservices. Die React SPA ist nicht aufgeführt, da keine Tests dafür geschrieben wurden. In der NestJS REST API bezieht sich die Coverage auf die gesamte Codebasis, während sich die Coverage im CUDA und Multithreading Option Pricer auf die "lib-option-pricing" bezieht. Die Test Coverage in der NestJS REST API wurde mittels Jest[9] ermittelt, während die Test Coverage im CUDA und Multithreading Option Pricer mittels OpenCppCoverage[10] bestimmt wurde. Ferner ist zu beachten, dass die Coverage im NestJS REST API nur die Unit-Tests, nicht aber die Integrationstests umfasst.

Tabelle 3.31: Test-Coverage pro Microservice

Microservice	**Test Coverage**
NestJS REST API	46.7%
Multithreading Option Pricer	98%
CUDA Option Pricer	74%

[9] Siehe: https://jestjs.io/
[10] Siehe: https://github.com/OpenCppCoverage/OpenCppCoverage

Kapitel 4

Diskussion

In diesem Kapitel werden die Resultate diskutiert und mögliche Ansätze für die Weiterentwicklung von StOP vorgeschlagen.

4.1 Bewertung der Ergebnisse

Mit StOP wurde eine Applikation entwickelt, die alle funktionalen sowie nicht-funktionalen Anforderungen erfüllt. Die gewählten Technologien haben sich bewährt. Das im Frontend im Einsatz stehende React-Framework ermöglichte die Aufteilung der grafischen Benutzeroberfläche in Komponenten, was einen positiven Effekt auf die Wartbarkeit hat. Analog dazu besteht die NestJS REST API aus mehreren Modulen. Dadurch dass eine Microservice-Architektur gewählt wurde, können die einzelnen Komponenten von StOP unabhängig voneinander weiterentwickelt werden, solange die Schnittstellen unverändert bleiben. Dass es sinnvoll war, die Optionsbewertungs-Algorithmen in eigene Bibliotheken auszulagern, hat sich bereits während des Projektes gezeigt. So griff die NestJS REST API zuerst über eine REST API auf den CUDA und Multithreading Option Pricer zu. Im Verlaufe des Projektes wurden diese REST-Schnittstellen durch einen RabbitMQ Message Broker ersetzt. Weil die Optionsbewertungs-Algorithmen in einer separaten Bibliothek abgelegt wurden, konnte dieser Umbau mit relativ geringem Aufwand durchgeführt werden. Damit konnte gezeigt werden, dass diese Algorithmen problemlos auch in andere Applikationen integriert werden können. Die Weiterentwicklung der Arbeit wird dadurch erleichtert, dass die REST-Schnittstelle nach OpenAPI und die RabbitMQ Schnittstelle nach AsyncAPI spezifiziert wurde. Dadurch dass für alle Microservices Dockerimages erstellt wurden, kann StOP trotz der Aufteilung in mehrere Dienste mit überschaubarem Aufwand ausgeliefert werden.

Es wurde gezeigt, dass die Bewertung von europäischen Optionen mittels Monte-Carlo-Simulation durch Multithreading und die Nutzung einer Grafikkarte massiv beschleunigt werden kann. Beim Binomialmodell hat ausschliesslich die Parallelisierung durch die Nutzung einer Grafikkarte zu einem signifikanten Speedup geführt. Die Hypothese von Greinsmark und Lindström [11], dass durch die Batch-Verarbeitung von mehreren Binomialbäumen ein signifikanter Speedup erreicht werden kann, wurde bestätigt. So erreicht unsere beste Multibinomialmodell Variante einen Speedup von 900 bei der Bewertung von 1'571 Optionen. Es ist davon auszugehen, dass die Speedups der einzelnen Varianten verbessert werden können, da in dieser Arbeit der Fokus mehr auf dem Ausprobieren von verschiedenen Ansätzen als auf der Optimierung eines einzelnen Ansatzes lag. Das Ergebnis, dass die Optionsbewertung mittels Monte-Carlo-Simulation und Binomialmodell durch Parallelisierung beschleunigt werden kann, stimmt mit den Ergebnissen der Literatur überein.

4.2 Ausblick

Die Funktionalitäten des NestJS REST API könnten auf mehrere Microservices aufgeteilt werden. Es könnte beispielsweise jeweils ein Microservice für die Persistenz, Nutzerverwaltung, Black-Scholes Bewertung und Ermittlung von Daten von Yahoo Finance erstellt werden. Damit könnten diese Funktionen unabhängig voneinander gewartet und weiterentwickelt werden.

Momentan pollt das Frontend die NestJS REST API, um das Ergebnis einer Bewertungsanfrage zu erhalten. Der Polling-Mechanismus könnte durch eine Websocket-Verbindung ersetzt werden. So könnte die Latenz bis zum Eintreffen des Resultates etwas reduziert werden.

Die zeitliche Performance von StOP ist durch die Yahoo Finance API begrenzt. Am stärksten zeigt sich das beim Profit-Spotter. Die Zeit zur Berechnung der Optionswerte ist häufig im zweistelligen Millisekundenbereich, während die Zeit zum Herunterladen aller benötigten Informationen von Yahoo Finance im Bereich von Sekunden liegt. Dies liegt auch daran, dass mehrere HTTP-Requests gemacht werden müssen, da es nicht möglich ist, alle Optionen zu einer Aktie auf einmal anzufordern, sondern jeweils nur alle Optionen zu einem Fälligkeitsdatum. Auch beim Suchen nach Aktien muss für jede gefundene Aktie ein HTTP-Request abgesendet werden, um zu prüfen, ob zu der jeweiligen Aktie auch Optionen existieren. Dazu kommt, dass die Yahoo Finance API keine offizielle, für die Öffentlichkeit bestimmte API ist, sondern eigentlich nur für den Betrieb der Yahoo Finance Webapplikation bestimmt ist. Dies bedeutet erstens, dass es keine offizielle API-Dokumentation gibt und zweitens die API jederzeit für externe gesperrt werden könnte. Ausserdem gibt es keine Zeit- oder Korrektheitsgarantien. So liefert die Yahoo Finance API bei Optionen mit grossen Strike Preisen teilweise unplausible Marktpreise. Wenn StOP zu viele API Anfragen an die Yahoo Finance API senden würde, könnten zudem rechtliche Probleme auftreten oder die IP-Adresse könnte gesperrt werden. Es wäre deshalb empfehlenswert, eine alternative Quelle für die Finanzdaten zu nutzen. Um die beschriebenen Probleme zu vermeiden, würde sich ein kommerzieller Anbieter eignen, der zwar Kosten verursacht, dafür eine dokumentierte API, Zeit- und Korrektheitsgarantien bietet.

Die Monte-Carlo-Simulation könnte angepasst werden, so dass keine konstante Volatilität und kein konstanter risikofreier Zinssatz verwendet wird, sondern diese beiden Parameter als Funktion der Zeit modelliert werden. Es wäre zudem sinnvoll, dem Nutzer den Standardfehler anzuzeigen. Um ein besseres Resultat bei gleicher Anzahl Preispfade zu erreichen, könnten verschiedene Ansätze zur Varianzreduktion, wie beispielsweise von Joshi beschrieben [1], implementiert werden. In unserer Arbeit wurde die Güte der verwendeten Zufallszahlen vernachlässigt. Es wäre möglich, dass ein Zufallsgenerator zwar schneller ist als ein anderer, aber weil er Zufallszahlen mit schlechteren statistischen Eigenschaften generiert, eine grössere Anzahl Preispfade benötigt um die gleiche Präzision wie ein anderer Zufallsgenerator zu erreichen.

Sowohl die Monte-Carlo-Simulation als auch das Binomialmodell könnten dahingehend erweitert werden, dass allfällige Dividenden auf den Basiswert berücksichtigt werden.

In der Benutzeroberfläche könnten die Binomialbäume grafisch dargestellt werden. Dabei müssten allerdings einige Interfaces geändert werden, da momentan nicht der vollständige Binomialbaum, sondern ausschliesslich der Optionswert an das Frontend gesendet wird. Ferner könnte das Binomialmodell mit doppelter Präzision implementiert werden. Damit wäre es möglich, die Beschränkungen der Parameter zu lockern und es könnten Binomialbäume mit mehr Zeitabschnitten erstellt werden, ohne dass ein Overflow auftritt.

Es könnte ein zusätzlicher Microservice erstellt werden, der die Optionsbewertung mit-

tels Multiprocessing, beispielsweise unter Verwendung von MPI, ermöglicht. Insbesondere die Monte-Carlo-Simulation und das Multibinomialmodell könnten dadurch im Vergleich zum Multithreading stark beschleunigt werden.

Gegenwärtig kann die Vector Extensions Variante ausschliesslich mit MSVC kompiliert werden. Es wäre wünschenswert diese Variante plattformunabhängig zu implementieren. Ferner wäre es interessant zu prüfen, ob das Binomialmodell durch die Verwendung von Vector Extensions beschleunigt werden kann.

Denkbar wäre es auch, exotische Optionstypen, wie beispielsweise asiatischen Optionen zu unterstützen. Das wäre jedoch mit erheblichem Aufwand verbunden, da das gegenwärtige Interface für Optionen um die Angabe, aus welchen Zeitpunkten sich der Durchschnittspreis des Basiswertes zusammensetzt und welche Mittelwertfunktion genutzt wird, ergänzt werden müsste.

Literatur

[1] M. Joshi, *The concepts and practice of mathematical finance*. Cambridge: Cambridge University Press, 2003.

[2] "thismatter. The Present Value and Future Value of Money." (n.d.), Adresse: https://thismatter.com/money/investments/present-value-future-value-of-money.htm (besucht am 14. 06. 2022).

[3] C. Banton. "The Importance of Time Value in Options Trading." (Jan. 2022), Adresse: https://www.investopedia.com/articles/optioninvestor/02/021302.asp (besucht am 15. 06. 2022).

[4] J. C. Hull, *Options, futures, and other derivatives*. Boston: Pearson, 2015.

[5] J. C. Cox, S. A. Ross und M. Rubinstein, "Option pricing: a simplified approach," *Journal of Financial Economics*, S. 229–263, Juli 1979.

[6] A. Hayes. "What is the risk-free rate of return?" (Aug. 2021), Adresse: https://www.investopedia.com/terms/r/risk-freerate.asp (besucht am 03. 03. 2022).

[7] "Github. NVIDIA CUDA Samples." (n.d.), Adresse: https://github.com/NVIDIA/cuda-samples/tree/master/Samples/5_Domain_Specific/MonteCarloMultiGPU (besucht am 04. 06. 2022).

[8] V. Podlozhnyuk und M. Harris, "Monte Carlo Option Pricing," Dez. 2013.

[9] S. Grauer-Gray, W. Killian, R. Searles und J. Cavazos, "Accelerating Financial Applications on the GPU," in *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units*, Ser. GPGPU-6, Houston, Texas, USA: Association for Computing Machinery, 2013, 127–136, ISBN: 9781450320177. DOI: 10.1145/2458523.2458536.

[10] L. A. Abbas-Turki, S. Vialle, B. Lapeyre und P. Mercier, "Pricing derivatives on graphics processing units using Monte Carlo simulation," *Concurrency and Computation: Practice and Experience*, Jg. 26, Nr. 9, S. 1679–1697, Juni 2014. DOI: 10.1002/cpe.2862. Adresse: https://hal-supelec.archives-ouvertes.fr/hal-00773740.

[11] H. Greinsmark und E. Lindström, "Optimization of American option pricing through GPU computing," 2017.

[12] N. Zhang, E. Lim, K. Man und C.-U. Lei, "CPU-GPU hybrid parallel binomial American option pricing," *Lecture Notes in Engineering and Computer Science*, Jg. 2196, März 2012. DOI: 10.1142/9789814439084_0013.

[13] S. K. Popuri, A. M. Raim, N. K. Neerchal und M. K. Gobbert, *An Implementation of Binomial Method of Option Pricing using Parallel Computing*, 2018.

[14] N. Zhang und K. L. Man, "Accelerating Financial Code through Parallelisation and Source-Level Optimisation," Ser. Proceedings of the International MultiConference of Engineers and Computer Scientists 2014, Bd. 2, März 2014.

[15] "ISO/IEC 25010." (n.d.), Adresse: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010 (besucht am 15. 06. 2022).

[16] "State of JavaScript. Front-end Frameworks." (n.d.), Adresse: https://2021.stateofjs.com/en-US/libraries/front-end-frameworks/ (besucht am 08. 06. 2022).

