



Abschlussprüfung Sommer 2020
Fachinformatiker Anwendungsentwicklung

DOKUMENTATION

Webservice für camt053 – Kontoauszüge
Einlesen und Verarbeiten von camt053 - Kontoauszügen
Mit anschließender Ausgabe von einem dazugehörigen Webservice

Prüfungsausschuss: FIAN_12
Abgabedatum: 04.06.2020

PRÜFLING:

Marvin Viedt
Bornhagenweg 39
12309 Berlin

AUSBILDUNGSBETRIEB:

a.b.s. Rechenzentrum GmbH
Invalidenstraße 34
10115 Berlin



Inhaltsverzeichnis**Abbildungsverzeichnis** **III****Tabellenverzeichnis** **IV****Glossar** **V**

1. Einleitung	1
1.1 Projektbeschreibung	1
1.2 Projektziele	1
1.3 Projektumfeld	2
1.4 Projektbegründung	2
1.5 Projektschnittstellen	2
1.6 Projektabgrenzung	2
2. Projektanalyse	2
2.1 Projektphasen	2
2.2 Ressourcenplanung	3
2.3 Entwicklungsprozess	3
2.4 Ist-Zustand	4
2.5 Soll-Konzept	4
2.6 Abweichungen zum Projektantrag	4
2.7 Wirtschaftlichkeitsanalyse	4
2.7.1 Projektkosten	4
2.7.2 Amortisationsdauer	5
2.8 Use-Cases	5
2.9 Lastenheft	5
3. Projektplanung	6
3.1 Programmablaufplan (PAP)	6
3.2 Pseudocode	7
3.3 Projektverzeichnis mit Git	8
3.4 Pflichtenheft	8
4. Projektumsetzung	8
4.1 Einführung in die Programmiersprache GO	8
4.2 Implementierung des Webservers	9
4.3 Integration: Einlesen und Parsen von XML Dateien	10
4.4 Parsen der XML-Daten mit CSS aufbereiten	12
4.5 Implementieren der DOM	12
4.6 Testverfahren	13
5. Projektabschluss	13
5.1 Projektabnahme	13

Inhaltsverzeichnis

5.2	Einführung	13
6.	Dokumentation	14
7.	Projektfazit	14
7.1	Soll/Ist Vergleich	14
7.2	Ausblick	15
	Literaturverzeichnis	16
A.	Anhang	i
A.1	Zeitplanung im Detail	i
A.2	Eingesetzte Ressourcen	ii
A.3	Programmablaufplan	iii
A.4	Lastenheft (Auszug)	iv
A.5	Pflichtenheft (Auszug)	v
A.6	Use-Case-Diagramm	vi
A.7	Go XML Demonstration.	vii
A.8	Grundlegender Go Webserver (Beispiel)	viii
A.9	A Tour of Go	ix
A.10	Pseudocode (Beispiel)	x
A.11	main.go (Go-Quellcode).	xi
A.12	webseite.go (Go-Quellcode)	xv
A.13	Webseite.css (css-Quellcode)	xv
A.14	Kontoauszugs-View.go (Go-Quellcode).	xvi
A.15	Kontoauszugs-View.css (css-Quellcode)	xvii
A.16	XML-View.go (Go-Quellcode)	xviii
A.17	XML-View.css (css-Quellcode)	xviii
A.18	Browseransicht: Hauptseite + Eingabefeld	xix
A.19	Browseransicht: Kontoauszugs-View	xix
A.20	Browseransicht: XML-View	xx
A.21	CAMT053 Datei (Auszug)	xxi
A.22	Benutzerdokumentation (Auszug)	xxiv
A.23	Entwicklerdokumentation (Read Me)	xxv

Abbildungsverzeichnis

1	Programmablaufplan	iii
2	Use-Case-Diagramm	vi
3	Go XML Demonstration (Quellcode)	vii
4	Go grundlegender Webserver (main.go)	viii
5	Go grundlegender Webserver (main.go mit HTML Template).	viii
6	Go grundlegender Webserver (webseite.go)	viii
7	A Tour of Go	ix

Tabellenverzeichnis

Tabellenverzeichnis

1	Zeitplanung	2
2	Kostenaufstellung	4
3	PAP-Elemente	7
4	Go-Webserver-Packages	11

Glossar

Glossar

HTML	Hypertext Markup Language
XML	Extensible Markup Language
SQL	Structured Query Language
UML	Unified Modeling Language
DSGVO	Datenschutz-Grundverordnung
EBICS	Electronic Banking Internet Communication Standard
DIN	Deutsche Industrie Norm
DOM	Document Object Model
SVN	Subversion
CSS	Cascading Style Sheets
CLI	Command Line Interface

1. Einleitung

1. Einleitung

Die folgende Projektdokumentation soll den Bearbeitungsprozesses des IHK-Abschlussprojektes, welches der Autor im Rahmen seiner Ausbildung zum Fachinformatiker Schwerpunkt Anwendungsentwicklung durchgeführt hat ausführlich vorstellen. Der zuständige Ausbildungsbetrieb ist die a.b.s. Rechenzentrum GmbH, ein Rechenzentrum, dass sich seit 60 Jahren auf die Erstellung von Lohnabrechnungen und Finanzbuchhaltungen spezialisiert hat. Es betreut heute über 4000 Firmen an insgesamt drei Standorte bundesweit. Die langjährige Erfahrung in der dazugehörigen Dienstleistung werden durch Zertifikate und einer hohen Qualitätsprüfung manifestiert.

