FOM Hochschule für Oekonomie & Management

Hochschulzentrum Bremen

Bachelor-Thesis

im Studiengang Wirtschaftsinformatik

zur Erlangung des Grades eines

Bachelor of Science (B.Sc.)

über das Thema

GoBD-konforme Datenbanken auf Grundlage der Blockchain Technologien am Beispiel der Steuerberatungspraxis

von

Tobias Unruh

Erstgutachter	Dr.-Ing. Jörg Höhne
Matrikelnummer	560681
Abgabedatum	05.08.2024

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Abkürzungsverzeichnis

GoBD	Grundsätze zur ordnungsmäßigen Führung und Aufbewahrung von Büchern, Aufzeichnungen und Unterlagen in elektronischer Form sowie zum Datenzugriff
HGB	Handelsgesetzbuch
AO	Abgabenordnung
OHG	offene Handelsgesellschaft
KG	Kommanditgesellschaft
AG	Aktiengesellschaft
AktG	Aktiengesetz
GmbH	Gesellschaft mit beschränkter Haftung
GmbHG	Gesetz betreffend die Gesellschaften mit beschränkter Haftung
UStG	Umsatzsteuergesetz
BMF	Bundesministerium für Finanzen
EStG	Einkommensteuergesetz
IKS	internes Kontrollsystem
Nonce	Number only used once
CRUD	Create/Read/Update/Delete

1 Einleitung

Nachfolgend werden die Grundlagen der Arbeit dargestellt. Im Abschnitt 1.1 wird auf die Konventionen der Arbeit eingegangen. Danach wird in Abschnitt 1.2 die Motivation der Arbeit vorgestellt. Im Abschnitt 1.3 wird das zu lösende Problem dargestellt und nachfolgend im Abschnitt 1.4 erklärt, welches Ziel erreicht werden soll und in welchem Anwendungskontext die erarbeitete Lösung verwendet werden soll.
Abschließend wird in Abschnitt 1.5 die Forschungsfrage ausformuliert und in Abschnitt 1.6 der Aufbau der Arbeit vorgestellt.

1.1 Allgemeines und Konventionen dieser Arbeit

Der im Rahmen dieser Arbeit erstellte Code ist unter http://github.com/Dixieslicer/BCDB, zusammen mit einer PDF-Version der Arbeit, abrufbar. Alle Programmierungen sind in C# vorgenommen und kompatibel mit .NET in der Version 8. Die Erläuterung des Quellcodes ist im Anhang A.1 zu finden. Die Programmierung wurde in Visual Studio 2022 Community-Edition in Version 17.10.0 durchgeführt.
Zur besseren Lesbarkeit wird in dieser Arbeit das generische Maskulinum verwendet. Die in dieser Arbeit verwendeten Personenbezeichnungen beziehen sich – sofern nicht anders kenntlich gemacht – auf alle Geschlechter.

1.2 Motivation

Die Grundsätze zur ordnungsmäßigen Führung und Aufbewahrung von Büchern, Aufzeichnungen und Unterlagen in elektronischer Form sowie zum Datenzugriff (GoBD) sind für alle Systeme, welche Daten in die steuerlichen Aufzeichnungen einfließen lassen, verpflichtend. Teutemacher, 2020 geht sogar so weit, dass in einem Café selbst der interne Zähler der Kaffeemaschine, wenn diese mit der Kasse vernetzt ist, den Ansprüchen der GoBD standhalten muss.
Trotzdem kommt es in der Praxis immer wieder zu Situationen, in denen Software die GoBD nicht oder nur ungenügend umsetzt. Im Falle einer Prüfung kann es hierbei zu Schätzungsbefugnissen seitens des Finanzamts kommen und ein enormer Schaden für

den Steuerpflichtigen entstehen. Im Rahmen der Betriebsprüfungen wurde im Jahr 2022 insgesamt ein Mehrergebnis in Höhe von 10,8 Mrd. Euro durch die Prüfer erzielt (Bundesministerium für Finanzen, 2023). Durch Schaffung einer Technologie, welche die GoBD bereits auf Ebene der Datenhaltung berücksichtigt, soll gewährleistet werden, dass weniger Software entsteht, die nicht GoBD-konform ist. Hierdurch entsteht sowohl für Steuerberatungskanzleien als auch für die Betriebe eine Sicherheit, dass das Finanzamt keine Schätzungen auf Grundlage der GoBD durchführen kann. Eine Verletzung der GoBD sollte dann nur noch entstehen können, wenn der Entwickler oder Anwender dies vorsätzlich herbeiführt.

1.3 Problembeschreibung

Die Grundlagen der GoBD sind für alle Unternehmen in Deutschland bindend. Zum einen werden diese im Handelsgesetzbuch (HGB) für die Kaufleute festgelegt und zum anderen befinden sich Regelungen zu der Beschaffenheit von Buchführungen und Aufzeichnungen in den Regelungen der Abgabenordnung (AO).
Die derzeitige Umsetzung geschieht über klassische Datenbanken, welche durch Passwörter geschützt werden und nur über Umwege den GoBD gerecht werden. Einer der wichtigsten Punkte hierbei ist, dass Buchungen nicht im Nachhinein verändert werden dürfen und jede Änderung dokumentiert werden muss. Bei klassischen Datenbanken kann bei vorhandenem Passwort aber ein Datensatz spurlos aus der Datenbank entfernt werden, wenn man auch die dazugehörigen Logs bereinigt.
Der Aufwand, die Konformität mit den Gesetzen herzustellen, liegt derzeit bei den Softwareerstellern selbst und kann bei ungenauer Kenntnis der GoBD bis zur kompletten Ungültigkeit der Buchführung führen.

1.4 Anwendungskontext und Zielsetzung

Zielsetzung der Arbeit ist es, einen Prototyp für eine Datenbank zu entwickeln, welcher auf technischer Ebene die Konformität mit den gesetzlichen Vorschriften herstellt, und nicht durch eine Vereinbarung, welche durch die Entwickler umgesetzt werden muss. Hierdurch soll es Entwicklern erleichtert werden, Anwendungen im Unternehmenskontext zu entwickeln, ohne die GoBD zu erarbeiten.
Die entwickelte Lösung soll im Kontext der Anwendung in der Steuerberatungspraxis entwickelt werden und die hieraus zusätzlich entstehenden Anforderungen erfüllen.
Die Betrachtung wird allerdings auf den Bereich der Finanzbuchhaltung begrenzt, da ansonsten auch alle Nebensysteme wie die Stammdatenverwaltung, Rechnungsschreibung

und Ähnliches betrachtet werden müssten und sich zu viele kleinteilige Besonderheiten ergeben würden.

1.5 Forschungsfragen und Methodik

Die in dieser Arbeit zu untersuchende These lautet „Mit den Grundlagen der Blockchain-Technologien kann eine GoBD-konforme Datenbank erstellt werden.“
Darüber hinaus stellen sich noch einige Nebenfragen, welche geklärt werden sollen:

- Muss der komplette Technologie-Stack der Blockchain für die Implementierung genutzt werden oder reichen Teilbereiche?
- Welche Auswirkung hat die Nutzung einer Blockchain auf die Performance einer Datenbank?
- Wie können die Daten am besten dargestellt werden, damit diese auch für spätere KI-Anwendungen verwendbar sind?
- Wie kann der Eingriff in die Datenhaltung nachgewiesen werden?
- Welche Design-Entscheidungen werden benötigt, um die Daten nach einem Eingriff in die Datenbank wieder nutzbar zu machen?

Die Bearbeitung des Problems geschieht angelehnt an die Vorgehensweise, welche Weber, 2022, für Abschlussarbeiten in anwendungsorientierten Studiengängen vorschlägt und daher anhand des Ansatzes der Design-Science-Research.
Hierzu wird in den Kapiteln 2 bis 3 und dem Kapitel 5 die Literaturrecherche als Methodik angewandt. Im Kapitel 3 werden des weiteren User-Storys verwendet, um die Anforderungen, die sich aus dem Anwendungsgebiet ergeben, zu vermitteln.
In Kapitel 4 wird das Problemfeld, mit dem sich diese Arbeit beschäftigt, genauer beschrieben und argumentiert, wieso derzeitige Lösungen nicht vollends geeignet sind. Das Kapitel 6 beschreibt den per Prototyping erstellten Prototyp, die Bedienung dieses Prototyps wird anschließend in Kapitel 7 erläutert.
Abgeschlossen wird die Arbeit mit der Evaluation der in Kapitel 3 ermittelten Anforderungen in Kapitel 8 und der hieran anschließenden Diskussion der Ergebnisse in 9.

1.6 Aufbau der Arbeit

Die nachfolgenden Kapitel sollen die zuvor genannten Fragen beantworten. Hierzu ist die Arbeit in drei Teile geteilt.

Der erste Teil besteht aus den Kapiteln 2 bis 4. Hier werden die Grundlagen der GoBD im Detail vorgestellt und anschließend auf die Anforderungen eingegangen, welche ein System zur GoBD-konformen Datenhaltung einhalten muss. Abschließend wird in diesem Teil das Problemfeld, welches die Arbeit lösen soll, vorgestellt. Der erste Teil führt die theoretischen Grundlagen ein und stellt die Problematik dar.
Im zweiten Teil der Arbeit mit den Kapiteln 5 bis 7 wird vorgestellt, wie das Problem gelöst werden soll. Hierzu werden zunächst im Kapitel 5 die Grundlagen der Blockchain eingeführt, anschließend wird in Kapitel 6 der im Rahmen der Arbeit erstellte Prototyp beschrieben. Abschließend wird die Funktionsweise des Prototyps im Kapitel 7 vorgeführt.
Der dritte Teil der Arbeit beschäftigt sich mit der Frage, ob die im zweiten Teil der Arbeit erstellte Lösung die Anforderungen des ersten Teils erfüllen kann. Hierzu wird zunächst im Kapitel 8 evaluiert, inwiefern die ermittelten Anforderungen erfüllt werden konnten, anschließend werden die Ergebnisse im Kapitel 9 diskutiert und abschließend im Kapitel 10 ein Fazit gezogen.

2 GoBD – Rechtliche Grundlagen

Nachfolgend werden die GoBD vorgestellt. Hierbei wird sich auf eine steuerrechtliche Sicht der Vorgaben beschränkt und die Auswirkungen einer Verletzung der GoBD auf die Besteuerung dargestellt. Hierzu werden zunächst die handelsrechtlichen Vorschriften in Abschnitt 2.1 erläutert. Anschließend wird in Abschnitt 2.2 auf die Bedeutung der handelsrechtlichen Vorschriften für das Steuerrecht und auf die steuerliche Version der GoBD eingegangen, und die steuerrechtlichen Konsequenzen eines Verstoßes werden aufgezeigt. Unabhängig vom Rechtsgebiet sind die GoBD die Vorgabe, wie Bücher und andere Aufzeichnungen beschaffen sein müssen, damit diese für den Zweck des Gesetzes anerkannt werden können. Dies hat im Steuerrecht den Zweck, als Grundlage der Besteuerung zu dienen und das Unternehmensergebnis für die Berechnung der Steuern zu bestimmen (Der Bundesbeauftragte für den Datenschutz und die Informationsfreiheit, o. d.).

2.1 Handelsrechtliche Grundlagen

Die Pflicht, eine Buchführung im Rahmen des Handelsgesetzbuches zu erstellen, regelt der § 238 HGB. Dieser Paragraf bestimmt, dass alle Kaufleute im Sinne des Gesetzes verpflichtet sind, Bücher zu führen. Generell gibt es hierbei verschiedene Arten von Kaufleuten:

Ist-Kaufleute
: Ist-Kaufleute sind Kaufleute im Sinne des § 1 HGB, welche ein Handelsgewerbe betreiben.

Kann-Kaufleute
: § 2 HGB regelt, dass jeder Gewerbetreibende zum Kaufmann wird, wenn er sich freiwillig im Handelsregister eintragen lässt, diese Option wird durch den § 3 HGB auch auf Betriebe der Land- und Forstwirtschaft erweitert.

Form-Kaufleute
: nach § 6 HGB sind auch Handelsgesellschaften Kaufleute. Dies betrifft die offene Handelsgesellschaft (OHG) und die Kommanditgesellschaft (KG), aber auch die

Aktiengesellschaft (AG), welche nach § 3 Abs. 1 Aktiengesetz (AktG) als Handelsgesellschaft gilt, und die Gesellschaft mit beschränkter Haftung (GmbH) nach § 13 Abs. 3 Gesetz betreffend die Gesellschaften mit beschränkter Haftung (GmbHG).

Alle Betriebe, welche diese Voraussetzungen erfüllen, sind somit verpflichtet, handelsrechtliche Bücher zu führen, welche die Anforderungen, die das Gesetz an die Buchführung stellt, erfüllen müssen. Diese werden in den beiden nachfolgenden Abschnitten erläutert.

2.1.1 Allgemeine Grundlagen der Buchführung

Die Regelungen zu der Beschaffenheit der Buchführung finden sich hauptsächlich in den §§ 238 f. HGB. Diese bilden einen Teil der durch das Bundesministerium für Finanzen (BMF) festgesetzten Regelungen. Sie beinhalten die folgenden Punkte:

Prüfbarkeit

Die Prüfbarkeit ist im § 238 Abs. 1 S. 2 u. 3 HGB geregelt. Sowohl Teutemacher, 2020 (S. 26 ff.), als auch Kling, 2020 (S. 25 ff.) beschreiben dieses Kriterium mit der Belegfunktion. In Buchführungskreisen hat sich der Merksatz *Keine Buchung ohne Beleg* für diesen Grundsatz etabliert. Insgesamt soll sowohl eine progressive Prüfung vom Beleg über die Aufzeichnungen in den Grund- und Hauptbüchern bis in die Bilanz und später in die Steuererklärung möglich gemacht werden, sodass man für jeden Beleg den Verlauf im Unternehmen prüfen kann. Die retrograde Prüfbarkeit beschreibt genau den umgekehrten Weg. Hierbei soll es möglich sein, ausgehend von der Steuererklärung nachzuweisen, welche Belege in eine Summe innerhalb der Steuererklärung eingeflossen sind.

Vollständigkeit

Kling, 2020 (S. 41) beschreibt das Kriterium der Vollständigkeit als nicht dem Schutze der Daten dienend (siehe hierzu 2.1.2). Er betont die im Bundesministerium für Finanzen, 2019 (Rz. 36) enthaltene Regelung, dass Geschäftsvorfälle fortlaufend und lückenlos aufzuzeichnen seien. Eine mehrfache Erfassung ist nicht erlaubt, die Vollständigkeit der Informationen digitalisierter Belege ist einzuhalten. Des Weiteren wird der § 14 Abs. 4 Umsatzsteuergesetz (UStG) genannt, der vorschreibt, dass Ausgangsrechnungen eine fortlaufende Nummerierung erhalten müssen.

Richtigkeit

Die Richtigkeit im Sinne des Gesetzes besagt laut Teutemacher, 2020 (S. 31 ff.), dass die Geschäftsvorfälle so aufzuzeichnen sind, dass sie den wahren Begebenheiten entsprechen. Fehler können hierbei auf verschiedenen Ebenen entstehen. Beispiele für den Verstoß

wären sogenannte Schein- und Gefälligkeitsrechnungen, bei denen Rechnungen für nicht erbrachte Leistungen geschrieben werden, um die Steuerlast zu senken.

Ordnung

Das Prinzip der Ordnung beschreibt, dass die Buchungen in einem geordneten Rahmen in ein Journal und auf verschiedene Konten zu buchen sind (Kling, 2020 (S. 44 f.)). Ein Journal ist die Auflistung aller Geschäftsvorfälle mit Angaben der angesprochenen Konten. Ergänzend hierzu führt Teutemacher, 2020 (S. 35 ff.) aus, dass hierdurch die prüfenden Stellen in einer angemessenen Zeit die Vollständigkeit und Unveränderbarkeit der Buchungen prüfen können.

2.1.2 Technische Aspekte im Handelsrecht

Im § 239 Abs. 4 HGB wird geregelt, dass die Buchführung auch auf Datenträgern, und somit auch digital, erstellt werden kann, solange die Regelungen zu der Beschaffenheit der Buchführung eingehalten werden.
Darüber hinaus führt der § 239 Abs. 3 HGB die aus technischer Sicht interessanteste Vorschrift ein, hier heißt es:

> „Eine Eintragung oder eine Aufzeichnung darf nicht in einer Weise verändert werden, daß der ursprüngliche Inhalt nicht mehr feststellbar ist. Auch solche Veränderungen dürfen nicht vorgenommen werden, deren Beschaffenheit es ungewiß läßt, ob sie ursprünglich oder erst später gemacht worden sind“ [sic!].

Aus der Regelung ergibt sich, dass jede einmal erfasste Buchung aufgezeichnet werden muss und nachvollziehbar sein muss, wie die Ursprungsbuchung einmal ausgesehen hat. Änderungen und Löschungen müssen so aufgezeichnet werden, dass die Ursprungsbuchung unverletzt bleibt.
Hierzu stellt Teutemacher, 2020 (S. 38 ff.) dar, dass die Regelung schon Bestand hatte, bevor die digitale Buchführung entstand und die Bücher noch auf Papier erstellt wurden. Hier war es unter anderem verboten, Rasuren, Überschreibungen, Radierungen, Überklebungen, Übermalungen und Ähnliches durchzuführen.
Als gängige Wege gibt Kling, 2020 (S. 32 ff.) an, dass man in hardwaremäßige, softwaremäßige und organisatorische Mittel zur Umsetzung der Vorgabe unterscheidet. Die hardwaremäßige Lösung ist beispielsweise eine nach § 6 KassenSichV vorgegebene TSE, welche die Vollständigkeit der Aufzeichnungen von elektronischen Kassen gewähren soll. Softwareseitig werden Festschreibung, Versionierung und Historien genannt. Die Festschreibung beschreibt, dass nach endgültiger Auswertung eines Buchungsstapels dieser

für den Anwender gesperrt wird. Die Versionierung ist für die Erstellung von Geschäftsbriefen und Ähnliches vorgesehen und Historien sollen insbesondere bei der Arbeit mit Stammdaten die zu einem bestimmten Zeitpunkt geltenden Daten darstellen.

2.2 Steuerrechtliche Grundlagen

Für die steuerrechtliche Sicht auf die Grundlagen der Buchführung sind zum einen die als GoBD bezeichneten Ausführungen des BMF-Schreibens vom 28.11.2019 und der Ergänzung zu diesem Schreiben vom 11.03.2024 relevant sowie die Regelungen in der Abgabenordnung.
Alle handelsrechtlichen Vorschriften sind so wie im HGB auch im Steuerrecht zu finden. Dies ist daher notwendig, da diese nicht für alle steuerpflichtigen Unternehmen gelten. Dabei handelt es sich um die Nichtkaufleute, hierzu zählen Freiberufler, steuerlich buchführungspflichtige Unternehmen und Unternehmen, welche den Gewinn nach § 4 Abs. 3 Einkommensteuergesetz (EStG) ermitteln.
Nachfolgend wird in Abschnitt 2.2.1 erklärt, wann steuerrechtlich die handelsrechtlichen Vorschriften gelten, anschließend werden die ergänzenden Regelungen der AO in Abschnitt 2.2.2 vorgestellt, und in Abschnitt 2.2.3 wird auf technische Ergänzungen durch das BMF-Schreiben vom 29.11.2019 eingegangen. Abschließend werden in Abschnitt 2.2.4 die Konsequenzen des Verstoßes gegen die Vorgaben vorgestellt.

2.2.1 Nutzbarmachung außersteuerlicher Pflichten für das Steuerrecht

Der § 140 AO schreibt vor: „Wer nach anderen Gesetzen als den Steuergesetzen Bücher und Aufzeichnungen zu führen hat, die für die Besteuerung von Bedeutung sind, hat die Verpflichtungen, die ihm nach den anderen Gesetzen obliegen, auch für die Besteuerung zu erfüllen.“
Diese Regelung wird auch zu Beginn des Schreibens zu den GoBD vom Bundesministerium für Finanzen, 2019 (Rz. 3) genannt. Hier wird vor allem auf den § 238 ff. HGB verwiesen, welcher zuvor behandelt wurde. Hieraus lässt sich ableiten, dass Kaufleute im Sinne des HGBs verpflichtet sind, auch steuerrechtlich Bücher zur führen und auch die Vorschriften des Handelsrechts für die steuerrechtlichen Aufzeichnungen einzuhalten.
In den Vorgaben zur Ermittlung des steuerrechtlichen Gewinns in § 5 Abs. 1 S. 1 HS. 1 EStG wird hierzu vorgegeben, dass Steuerpflichtige, die auf Grundlage des § 140 AO Bücher führen müssen, den Gewinn der Handelsbilanz als Ausgangspunkt der steuerlichen Gewinnermittlung verwenden müssen. Der steuerliche Gewinn stimmt aber nicht

immer mit dem handelsrechtlichen Gewinn überein, da es bei der Bewertung bestimmter Sachverhalte unterschiedliche Regelungen zwischen Handels- und Steuerrecht gibt.

2.2.2 Ergänzende Vorschriften des Steuerrechts

Die steuerrechtlichen Vorschriften sind den handelsrechtlichen Vorschriften sehr ähnlich. Es werden aber noch zwei weitere einzuhaltende Vorschriften vorgegeben: zum einen die zeitgerechte Aufzeichnung, zum anderen die Lesbarkeit. Diese werden nachfolgend kurz erläutert.

Zeitgerechte Aufzeichnung

Die zeitgerechte Aufzeichnung ist im § 146 Abs. 1 AO geregelt. Das Bundesministerium für Finanzen, 2019 interpretiert die Regelung so, dass alle Geschäftsvorfälle zeitnah, am besten unmittelbar nach der Entstehung, in einem Grundbuch erfasst werden müssen. Diese Regelung betrifft zwar Geschäftsvorfälle, welche in der Buchführung später als Buchung erfasst werden, aber nicht zwingend die Buchung selbst, sondern auch eine Erfassung in einem Grundbuch, wie einem Rechnungsein- bzw. -ausgangsbuch oder Kassenbuch.

Lesbarkeit

Die Vorgabe der Lesbarkeit nach § 147 Abs. 5 AO regelt, dass derjenige, der seine Bücher digital führt und aufbewahrt, diese den Finanzbehörden in einer Form auszuhändigen hat, welche diese auswerten können. Alternativ muss er auf eigene Kosten Hilfsmittel für die Herstellung der Lesbarkeit zur Verfügung stellen.

2.2.3 Technische Ergänzungen

Mit dem BMF-Schreiben vom 29.11.2019 werden die zuvor vorgestellten Punkte weiter vertieft und noch weitere Pflichten eingeführt.
Neben der Beschaffenheit der Buchführung bestimmt das Bundesministerium für Finanzen, 2019 (Rz. 61 ff.), die Funktionen des Belegwesens und die an Belege gestellten Anforderungen. Die Regelungen umfassen die Pflicht, Belege zu kontieren, geordnet abzulegen und mit dem Buchungsdatum zu versehen. Aber auch die Belegsicherung wird beschrieben.
Weitere Regelungen des Schreibens sind die Einführung eines internen Kontrollsystems (Rz. 100 ff.), Vorschriften zur Sicherung der Daten vor Diebstahl, Verlust und Vernichtung (Rz. 103 ff.), zur Aufbewahrung der Daten (Rz.113 ff.) und zur Nachvollziehbarkeit und Nachprüfbarkeit (Rz. 145 ff.) durch unter anderem die Erstellung einer Verfahrensdoku-

mentation. Abschließend wird geregelt, wie der Datenzugriff durch die Finanzbehörden zu erfolgen hat (Rz.158 ff.)

2.2.4 Konsequenzen eines Verstoßes

Konsequenzen der Nichteinhaltung lassen sich bereits im § 158 Abs. 1 AO erahnen. Hier heißt es, dass die Buchführung der Besteuerung zugrunde zu legen ist, wenn die Vorgaben der §§ 140 bis 148 AO erfüllt sind. Entsprechend kann ein Verstoß dazu führen, dass die Besteuerungsgrundlage wegfällt.
Teutemacher, 2020 (S. 8 ff.), unterscheidet zwischen formellen und materiellen Mängeln, wobei formelle Mängel die Organisation der Buchführung, die Aufbewahrung oder die Einhaltung der Ordnungsvorschriften nach §§ 140 bis 148 AO betreffen. Bei den formellen Mängel kommt es für die Konsequenzen am Ende auf den Grad der Verletzungen an und darauf, ob diese die Nachvollziehbarkeit der Buchführung beeinträchtigen. Als Beispiele für materielle Mängel werden die Nichterfassung von Einnahmen, die Buchung von Einlagen und Entnahmen ohne Eigenbeleg, Verbuchung von Scheinrechnungen, falsche Buchungen und Manipulationen an Kassensystemen angegeben.
Nach Teutemacher, 2020 (S. 14) drohen bei formellen Mängeln keine Konsequenzen, sofern die Besteuerung trotz der Mängel nicht gefährdet ist. Bei materiellen Mängeln entsteht das Recht, eine Zuschätzung für den nicht ordnungsgemäßen Teil der Buchführung vornehmen zu dürfen. Bei gravierenden formellen oder materiellen Mängeln, bei denen die Grundlage für die Besteuerung nicht mehr angenommen werden kann, kann der Prüfer eine Vollschätzung der Besteuerungsgrundlage durchführen.
Weitere Konsequenzen zusätzlich zu den Schätzungen können die Einleitung eines Ordnungswidrigkeitsverfahrens nach § 379 Abs. 1 AO, ein Ermittlungsverfahren aufgrund von Steuerhinterziehung § 370 AO oder leichtfertiger Steuerverkürzung § 378 AO sowie die Androhung und Festsetzung von Zwangsgeldern bis zu 25.000,00 € nach §§ 328 ff. AO sein.

3 Anforderungen an eine GoBD-konforme Datenbank

In den nachfolgenden Abschnitten werden die funktionellen Anforderungen an eine GoBD-konforme Datenbank ermittelt. Hierzu werden im Abschnitt 3.1 die Anforderungen an die Datenhaltung aus dem BMF-Schreiben vom 28.11.2019 dargestellt. Anschließend werden im Abschnitt 3.2 Anforderungen an Datenbanken im Allgemeinen ermittelt und abschließend die Anforderungen, die sich durch den Einsatz in der Steuerberatungspraxis an das System ergeben, im Abschnitt 3.3.
Die Formulierung der Anforderungen wird in der Form von User Storys umgesetzt, welche dem Schablonenansatz nach Rupp, 2021 (S. 361 ff.) folgen. Hierbei wird die Anforderung aus Sicht des Systems formuliert. Die Priorität der Anforderung wird über das Verb der Formulierung bestimmt. Hierbei wird zwischen den Verben *Muss*, *Sollte* und *Wird* unterschieden. *Muss* wird für Anforderungen verwendet, welche zwingend umgesetzt werden müssen. *Sollte* wiederum für Anforderungen, bei denen eine Umsetzung wünschenswert ist, aber nicht zwingend erforderlich. Das Verb *Wird* wird für Anforderungen verwendet, welche in der Zukunft relevant werden, aber bei der aktuellen Umsetzung eventuell schon für den Entwickler im Bewusstsein sein sollten.
Anschließend wird formuliert, was das System für wen erledigen können soll. Durch den Schablonenansatz werden die Anforderungen stringent formuliert und sind gut miteinander vergleichbar.