[17] B. Shoshany, "A C++17 Thread Pool for High-Performance Scientific Computing," *arXiv e-prints*, arXiv:2105.00613, Mai 2021. DOI: 10.5281/zenodo.4742687. arXiv: 2105.00613.

[18] "cppreference. C++ Compiler Support." (n.d.), Adresse: https://en.cppreference.com/w/cpp/compiler_support (besucht am 13. 06. 2022).

[19] "arc42 Documentation." (n.d.), Adresse: https://docs.arc42.org/section-5/ (besucht am 16. 03. 2022).

[20] *CUDA C++ Programming Guide*. Nvidia, 2022.

[21] "Unified Memory for CUDA." (2017), Adresse: https://developer.nvidia.com/blog/unified-memory-cuda-beginners/ (besucht am 13. 06. 2022).

[22] "NVIDIA cuRAND. Device API Overview." (n.d.), Adresse: https://docs.nvidia.com/cuda/curand/device-api-overview.html (besucht am 06. 06. 2022).

[23] P. Xu, S. Shi und X. CHU, "Performance Evaluation of Deep Learning Tools in Docker Containers," Ser. Proceedings - 2017 3rd International Conference on Big Data Computing and Communications, BigCom 2017, Institute of Electrical und Electronics Engineers Inc., Nov. 2017, S. 395–403. DOI: 10.1109/BIGCOM.2017.32.

[24] D. T. Marr, F. Binns, D. L. Hill u. a., "Hyper-Threading Technology Architecture and Microarchitecture," *Intel Technology Journal*, Jg. 6, Nr. 1, 2002, ISSN: 1535864X.

[25] *CUDA C++ Best Practices Guide*. Nvidia, 2022.

Glossar

AsyncAPI Standard zur Beschreibung von asynchronen Schnittstellen. Siehe https://www.asyncapi.com/. 73, 77

Common Log Format Standard, der das Format von Log-Einträgen für Webserver beschreibt. Durch die Standardisierung können Drittprogramme die generierten Log-Dateien weiterverwenden. Siehe https://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format. 18, 73

cuRAND Mit cuRAND stellt NVIDIA eine Bibliothek zur Generierung von Zufallszahlen auf der GPU zur Verfügung. Siehe:https://docs.nvidia.com/cuda/curand/index.html. 11, 40, 51, 52, 68

doppelte Genauigkeit Gleitkommazahl, die nach IEEE 754 mit 8 Bytes repräsentiert wird. 43

einfache Genauigkeit Gleitkommazahl, die nach IEEE 754 mit 4 Bytes repräsentiert wird. 11, 40, 43

Fully Dressed Ein Use Case im Fully Dressed Format beschreibt detailliert alle Schritte und Variationen. Siehe http://www.csci.csusb.edu/dick/cs375/fullydressed.html. 12

Model View Controller Pattern zur Unterteilung einer Applikation in Model, View und Controller. Das Model ist für Datenverwaltung zuständig. Die View kümmert sich um die Darstellung der Daten und die Interaktion mit dem Nutzer. Im Controller ist die Logik untergebracht. Der Controller empfängt Benutzereingaben von der View und ändert das Model entsprechend. Die Änderungen im Model sind anschliessend in der View sichtbar. 20, 23

OpenAPI Standard zur Beschreibung von REST-Schnittstellen. Siehe https://www.openapis.org/. 73, 77

Race Condition Race Conditions treten auf, wenn unsynchronisierte Schreib- und Lesezugriffe auf eine Variable durchgeführt werden. 47

Single-Page-Webanwendung Webanwendung, die aus einem einzelnen HTML-Dokument bestehen und deren Inhalte dynamisch nach bedarf nachgeladen werden. 20

Thrust Thrust ist eine Bibliothek, die Mechanismen zur Parallelisierung zur Verfügung stellt. Die API von Thrust ähnelt der C++ Standard Template Library. Siehe:https://thrust.github.io/doc. 11, 40

Unified Memory Unified Memory ist ein Feature von CUDA, dass es ermöglicht, Speicher zu allozieren, auf den von der CPU als auch von GPU aus zugegriffen werden kann. Damit entfällt das manuelle Kopieren von Daten von der CPU auf die GPU und umgekehrt. 40

Websocket Protokoll, dass eine bidirektionale Verbindung zwischen einem Client und einem Server ermöglicht. Im Gegensatz zu einer HTTP-Verbindung ist es damit dem Server möglich, Nachrichten unaufgefordert an den Client zu senden. 78

Akronyme

AMQP Advanced Message Queuing Protocol. 25

CUDA Compute Unified Device Architecture. 11, 13, 14, 16, 19, 24, 32, 40, 43, 55, 73

DTO Data Transfer Object. 75

GCC GNU Compiler Collection. 24

JSON JavaScript Object Notation. 29, 55, 73

MPI Message Passing Interface. 79

MSVC Microsoft Visual C++. 24, 55, 79

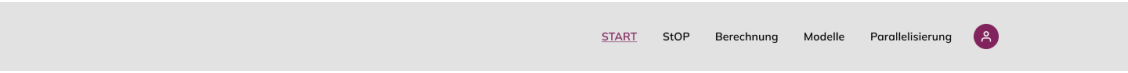
NVCC Nvidia CUDA Compiler. 24, 32

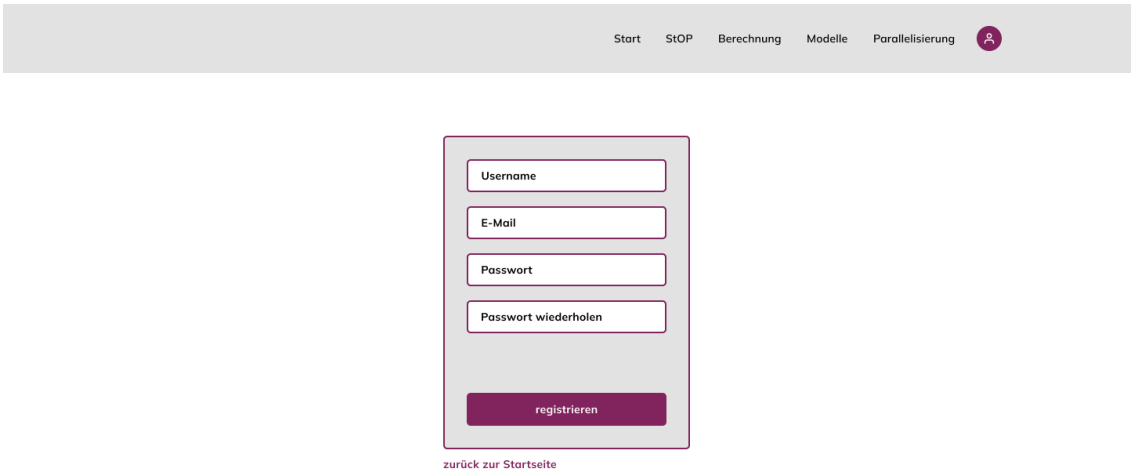
REST Representational State Transfer. 19, 20, 23, 31, 77

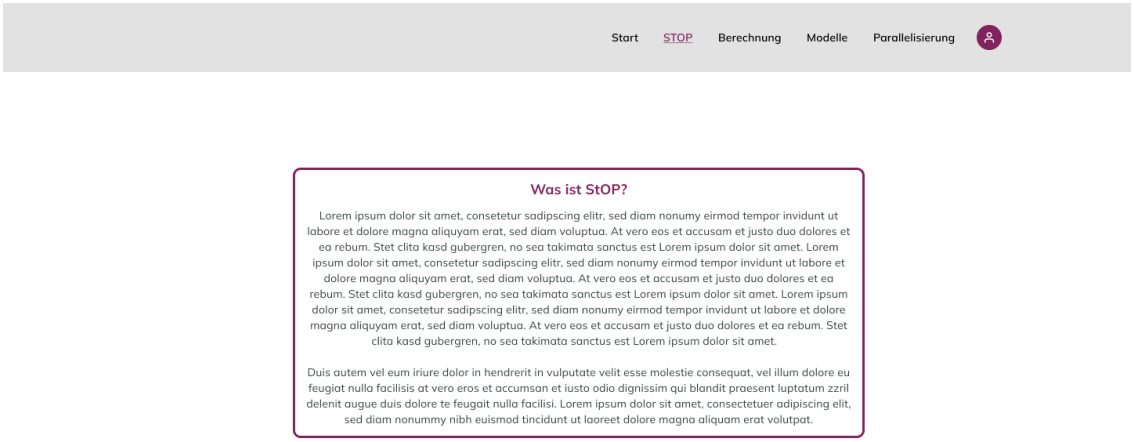
SIMD Single Instruction, Multiple Data. 44