1.1 Projektbeschreibung

Wie jede andere Firma auch, muss auch die a.b.s. Rechenzentrum GmbH ihre Umsätze genau im Überblick behalten. Das geschieht in Form von Kontoauszügen, die eine detaillierte Auflistung aller Umsätze, Überweisungen, Saldo und weitere zusätzliche Informationen beinhalten. Allerdings werden viele betriebsinterne Kontoauszüge teilweise auch mit Hilfe einer CAMT053 Datei von der jeweiligen Bank mit Hilfe des EBICS-Verfahrens zur Verfügung gestellt. Die in dieser Datei enthaltenden Daten sind im XML Format gespeichert. XML ist eine Auszeichnungssprache, die eine komplexe und beliebige Anzahl von Daten beinhalten kann. Zwar hat XML viele Ähnlichkeiten mit HTML, jedoch ist es ohne Programmierkenntnisse schwierig und meist zeitaufwendig diese Dateien manuell durchzugehen. In diesem Projekt, soll die oben genannte Problemstellung in Form eines Webservices gelöst werden.

1.2 Projektziele

Das Ziel dieses Projektes ist es einen funktionsfähigen Webservice zu erstellen, der die jeweiligen Daten aus einer zuvor vom Benutzer eingegebenen CAMT053 Datei einliest, anschließend aufbereitet und im Browser mit ansprechender Optik ausgibt. Jede einzelne Datei ist Teil einer großen Sammlung mehrerer Dateien, die jeweils die entsprechenden Daten von unterschiedlichen Kontoauszügen als XML-Datenbank enthält.

Für die Umsetzung des Webservices kommt hierbei die neuartige Programmiersprache Go zum Einsatz, die in diesem Projekt ab Kapitel 4 [Projektumsetzung](#) dieser Dokumentation ausführlich vorgestellt und praxisnah demonstriert wird.

Die seit 2009 der Öffentlichkeit vorgestellte junge kompilierbare Programmiersprache Go hat eine auf C basierte Syntax. Allerdings besitzt sie auch viele Ähnlichkeiten zu anderen Programmiersprachen wie C++ und Java. Darüber hinaus bietet sie sehr viele nützliche Vorteile, wie die Unterstützung von Multithreading, einer automatischen Speicherbereinigung, geringer Speicherbedarf der Go-Programme, OpenSource, die Fähigkeit leicht erlernt zu werden und einer hohen Übersetzungsgeschwindigkeit.

Ein weiterer Aspekt für die Verwendung von Go besteht darin, dass eine Vielzahl von Packages und Bibliotheken dem Nutzer zur Verfügung stehen. Im Falle des vorliegenden Projektes kommen demnach folgende Go Packages wie http, strings, fmt, regexp, encoding/xml und Bibliotheken wie XPath oder XQuery zum Einsatz.

Um auf die Daten der XML-Datenbanken zugreifen zu können, findet dabei ein zusätzliches XPath Package unter dem Namen xmlQuery Verwendung, dass mit Hilfe von XPath Ausdrücken, wie beispielsweise xmlQuery.FindOne (Vergleichbar mit SQL SELECT), gezielt auf XML-Tags sowie deren Werte direkt zugreifen und sie mit Hilfe einer Schleife ausgeben kann.

1. Einleitung

1.3 Projektumfeld

Der Auftraggeber des Projektes ist die Geschäftsleitung der Berliner EDV Abteilung der a.b.s. Rechenzentrum GmbH. Die Niederlassung in Berlin bildet zusammen mit der Niederlassung in Chemnitz die Abteilung a.b.s. Nord. Da die Niederlassung in Chemnitz auch den überwiegenden Teil der Finanzbuchhaltung der Firma übernimmt und die dortigen Programmierer mit anderen betriebsinternen Prozessen beschäftigt sind, wurde die Bearbeitung des vorliegenden Projektes von der Geschäftsleitung an den Autor dieser Dokumentation übertragen. Dadurch ist eine direkte Rücksprache und Kommunikation beider Parteien zwingend erforderlich.

1.4 Projektbegründung

Da die CAMT053-Dateien im XML Format zur Verfügung gestellt werden und somit viele Ähnlichkeiten zu HTML aufweisen, ist ein nötiges Fachwissen durchaus erforderlich, um diese auch gut nachvollziehen zu können. Gerade für Personen die in der Softwareentwicklung kein Fachwissen verfügen, können aufgrund der Komplexität der einer XML-Daten schnell überfordert werden. Zudem ist die manuelle Extrahierung der XML-Daten aufgrund des Umfangs sehr zeitaufwendig und erfordert zusätzlich viel Recherchearbeit. Dadurch hat sich die a.b.s. Rechenzentrum GmbH entschieden den Webservice in Auftrag zu geben.