3.1 BMF-Schreiben vom 28.11.2019

Aus den in Kapitel 2 genannten Grundsätzen sind in der Betrachtung der Datenhaltung insbesondere das Kriterium der Ordnung und das der Unveränderbarkeit interessant. Des Weiteren sind im BMF-Schreiben vom 28.11.2019 die Regelungen zum internes Kontrollsystem (IKS), Vorschriften zur Datensicherung und Regelungen zum Datenzugriff durch die Steuerbehörden vorgegeben.
Bezüglich des internen Kontrollsystems schreibt das Bundesministerium für Finanzen, 2019 (Rz. 100 ff.) vor, dass verschiedene Sachverhalte im Datenbestand geprüft und die

Prüfung protokolliert werden muss. Hierzu werden folgende Beispiele aufgezählt: Die Kontrolle der Zugangs- und Zugriffsberechtigungen, Prüfung von Funktionstrennungen, Erfassungskontrollen, Abstimmungskontrollen, Verarbeitungskontrollen und Schutzmaßnahmen vor Verfälschung von Daten, Programmen oder Dokumenten.

Die Vorgabe des Datenschutzes besagt, dass der Steuerpflichtige sich vor Datenverlust zu schützen hat, da er ansonsten nicht mehr auf die Buchführung als Grundlage seiner Besteuerung zurückgreifen kann (Bundesministerium für Finanzen, 2019 Rz. 103 ff.).

Darüber hinaus dürfen die Finanzbehörden die Daten des Steuerpflichtigen prüfen. Hierbei definiert das Bundesministerium für Finanzen, 2019 (Rz. 158 ff.), drei Arten des Datenzugriffs. Beim unmittelbaren Datenzugriff greift die Finanzbehörde rein lesend mit der Hard- und Software des Steuerpflichtigen auf die Buchführungsdaten zu. Ein Fernzugriff ist ausgeschlossen. Beim mittelbaren Datenzugriff hat der Steuerpflichtige die Daten der Finanzbehörde derart aufzubereiten, wie dies von der Finanzbehörde vorgegeben wird. Die letzte Zugriffsart ist die Datenträgerüberlassung, bei der die Finanzbehörde vom Steuerpflichtigen einen Datenträger mit den Daten erhält und diesen auch für die Prüfung aus dem Betrieb des Steuerpflichtigen mitnehmen darf.

Tabelle 3.1: Anforderungen an die Datenhaltung aus den GoBD

Titel	**User-Story**	Rz.
Ordnung	Die Datenhaltung **muss** eine Ordnung der Buchungssätze ermöglichen. Hierfür wird die Führung eines Journals mit allen Buchungssätzen und eine Aufteilung der Buchungen auf unterschiedliche Konten vorgeschrieben.	53-57
Unveränderbarkeit	Die Datenhaltung **muss** sicherstellen, dass Ursprungsbuchungen nicht gelöscht werden können und nachvollziehbar ist, ob Änderungen ursprünglich sind oder später vorgenommen wurden.	58-60 107-112
IKS	Die Datenhaltung **muss** Kontrollen am Datenbestand durchführen und protokollieren können.	100-102
Datensicherheit	Die Datenhaltung **muss** durch den Steuerpflichtigen gegen Verlust gesichert werden können.	103-106
Datenzugriff	Die Datenhaltung **muss** den Datenzugriff durch die Finanzbehörden auf den vorgegebenen Wegen ermöglichen.	158-170

Die Anforderungen, welche sich aus den GoBD für die Datenhaltung ergeben, sind in der Tabelle 3.1 zusammengefasst. In der letzten Spalte ist die jeweilige Fundstelle der Regelung im BMF-Schreiben angegeben.

3.2 Anforderungen an Datenbanken

Nachfolgend werden die Anforderungen an eine Datenbank formuliert. Hierzu soll zunächst betrachtet werden, was eine Datenbank überhaupt ist.

> „Eine Datenbank ist eine Sammlung von Daten, die untereinander in einer logischen Beziehung stehen und von einem eigenen Datenbankverwaltungssystem (Database Management System DBMS) verwaltet werden.“ (Schicker, 2017 S. 3)

Hierzu führt Schicker, 2017 (S. 3) weiter aus, dass sich ableiten ließe, dass eine Datenbank aus Daten und einem zugehörigen Verwaltungssystem besteht. Das Verwaltungssystem stellt die Schnittstelle nach außen für die Anwendungsprogramme dar und steuert den Zugriff auf die Daten. Des Weiteren sollen Daten, die nicht zueinander gehören, in separaten Datenbanken gespeichert werden.
Nachfolgend werden im Abschnitt 3.2.1 die Anforderungen an Datenbanken im Allgemeinen und anschließend in Abschnitt 3.2.2 die Anforderungen bezüglich der Datensicherheit analysiert.

3.2.1 Allgemeine Anforderungen an Datenbanken

Tabelle 3.2: Anforderungen an Datenbanken nach Schicker, 2017

Titel	**User-Story**
Sammlung logisch verbundener Daten	Das Datenbanksystem **muss** in der Lage sein, nur zusammengehörige Daten zu speichern und nicht zusammengehörige Daten getrennt zu verwalten
Abfragemöglichkeit und Änderbarkeit von Daten	Das Datenbanksystem **muss** die Möglichkeit bieten, Daten abzufragen und zu ändern.
Speicherung mit wenig Redundanz	Das Datenbanksystem **sollte** in der Lage sein, Daten wenn möglich nur einmalig zu speichern und somit den benötigten Speicherplatz zu reduzieren.
logische Unabhängigkeit der Daten von der physischen Infrastruktur	Das Datenbanksystem **muss** die Möglichkeit bieten, auf die Daten zuzugreifen, ohne dass der Anwender die Organisation der Daten auf dem Server kennen muss.
Integrität	Das Datenbanksystem **sollte** möglichst immer korrekte Daten speichern und zurückgeben.
Mehrfachzugriff	Das Datenbanksystem **sollte** die Möglichkeit bieten, dass mehrere Benutzer gleichzeitig mit ihm arbeiten können.
Kontrolle	Das Datenbanksystem **sollte** die Möglichkeit bieten, den derzeitigen Zustand der Datenbank einsehen zu können.

Die Anforderungen an eine Datenbank im Allgemeinem wurden von Schicker, 2017 (S. 6 ff.) bereits zusammengetragen. Diese sind in der Tabelle 3.2 zusammengefasst:

3.2.2 Anforderungen an die Datensicherheit in Datenbanken (ACID)

Das Thema Datensicherheit ist im Allgemeinen das Hauptthema bei der Frage der Datenhaltung. Hierzu führt Kofler, 2024 (S. 68 f.) an, dass mehrere Themen in diesem Bereich zusammenkommen. Zum einen nennt er die Benutzerverwaltung, Logging und Backups, aber auch den Themenbereich der Replikation und High Availability. Das Thema Foreign Keys, welches ebenfalls als sicherheitsrelevant eingestuft wird, bezieht sich speziell auf relationelle Datenbanken und wird daher nachfolgend nicht näher erläutert.
Neben den zuvor genannten Punkten wird das Akronym ACID als Datensicherheitsmerkmal für Datenbanken, bei denen mehrere Nutzer parallel auf die Daten zugreifen können, genannt. ACID steht hierbei für Atomicity, Consistency, Isolation und Durability. Die ge-

wünschten Eigenschaften sind nachfolgend kurz zusammengefasst (Kofler, 2024 S. 69 ff.):

Atomicity
: Eine Transaktion wird immer im Ganzen oder gar nicht verarbeitet. Eine Verarbeitung eines Teils der Transaktion wird nicht durchgeführt.

Consistency
: Eine Transaktion darf nur ausgeführt werden, wenn die Konsistenz der Datenbank dadurch nicht beeinträchtigt wird. Es darf beispielsweise kein Datensatz gelöscht werden, auf den ein anderer Datensatz weiterhin verweist, da ansonsten dieser Datensatz unvollständig wird.

Isolation
: Transaktionen, die parallel ausgeführt werden, dürfen sich nicht gegenseitig in ihren Ergebnissen beeinflussen. Zwei Transaktionen dürfen nicht zeitgleich auf denselben Wert zugreifen und diesen ändern.

Durability
: Transaktionen, welche von der Datenbank als ausgeführt gemeldet werden, müssen auch wirklich vollständig ausgeführt sein.

Nachfolgend sollen die weiterhin für Datensicherheit sorgenden Bestandteile, welche eingangs erwähnt wurden, näher betrachtet werden, beginnend mit dem Thema Benutzerverwaltung. Hierzu führt Kofler, 2024 (S. 519 ff.) aus, dass die Benutzerverwaltung zum einen aus der Authentifizierung der Benutzer, also einem Schutz vor Zugriff durch der Datenbank unbekannte Personen, als auch der Möglichkeit, verschiedene Rollen und Privilegien an die Nutzer zu verteilen, besteht. Hierdurch kann nicht nur sichergestellt werden, dass nur berechtigte Personen überhaupt Zugriff auf die Datenbank haben, sondern auch, dass nicht jeder Nutzer alle Transaktionen auf der Datenbank ausführen kann und somit eventuell aus Versehen Schaden an der Datenbank anrichtet.
Logging und Backup werden in einem Punkt genannt, stehen aber für zwei separate Mechanismen, welche eng miteinander zusammenarbeiten. Logging beschreibt hierbei Events, welche die Datenbank betreffen, in eine Datei zu schreiben. Es kann unterschiedliche Logs geben: Gängig sind Update-Logs, Transaktions-Logs und Fehler-Logs. Besonders wichtig ist das Update-Log, welches die Änderungen an der Datenbank protokolliert und als Grundlage für die Backups dient. Backups sichern den Zustand des Systems und ermöglichen, dieses bei Datenverlust wieder herzustellen (Kofler, 2024 S. 539 ff.).
Replikation beschreibt die Spiegelung des Zustands eines Datenbankservers durch einen zweiten Datenbankserver. Hierdurch kann die Ausfallsicherheit erhöht werden. Wenn der

erste Server ausfällt, kann der zweite einspringen. Lastverteilung wird möglich, denn Anfragen an die Datenbank können von allen Servern, auf denen diese vorliegt, bearbeitet werden. Zudem können weit auseinanderliegende Firmenstandorte miteinander verknüpft werden, da ansonsten eventuell hohe Latenzzeiten für Zugriffe entstehen könnten. Hiermit verbunden beschreibt der Begriff High Availability, dass der Replikationsserver bei Ausfall des Hauptservers automatisiert dessen Aufgaben nahtlos übernimmt (Kofler, 2024 S. 555 ff.).
In der nachfolgenden Tabelle 3.3 sind die Anforderungen, welche sich aus der Betrachtung der Datensicherheit für Datenbanken ergeben, in Form von User-Storys abgebildet:

Tabelle 3.3: Anforderungen an Datenbanken für die Datensicherheit nach Kofler, 2024

Titel	**User-Story**
ACID-konform	Das Datenbanksystem **muss** ACID-konform sein.
Benutzerverwaltung	Das Datenbanksystem **muss** die Möglichkeit bieten, den Zugriff auf die Daten nur bestimmten Personen zu gewähren und die Berechtigungen dieser Personen müssen steuerbar sein.
Backupfähigkeit	Das Datenbanksystem **muss** die Möglichkeit bieten, Backups zur Datensicherung zu erstellen.
Logging	Das Datenbanksystem **sollte** Logs sowohl über die zu dem Zustand geführten Befehle als auch die beim Betrieb angefallenen Fehler führen.
Replizierbarkeit	Das Datenbanksystem **sollte** die Möglichkeit bieten, Replikationsserver zu betreiben.
High-Availability	Das Datenbanksystem **sollte** die Möglichkeit bieten, bei Ausfall des Hauptservers einen Ersatzserver einspringen zu lassen.

3.3 Anforderungen durch die Steuerberatungstätigkeit

In der Berufsbeschreibung des Steuerberaters auf der Seite der Bundessteuerberaterkammer werden die nachfolgenden Aufgabenbereiche für die Arbeit als Steuerberater und somit auch für die Steuerberatungskanzlei benannt (Bundessteuerberaterkammer, o. d.):

- Vertretung und Beratung von **Mandanten** in Steuersachen
- Bearbeitung von Steuerangelegenheiten und Hilfeleistung bei der Erfüllung der steuerlichen Pflichten der **Mandanten**
- Hilfeleistung bei der Erfüllung der Buchführungspflichten des **Mandanten**

- Aufstellung von Bilanzen
- steuerrechtliche Beurteilung von Bilanzen
- Hilfeleistung in Steuerstrafsachen und Bußgeldangelegenheiten

Der Beruf des Steuerberaters deckt entsprechend eine große Anzahl an Aufgaben für mehrere Kunden, welche Mandanten genannt werden, ab. Durch das breite Aufgabenspektrum und die Vertrauensstellung des Berufs entstehen zusätzliche Anforderungen an die Datenbank, welche in der nachfolgenden Tabelle 3.4 in der Form der User-Storys aufgeführt sind. Hierbei sind diese nach absteigender Wichtigkeit sortiert.

Tabelle 3.4: Anforderungen an Datenbanken aus Sicht der Steuerberatung

Titel	**User-Story**
GoBD-konform	Das Datenbanksystem **muss** GoBD-konform sein.
Mehrmandatsfähigkeit	Das Datenbanksystem **muss** die Möglichkeit bieten, die Daten mehrerer Mandate separat zu speichern.
Sicherheit	Das Datenbanksystem **muss** die Sicherheit der Daten gewährleisten.
Anpassbarkeit	Das Datenbanksystem **sollte** die Möglichkeit bieten, dass mehrere unterschiedliche Programme ihre Daten für den Programmzweck auf dem effektivsten Weg sichern können.
Stammdaten	Das Datenbanksystem **sollte** die Möglichkeit bieten, Stammdaten und deren Historie verwalten zu können.
KI	Das Datenbanksystem **wird** die Möglichkeit bieten, Daten auch durch KI-Anwendungen analysieren zu lassen.

4 Problemfeld GoBD

Nachfolgend wird aufgezeigt, warum die GoBD ein Problem bei der Entwicklung von Software darstellen könnten und warum klassische Datenbanksysteme die GoBD nicht von Haus aus abdecken. Dazu wird zunächst in Abschnitt 4.1 darauf eingegangen, warum gängige Datenbanksysteme die GoBD nicht umsetzen werden und können. Anschließend wird in Abschnitt 4.2 auf Möglichkeiten für die Umsetzung der GoBD mit gängigen Datenbanksystemen eingegangen. Als Abschluss des Kapitels in Abschnitt 4.3 wird die Idee vorgestellt, mit der eine GoBD-konforme Datenbank ohne zusätzlichen Aufwand für den Entwickler erstellt werden soll.

4.1 Gängige Datenbanksysteme

Datenbanksysteme haben das Ziel, möglichst viele Anwendungsfelder abzudecken. Hierbei bilden die GoBD als nationales Recht, welches nur für Steuerpflichtige in Deutschland gilt, eine Regelung, welche für das internationale Geschäft von untergeordneter Bedeutung sein sollte.

In Deutschland waren im Jahr 2022 insgesamt 8.409.661 Unternehmen steuerpflichtig, wie das Bundesministerium für Finanzen, 2023 im Bericht der Betriebsprüfungen für 2022 aufführt. Hierzu zählen auch internationale Firmen, welche nur eine Steuerpflicht in Deutschland haben, aber nicht ihren Sitz. Mit Verweis auf den § 140 AO muss gesagt werden, dass für diese Steuerpflichtigen auch die GoBD gelten.

Insgesamt machen deutsche Unternehmen nur einen Anteil von 3,15 % (IMF, 2024) am weltweiten Bruttosozialprodukt aus. Durch eine reine GoBD-Datenbank würden sich Datenbankanbieter auf ein sehr kleines Segment der weltweiten Unternehmen beschränken oder müssten für jedes Land eine eigene Datenbankversion anbieten. Dies wäre sehr umständlich und aus betriebswirtschaftlicher Sicht nicht sinnvoll.

Viele Datenbanken besitzen zwar einen Log, welcher die Eingaben, die für die Erreichung des Zustandes benötigt wurden, aufzeichnen, aber für alle Datenbanken gibt es Anleitungen, wie man diesen Log löschen kann, ohne die Daten in der Datenbank zu löschen. Hierdurch ist der Grundsatz der Unveränderbarkeit nicht mehr gegeben.

Am Ende muss die Umsetzung der GoBD durch den Softwarehersteller durchgeführt werden, der die Datenbank verwendet, und somit muss der Steuerpflichtige auf dessen Expertise vertrauen. Ansonsten kann das gekaufte Programm zu einer enormen Steuernachzahlung führen.

4.2 Umsetzung der GoBD in gängigen Datenbanken

Die Umsetzung der GoBD hängt, wie zuvor beschrieben, bei der Verwendung einer Multi-Purpose Datenbank vom Entwickler der Anwendung, welche die Datenbank verwendet, ab. Einige Punkte, die hierbei beachtet werden sollten, sind die nachfolgenden:

Verwendung der Delete-Anweisungen vermeiden

Da laut den GoBD keine Buchungen spurlos gelöscht werden dürfen, darf die Delete-Anweisung der Datenbank nicht verwendet werden. Anstatt den Datensatz zu löschen, sollte entweder eine Gegenbuchung als zusätzliche Buchung, welche die erste Buchung neutralisiert, oder ein Löschvermerk als zusätzliches Feld verwendet werden.
Werden Änderungen an einer Buchung vorgenommen, sollte dies ebenfalls als eine Extra-Buchung vermerkt werden, welche in Addition mit der Ursprungsbuchung den gewünschten Zustand herbeiführt.

Rollen und Authentifizierung nutzen

Durch Rollen und Authentifizierung kann der Zugriff für andere Nutzer auf die Datenbank beschränkt werden. Hierdurch kann bereits auf der Ebene der Datenbank verhindert werden, dass der Nutzer Funktionen ausführt, welche den GoBD widersprechen. Zum Beispiel sollte der Nutzer keine Datenbanken löschen können und auch nicht einzelne Datensätze.
Bei Verwendung von Festschreibungskennzeichen sollten diese nicht durch den Nutzer rückgängig gemacht werden können. Wenn diese gesetzt sind, sollten alle Zugriffe auf die Buchung gesperrt werden.

Eingeschränkte Datenbankverwaltungstools

Eine weitere Möglichkeit, den Nutzer an schädlichen Zugriffen auf die Datenbank zu hindern, ist die, kein vollständiges Datenbankverwaltungssystem mit der Datenbank auszuliefern. Neben der Möglichkeit der Festschreibungskennzeichen und beschränkten Befugnissen ist dies einer der Wege, welche die Datev EG gewählt hat. Diese liefert einen SQL-Manager zu der Microsoft SQL Datenbank, welche in ihren Programmen genutzt wird, aus, der nicht alle Funktionen des Standard SQL Managers von Microsoft enthält (Datev eG, 2024).

Weiter wird das Administrator-Passwort für die Datenbank von der Datev eG nicht an den Kunden weitergegeben, da hierdurch die Nachweiswirkung der Programme nicht mehr gegeben wäre.

4.3 Lösungsversuch: Auf Blockchain basierende Datenbank

Damit nicht für jedes Anwendungsprogramm eine individuelle Lösung für die Umsetzung der GoBD entwickelt werden muss, sollen die technischen Aspekte der GoBD in der Datenhaltung umgesetzt werden. Hierzu soll eine Datenbank basierend auf den Technologien, welche bei der Blockchain Anwendung finden, erstellt werden.
Hierzu soll eine Lösung entwickelt werden, die nicht selber als Datenbank fungiert, sondern als Fassade vor eine Datenbank geschaltet werden kann und somit die Befehle, die in die Datenbank eingeflossen sind, abspeichert, ohne die Datenbank selber zu beeinflussen. Vorteil einer solchen Lösung ist, dass diese ungeachtet der Form der Datenbank genutzt werden kann und somit auch gängige Datenbanken ohne großen Aufwand für den Entwickler gegen GoBD-Verstöße sichern kann.
Somit soll die Last der Umsetzung der GoBD vom Anwendungsentwickler genommen werden und der Steuerpflichtige ein wenig mehr Sicherheit in den steuerlichen Pflichten seines Unternehmens erhalten.

5 Blockchain-Technologien

Um die in Kapitel 3 beschriebenen Anforderungen zu erfüllen, sollen Teilbereiche des Technologie-Stacks der Blockchain-Technologie eingesetzt werden. Blockchains haben in den letzten Jahren massiv an Bekanntheit gewonnen, da sie die Grundlage von Kryptowährungen bilden, welche in den letzten Jahren einen enormen Nutzerzuwachs erfahren haben. Wie eine Umfrage von Statista Consumer Insights, 2023, ergeben hat, ist der Bitcoin nach wie vor die bekannteste Kryptowährung. Die beim Bitcoin verwendete Blockchain soll nachfolgend als Referenz für den Technologie-Stack verwendet werden.
Zuerst wird im Abschnitt auf ausgewählte Bestandteile der Bitcoin-Blockchain eingegangen, und diese werden kurz erläutert. Danach wird im Abschnitt 5.2 das Zusammenspiel der einzelnen Komponenten erläutert und erklärt, wie hieraus eine Sicherheit für die Nutzer entsteht.

5.1 Einzelne Bestandteile

Die Referenzimplementierung des Bitcoins wurde 2009 mit dem Bitcoin-Whitepaper öffentlich gemacht. Diese erste Implementierung wurde als Open-Source Projekt unter MIT-Lizenz veröffentlicht. Durch Weiterentwicklungen und Anpassungen hat diese sich mit der Zeit entwickelt und wird heute als Bitcoin-Core bezeichnet (Antonopoulos, 2018, S. 33 ff.).
In den nächsten Abschnitten werden ausgewählte Bestandteile der Bitcoin-Blockchain beschrieben. Orientiert wird sich hierbei an dem Bitcoin-Whitepaper von 2009.

5.1.1 Transaktionen

Da es sich bei der Bitcoin-Blockchain um die Grundlage einer Kryptowährung handelt, ist es das Ziel, Transaktionen sicher abzuwickeln. Hierzu schreibt Nakamoto, 2009 (S. 2 u. S. 5), dass Transaktionen im Sinne der Blockchain eine Reihe von signierten Hashwerten sind. Der Besitzer eines virtuellen Coins kann diesen an einen anderen Teilnehmer der Blockchain weiterreichen, indem er aus dem öffentlichen Schlüssel des Zahlungsempfangenden und dem Hashwert (siehe hierzu 5.1.2) der Transaktion, mit dem er selber

den Coin erhalten hat, einen neuen Hashwert berechnet. Der neue Hashwert wird dann an den Coin angehängt und im Blockchain-Netzwerk veröffentlicht. Über seinen privaten Schlüssel kann der Empfänger nun prüfen, ob die Transaktion in der Blockchain eingestellt wurde.

Antonopoulos, 2018 (S. 20 ff.) beschreibt, dass eine Transaktion immer im Ganzen verarbeitet wird. Vergleichbar ist dies mit Münzen, welche auch einen festen Wert haben. Die Transaktion kann aber beim Weitersenden in mehrere Transaktionen aufgeteilt werden. So wird bei einer Zahlung eines kleineren Betrages als dem der Vorgängertransaktion eine Art Wechselgeldtransaktion erstellt, welche an den Zahlenden selbst als Zahlungsempfänger gerichtet ist. Andersherum können auch mehrere kleinere Beträge zu einem großen Betrag zusammengefügt werden.

5.1.2 Hashwertfunktionen

Hashwertfunktionen (auch Streuwertfunktionen) bilden eine beliebige Eingabemenge auf einen festgelegten Ausgabebereich ab. Eine gute Hashwertfunktion besitzt eine hohe Streuung, das bedeutet, dass bereits eine kleine Änderung eine große Wirkung in der Ausgabe nach sich zieht. Ebenso wichtig ist ein gewisser Grad an Kollisionssicherheit. Diese beschreibt den Umstand, dass möglichst wenige Eingaben dieselbe Ausgabe erzeugen. Da es im Prinzip unendlich viele verschiedene Eingaben gibt, welche auf einem endlichen Raum dargestellt werden sollen, kann aber eine doppelte Vergabe eines Hashwertes nicht ausgeschlossen werden. (Fill und Meier, 2020, S. 5 f.).

In der nachfolgenden Tabelle 5.1 wird die Streuung der in der Bitcoin verwendeten Hashfunktion SHA-256 aufgezeigt. Hier sind die beiden Hashwerte des Wortes Blockchain einmal mit großem und einmal mit kleinen *B* dargestellt. Für die Darstellung wurden die letzten Zeichen gekürzt. Es entsteht durch die kleine Änderung ein komplett anderer Wert.

Tabelle 5.1: Beispiel der Streuwirkung des SHA-256 Algorithmus

Eingabe	**Hashwert**
blockchain	ef7797e13d3a75526946a3bcf00daec9fc9c9c4d51ddc7cc5df888f74dd4...
Blockchain	625da44e4eaf58d61cf048d168aa6f5e492dea166d8bb54ec06c30de07db...

Durch die oben genannten Eigenschaften können mit Hashfunktion Änderungen in Dokumenten oder im Falle der Blockchain in den Blöcken nachgewiesen werden. Sie bilden mit eines der Fundamente der Blockchain. Die nachfolgenden Bestandteile nutzen alle Hashwertfunktionen.

5.1.3 Mining

Als Mining oder auch Proof-of-Work wird der Mechanismus bezeichnet, welcher bei der Bitcoin-Blockchain für die Sicherheit der Transaktionen zuständig ist. Das mit dem Mining gelöste Problem ist, dass Gelder nicht mehrfach ausgegeben werden können. Hierzu erklärt Nakamoto, 2009, dass immer die Transaktion eines Coins gelten solle, welche als Erste durchgeführt wurde. Alle Transaktionen werden im Bitcoin-Netzwerk veröffentlicht.

Spezielle Knotenpunkte sammeln alle während eines festgelegten Intervalls angefallenen Transaktionen in einem Block. Ein Block besteht aus: einem Blockheader, welcher die Meta-Daten des Blocks speichert, wie die Version der Implementierung unter der der Block erstellt wurde, den Hashwert des Vorgängerblocks, die Merkle-Root aller Transaktionen im Blockbody, einem Timestamp, der den Zeitpunkt der Erstellung des Blocks angibt, dem Difficulty-Target und der Number only used once (Nonce); sowie dem Body des Blocks, in dem die Transaktionen gespeichert werden (Antonopoulos, 2018, S. 198 f.).

Das Difficulty-Target wird von den Minern selber angepasst, sodass möglichst alle zehn Minuten ein neuer Block entsteht. Aus dem Target lässt sich ein Zielwert für den Hash des Blocks berechnen, dieser Zielwert muss durch den Hash unterschritten werden. Um die Zielvorgabe erfüllen zu können, wird von den Minern die Nonce so lange angepasst, bis der Zielwert unterschritten wird. Wenn die Erreichung des Zielwertes durch eine Veränderung der Rechenleistung oder der Anzahl der Teilnehmer im Mining aus dem Takt gerät, wird alle 2.016 Blöcke eine Anpassung der Schwierigkeit vorgenommen. Geht es zu schnell, wird der Zielwert nach unten verschoben, geht es zu langsam, wird ein höherer Zielwert vorgegeben (Antonopoulos, 2018, S. 230 ff.).

Für die verrichtete Arbeit erhalten die Miner eine Belohnung. Diese wird bereits vor dem Beginn des Minings als Transaktion in den Block eingetragen und berechnet sich aus einer festen Belohnung, die alle 210.000 Blöcke halbiert wird, und den Transaktionsgebühren, welche in den Transaktionen vorhanden sind (Nakamoto, 2009).

5.1.4 Merkle-Trees

Merkle-Trees werden in der Blockchain genutzt, um die in einem Block enthaltenen Transaktionen zu einem einzelnen Hashwert zusammenzufassen. Es handelt sich hierbei um einen Binärbaum, der mit den folgenden Regeln berechnet wird:

1. Die Ursprungswerte werden gehasht.

2. Es werden immer die Hashwerte eines Paares (1. und 2. Eintrag, 3. und 4., ...) konkateniert und aus diesem Wert wieder ein Hashwert gebildet. Sollte die Anzahl der Elemente ungerade sein, wird das letzte Element mit sich selbst konkateniert.
3. Die in Schritt zwei erhaltenen Hashwerte bilden die nächste Ebene des Baumes.
4. Die Schritte 2 und 3 werden nun mit den aus dem letzten Durchgang berechneten Hashwerten so lange wiederholt, bis nur noch ein Wert vorhanden ist.