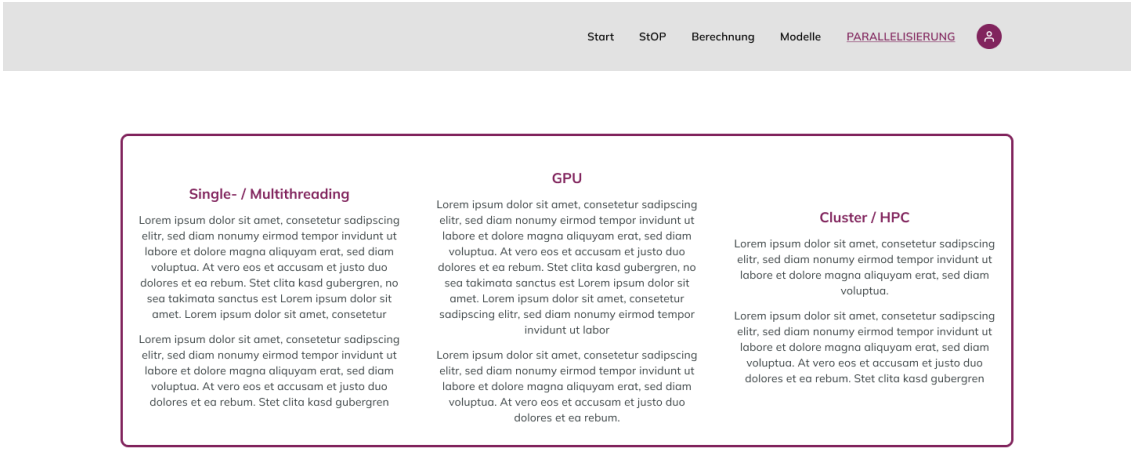
Anhang A

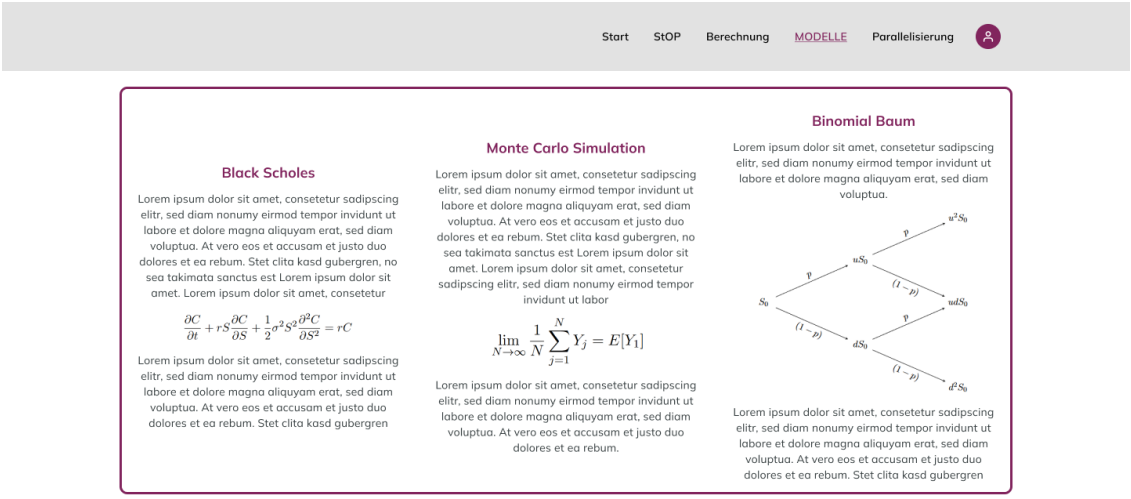
Wireframes

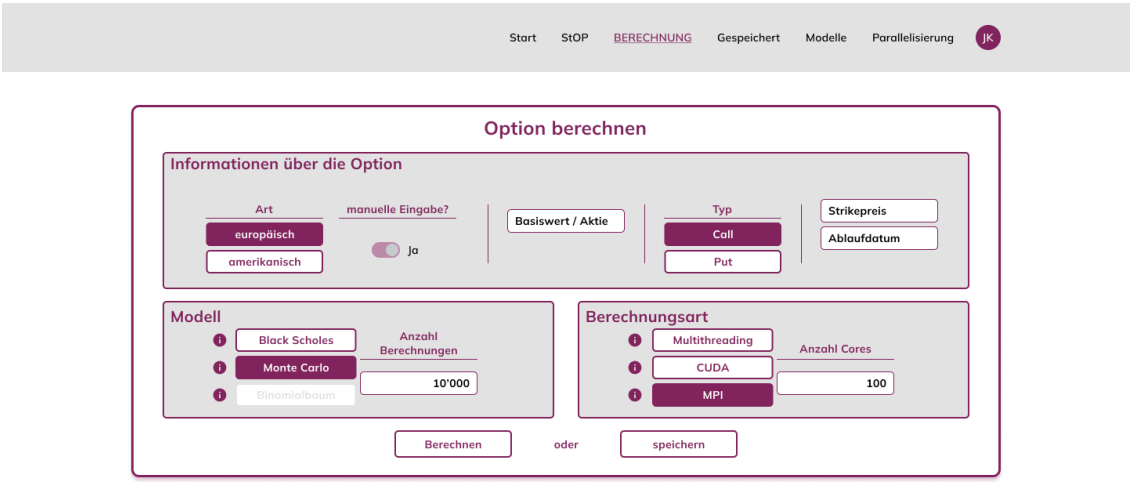


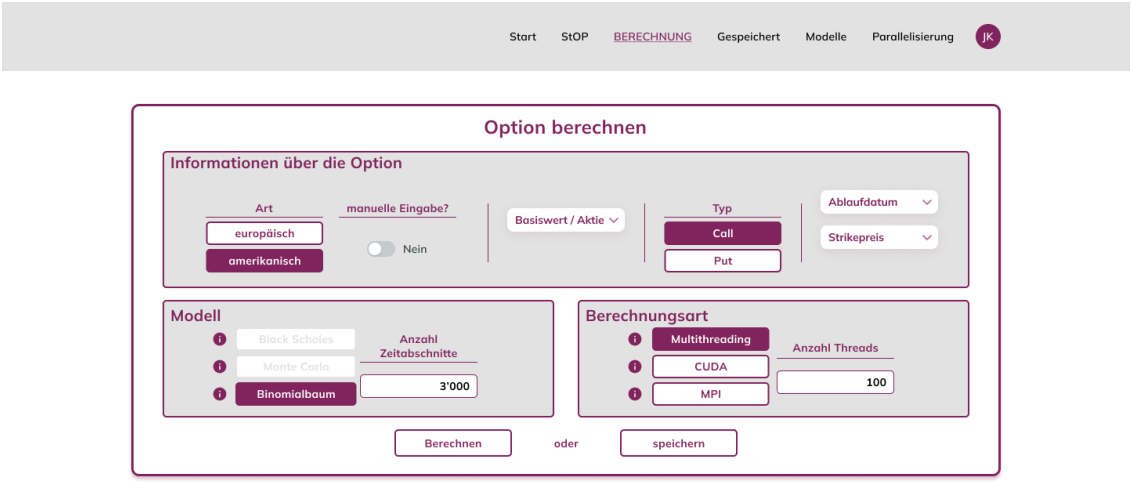

Start
StOP
Berechnung
Modelle
Parallelisierung
Username
E-Mail
Passwort
Passwort wiederholen
registrieren
zurück zur Startseite

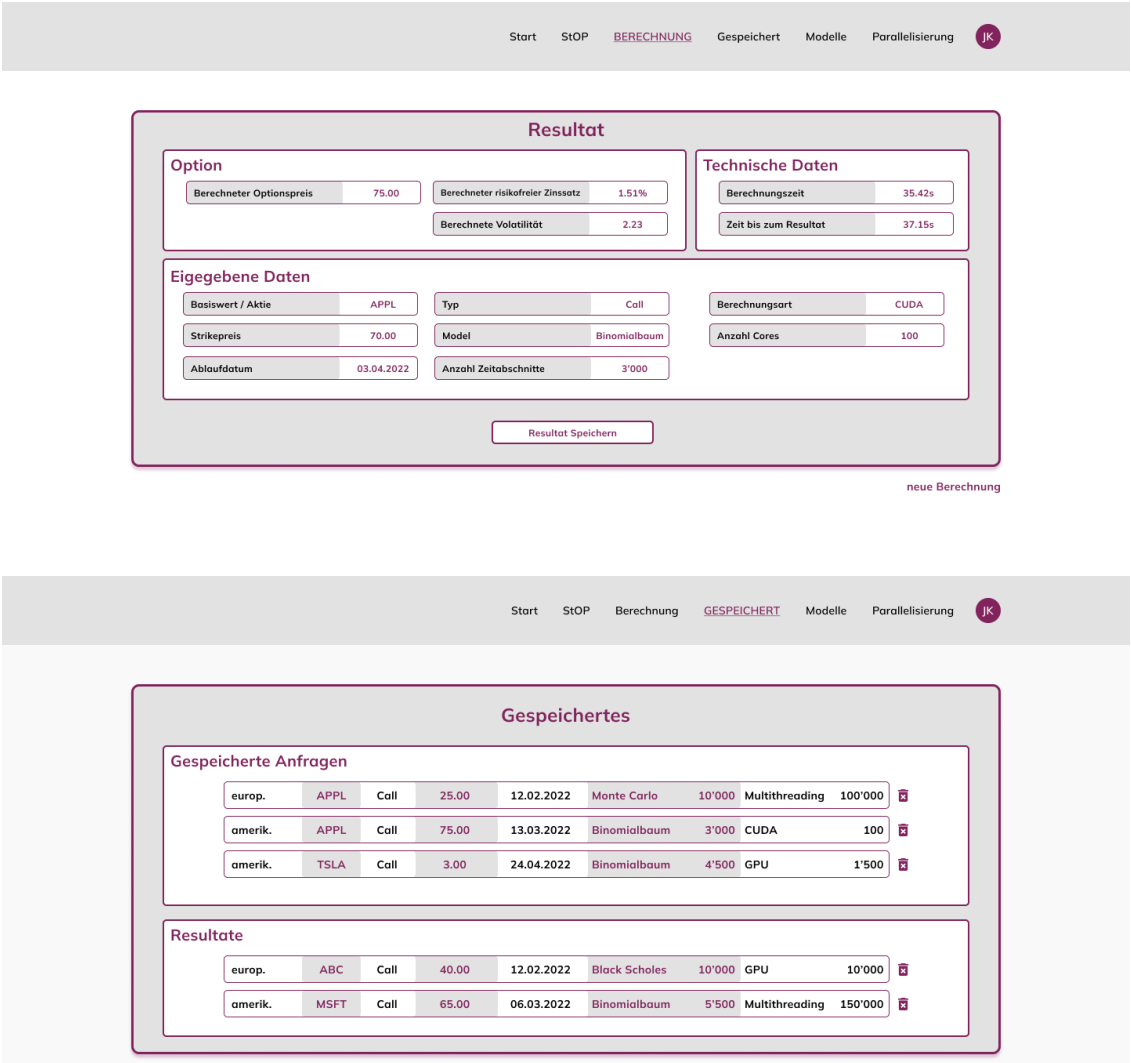

Start
STOP
Berechnung
Modelle
Parallelisierung
Was ist StOP?
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.
Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.


Start
StOP
Berechnung
Modelle
PARALLELISIERUNG
Single- / Multithreading
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren
GPU
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labor
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum.
Cluster / HPC
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren


Start
StOP
Berechnung
MODELLE
Parallelisierung
Black Scholes
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur
$\frac{\partial C}{\partial t} + rS\frac{\partial C}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} = rC$
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren
Monte Carlo Simulation
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labor
$\lim_{N\to\infty} \frac{1}{N} \sum_{j=1}^{N} Y_j = E[Y_1]$
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum.
Binomial Baum
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.
S_0
uS_0
dS_0
u^2S_0
udS_0
d^2S_0
p
$(1-p)$
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren


Start
StOP
BERECHNUNG
Gespeichert
Modelle
Parallelisierung
JK
Option berechnen
Informationen über die Option
Art
europäisch
amerikanisch
manuelle Eingabe?
Ja
Basiswert / Aktie
Typ
Call
Put
Strikepreis
Ablaufdatum
Modell
Black Scholes
Monte Carlo
Binomialbaum
Anzahl Berechnungen
10'000
Berechnungsart
Multithreading
CUDA
MPI
Anzahl Cores
100
Berechnen
oder
speichern


Start
StOP
BERECHNUNG
Gespeichert
Modelle
Parallelisierung
JK
Option berechnen
Informationen über die Option
Art
europäisch
amerikanisch
manuelle Eingabe?
Nein
Basiswert / Aktie
Typ
Call
Put
Ablaufdatum
Strikepreis
Modell
Black Scholes
Monte Carlo
Binomialbaum
Anzahl Zeitabschnitte
3'000
Berechnungsart
Multithreading
CUDA
MPI
Anzahl Threads
100
Berechnen
oder
speichern


Start
StOP
BERECHNUNG
Gespeichert
Modelle
Parallelisierung
JK
Resultat
Option
Berechneter Optionspreis
75.00
Berechneter risikofreier Zinssatz
1.51%
Berechnete Volatilität
2.23
Technische Daten
Berechnungszeit
35.42s
Zeit bis zum Resultat
37.15s
Eigegebene Daten
Basiswert / Aktie
APPL
Typ
Call
Berechnungsart
CUDA
Strikepreis
70.00
Model
Binomialbaum
Anzahl Cores
100
Ablaufdatum
03.04.2022
Anzahl Zeitabschnitte
3'000
Resultat Speichern
neue Berechnung

Start
StOP
Berechnung
GESPEICHERT
Modelle
Parallelisierung
JK
Gespeichertes
Gespeicherte Anfragen
europ. APPL Call 25.00 12.02.2022 Monte Carlo 10'000 Multithreading 100'000
amerik. APPL Call 75.00 13.03.2022 Binomialbaum 3'000 CUDA 100
amerik. TSLA Call 3.00 24.04.2022 Binomialbaum 4'500 GPU 1'500
Resultate
europ. ABC Call 40.00 12.02.2022 Black Scholes 10'000 GPU 10'000
amerik. MSFT Call 65.00 06.03.2022 Binomialbaum 5'500 Multithreading 150'000

Anhang B

Systemtests

B.1 Systemtest M4

Dieser Systemtest wurde mit dem Tag “system_test_01” am *09.05.2022* durchgeführt. Ziel dieses Systemtest ist es, möglichst alle Variationen von Usereingaben zu testen, um zu erkennen, ob alles wie erwartet funktioniert, oder noch Verbesserungen fällig sind.