1.5 Projektschnittstellen

Da die CAMT053 Dateien für das Projekt mit Hilfe des sogenannten EBICS-Verfahrens an die a.b.s. Rechenzentrum GmbH übertragen werden, unterliegen diese den strengen Sicherheitsrichtlinien der DSGVO. Da die dort enthaltenden Daten den Anforderungen des SEPA CAMT053 unterliegen, können diese direkt auf den jeweiligen Internetseiten und dazugehörigen Dokumenten analysiert werden. Dadurch soll es möglich sein die CAMT053 Dateien gezielt extrahieren zu können und ansprechende Ergebnisse zu liefern. Um einen Einblick in die Struktur einer CAMT053 Datei zu erhalten befindet sich im Anhang [A.21: CAMT053 Datei \(Auszug\)](#) ab Seite [xxi](#) ein entsprechender Auszug.

1.6 Projektabgrenzung

Da der Projektumfang für das Abschlussprojekt festgelegt ist, wird auf die Implementierung möglicher zusätzlicher Features für den Webservice vorerst verzichtet.

2. Projektanalyse

2.1 Projektphase

Für die Umsetzung des Projektes standen dem Autor 70 Stunden zur Verfügung. Diese wurden vor dem Beginn des Projektes auf verschiedene Phasen aufgeteilt, die während der Umsetzung der Software durchlaufen wurden. Als grobe Übersicht über die Zeiteinteilung zu den Hauptphasen des Projektes können der Tabelle 1: [Zeitplanung](#) entnommen werden. Eine detaillierte Übersicht der jeweiligen Phasen und deren Meilensteine befindet sich im Anhang [A.1: Zeitplanung im Detail](#) auf Seite [i](#).

Projektphase	Geplante Zeit
Projektanalyse	6h
Projektplanung	11h
Projektumsetzung	39h
Projektabgabe	5h
Projektdokumentation	9h
Gesamtdauer	70h

Tabelle 1: Zeitplanung

2. Projektanalyse

2.2 Ressourcenplanung

Eine ausführliche Übersicht, über die im Projekt verwendeten Ressourcen befinden sich im Anhang [A.2: Eingesetzte Ressourcen](#) auf der Seite [ii](#). Gemeint sind hierbei sowohl verwendete Hard- und Softwareressourcen als auch Personal. Bei der Auswahl, der verwendeten Software wurde besonders darauf geachtet, dass diese kostenfrei (z.B. Open Source) zur Verfügung steht oder dass die a.b.s. Rechenzentrum GmbH bereits Lizenzen für diese besitzt. Dadurch sollen eventuell anfallende Projektkosten möglichst auf einem geringem Niveau bleiben.

2.3 Entwicklungsprozess

Wie in jedem umfangreichem Projekt ist es wichtig sich vor der Umsetzung über einen optimalen Entwicklungsprozesses Gedanken zu machen. Zu diesem Zweck entschied sich der Autor für einen agilen Entwicklungsprozess auf Basis der Scrum-Methode. Sinn und Zweck der agilen Softwareentwicklung ist es schnell auf sich wechselnde Anforderungen reagieren zu können. Anders als in der klassischen Vorgehensweise, wo ein Projekt vom Start- bis Endpunkt durchweg geplant und anschließend in einem langen Entwicklungsprozess entwickelt wird, ist in der agilen Softwareentwicklung ein iterativer Entwicklungsprozess vertreten. Hierbei wird die Entwicklung sowohl in mehrere, als auch kurze Abschnitte, sogenannten Meilensteinen aufgeteilt.

Der Vorteil liegt darin, dass mit Hilfe der abgearbeiteten Meilensteine ein besserer Überblick über den Entwicklungsstandes des Projektes dem Auftraggeber zwischenzeitlich präsentiert werden kann. Dadurch ist es jederzeit möglich eventuell aufkommende Anpassungswünsche in den Entwicklungsprozesses zu integrieren. Da der gewählte Entwicklungsprozess auch auf die Scrum-Methode Bezug nimmt, sind zudem drei dementsprechende Rollen vertreten. Der sogenannte Product Owner stellt hierbei die Geschäftsleitung der a.b.s. Rechenzentrum GmbH da, die eine funktionfähige Software für ihre Finanzbuchhaltung bestellen.

Das Team wurde hierbei aus dem Autor, der Geschäftsleitung und der Buchhaltung gebildet. Der Autor selbst war für das eigene Projektmanagement und der selbstständigen Entwicklung der bestellten Software verantwortlich. Als Qualitätsmanagement fungierte hierbei die Geschäftsleitung und die Buchhaltung, die sich zwischenzeitlich über die Entwicklung einen Überblick verschaffte. Die letzte Rolle der sogenannte Scrum Master wurde aus den übrigen Mitarbeitern der a.b.s. Rechenzentrum GmbH gebildet, die dem Autor als auch der Geschäftsleitung als zusätzliche Ansprechpartner fungierten.