Der einzelne Hashwert, der am Ende übrigbleibt, wird als Merkle-Root bezeichnet (Fill und Meier, 2020 S. 8 f.).

5.2 Funktionsweise von Blockchains

Die zuvor genannten Bestandteile sind nicht als abschließende Aufzählung zu verstehen, sind aber für die Schaffung von Vertrauen im Netzwerk von großer Bedeutung. Das größte Problem bei der Schaffung einer digitalen Währung ohne zentrale Autorität ist laut Nakamoto, 2009, dass kontrolliert werden muss, dass Geld nicht zweimal ausgegeben wird. Dies wird durch den Konsens innerhalb der Blockchain erreicht, dass immer die erste Transaktion des Coins ausgewertet wird. Nachfolgend wird erklärt, wie die Blockchain die Sicherheit der Transaktionen gewährleistet.
Dadurch, dass die Blöcke im Header den Hashwert des Vorgängerblocks enthalten und diesen für die Berechnung des eigenen Hashwertes nutzen, entsteht eine lange Kette von Blöcken, welche sich bis auf den Anfang der Blockchain, den sogenannten Genesisblock, zurückverfolgen lässt (Antonopoulos, 2018, S. 197 ff.).
Die Sicherheit, dass die Blockchain nicht einfach geändert werden kann, entsteht bei Bitcoin auch durch den Proof of Work Mechanismus, da eine Änderung an einer Transaktion dazu führen würde, dass alle nachfolgenden Blöcke neu gemint werden müssten. Hierdurch würde ein enormer Rechenaufwand entstehen (Antonopoulos, 2018 S. 197 ff.).
Wenn kein neues Mining durchgeführt werden würde, könnte jeder Rechner den geänderten Block als verändert erkennen, da es zwar aufwendig ist, einen neuen Block in die Chain einzufügen, aber das Überprüfen, ob der Hashwert mit den angegebenen Daten auch dem Target entspricht, in wenigen Millisekunden erledigt werden kann (Antonopoulos, 2018 S. 197 ff.).
Die Blockchain dient im Falle des Bitcoins als großes Kassenbuch. Alle Transaktionen werden hier erfasst und können von allen am Netz Beteiligten nachvollzogen werden. Durch den Proof of Work kann jeder Teilnehmer darauf zählen, dass eine Transaktion in einem Block auch gesichert ist. Je mehr Blöcke auf den Block folgen, in dem sich eine Transaktion befindet, desto sicherer ist diese ((Antonopoulos, 2018 S. 197 ff.).

6 Beschreibung des Prototyps

Das Ziel des Prototyps ist es, aufzuzeigen, dass die Nutzung einer an die Blockchain angelehnten Datenstruktur die Prüfung, ob an den Daten Änderungen durchgeführt wurden, vereinfachen. Somit soll eine erhöhte Sicherheit in Rahmen der GoBD gewährt werden. Als sekundäres Ziel soll der Prototyp so gestaltet sein, dass eine Nutzung möglichst mit bekannten Datenbanken vergleichbar ist.
In den nachfolgenden Abschnitten wird der erstellte Prototyp dargestellt. Der vollständige Quellcode wird im Anhang A.1 erläutert. Im nachfolgenden Abschnitt 6.1 wird auf die Architektur des Prototyps eingegangen. Anschließend wird im Abschnitt 6.2 die Verwendung der Blockchain im Prototyp erläutert. Im Abschnitt 6.3 wird abschließend auf die Möglichkeiten zur Nutzung des Prototyps in eigenen Programmen eingegangen.

6.1 Architektur

Der Prototyp wurde gemäß der in Fowler, 2013 (S. 17 ff.) vorgestellten Schichtenarchitektur entwickelt. Hierbei wurden die drei Schichten Data-Source, Domain und Presentation verwendet. Die Namen wurden im Programm nicht direkt verwendet. Die Tabelle 6.1 zeigt sowohl die Namen der Schichten im Prototyp als auch die Art ihrer Umsetzung im Prototyp auf. Insbesondere wurde bei der Entwicklung beachtet, dass jede Schicht nur die Schicht unter sich kennt und der Zugriff nur einseitig erfolgt. Dazu wurden die Schichten in zwei separaten Projekten entwickelt und die Domain-Schicht später als Klassenbibliothek in die Präsentationsschicht eingebunden.

Tabelle 6.1: Umsetzung der Schichten nach Fowler, 2013 im Prototyp

Originalschichten	**Namespace im Projekt**	**Umsetzung**
Presentation	Presentation_Layer	Programmierung
Domain	Data_Security_Layer	Programmierung
Data Source	.bcdb-Dateien	Dateiebene

Die Data Source wird im Prototyp durch die Menge der Dateien auf der Festplatte dargestellt. Es wird je eine Datei für den Genesisblock (siehe 6.2.1), die Chains (siehe 6.2.2)

und die einzelnen Blöcke (siehe 6.2.3) erstellt. Die Dateien werden mit dem Dateitypen .bcdb versehen. Es handelt sich um einfache Dateien mit einem JSON-String im UTF-8 Encoding, sie können daher mit jedem Texteditor geöffnet werden. Auf Maßnahmen, die dazu führen, die Daten nicht menschenlesbar darzustellen oder zu verschlüsseln, wurde im Rahmen der prototypischen Entwicklung verzichtet.

Die Data-Security-Layer beinhaltet die Logik der Blockchainumsetzung und sichert die Daten, welche in die Datenbank einfließen. Sie beinhaltet die logischen Bestandteile der Blockchain. Nach jeder Übergabe neuer Daten werden die Blöcke direkt gespeichert und somit der Fortschritt sofort persistiert. Eine Löschung von Daten ist derzeit nicht vorgesehen. Den Namen hat die Schicht erhalten, da sie für die Sicherheit der Daten im Rahmen der GoBD zuständig ist.

Die Presentation-Layer besteht aus drei Teilen: zum einen einer Schnittstelle für die Nutzung in anderen Programmen, welche Funktionen für die Ausführung der Create/Read/Update/Delete (CRUD)-Operation bereitstellt; zum anderen aus einer Zuständigkeitskette wie in Geirhos, 2021 (S. 211 ff.) beschrieben, welche aus drei Bearbeitern besteht. Ein Bearbeiter ist für die Ausführung von Befehlen im Datenbankbereich, einer für die Befehle mit Datenbezug zuständig und der Letzte, um die Befehle an die Data-Security-Layer zu übergeben. Werden die Daten geladen, wird eine verkürzte Zuständigkeitskette ohne erneute Sicherung in die Data-Security-Layer aufgerufen. Der letzte Teil wird von der In-Memory-Datenbank gebildet, welche je nach Nutzen verschiedene Formen annehmen kann.

Neben der Aufgabe eine Anpassbarkeit, an verschiedene Szenarien zu bieten, soll die In-Memory-Datenbank auch die Zugriffsgeschwindigkeiten steigern. Die Variante der Umsetzung wurde gewählt, da die In-Memory-Datenbank stateless ist und damit kein zusätzlicher Speicherplatz für die Speicherung des Zustandes verwendet wird. Es wäre auch möglich gewesen eine Datenbank mit State zu verwenden und nur die Eingaben in der Blockchain zusätzlich zu sichern, aber hierdurch wären zusätzliche Daten angefallen und mehr Speicherplatz verbraucht worden. Dies könnte bei größeren Systemen ein Problem darstellen.

Für die Persistierung der Daten wird Event-Sourcing genutzt, wie in Vernon, 2015 (S. 160 ff.) dargestellt. Die Sicherung der Daten wird nicht direkt vorgenommen, sondern der Entstehungsweg der aktuellen Daten wird in Form der Befehle, welche zu dem Zustand geführt haben, aufgezeichnet. Die Integrität der gespeicherten Befehle wird über die durch Hashwerte verknüpften Blöcke, in denen die Transaktionen gespeichert sind, sichergestellt. Beim Laden einer Datenbank wird daher die komplette Chain geladen und verarbeitet, um den aktuellen Stand wieder herzustellen, indem die Befehle erneut in derselben Reihenfolge auf eine leere Datenbank ausgeführt werden.

6.2 Anwendung Blockchain

In den nachfolgenden Abschnitten wird die Umsetzung der Blockchain-Technologien in der Anwendung dargestellt. Insbesondere wird auf die Besonderheiten der erstellten Blockchain eingegangen. Typische Elemente wie die Hashwerte und Merkle-Trees werden nicht noch einmal gesondert behandelt, da hier keine Abweichung zu der in Kapitel 5 vorgestellten Referenz vorliegt.

6.2.1 Genesisblock

Der Genesisblock wurde im Prototyp so implementiert, dass dieser als Basis des Datenbanksystems fungiert. Alle im System verwendeten Chains haben ihren Ursprung in einem Genesisblock.

Der Hashwert dieses Blocks wird aus dem Namen des Datenbanksystems und einem Schlüsselwort berechnet. Über die Eingabe dieser beiden Werte kann die entsprechende Datei wieder aufgefunden werden. In Anlehnung an die Bitcoin-Referenz besteht der Header des Blocks aus dem Hash des Genesisblocks, einer Versionsnummer, der Merkle Root der enthaltenen Transaktionen, einem Timestamp und einer Liste der dem Block nachfolgenden Chains. Im Body befinden sich die entsprechenden Transaktionen, die generiert werden, wenn eine neue Chain erstellt wird.

Die Erstellung einer neuen Chain funktioniert, indem der Genesisblock den Namen der Chain sowie einen Schlüssel für die Chain übergeben bekommt. Der Name der Chain wird in dem String *NewChain:+Name*, wobei Name der Name der Chain ist, in die Transaktionen eingetragen. Es wird die Merkle Root der Transaktionen berechnet und anschließend aus der Merkle Root und dem Schlüssel der Hashwert der Beschreibungsdatei der Chain ermittelt.

Soll nun auf eine Chain zugegriffen werden, werden dem Genesisblock Name und Schlüssel der Chain übergeben. Aus der Liste der im Block enthaltenen Transaktionen wird der Index der gewünschten Chain ermittelt. Mit dem Index wird eine Teilliste der Transaktionen erstellt und eine Merkle Root für die Teilliste der Transaktionen ermittelt. Mit dieser Merkle Root und dem Schlüssel kann dann der Hashwert der Beschreibungsdatei der Chain ermittelt werden.

Somit stellt der Genesisblock den zentralen Einstiegspunkt dar und kann die Pfade der verschiedenen Datenbestände ermitteln und zurückgeben. Des Weiteren speichert der Genesisblock, welche Datenbestände in der Datenbank vorhanden sind.

6.2.2 Chains

Die Blöcke, welche die Chains bilden, beinhalten als Transaktionen die Events, welche von der In-Memory-Datenbank erstellt wurden, und persistieren diese für das erneute Ausführen und somit das Laden der Datenbank. Jede Chain besteht aus einer Beschreibungsdatei, die als Daten den Hashwert der Chain bzw. der Beschreibungsdatei, den Hashwert des derzeit letzten Blocks der Chain und den Namen des Datenbestandes enthält.
Der Hash des ersten Blocks der Chain wird berechnet aus der Kombination des Pfades der Chain und dem ChainHash. Die nachfolgenden Blöcke erhalten ihren Hashwert aus der Kombination der Merkle Root des Vorgängers und dessen eigenem Hash. Ein neuer Block wird immer nach 64 Transaktionen erstellt.

6.2.3 Proof of Work und Netzwerk

Auf die Implementationen eines Proof-of-Work-Mechanismus wurde im Rahmen des Prototyps verzichtet. Insgesamt sollte betrachtet werden, ob dieser für die Umsetzung einer GoBD-konformen Datenbank zwingend notwendig ist.
Proof of Work fügt der Blockchain zwar eine Sicherheitsebene hinzu, indem Änderungen an den Transaktionen extrem erschwert werden, aber gleichzeitig hat dieser den Nachteil, dass extreme Ressourcen für die Erstellung neuer Blöcke benötigt werden. Bei digitalen Währungen ist dieser Prozess ausschlaggebend für das Vertrauen der ordentlichen Abwicklung von Zahlungen.
In einer Datenbank, die auf einem Unternehmensserver betrieben wird, sollte darüber nachgedacht werden, andere Sicherheitsmechanismen, wie beispielsweise eine Verschlüsselung der Daten oder eine zusätzliche externe Absicherung ähnlich einer TSE in Kassensystemen, einzuführen.
Des Weiteren wurde bei der Implementation der Ansatz, die Blockchain als verteiltes System auf mehrere Knotenpunkte zu verteilen, nicht umgesetzt. Dies liegt darin begründet, dass Buchführungsdaten mit zu den schützenswertesten Informationen eines Unternehmens zählen. Für die Validierung von Blöcken müssten diese für die anderen Knotenpunkte offengelegt werden. Dies wäre nicht nur ein Verstoß gegen die Datenschutzgrundverordnung, sondern auch gegen das Steuergeheimnis.

6.3 Nutzung der Datenbank

Im Rahmen des Prototyps ist die endgültige Nutzungsart der Datenbank nicht umgesetzt. Es würde sich bei der beschriebenen Architektur so darstellen, dass der Nutzer nicht in Kontakt mit der Datenbank an sich kommen würde. Programme, welche auf die Daten-

bank aufbauen, würden mit der In-Memory-Datenbank kommunizieren und von dieser die Daten beziehen sowie alle normalen Transaktionen durchführen können.
Hierzu kann ein Interpreter für die gängigen Befehle eingebaut werden, welcher die Kommandos, die anschließend in der Blockchain persistiert werden, aus gängigen Datenbanksprachen wie SQL extrahiert. Die Befehle werden beim Laden einer Datenbank in den Arbeitsspeicher wieder in derselben Reihenfolge ausgeführt und dadurch die In-Memory-Datenbank wieder hergestellt.
Ein Vorteil dieser Umsetzung ist neben den schnellen Zugriffszeiten einer In-Memory-Datenbank, dass diese für den Nutzer jede Gestalt annehmen kann, ohne dass die GoBD-konforme Sicherung der Befehle verändert werden muss. Denkbar wären hier klassische Systeme wie relationelle Datenbanken oder NO-SQL-Datenbanken, aber auch andere Strukturen wie Objetkdatenbanken oder Data-Warehouses.
Nachteile bei der Arbeit könnten die eventuell langen Ladezeiten für den initialen Zugriff sein, welche insbesondere bei einmaligen Zugriffen problematisch sein könnten, aber auch die erhöhte Datenmenge, welche für die Speicherung verbraucht wird. Ein gelöschter Datensatz wird auf der Speicherebene nicht gelöscht, sondern ein zusätzlicher Datensatz, der beim Laden der Datenbank für die Löschung der Daten In-Memory sorgt, wird sogar noch zusätzlich gespeichert.
Wie eingangs beschrieben, wurde kein Sprachinterpreter für den Prototyp erstellt. Als Interface dienen Funktionen, welche die datenbankspezifischen Aufgaben sowie die CRUD-Operationen für die Daten bereitstellen. Als Datenbankoptionen wurden eine KV-Datenbank sowie eine Accounting-Datenbank, welche aus einem Journal und Konten besteht und für die Abbildung von Buchführungen gedacht ist, erstellt.

7 Herstellung der Unveränderbarkeit

In den nachfolgenden Abschnitten wird auf die Funktionsweise des Prototyps eingegangen und dargestellt, wie die Unveränderbarkeit der Daten umgesetzt wird. Hierfür wird zunächst im Abschnitt 7.1 die Struktur der Daten im Prototyp genauer erklärt. Anschließend wird in Abschnitt 7.2 die Kontrolle der Änderung der Daten aufgezeigt.

7.1 Datenstruktur

Die Datenstruktur, welche für die Sicherung der Daten gegen Veränderung verwendet wird, besteht aus einem Genesisblock, welcher die Berechnungsgrundlage für die Pfade der Chains bildet. Die Chains, welche die Blöcke beinhalten, bestehen aus Metainformationen wie beispielsweise dem Hashwert des letzten Blocks der Kette.

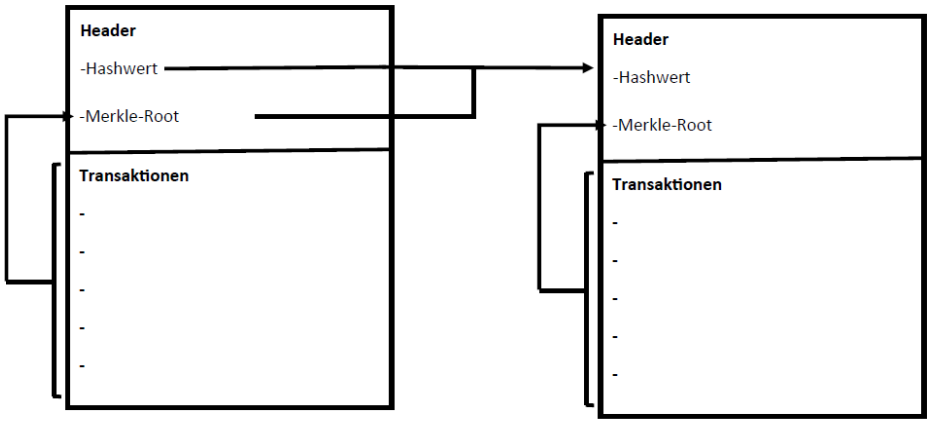


Abbildung 7.1: Darstellung der Verknüpfung zwischen den Blöcken

Die Blöcke selber sind nicht in einer Datenstruktur innerhalb der Kette miteinander verbunden, und keinem Bestandteil der Datenstruktur ist die gesamte Kette bekannt. Die Blöcke sind über die in ihnen enthaltenen Daten miteinander verbunden. In der Abbildung 7.1 ist dargestellt, dass die Adresse eines Blocks über die Merkle-Root des Vorgängerblocks und seinen eigenen Hashwerts berechnet wird. Der Hashwert ist gleich der

Adresse des Blocks. Hierdurch entsteht eine Kette, welche durch die Daten zusammengehalten wird.
Durch die Struktur entstehen, anders als bei gängigen Blockchain-Implementationen, keine zusammenhängende Kette, sondern viele Ketten, welche denselben Block als Ursprung haben. Der Vorteil hierbei ist, dass pro Datenbank eine Kette entsteht und nicht gefiltert werden muss, zu welcher Datenbank die einzelnen Daten gehören.

7.2 Prüfung der Datenintegrität

Die Prüfung der Integrität der Daten ist für die Umsetzung der Unveränderbarkeit essenziell. Hierbei hilft die zuvor erläuterte Datenstruktur. Für die Prüfung wurde sowohl eine Prüfung vom Genesisblock zum letzten Block als auch vom letzten Block bis zur Chain implementiert. Beide Prüfungsformen werden nachfolgend vorgestellt.

Vorwärtsprüfung

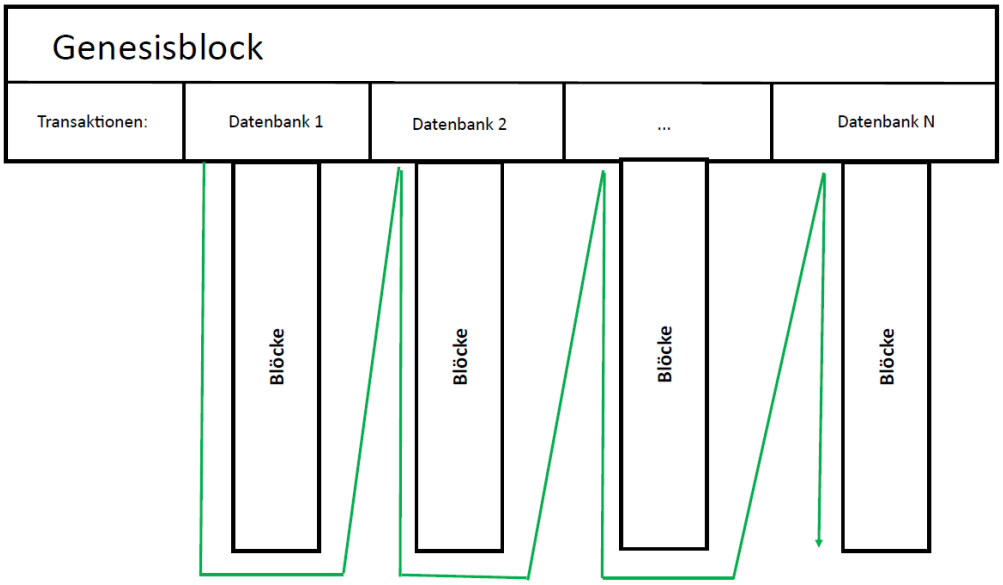


Abbildung 7.2: Darstellung der Vorwärtsprüfung

Bei der Vorwärtsprüfung werden die Ketten vom Genesisblock bis zum jeweils letzten Block durchlaufen. Hierbei wird jeweils der Block geladen und der Hashwert des nachfolgenden Blocks erneut berechnet. Über diesen Hashwert wird dieser als nächster Block geladen. Wenn der Hashwert erreicht ist, der in der Chain als letzter Block gespeichert ist, dann gilt die Prüfung als bestanden.
Wurden Daten geändert, ergibt sich ein Hashwert, unter dessen Namen keine Datei existiert. Denn bereits eine kleine Änderung in der Eingabe führt zu einer großen Verände-

rung des Hashwerts. Kann ein Block bei der Prüfung nicht geladen werden, dann wurden die Daten verändert.
Die Abbildung 7.2 zeigt den Verlauf der Prüfung. Der Genesisblock ist der Ursprung aller Ketten, welche hier als senkrecht ausgerichtete Rechtecke dargestellt sind. Die Transaktionen zur Erstellung der Datenbanken und der Hashwert des Genesisblocks bilden die Berechnungsgrundlage für die Hashwerte der einzelnen Ketten. Die grüne Linie in der Abbildung zeigt den Verlauf der Prüfung über die Datenstruktur.

Rückwärtsprüfung

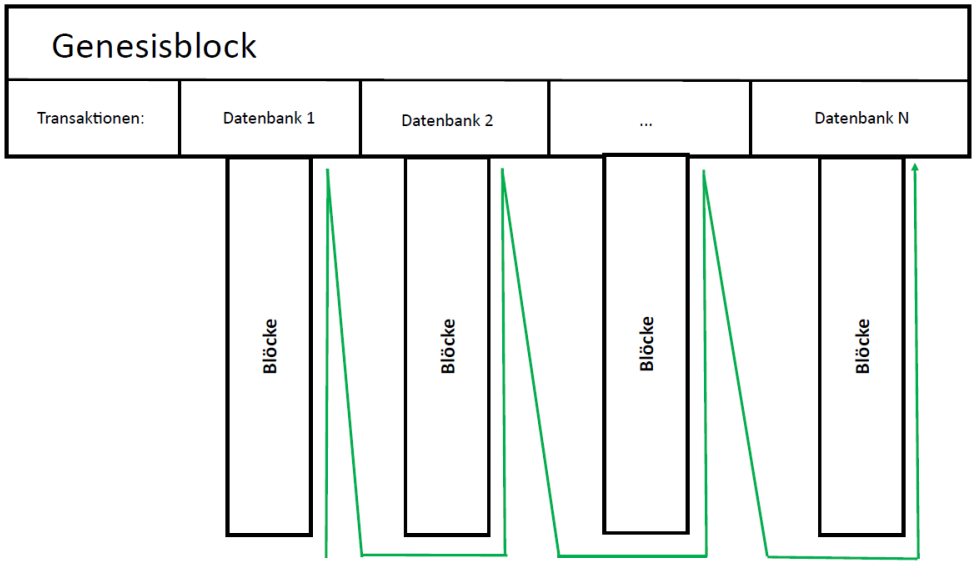


Abbildung 7.3: Darstellung der Rückwärtsprüfung

Im Unterschied zur Vorwärtsprüfung wird bei der Rückwärtsprüfung die Kette vom höchsten Block zurück zum Genesisblock durchlaufen. Hierbei wird geprüft, ob die gespeicherte Merkle-Root mit den aktuellen Transaktionen übereinstimmt und ob der Verweis auf den Vorgängerblock korrekt ist.
Die Prüfung ist in der Abbildung 7.2 dargestellt. Hierbei wird der Verlauf der Prüfung wie bereits bei der Vorwärtsprüfung durch den grünen Pfeil dargestellt.
Wenn eine der beiden Prüfungen fehlschlägt, wird im Prototypen eine Ausnahme ausgelöst. Durch die Ausnahme kann dem Nutzer mitgeteilt werden, dass ein Eingriff in die Daten erfolgt ist und die Daten nicht mehr nutzbar sind.

Unveränderbarkeit

Eine Veränderung der Daten ist im Prototyp nicht ausgeschlossen. Die Unveränderbarkeit der Daten im Sinne der GoBD ist dadurch sichergestellt, dass alle Änderungen erkannt werden können. Die Daten können im Prototyp verändert werden. Dadurch, dass

die Änderung erkannt werden kann, können nach einem Eingriff Maßnahmen ergriffen werden, um die Integrität der Daten wieder herzustellen. Dies kann durch das Einspielen des Backups geschehen.

8 Evaluation

In den nachfolgenden Abschnitten wird untersucht, ob die Anforderungen, die in Kapitel 3 erarbeitet wurden, erfüllt wurden. Hierzu wird zunächst in Abschnitt 8.1 untersucht, ob der Prototyp die Ziele erreicht hat, die für die Entwicklung angedacht waren. Anschließend wird in Abschnitt 8.2 erklärt, ob und wie die Anforderungen an Datenbanken und insbesondere die Anforderungen der GoBD an die Datenhaltung mit dem Lösungsansatz eingehalten werden.

8.1 Evaluation des Prototyps

Das Augenmerk bei der Entwicklung des Prototyps lag auf der Erfüllung der Unveränderbarkeit im Sinne der GoBD und der Erstellung einer Architektur, die sich an verschiedene Einsatzszenarios anpassen lässt.
Als nachgeordnete Anforderungen an den Prototyp wurden die Benutzerfreundlichkeit, Performance und Themen wie Benutzerverwaltung, Schutz der Dateien und Abdeckung der Edge-Cases behandelt. Nachfolgend wird die Erfüllung der Ziele erläutert:

Unveränderbarkeit

Für die Unveränderbarkeit im Sinne der GoBD wird im Prototyp auf Teile des Blockchain-Technologiestacks zurückgegriffen. So entstehen bei der Nutzung der Datenbank viele verschiedene Dateien, welche eine Bezeichnung in Form eines hexadezimalen Hashwertes erhalten. Diese Dateien bilden die Persistierung für die Bausteine der Datenbank.
In der untersten Ebene in Form der Blöcke werden die Transaktionen, welche auf der Datenbank ausgeführt werden, gespeichert. Soll die Datenbank neu geladen werden, werden die Transaktionen aus den Blöcken geladen und erneut ausgeführt. Durch die Speicherung der Transaktionen anstatt der eigentlichen Daten kann der Zustand eines Datums zu jedem Zeitpunkt rekonstruiert werden.
Im Sinne der Vorgabe bleibt der Zugriff auf die Ursprungsbuchung immer bestehen, und die Änderungen, die zu dem aktuellen Zustand geführt haben, sind getrennt von der Ursprungsbuchung aufgezeichnet. Durch diese Implementierung hat der Nutzer keine Möglichkeit, die Daten über einen vom Programm zur Verfügung gestellten Weg zu entfernen.

Wenn ein Datum entfernt wurde, wird der Zugriff auf die Datenbank gesperrt, dies wird über die Nutzung der Blockchain-Elemente erreicht.