B.1.1 Identity & Access Management

B.1.1.1 Valide Registrierung

Ablauf

1. Drücke den “einloggen”-Button
2. Drücke den “oder registrieren”-Button
3. Gebe als Username “Superuser” ein
4. Gebe als Passwort “12345OST” ein
5. Wiederhole das Passwort mit “12345OST”
6. Drücke den “registrieren”-Button

Erwartetes Ergebnis

- Ein neuer User ist in der Datenbank angelegt
- Das JWT ist gesetzt und der User angemeldet
- Der User wird zur Startseite weitergeleitet

Abweichungen zum erwarteten Ergebnis keine

B.1.1.2 Registrieren mit vorhandenem Usernamem

Ablauf

1. Drücke den “einloggen”-Button
2. Drücke den “oder registrieren”-Button

3. Gebe als Username “Superuser” ein
4. Gebe als Passwort “12345OST” ein
5. Wiederhole das Passwort mit “12345OST”
6. Drücke den “registrieren”-Button

Erwartetes Ergebnis

- User erhält Benachrichtigung, dass dieser Username bereits vergeben ist
- Kein neuer Eintrag in der Datenbank

Abweichungen zum erwarteten Ergebnis keine

B.1.1.3 Valides Login

Ablauf

1. Drücke den “einloggen”-Button
2. Gebe als Username “Superuser” ein
3. Gebe als Passwort “12345OST” ein
4. Drücke den “einloggen”-Button

Erwartetes Ergebnis

- Das JWT ist gesetzt und der User angemeldet
- Der User wird zur Startseite weitergeleitet

Abweichungen zum erwarteten Ergebnis keine

B.1.2 Optionen berechnen

B.1.2.1 Europäische Option mit Black-Scholes berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktien”-Feld die Aktie “APPL” ein
3. Gebe im “Ablaufdatum”-Feld das Datum “18.11.2022” ein
4. Gebe im “Strikepreis”-Feld den Wert “100” ein
5. Wähle bei den Modellen “Black Scholes” aus
6. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis keine

B.1.2.2 Europäische Option mit Monte-Carlo-Simulation berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktien”-Feld die Aktie “APPL” ein
3. Gebe im “Ablaufdatum”-Feld das Datum “18.11.2022” ein
4. Gebe im “Strikepreis”-Feld den Wert “100” ein
5. Wähle bei den Modellen “Monte Carlo” aus
6. Gebe für die Anzahl Berechnungen “1000” ein
7. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis keine

B.1.2.3 Europäische Option mit Multithreading berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktien”-Feld die Aktie “APPL” ein
3. Gebe im “Ablaufdatum”-Feld das Datum “18.11.2022” ein
4. Gebe im “Strikepreis”-Feld den Wert “100” ein
5. Wähle bei den Modellen “Monte Carlo” aus
6. Gebe für die Anzahl Berechnungen “1000” ein
7. Wähle bei den Berechnungsarten “Multithreading” aus
8. Gebe für die Anzahl Threads “8” ein
9. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis keine

B.1.2.4 Europäische Option mit CUDA berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktien”-Feld die Aktie “APPL” ein
3. Gebe im “Ablaufdatum”-Feld das Datum “18.11.2022” ein
4. Gebe im “Strikepreis”-Feld den Wert “100” ein
5. Wähle bei den Modellen “Monte Carlo” aus
6. Gebe für die Anzahl Berechnungen “1000” ein
7. Wähle bei den Berechnungsarten “CUDA” aus
8. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis keine

B.1.2.5 Amerikanische Option mit dem Binomialmodell berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Wähle als Optionsstyle “amerikanisch” aus
3. Gebe im “Aktien”-Feld die Aktie “APPL” ein
4. Gebe im “Ablaufdatum”-Feld das Datum “18.11.2022” ein
5. Gebe im “Strikepreis”-Feld den Wert “100” ein
6. Wähle bei den Modellen “Binomialbaum” aus
7. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis keine

B.1.2.6 Amerikanische Option mit Multithreading berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Wähle als Optionsstyle “amerikanisch” aus
3. Gebe im “Aktien”-Feld die Aktie “APPL” ein
4. Gebe im “Ablaufdatum”-Feld das Datum “18.11.2022” ein
5. Gebe im “Strikepreis”-Feld den Wert “100” ein
6. Wähle bei den Modellen “Binomialbaum” aus
7. Gebe für die Anzahl Berechnungen “1000” ein
8. Wähle bei den Berechnungsarten “Multithreading” aus
9. Gebe für die Anzahl Threads “8” ein
10. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis keine

B.1.2.7 Amerikanische Option mit CUDA berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Wähle als Optionsstyle “amerikanisch” aus
3. Gebe im “Aktien”-Feld die Aktie “APPL” ein
4. Gebe im “Ablaufdatum”-Feld das Datum “18.11.2022” ein
5. Gebe im “Strikepreis”-Feld den Wert “100” ein
6. Wähle bei den Modellen “Binomialbaum” aus
7. Gebe für die Anzahl Berechnungen “1000” ein
8. Wähle bei den Berechnungsarten “CUDA” aus
9. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis keine

B.1.3 User Management

B.1.3.1 Bewertungsanfragen speichern

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktien”-Feld die Aktie “APPL” ein
3. Gebe im “Ablaufdatum”-Feld das Datum “18.11.2022” ein
4. Gebe im “Strikepreis”-Feld den Wert “100” ein
5. Klicke auf den “speichern”-Button

B.1.3.2 Erwartetes Ergebnis

- Die aktuellen Eingaben werden in der Datenbank gespeichert
- Dem User wird eine Nachricht angezeigt, dass das Formular gespeichert wurde

Abweichungen zum erwarteten Ergebnis keine

B.1.3.3 Bewertungsresultate speichern

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktien”-Feld die Aktie “APPL” ein
3. Gebe im “Ablaufdatum”-Feld das Datum “18.11.2022” ein
4. Gebe im “Strikepreis”-Feld den Wert “100” ein
5. Klicke auf den “Berechnen”-Button
6. Klicke auf den “speichern”-Button

Erwartetes Ergebnis

- Das Resultat wird in der Datenbank gespeichert
- Dem User wird eine Nachricht angezeigt, dass das Formular gespeichert wurde

Abweichungen zum erwarteten Ergebnis keine

B.1.4 Gespeichertes ansehen

B.1.4.1 Gespeicherte Bewertungsanfragen ansehen

Ablauf

1. Drücke den “Gespeichert”-Button in der NavBar
2. Wähle eine Eingabe aus der Liste aus

Erwartetes Ergebnis

- Der User wird auf die Berechnungsseite weitergeleitet
- Die Felder sind gemäss den gespeicherten Eingaben ausgefüllt

Abweichungen zum erwarteten Ergebnis keine

B.1.4.2 Gespeichertes Bewertungsresultat ansehen

Ablauf

1. Drücke den “Gespeichert”-Button in der NavBar
2. Wähle ein Resultat aus der Liste aus

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
- Die Felder sind gemäss dem gespeicherten Resultat ausgefüllt

Abweichungen zum erwarteten Ergebnis keine

B.1.5 Gespeichertes löschen

B.1.5.1 Gespeicherte Bewertungsanfrage löschen

Ablauf

1. Drücke den “Gespeichert”-Button in der NavBar
2. Wähle das Müllkorb Icon einer Eingabe aus der Liste aus, um diese zu löschen

Erwartetes Ergebnis

- Der Eintrag in der Datenbank wird gelöscht
- Dem User wird eine Nachricht angezeigt, dass die Eingabe gelöscht wurde

Abweichungen zum erwarteten Ergebnis keine

B.1.5.2 Gespeichertes Bewertungsresultat löschen

Ablauf

1. Drücke den “Gespeichert”-Button in der NavBar
2. Wähle das Müllkorb Icon eines Resultats aus der Liste aus, um dieses zu löschen

Erwartetes Ergebnis

- Der Eintrag in der Datenbank wird gelöscht
- Dem User wird eine Nachricht angezeigt, dass das Resultat gelöscht wurde

Abweichungen zum erwarteten Ergebnis keine

B.2 Systemtest M5

Dieser Systemtest wurde mit dem Tag “systemtestM5” am *05.06.2022* durchgeführt. Ziel dieses Systemtest ist es, möglichst alle Variationen von Usereingaben zu testen, um zu erkennen, ob alles wie erwartet funktioniert, oder noch Verbesserungen fällig sind. Bei jedem einzelnen Test wir davon ausgegangen dass der User gerade erst auf die Applikation gekommen ist.

B.2.1 Identity & Access Management

B.2.1.1 Valide Registrierung

Ablauf

1. Drücke den “einloggen”-Button
2. Drücke den “oder registrieren”-Button
3. Gebe als Username “Superuser” ein
4. Gebe als Passwort “12345OST” ein
5. Wiederhole das Passwort mit “12345OST”
6. Drücke den “registrieren”-Button

Erwartetes Ergebnis

- Ein neuer User ist in der Datenbank angelegt
- Das JWT ist gesetzt und der User angemeldet
- Der User wird zur Startseite weitergeleitet

Abweichungen zum erwarteten Ergebnis keine

B.2.1.2 Registrieren mit vorhandenem Username

Ablauf

1. Drücke den “einloggen”-Button
2. Drücke den “oder registrieren”-Button
3. Gebe als Username “Superuser” ein
4. Gebe als Passwort “12345OST” ein
5. Wiederhole das Passwort mit “12345OST”
6. Drücke den “registrieren”-Button

Erwartetes Ergebnis

- User erhält Benachrichtigung, dass dieser Username bereits vergeben ist
- Kein neuer Eintrag in der Datenbank

Abweichungen zum erwarteten Ergebnis keine

B.2.1.3 Valides Login

Ablauf

1. Drücke den “einloggen”-Button
2. Gebe als Username “Superuser” ein
3. Gebe als Passwort “12345OST” ein
4. Drücke den “einloggen”-Button

Erwartetes Ergebnis

- Das JWT ist gesetzt und der User angemeldet
- Der User wird zur Startseite weitergeleitet

Abweichungen zum erwarteten Ergebnis keine

B.2.1.4 Logout

Ablauf

1. Drücke den “einloggen”-Button
2. Gebe als Username “Superuser” ein
3. Gebe als Passwort “12345OST” ein
4. Drücke den “einloggen”-Button
5. Drücke auf den Avatar (oben rechts)
6. Drücke den “ausloggen”-Button

Erwartetes Ergebnis

- Das JWT wird im Backend entfernt und der User abgemeldet
- Der User wird zur Loginseite weitergeleitet

Abweichungen zum erwarteten Ergebnis keine

B.2.2 Einzelne Option berechnen

Zur Überprüfung, ob die Resultate einer berechneten Option plausibel sind, werden diese mit dem online Calculator von hoadley[1] verglichen. Die Option ist daher für alle Tests dieselbe:

- Aktie: **AAPL**
- Basiswertpreis zum Bewertungszeitpunkt: **145.38** (zum Zeitpunkt dieser Durchführung aktuell)

[1] Siehe: https://www.hoadley.net/options/binomialtree.aspx?tree=B

- Fälligkeitsdatum: **18.11.2022**
- Strike Preis: **100**
- Volatilität: **38.59**
- Risikofreier Zinssatz: **1.14**
- Anzahl Tage bis zur Fälligkeit: **166** (05.06 - 18.11.2022)

Der daraus Resultierende Optionswert beträgt: **45.38**

B.2.2.1 Europäische Option mit Black-Scholes berechnen

Ablauf

1. Drücke den "Berechnen"-Button in der NavBar
2. Gebe im "Aktien"-Feld die Aktie "APPL" ein
3. Wähle im "Ablaufdatum"-Feld das Datum "18.11.2022" aus
4. Gebe im "Strikepreis"-Feld den Wert "100" ein
5. Wähle bei den Modellen "Black Scholes" aus
6. Klicke auf den "Berechnen"-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
  - Bei den "Eingegebenen Daten" sind die Angaben gemäss den Eingaben
  - Der Optionswert ist im Bereich des Beispielwertes ± 3
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis Resultiertes Ergebnis: **46.9**

B.2.2.2 Europäische Option mit Monte-Carlo-Simulation berechnen

Ablauf

1. Drücke den "Berechnen"-Button in der NavBar
2. Gebe im "Aktien"-Feld die Aktie "APPL" ein
3. Wähle im "Ablaufdatum"-Feld das Datum "18.11.2022" aus
4. Gebe im "Strikepreis"-Feld den Wert "100" ein
5. Wähle bei den Modellen "Monte Carlo" aus
6. Gebe für die Anzahl Berechnungen "1000" ein
7. Klicke auf den "Berechnen"-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
  - Bei den “Eingegebenen Daten” sind die Angaben gemäss den Eingaben
  - Der Optionswert ist im Bereich des Beispielwertes ± 3
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis Resultiertes Ergebnis: **46.75**

B.2.2.3 Europäische Option mit Multithreading berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktien”-Feld die Aktie “APPL” ein
3. Wähle im “Ablaufdatum”-Feld das Datum “18.11.2022” aus
4. Gebe im “Strikepreis”-Feld den Wert “100” ein
5. Wähle bei den Modellen “Monte Carlo” aus
6. Gebe für die Anzahl Berechnungen “1000” ein
7. Wähle bei den Berechnungsarten “Multithreading” aus
8. Gebe für die Anzahl Threads “8” ein
9. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
  - Bei den “Eingegebenen Daten” sind die Angaben gemäss den Eingaben
  - Der Optionswert ist im Bereich des Beispielwertes ± 3
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis Resultiertes Ergebnis: **46.04**

B.2.2.4 Europäische Option mit CUDA berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktien”-Feld die Aktie “APPL” ein
3. Wähle im “Ablaufdatum”-Feld das Datum “18.11.2022” aus
4. Gebe im “Strikepreis”-Feld den Wert “100” ein