Die Scrum-Methode wird neben den drei Rollen auch aus vier Ereignissen gebildet. Sprint Planning beschreibt hierbei die Aufteilung bestimmter Aufgaben (Tasks), die an einem oder mehreren Tagen abgearbeitet werden mussten. Hierbei war auch eine Kommunikation unter den drei Rollen wichtig, die hierbei in kurzen Meetings erfolgte. Das Daily Scrum wurde in dem Projekt nicht immer täglich sondern eher wöchentlich abgehalten, da mehrere Entwicklungsabschnitte mehr Zeit in Anspruch nahmen. Aufkommende Probleme wurden dann dementsprechend gelöst. Mit Hilfe des Sprintreviews wurden bestimmte Abschnitte des Projektes abgeschlossen und bei Bedarf der Geschäftsleitung präsentiert. Zwischenzeitlich wurde mit der Sprint Retrospective ein Zwischenstand über die Arbeitsweise vermittelt, um Behinderungen und aufkommende Probleme zu bewältigen. Das Projekt selbst wurde durch die Geschäftsleitung, die laut Scrum der Product Owner darstellt, mit einer festgelegten To-Do Liste aus Anforderungen (Requiriments) gebildet.

Der Autor hatte jedenfalls drei Hauptaufgaben. Zum einen sollte er zuerst den Webservice in Go programmieren und den Inhalt einer CAMT053 Datei auf dem Browser parsen. Der nächste Schritt war die visuelle Aufbereitung der geparsen XML-Daten auf dem Browser mit CSS. Die letzte Aufgabe bestand darin ein DOM in das Projekt zu integrieren. Um den Webservice nach Fehler oder funktionellen Schwachstellen zu untersuchen, wurde der Quellcode zwischenzeitlich mit dem Whitebox Testverfahren und mehreren Zwischentests geprüft. Die verwendeten Testverfahren werden in dieser Dokumentation im Kapitel [4: Projektumsetzung](#) näher betrachtet.

2. Projektanalyse

2.4 Ist-Zustand

Wie bereits im Kapitel [1.1 Projektbeschreibung](#) erwähnt wurde, werden die CAMT053 Dateien mit Hilfe des EBICS-Verfahrens an die a.b.s. Rechenzentrum GmbH übertragen. Die dort enthalten Daten, sind auf das Schema des XML-Formates dementsprechend zugeschnitten. Da XML strukturelle und identische Eigenschaften wie HTML aufweist, indem es die Daten in jeweilige Tags abspeichert, ist das Parsen der Daten relativ simpel. Da die CAMT053 Dateien allerdings hochsensible Daten beinhalten, sind diese selbstverständlich streng vertraulich und nicht jedem zugänglich. Da es sich hierbei um Daten von Kontoauszügen handelt, können diese auf herkömmlicher Art direkt vom Finanzinstitut entweder online abgerufen oder postalisch an den jeweiligen Empfänger versendet wurden. Die Kontoauszüge haben dementsprechend das Design des jeweiligen Finanzinstitutes. Durch die verschlüsselte Versendung der Daten als CAMT053 Datei über das EBICS-Verfahren, können diese anschließend so visuell aufbereitet werden, so dass Kontoauszüge auch mit dem Design der jeweiligen Firma dementsprechend gestaltet werden. Dadurch ist es möglich, dass die Kontoauszüge auch der firmeneigenen Design Philosophie ihrer Dokumente entspricht.

Der gesamte Prozess, um die Kontoauszüge zu verwalten, wurde vor Beginn des Projektes über den herkömmlichen Weg durchgeführt.

2.5 Soll-Konzept

Nach Abschluss des Projektes, soll die a.b.s. Rechenzentrum GmbH über einen sogenannten XML-View Webservice verfügen. Mit diesem Webservice soll es möglich sein CAMT053-Dateien, die über das EBICS-Verfahren an das a.b.s. Rechenzentrum GmbH versendet wurden individuell, einzulesen. Die jeweiligen Daten sollen dann extrahiert und im Browser visuell aufbereitet werden, so dass diese der Design Philosophie der Firma entspricht.

2.6 Abweichungen zum Projektantrag

Im Projektantrag wurde als einziges Testverfahren nur der Whitebox Test genannt. Das Projekt wurde im Nachhinein allerdings mit weiteren Testverfahren ergänzt. Zudem wurden im Nachhinein auch die zwei Ansichten für die Darstellung der CAMT053 Daten festgelegt. Diese wurden nach Abgabe des Projektantrages erst im Nachhinein hinzugefügt. Grund dafür war lediglich die Unkenntnis seitens des Autors wie die Ausgabe der CAMT053 Daten auf dem Browser dargestellt werden sollte.

2.7 Wirtschaftlichkeitsanalyse

Aufgrund der Problematik die CAMT053 Dateien mit sich bringen und diese bereits in den Kapiteln [1.4 Projektbegründung](#) und [2.4 Ist-Zustand](#) geschildert und erläutert wurden, ist die Umsetzung des Projektes unbedingt erforderlich. Ob sich die Umsetzung des Webservices auch aus wirtschaftlichen Faktoren in Form der Projektkosten lohnt, soll in den folgenden Kapiteln näher untersucht werden.