Die Vorgabe wurde durch den Prototyp im weitesten Sinne erfüllt. Änderungen und Ursprünge werden dokumentiert und Änderungen werden durch die Blockchainbasis erkannt. Aber, durch die fehlende Verschlüsselung der Dateien und den Verzicht auf einen an den Proof-of-Work-Mechanismus angelehnten Prozess könnten die Blockchainstruktur verändert und danach die Header neu berechnet und fortgeschrieben werden. Da es sich um einen Prototyp handelt, wird der Punkt dennoch als erfüllt angesehen, denn durch Verschlüsselung der zur Speicherung der Blockchain genutzten Dateien wird der erfolgreiche Zugriff zwecks Veränderung in diese sehr unwahrscheinlich.

Anpassbarkeit an verschiedene Einsatzszenarios

Wenn man betrachtet, wie viele verschiedene Softwareanwendungen existieren, ist die Anpassung an verschiedene Einsatzszenarios für eine Datenbank ein besonders interessanter Punkt. Dies gilt insbesondere beim Einsatz in Programmen, welche viele verschiedene Aufgabenbereiche in einem Unternehmen abdecken sollen.

Hierzu wurde im Prototyp eine zusätzliche Schicht überhalb der Umsetzung der Blockchain-Sicherung verwendet. In dieser Schicht können verschiedene Datenbankarten ausgeführt werden, solange die Sicherung der Befehle eingehalten wird. Die Form der Datenbank ist hierbei egal.

Hierdurch kann beispielsweise eine Mandanten-/Kundenliste als Key-Value-Datenbank ausgeführt werden, während eine SQL-Datenbank die Buchführungsdaten beinhaltet. Die Umsetzung der Anpassbarkeit im Prototyp wird als gelungen bewertet, denn dieser beinhaltet bereits zwei verschiedene Datenbanktypen.

8.2 Evaluation des Lösungsansatzes

In den nachfolgenden Abschnitten wird die Möglichkeit der Umsetzung der ermittelten Anforderungen, nach dem jeweiligen Anforderungsgebiet, mittels des vorgestellten Lösungsansatzes untersucht. Hierzu wird in Abschnitt 8.2.1 die Umsetzung der Vorgaben durch die GoBD erläutert. Nachfolgend in Abschnitt 8.2.2 wird auf die Anforderungen an Datenbanken, sowohl im Allgemeinen als auch für die Datensicherheit, eingegangen. Abschließend wird in Abschnitt 8.2.3 auf die Anforderungen durch den Einsatz in der Steuerberatungspraxis geschaut.

8.2.1 Vorgaben der GoBD

Nachfolgend werden die Anforderungen der GoBD betrachtet, und es wird evaluiert, ob diese mit der vorgestellten Architektur gelöst werden können:

Ordnung

Die Umsetzung des Kriteriums der Ordnung hängt von der In-Memory-Datenbank ab. Je nachdem, wie diese aufgebaut wurde, ist dieses Kriterium erfüllt oder nicht. Hierdurch fällt die Möglichkeit weg, eine Blockchainstruktur als reine Fassade vor einer Datenbank einzusetzen. Dies könnte zwar die Persistierung der Befehle, welche in die Datenbank eingeflossen sind, erfüllen, aber nicht den Aufbau der Datenbank an das Kriterium der Ordnung anpassen.

Zusammenfassend lässt sich sagen, dass die Lösung in der Lage ist, das Kriterium zu erfüllen. Die Möglichkeit, eine Fassade für andere Datenbanken einzusetzen, ist aber nur eingeschränkt rechtssicher. Hierzu muss das Kriterium der Ordnung in der eigentlichen Datenbank umgesetzt werden.

Unveränderbarkeit

Das Thema der Unveränderbarkeit lässt sich mit einer Blockchainstruktur umsetzen. Hierbei wird durch diese und die Speicherung der Befehle, welche einen Wert erstellt haben, die Möglichkeit geboten, jeden Zustand eines Wertes abzurufen. Der Punkt wird als vollkommen erfüllt eingestuft.

IKS

Das interne Kontrollsystem, welches in den GoBD vorgeschrieben ist, soll verschiedene Daten der Datenbank prüfen und dem Nutzer die Ergebnisse zur Verfügung stellen. Einige der zu dokumentierenden Punkte können auch außerhalb des Datenbanksystems dokumentiert werden. Die Umsetzung der anderen Prüfungen wird, wie bei herkömmlichen Datenbanken auch, zum großen Teil in der Anwendung selber erfolgen müssen.

Eine vollständige Umsetzung in der Datenbank ist nicht möglich, da viele zu erhebende Metriken vom Anwendungsprogramm abhängig sind und die Semantik nicht immer der Datenhaltung bekannt ist. Es müssten ansonsten für viele Abfragen eigene Anwendung in die Datenbank einprogrammiert werden. Allgemeine Daten, wie Protokollierung, wer die Daten geändert hat und wer in welchem Umfang Zugriff auf die Daten hat, sollten dagegen schon in der Datenbank stattfinden.

Diese Anforderung kann von der Lösung nur teilweise umgesetzt werden. Ein Teil der Funktionen kann sicherlich in der Datenbank ausgeführt werden und die Datenbank kann als Speicher für die Protokollierung dienen. Allumfassend kann aber keine Erfüllung dieser Vorschrift in der Datenbank erfolgen.

Datensicherheit

Der Punkt der Datensicherheit ist vollständig erfüllt. Die bei der Nutzung der Lösung entstehende Blockchain kann durch ein Backupsystem gesichert und jederzeit wieder in das System eingespielt werden. Es könnte auch bei einem Serverumzug in einen neuen Datenbankserver verwendet werden, solange dieselben Befehle implementiert sind. Wenn dies nicht der Fall ist, kann eventuell softwareseitig eine Umsetzung auf die neuen Befehle erfolgen.

Datenzugriff

Für den Datenzugriff sind von den Finanzbehörden Dateiformate und Zugriffswege vorgegeben. Diese müssen als Ausgabe implementiert werden. Die Umsetzung dieser Vorgaben kann in der Datenbank, wie diese vorgestellt wurde, durchgeführt werden. Hierzu müssten bei der Erstellung der Datenbank bestimmte Pflichtfelder vorgegeben werden, um den Anforderungen gerecht zu werden.
Durch die Vorgabe des Datenzugriffs ergibt sich allerdings eine Einschränkung bei der Verwendung unterschiedlicher Datenbankarten im System. Entweder muss für jede Datenbank vorgegeben werden, welche Pflichtfelder zu verwenden sind, was einen Eingriff der Datenbank in die Anwendungslogik bedeuten würde oder alternativ eine Überleitungstabelle, welche durch den Programmierer gefüllt werden muss.
Die Vorgabe kann insofern als erfüllt angesehen werden, da die Bereitstellung der Daten mit wenigen Vorgaben und eventuell einigen Anpassungen möglich ist. Es wird aber nicht ohne zusätzlichen Aufwand für den Entwickler möglich.

Zusammenfassung

Die Lösung in Form einer In-Memory-Datenbank, deren ausgeführten Befehle mit Event-Sourcing mit einer Blockchain persistiert wird, eignet sich für eine Umsetzung der Unveränderbarkeit im Rahmen der GoBD. Des Weiteren kann auch eine Datensicherung gut umgesetzt werden. Aber alle anderen zu erfüllenden Anforderungen werden nicht vollständig umgesetzt und bedürfen weiter der Aufmerksamkeit des Anwendungsentwicklers.

8.2.2 Eigenschaften von Datenbanken

Die Anforderungen und Eigenschaften von Datenbanken werden in der Lösung durch die In-Memory-Datenbank umgesetzt. Die Blockchain als reines Transaktionslog ist hierbei nur ein Erfüllungsgehilfe.
Innerhalb der allgemeinen Anforderungen an Datenbanken befindet sich allerdings die Forderung, Daten mit einer möglichst geringen Redundanz abzuspeichern. Diese wird durch die Sicherung in der Blockchain und die Nutzung von Event-Sourcing erheblich erschwert, da hier nicht die Daten an sich, sondern alle Befehle gespeichert werden. Bei

Änderung von Daten werden sowohl die alten Daten als auch die neuen gespeichert. Wird ein Datum gelöscht, wird dieses über einen Löschbefehl ausgeführt und alle anderen Befehle werden ebenfalls zusätzlich weiterhin gespeichert.
Es kann also gesagt werden, dass die Verwendung der Blockchain für die Persistierung einen der Grundsätze von Datenbanken durchbricht.
Bei der Umsetzung der Datensicherheitsanforderungen muss auch wieder zum großen Teil auf die In-Memory-Datenbank verwiesen werden, welche die Punkte ACID-Konformität, Benutzerverwaltung, Logging übernehmen sollte. Die Backupfähigkeit kann durch Sicherung der entstandenen Blockchain gewährleistet werden.
Interessanter sind bei der Betrachtung der Anforderungen an die Datensicherheit die Punkte der Replizierbarkeit und der High-Availability. Für die Replizierbarkeit muss zwischen den betriebenen Servern ein Konsens gefunden werden, in welcher Reihenfolge die Transaktionen zu verarbeiten sind. Transaktionen müssen immer an alle an der Datenbank beteiligten Server bekanntgegeben werden. Hierbei kann sich aber an Techniken, die für Kryptowährungen genutzt werden, bedient werden.
Dies ist deshalb möglich, weil das Problem, eine Blockchain auf mehreren Computern zu synchronisieren, in den Netzwerken der Kryptowährungen bereits gelöst wurde. Die Herstellung einer High-Availability kann mit denselben Mitteln, die auch bei anderen Datenbanksystemen Verwendung finden, erreicht werden.
Zusammenfassend lässt sich sagen, dass alle Anforderungen, welche an Datenbanken gestellt werden, durch den Lösungsansatz umsetzbar sind. Einzig die Anforderung nach möglichst geringer Redundanz durch die Sicherung aller Befehle ist nicht umsetzbar.

8.2.3 Anforderungen durch die Steuerberatungstätigkeit

In den nachfolgenden Absätzen werden die Anforderung, die sich aus der Steuerberatungspraxis an die Datenhaltung stellen, untersucht, beginnend mit der Anforderung, dass die Lösung GoBD-konform sein soll.
Dies ist natürlich die Kernanforderung für den Einsatz in diesem Berufsstand. Allerdings, wie im Abschnitt 8.2.1 erläutert, wird dies mit der Lösung nicht vollständig erreicht. Hieraus ergibt sich, dass die Datenbank allein die Anforderung nicht erfüllen wird.
Die Mehrmandatsfähigkeit dagegen ist gegeben, da sowohl je nach Umsetzung die In-Memory-Datenbank dies lösen kann oder einfach mehrere Blockchains gespeichert werden könnten, je eine pro Mandat. Dies hätte darüber hinaus den Vorteil, dass nicht alle Daten immer komplett geladen werden müssen, wenn man nur an einem Mandanten arbeiten möchte.
Der Punkt der Sicherheit wird sowohl durch die Blockchain an sich, aber auch durch die Backupfähigkeit abgedeckt. Dieser Punkt wird als erfüllt bewertet. Ebenso wird die

Forderung als erfüllt bewertet, die Datenbank auf verschiedene Programme anpassen zu können, dies wird durch die In-Memory-Datenbank möglich.
Die Anforderung, Stammdatenhistorien nachvollziehen zu können, ist von der Form der verwendeten Datenbank abhängig. Wenn diese eine Funktion zur Speicherung dieser Daten zur Verfügung stellt, dann kann dies problemlos durchgeführt werden. Ist dies nicht erfüllt, dann könnte in der Presentation-Layer eine Funktion für die teilweise Ausführung der gespeicherten Befehle implementiert werden, bei der nach jedem Schritt der Zustand des Feldes extra gesichert wird und somit der Verlauf berechnet. Dies wäre allerdings extrem Aufwendig, je nachdem, wie viele Daten in der Datenbank enthalten sind. Der Punkt wird daher mit einer teilweisen Erfüllung bewertet.
Der letzte Punkt, die Möglichkeit, Daten mit KI-Anwendungen analysieren zu lassen, ist ebenfalls gegeben, da die KI ebenfalls ihre Daten aus der In-Memory-Datenbank ziehen kann und diese auswertet.
In summa können die Anforderungen als nicht erfüllt angesehen werden. Viele der Nebenanforderungen können zwar problemlos umgesetzt werden, allerdings wird die Hauptanforderung, in Form der GoBD, welche für die Tätigkeit das größte Gewicht hat, nur unzureichend umgesetzt.

9 Diskussion des Ergebnisses

Die Diskussion der in dieser Arbeit entstandenen Ergebnisse ist auf zwei Teile aufgeteilt. Zum einen wird in Abschnitt 9.1 der entstandene Prototyp besprochen, zum anderen in Abschnitt 9.2 das Gesamtergebnis betrachtet.

9.1 Prototyp-bezogene Diskussion

Im Rahmen dieser Arbeit wurde ein Prototyp erstellt, welcher den Nachweis erbringen sollte, dass durch den Einsatz von Teilen der Blockchain-Technologie die Unveränderbarkeit im Sinne der GoBD umsetzbar ist. Als weiteres Ziel sollte eine Datenstruktur geschaffen werden, die in der Lage ist, sich verschiedenen Einsatzszenarios anzupassen und für Entwickler unkompliziert verwendet werden kann.
Wie in Kapitel 8 untersucht, erfüllt der Prototyp die Forderung nach Unveränderbarkeit im weitesten Sinne. Die Anforderung, dass dieser in verschiedenen Szenarios sinnvoll eingesetzt werden kann, wurde als umgesetzt bewertet.
Die Anforderung der Unveränderbarkeit wurde im Prototyp nicht vollends umgesetzt. Die Umsetzung könnte aber durch kleinere Änderungen schnell wirksam werden, indem beispielsweise die Dateien verschlüsselt werden. Die Prüfung, ob die Dateien verändert wurden, funktioniert.
Die Unveränderbarkeit fordert insbesondere, dass die ursprüngliche Buchung nicht gelöscht werden kann und Änderungen, welche zu dem aktuellen Zustand geführt haben, aufgezeichnet werden. Dies geschieht im Prototyp, indem die Änderungen an der Datenbank durch Events dargestellt werden. Die Argumente, welche den Events übergeben werden, werden anschließend als Transaktionen in der Datenbank gespeichert.
Durch diese Art der Speicherung wird die Ursprungsbuchung immer als erste Transaktion in der Blockchain bestehen bleiben, während die Änderungen an der Buchung später folgen.
Dass die Umsetzung der Anpassbarkeit geglückt ist, könnte die Akzeptanz von Entwicklern für die Lösung steigern. Entwickler können weiter mit einer Datenbankart arbeiten, welche Ihnen vertraut ist, haben aber die Unveränderbarkeit ohne spezielle Vorkehrungen in der Datenbank umgesetzt.

Der Prototyp an sich setzt die geforderten Anforderungen grundlegend um. Um eine absolute Sicherheit für die Unveränderbarkeit der Eintragungen zu erhalten, sollte weiter geforscht werden, ob die Eingrenzung der gewählten Bestandteile der Blockchain für diese Sicherheit ausreicht. Für die Herstellung einer Marktreife müssen noch ein Interpreter für Datenbankbefehlssprachen und eine Abdeckung von Edge-Cases in der Fehlerbehandlung erfolgen. Ein weiterer wichtiger Punkt wäre die Umsetzung von Multi-Threading für die Verarbeitung von parallel eingehenden Befehlen.

9.2 Gesamtergebnis

Die vorliegende Arbeit hatte das Ziel zu untersuchen, ob es möglich ist, eine Datenbank zu erstellen, welche GoBD-Konformität für Anwendungssoftware herstellen kann. Da ein Teil der Anforderungen organisatorischer Natur ist, sollten zumindest die technischen Aspekte der GoBD durch die Datenbankarchitektur umgesetzt werden.
Für die Prüfung dieser These wurden die Grundlagen der GoBD und die Anforderungen, welche diese an die Buchführung stellen, erarbeitet. Des Weiteren wurden die Anforderungen an Datenbanken im Allgemeinen und die für den Einsatz im Steuerbüro auftretenden Anforderungen ermittelt.
Anschließend wurde das Problem noch einmal detaillierter beschrieben und ein Prototyp entwickelt, welcher die Architektur umsetzen sollte, die eine Lösung des Problems versprach. Der Fokus des Prototyps lag auf dem Punkt der Unveränderbarkeit der Buchungen, da diese als Hauptproblem festgelegt wurden.
Mit der Demonstration des Prototyps wurde die Bedienung vorgeführt und gezeigt, dass ein manueller Eingriff in diesen zu einer Sperrung der Datenbank führen und somit nachgewiesen werden kann.
Abschließend wurden anhand der Erfahrungen aus der Implementierung und der Recherche die Anforderungen, welche zuvor ermittelt wurden, ausgewertet. Die aus dieser Evaluation gewonnen Erkenntnisse sollen in den folgenden Absätzen betrachtet werden.
Das Hauptergebnis der Arbeit ist, dass es nicht möglich ist, alle technischen Aspekte der GoBD in einer Datenbank umzusetzen. Dies führt auch dazu, dass Anwendungen durch die Verwendung einer Datenbank nicht für den Einsatz im Steuerbüro gerüstet werden können. Allerdings können in der geplanten Architektur fast alle Anforderungen, welche sich an Datenbanken stellen, erfüllt werden, außer die Forderungen nach geringer Datenredundanz.
Die nicht vollständige Umsetzbarkeit der GoBD auf Ebene der Datenbank führt dazu, dass Entwickler von Anwendungssoftware sich weiterhin mit den GoBD beschäftigen und individuelle Lösungen für die Umsetzung finden müssen. Die entwickelte Lösung

könnte hierbei allerdings als ein Baustein für die Umsetzung der Unveränderbarkeit zum Einsatz kommen.

Hierbei sollte aber in Betracht gezogen werden, dass die Lösung zusätzlichen Speicherplatz bei der Verwendung als Vorschaltung zu einer Datenbank verbraucht. Ebenso muss überlegt werden, gegen welche Personen der Schutz auf der Datenbank aufgebaut werden muss. Dies sind zum großen Teil unternehmensinterne Kräfte, welche einen Fehler in der Buchhaltung vertuschen wollen, oder auch Unternehmen, welche den Gewinn durch Eingriffe in die Daten senken wollen. Hierbei sollte eine Sicherung über ein den Nutzern unbekanntes Administrationspasswort und eine Umsetzung der Transaktionen in einer Form, in der keine Eintragungen gelöscht werden können, ausreichen.

Für Eindringlinge von außerhalb des Unternehmens sollten von vornherein Maßnahmen im Rahmen der IT-Strategie ergriffen und nicht erst der Schutz auf Ebene der Daten hergestellt werden. Eine Verschlüsselung der Datenbank kann aber auch externe Angriffe aufhalten oder zumindest verzögern.

Die Sicherstellung der Anforderungen an Datenbanken kann durch die Verwendung einer bereits bestehenden In-Memory-Datenbankanwendung oder durch die Nutzung des Systems als Fassade vor einer Datenbank erreicht werden. Hierdurch kann eine Menge zusätzlicher Aufwand bei der Erstellung eines Datenbanksystems gespart werden. Gleichzeitig wird es möglich, verschiedene Datenbanken zu nutzen und somit die richtige Datenbank für jeden Anwendungstyp zu erstellen. Damit kann sowohl vermieden werden, eine viel zu umfangreiche Datenbank für eine kleinere Anwendung zu verwenden als auch, dass eine komplexe Anwendung nicht genügend Funktionen in der Datenbank findet. Trotzdem können beide Anwendungen dasselbe Interface nutzen.

Das letzte Ergebnis, dass die Verwendung im Steuerbüro nicht sinnvoll ist, ergibt sich bereits dadurch, dass die Datenbank nicht alle GoBD-Anforderungen umsetzt.

Die zu Beginn der Arbeit aufgestellte These, dass eine GoBD-konforme Datenbank mit Hilfe von Bestandteilen des Technologie-Stacks hergestellt werden kann, konnte nicht bestätigt werden. Nicht einmal die rein technischen Aspekte werden vollständig von der Lösung abgedeckt. Zum einen liegt dies darin, dass ein Teil der GoBD organisatorischer Natur sind. Aber auch die technischen Aspekte sind zum großen Teil anwendungsbezogen, insbesondere die Punkte IKS und Ordnung.

Die Datenbank ist im Sinne der GoBD als Erfüllungsgehilfe zu betrachten, aber nicht als Ort für die Umsetzung aller Vorschriften bestimmt.

Die Fragestellung, inwiefern der komplette Technologie-Stack der Blockchain für die Implementierung umgesetzt werden muss, lässt sich insofern beantworten, dass für eine allgemeine Umsetzung der Datenstruktur, wie im Prototyp geschehen, nicht der volle Stack

benötigt wird. Ob dies auch dieselbe Sicherheit wie eine volle Implementierung bietet, ist nicht Gegenstand der Betrachtung.
Die Frage der Performance-Auswirkungen wurde ebenfalls nicht endgültig untersucht. Insgesamt sollte sich bei der Verwendung keine merkliche Veränderung ergeben. Insbesondere Lesebefehle sind nicht von der Änderung betroffen, sondern werden direkt auf der Datenbank ausgeführt. Veränderung sollte es nur bei Befehlen geben, welche Daten ändern und beim Laden einer Datenbank aus der Blockchain in den Arbeitsspeicher.
Die Daten für die Verarbeitung mit KI werden über die Datenbank dargestellt. Die Blockchain hat hierauf keine Auswirkungen. Allerdings wäre es möglich, über eine Teilverarbeitung der Befehle in der Blockchain auch die Entwicklung der Daten in der Datenbank darzustellen und auszuwerten.
Ein Eingriff in die Daten kann über das Ablaufen der Blockchain und der erneuten Berechnung der Hashwerte nachgewiesen werden. Wenn die errechneten Daten nicht mit den gespeicherten Meta-Daten zusammenpassen oder die Kette nicht mehr abgelaufen werden kann, da ein Wert ins Leere zeigt, wurde in die Daten eingegriffen.
Die Wiederherstellung der Datenbank sollte über die Datensicherung abgebildet werden. Diese durchzuführen ist nach den GoBD vorgeschrieben, und die Datenbank kann aus diesen Daten wieder auf den Zustand vor dem Eingriff zurückgesetzt werden.
Für die Bestätigung der Erfüllung des Kriteriums der Unveränderbarkeit müsste erforscht werden, ob die Blockchain im verminderten Umfang dieselbe Sicherheit bietet wie eine vollständige Blockchain oder zumindest ausreichenden Schutz für den Einsatzzweck.
Abschließend lässt sich sagen, dass zwar keine Lösung gefunden wurde, die Entwicklern die Beschäftigung mit den GoBD erspart, dennoch wurde hoffentlich das Bewusstsein für die Vorschrift an sich erhöht, damit die Grundlagen häufiger beachtet und somit Unternehmen vor Nachzahlungen im Rahmen von Prüfungen bewahrt werden.

10 Fazit

Die Buchführung bildet einen wichtigen Teil eines Unternehmens. Die Sicherung im Rahmen der rechtlichen Vorgaben kann mitunter hohe Steuernachzahlungen vermeiden. Gleichzeitig bildet sie den Ausgangspunkt für viele Entscheidungen für die Zukunft des Unternehmens.

Die Vorgaben, die eine Buchführung für die Verwendung als Grundlage der Besteuerung mit sich bringen muss, wurden zu Beginn dieser Arbeit erläutert. Anschließend wurde der Versuch gewagt, einen Prototyp zu entwickeln, der das Kriterium der Unveränderbarkeit auf Basis der Blockchain-Technologien umsetzen sollte.

Die Entwicklung des Prototyps ist als geglückt zu bewerten. Es konnte eine Datenstruktur entwickelt werden, welche die Voraussetzungen der Unveränderbarkeit umsetzt und gleichzeitig Änderungen an den Daten erkennen kann.

Nach der Vorstellung des Prototyps wurden die aufgestellten Anforderungen an die Lösung anhand der erhaltenden Erkenntnisse evaluiert. Hier wurden die Anforderungen der GoBD auf technischer Ebene betrachtet, die Anforderungen, die für Datenbanken gelten, und die Anforderungen, welche durch den Einsatz im Steuerbüro entstehen.

Das wichtigste Ergebnis ist, dass eine vollkommene Umsetzung der GoBD in der Datenbank nicht möglich ist. Dies liegt zum einen daran, dass viele Aspekte nicht technischer Natur sind. Aber auch die technischen Aspekte lassen sich nicht vollkommen auf die Datenbank beschränken. Hier spielen immer auch Aspekte in der Anwendung an sich eine Rolle.

Ein weiteres Ergebnis der Arbeit ist, dass sich eine vollständige Datenbank mit der gewählten Architektur umsetzen lässt. Mit Blick auf andere Lösungen und den Datenbedarf einer solchen Lösung sollte aber ein Einsatz gründlich abgewogen werden.

Die der Arbeit zugrunde liegende These „Mit den Grundlagen der Blockchain-Technologien kann eine GoBD-konforme Datenbank erstellt werden“ wurde anhand der Literaturrecherche und der Erfahrungen aus der Implementierung des Prototyps beantwortet. So musste festgestellt werden, dass die technischen Aspekte der GoBD nicht vollständig in der Datenbank gelöst werden können. Eine vollkommene Entlastung von Entwicklern für Unternehmenssoftware war nicht möglich.

Bei der Recherche und Bearbeitung der Arbeit ist aufgefallen, dass zu dem Themenbereich der GoBD die Literaturauswahl sehr eingeschränkt ist. Insbesondere aus Sicht der Informatik wird dieser Themenbereich selten behandelt. Die meiste Literatur findet sich im Bereich des Steuerrechts.
Aufgrund der mitunter schwerwiegenden Folgen der Nichteinhaltung dieser Regelungen sollte die Aufmerksamkeit auch im Fachbereich Informatik erhöht werden. Allein die Anzahl der Anwendungen, welche durch diese Regelungen betroffen sind, sollte den Anlass hierzu bieten.
Eine Fortsetzung der Forschung für die Entwicklung eines Leitfadens für die Umsetzung der GoBD in Software könnte zum einen interessant sein und zum anderen für mehr Sicherheit in der steuerlichen Veranlagung in der Unternehmenswelt führen.

Literatur

Antonopoulos, A. M. (2018). *Bitcoin und Blockchain - Grundlagen und Programmierung: Die Blockchain verstehen, Anwendungen entwickeln* (P. Klicman, Hrsg.; 2. Auflage) [Description based on publisher supplied metadata and other sources.]. O'Reilly.

Bundesministerium für Finanzen. (2019). Grundsätze zur ordnungsgemäßen Führung und Aufbewahrung von Büchern, Aufzeichnungen und Unterlagen in elektronischer Form sowie zum Datenzugriff (GoBD). *Bundessteuerblatt 2019*. https://www.finanzamt.bayern.de/Informationen/Steuerinfos/Weitere_Themen/Aussenpruefung/2019-11-28-GoBD-1.pdf

Bundesministerium für Finanzen. (2023). Ergebnisse der steuerlichen Betriebsprüfungen der Länder 2022. *Monatsbericht des BMF*, 5. https://www.bundesfinanzministerium.de/Monatsberichte/2023/10/Inhalte/Kapitel-3-Analysen/3-2-steuerliche-betriebspruefung-2022-pdf.pdf?__blob=publicationFile&v=4#:~:text=Auf%20der%20Grundlage%20von%20Meldungen,der%20steuerlichen%20Betriebspr%C3%BCfung%20der%20L%C3%A4nder.&text=In%20den%20Betriebspr%C3%BCfungen%20der%20L%C3%A4nder,Euro%20festgestellt.

Bundessteuerberaterkammer. (o. d.). Der Steuerberater. https://www.bstbk.de/de/berufsbild-steuerberater/der-steuerberater

Datev eG. (2024). Fragen und Antworten zur Microsoft SQL-Datenhaltung. *Datev Hilfe-Center*. https://apps.datev.de/help-center/documents/1033860

Der Bundesbeauftragte für den Datenschutz und die Informationsfreiheit. (o. d.). Was bedeutet GoBD. https://www.bfdi.bund.de/DE/Buerger/Inhalte/Finanzen-Steuern/ABC_GoBD.html

Fill, H.-G., & Meier, A. (Hrsg.). (2020). *Blockchain: Grundlagen, Anwendungsszenarien und Nutzungspotenziale*. Springer Fachmedien.