5. Wähle bei den Modellen “Monte Carlo” aus
6. Gebe für die Anzahl Berechnungen “1000” ein
7. Wähle bei den Berechnungsarten “CUDA” aus
8. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
  - Bei den “Eingegebenen Daten” sind die Angaben gemäss den Eingaben
  - Der Optionswert ist im Bereich des Beispielwertes ± 3
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis Resultiertes Ergebnis: **46.53**

B.2.2.5 Europäische Option mit benutzerdefinierter Volatilität berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktie”-Feld die Aktie “APPL” ein
3. Wähle im “Ablaufdatum”-Feld das Datum “18.11.2022” aus
4. Gebe im “Strikepreis”-Feld den Wert “100” ein
5. Drücke bei “Volatilität” auf “Benutzerdefiniert”
6. Gebe im “Volatilität”-Feld den Wert “42” ein
7. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
  - Bei den “Eingegebenen Daten” sind die Angaben gemäss den Eingaben
  - Der Optionswert ist im Bereich von **45.38** ± 3
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis Resultiertes Ergebnis: **47.31**

B.2.2.6 Europäische Option mit benutzerdefinierter historischer Volatilität berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktie”-Feld die Aktie “APPL” ein
3. Wähle im “Ablaufdatum”-Feld das Datum “18.11.2022” aus
4. Gebe im “Strikepreis”-Feld den Wert “100” ein
5. Drücke bei “Volatilität” auf “Fix Historisch”
6. Gebe im “Anz. Tage zurückgehen”-Feld den Wert “250” ein
7. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
  - Bei den “Eingegebenen Daten” sind die Angaben gemäss den Eingaben
  - Der Optionswert ist im Bereich des Beispielwertes ± 3
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis Resultiertes Ergebnis: **48.79** (gegenüber 45.38)

B.2.2.7 Europäische Option mit historischer Volatilität berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktie”-Feld die Aktie “APPL” ein
3. Wähle im “Ablaufdatum”-Feld das Datum “18.11.2022” aus
4. Gebe im “Strikepreis”-Feld den Wert “100” ein
5. Drücke bei “Volatilität” auf “Historisch”
6. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
  - Bei den “Eingegebenen Daten” sind die Angaben gemäss den Eingaben
  - Der Optionswert ist im Bereich des Beispielwertes ± 3
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis Resultiertes Ergebnis: **46.9**

B.2.2.8 Europäische Option mit berechnetem risikofreien Zinssatz berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktie”-Feld die Aktie “APPL” ein
3. Wähle im “Ablaufdatum”-Feld das Datum “18.11.2022” aus
4. Gebe im “Strikepreis”-Feld den Wert “100” ein
5. Drücke bei “Risikofreier Zinssatz” auf “Berechnet”
6. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
  - Bei den “Eingegebenen Daten” sind die Angaben gemäss den Eingaben
  - Der Optionswert ist im Bereich des Beispielwertes ± 3
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis Resultiertes Ergebnis: **46.9**

B.2.2.9 Europäische Option mit benutzerdefinierten risikofreien Zinssatz berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktie”-Feld die Aktie “APPL” ein
3. Wähle im “Ablaufdatum”-Feld das Datum “18.11.2022” aus
4. Gebe im “Strikepreis”-Feld den Wert “100” ein
5. Drücke bei “Risikofreier Zinssatz” auf “Benutzerdefiniert”
6. Gebe im “Risikofreier Zinssatz (in Prozent)”-Feld den Wert “13” ein
7. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
  - Bei den “Eingegebenen Daten” sind die Angaben gemäss den Eingaben
  - Der Optionswert ist im Bereich des Beispielwertes ± 3
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis Resultiertes Ergebnis: **51.72** (gegenüber 45.38)

B.2.2.10 Leeres Aktienfeld

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktie”-Feld die Aktie “” ein (nichts eingeben)
3. Klicke auf den “Berechnen”- oder “speichern”-Button

Erwartetes Ergebnis

- Das “Aktien”-Feld ist rot markiert
- Beim Versuch zu speichern/berechnen bekommt der User eine Meldung, dass die eingegebene Aktie ungültig ist

Abweichungen zum erwarteten Ergebnis keine

B.2.2.11 Ablaufdatum in der Vergangenheit angeben

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktie”-Feld die Aktie “APPL” ein
3. Wähle im “Ablaufdatum”-Feld das Datum “01.05.2022” aus
4. Gebe im “Strikepreis”-Feld den Wert “100” ein
5. Klicke auf den “Berechnen”- oder “speichern”-Button

Erwartetes Ergebnis

- Das “Ablaufdatum”-Feld ist rot markiert
- Beim Versuch zu speichern/berechnen bekommt der User eine Meldung, dass das eingegebene Ablaufdatum ungültig ist

Abweichungen zum erwarteten Ergebnis keine

B.2.2.12 Strikepreis ≤ 0

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktie”-Feld die Aktie “APPL” ein
3. Wähle im “Ablaufdatum”-Feld das Datum “18.11.2022” aus
4. Gebe im “Strikepreis”-Feld den Wert “0” ein
5. Klicke auf den “Berechnen”- oder “speichern”-Button

Erwartetes Ergebnis

- Das “Strikepreis”-Feld ist rot markiert
- Beim Versuch zu speichern/berechnen bekommt der User eine Meldung, dass der eingegebene Strike Preis ungültig ist

Abweichungen zum erwarteten Ergebnis keine

B.2.2.13 Amerikanische Option mit Yahoo Finance Daten berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Wähle als Optionsstyle “amerikanisch” aus
3. Schalte den Switcher bei “Manuelle Eingabe” aus
4. Gebe im “Aktien”-Feld “aa” ein und wähle aus der Liste “APPL” aus
5. Wähle beim “Ablaufdatum”-Feld das Datum “18.11.2022” aus
6. Wähle beim “Strikepreis”-Feld den Wert “100” aus
7. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
  - Bei den “Eingegebenen Daten” sind die Angaben gemäss den Eingaben
  - Der Optionswert ist im Bereich des Beispielwertes ± 3
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis Resultiertes Ergebnis: **45.9**

B.2.2.14 Amerikanische Option mit dem Binomialbaum berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Wähle als Optionsstyle “amerikanisch” aus
3. Gebe im “Aktien”-Feld die Aktie “APPL” ein
4. Wähle im “Ablaufdatum”-Feld das Datum “18.11.2022” aus
5. Gebe im “Strikepreis”-Feld den Wert “100” ein
6. Wähle bei den Modellen “Binomialbaum” aus
7. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
  - Bei den “Eingegebenen Daten” sind die Angaben gemäss den Eingaben
  - Der Optionswert ist im Bereich des Beispielwertes ± 3
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis Resultiertes Ergebnis: **45.9**

B.2.2.15 Amerikanische Option mit Multithreading berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Wähle als Optionsstyle “amerikanisch” aus
3. Gebe im “Aktien”-Feld die Aktie “APPL” ein
4. Wähle im “Ablaufdatum”-Feld das Datum “18.11.2022” aus
5. Gebe im “Strikepreis”-Feld den Wert “100” ein
6. Wähle bei den Modellen “Binomialbaum” aus
7. Gebe für die Anzahl Berechnungen “1000” ein
8. Wähle bei den Berechnungsarten “Multithreading” aus
9. Gebe für die Anzahl Threads “8” ein
10. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
  - Bei den “Eingegebenen Daten” sind die Angaben gemäss den Eingaben
  - Der Optionswert ist im Bereich des Beispielwertes ± 3
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis Resultiertes Ergebnis: **45.9**

B.2.2.16 Amerikanische Option mit CUDA berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Wähle als Optionsstyle “amerikanisch” aus
3. Gebe im “Aktien”-Feld die Aktie “APPL” ein
4. Wähle im “Ablaufdatum”-Feld das Datum “18.11.2022” aus
5. Gebe im “Strikepreis”-Feld den Wert “100” ein
6. Wähle bei den Modellen “Binomialbaum” aus
7. Gebe für die Anzahl Berechnungen “1000” ein
8. Wähle bei den Berechnungsarten “CUDA” aus
9. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
  - Bei den “Eingegebenen Daten” sind die Angaben gemäss den Eingaben
  - Der Optionswert ist im Bereich des Beispielwertes ± 3
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis Resultiertes Ergebnis: **45.53**

B.2.2.17 Amerikanische Option mit impliziter Volatilität berechnen

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Wähle als Optionsstyle “amerikanisch” aus
3. Schalte den Switcher bei “Manuelle Eingabe” aus
4. Gebe im “Aktien”-Feld “aa” ein und wähle aus der Liste “APPL” aus
5. Wähle beim “Ablaufdatum”-Feld das Datum “18.11.2022” aus
6. Wähle beim “Strikepreis”-Feld den Wert “100” aus
7. Drücke bei “Volatilität” auf “Implizit”
8. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
  - Bei den “Eingegebenen Daten” sind die Angaben gemäss den Eingaben
  - Der Optionswert ist im Bereich des Beispielwertes ± 3
- Dem User werden die Resultatwerte der Berechnung angezeigt

Abweichungen zum erwarteten Ergebnis Resultiertes Ergebnis: **45.9**

B.2.3 User Management

B.2.3.1 Bewertungsanfragen speichern

Ablauf

1. Drücke den “Berechnen”-Button in der NavBar
2. Gebe im “Aktien”-Feld die Aktie “APPL” ein
3. Wähle im “Ablaufdatum”-Feld das Datum “18.11.2022” aus
4. Gebe im “Strikepreis”-Feld den Wert “100” ein
5. Klicke auf den “speichern”-Button

B.2.3.2 Erwartetes Ergebnis

- Die aktuellen Eingaben werden in der Datenbank gespeichert
- Dem User wird eine Nachricht angezeigt, dass das Formular gespeichert wurde

Abweichungen zum erwarteten Ergebnis keine

B.2.3.3 Bewertungsresultate speichern

Ablauf

1. Drücke den "Berechnen"-Button in der NavBar
2. Gebe im "Aktien"-Feld die Aktie "APPL" ein
3. Wähle im "Ablaufdatum"-Feld das Datum "18.11.2022" aus
4. Gebe im "Strikepreis"-Feld den Wert "100" ein
5. Klicke auf den "Berechnen"-Button
6. Klicke auf den "speichern"-Button

Erwartetes Ergebnis

- Das Resultat wird in der Datenbank gespeichert
- Dem User wird eine Nachricht angezeigt, dass das Formular gespeichert wurde

Abweichungen zum erwarteten Ergebnis keine

B.2.4 Gespeichertes ansehen

B.2.4.1 Gespeicherte Bewertungsanfrage ansehen

Ablauf

1. Drücke den "Gespeichert"-Button in der NavBar
2. Wähle eine Eingabe aus der Liste aus

Erwartetes Ergebnis

- Der User wird auf die Berechnungsseite weitergeleitet
- Die Felder sind gemäss den gespeicherten Eingaben ausgefüllt

Abweichungen zum erwarteten Ergebnis keine

B.2.4.2 Gespeichertes Bewertungsresultat ansehen

Ablauf

1. Drücke den "Gespeichert"-Button in der NavBar
2. Wähle ein Resultat aus der Liste aus

Erwartetes Ergebnis

- Der User wird auf die Resultatseite weitergeleitet
- Die Felder sind gemäss dem gespeicherten Resultat ausgefüllt

Abweichungen zum erwarteten Ergebnis keine

B.2.5 Gespeichertes löschen

B.2.5.1 Gespeicherte Bewertungsanfrage löschen

Ablauf

1. Drücke den "Gespeichert"-Button in der NavBar
2. Wähle das Müllkorb Icon einer Eingabe aus der Liste aus, um diese zu löschen

Erwartetes Ergebnis

- Der Eintrag in der Datenbank wird gelöscht
- Dem User wird eine Nachricht angezeigt, dass die Eingabe gelöscht wurde

Abweichungen zum erwarteten Ergebnis keine

B.2.5.2 Gespeichertes Bewertungsresultat löschen

Ablauf

1. Drücke den "Gespeichert"-Button in der NavBar
2. Wähle das Müllkorb Icon eines Resultats aus der Liste aus, um dieses zu löschen

Erwartetes Ergebnis

- Der Eintrag in der Datenbank wird gelöscht
- Dem User wird eine Nachricht angezeigt, dass das Resultat gelöscht wurde