2. Projektanalyse

2.7.1 Projektkosten

Die Projektkosten, die während der Umsetzung des Projektes anfallen, sollen in diesem Kapitel veranschaulicht und kalkuliert werden. Dafür müssen neben den Personalkosten auch die dementsprechenden Ressourcen (Hard- und Software, Büroarbeitsplatz etc.), die während der Realisierung des Projektes fällig bzw. verwendet wurden berücksichtigt werden. Da die genauen Personalkosten aufgrund des Datenschutzes nicht herausgegeben werden dürfen, wird die Kalkulation auf Basis von Stundensätzen durchgeführt, die von der Personalabteilung festgelegt wurden. Der Stundensatz eines Auszubildenden beträgt demzufolge 10€, der eines Mitarbeiters 25€. Für die Nutzung von Ressourcen wurde ein pauschaler Stundensatz von 20€ festgelegt.

Die jeweiligen Kosten, die für die einzelnen Prozesse des Projektes anfallen sowie die Projektkosten können in der folgenden Tabelle 2: [Kostenaufstellung](#) entnommen werden.

Vorgang	Mitarbeiter	Zeit	Personal	Ressourcen	Gesamt
Entwicklungskosten	1 x Auszubildener	70h	700,00 €	1.400,00 €	2100,00 €
Fachgespräch	2 x Mitarbeiter	4h	200,00 €	160,00 €	360,00 €
Code-Review	1 x Mitarbeiter	3h	75,00 €	60,00 €	135,00 €
Abnahme	2 x Mitarbeiter	0,5h	25,00 €	20,00 €	45,00 €
Projektkosten gesamt					2.640,00 €

Tabelle 2: Kostenaufstellung

2.7.2 Amortisationsdauer

Im folgendem Kapitel wird ermittelt, ab welchem Zeitpunkt sich die Entwicklung der Software amortisiert hat. Mithilfe dieses Wertes kann genau bestimmt werden, ob sich die Umsetzung des Projektes aus wirtschaftlicher Sicht als sinnvoll erwiesen hat und ob sich während der Entwicklungsdauer auch Kostenvorteile für die Firma ergeben werden. Um die Amortisationsdauer zu berechnen, müssen die Anschaffungskosten durch die laufende Kostenersparnis, die durch die neue Software entsteht, dividiert werden.

Der Go Quellcode eines davor bearbeiteten Projektes, der für das etablieren des Webservers gedacht ist konnte daher leichter in das vorliegende Projekt übernommen werden. Dadurch war es möglich zusätzliche Bearbeitungszeit des Projektes gleich zu Beginn reduzieren zu können. Eine genaue Übersicht über die Zeiteinsparung der jeweiligen Projektprozesse wird in der folgenden Tabelle 3: [Zeiteinsparung](#) dargestellt.

2.8 Use-Cases

Um eine genaue Übersicht der Use-Cases (Anwendungsfälle) zu erhalten, die von der in diesem Projekt umgesetzten Software abgedeckt wird, wurde im Laufe der Projektplanung ein Use-Case-Diagramm erstellt. Dieses Use-Case Diagramm kann als Anhang [A.6: Use-Case-Diagramm](#) auf der Seite [vi](#) entnommen werden. Es beinhaltet lediglich alle Informationen, die für einen Endnutzer der Software relevant sind.

2.9 Lastenheft

Das letzte Kapitel der Projektplanung dieser Dokumentation, beschäftigt sich hierbei mit dem entsprechende Lastenheft, dass von der Geschäftsleitung der a.b.s. Rechenzentrum GmbH für dieses Projekt erstellt wurden. Als Inhalt des Lastenheftes stehen somit alle Anforderungen des Auftraggebers im Mittelpunkt, die von der Software dieses Projektes erfüllt werden müssen. Ein Auszug dieses Lastenheftes befindet sich im Anhang [A.4: Lastenheft \(Auszug\)](#) auf Seite [iv](#).

3. Projektplanung

3. Projektplanung

3.1 Programmablaufplan (PAP)

Mithilfe eines Programmablaufplanes (PAP), auch Flussdiagramm genannt, lässt sich ein Algorithmus eines Programmes grafisch darstellen. Hierbei werden auch die abgearbeiteten Operationen, die das Programm für die Lösung einer Aufgabe beinhaltet, beschrieben. Die genaue Darstellung erfolgt lediglich über Elemente, die nach der DIN 66001 genormt sind. Diese folgenden Grundelemente, stehen bei der Erstellung eines Programmablaufplanes zur Verfügung:

- Rechteck
- Rechteck (mit doppelten und vertikalen Linien)
- Kreis/Oval/abgerundetes Rechteck
- Pfeil/Linie
- Raute
- Parallelogramm

Jedes einzelne der oben aufgelisteten Elemente erfüllt innerhalb des Programmablaufplanes einen bestimmten Zweck. Diese werden nun in der folgenden Tabelle 3: [PAP-Elemente](#) näher erläutert.







Grundelement	Zweck
	Dieses Element beschreibt lediglich eine Operation/Tätigkeit, die im PAP abgearbeitet wird.
	Dieses spezielle Element, beschreibt lediglich die Ausführung eines Unterprogrammes innerhalb des PAPs.
	Dieses Element steht dafür, einen Prozess innerhalb des PAPs zu terminieren. Wird daher auch Terminator genannt.
	Diese zwei logischen Elemente stehen dafür eine Verbindung zum nächsten Prozess zu definieren.
	Um Verzweigungen oder auch Fallentscheidungen innerhalb des PAPs zu beschreiben, kommt dieses Element zum Einsatz.
	Mit diesem Element wird die Ein- und Ausgabe definiert, wird aber in der Regel im PAP nicht verwendet.