Fowler, M. (2013). *Patterns of enterprise application architecture* (Nineteenth printing). Addison-Wesley.

Geirhos, M. (2021). *Entwurfsmuster: Das umfassende Handbuch* (1. Auflage 2015, 4., korrigierter Nachdruck) [Älle Beispielprojekte aus dem Buch zum Download - Für alle OO-Sprachen geeignet Bucheinband]. Rheinwerk Computing.

IMF. (2024). World Economic Outlook Database, April 2024. https://de.statista.com/statistik/daten/studie/166229/umfrage/ranking-der-20-laender-mit-dem-groessten-anteil-am-weltweiten-bruttoinlandsprodukt/

Kling, P. (2020). *GoBD: Anforderungen, praktische Umsetzungstipps und Beispiele*. IDW Verlag GmbH.

Kofler, M. (2024). *Datenbanksysteme: Das umfassende Lehrbuch* (2., durchgesehene Auflage) [Includes index]. Rheinwerk Computing.

Nakamoto, S. (2009). Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf

Rupp, C. (2021). *Requirements-Engineering und -Management: Das Handbuch für Anforderungen in jeder Situation* (7., aktualisierte und erweiterte Auflage) [Literaturverzeichnis: 563 - 572]. Hanser.

Schicker, E. (2017). *Datenbanken und SQL: Eine praxisorientierte Einführung mit Anwendungen in Oracle, SQL Server und MySQL* (5. Aufl. 2017). Springer Vieweg.

Statista Consumer Insights. (2023). Ranking der wichtigsten Kryptowährungen in Deutschland nach Markenbekanntheit im Jahr 2023. https://de.statista.com/statistik/daten/studie/1362614/umfrage/bekannteste-kryptowaehrungen-in-deutschland/

Teutemacher, T. (2020). *Praxishandbuch GoBD: Handbuch zur praktischen Umsetzung der Grundsätze zur ordnungsmäßigen Führung und Aufbewahrung von Büchern, Aufzeichnungen und Unterlagen in elektronischer Form sowie zum Datenzugriff (GoBD)* (1. Auflage) [Literaturangaben. - Zusätzliches Online-Angebot unter www.nwb.de]. NWB.

Vernon, V. (2015). *Implementing domain-driven design* (Fourth printing). Addison-Wesley.

Weber, I. (2022). Design Science Research als wissenschaftliche Herangehensweise für Abschlussarbeiten mit Gestaltungsauftrag in anwendungsorientierten Studiengängen. In *Angewandte Forschung in der Wirtschaftsinformatik 2022* (S. 101–116). GITO Verlag. https://doi.org/10.30844/akwi_2022_07

A Anhang

A.1 Codebeschreibung

Nachfolgend wird die Funktionalität des Prototyps anhand des Programmcodes beschrieben. Hierfür wird für jede Komponente der Programmcode abgebildet und anschließend der Ablauf im Programm erläutert. Im Abschnitt A.1.1 wird mit der Beschreibung der Data-Security-Layer begonnen. Anschließend wird in Abschnitt A.1.2 auf die Data-Presentation-Layer eingegangen.

A.1.1 Data-Security-Layer

Die Data-Security-Layer verwaltet die Daten und sichert diese. Nachfolgend werden die einzelnen Komponenten der Schicht erläutert und der Funktionsablauf dargestellt. In Abbildung A.1.1 ist die Dateistruktur der Software abgebildet. Hierbei ist zu beachten, dass für jede Klasse eine einzelne Datei angelegt wurde.

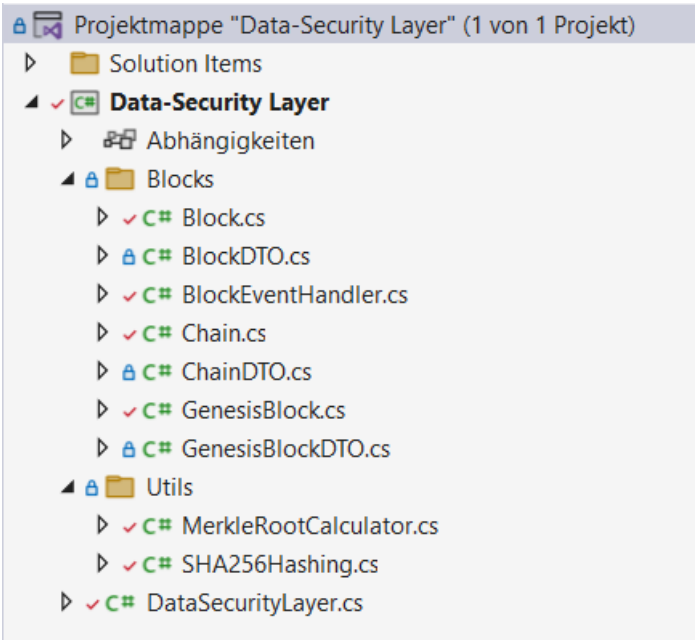


Abbildung A.1: Dateistruktur der Data-Security-Layer

Den Einstieg in die Schicht stellt die Klasse DataSecurityLayer dar, welche als erstes vorgestellt wird. Im Ordner Blocks sind die Klassen enthalten welche die Implementierung der Blockchain umsetzen. Der Ordner Utils enthält Klassen, welche von mehreren Klassen verwendet werden und die eine mehrfache Implementierung bestimmter Funktionen verhindern sollen.

DataSecurityLayer.cs

Listing A.1: Code der Klasse DataSecurityLayer

```
using Data_Security_Layer.Blocks;
using Data_Security_Layer.Utils;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Text.Json;
using System.Diagnostics;

namespace Data_Security_Layer
{
    public class DataSecurityLayer
    {
        //Path of the Database
        internal static string? _BasePath;
        private GenesisBlock _GenesisBlock;

        //instance used to build a Singleton
        internal static DataSecurityLayer instance;

        internal GenesisBlock GenesisBlock
        {
            get
            {
                return _GenesisBlock;
            }
        }

        // internal Constructor for use in the singleton-pattern
        private DataSecurityLayer(string path, string name, string
            secretKey)
        {
            _BasePath = path;
```

```
35              //Calcution of the Path with the name of the Databasesystem and
                    the secret key
36              string hash = SHA256Hashing.getHash(name+secretKey);
37              string GenesisPath = Path.Combine(path, hash);
38              //Checking if the Databasesystem is already existing and
                   loading/creating the Genesisblock
39              if (File.Exists(Path.ChangeExtension(GenesisPath, "bcdb")) ==
                   false)
40              {
41                  Directory.CreateDirectory(GenesisPath);
42                  _GenesisBlock = new GenesisBlock(hash);
43              }
44              else
45              {
46                  _GenesisBlock = GenesisBlock.loadGenesisBlockFromFile(
                       GenesisPath);
47              }
48          }
49
50          //Public function for getting the instance of the Databasesystem,
                part of the singleton-Pattern
51          public static DataSecurityLayer getDatabaseSystem(string path,
                string name, string secretKey)
52          {
53              if (instance == null)
54              {
55                  instance = new DataSecurityLayer(path, name, secretKey);
56              }
57              //ConsistencyTest when loading the Databasesystem
58              if(!(instance.ConsistencyTest(ConsistencyTestmode.Forwards) &
                   instance.ConsistencyTest(ConsistencyTestmode.Backwards)))
59              {
60                  throw new Exception("Es sind Inkonsistenzen in der
                       Datenbank aufgetreten");
61              }
62              return instance;
63          }
64
65          //Function for getting the Path up to the Chains, used as an
                Shortcut for Calculating the Blockpaths
66          internal string GetChainsPath()
67          {
68              return Path.Combine(_BasePath??"", _GenesisBlock.Hash);
69          }
70
```

```
71          //Function for adding a new Database to the Databasesystem
72          public void newDatabase(string Name, string SecretKey)
73          {
74              try
75              {
76                  this._GenesisBlock.addChain(Name, SecretKey);
77              }
78              catch (Exception e)
79              {
80                  Debug.WriteLine(e.Message);
81              }
82          }
83
84          //Function for getting the whole Data out of a Database
85          public List<string> getData(string DatabaseName, string SecretKey)
86          {
87              Chain chain = GenesisBlock.getDataBase(DatabaseName, SecretKey)
                    ;
88              return chain.getData();
89          }
90
91          //Function for adding Data to a Database
92          public void AddData(string DataBaseName, string SecretKey, string
                Data)
93          {
94              Chain chain = GenesisBlock.getDataBase(DataBaseName, SecretKey)
                    ;
95              chain.newData(Data);
96          }
97
98          //Testmodes for the ConsistencyTest
99          public enum ConsistencyTestmode
100         {
101             Backwards = 0, //Testing from the Chain to the last Block
102             Forwards = 1 //Testing from the Last Block back to the Chain
103         }
104
105         //Function for Testing the Consistency in the Blockchain
                Datastructure
106         public bool ConsistencyTest(ConsistencyTestmode testmode)
107         {
108             //result as bool True = All Databases are consistent
109             bool result = true;
110             List<string> ChainHashes = _GenesisBlock.getDataBases();
111             if (ChainHashes.Count == 0)
```

```
112                 {
113                     return true;
114                 }
115                 foreach (string ChainHash in ChainHashes)
116                 {
117                     Chain testChain = Chain.loadChainFromFile(ChainHash);
118                     switch (testmode)
119                     {
120                         case ConsistencyTestmode.Backwards:
121                             result = testChain.BackwardsConsistencyTest();
122                             break;
123                         case ConsistencyTestmode.Forwards:
124                             result = testChain.ForwardsConsistencyTest();
125                             break;
126                     }
127                     if (result == false)
128                     {
129                         return result;
130                     }
131                 }
132                 return result;
133             }
134         }
135 }
```

Im Code A.1 ist die Einstiegsklasse der Data-Security-Layer abgedruckt. Die Klasse ist als Singleton implementiert und kann daher nur einmal erzeugt werden. Hierdurch soll sichergestellt werden, dass die Schicht nur einmal erstellt wird und nicht mehrere Versionen in unterschiedlichen Zuständen vorliegen.

Über die Funktion getDataBaseSystem (Zeile 54) kann von einer über der Data-Security-Layer gelegenen Schicht eine Instanz der Data-Security-Layer erhalten werden. Wenn diese noch nicht erstellt wurde, dann wird dies nun über den privaten Konstrukor (Zeile 33) durchgeführt und dann im Attribut instance (Zeile 21) gespeichert.

Des Weiteren stellt die Klasse die nachfolgenden Funktionen zur Verfügung:

getChainsPath (Zeile 66)
: Gibt den Pfad innerhalb der Datenstruktur bis zu dem Ordner des GenesisBlocks zurück. Diese werden für interne Zwecke genutzt, um die Pfade zu den einzelnen Blöcken zu ermitteln.

newDatabase (Zeile 72)
: Erstellt eine neue Chain in dem der Name der Chain und das Passwort der Daten-

bank an den aktuellen GenesisBlock weitergegeben wird. Der Genesisblock verwaltet die einzelnen Ketten.

getData (Zeile 85)
: Liest alle Transaktionen aus einer Kette aus. Hierfür wird der Befehl ebenfalls an den aktuellen Genesisblock weitergegeben. Die Daten werden als Liste zurückgegeben, welche vom Aufrufer verarbeitet werden können.

AddData (Zeile 92)
: Nimmt neue Daten entgegen und gibt diese an den Genesisblock weiter. Dieser leitet diese an die entsprechende Kette zur Speicherung weiter.

ConsistencyTestmode (Zeile 99)
: Bildet die beiden Testmodi der Konsistenzprüfung ab.

ConsistencyTest (Zeile 106)
: Führt eine Konsistenzprüfung auf den Blöcken in der Chain durch. Hierbei wird entweder Vorwärts oder Rückwärts geprüft und sichergestellt, dass keine Änderung der Daten vorgenommen wurde.

Block.CS

Listing A.2: Code der Klasse Block

```
1  using Data_Security_Layer.Utils;
2  using System.Text.Json;
3
4  namespace Data_Security_Layer.Blocks
5  {
6      internal class Block
7      {
8          private string _chainHash;
9          private string _hash;
10         private string _previousHash;
11         private string _version = "1";
12         private string _merkleRoot;
13         private DateTime _time;
14         private List<string> _transactions = new List<string>();
15
16         //Event-Handler
17         public event EventHandler? DataAdded;
18         public event EventHandler? BlockFilled;
19
20         //Standard Contructor for building new Blocks
```

```
21          public Block(string ChainHash, string Hash,string previousHash,
                EventHandler onBlockFilled)
22          {
23              _chainHash = ChainHash;
24              _hash = Hash;
25              _time = DateTime.Now;
26              _merkleRoot = string.Empty;
27              _previousHash = previousHash;
28              DataAdded += BlockEventHandler.OnDataAdded;
29              DataAdded?.Invoke(this, EventArgs.Empty);
30              BlockFilled += onBlockFilled;
31          }
32
33          //Constructor for Loading Blocks from File
34          private Block(BlockDTO blockDTO, EventHandler onBlockFilled)
35          {
36              _chainHash = blockDTO.ChainHash??"";
37              _hash = blockDTO.Hash??"";
38              _merkleRoot= blockDTO.MerkleRoot??"";
39              _previousHash = blockDTO.PreviousHash??"";
40              _time = DateTime.Parse(blockDTO.Time??"");
41              _transactions = blockDTO.Transactions??[];
42              DataAdded += BlockEventHandler.OnDataAdded;
43              BlockFilled += onBlockFilled;
44          }
45
46          //Public Properties for the JSON-Serializer
47          public string ChainHash { get { return _chainHash; } }
48          public string Hash { get { return _hash; } }
49          public string PreviousHash {  get { return _previousHash; } }
50          public string MerkleRoot {  get { return _merkleRoot; } }
51          public string Time { get { return _time.ToString(); } }
52          public List<string> Transactions { get { return _transactions; } }
53
54          //Function to Load Blocks from File
55          public static Block loadBlockFromFile(string chainhash, string hash
                , EventHandler onBlockfilled)
56          {
57              string path = Path.Combine(DataSecurityLayer.instance.
                    GetChainsPath(), chainhash, hash);
58              path = Path.ChangeExtension(path, "bcdb");
59              FileStream File = new FileStream(path, FileMode.Open,
                    FileAccess.Read);
60              StreamReader Reader = new StreamReader(File);
61              string JSON = Reader.ReadToEnd();
```

```
62              File.Close();
63              Reader.Close();
64              Block? block = new Block(JsonSerializer.Deserialize<BlockDTO>(
                    JSON) ?? throw new ArgumentNullException(nameof(BlockDTO)),
                    onBlockfilled);
65              return block;
66          }
67
68          //Function for adding new Data to the Block
69          public void addData(string Data)
70          {
71              this._transactions.Add(Data);
72              _merkleRoot = MerkleRootCalculator.calculateMerkleRoot(
                    _transactions);
73              DataAdded?.Invoke(this, EventArgs.Empty);
74              if (this._transactions.Count == 64)
75              {
76                  BlockFilled?.Invoke(this, EventArgs.Empty);
77              }
78          }
79
80          //Function to get the Data from the Block
81          public List<string> getData()
82          {
83              return _transactions;
84          }
85
86          //Function for the Consistencytest for testing if the MerkleRoot is
                Still working out
87          internal string recalculateMerkleroot()
88          {
89              return MerkleRootCalculator.calculateMerkleRoot(this.
                    _transactions);
90          }
91      }
92  }
```

Die Klasse Block, welche im Listing A.2 abgebildet ist, setzt das Verhalten der Blöcke um. Hierzu werden die Daten des Blocks als Attribute bereitgestellt und Funktionen um das Verhalten zu steuern. Neben dem öffentlichen Konstrukor für die Erstellung eines neuen Blocks (Zeile 21), steht noch ein privater Konstrukor (Zeile 34) für das Laden eines Blocks aus einem BlockDTO zur Verfügung.

Für die Serialisierung in JSON stellt die Klasse einige öffentliche Eigenschaften zur Verfügung (Zeile 47ff.), da nur diese durch die genutzten JSON Bibliothek gespeichert wer-

den können. Die gespeicherten Eigenschaften können durch die statische Funktion loadBlockFromFile (Zeile 55) aus der gespeicherten Datei gelesen und hieraus ein BlockDTO erstellt werden. Mit diesen kann mittels des privaten Konstruktors wieder ein Blockobjekt erstellt werden.
Die Funktion addData (Zeile 69) stellt die Funktion zum Hinzufügen von Daten in den Block bereit. Hierbei kommt eine Besonderheit der Implementierung zu tragen. Die Funktion löst das Event DataAdded (Zeile 17) aus. Hierdurch wird das Blockobjekt selber an den Eventhandler übergeben und die Datei, welche die öffentlichen Eigenschaften des Blocks beinhaltet wird aktualisiert oder wenn diese noch nicht existiert, wird sie erstellt. Ebenfalls kann die Funktion auch das Event BlockFilled (Zeile 18) auslösen, hierdurch wird der Block selber an die Kette übergeben und diese berechnet die Adresse des neuen Blocks und erstellt diesen. Das Event wird ausgelöst, wenn sich 64 Transaktionen im Block befinden.
Die Funktion getData (Zeile 81) gibt dem Aufrufer die Liste der Transaktionen zurück, damit diese verarbeitet werden können. Die Funktion recalculateMerkleroot (Zeile 87), ist ein Teil der Datenintegritätsprüfung und berechnet die Merkle-Root der gespeicherten Transaktionen neu, um diese mit dem im Block-Header gespeicherten Wert vergleichen zu können.

BlockDTO.cs

Listing A.3: Code der Klasse BlockDTO

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Security.Cryptography.X509Certificates;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace Data_Security_Layer.Blocks
9  {
10     internal class BlockDTO
11     {
12         public string? ChainHash {  get; set; }
13         public string? Hash {  get; set; }
14         public string? MerkleRoot {  get; set; }
15         public string? PreviousHash { get; set; }
16         public string? Time {  get; set; }
17         public List<string>? Transactions { get; set; }
18     }
19
20
```

```
}
```

Die Klasse BlockDTO, dargestellt in Listing A.3, ist eine Hilfsklasse für das erstellen eines Blocks aus der JSON-Datei, welche seine Eigenschaften gespeichert hat. Hierfür werden für alle Eigenschaften, welche geladen werden, in dem DTO gespeichert und dieses Objekt dann an den privaten Konstruktor der Block-Klasse übergeben. Auf die Implementierung eines Konstruktors für die Klasse wurde verzichtet, der automatisch entstehende Standardkonstruktor ist ausreichend.

BlockEventHandler

Listing A.4: Code der Klasse BlockEventHandler

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Security.Cryptography;
using System.Text;
using System.Text.Json;
using System.Threading.Tasks;

namespace Data_Security_Layer.Blocks
{
    internal static class BlockEventHandler
    {
        public static void OnDataAdded(object? sender, EventArgs e)
        {
            //test if the sender is valid
            if (sender == null)
            {
                throw new ArgumentNullException(nameof(sender));
            }
            else
            {
                //Calculating the string for saving the sender
                string path = string.Empty;
                //Calculation if the sender is a GenesisBlock
                if (sender.GetType() == typeof(GenesisBlock))
                {

                    GenesisBlock genesisBlock = sender as GenesisBlock??
                        throw new ArgumentNullException(nameof(sender));
                    path = Path.Combine(DataSecurityLayer._BasePath??throw
                        new ArgumentNullException(nameof(DataSecurityLayer))
                        , genesisBlock.Hash);
```

```
31                      path = Path.ChangeExtension(path, ".bcdb");
32                  }
33                  //Calculation if the sender is a Block
34                  else if (sender.GetType() == typeof(Block))
35                  {
36                      Block? block = sender as Block ?? throw new
                            ArgumentNullException(nameof(sender));
37                      path = Path.Combine(DataSecurityLayer.instance.
                            GetChainsPath(), block.ChainHash, block.Hash);
38                      path = Path.ChangeExtension(path, ".bcdb");
39                  }
40                  //Calculation if the sender is a Chain
41                  else if (sender.GetType() == typeof(Chain))
42                  {
43                      Chain? chain = sender as Chain ?? throw new
                            ArgumentNullException(nameof(sender));
44                      path = Path.Combine(DataSecurityLayer.instance.
                            GetChainsPath(), chain.Hash);
45                      path = Path.ChangeExtension(path, ".bcdb");
46                  }
47                  //Abbort if the sender is not a Element to be saved
48                  else
49                  {
50                      return;
51                  }
52                  //Saving the Data of the sender
53                  string JSON = JsonSerializer.Serialize(sender);
54                  FileStream File = new FileStream(path, FileMode.
                        OpenOrCreate, FileAccess.Write);
55                  StreamWriter Writer = new StreamWriter(File, Encoding.UTF8)
                        ;
56                  Writer.Write(JSON);
57                  Writer.Flush();
58                  File.Flush();
59                  File.Close();
60              }
61
62          }
63      }
64  }
```

Die statische Klasse BlockEventHandler, deren Code in Listing A.4 abgebildet ist, stellt eine zentrale Klasse für die verschiedenen Blöcke da. Sowohl der GenesisBlock, als auch die Ketten und die Blöcke greifen auf den von der Klasse zur Verfügung gestellten Event-Handler OnDataAdded (Zeile 14) zu.

Wenn ein Objekt das Event DataAdded auslöst, dann übergibt sich dieses selbst an den EventHandler. Dieser ermittelt, um welche Art Objekt es sich handelt (Zeile 26 - 51) und ermittelt für die verschiedenen Objekttypen den Pfad unter denen diese gespeichert werden sollen.

Anschließend wird die Speicherung des Daten durchgeführt, in dem mit dem JSONSerializer ein JSONstring erzeugt wird und dieser unter dem Pfad in einer BCDB-Datei (Akronym für BlockChainDataBase) gespeichert wird. Die Dateien können mit einem beliebigen Texteditor eingesehen werden.

Chain.cs

Listing A.5: Code der Klasse Chain

```
1  using System;
2  using System.Collections.Generic;
3  using System.IO;
4  using System.Linq;
5  using System.Net.Security;
6  using System.Reflection;
7  using System.Runtime.CompilerServices;
8  using System.Text;
9  using System.Text.Json;
10 using System.Threading.Tasks;
11 using Data_Security_Layer.Utils;
12
13 namespace Data_Security_Layer.Blocks
14 {
15     internal class Chain
16     {
17         public event EventHandler DataAdded;
18         private List<Block> _Blocks;
19         private string _HashOfHighestBlockATM;
20         private string _chainHash;
21         private string _chainName;
22         private Block _actualBlock;
23
24         //Constructor to Build a new Chain
25         public Chain(string ChainHash, string ChainName)
26         {
27             _Blocks = new List<Block>();
28             _HashOfHighestBlockATM = string.Empty;
29             _chainHash = ChainHash;
30             _chainName = ChainName;
31             DataAdded += BlockEventHandler.OnDataAdded;
```

```
32              Directory.CreateDirectory(Path.Combine(DataSecurityLayer.
                    instance.GetChainsPath(), _chainHash));
33              _Blocks.Add(firstBlock());
34              _actualBlock = _Blocks[0];
35              _HashOfHighestBlockATM = _actualBlock.Hash;
36              DataAdded?.Invoke(this, EventArgs.Empty);
37          }
38
39          //Property for use in the Class own functions
40          private Block ActualBlock
41          {
42              set
43              {
44                  _actualBlock = value;
45              }
46          }
47
48          //Private Constructor for loading Chains from file
49          private Chain(ChainDTO chainDTO)
50          {
51              _Blocks = new List<Block>();
52              _HashOfHighestBlockATM = chainDTO.HighestHash??"";
53              _chainHash = chainDTO.Hash??"";
54              _chainName = chainDTO.ChainName??"";
55              DataAdded += BlockEventHandler.OnDataAdded;
56              _actualBlock = Block.loadBlockFromFile(_chainHash,
                    _HashOfHighestBlockATM, NewBlock);
57          }
58
59          //Properties for the JSON-Serializer
60          public string Hash
61          {
62              get
63              {
64                  return _chainHash;
65              }
66          }
67
68          public string HighestHash
69          {
70              get
71              {
72                  return _HashOfHighestBlockATM;
73              }
74          }
```

```
75
76          public string ChainName
77          {
78              get
79              {
80                  return _chainName;
81              }
82          }
83
84          //Function to load a Chain from File
85          public static Chain loadChainFromFile(string Hash)
86          {
87              string path = Path.Combine(DataSecurityLayer.instance.
                    GetChainsPath(), Hash);
88              path = Path.ChangeExtension(path, "bcdb");
89              FileStream File = new FileStream(path, FileMode.Open,
                    FileAccess.Read);
90              StreamReader Reader = new StreamReader(File);
91              string JSON = Reader.ReadToEnd();
92              File.Close();
93              Reader.Close();
94              Chain? chain = new Chain(JsonSerializer.Deserialize<ChainDTO>(
                    JSON) ?? throw new ArgumentNullException(nameof(ChainDTO)));
95              return chain;
96          }
97
98          //Function to add new Data to the last Block of the Chain
99          internal void newData(string data)
100         {
101             _actualBlock.addData(data);
102         }
103
104         //Function to build the first Block of the chain
105         private Block firstBlock()
106         {
107             Block block = new Block(_chainHash, SHA256Hashing.getHash(Path.
                    Combine(DataSecurityLayer.instance.GetChainsPath(),
                    _chainHash)),_chainHash, NewBlock);
108             _HashOfHighestBlockATM = block.Hash;
109             return block;
110         }
111
112         //Event-Handler to build a new Block when a Block is filled
113         private void NewBlock(object? sender, EventArgs e)
114         {
```

```
115             if(sender == null)
116             {
117                 throw new ArgumentNullException(nameof(sender));
118             }
119             else
120             {
121                 Block senderBlock = sender as Block ?? throw new
                        ArgumentNullException(nameof(sender));
122                 Block block = new Block(senderBlock.ChainHash, SHA256
                        Hashing.getHash(senderBlock.MerkleRoot + senderBlock.
                        Hash), senderBlock.Hash, NewBlock);
123                 _actualBlock = block;
124                 _HashOfHighestBlockATM = block.Hash;
125                 DataAdded?.Invoke(this, EventArgs.Empty);
126             }
127
128         }
129
130         //Function to receive the whole Data saved in the Blocks of the
                Chain
131         public List<string> getData()
132         {
133             Block tmpBlock = Block.loadBlockFromFile(_chainHash, SHA256
                    Hashing.getHash(Path.Combine(DataSecurityLayer.instance.
                    GetChainsPath(), _chainHash)), NewBlock);
134             List<string> data = new List<string>();
135             bool firstRun = true;
136             do
137             {
138                 if (!firstRun)
139                 {
140                     tmpBlock = Block.loadBlockFromFile(_chainHash, SHA256
                            Hashing.getHash(tmpBlock.MerkleRoot + tmpBlock.Hash)
                            , NewBlock);
141                 }
142                 firstRun = false;
143                 data.AddRange(tmpBlock.getData());
144             } while (tmpBlock.Hash != _HashOfHighestBlockATM);
145             return data;
146         }
147
148         //Function to test if there was a change in the Data
149         internal bool BackwardsConsistencyTest()
150         {
151             string ParentHash = string.Empty;
```

```
152             bool result = true;
153             Block testBlock = Block.loadBlockFromFile(_chainHash,
                    HighestHash, BlockEventHandler.OnDataAdded);
154             bool FirstRun = true;
155             do
156             {
157                 if (!FirstRun)
158                 {
159                     testBlock = Block.loadBlockFromFile(_chainHash,
                            ParentHash, BlockEventHandler.OnDataAdded);
160                 }
161                 FirstRun = false;
162                 ParentHash = testBlock.PreviousHash;
163                 if(testBlock.MerkleRoot != testBlock.recalculateMerkleroot(
                        ))
164                 {
165                     result = false;
166                     break;
167                 }
168             } while (ParentHash != _chainHash && result == true);
169             return result;
170         }
171 
172         //Function to test if there was a change in the Data
173         internal bool ForwardsConsistencyTest()
174         {
175             bool result = true;
176             Block testBlock = Block.loadBlockFromFile(_chainHash,
                    HighestHash, BlockEventHandler.OnDataAdded);
177             do
178             {
179                 if (testBlock.MerkleRoot != testBlock.recalculateMerkleroot
                        ())
180                 {
181                     result = false;
182                     break;
183                 }
184                 try
185                 {
186                     testBlock = Block.loadBlockFromFile(_chainHash, SHA256
                            Hashing.getHash(testBlock.MerkleRoot + testBlock.
                            Hash), NewBlock);
187                 }
188                 catch
189                 {
```

```
190                     result |= false;
191                     break;
192                 }
193
194             } while (testBlock.Hash != _HashOfHighestBlockATM && result ==
                    true);
195             return result;
196
197         }
198     }
199 }
```

Das Listing A.5 zeigt den Code der Klasse Chain. Diese stellt ähnlich wie die Klasse Block die Attribute für die Ketten der Blockchain bereit (Zeile 18-22). Ebenfalls wird das Event DataAdded umgesetzt, welches die Daten der Chain bei Änderungen an dieser speichert. Wie auch der Block besitzt die Chain-Klasse zwei Konstruktoren. Einen öffentlichen (Zeile 25) für die Erstellung einer neuen Chain und einen privaten (Zeile 49), welcher über einen ChainDTO aus den Dateien eine Chain erstellt.
Neben den öffentlichen Eigenschaften (Zeile 59-82) für die Erstellung der JSONstrings ist ebenfalls eine Eigenschaft als Setter für den Wert des aktuell letzten Blocks enthalten (Zeile 40). Nachfolgend werden die weiteren Funktionen beschrieben:

loadChainFromFile (Zeile 85)
: Funktion lädt ein Chain-Objekt aus den JSON-Dateien, indem die Dateien gelesen werden und über den JSONSerializer in eine ChainDTO deserialisiert werden. Anschließend wird das DTO-Objekt an privaten Konstruktor der Klasse übergeben.

newData (Zeile 99)
: Übernimmt neue Daten vom Genesisblock und übergibt diese an den aktuell höchsten Block der Kette für die Speicherung.

firstBlock (Zeile 105)
: Erstellt den ersten Block der Kette, wenn die Chain erstellt wird. Für den Hashwert, wird der Hashwert der Chain mit dem Hashwert des GenesisBlocks kombiniert und hieraus der Hashwert des ersten Blocks berechnet.