Abweichungen zum erwarteten Ergebnis keine

B.2.6 Profit-Spotter

B.2.6.1 Aktienoptionen mit einfacher Präzision berechnen

Ablauf

1. Drücke den “Profit Spotter”-Button in der NavBar
2. Gebe im “Aktien”-Feld “aa” ein und wähle aus der Liste “APPL” aus
3. Wähle im “Präzision”-Feld “31 Zeitabschnitte” aus
4. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Nach ein paar Sekunden Wartezeit wird der User auf die Resultatseite weitergeleitet
- Dem User wird eine Liste von Optionen angezeigt, die er nach Belieben sortieren kann

Abweichungen zum erwarteten Ergebnis keine

B.2.6.2 Aktienoptionen mit doppelter Präzision berechnen

Ablauf

1. Drücke den “Profit Spotter”-Button in der NavBar
2. Gebe im “Aktien”-Feld “aa” ein und wähle aus der Liste “APPL” aus
3. Wähle im “Präzision”-Feld “63 Zeitabschnitte” aus
4. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Nach ein paar Sekunden Wartezeit wird der User auf die Resultatseite weitergeleitet
- Dem User wird eine Liste von Optionen angezeigt, die er nach Belieben sortieren kann

Abweichungen zum erwarteten Ergebnis keine

B.2.6.3 Aktienoptionen mit Multithreading berechnen

Ablauf

1. Drücke den “Profit Spotter”-Button in der NavBar
2. Gebe im “Aktien”-Feld “aa” ein und wähle aus der Liste “APPL” aus
3. Wähle im “Präzision”-Feld “31 Zeitabschnitte” aus
4. Wähle bei der Berechnungsart “Multithreading aus”
5. Gebe bei den “Anz. Threads” 8 ein

6. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Nach ein paar Sekunden Wartezeit wird der User auf die Resultatseite weitergeleitet
- Dem User wird eine Liste von Optionen angezeigt, die er nach Belieben sortieren kann

Abweichungen zum erwarteten Ergebnis keine

B.2.6.4 Aktienoptionen mit CUDA berechnen

Ablauf

1. Drücke den “Profit Spotter”-Button in der NavBar
2. Gebe im “Aktien”-Feld “aa” ein und wähle aus der Liste “APPL” aus
3. Wähle im “Präzision”-Feld “31 Zeitabschnitte” aus
4. Wähle bei der Berechnungsart “CUDA aus”
5. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Nach ein paar Sekunden Wartezeit wird der User auf die Resultatseite weitergeleitet
- Dem User wird eine Liste von Optionen angezeigt, die er nach Belieben sortieren kann

Abweichungen zum erwarteten Ergebnis keine

B.2.6.5 Aktienoptionen mit benutzerdefinierter Volatilität berechnen

Ablauf

1. Drücke den “Profit Spotter”-Button in der NavBar
2. Gebe im “Aktien”-Feld “aa” ein und wähle aus der Liste “APPL” aus
3. Drücke bei “Volatilität” auf “Benutzerdefiniert”
4. Gebe im “Volatilität”-Feld den Wert “42” ein
5. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Nach ein paar Sekunden Wartezeit wird der User auf die Resultatseite weitergeleitet
- Dem User wird eine Liste von Optionen angezeigt, die er nach Belieben sortieren kann

Abweichungen zum erwarteten Ergebnis keine

B.2.6.6 Aktienoptionen mit impliziter Volatilität berechnen

Ablauf

1. Drücke den “Profit Spotter”-Button in der NavBar
2. Gebe im “Aktien”-Feld “aa” ein und wähle aus der Liste “APPL” aus
3. Drücke bei “Volatilität” auf “Implizit”
4. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Nach ein paar Sekunden Wartezeit wird der User auf die Resultatseite weitergeleitet
- Dem User wird eine Liste von Optionen angezeigt, die er nach Belieben sortieren kann

Abweichungen zum erwarteten Ergebnis keine

B.2.6.7 Aktienoptionen mit benutzerdefinierter historischer Volatilität berechnen

Ablauf

1. Drücke den “Profit Spotter”-Button in der NavBar
2. Gebe im “Aktien”-Feld “aa” ein und wähle aus der Liste “APPL” aus
3. Drücke bei “Volatilität” auf “Fix Historisch”
4. Gebe im “Anz. Tage zurückgehen”-Feld den Wert “250” ein
5. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Nach ein paar Sekunden Wartezeit wird der User auf die Resultatseite weitergeleitet
- Dem User wird eine Liste von Optionen angezeigt, die er nach Belieben sortieren kann

Abweichungen zum erwarteten Ergebnis keine

B.2.6.8 Aktienoptionen mit historischer Volatilität berechnen

Ablauf

1. Drücke den “Profit Spotter”-Button in der NavBar
2. Gebe im “Aktien”-Feld “aa” ein und wähle aus der Liste “APPL” aus
3. Drücke bei “Volatilität” auf “Historisch”
4. Klicke auf den “Berechnen”-Button

Erwartetes Ergebnis

- Nach ein paar Sekunden Wartezeit wird der User auf die Resultatseite weitergeleitet
- Dem User wird eine Liste von Optionen angezeigt, die er nach Belieben sortieren kann

Abweichungen zum erwarteten Ergebnis keine

B.2.6.9 Aktienoptionen mit berechnetem risikofreien Zinssatz berechnen

Ablauf

1. Drücke den "Profit Spotter"-Button in der NavBar
2. Gebe im "Aktien"-Feld "aa" ein und wähle aus der Liste "APPL" aus
3. Drücke bei "Risikofreier Zinssatz" auf "Berechnet"
4. Klicke auf den "Berechnen"-Button

Erwartetes Ergebnis

- Nach ein paar Sekunden Wartezeit wird der User auf die Resultatseite weitergeleitet
- Dem User wird eine Liste von Optionen angezeigt, die er nach Belieben sortieren kann

Abweichungen zum erwarteten Ergebnis keine

B.2.6.10 Aktienoptionen mit benutzerdefinierten risikofreien Zinssatz berechnen

Ablauf

1. Drücke den "Profit Spotter"-Button in der NavBar
2. Gebe im "Aktien"-Feld "aa" ein und wähle aus der Liste "APPL" aus
3. Drücke bei "Risikofreier Zinssatz" auf "Benutzerdefiniert"
4. Gebe im "Risikofreier Zinssatz (in Prozent)"-Feld den Wert "13" ein
5. Klicke auf den "Berechnen"-Button

Erwartetes Ergebnis

- Nach ein paar Sekunden Wartezeit wird der User auf die Resultatseite weitergeleitet
- Dem User wird eine Liste von Optionen angezeigt, die er nach Belieben sortieren kann

Abweichungen zum erwarteten Ergebnis keine

Anhang C

Schnittstellenbeschreibungen

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

API Reference

StOP NestJS REST API

API Version: 1.0.0

This resource describes the StOP NestJS REST API which is used by the React SPA.

INDEX

API

1. AUTHENTICATION

1.1 POST /register

REQUEST

REQUEST BODY - application/json

```
{
  username string 1 to 20 chars
  password string 8 to 20 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  max:20 chars
}
```

STATUS CODE - 400: Username already in use or password does not meet requirements.

1.2 POST /login

REQUEST

REQUEST BODY - application/json

```
{
  username string 1 to 20 chars
  password string 8 to 20 chars
}
```

RESPONSE

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

RESPONSE MODEL - application/json

```
{
  id       integer
  username string  max:20 chars
}
```

STATUS CODE - 401: Unsuccessfully authenticated

1.3 GET /loginState

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200: Description

RESPONSE MODEL - application/json

```
{
  isLoggedIn  boolean
}
```

1.4 POST /logout

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200: Successfully logged out

STATUS CODE - 401: Requesting user was not logged in

---

2. PERSISTENCE

2.1 GET /option/{id}

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*id	number >=0	

RESPONSE

STATUS CODE - 200: Returns the option with the given id

RESPONSE MODEL - application/json

```
{
  id              integer  >=0
  exerciseStyle   enum     ALLOWED:AMERICAN, EUROPEAN
  stockSymbol     string
  strikePrice     integer  >=0
  expirationDate  string
  type            enum     ALLOWED:CALL, PUT
}
```

STATUS CODE - 400: Invalid id

2.2 GET /valuation-input

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200: Returns all saved ValuationInputs for the authenticated user.

RESPONSE MODEL - application/json

```
{
  savedValuationInputs [{
  Array of object:
    id                      number                  >=0
    optionId                number                  >=0
    model
      ONE OF
      OPTION 1 {
        modelName  enum  ALLOWED:BLACKSCHOLES
      }
      OPTION 2 {
        modelName           enum     ALLOWED:MONTECARLO
        numberOfPricePaths  integer  >=1
      }
```

```
      OPTION 3{
        modelName          enum     ALLOWED:BINOMIALTREE
        numberOfTimesteps  integer  >=1
      }
    parallelization
      ONE OF
      OPTION 1{
        parallelizationName  enum  ALLOWED:NOPARALLELIZATION
      }
      OPTION 2{
        parallelizationName  enum     ALLOWED:MULTITHREADING
        numberThreads        integer  >=1
      }
      OPTION 3{
        parallelizationName   enum   ALLOWED:CUDA
        gpu {
          gpuName                      string
          numberOfStreamingProcessors  integer
        }
      }
    riskFreeRate
      ONE OF
      OPTION 1{
        riskFreeRateName  enum    ALLOWED:CUSTOMRISKFREERATE
        riskFreeRate      number  between -1 and  1
      }
      OPTION 2{
        riskFreeRateName  enum  ALLOWED:TREASURYBILLRISKFREERATE
      }
    volatility
      ONE OF
      OPTION 1{
        volatilityName  enum    ALLOWED:CUSTOMVOLATILITY
        volatility      number  between 0 and  1
      }
      OPTION 2{
        volatilityName  enum  ALLOWED:IMPLIEDVOLATILITY
      }
      OPTION 3{
        volatilityName  enum    ALLOWED:HISTORICALFIXEDVOLATILITY
        daysBack        number  between 3 and  1825
      }
      OPTION 4{
        volatilityName  enum  ALLOWED:HISTORICALEXPIRATIONVOLATILITY
      }
  }]
}
```

2.3 POST /valuation-input

REQUEST

REQUEST BODY - application/json

```
{
  option {
    exerciseStyle  enum     ALLOWED:AMERICAN, EUROPEAN
```

```
  stockSymbol     string
  strikePrice     integer  between 0 and  10000
  expirationDate  string   Minimum: today, maximum: today + 1825 days (about 5 years)
  type            enum     ALLOWED:CALL, PUT
}
model
  ONE OF
  OPTION 1{
    modelName  enum  ALLOWED:BLACKSCHOLES
  }
  OPTION 2{
    modelName           enum     ALLOWED:MONTECARLO
    numberOfPricePaths  integer  >=1
  }
  OPTION 3{
    modelName          enum     ALLOWED:BINOMIALTREE
    numberOfTimesteps  integer  >=1
  }
parallelization
  ONE OF
  OPTION 1{
    parallelizationName  enum  ALLOWED:NOPARALLELIZATION
  }
  OPTION 2{
    parallelizationName  enum     ALLOWED:MULTITHREADING
    numberThreads        integer  >=1
  }
  OPTION 3{
    parallelizationName   enum   ALLOWED:CUDA
    gpu {
      gpuName                     string
      numberOfStreamingProcessors integer
    }
  }
riskFreeRate
  ONE OF
  OPTION 1{
    riskFreeRateName  enum    ALLOWED:CUSTOMRISKFREERATE
    riskFreeRate      number  between -1 and  1
  }
  OPTION 2{
    riskFreeRateName  enum  ALLOWED:TREASURYBILLRISKFREERATE
  }
volatility
  ONE OF
  OPTION 1{
    volatilityName  enum    ALLOWED:CUSTOMVOLATILITY
    volatility      number  between 0 and  1
  }
  OPTION 2{
    volatilityName  enum  ALLOWED:IMPLIEDVOLATILITY
  }
  OPTION 3{
    volatilityName  enum    ALLOWED:HISTORICALFIXEDVOLATILITY
    daysBack        number  between 3 and  1825
  }
  OPTION 4{
```

```
      volatilityName  enum  ALLOWED:HISTORICALEXPIRATIONVOLATILITY
    }
}
```