Tabelle 3: PAP-Elemente

Mit einem Programmablaufplan (PAP) ist es daher möglich den kompletten Ablauf eines Programmes übersichtlich und grafisch darzustellen.

Im Falle des vorliegenden Projektes, wurde im Sinne der Projektplanung ein detaillierter und umfassender Programmablaufplan (PAP) erstellt, das den genauen Ablauf des Programmes möglichst verständlich darstellen soll. Um einen Einblick in den erstellten PAP zu bekommen, ist er im Anhang [A.3: Programmablaufplan](#) auf Seite [iii](#) entnehmbar.

3 Projektplanung

3.2 Pseudocode

Neben dem Programmablaufplan (PAP), der im letzten Kapitel vorgestellt wird, bietet sich die Verwendung eines Pseudocodes an. Ein Pseudocode ist lediglich ein Programmcode, der nicht zur maschinellen Interpretation, sondern zur Veranschaulichung eines Algorithmus gedacht ist. Im Grunde ist ein Pseudocode nichts anderes als ein Entwurf, der Schritt für Schritt alle notwendigen Prozesse beinhaltet, die für die Umsetzung des Programms relevant sind.

Der Pseudocode wird von anderen Programmierern überwiegend zwischen dem Anfangs- und Entwicklungsstadium eines Programms eingesetzt und dient daher als Zwischenschritt in der Programmierung. Er bietet sehr viele nützliche Vorteile gegenüber anderen Arten der Veranschaulichungen eines Programms. Diese Vorteile werden nun näher beleuchtet:

Wie bereits erwähnt, lässt sich durch die Verwendung eines Pseudocodes die Funktionsweise eines Algorithmus beschreiben. Demzufolge können bestimmte Abschnitte eines Programms gezielt nach bestimmten Techniken, Mechanismen und Konstrukten begutachtet werden. Viele Programmierer benutzen Pseudocodes daher auch um die Schritte ihres Programms schneller erklären zu können.

Der Pseudocode wird unter anderem in einer nicht technisch festgelegten Sprache formuliert. Das bedeutet, dass jeder Nicht-Programmierer in der Lage ist den Pseudocode zu verstehen. Allerdings kann der Pseudocode unter Umständen auch an einer bestimmten Programmiersprache angelehnt sein.

Zudem kann der Pseudocode auch dazu dienen, innerhalb des Entwicklungsprozesses auftretende Probleme zu lösen. Das ist insbesondere in einer angehenden Teamarbeit mit anderen Programmierern von Vorteil, denn dadurch kann man das Problem besser vermitteln.

Standardgemäß unterliegt Pseudocode keiner bestimmten fest definierten Richtlinie, wie er zu schreiben ist. Zwar kann dieser unter Umständen, wie oben bereits erwähnt, an Programmiersprachen angelehnt sein folgt, aber nicht deren Syntax in ausführlicher Form. Allerdings kann Pseudocode in eine beliebigen Programmiersprache umgeschrieben werden.

Um das Prinzip des Pseudocodes anhand eines praktischen Beispiels zu veranschaulichen, wurde ein bestimmter Teil des Quellcodes die Funktion `readxmlfile` des vorliegenden Projektes, als Vorlage genommen.

Einen Auszug des dazu erstellten Pseudocodes kann im Anhang [A.10: Pseudocode \(Beispiel\)](#) auf Seite [x](#) als Einblick entnommen werden.

4. Projektumsetzung

3.3 Projektverzeichnis mit Git

Um das Projekt verwalten und leichter auf den neuesten Stand zu halten, wurde hierbei auf ein Versionsverwaltungssystem zugegriffen. Versionsverwaltungssysteme haben daher den Zweck jegliche Arten von Änderungen an Projektdokumenten oder Dateien zu erfassen. Hierbei wurden alle Versionen in einem Archiv mit Zeitstempel und User-Athentifizierung abgelegt. Dadurch ist es jederzeit möglich auf ältere Stände des Projektes zuzugreifen und gegebenenfalls wiederherzustellen.

Zwei der bekanntesten Versionsverwaltungssysteme, die auch in der a.b.s. Rechenzentrum GmbH eingesetzt werden, sind SVN und Git. Da der Autor allerdings neben seinem beruflichen Werdegang Git auch für andere Projekte verwendet, entschied er sich für dieses Versionsverwaltungssystem aus folgenden Gründen:

Das Versionsverwaltungssystem Git besitzt einen Onlinedienst der als Github bekannt ist. Mit dem ist es möglich die erstellten Git-Repositories gezielt einzusehen und gegebenenfalls auch visuell überprüfen zu können. Zwar haben Git und SVN, was deren Befehle angeht, viele Ähnlichkeiten, doch fiel es dem Autor leichter die Versionsverwaltung seiner Projekte mit Git vorzunehmen.