NewBlock (Zeile 113)
: Dient als Eventhandler für das DataFilled-Event der einzelnen Blöcke und erzeugt aus den Daten das Blocks einen neuen Block, indem dessen Hashwert mit der MerkleRoot konkateniert wird und das Ergebnis gehasht wird. Gleichzeitig wird der neue Block als neuer höchster Block gespeichert und der Hashwert des höchsten Blocks aktualisiert.

getData (Zeile 131)
: Durchläuft alle Blöcke der Kette und fügt die in den jeweiligen Blöcken gespeicherten Transaktionen zu einer Liste zusammen.

BackwardsConsistencyTest (Zeile 149)
: Funktion zum Testen der Datenintegrität vom aktuell letzten Block zur Chain. Indem die MerkleRoot neu kalkuliert wird und über den im Block gespeicherten Hashwerts des Vorgänger Blocks dieser geladen wird. Bis der Hashwert gleich dem Hashwert der Chain ist.

ForwardsConsistencyTest (Zeile 173)
: Funktion zum Testen der Datenintegrität von der Chain bis zum aktuell letzten Block, indem beginnend mit dem ersten Block der Chain die Hashwerte des Nachfolge Blocks berechnet werden, bis dieser Hashwert dem gespeicherten Wert des aktuell höchsten Blocks entspricht.

ChainDTO.cs

Listing A.6: Code der Klasse ChainDTO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Data_Security_Layer.Blocks
{
    internal class ChainDTO
    {
        public string? Hash {  get; set; }
        public string? HighestHash {  get; set; }
        public string? ChainName { get; set; }
    }
}
```

Genau wie die BlockDTO-Klasse ist auch die ChainDTO-Klasse, welche in Listing A.6 abgebildet ist, als Hilfmittel für das Laden der Chain-Blöcken aus den erzeugten Dateien zuständig. Hierbei wurden ebenfalls die gespeicherten Attribute abgebildet und auf einen Konstruktor verzichtet.

GenesisBlock.cs

Listing A.7: Code der Klasse GenesisBlock

```
using Data_Security_Layer.Utils;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Text.Json;

namespace Data_Security_Layer.Blocks
{
    internal class GenesisBlock
    {
        public event EventHandler DataAdded;
        private string _hash;
        private string _version = "1";
        private string _merkleroot;
        private DateTime _time;
        private List<string> _transactions;
        private List<string> _chains;
        private Dictionary<string, Chain> _loadedChains;

        //public Properties for the JSONSerializer
        public string Hash
        {
            get
            {
                return _hash;
            }
        }

        public string Version
        {
            get
            {
                return _version;
            }
        }

        public string Merkleroot
```

```
43          {
44              get
45              {
46                  return _merkleroot;
47              }
48          }
49
50          public string Time
51          {
52              get
53              {
54                  return _time.ToString();
55              }
56          }
57
58          public List<string> Chains
59          {
60              get
61              {
62                  return _chains;
63              }
64          }
65
66          public List<string> Transactions
67          {
68              get
69              {
70                  return _transactions;
71              }
72          }
73
74          //Public Constructor
75          public GenesisBlock(string hash)
76          {
77              _hash = hash;
78              _merkleroot = string.Empty;
79              _chains = new List<string>();
80              _time = DateTime.Now;
81              Directory.CreateDirectory(DataSecurityLayer._BasePath + "\\" +
                    _hash);
82              DataAdded += BlockEventHandler.OnDataAdded;
83              DataAdded.Invoke(this, EventArgs.Empty);
84              _transactions = new List<string>();
85              _loadedChains = new Dictionary<string, Chain>();
86          }
```

```
87
88          //Constructor for Loading the Block from File
89          private GenesisBlock(GenesisBlockDTO DTO)
90          {
91              _hash = DTO.Hash;
92              _version = DTO.Version;
93              _merkleroot = DTO.Merkleroot;
94              _time = DateTime.Parse(DTO.Time);
95              _transactions = DTO.Transactions?? new List<string>();
96              _chains = DTO.Chains?? new List<string>();
97              DataAdded += BlockEventHandler.OnDataAdded;
98              _loadedChains = new Dictionary<string, Chain>();
99          }
100
101         //Adding a new DataBase
102         public void addChain(string Name, string secretKey)
103         {
104             string searchstring = "NewChain:" + Name;
105
106             //Checking if the Database is already existing
107             if (_transactions.Contains(searchstring))
108             {
109                 throw new ArgumentException("The Database is already
                        existing");
110             }
111             else
112             {
113                 //Adding the new Chain
114                 _transactions.Add("NewChain:" + Name);
115                 _merkleroot = MerkleRootCalculator.calculateMerkleRoot(
                        _transactions);
116                 string chainhash = SHA256Hashing.getHash(_merkleroot +
                        secretKey);
117                 Chain chain = new Chain(chainhash, Name);
118                 _chains.Add(chainhash);
119                 _loadedChains.Add(Name, chain);
120                 DataAdded?.Invoke(this, EventArgs.Empty);
121             }
122
123         }
124
125         //Static Function for loading the Genesisblock from JSON
126         internal static GenesisBlock loadGenesisBlockFromFile(string path)
127         {
128             path = Path.ChangeExtension(path, "bcdb");
```

```
129             FileStream File = new FileStream(path, FileMode.Open,
                    FileAccess.Read);
130             StreamReader Reader = new StreamReader(File);
131             string JSON = Reader.ReadToEnd();
132             File.Close();
133             Reader.Close();
134             GenesisBlock? genesisBlock = new GenesisBlock(JsonSerializer.
                    Deserialize<GenesisBlockDTO>(JSON)??throw new
                    ArgumentNullException(nameof(JsonSerializer)));
135             return genesisBlock;
136         }
137
138         //Function for getting the acces to a DataBase
139         internal Chain getDataBase(string DataBaseName, string SecretKey)
140         {
141             string searchstring = "NewChain:" + DataBaseName;
142             if (_transactions.Contains(searchstring))
143             {
144                 int index = _transactions.IndexOf(searchstring);
145                 List<string> transactionsForMerkleRootCalculation = new
                        List<string>();
146                 transactionsForMerkleRootCalculation = _transactions.Slice(
                        0,index+1);
147                 string chainHash = SHA256Hashing.getHash(
                        MerkleRootCalculator.calculateMerkleRoot(
                        transactionsForMerkleRootCalculation) + SecretKey);
148                 if (_loadedChains.ContainsKey(chainHash))
149                 {
150                     return _loadedChains[chainHash];
151                 }
152                 else
153                 {
154                     _loadedChains.Add(chainHash, Chain.loadChainFromFile(
                            chainHash));
155                     return _loadedChains[chainHash];
156                 }
157             }
158             else
159             {
160                 throw new ArgumentException("The called Database is not
                        existend");
161             }
162         }
163
```

```
164         //getting the Database over the DataBaseHash instead of using the
                name and key
165         internal Chain getDataBase(string DataBaseHash)
166         {
167             if (_loadedChains.ContainsKey(DataBaseHash))
168             {
169                 return _loadedChains[DataBaseHash];
170             }
171             else
172             {
173                 _loadedChains.Add(DataBaseHash, Chain.loadChainFromFile(
                        DataBaseHash));
174                 return _loadedChains[DataBaseHash];
175             }
176         }
177
178         //Function for getting the Hashes of all Databases in this
                Databasesystem
179         internal List<string> getDataBases()
180         {
181             return Chains;
182         }
183     }
184 }
```

Die Klasse des Genesisblocks, welche in Listing A.7 dargestellt ist, stellt die Basis für alle Chains dar. Wie bereits die Klassen Block und Chain stellt die Klasse einen öffentlichen Konstruktor für die Erstellung eines neuen GenesisBlocks (Zeile 75) und einen privaten Konstruktor (Zeile 89) für das Laden aus der Datei zur Verfügung. Ebenfalls implementiert die Klasse das Event DataAdded (Zeile 16) und realisiert hierüber die Speicherung der Daten.
Hierüber hinaus werden die nachfolgenden Funktionen zur Verfügung gestellt:

addChain (Zeile 102)

Fügt eine neue Datenbank zum Datenbanksystem hinzu. Hierfür wird eine Transaktion zum GenesisBlock hinzugefügt. Diese Besteht aus dem String „NewChain:*(Name der neuen Datenbank)* “. Anschließend wird aus den Transaktionen eine neue Merkleroot berechnet. Für den Hashwert der neuen Kette wird die MerkleRoot mit dem angegeben Schlüssel für die Kette konkateniert und aus dem Ergebnis der Hashwert berechnet. Nun wird die Chain als Objekt erstellt und anschließend die Metadaten des Genesisblocks aktualisiert und der Block gespeichert, indem das Event DataAdded ausgelöst wird.

loadGenesisBlockFromFile (Zeile 126)

Die Funktion loadGenesisBlockFromFile stellt wie bei der Implementierung der Chain und des Blocks die Funktion dar, welche das Objekt mit Hilfe des zugehörigen DTOs aus den JSON Dateien laden kann.

getDataBase (Zeile 139 + 165)

Ermöglicht den Aufrufer eine Instanz einer Datenbank zu erhalten. Die Überladung in Zeile 139 benötigt hierfür den Namen und den Schlüssel der Datenbank und berechnet hieraus den Hashwert. Die zweite Überladung in Zeile 165 kann genutzt werden, wenn der Hashwert bekannt ist.

getDataBases (Zeile 179)

Liefert eine Liste aller Hashwerte, welche im GenesisBlock geladen sind an den Aufrufer.

GenesisBlockDTO.cs

Listing A.8: Code der Klasse GenesisBlockDTO

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Data_Security_Layer.Blocks
8  {
9      internal class GenesisBlockDTO
10     {
11         public string Hash { get; set; }
12         public string Version { get; set; }
13         public string Merkleroot { get; set; }
14         public string Time { get; set; }
15         public List<string> Transactions { get; set; }
16         public List<string>Chains{ get; set; }
17     }
18 }
```

Auch die in Listing A.8 abgebildete Klasse ist wie die anderen beiden DTO-Klassen für das Laden der entsprechenden Klasse aus den Dateien mit verantwortlich. Es sind ebenfalls die gespeicherten Eigenschaften als Attribute abgebildet und kein gesonderter Konstruktor erstellt.

MerkleRootCalculator.cs

Listing A.9: Code der Klasse MerkleRootCalculator

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Intrinsics.Arm;
using System.Text;
using System.Threading.Tasks;
using System.Security.Cryptography;
using System.Transactions;

namespace Data_Security_Layer.Utils
{
    internal static class MerkleRootCalculator
    {
        //Geting the strings to Hashvalues for the first level of nodes
        public static string calculateMerkleRoot(List<String> transactions)
        {
            List<String> HashList = new List<string>();
            using (SHA256 mySHA256 = SHA256.Create())
            {
                if (transactions.Count == 0)
                {
                    return "";
                }

                foreach (string transaction in transactions)
                {
                    byte[] hashValue = Encoding.UTF8.GetBytes(transaction);
                    byte[] hash = mySHA256.ComputeHash(hashValue);
                    string HashString = string.Empty;
                    foreach (byte b in hash)
                    {
                        HashString += String.Format("{0:x2}", b);
                    }
                    HashList.Add(HashString);
                }
            }

            return calculateMerkleRoot(HashList, true);
        }

        //condensing the list of Hashvalues to one Hashvalue
```

```
42          public static string calculateMerkleRoot(List<String> transactions,
                bool alreadyHashed)
43          {
44              List<string> HashList = new List<string>();
45              for (int i = 0; i < transactions.Count; i++)
46              {
47                  string tmpHashValueString = string.Empty;
48  //Diasabling the warning, that the ArgumentOutOfRangeException variable e
        is never used
49  #pragma warning disable CS0168 // Variable ist deklariert, wird jedoch
        niemals verwendet
50                  try
51                  {
52                      tmpHashValueString = transactions[i] + transactions[++i
                            ];
53
54                  }
55                  catch (ArgumentOutOfRangeException e)
56                  {
57
58                      tmpHashValueString = transactions[--i] + transactions[i
                            ];
59                  }
60  #pragma warning restore CS0168 // Variable ist deklariert, wird jedoch
        niemals verwendet
61                  using (SHA256 mySHA256 = SHA256.Create())
62                  {
63                      byte[] hashValue = Encoding.UTF8.GetBytes(
                            tmpHashValueString);
64                      byte[] hash = mySHA256.ComputeHash(hashValue);
65                      string HashString = string.Empty;
66                      foreach (byte b in hash)
67                      {
68                          HashString += String.Format("{0:x2}", b);
69                      }
70                      HashList.Add(HashString);
71                  }
72              }
73              if (HashList.Count == 1)
74              {
75                  return HashList[0];
76              }
77              else
78              {
79                  return calculateMerkleRoot(HashList, true);
```

```
80                 }
81             }
82         }
83 }
```

Die Klasse MerkleRootCalculator ist eine der beiden Hilfsklassen für die gesamte Schicht. Der zugehörige Code ist in Listing A.9 abgebildet. Es handelt sich um eine statische Klasse, welche zwei gleichnamige Funktionen bereitstellt, welche sich nur anhand der Parameter unterscheiden.
Die erste Funktion calculateMerkleRoot (Zeile 15) stellt die Funktion da, welche von den anderen Klassen in der Schicht aufgerufen wird. Ihr wird die Liste der Transaktionen in einem Block übergeben. Diese Funktion wandelt die gesamte Liste in Hashwerte um und fügt diese einer neuen Liste hinzu.
Die Liste der Hashwerte wird nun der zweiten Funktion calculateMerkleRoot (Zeile 42) übergeben und der Rückgabewert an den Aufrufer der ersten Funktion zurückgegeben. Die zweite Funktion kann nun aus der Liste der Hashwerte einen einzelnen Wert erstellen, indem immer zwei Einträge der Liste konkateniert werden und aus diesen ein neuer Hashwert erstellt wird. Sollte die Liste eine ungerade Anzahl an Werten beinhalten, wird der letzte Wert mit sich selbst konkateniert und gehasht.
Die neuen Hashwerte werden zu einer neuen Liste zusammengefasst. Mit dieser neuen Liste ruft sich die Funktion selber auf und fasst diese wieder auf die selbe Weise zusammen, bis nur noch ein einzelner Wert in der Liste ist, dann wird dieser Wert wieder an die erste Funktion zurückgegeben.

SHA256Hashing.cs

Listing A.10: Code der Klasse SHA256Hashing

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.Transactions;
7  using System.Security.Cryptography;
8  using System.Runtime.Intrinsics.Arm;
9
10 namespace Data_Security_Layer.Utils
11 {
12     internal static class SHA256Hashing
13     {
14         //Function for unifying the hashingprocess in the project
15         internal static string getHash(string Input)
```

```
16          {
17              using(SHA256 mySHA256 = SHA256.Create())
18              {
19                  byte[] hashValue = Encoding.UTF8.GetBytes(Input);
20                  byte[] hash = mySHA256.ComputeHash(hashValue);
21                  string HashString = string.Empty;
22                  foreach (byte b in hash)
23                  {
24                      HashString += String.Format("{0:x2}", b);
25                  }
26                  return HashString;
27              }
28
29          }
30      }
31  }
```

Im Listing A.10 ist die Klasse SHA256Hashing dargestellt. Dies ist die zweite Hilfsklasse für die Schicht. Die Funktion ist statisch implementiert und stellt die statische Funktion getHash (Zeile 15) zur Verfügung. Diese Funktion nimmt einen String entgegen und bildet aus diesem einen Hashwert.

A.1.2 Data-Presentation-Layer

Nachfolgend wird der Code der Data-Presentation-Layer abgebildet und erläutert. Hierzu werden vier Abschnitte verwendet jeweils einer pro Namespace. Begonnen wird mit dem allgemeinen Namespace *Presentation_Layer* in Abschnitt A.1.2.1, anschließend werden die Eventargumente für die Layer in Abschnitt A.1.2.2 näher vorgestellt.
Des Weiteren wird in Abschnitt A.1.2.3 der Namespace *Presentation_Layer.Databases* behandelt. Abschließend wird der Namespace *Presentation_Layer.WorkerChain* in Abschnitt A.1.2.4 vorgestellt.
In der Abbildung A.1.2 ist die Dateistruktur der Schicht abgebildet. Die zuvor genanten Namespaces befinden sich jeweils in den gleichnamigen Ordnern. Die Beiden Klassen des Namespaces *Presentation_Layer* befinden sich ganz unten in der Abbildung außerhalb der Ordner.

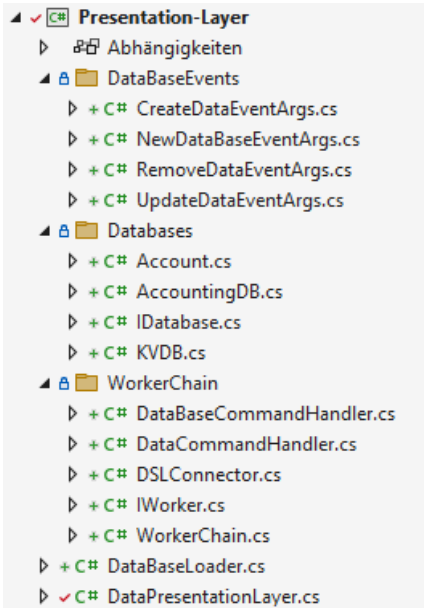


Abbildung A.2: Dateistruktur der Data-Presentation-Layer

Die Klasse DataPresentationLayer bildet hierbei den Einstieg in die Schicht und stellt alle Befehle zur Verfügung, welche für ein die Schicht nutzendes Programm verfügbar sind. Die Klasse DataBaseLoader ist für das Laden der Datenbank aus der Datei verantwortlich. Im Namespace *DataBaseEvents* befinden sich die Klassen, welche für die Datenbankevents benötigt werden. Hierbei handelt es sich jeweils um eine Klasse, welche die Eigenschaften der in C# enthaltenen Klasse EventArgs geerbt hat und diese erweitert.
Die Implementierung der beiden Datenbankarten findet im Namespace *Databases* statt. Bei der Klasse KVDB handelt es sich um eine einfach Key-Value-Datenbank, während die Klasse AccountingDB eine Datenbank im Minimalentwurf für die Verwaltung von

Buchhaltungsdaten darstellt. Die beiden anderen Klassen sind Hilfsklassen, welche später erläutert werden.
Der letzte Namespace *WorkerChain* enthält die Zuständigkeitskette, welche die Befehle der Datenbanken verarbeitet. Und somit die Verbindung zwischen der Klasse DataPresentationLayer, den Datenbanken und der Data-Security-Layer darstellt.

A.1.2.1 Namespace: Presentation_Layer

Der Namespace *Presentation_Layer* stellt zwei Klassen bereit. Zum einen die Klasse DataPresentationLayer, welche den Einstieg in die Data-Presentation-Layer bildet und dem Nutzer die Funktionen bietet mit den Datenbanken zu arbeiten, zum anderen die Klasse DataBaseLoader, welche die aus der Data-Security-Layer abgerufenen Events verarbeitet und erneut ausführt.

DataPresentationLayer.cs

Listing A.11: Code der Klasse DataPresentationLayer

```
using Presentation_Layer.Databases;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Presentation_Layer.DataBaseEvents;
using Presentation_Layer.WorkerChain;
using System.Security.Cryptography;
using System.Reflection.Metadata.Ecma335;
using System.Diagnostics;
using System.Runtime.Intrinsics.X86;
using System.Xml.Linq;

namespace Presentation_Layer
{
    internal class DataPresentationLayer
    {
        //Instance as part of the Singleton-Pattern
        private static DataPresentationLayer? _instance;
        //DataBaseAcces through DataBaseNaming
        private Dictionary<string, IDatabase>? _databases;
        //Path of the DataBase System
        private string? _path;
        //Data-Security-Layer Connector
        private DSLConnector? _connector;
```

```
28          //WorkerChain
29          private WorkerChain.WorkerChain? newCommands;
30          //DataBaseLoader for Loading from File
31          private DataBaseLoader? _loader;
32          //Events for the Database
33          internal event NewDataBaseEvent? CreateNewDataBase;
34          internal event CreateDataEvent? CreateNewData;
35          internal event UpdateDataEvent? UpdateData;
36          internal event RemoveDataEvent? RemoveData;
37
38          //private Constructor for the Singleton-Pattern
39          private DataPresentationLayer()
40          {
41
42
43          }
44
45          //Setting the path of the DataBasesystem
46          public void activateDatabase(string path, string DataBaseSystemName
                , string secretkey)
47          {
48              _path = path;
49              _connector = new DSLConnector(_path, DataBaseSystemName,
                    secretkey);
50              newCommands = new WorkerChain.WorkerChain();
51              CreateNewDataBase += newCommands.execute;
52              CreateNewData += newCommands.execute;
53              UpdateData += newCommands.execute;
54              RemoveData += newCommands.execute;
55              newCommands.AddWorkers(new DataBaseCommandHandler());
56              newCommands.AddWorkers(new DataCommandHandler ());
57              newCommands.AddWorkers(_connector);
58              _databases = new Dictionary<string, IDatabase>();
59              _loader = new DataBaseLoader(newCommands);
60              try
61              {
62                  loadDatabase("PasswordDB");
63              }catch
64              {
65                  List<string> Fields = new List<string>()
66                  {
67                      "DBname", "SecretKey"
68                  };
69                  List<string> optionalFields = new List<string>();
70                  DataBasetype DBType = DataBasetype.KVDB;
```

```
71              NewDataBase("PasswordDB", "Passwords", DBType, Fields,
                    optionalFields);
72          }
73      }
74
75      //Delegates for the DataBaseEvents
76      public delegate void NewDataBaseEvent(object sender,
            NewDataBaseEventArgs e);
77      public delegate void CreateDataEvent(object sender,
            CreateDataEventArgs e);
78      public delegate void UpdateDataEvent(object sender,
            UpdateDataEventArgs e);
79      public delegate void RemoveDataEvent(object sender,
            RemoveDataEventArgs e);
80
81
82      //Property for the Singleton-Pattern
83      public static DataPresentationLayer Instance
84      {
85          get
86          {
87              if(_instance == null)
88              {
89                  _instance = new DataPresentationLayer();
90                  return _instance;
91              }
92              return _instance;
93          }
94      }
95
96      //returns a Dictionary with:
97      //1. A List of the non optional Fields in the Database
98      //2. A Blanko Dictionary for all the Fields in the Database
99      public Dictionary<string, object> GetDataPattern(string
            DatabaseName)
100     {
101         if(DatabaseName != null)
102         {
103             return _databases[DatabaseName].GetDataPattern();
104         }
105         else
106         {
107             throw new InvalidOperationException("The Databasesystem
                        musst be activated to use this function");
108         }
```

```
109         }
110
111
112         //Enum with the DataBaseTypes
113         public enum DataBasetype
114         {
115             KVDB,
116             Accounting
117         }
118
119         //Function to build a new DataBase
120         public void NewDataBase(string DataBaseName, string secretKey,
                DataBasetype dataBaseType, List<string>fields, List<string>
                optionalFields)
121         {
122             NewDataBaseEventArgs e = new NewDataBaseEventArgs(DataBaseName,
                    secretKey, dataBaseType, fields, optionalFields, false);
123             if (CreateNewDataBase != null)
124             {
125                 if (DataBaseName != "PasswordDB")
126                 {
127                     //Checking if the name of the new Database is already
                        in use
128                     List<Dictionary<string, string>> PasswordDBData =
                        _databases["PasswordDB"].ReadAll();
129                     foreach (Dictionary<string, string> dataset in
                        PasswordDBData)
130                     {
131                         if (dataset["DBName"] == DataBaseName)
132                         {
133                             throw new ArgumentException("There is already a
                                Database with this name");
134                         }
135                     }
136                     //Adding new Database to PasswordDB
137                     Dictionary<string, string> data = new Dictionary<string
                        , string>();
138                     data.Add("DBName", DataBaseName);
139                     data.Add("Passwords", secretKey);
140                     Create("PasswordDB", data);
141                 }
142                 CreateNewDataBase(this, e);
143             }
144             else
145             {
```

```
146             throw new InvalidOperationException("The Databasesystem
                    musst be activated to use this function");
147         }
148
149     }
150
151     //Function to Save new Data into the DataBase
152     public void Create(string DataBaseName, Dictionary<string, string>
            data)
153     {
154         if (_databases != null & CreateNewData != null)
155         {
156             string key;
157             try
158             {
159                 key = _databases[DataBaseName].GetActualKey();
160             }catch
161             {
162                 loadDatabase(DataBaseName);
163                 key = _databases[DataBaseName].GetActualKey();
164             }
165
166             CreateDataEventArgs e = new CreateDataEventArgs(data, key,
                    _databases[DataBaseName], false);
167             CreateNewData(this, e);
168         }
169         else
170         {
171             throw new InvalidOperationException("The Databasesystem
                    musst be activated to use this function");
172         }
173
174     }
175
176     //Functions to Read Data from the DataBase
177     public Dictionary<string,string> ReadKey(string DataBaseName,
            string key)
178     {
179         IDatabase database = _databases[DataBaseName];
180         return database.ReadKeyBased(key);
181     }
182
183     public List<Dictionary<string, string>>ReadFullDataBase(string
            DataBaseName)
184     {
```

```
185             IDatabase database = _databases[DataBaseName];
186             return database.ReadAll();
187         }
188 
189         //Function to read an Account out of a AccountingDB
190         public Dictionary<string, List<Dictionary<string, string>>>
                ReadAccount(string DataBaseName, string account)
191         {
192             IDatabase database = _databases[DataBaseName];
193             if (database.GetType() == typeof(AccountingDB))
194             {
195                 AccountingDB Database = (AccountingDB)database;
196                 return Database.ReadAccount(account);
197             }
198             else
199             {
200                 throw new InvalidOperationException("The choosen Database
                        is not an AccountingDB");
201             }
202 
203         }
204 
205         //Function to Update Data from the DataBase
206         public void Update(string DataBaseName, Dictionary<string, string>
                newData, string key)
207         {
208             UpdateDataEventArgs e = new UpdateDataEventArgs(newData, key,
                    _databases[DataBaseName], false);
209             UpdateData(this, e);
210         }
211 
212         //Function to Delete Data from a DataBase
213         public void Delete(string DataBaseName, string key)
214         {
215             RemoveDataEventArgs e = new RemoveDataEventArgs(key,_databases[
                    DataBaseName], false);
216             RemoveData(this, e);
217         }
218 
219         //getDatabase out of _Databases
220         internal IDatabase getDataBase(string DataBaseName)
221         {
222             if (_databases != null)
223             {
224                 return _databases[DataBaseName];
```

```
225             }
226             else
227             {
228                 throw new InvalidOperationException("The databasesystem
                        musst be activated before this function can be used");
229             }
230
231         }
232
233         //add Database to _Databases
234         internal void addDatabase(IDatabase database)
235         {
236             if(_databases != null)
237             {
238                 _databases.Add(database.Name, database);
239             }
240         }
241
242
243         //Loading a Database from File with the loader
244         public void loadDatabase(string DataBaseName)
245         {
246             _loader.ReloadDataBase(_connector.loadData(DataBaseName));
247         }
248
249         //get the Key from the PasswordDB
250         internal string GetKeyFromPasswordDB(string DataBaseName)
251         {
252             List<Dictionary<string, string>> PasswordDBData = _databases["
                    PasswordDB"].ReadAll();
253             foreach(Dictionary<string, string> data in PasswordDBData)
254             {
255                 if (data["DBName"] == DataBaseName)
256                 {
257                     return data["Passwords"];
258                 }
259                 else
260                 {
261                     throw new ArgumentOutOfRangeException("The database isn
                            't in the PasswordDB");
262                 }
263             }
264             throw new ArgumentOutOfRangeException("The database isn't in
                    the PasswordDB");
265         }
```

```
266         }
267 }
```