RESPONSE

STATUS CODE - 200: Returns the persisted ValuationInput

RESPONSE MODEL - application/json

```
{
  id                      number                 >=0
  optionId                number                 >=0
  model
    ONE OF
    OPTION 1{
      modelName  enum  ALLOWED:BLACKSCHOLES
    }
    OPTION 2{
      modelName           enum     ALLOWED:MONTECARLO
      numberOfPricePaths  integer  >=1
    }
    OPTION 3{
      modelName          enum     ALLOWED:BINOMIALTREE
      numberOfTimesteps  integer  >=1
    }
  parallelization
    ONE OF
    OPTION 1{
      parallelizationName  enum  ALLOWED:NOPARALLELIZATION
    }
    OPTION 2{
      parallelizationName  enum     ALLOWED:MULTITHREADING
      numberThreads        integer  >=1
    }
    OPTION 3{
      parallelizationName   enum   ALLOWED:CUDA
      gpu {
        gpuName                      string
        numberOfStreamingProcessors  integer
      }
    }
  riskFreeRate
    ONE OF
    OPTION 1{
      riskFreeRateName  enum    ALLOWED:CUSTOMRISKFREERATE
      riskFreeRate      number  between -1 and  1
    }
    OPTION 2{
      riskFreeRateName  enum  ALLOWED:TREASURYBILLRISKFREERATE
    }
  volatility
    ONE OF
    OPTION 1{
      volatilityName  enum    ALLOWED:CUSTOMVOLATILITY
      volatility      number  between 0 and  1
    }
    OPTION 2{
```

```
      volatilityName  enum  ALLOWED:IMPLIEDVOLATILITY
    }
    OPTION 3 {
      volatilityName  enum    ALLOWED:HISTORICALFIXEDVOLATILITY
      daysBack        number  between 3 and  1825
    }
    OPTION 4 {
      volatilityName  enum  ALLOWED:HISTORICALEXPIRATIONVOLATILITY
    }
}
```

2.4 DELETE /valuation-input

REQUEST

REQUEST BODY - application/json

```
{
  id  number  >=0
}
```

RESPONSE

STATUS CODE - 200: Successfully deleted

STATUS CODE - 400: ValuationInput with this id does not exist

STATUS CODE - 401: Can only delete ValuationInput belonging to the authenticated user

2.5 GET /valuation-output

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200: Returns all saved ValuationOutputs for the authenticated user.

RESPONSE MODEL - application/json

```
{
  savedValuationOutputs [{
  Array of object:
    id                                          integer       >=0
    optionId                                    integer       >=0
    optionValue                                 integer       >=0
    riskFreeRate
      ONE OF
      OPTION 1 {
        riskFreeRateName  enum    ALLOWED:CUSTOMRISKFREERATE
        riskFreeRate      number
      }
      OPTION 2 {
        riskFreeRateName  enum    ALLOWED:TREASURYBILLRISKFREERATE
        riskFreeRate      number
      }
    volatility
```

```
      ONE OF
      OPTION 1 {
        volatilityName  enum    ALLOWED:CUSTOMVOLATILITY
        volatility      number
      }
      OPTION 2 {
        volatilityName  enum    ALLOWED:IMPLIEDVOLATILITY
        volatility      number
      }
      OPTION 3 {
        volatilityName  enum    ALLOWED:HISTORICALFIXEDVOLATILITY
        numberOfDays    number
        volatility      number
      }
      OPTION 4 {
        volatilityName  enum    ALLOWED:HISTORICALEXPIRATIONVOLATILITY
        volatility      number
      }
    model
      ONE OF
      OPTION 1 {
        modelName  enum  ALLOWED:BLACKSCHOLES
      }
      OPTION 2 {
        modelName           enum     ALLOWED:MONTECARLO
        numberOfPricePaths  integer  >=1
      }
      OPTION 3 {
        modelName          enum     ALLOWED:BINOMIALTREE
        numberOfTimesteps  integer  >=1
      }
    parallelization
      ONE OF
      OPTION 1 {
        parallelizationName  enum  ALLOWED:NOPARALLELIZATION
      }
      OPTION 2 {
        parallelizationName  enum     ALLOWED:MULTITHREADING
        numberThreads        integer  >=1
      }
      OPTION 3 {
        parallelizationName   enum   ALLOWED:CUDA
        gpu {
          gpuName                    string
          numberOfStreamingProcessors  integer
        }
      }
    parallelExecutionTimeMilliseconds       number        >=0
    totalTimeMilliseconds                   number        >=0
  }]
}
```

2.6 POST /valuation-output

REQUEST

REQUEST BODY - application/json

```
{
  option {
    exerciseStyle   enum     ALLOWED:AMERICAN, EUROPEAN
    stockSymbol     string
    strikePrice     integer  between 0 and  10000
    expirationDate  string   Minimum: today, maximum: today + 1825 days (about 5 years)
    type            enum     ALLOWED:CALL, PUT
  }
  optionValue                              integer        >=0
  riskFreeRate
    ONE OF
    OPTION 1{
      riskFreeRateName  enum    ALLOWED:CUSTOMRISKFREERATE
      riskFreeRate      number
    }
    OPTION 2{
      riskFreeRateName  enum    ALLOWED:TREASURYBILLRISKFREERATE
      riskFreeRate      number
    }
  volatility
    ONE OF
    OPTION 1{
      volatilityName  enum    ALLOWED:CUSTOMVOLATILITY
      volatility      number
    }
    OPTION 2{
      volatilityName  enum    ALLOWED:IMPLIEDVOLATILITY
      volatility      number
    }
    OPTION 3{
      volatilityName  enum    ALLOWED:HISTORICALFIXEDVOLATILITY
      numberOfDays    number
      volatility      number
    }
    OPTION 4{
      volatilityName  enum    ALLOWED:HISTORICALEXPIRATIONVOLATILITY
      volatility      number
    }
  model
    ONE OF
    OPTION 1{
      modelName  enum  ALLOWED:BLACKSCHOLES
    }
    OPTION 2{
      modelName           enum     ALLOWED:MONTECARLO
      numberOfPricePaths  integer  >=1
    }
    OPTION 3{
      modelName          enum     ALLOWED:BINOMIALTREE
      numberOfTimesteps  integer  >=1
    }
  parallelization
    ONE OF
    OPTION 1{
      parallelizationName  enum  ALLOWED:NOPARALLELIZATION
    }
```

```
    OPTION 2 {
      parallelizationName  enum     ALLOWED:MULTITHREADING
      numberThreads        integer  >=1
    }
    OPTION 3 {
      parallelizationName   enum    ALLOWED:CUDA
      gpu {
        gpuName                     string
        numberOfStreamingProcessors integer
      }
    }
  parallelExecutionTimeMilliseconds       number        >=0
  totalTimeMilliseconds                   number        >=0
}
```

RESPONSE

STATUS CODE - 200: Returns the persisted ValuationOutput

RESPONSE MODEL - application/json

```
{
  id                                      integer       >=0
  optionId                                number        >=0
  optionValue                             integer       >=0
  riskFreeRate
    ONE OF
    OPTION 1 {
      riskFreeRateName  enum    ALLOWED:CUSTOMRISKFREERATE
      riskFreeRate      number
    }
    OPTION 2 {
      riskFreeRateName  enum    ALLOWED:TREASURYBILLRISKFREERATE
      riskFreeRate      number
    }
  volatility
    ONE OF
    OPTION 1 {
      volatilityName  enum    ALLOWED:CUSTOMVOLATILITY
      volatility      number
    }
    OPTION 2 {
      volatilityName  enum    ALLOWED:IMPLIEDVOLATILITY
      volatility      number
    }
    OPTION 3 {
      volatilityName  enum    ALLOWED:HISTORICALFIXEDVOLATILITY
      numberOfDays    number
      volatility      number
    }
    OPTION 4 {
      volatilityName  enum    ALLOWED:HISTORICALEXPIRATIONVOLATILITY
      volatility      number
    }
  model
    ONE OF
    OPTION 1 {
      modelName  enum  ALLOWED:BLACKSCHOLES
```

```
    }
    OPTION 2 {
      modelName           enum     ALLOWED:MONTECARLO
      numberOfPricePaths  integer  >=1
    }
    OPTION 3 {
      modelName          enum     ALLOWED:BINOMIALTREE
      numberOfTimesteps  integer  >=1
    }
  parallelization
    ONE OF
    OPTION 1 {
      parallelizationName  enum  ALLOWED:NOPARALLELIZATION
    }
    OPTION 2 {
      parallelizationName  enum     ALLOWED:MULTITHREADING
      numberThreads        integer  >=1
    }
    OPTION 3 {
      parallelizationName   enum   ALLOWED:CUDA
      gpu {
        gpuName                     string
        numberOfStreamingProcessors integer
      }
    }
  parallelExecutionTimeMilliseconds       number         >=0
  totalTimeMilliseconds                   number         >=0
}
```

2.7 DELETE /valuation-output

REQUEST

REQUEST BODY - application/json

```
{
  id  number  >=0
}
```

RESPONSE

STATUS CODE - 200: Successfully deleted

STATUS CODE - 400: ValuationOutput with this id does not exist

STATUS CODE - 401: Can only delete ValuationOutput belonging to the authenticated user

3. PRICING

3.1 GET /job/{id}

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*secret	string	

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

ONE OF

OPTION 1 {
  options [{
  Array of object:
    expirationDate string
    type enum ALLOWED:CALL, PUT
    strikePrice number >=0
    optionValueFromYahoo number >=0
    optionValue number >=0
    valueDifference number >=0
    yahooLink string
  }]
  riskFreeRate number As decimal
  volatility [number]
  parallelExecutionTimeMilliseconds number >=0
}

OPTION 2 {
  optionValue integer >=0
  stockPrice number
  riskFreeRate number As decimal
  volatility number >=0
  parallelExecutionTimeMilliseconds number >=0
}

STATUS CODE - 202: Job is not available

3.2 POST /option-value

REQUEST

REQUEST BODY - application/json

{
  option* {
    exerciseStyle enum ALLOWED:AMERICAN, EUROPEAN
    stockSymbol string
    strikePrice integer between 0 and 10000
    expirationDate string Minimum: today, maximum: today + 1825 days (about 5 years)

```
    type            enum     ALLOWED:CALL, PUT
  }
  model*
    ONE OF
    OPTION 1{
      modelName  enum  ALLOWED:BLACKSCHOLES
    }
    OPTION 2{
      modelName           enum     ALLOWED:MONTECARLO
      numberOfPricePaths  integer  >=1
    }
    OPTION 3{
      modelName          enum     ALLOWED:BINOMIALTREE
      numberOfTimesteps  integer  >=1
    }
  parallelization*
    ONE OF
    OPTION 1{
      parallelizationName  enum  ALLOWED:NOPARALLELIZATION
    }
    OPTION 2{
      parallelizationName  enum     ALLOWED:MULTITHREADING
      numberThreads        integer  >=1
    }
    OPTION 3{
      parallelizationName   enum   ALLOWED:CUDA
      gpu {
        gpuName                      string
        numberOfStreamingProcessors  integer
      }
    }
  volatility
    ONE OF
    OPTION 1{
      volatilityName  enum    ALLOWED:CUSTOMVOLATILITY
      volatility      number  between 0 and  1
    }
    OPTION 2{
      volatilityName  enum  ALLOWED:IMPLIEDVOLATILITY
    }
    OPTION 3{
      volatilityName  enum    ALLOWED:HISTORICALFIXEDVOLATILITY
      daysBack        number  between 3 and  1825
    }
    OPTION 4{
      volatilityName  enum  ALLOWED:HISTORICALEXPIRATIONVOLATILITY
    }
  riskFreeRate
    ONE OF
    OPTION 1{
      riskFreeRateName  enum    ALLOWED:CUSTOMRISKFREERATE
      riskFreeRate      number  between -1 and  1
    }
    OPTION 2{
      riskFreeRateName  enum  ALLOWED:TREASURYBILLRISKFREERATE
    }
}
```