3.4 Pflichtenheft

Basierend auf das vom Auftraggeber erstelltem Lastenheft (Kapitel 2.9: [Lastenheft](#)) wurde zum Ende der Projektplanung ein Pflichtenheft erstellt. Darin wird beschrieben, wie und womit der Autor die geforderten Anforderungen des Auftraggebers umsetzen möchte. Das Pflichtenheft, dient daher als Leitfaden, um die Umsetzung des Projektes durchführen zu können. Um einen Einblick in das Pflichtenheft zu bekommen ist dies als Auszug im Anhang [A.5: Pflichtenheft \(Auszug\)](#) auf Seite [v](#) entnehmbar.

4. Projektumsetzung

4.1 Einführung in die Programmiersprache Go

Die seit 2009 der Öffentlichkeit vorgestellten jungen kompilierbaren Programmiersprache Go hat eine auf C basierte Syntax. Allerdings besitzt sie auch viele Ähnlichkeiten zu anderen Programmiersprachen wie C++ und Java. Darüberhinaus bietet sie sehr viele nützliche Vorteile, wie die Unterstützung von Multithreading, einer automatischen Speicherbereinigung, geringer Speicherbedarf der Go-Programme, OpenSource, die Fähigkeit leicht erlernt zu werden und einer höheren Übersetzungsgeschwindigkeit. Die aktuellste Version der Programmiersprache Go kann von der Homepage: www.golang.org für die Betriebssysteme: Windows, MAC OS und Linux entweder direkt als Dateidownload oder übers Terminal mit Hilfe von Befehlen bezogen und installiert werden. Auf der Homepage von Go lassen sich zudem eine umfassende Dokumentation der jeweiligen Funktionen, Packages als auch mehrere Referenzen entnehmen.

Jeder Programmierer, der in dieser Programmiersprache einsteigen möchte, hat auch die Möglichkeit eine sogenannte „A Tour of Go“ zu absolvieren. Dabei handelt es sich um ein umfassendes Tutorial, dass sowohl die Basics als auch erweiterten Funktionen von Go praxisnah erläutert. Der Programmierer ist zu dem auch in der Lage dieses Tutorial seiner jeweiligen Sprache zuvor anzupassen. Unterstützte Sprachen sind hierbei beispielsweise Deutsch, Englisch, Russisch, Spanisch, Japanisch und viele weitere. Da dieses Tutorial auf einem Webservice unter den Namen „Go Playground“ aufgebaut ist, kann dadurch jeder Programmierer, den Go-Quellcode, der neben dem Tutorial als Demonstrationzweck dient jederzeit individuell erweitern, ändern und anschließend direkt ausführen. Daher können sowohl Anfänger als auch Profis ihre Ideen oder Konzepte direkt ausprobieren, bevor diese anschließend in den jeweiligen angehenden Projekten implementiert werden.

Um ein visuelles Beispiel sowohl als Einblick in dieses Tutorial, als auch des dazugehörigen Webservice zu erhalten, wird im Anhang [A.9: A Tour of Go](#) auf Seite [ix](#) als Screenshot dargestellt. Ein weiterer Aspekt für die Verwendung von Go besteht darin, dass jeder Programmierer auf einer Vielzahl von Packages und Bibliotheken zugreifen kann. Im Falle des vorliegenden Projektes kommen demnach folgende Go Packages wie http, strings, fmt, regexp, encoding/xml und Bibliotheken wie XPath oder XQuery zum Einsatz.

4. Projektumsetzung

4.2 Implementierung des Webservers

Der erste Schritt für die Umsetzung der Projektsoftware war es den Webserver zu implementieren. Da hierbei als Vorlage einige Quellcode Segmente verwendet wurden, die der Autor bereits in vorherigen Projekten bezüglich des Webservers angewendet hatte, waren hier wenige Schritte nötig, um den Quellcode des Webservers auf die neue Software anzupassen. Damit man allerdings eine genaue Vorstellung davon bekommt, wie man in Go einen einfachen Webserver erstellt, wird nun anhand eines praktischen Beispiels demonstriert.

Damit der Go Compiler nach Betätigung des Buttons „Run“ auch weiß, dass der nun folgende Go Quellcode auch zu einem ausführbaren Programm übersetzt wird, muss als erstes das Package „main“ angegeben werden. Tut man das nicht, wird der Go Quellcode als eine freigegebene Bibliothek (shared library) abgearbeitet. Das Package beinhaltet logischerweise die Funktion main(), die für die weiteren Schritte erforderlich ist. Als letztes werden weitere Packages in den Go Quellcode importiert. Dabei handelt es sich um die Packages fmt, log und net/http. In der folgenden Tabelle Go-Webserver-Packages werden die vier einzelnen Packages genauer analysiert.