Die in Listing A.11 abgebildete Klasse DataPresentationLayer ist die Klasse, welche die Funktionen für Programme bereitstellt, welche den Prototypen nutzen wollen. Sie ist als Singleton implementiert. Hierdurch kann eine Instanz der Klasse nur über die Eigenschaft Instance (Zeile 83 - 94) abgerufen werden, wenn noch keine Instanz der Klasse in dieser selbst gespeichert ist, wird eine mit dem privaten Konstruktor (Zeile 39 - 43) erstellt und im Feld instance (Zeile 21) gespeichert.
Weitere in der Klasse gespeicherten Felder sind: ein Dicitionary mit den geladenen Datenbanken für den schnelleren Zugriff und Verkürzung der Ladezeiten beim Zugriffe (Zeile 23). Der Pfad unter dem die Daten der Datenbank gespeichert werden sollen (Zeile 25), der Connector zur Data-Security-Layer (Zeile 27), die Zuständigkeitskette für die Verarbeitung der Befehle (Zeile 29) und eine Instanz des DataBaseLoaders, welcher das Laden von Datenbanken aus den gespeicherten Events ermöglicht (Zeile 31).
Die Events für die Verarbeitung der Befehle sind in den Zeilen 33 bis 36 definiert, da es sich um Individuelle Events handelt und nicht um das Standardevent, werden hierzu eigene Delegates definiert (Zeile 76 - 79).
Die Funktion activateDatebase (Zeile 46 - 73) aktiviert, dass System. Hierfür wird der Connector für die Data-Security-Layer angelegt und wenn nötig eine neue Blockchain angelegt. Die WorkerChain initialisiert und anschließend wird versucht die PasswordDB aus dem Datenbanksystem zu laden, sollte keine PasswordDB angelegt sein, wird diese nun erstellt.
Die Funktion GetDataPattern (Zeile 99 - 109) bekommt vom Aufrufer den Namen der Datenbank übergeben und gibt das DataPattern der Datenbank zurück. Das DataPattern ist ein Dicitionary, welches zwei Einträge besitzt zum einen den Schlüssel dataDictionary, unter welchem ein Dicitionary, welches alle Felder der Datenbank als Schlüssel ohne Werte enthält und vom Aufrufer befüllt werden kann, gespeichert ist, zum anderen unter dem Schlüssel optionalFields die optionalen Felder der Datenbank.
Die CRUD-Operationen sind in den nachfolgend erläuterten Funktionen umgesetzt:

NewDataBase (Zeile 120 - 149)
: Funktion zum Erzeugen einer neuen Datenbank. Hierfür übergibt der Aufrufer den Namen der Datenbank, das Password, den Datenbanktyp aus der Datenbanktypenumeration (Zeile 113 - 117), eine Liste der Felder und eine der optionalen Felder. Anschließend wird das Event für eine neue Datenbank ausgelöst.

Create (Zeile 152 - 174)
: Hinzufügen von Daten zu der per Namen bestimmten Datenbank, indem das Event für neue Daten ausgelöst wird.

ReadKey (Zeile 177 - 181), ReadAccount (Zeile 190 - 203), ReadFullDataBase (Zeile 183 - 187)
Funktionen zum Lesen der Daten aus den einzelnen Datenbanken, die Befehle, werden direkt an die Datenbank weitergegeben ohne Event.

Update (Zeile 206 - 210)
Funktion zum Ersetzen von Daten, welche unter einem bestimmten Schlüssel gespeichert sind über das Update-Event.

Delete (Zeile 213 - 217)
Funktion zum Löschen von Daten über das entsprechende Event.

Neben den Datenoperation sind noch allgemeine Funktionen für den Ablauf in der Data-Presentation-Layer implementiert. getDataBase (Zeile 220 - 231) gibt eine Datenbank, welche im Feld _databases gespeichert ist an den Aufrufer zurück. Die Funktion addDatabase (Zeile 234 - 240) fügt dem Feld _databases eine neue Datenbank hinzu. Über die Funktion loadDatabase (Zeile 244 - 247) wird eine Liste mit JSON-Texten der Evente einer Datenbank aus der Data-Security-Layer abgerufen und an den DataBaseLoader übergeben. Die Funktion GetKeyFromPasswordDB (Zeile 250 - 266) gibt das Passwort einer Datenbank für die Kommunikation mit der Data-Security-Layer für den Aufrufer zur Verfügung.

DataBaseLoader.cs

Listing A.12: Code der Klasse DataBaseLoader

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Runtime.CompilerServices;
5  using System.Runtime.InteropServices;
6  using System.Text;
7  using System.Text.Json;
8  using System.Text.Json.Nodes;
9  using System.Threading.Tasks;
10 using Presentation_Layer.DataBaseEvents;
11 using Presentation_Layer.Databases;
12 using Presentation_Layer.WorkerChain;
13
14 namespace Presentation_Layer
15 {
16     internal class DataBaseLoader
17     {
18         //properties
```

```
19          private WorkerChain.WorkerChain _workers;
20          //Events for the Database
21          internal event NewDataBaseEvent? CreateNewDataBase;
22          internal event CreateDataEvent? CreateNewData;
23          internal event UpdateDataEvent? UpdateData;
24          internal event RemoveDataEvent? RemoveData;
25
26          //Constructor
27          public DataBaseLoader(WorkerChain.WorkerChain workers)
28          {
29          _workers = workers;
30          CreateNewDataBase += _workers.execute;
31          CreateNewData += _workers.execute;
32          UpdateData += _workers.execute;
33          RemoveData += _workers.execute;
34          }
35
36          //Delegates for the DataBaseEvents
37          public delegate void NewDataBaseEvent(object sender,
                NewDataBaseEventArgs e);
38          public delegate void CreateDataEvent(object sender,
                CreateDataEventArgs e);
39          public delegate void UpdateDataEvent(object sender,
                UpdateDataEventArgs e);
40          public delegate void RemoveDataEvent(object sender,
                RemoveDataEventArgs e);
41
42          //Sorting the loaded Events
43          internal void ReloadDataBase(List<string> eventlist)
44          {
45              foreach(string EventArgsJson in eventlist)
46              {
47                  var JSONDom = JsonSerializer.Deserialize<JsonObject>(
                        EventArgsJson);
48                  string eventType = (string)JSONDom["Eventtype"];
49                  switch (eventType)
50                  {
51                      case "CreateData":
52                          ReloadNewData(JSONDom);
53                          break;
54                      case "RemoveData":
55                          ReloadRemoveData(JSONDom);
56                          break;
57                      case "NewDataBase":
58                          ReloadDataBase(JSONDom);
```

```
59                      break;
60                  case "UpdateData":
61                      ReloadUpdateData(JSONDom);
62                      break;
63                  default:
64                      throw new Exception("this is not an valid eventtype
                            ");
65              }
66          }
67      }
68
69      //rebuild the NewDataBase Command
70      private void ReloadDataBase(JsonObject JSON)
71      {
72          string? Databasename = (string?)JSON["DatabaseName"];
73          string secretKey;
74          if (Databasename == "PasswordDB")
75          {
76              secretKey = "Passwords";
77          }
78          else
79          {
80              secretKey = DataPresentationLayer.Instance.
                    GetKeyFromPasswordDB(Databasename);
81          }
82          DataPresentationLayer.DataBasetype? DatabaseType = null;
83
84          if ((string)JSON["Databasetype"] == "KVDB")
85          {
86              DatabaseType = DataPresentationLayer.DataBasetype.KVDB;
87          }else if ((string)JSON["Databasetype"] == "Accounting")
88          {
89              DatabaseType = DataPresentationLayer.DataBasetype.
                    Accounting;
90          }
91          List<string> fields = JsonSerializer.Deserialize<List<string>>(
                JSON["Fields"]);
92          List<string> optionalFields = JsonSerializer.Deserialize<List<
                string>>(JSON["OptionalFields"]);
93          bool isreloaded = true;
94          NewDataBaseEventArgs e = new NewDataBaseEventArgs(Databasename,
                 secretKey, (DataPresentationLayer.DataBasetype)DatabaseType
                , fields, optionalFields, isreloaded);
95          CreateNewDataBase(this, e);
96      }
```

```
97
98          //rebuild the CreateData Command
99          private void ReloadNewData(JsonObject JSON)
100         {
101             Dictionary<string, string> data = JsonSerializer.Deserialize<
                    Dictionary<string, string>>(JSON["Data"]);
102             IDatabase Database = DataPresentationLayer.Instance.getDataBase
                    ((string)JSON["DataBaseName"]);
103             bool isreloaded = true;
104             string key = Database.GetActualKey();
105             CreateDataEventArgs e = new CreateDataEventArgs(data, key,
                    Database, isreloaded);
106             CreateNewData(this, e);
107         }
108
109         //rebuild the UpdateData Command
110         private void ReloadUpdateData(JsonObject JSON)
111         {
112             Dictionary<string, string> data = JsonSerializer.Deserialize<
                    Dictionary<string, string>>(JSON["Data"]);
113             string key = (string)JSON["Key"];
114             IDatabase database = DataPresentationLayer.Instance.getDataBase
                    ((string)JSON["DataBaseName"]);
115             bool isreloaded = true;
116             UpdateDataEventArgs e = new UpdateDataEventArgs(data, key,
                    database, isreloaded);
117             UpdateData(this, e);
118         }
119
120         //rebuild the RemoveData Command
121         private void ReloadRemoveData(JsonObject JSON)
122         {
123             string key = (string)JSON["Key"];
124             IDatabase database = DataPresentationLayer.Instance.getDataBase
                    ((string)JSON["DataBaseName"]);
125             bool isreloaded = true;
126             RemoveDataEventArgs e = new RemoveDataEventArgs(key, database,
                    isreloaded);
127             RemoveData(this, e);
128         }
129     }
130 }
```

In Listing A.12 ist die Klasse DataBaseLoader dargestellt. Diese ist für das Laden einer Datenbank aus der Data-Security-Layer zuständig. Hierfür werden wir bei der DataPre-

sentationLayer eine WorkerChain (Zeile 19) geladen sowie die selben Events definiert (Zeile 21 - 24). Allen Events wird die execute-Methode der WorkerChain als Eventhandler zugewiesen (Zeile 27 - 34).
In der Funktion ReloadDataBase (Zeile 43 - 66) werden die Befehle anhand des Feldes eventType den verschiedenen Funktionen zum Wiederherstellen der Event-Argumente übergeben, welche dann die entsprechenden Events auslösen (Zeile 70 - 129).

A.1.2.2 Namespace: Presentation_Layer.DataBaseEvents

Die DataBaseEvents beinhalten die verschiedenen Informationen, welche für die Ausführung der Operationen auf den Datenbanken benötigt werden. Sie werden durch die DataPresentationLayer-Klasse erstellt und dann an das entsprechende Event übergeben. Als Eventhandler dient hierbei die WorkerChain. Es gibt vier EventArgument-Klassen: CreateDataEventArgs für das hinzufügen von neuen Daten zu der Datenbank, NewDataBaseEventArgs für das Erstellen einer neuen Datenbank, RemoveDataEventArgs für das Entfernen von Daten aus der In-Memory-Datenbank und UpdateDataEventArgs für das überschreiben eines Wertes in der Datenbank. Für alle Klassen ist nachfolgend der Code abgedruckt und anschließend die Funktionsweise erläutert.

CreateDataEventArgs.cs

Listing A.13: Code der Klasse CreateDataEventArgs

```
1  using Presentation_Layer.Databases;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace Presentation_Layer.DataBaseEvents
9  {
10     internal class CreateDataEventArgs:EventArgs
11     {
12         //Eventtype for determing which event has to be created when
               loading from file
13         private string _eventtype = "CreateData";
14         //Database to which the Data should be added
15         private IDatabase _database;
16         //Data to add
17         private Dictionary<string, string> _data;
18         //Key for the new Data
19         private string _key;
```

```
20          //marker if the Command is created or reloaded
21          private bool _isreloaded;
22
23          //Public Properties for the JSONSerializer
24          public string Eventtype
25          {
26              get
27              {
28                  return _eventtype;
29              }
30          }
31          public string Key
32          {
33              get
34              {
35                  return _key;
36              }
37          }
38
39          public Dictionary<string, string> Data
40          {
41              get
42              {
43                  return _data;
44              }
45          }
46
47          public string DataBaseName
48          {
49              get
50              {
51                  return _database.Name;
52              }
53          }
54
55          //internal Properties for the Project
56          internal IDatabase Database
57          {
58              get
59              {
60                  return _database;
61              }
62          }
63
64          internal bool Isreloaded
```

```
65          {
66              get
67              {
68                  return _isreloaded;
69              }
70          }
71
72          //Constructor for the EventArgs
73          public CreateDataEventArgs(Dictionary<string, string> data, string
                key, IDatabase database, bool isreloaded)
74          {
75              _data = data;
76              _key = key;
77              _database = database;
78              _isreloaded = isreloaded;
79          }
80      }
81  }
```

Die Klasse CreateDataEventArgs, deren Code in Listing A.13 abgebildet ist, beinhaltet als gespeicherte Daten die Bezeichnung des Eventtyps (Zeile 13), um dieses später beim Lesen aus einer Datei zuordnen zu können. Des Weiteren sind als Werte die Datenbank (Zeile 15), auf der die Operation ausgeführt werden soll, die neuen Daten (Zeile 17), der Schlüssel unter dem die Daten gespeichert werden sollen (Zeile 19) und ein Marker, ob der Befehl neu erzeugt wurde oder aus einer Datei geladen wurde (Zeile 21), gespeichert. Ein weiterer Bestandteil der Klasse sind die öffentlich Eigenschaften (Zeile 24 - 53), welche für den JSON-Serializer benötigt werden. Die Event-Argumente werden nach der Verarbeitung in der Data-Security-Layer als Werte gespeichert und können aus dieser abgefragt werden, um die Befehle erneut durchzuführen. Die internen Eigenschaften (Zeile 56 - 70) sind für die Nutzung durch die Worker der WorkerChain bestimmt. Der letzte Bestandteil ist ein Konstrukor (Zeile 73 - 79) für die Erstellung von Objekten aus der Klasse.

NewDataBaseEventArgs.cs

Listing A.14: Code der Klasse NewDataBaseEventArgs

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Runtime.CompilerServices;
5 using System.Text;
6 using System.Threading.Tasks;
7 using Presentation_Layer;
```

```
using Presentation_Layer.Databases;

namespace Presentation_Layer.DataBaseEvents
{
    internal class NewDataBaseEventArgs : EventArgs
    {
        //Eventtype for determing which event has to be created when
            loading from file
        private string _eventtype = "NewDataBase";
        //Name of the new Database
        private string _databaseName;
        //Password of the new Database
        private string _secretKey;
        //type of the new Database
        DataPresentationLayer.DataBasetype _databasetype;
        //fields for the new Database, additional fields when it is an
            AccountingDB
        private List<string> _fields;
        //list with fields, that are optional
        private List<string> _optionalFields;
        //The Database, when ist is created
        private IDatabase? _createdDatabase;
        //marker if the event is newly created or laoded from file
        private bool _isreloaded;

        //Properties for JSON-Serialization
        public string Eventtype
        {
            get
            {
                return _eventtype;
            }
        }
        public string DatabaseName
        {
            get
            {
                return _databaseName;
            }
        }
        public string SecretKey
        {
            get
            {
                return _secretKey;
```

```
51              }
52          }
53          public string Databasetype
54          {
55              get
56              {
57                  return _databasetype.ToString();
58              }
59          }
60
61          public List<string> Fields
62          {
63              get
64              {
65                  return _fields;
66              }
67          }
68
69          public List<string> OptionalFields
70          {
71              get
72              {
73                  return _optionalFields;
74              }
75          }
76
77          //internal Properties
78          internal IDatabase CreatedDatabase
79          {
80              get
81              {
82                  if(_createdDatabase == null)
83                  {
84                      throw new ArgumentNullException("Es wurde keine
                            Datenbank erstellt");
85                  }
86                  return _createdDatabase;
87              }
88          }
89
90          internal DataPresentationLayer.DataBasetype DatabaseType
91          {
92              get
93              {
94                  return _databasetype;
```

```
95              }
96          }
97
98          internal bool Isreloaded
99          {
100             get
101             {
102                 return _isreloaded;
103             }
104         }
105
106         //Constructor for Event
107         public NewDataBaseEventArgs(string name, string secretKey,
                DataPresentationLayer.DataBasetype dataBasetype, List<string>
                fields, List<string> optionalFields, bool isreloaded)
108         {
109             _databaseName = name;
110             _secretKey = secretKey;
111             _databasetype = dataBasetype;
112             _fields = fields;
113             _optionalFields = optionalFields;
114             _isreloaded = isreloaded;
115         }
116
117         //function to give the Database to the next Levels in the
                WorkerChain
118         internal void setDataBase(IDatabase createdDatabase)
119         {
120             _createdDatabase = createdDatabase;
121         }
122     }
123 }
```

Der Aufbau der Klasse NewDataBaseEventArgs, Code in Listing A.14, ähnelt der Klasse CreateDataEventArgs. Unterschiedlich sind die gespeicherten Werte. Gleich sind die gespeicherten Werte Eventtyp (Zeile 15), die betroffene Datenbank (Zeile 27), die hier erst erstellt und später gesetzt wird, und der Marker, ob die Daten geladen wurden oder nicht (Zeile 29). Neue Angaben sind der Name der neuen Datenbank (Zeile 17), das Passwort der Datenbank (Zeile 19), der Datenbanktyp (Zeile 21), die Felder der Datenbank beziehungsweise die zusätzlichen Felder bei einer AccountingDB (Zeile 23), sowie die Angabe, welche der Felder optional sind (Zeile 25).

Ebenfalls haben die beiden Klassen die Unterscheidung in öffentliche (Zeile 32 - 75) und interne (Zeile 78 - 104) Eigenschaften gemeinsam und das zur Verfügung stellen eines

Konstruktors (Zeile 107 - 115). Die Klasse NewDataBaseEventArgs besitzt zusätzlich noch eine Funktion um den Wert der gespeicherten Datenbank zu setzen (Zeile 118 - 121).

RemoveDataEventArgs.cs

Listing A.15: Code der Klasse RemoveDataEventArgs

```
1  using Presentation_Layer.Databases;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace Presentation_Layer.DataBaseEvents
9  {
10     internal class RemoveDataEventArgs:EventArgs
11     {
12         //Eventtype for determing which event has to be created when
               loading from file
13         private string _eventtype = "RemoveData";
14         //Database the data should be removed from
15         private IDatabase _database;
16         //Key that should be deleted in the database
17         private string _key;
18         //marker if the event is newly created or laoded from file
19         private bool _isreloaded;
20
21         //public Properties for JSONSerializer
22         public string Eventtype
23         {
24             get
25             {
26                 return _eventtype;
27             }
28         }
29         public string DataBaseName
30         {
31             get
32             {
33                 return _database.Name;
34             }
35         }
36
37         public string Key
```

```
38          {
39              get
40              {
41                  return _key;
42              }
43          }
44
45          //Constructor
46          public RemoveDataEventArgs(string key, IDatabase database, bool
                isreloaded)
47          {
48              _database = database;
49              _key = key;
50              _isreloaded = isreloaded;
51          }
52
53          //internal Properties
54          internal IDatabase Database
55          {
56              get
57              {
58                  return _database;
59              }
60          }
61
62          internal bool Isreloaded
63          {
64              get
65              {
66                  return _isreloaded;
67              }
68          }
69      }
70  }
```

Wie die beiden zuvor genannten Klassen besteht auch die Klasse RemoveDataEventArgs, deren Code in Listing A.15 abgebildet ist, aus den gespeicherten Werten für den Eventtyp (Zeile 13), die Datenbank (Zeile 15) und dem Marker, ob der Befehl geladen oder erzeugt wurde (Zeile 19). Ergänzend hierzu ist noch der Wert für den Key, dessen Wert gelöscht werden soll vorhanden (Zeile 17). Des weiteren gibt es einen Konstruktor (Zeile 46 - 51) , sowie interne (Zeile 54 - 70) und öffentliche Eigenschaften (Zeile 22 - 43).

UpdateDataEventArgs.cs

Listing A.16: Code der Klasse UpdateDataEventArgs

```
using Presentation_Layer.Databases;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Presentation_Layer.DataBaseEvents
{
    internal class UpdateDataEventArgs:EventArgs
    {
        //Eventtype for determing which event has to be created when
           loading from file
        private string _eventtype = "UpdateData";
        //Database where the Data should be updated
        private IDatabase _database;
        //new Data
        private Dictionary<string, string> _data;
        //Key for the data that should be replaced
        private string _key;
        //marker if the event is newly created or laoded from file
        private bool _isreloaded;

        //Public Properties for the JSONSerializer
        public string Eventtype
        {
            get
            {
                return _eventtype;
            }
        }
        public string Key
        {
            get
            {
                return _key;
            }
        }

        public Dictionary<string, string> Data
        {
            get
            {
                return _data;
```

```
44              }
45          }
46
47          public string DataBaseName
48          {
49              get
50              {
51                  return _database.Name;
52              }
53          }
54
55          //internal Properties for the Projekt
56          internal IDatabase Database
57          {
58              get
59              {
60                  return _database;
61              }
62          }
63
64          internal bool Isreloaded
65          {
66              get
67              {
68                  return _isreloaded;
69              }
70          }
71
72          //Constructor
73          public UpdateDataEventArgs(Dictionary<string, string> data, string
                key, IDatabase database, bool isreloaded)
74          {
75              _data = data;
76              _key = key;
77              _database = database;
78              _isreloaded = isreloaded;
79          }
80      }
81  }
```

Auch die Klasse UpdateDataEventArgs, die in Listing A.16 dargestellt ist, besteht aus den selben Bestandteilen, wie die anderen Klassen des Namespaces. Neben den immer gespeicherten Werten, speichert diese noch die neuen Daten (Zeile 17) und den Schlüssel dessen Daten ersetzt werden sollen (Zeile 19).