RESPONSE

STATUS CODE - 200: Description

RESPONSE MODEL - application/json

```
{
  jobId   string
  secret  string
}
```

STATUS CODE - 400: Invalid request

3.3 POST /profit-spotter

REQUEST

REQUEST BODY - application/json

```
{
  stockSymbol*          string
  precision*            enum            ALLOWED:SINGLE, DOUBLE
  parallelization*
    ONE OF
    OPTION 1{
      parallelizationName  enum      ALLOWED:MULTITHREADING
      numberThreads        integer   >=1
    }
    OPTION 2{
      parallelizationName   enum    ALLOWED:CUDA
      gpu {
        gpuName                      string
        numberOfStreamingProcessors  integer
      }
    }
  volatility*
    ONE OF
    OPTION 1{
      volatilityName  enum    ALLOWED:CUSTOMVOLATILITY
      volatility      number  between 0 and  1
    }
    OPTION 2{
      volatilityName  enum  ALLOWED:IMPLIEDVOLATILITY
    }
    OPTION 3{
      volatilityName  enum    ALLOWED:HISTORICALFIXEDVOLATILITY
      daysBack        number  between 3 and  1825
    }
    OPTION 4{
      volatilityName  enum  ALLOWED:HISTORICALEXPIRATIONVOLATILITY
    }
  riskFreeRate
    ONE OF
    OPTION 1{
      riskFreeRateName  enum    ALLOWED:CUSTOMRISKFREERATE
      riskFreeRate      number  between -1 and  1
    }
    OPTION 2{
```

```
    riskFreeRateName  enum  ALLOWED:TREASURYBILLRISKFREERATE
  }
}
```

RESPONSE

STATUS CODE - 200: Description

RESPONSE MODEL - application/json

```
{
  jobId   string
  secret  string
}
```

STATUS CODE - 400: The symbol is not valid

3.4 GET /hardware-capabilities

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200: Returns the capabilities of the connected hardware

RESPONSE MODEL - application/json

```
{
  numberThreads          integer  >=0
                                  null indicates that the hardware is not accessible
  gpu {
  null indicates that the hardware is not accessible
    gpuName                       string
    numberOfStreamingProcessors   integer
  }
  maxNumberOfPricePaths  number
  maxNumberOfTimesteps   number
}
```

3.5 GET /symbol-from-yahoo?{searchString}

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*searchString	string 1 to 10 chars	

RESPONSE

STATUS CODE - 200: Returns a list of stock symbols for a given search string

RESPONSE MODEL - application/json

```
{
```

```
  stockSymbolsFound [{
  Array of object:
    symbol string
    name   string
  }]
}
```

3.6 GET /expiration-dates-from-yahoo?{stockSymbol}

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*stockSymbol	string 1 to 8 chars	

RESPONSE

STATUS CODE - 200: Returns an array of all possible expiration dates for the options of the stock

RESPONSE MODEL - application/json

```
{
  expirationDates [string]
}
```

3.7 GET /options-from-yahoo?{stockSymbol}&{expirationDate}

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*stockSymbol	string 1 to 8 chars	
*expirationDate	date	

RESPONSE

STATUS CODE - 200: Returns all options with the given expiration date for the stock

RESPONSE MODEL - application/json

```
{
  yahooOptions [{
  Array of object:
    stockSymbol    string
    strikePrice    number
    expirationDate string
    type           enum   ALLOWED:CALL, PUT
  }]
}
```

3.8 GET /option-value-from-yahoo?{stockSymbol}&{expirationDate}&{type}&{strikePrice}

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*stockSymbol	string 1 to 8 chars	
*expirationDate	date	
*type	enum ALLOWED: CALL, PUT	
*strikePrice	number	

RESPONSE

STATUS CODE - 200: Returns the current market price for an option from yahoo finance

RESPONSE MODEL - application/json

```
{
  optionValueFromYahoo  number
}
```

StOP Asynchronous API 1.0.0 documentation

This API is used for the communication between the NestJS REST API and the CUDA and Multithreading Option Pricer.

Table of Contents

Servers

`default` Server

- URL: `rabbitmq:5672`
- Protocol: `amqp`

RabbitMQ Message Broker

Operations

PUB `cuda-key` Operation

`amqp` Operation specific information

Name	Type	Description	Value	Constraints	Notes
is	-	-	`"routingKey"`	-	-
queue	-	-	-	-	-
queue.name	-	-	`"cuda"`	-	-
exchange	-	-	-	-	-
exchange.name	-	-	`"global-exchange"`	-	-
exchange.type	-	-	`"direct"`	-	-

Accepts **one of** the following messages:

Message `hardware-capabilities`

- Content type: application/json (https://www.iana.org/assignments/media-types/application/json)

Payload

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

Examples of payload *(generated)*

```
{}
```

`amqp` Message specific information

Name	Type	Description	Value	Constraints	Notes
messageType	-	-	`"hardware-capabilities"`	-	-

Message `cuda-option-value`

- Content type: application/json (https://www.iana.org/assignments/media-types/application/json)

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
option	object	-	-	-	**additional properties are allowed**
option.exerciseStyle	string	-	allowed (`"AMERICAN"`, `"EUROPEAN"`)	-	-
option.strikePrice	integer	-	-	>= 0	-
option.yearsUntilExpiration	number	-	-	-	-
option.type	string	-	allowed (`"CALL"`, `"PUT"`)	-	-
volatility	number	-	-	>= 0	-
riskFreeRate	number	-	-	-	-
stockPrice	number	-	-	>= 0	-
model	oneOf	-	-	-	**additional properties are allowed**
model.0 (oneOf item)	object	-	-	-	**additional properties are allowed**
			allowed		

model.0.modelName	string	-	("MONTECARLO")	-	-
model.0.numberOfPricePaths	integer	-	-	>= 1	-
model.1 (oneOf item)	object	-	-	-	**additional properties are allowed**
model.1.modelName	string	-	allowed ("BINOMIALTREE")	-	-
model.1.numberOfTimesteps	integer	-	-	>= 1	-

Examples of payload *(generated)*

```
{
  "option": {
    "exerciseStyle": "AMERICAN",
    "strikePrice": 0,
    "yearsUntilExpiration": 0,
    "type": "CALL"
  },
  "volatility": 0,
  "riskFreeRate": 0,
  "stockPrice": 0,
  "model": {
    "modelName": "MONTECARLO",
    "numberOfPricePaths": 1
  }
}
```

`amqp` Message specific information

Name	Type	Description	Value	Constraints	Notes
messageType	-	-	`"option-value"`	-	-

Message `cuda-profit-spotter`

- Content type: application/json (https://www.iana.org/assignments/media-types/application/json)

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
options	array<object>	-	-	-	-
options.strikePrice	integer	-	-	>= 0	-
options.yearsUntilExpiration	number	-	- allowed	-	-

options.type	string	-	("CALL", "PUT")	-	-
volatility	array<number>	-	-	-	-
volatility (single item)	number	-	-	-	-
riskFreeRate	number	-	-	-	-
stockPrice	number	-	-	>= 0	-
precision	string	-	allowed ("SINGLE", "DOUBLE")	-	-

Examples of payload *(generated)*

```
{
  "options": [
    {
      "strikePrice": 0,
      "yearsUntilExpiration": 0,
      "type": "CALL"
    }
  ],
  "volatility": [
    0
  ],
  "riskFreeRate": 0,
  "stockPrice": 0,
  "precision": "SINGLE"
}
```

`amqp` Message specific information

Name	Type	Description	Value	Constraints	Notes
messageType	-	-	"profit-spotter"	-	-

PUB `multithreading-key` Operation

`amqp` Operation specific information

Name	Type	Description	Value	Constraints	Notes
is	-	-	"routingKey"	-	-
queue	-	-	-	-	-
queue.name	-	-	"multithreading"	-	-
exchange	-	-	-	-	-
exchange.name	-	-	"global-exchange"	-	-
exchange.type	-	-	"direct"	-	-

Accepts **one of** the following messages:

Message `hardware-capabilities`

- Content type: application/json (https://www.iana.org/assignments/media-types/application/json)

Payload

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

Examples of payload *(generated)*

```
{}
```

`amqp` Message specific information

Name	Type	Description	Value	Constraints	Notes
messageType	-	-	`"hardware-capabilities"`	-	-

Message `multithreading-option-value`

- Content type: application/json (https://www.iana.org/assignments/media-types/application/json)

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
option	object	-	-	-	**additional properties are allowed**
option.exerciseStyle	string	-	allowed (`"AMERICAN"`, `"EUROPEAN"`)	-	-
option.strikePrice	integer	-	-	>= 0	-
option.yearsUntilExpiration	number	-	-	-	-
option.type	string	-	allowed (`"CALL"`, `"PUT"`)	-	-
volatility	number	-	-	>= 0	-
riskFreeRate	number	-	-	-	-
stockPrice	number	-	-	>= 0	-
model	oneOf	-	-	-	**additional properties are allowed**
model.0 (oneOf item)	object	-	-	-	**additional properties**

					are allowed
model.0.modelName	string	-	allowed ("MONTECARLO")	-	-
model.0.numberOfPricePaths	integer	-	-	>= 1	-
model.1 (oneOf item)	object	-	-	-	**additional properties are allowed**
model.1.modelName	string	-	allowed ("BINOMIALTREE")	-	-
model.1.numberOfTimesteps	integer	-	-	>= 1	-
numberThreads	integer	-	-	>= 1	-

Examples of payload *(generated)*


```
{
  "option": {
    "exerciseStyle": "AMERICAN",
    "strikePrice": 0,
    "yearsUntilExpiration": 0,
    "type": "CALL"
  },
  "volatility": 0,
  "riskFreeRate": 0,
  "stockPrice": 0,
  "model": {
    "modelName": "MONTECARLO",
    "numberOfPricePaths": 1
  },
  "numberThreads": 1
}
```


`amqp` Message specific information

Name	Type	Description	Value	Constraints	Notes
messageType	-	-	`"option-value"`	-	-

Message `multithreading-profit-spotter`

- Content type: [application/json (https://www.iana.org/assignments/media-types/application/json)]

Payload

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

					allowed
option	array<object>	-	-	-	-
option.strikePrice	integer	-	-	>= 0	-
option.yearsUntilExpiration	number	-	-	-	-
option.type	string	-	allowed ("CALL", "PUT")	-	-
volatility	array<number>	-	-	-	-
volatility (single item)	number	-	-	-	-
riskFreeRate	number	-	-	-	-
stockPrice	number	-	-	>= 0	-
precision	string	-	allowed ("SINGLE", "DOUBLE")	-	-
numberThreads	integer	-	-	>= 1	-

Examples of payload *(generated)*

```
{
  "option": [
    {
      "strikePrice": 0,
      "yearsUntilExpiration": 0,
      "type": "CALL"
    }
  ],
  "volatility": [
    0
  ],
  "riskFreeRate": 0,
  "stockPrice": 0,
  "precision": "SINGLE",
  "numberThreads": 1
}
```

`amqp` Message specific information

Name	Type	Description	Value	Constraints	Notes
messageType	-	-	`"profit-spotter"`	-	-

SUB response-key Operation

`amqp` Operation specific information

Name	Type	Description	Value	Constraints	Notes
is	-	-	`"routingKey"`	-	-
queue	-	-	-	-	-
queue.name	-	-	`"response"`	-	-

exchange	-	-	-	-	-
exchange.name	-	-	`"global-exchange"`	-	-
exchange.type	-	-	`"direct"`	-	-

Accepts **one of** the following messages:

Message `hardware-capabilities`

- Content type: application/json (https://www.iana.org/assignments/media-types/application/json)

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	oneOf	-	-	-	**additional properties are allowed**
0 (oneOf item)	object	-	-	-	**additional properties are allowed**
numberThreads	integer	-	-	-	-
1 (oneOf item)	object	-	-	-	**additional properties are allowed**
1.gpuName	string	-	-	-	-
1.numberOfStreamingProcessors	integer	-	-	-	-

Examples of payload *(generated)*

```
{
  "numberThreads": 0
}
```

`amqp` Message specific information

Name	Type	Description	Value	Constraints	Notes
messageType	-	-	`"hardware-capabilities"`	-	-

Message `option-value`

- Content type: application/json (https://www.iana.org/assignments/media-types/application/json)

Payload

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

optionValue	number	-	-	-	-
parallelExecutionTimeMilliseconds	number	-	-	-	-

Examples of payload *(generated)*


```
{
  "optionValue": 0,
  "parallelExecutionTimeMilliseconds": 0
}
```


`amqp` Message specific information

Name	Type	Description	Value	Constraints	Notes
messageType	-	-	`"option-value"`	-	-

Message `profit-spotter`

- Content type: application/json (https://www.iana.org/assignments/media-types/application/json)

Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	additional properties are allowed
optionValues	array<number>	-	-	-	-
optionValues (single item)	number	-	-	-	-
parallelExecutionTimeMilliseconds	number	-	-	-	-

Examples of payload *(generated)*


```
{
  "optionValues": [
    0
  ],
  "parallelExecutionTimeMilliseconds": 0
}
```


`amqp` Message specific information

Name	Type	Description	Value	Constraints	Notes
messageType	-	-	`"profit-spotter"`	-	-