A.1.2.3 Namespace: Presentation_Layer.Databases

Der Namespace *Presentation_Layer.Databases* beinhaltet die Umsetzung der In-Memory-Datenbanken. Hierzu bildet das Interface IDatabase die gemeinsamen Grundlagen der beiden Klassen KVDB und AccountingDB. KVDB bildet eine Key-Value-Datenbank und AccountingDB eine Datenbank für die Verwendung in Buchhaltungsprogrammen. Die Klasse Account ist eine Hilfsklasse für die AccountingDB.
Nachfolgend werden die Klassen einzeln vorgestellt:

Account.cs

Listing A.17: Code der Klasse Account

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Presentation_Layer.Databases
{
    internal class Account
    {
        //List with entrys on the debit side
        private List<string> _debitEntrys = new List<string>();
        //List with entrys on the credit side
        private List<string> _creditEntrys = new List<string>();
        //Database the Account is a part of, needed to get the
            journalentrys, when the Account is readed
        private AccountingDB _db;

        //Constructor
        internal Account(AccountingDB db)
        {
            _db = db;
        }

        //function to add entry on the credit side
        internal void addCreditEntry(string key)
        {
            _creditEntrys.Add(key);
        }

        //function to add entry on the debit side
        internal void addDebitEntry(string key)
        {
```

```
33              _debitEntrys.Add(key);
34          }
35
36          //function to remoce entry on the credit side
37          internal void removeCreditEntry(string key)
38          {
39              _creditEntrys.Remove(key);
40          }
41
42          //function to remoce entry on the debit side
43          internal void removeDebitEntry(string key)
44          {
45              _debitEntrys.Remove(key);
46          }
47
48          //function to read the account. Returns a Dictionary with two Lists
                , which are representing the debit and the credit side
49          internal Dictionary<string, List<Dictionary<string, string>>>
                ReadAccount()
50          {
51              //getting a List with the journal entrys on the Credit side
52              List<Dictionary<string, string>> CreditEntrys = new List<
                    Dictionary<string, string>>();
53              foreach (string key in _creditEntrys)
54              {
55                  CreditEntrys.Add(_db.ReadKeyBased(key));
56              }
57              //getting a List with the journal entrys on the Debit side
58              List<Dictionary<string, string>> DebitEntrys = new List<
                    Dictionary<string, string>>();
59              foreach (string key in _debitEntrys)
60              {
61                  DebitEntrys.Add(_db.ReadKeyBased(key));
62              }
63              //building the Dicitionary with both Lists
64              return new Dictionary<string, List<Dictionary<string, string>>>
                    ()
65              {
66                  { "CreditEntrys", CreditEntrys },
67                  {"DebitEntrys", DebitEntrys }
68              };
69          }
70      }
71  }
```

In Listing A.17 ist der Code der Klasse Account abgebildet. Diese ist eine Hilfsklasse für die AccountingDB. Nachgebildet wird ein Konto indem die Schlüssel von Buchungen im Journal, welches in der Klasse AccountingDB enthalten ist auf Soll und Haben aufgeteilt wird. Hierbei besteht die Klasse aus den nachfolgenden Bestandteilen:

Liste mit Sollbuchungen (Zeile 12)
: Beinhaltet die Buchungen, bei denen das Konto als Sollkonto fungiert.

Liste mit Habenbuchungen (Zeile 14)
: Beinhaltet die Buchungen, bei denen das Konto als Habenkonto fungiert.

Datenbank deren Teil das Konto ist (Zeile 16)
: Benötigt um die Buchungen aus dem Journal abrufen zu können für die Ausgabe des Kontos.

Konstruktur (Zeile 19)
: Übergabe der Datenbank an die Klasse.

Funktionen zum Hinzufügen von Soll- (Zeile 31) und Habenbuchungen (Zeile 25)
: Neue Buchungen für das Konto.

Funktionen zum Löschen von Soll- (Zeile 43) und Habenbuchungen (Zeile 37)
: Buchungen bei der Löschung aus dem Journal auch auf dem Konto entfernen.

Ausgeben des Kontos (Zeile 48 - 69)
: Ausgaben der Buchungen auf dem Konto indem zuerst die Habenbuchungen zu einer Liste und anschließend die Sollbuchungen zu einer Liste zusammengefasst werden. Hierzu werden aus der Datenbank die Buchungen mit den gespeicherten Schlüsseln abgefragt und das Ergebnis gespeichert. Die beiden Listen werden anschließend in ein Dictionary zusammengefasst und an den Aufrufer übergeben.

AccountingDB

Listing A.18: Code der Klasse AccountingDB

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using Presentation_Layer.Databases;
7
8 namespace Presentation_Layer.Databases
9 {
```

```
10      internal class AccountingDB:IDatabase
11      {
12          //additional fields to the set fields for accountingDBs
13          private List<string> _additionalFields;
14          //List with the fields which are optional
15          private List<string> _optionalFields;
16          //name of the DB
17          private string _name;
18          //journal with all entrys to the DB
19          private Dictionary<string, Dictionary<string, string>> _journal;
20          //Datapattern for the user
21          private Dictionary<string, string> _dataPattern;
22          //actual key
23          private int _key = 0;
24          //List with the fields, that must be used in all AccountingDBs
25          private static List<string> AccountingDBFields = new List<string>()
26          {
27              "Amount",
28              "DebitAccount",
29              "CreditAccount",
30              "Date"
31          };
32          //Dictionary for the Account class
33          Dictionary<string, Account> _accounts;
34
35          //Constructor
36          public AccountingDB(string name, List<string> additionalFields,
                List<string> optionalFields)
37          {
38              _name = name;
39              _additionalFields = additionalFields;
40              _optionalFields = optionalFields;
41              _dataPattern = new Dictionary<string, string>();
42              List<string> Fields = AccountingDBFields;
43              Fields.AddRange(_additionalFields);
44              foreach (string field in Fields)
45              {
46                  _dataPattern.Add(field, "");
47              }
48              _journal = new Dictionary<string, Dictionary<string, string>>()
                    ;
49              _accounts = new Dictionary<string, Account>();
50          }
51
52          //Public Property for implementing the Interface IDatabase
```

```
53          public string Name
54          {
55              get
56              {
57                  return _name;
58              }
59          }
60
61          //Function to Add data to the Database
62          public void AddData(string key, Dictionary<string, string> value)
63          {
64              _journal.Add(key, value);
65              _key++;
66              //checking if the Account is already existing
67              if (_accounts.ContainsKey(value["CreditAccount"]))
68              {
69                  _accounts[value["CreditAccount"]].addCreditEntry(key);
70              }
71              else
72              {
73                  _accounts[value["CreditAccount"]] = new Account(this);
74                  _accounts[value["CreditAccount"]].addCreditEntry(key);
75              }
76              //checking if the Account is already existing
77              if (_accounts.ContainsKey(value["DebitAccount"]))
78              {
79                  _accounts[value["DebitAccount"]].addDebitEntry(key);
80              }
81              else
82              {
83                  _accounts[value["DebitAccount"]] = new Account(this);
84                  _accounts[value["DebitAccount"]].addDebitEntry(key);
85              }
86          }
87
88          //Function to update data in the Database
89          public void UpdateData(string key, Dictionary<string, string> value
                )
90          {
91              //checking if the key is valid
92              if (_journal.ContainsKey(key))
93              {
94                  _journal[key] = value;
95              }
96              else
```

```
97              {
98                  throw new ArgumentOutOfRangeException("There is no such key
                        in the Database");
99              }
100         }
101
102         //Function to Delete data in the Database
103         public void DeleteData(string key)
104         {
105             //Fetching data to get the Accounts
106             Dictionary<string, string> data = _journal[key];
107             //deleting from Accounts
108             Account CreditAccount = _accounts[data["CreditAccount"]];
109             Account DebitAccount = _accounts[data["DebitAccount"]];
110             CreditAccount.removeCreditEntry(key);
111             DebitAccount.removeDebitEntry(key);
112             //deleting from journal
113             _journal.Remove(key);
114         }
115
116         //Function to get the dataPattern of the Database
117         public Dictionary<string, object> GetDataPattern()
118         {
119             return new Dictionary<string, object>()
120             {
121                 {"optionalFields", _optionalFields},
122                 {"dataDictionary", _dataPattern }
123             };
124         }
125
126         //Read the JournalData from one Key
127         public Dictionary<string, string> ReadKeyBased(string Key)
128         {
129             return _journal[Key];
130         }
131
132         //read the full journal
133         public List<Dictionary<string, string>> ReadAll()
134         {
135             List<Dictionary<string, string>> result = new List<Dictionary<
                    string, string>>();
136             foreach (string key in _journal.Keys)
137             {
138                 result.Add(_journal[key]);
139             }
```

```
140             return result;
141         }
142 
143         //Read the Data from one Account
144         internal Dictionary<string, List<Dictionary<string,string>>>
                ReadAccount(string Account)
145         {
146             return _accounts[Account].ReadAccount();
147         }
148 
149         //returns the actual Key
150         public string GetActualKey()
151         {
152             return _key.ToString();
153         }
154     }
155 }
```

Im Listing A.18 ist die Klasse AccountingDB dargestellt. Diese stellt eine der beiden im Prototyp enthaltenen Datenbankarten da. Die Klasse AccountingDB besitzt die nachfolgenden Felder: zusätzliche Felder (Zeile 13), die Felder, welche zusätzlich zu den immer enthaltenen Feldern (Zeile 25 - 31) gespeichert werden, optionale Felder (Zeile 15), Liste mit den Feldern, welche nur optional gefühlt werden müssen, Name der Datenbank (Zeile 17), Journal in Form eines Dicitonarys, welches alle Buchungen speichert (Zeile 19), dataPattern (Zeile 21), Dictionary mit den Feldern, die der Nutzer füllen kann, dem aktuellen Key (Zeile 23) sowie ein Dictionary mit den Konten der Datenbank (Zeile 33). Neben dem Konstruktor der Klasse (Zeile 36 - 50), werden die nachfolgenden Funktionen zur Verfügung gestellt:

AddData (Zeile 62 - 86)
: Fügt die übergebenen Daten in der Datenbank unter den angegeben Key ein und übergibt den Key an die entsprechenden Konten.

UpdateDate (Zeile 89 - 100)
: Ersetzt die unter dem übergebenen Key gespeicherten Daten mit den beim Aufruf übergebenen Daten.

DeleteData (Zeile 103 - 114)
: Löscht die Daten, die unter dem übergebenen Key gespeichert sind, und ebenfalls die Schlüssel in dem betroffenen Konten.

GetDataPattern (Zeile 117 - 124)
: Stellt dem Aufrufer das DataPattern der Datenbank zur Verfügung. Dies besteht aus

einem Dicitionary, welches alle Felder der Datenbank als Keys beinhaltet und der Liste der optionalen Felder.

ReadKeyBased (Zeile 127 - 130)
: Übergibt den unter einen bestimmten Key gespeicherten Daten an den Aufrufer.

ReadAll (Zeile 133 - 141)
: Übergibt alle in der Datenbank gespeicherten Daten an den Aufrufer.

ReadAccount (Zeile 144 - 147)
: Gibt die Buchungen zurück, die auf einem bestimmten Konto gespeichert sind, in Form eines Dictionarys, welches je ein Dictionary mit den den Buchungen auf der Soll- und eines für die Buchungen auf der Habenseite beinhaltet.

GetActualKey (Zeile 150 - 153)
: Liefert dem Aufrufer den nächsten freien Key in der Datenbank.

IDatabase.cs

Listing A.19: Code des Interfaces IDatabase

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Presentation_Layer.Databases
8  {
9      internal interface IDatabase
10     {
11         public string Name { get; }
12
13         public void AddData(string key, Dictionary<string,string> value);
14         public void UpdateData(string key, Dictionary<string, string> value
               );
15         public void DeleteData(string key);
16         public Dictionary<string, object> GetDataPattern();
17         public Dictionary<string,string> ReadKeyBased(string key);
18         public List<Dictionary<string, string>> ReadAll();
19         public string GetActualKey();
20     }
21 }
```

Das Interface IDatabase, Code in Listing A.19, stellt die gemeinsame Grundlage aller Datenbankarten da und besteht aus den verschiedenen Funktionen welche universal für

die Datenbanken verfügbar sein sollen. Dies sind das Hinzufügen von Daten (Zeile 13), das Erneuern von Daten (Zeile 14), das Löschen von Daten (Zeile 15), das Ausgeben eines DataPattern, welches durch den Anwender gefüllt werden kann (Zeile 16), Funktionen zum Key-basierten Lesen aus der Datenbank (Zeile 17) und Lesen der gesamten Daten der Datenbank (Zeile 18) sowie die Funktion zum Erhalt des aktuellen Keys (Zeile 19).

KVDB.cs

Listing A.20: Code der Klasse KVDB

```
1  using System;
2  using System.Collections.Generic;
3  using System.Globalization;
4  using System.Linq;
5  using System.Security.Cryptography.X509Certificates;
6  using System.Text;
7  using System.Threading.Tasks;
8
9  namespace Presentation_Layer.Databases
10 {
11     internal class KVDB:IDatabase
12     {
13         //list with the datafields in the DB
14         private List<string> _fields;
15         //list with the fields which are optional
16         private List<string> _opionalfields;
17         //name of the db
18         private string _name;
19         //Data of the DB
20         private Dictionary<string, Dictionary<string, string>> _data;
21         //Datapattern of the DB
22         private Dictionary<string, string> _dataPattern;
23         //actual key of the Database
24         private int _key = 0;
25
26         //Constructor
27         public KVDB(string name, List<string> fields, List<string>
               opionalfields)
28         {
29             _name = name;
30             _fields = fields;
31             _opionalfields = opionalfields;
32             _dataPattern = new Dictionary<string, string>();
33             foreach(string field in _fields)
34             {
35                 _dataPattern.Add(field, "");
```

```
36              }
37              _data = new Dictionary<string, Dictionary<string, string>>();
38          }
39
40          //Public Property for implementing the Interface IDatabase
41          public string Name
42          {
43              get { return _name; }
44          }
45
46          //Function to add Data to the DB
47          public void AddData(string key, Dictionary<string, string> value)
48          {
49              _data.Add(key, value);
50              _key++;
51          }
52
53          //Function to Update existing data in the DB
54          public void UpdateData(string key, Dictionary<string, string> value
                )
55          {
56              //Checking if the key is in the Database
57              if(_data.ContainsKey(key))
58              {
59                  _data[key] = value;
60              }
61              else
62              {
63                  throw new ArgumentOutOfRangeException("There is no such key
                        in the Database");
64              }
65
66          }
67
68          //Function for deleting data in the DB
69          public void DeleteData(string key)
70          {
71              _data.Remove(key);
72          }
73
74          //Function to get the dataPattern of the Database
75          public Dictionary<string, object> GetDataPattern()
76          {
77              return new Dictionary<string, object>()
78              {
```

```
79                  { "optionalFields", _opionalfields },
80                  {"dataDictionary", _dataPattern }
81              };
82
83          }
84
85          //Read the data from one Key
86          public Dictionary<string, string> ReadKeyBased(string key)
87          {
88              return _data[key];
89          }
90
91          //Read the full data
92          public List<Dictionary<string, string>> ReadAll()
93          {
94              List<Dictionary<string, string>> result = new List<Dictionary<
                    string, string>> ();
95              foreach (string key in _data.Keys)
96              {
97                  result.Add(_data[key]);
98              }
99              return result;
100         }
101
102         //returns the actual Key
103         public string GetActualKey()
104         {
105             return _key.ToString();
106         }
107     }
108 }
```

Die Klasse KVDB stellt eine der beiden im Prototypen enthaltenen Datenbankarten dar. Es handelt sich hierbei um eine einfache Key-Value-Datenbank. Der Code der Klasse ist in Listing A.20 abgedruckt. Die Klasse implementiert das Interface IDatabase.
Die Klasse hat folgende Attribute. Eine Liste mit den Feldern für die Daten (Zeile 14), eine Liste mit den optionalen Feldern (Zeile 16), den Namen der Datenbank (Zeile 18), ein Dicitionary, welches Dictionarys mit den Daten enthält (Zeile 20), hier werden die Daten gespeichert, ein Dicitionary, welches das DataPattern der Datenbank speichert (Zeile 22) und den aktuellen Key der Datenbank (Zeile 24).
Neben den Konstrukor der Klasse (Zeile 27 - 38) enthält die Klasse folgende Funktionen:

AddData (Zeile 47 - 51)

Fügt die übergebenen Daten in der Datenbank unter den angegeben Key ein.

UpdateDate (Zeile 54 - 66)
: Ersetzt die unter dem übergebenen Key gespeicherten Daten mit den beim Aufruf übergebenen Daten.

DeleteData (Zeile 69 - 72)
: Löscht die Daten, die unter dem übergebenen Key gespeichert sind.

GetDataPattern (Zeile 75 - 81)
: Stellt dem Aufrufer das DataPattern der Datenbank zur Verfügung. Dies besteht aus einem Dicitionary, welches alle Felder der Datenbank als Keys beinhaltet und der Liste der optionalen Felder.

ReadKeyBased (Zeile 86 - 89)
: Übergibt den unter einen bestimmten Key gespeicherten Daten an den Aufrufer.

ReadAll (Zeile 92 - 100)
: Übergibt alle in der Datenbank gespeicherten Daten an den Aufrufer.

GetActualKey (Zeile 103 - 106)
: Liefert dem Aufrufer den nächsten freien Key in der Datenbank.

A.1.2.4 Namespace: Presentation_Layer.WorkerChain

Die WorkerChain ist eine Zuständigkeitskette und stellt die Verbindung zwischen dem Einstiegspunkt der Ebene in Form der Klasse DataPresentationLayer über die Datenbanken hin zu der Data-Security-Layer, zur Sicherung der Daten, dar. Der Namespace enthält die Definition der WorkerChain selbst, das Interface IWorker als Grundlage der einzelnen Worker-Klassen und die drei Worker-Klassen DataBaseCommandHandler, DataCommandHandler und DSLConnector.

DataBaseCommandHandler.cs

Listing A.21: Code der Klasse DataBaseCommandHandler

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using Presentation_Layer.DataBaseEvents;
7  using Presentation_Layer.Databases;
8
9  namespace Presentation_Layer.WorkerChain
10 {
```

```
11      internal class DataBaseCommandHandler:IWorker
12      {
13          public void execute(EventArgs e)
14          {
15              if(e.GetType() == typeof(NewDataBaseEventArgs))
16              {
17                  NewDataBaseEventArgs newDataBase = e as
                        NewDataBaseEventArgs?? throw new ArgumentNullException("
                        The given Event is null:" + nameof(e));
18                  //Building the new Database depending on the Databasetype
19                  switch (newDataBase.DatabaseType)
20                  {
21                      case DataPresentationLayer.DataBasetype.KVDB:
22                          KVDB Kdb = new KVDB(newDataBase.DatabaseName,
                                newDataBase.Fields, newDataBase.OptionalFields);
23                          newDataBase.setDataBase(Kdb);
24                          break;
25                      case DataPresentationLayer.DataBasetype.Accounting:
26                          AccountingDB Adb = new AccountingDB(newDataBase.
                                DatabaseName, newDataBase.Fields, newDataBase.
                                OptionalFields);
27                          newDataBase.setDataBase(Adb);
28                          break;
29                  }
30              }
31
32          }
33      }
34  }
```

Die Klasse DataBaseCommandHandler, deren Code in Listing A.21 dargestellt ist, bearbeitet die Befehle, welche mit der Datenbank an sich zu tun haben und keinen Bezug zu den eigentlichen Daten haben. Dies ist derzeit nur der Befehl um eine neue Datenbank zu erstellen. Die Funktion execute (Zeile 13) nimmt hierfür die Event-Argumente entgegen und castet diese wenn möglich auf die Klasse NewDataBaseEventArgs (Zeile 15 - 17). Anschließend wird unterschieden, welche Datenbankart erstellt werden soll und diese erstellt (Zeile 19 - 32). Unabhängig der Datenbankart, wird diese an die Klasse DataPresentationLayer übergeben für den späteren Zugriff.

DataCommandHandler.cs

Listing A.22: Code der Klasse DataCommandHandler

```
1  using Presentation_Layer.DataBaseEvents;
2  using Presentation_Layer.Databases;
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Presentation_Layer.WorkerChain
{
    internal class DataCommandHandler:IWorker
    {
        public void execute(EventArgs e)
        {
            if (e == null)
            {
                return;
            }
            //Create new Data in the Database
            else if (e.GetType() == typeof(CreateDataEventArgs))
            {
                CreateDataEventArgs CreateData = (CreateDataEventArgs)e;
                CreateData.Database.AddData(CreateData.Key, CreateData.Data
                    );
                return;
            }
            //Update existing Data in the Database
            else if (e.GetType() == typeof(UpdateDataEventArgs))
            {
                UpdateDataEventArgs UpdateData = (UpdateDataEventArgs)e;
                UpdateData.Database.UpdateData(UpdateData.Key, UpdateData.
                    Data);
                return;
            }
            //Remove Data from the Database
            else if (e.GetType() == typeof(RemoveDataEventArgs))
            {
                RemoveDataEventArgs RemoveData = (RemoveDataEventArgs)e;
                RemoveData.Database.DeleteData(RemoveData.Key);
                return;
            }
        }
    }
}
```

Das Listing A.22 bildet die Klasse DataCommandHandler ab. Diese ist für die Ausführung der Befehle, welche mit den Daten in der Datenbank zu tun haben zuständig.

Hierfür steht ebenfalls die Methode execute (Zeile 13) zur Verfügung, welche die Event-Argumente entgegennimmt. Anschließend wird unterschieden, welche Art von Event übergeben wurde (Zeile 20 - 40) und die entsprechende Methode mit Ihren Parametern auf der Datenbank aufgerufen. Die entsprechende Datenbank ist in den Argumenten enthalten.

DSLConnector.cs

Listing A.23: Code der Klasse DSLConnector

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Text.Json;
6  using System.Threading.Tasks;
7  using Data_Security_Layer;
8  using Presentation_Layer.DataBaseEvents;
9  using Presentation_Layer.Databases;
10
11 namespace Presentation_Layer.WorkerChain
12 {
13     internal class DSLConnector: IWorker
14     {
15         //Instance of the Databasesystem
16         private DataSecurityLayer DSL;
17
18         //Constructor
19         internal DSLConnector(string DatabasesystemPath, string
               DatabaseSystemName, string SecretKey)
20         {
21             DSL = DataSecurityLayer.getDatabaseSystem(DatabasesystemPath,
                   DatabaseSystemName, SecretKey);
22         }
23
24         //Executing the given Command
25         public void execute(EventArgs e)
26         {
27             //filter out  Nulls
28             if (e == null)
29             {
30                 throw new ArgumentNullException("e");
31             }
32             //Handling new Databases
33             else if (e.GetType() == typeof(NewDataBaseEventArgs))
34             {
35                 NewDataBaseEventArgs newDataBase = (NewDataBaseEventArgs)e;
```

```
36                  if (newDataBase.Isreloaded)
37                  {
38                      DataPresentationLayer.Instance.addDatabase(newDataBase.
                            CreatedDatabase);
39                      return;
40                  }
41                  DSL.newDatabase(newDataBase.DatabaseName, newDataBase.
                        SecretKey);
42                  DataPresentationLayer.Instance.addDatabase(newDataBase.
                        CreatedDatabase);
43                  string JSON = JsonSerializer.Serialize(newDataBase);
44                  DSL.AddData(newDataBase.DatabaseName, newDataBase.SecretKey
                        , JSON);
45              }
46              //Handling new Data
47              else if (e.GetType() == typeof(CreateDataEventArgs))
48              {
49                  CreateDataEventArgs CreateData = (CreateDataEventArgs)e;
50                  if (CreateData.Isreloaded)
51                  {
52                      return;
53                  }
54                  string JSON = JsonSerializer.Serialize(CreateData);
55                  string secretKey;
56                  if (CreateData.DataBaseName == "PasswordDB")
57                  {
58                      secretKey = "Passwords";
59                  }
60                  else
61                  {
62                      secretKey = DataPresentationLayer.Instance.
                            GetKeyFromPasswordDB(CreateData.DataBaseName);
63                  }
64                  DSL.AddData(CreateData.Database.Name, secretKey, JSON);
65              }
66              //Handling Update Data
67              else if(e.GetType() == typeof(UpdateDataEventArgs))
68              {
69                  UpdateDataEventArgs UpdateData = (UpdateDataEventArgs)e;
70                  if (UpdateData.Isreloaded)
71                  {
72                      return;
73                  }
74                  string JSON = JsonSerializer.Serialize(UpdateData);
75                  string secretKey;
```

```
76                  if (UpdateData.DataBaseName == "PasswordDB")
77                  {
78                      secretKey = "Passwords";
79                  }
80                  else
81                  {
82                      secretKey = DataPresentationLayer.Instance.
                            GetKeyFromPasswordDB(UpdateData.DataBaseName);
83                  }
84                  DSL.AddData(UpdateData.Database.Name, secretKey, JSON);
85              }
86              //Handling Remove Commands
87              else if (e.GetType() == typeof(RemoveDataEventArgs))
88              {
89                  RemoveDataEventArgs RemoveData = (RemoveDataEventArgs)e;
90                  if (RemoveData.Isreloaded)
91                  {
92                      return;
93                  }
94                  string JSON = JsonSerializer.Serialize(RemoveData);
95                  string secretKey;
96                  if (RemoveData.DataBaseName == "PasswordDB")
97                  {
98                      secretKey = "Passwords";
99                  }
100                 else
101                 {
102                     secretKey = DataPresentationLayer.Instance.
                            GetKeyFromPasswordDB(RemoveData.DataBaseName);
103                 }
104                 DSL.AddData(RemoveData.Database.Name, secretKey, JSON);
105             }
106         }
107
108         //Loading data for the DataBaseLoader
109         internal List<string> loadData(string dataBaseName)
110         {
111             string secretKey;
112             if (dataBaseName == "PasswordDB")
113             {
114                 secretKey = "Passwords";
115             }
116             else
117             {
```

```
118                 secretKey = DataPresentationLayer.Instance.
                        GetKeyFromPasswordDB(dataBaseName);
119             }
120             return DSL.getData(dataBaseName, secretKey);
121         }
122     }
123 }
```

Die Klasse DSLConnector, deren Code in Listing A.23 abgebildet ist, ist für die Verbindung zwischen der Data-Presentation-Layer und der Data-Security-Layer verantwortlich. Hierzu wird eine Instance der Klasse DataSecurityLayer gespeichert (Zeile 16). Diese wird bei Aufruf des Konstruktors erstellt (Zeile 19 - 22). Wie auch die anderen Worker-Klassen implementiert auch die Klasse DSLConnector die Methode execute (Zeile 25), dieser werden ebenfalls die Argumente des Events, welches durch die WorkerChain bearbeitet wird übergeben.

Der DSLConnector filtert nun Event-Argumente raus, welche null sind (Zeile 28). Anschließend werden die Event-Argumente nach dem jeweiligen Typ sortiert. Alle Event-Argumente werden in der Data-Security-Layer gespeichert, wenn diese nicht aus dieser geladen wurden. Bei Argumenten des Typs NewDataBaseEventArgs wird zusätzlich noch eine neue Chain in der Data-Security-Layer angelegt. (Zeile 33 - 106)

Die Funktion loadData lädt eine Liste mit allen Events, welche auf einer bestimmten Datenbank ausgeführt wurden und gibt diese an den Aufrufer zurück (Zeile 109 - 122). Das Passwort für die PasswordDB ist fest hinterlegt, die Passwörter der üblichen Datenbanken werden aus der PasswordDB abgefragt.

IWorker.cs

Listing A.24: Code des Interfaces IWorker

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Presentation_Layer.WorkerChain
8  {
9      internal interface IWorker
10     {
11         internal void execute(EventArgs e);
12     }
13 }
```

Das Interface IWorker, welches in Listing A.24 abgebildet ist, stellt die Grundlage der Worker-Klassen dar. Hierzu wird definiert, dass alle Worker die Methode execute (Zeile 11) implementieren müssen.

Workerchain.cs

Listing A.25: Code der Klasse WorkerChain

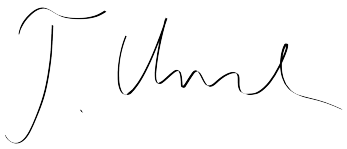
```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Presentation_Layer.WorkerChain
8  {
9      internal class WorkerChain
10     {
11         //List of all registered Workers
12         private List<IWorker>? Workers;
13
14         //Execution Method
15         internal void execute(object sender, EventArgs e)
16         {
17             if(Workers == null)
18             {
19                 throw new InvalidOperationException("There aren't Workers
                       set");
20             }else if(Workers.Count == 0)
21             {
22                 throw new InvalidOperationException("There aren't Workers
                       set");
23             }
24             foreach(IWorker worker in Workers)
25             {
26                 worker.execute(e);
27             }
28         }
29
30         //Contructor with a List of Workers
31         internal WorkerChain(List<IWorker> workers)
32         {
33             Workers = workers;
34         }
35         //Constructor for use without Workers
36         internal WorkerChain()
```

```
37          {
38              Workers = new List<IWorker>();
39          }
40
41          //Function to Add Workers
42          internal void AddWorkers(IWorker worker)
43          {
44              Workers.Add(worker);
45          }
46      }
47  }
```

In Listing A.25 ist der Code für die Klasse WorkerChain abgebildet. Diese besteht aus einer Liste von Klassen, welche IWorker implementieren (Zeile 12), zwei Konstruktoren zum Erzeugen eines Objekts, entweder mit einer Liste an Worker-Klassen (Zeile 31 - 34) oder komplett ohne Worker-Klassen (Zeile 36 - 39). Des Weiteren beinhaltet die Klasse noch eine Funktion um Worker zur Zuständigkeitskette hinzuzufügen (Zeile 42 - 45).
Das Herzstück der Klasse bildet der Eventhandler mit dem Name execute (Zeile 15 - 28), dieser prüft, ob das Event valide ist und gibt anschließend die Argumente des Arguments an alle in der Liste der Worker enthaltenen Elemente nacheinander weiter. Diese führen dann die Befehle in Form der Events aus.

Ehrenwörtliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit von mir selbstständig und ohne unerlaubte Hilfe angefertigt worden ist, insbesondere dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen sind, durch Zitate als solche gekennzeichnet habe. Ich versichere auch, dass die von mir eingereichte schriftliche Version mit der digitalen Version übereinstimmt. Weiterhin erkläre ich, dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde/Prüfungsstelle vorgelegen hat. Ich erkläre mich damit einverstanden, dass die Arbeit der Öffentlichkeit zugänglich gemacht wird. Ich erkläre mich damit einverstanden, dass die Digitalversion dieser Arbeit zwecks Plagiatsprüfung auf die Server externer Anbieter hochgeladen werden darf. Die Plagiatsprüfung stellt keine Zurverfügungstellung für die Öffentlichkeit dar.



Mellinghausen, 05.08.2024

(Ort, Datum)

(Eigenhändige Unterschrift)