Package	Zweck	Funktionen (Beispiele)
fmt	Ausgeben von Werten und anderen Daten	Printf(), Fprintf(), Println()
log	Implementierung eines Standard Logger, der nach stderr schreibt und die Ausgabe mit einem Zeitstempel (timestamp) ausgibt.	Fatal(), Printf(), Panic()
net/http	Bereitstellen von HTTP Client und Server Implementierungen.	HandleFunc(), Handle()
html/template	Implementieren von Templates, um die Ausgabe von HTML zu generieren.	New(), Parse(), Execute()

Tabelle 4: Go-Webserver-Packages

Nachdem wir nun die Packages, die wir benötigen in den Quellcode als Import definiert haben, folgt nun die allgemeine Umsetzung des Webservers selbst. Schauen wir uns zu diesem Zweck zuerst die Funktion main() an. Als erstes schreiben wir den Funktionsaufruf: `http.HandleFunc("/", handler)` in die Funktion „main“, welche dem Package http mitteilt alle Anfragen (requests) zu dem Web-Root („/“) mit handler abzuarbeiten. Als letztes schreiben wir: `log.Fatal(http.ListenAndServe(":3000", nil))` in die zweite Zeile der Funktion „main“. Die Funktion `log.Fatal()`, sorgt dafür, dass jeglicher unerwarteter Fehler, der von dem als Parameter gesetzten Funktionsaufruf `http.ListenAndServe()` auftritt, als Log im Standarderror (stderr) mit Zeitstempel als Fehlerbericht ausgegeben wird. Die Funktion `http.ListenAndServe()` definiert, dass der Port 3000 an jedem Interface (":3000") abgerufen wird. Somit ist die Funktion „main“ abgeschlossen.

Damit der Webserver nun auch ohne Fehler funktioniert, müssen wir noch eine Funktion handler() definieren. Diese Funktion, bekommt die Argumente `http.ResponseWriter` und `http.Request` als Parameter deklariert. Der beinhaltende Wert, der das Argument `http.ResponseWriter` mit liefert, sorgt dafür, dass die Antworten (Request) des Servers zusammengetragen werden, während Daten zu dem HTTP Client geschrieben werden. Die Daten-Struktur, die mit `http.Request` mitgeliefert wird, definiert den Http Request Client. Innerhalb der Funktion handler wird nun eine Ausgabe mit dem Funktionsaufruf `fmt.Fprintf()` deklariert. Mit dieser Funktion lassen sich Ausgaben direkt im Browser ausgeben, statt direkt auf dem Terminal. Um im Browser gleichzeitig auch ein typisches HTML Layout zur Verfügung zu stellen, ist es mit Hilfe des Packages `html/template` möglich HTML Quellcode über Templates einzubinden.

Zu diesem Zweck erstellen wir als ersten Schritt eine externe .go Datei mit dem Namen „webseite“. Diese Datei beinhaltet lediglich einen vereinfachten Quellcode, der aus zwei Elementen gebildet wird. Zum einen wird als erstes das Package main deklariert. Das bedeutet das der Go Compiler diese Datei als Teil der main.go ansieht und den beinhalteten Quellcode mit berücksichtigt. Als zweites wird mit `const webseite` eine konstante Variable vom Datentyp String deklariert. Als Zuweisung erhält sie den dazugehörigen HTML Quellcode für die Browseransicht unseres Webservices. Um die neue .go Datei auch in den Go Quellcode unseres Webservices zu integrieren werden nun zwei neue Variablen deklariert. Hierbei handelt es sich um die Variablen `tmpl` und `err`. Als Zuweisung erhalten die Variablen die Funktionsaufrufe `New()` und `Parse()`, die aus dem Package `html/template` stammen. Mit der Funktion `New("Webseite")` wird ein neues Template mit den Namen Webseite

4. Projektumsetzung

definiert. Die Funktion Parse erhält als Parameter den Namen unserer konstanten Variablen „webseite“ aus unserer Datei „webseite.go“. Dadurch wird der HTML Quellcode, der in der Variablen enthalten ist, in die Funktion Parse übergeben. Wobei die Variable err den Rückgabewert der beiden Funktionen erhält. Dieser kann gegebenenfalls mit einer Verzweigung (If-Anweisung) als Fehlerbehandlung weiterverwendet werden. Zum Schluss wird der Variablen err: `tmpl.Execute(w, nil)` zugewiesen. Dadurch wird das Template mit dem Namen `tmpl` mit dem `ResponseWriter` ausgeführt. Nun ist der Go Quellcode vollständig und kann vom Go Compiler mit dem Befehl `go build` kompiliert werden. Möchte man sich nun den Webserver direkt im Browser anschauen, muss man zuvor die ausführbare Datei, die mit `go build` erstellt wurde ausführen. Mit der Ausführung wird eine Sitzung des Webservers im Browser gestartet. Gibt man im Browser nun die folgende URL an: „`http://localhost:3000/`“, wird der Webserver im Browser aufgerufen. Der für dieses Beispiel generierte Go Quellcode, befindet sich zur Einsicht als Anhang [A.8: Grundlegender Go Webserver](#) auf der Seite [viii](#).

4.3 Integration: Einlesen und Parsen von CAMT053 Dateien

Da eine CAMT053 Datei im allgemeinen im XML Format zur Verfügung gestellt wird ist es zu Beginn dieses Kapitels sinnvoll einen kurzen praktischen Einblick zu geben, wie mit Go eine einfache XML Datei erstellt wird. Diese Demonstration wurde mit Hilfe des Go Playground (Webservice) von golang.org durchgeführt.

Wie auch in der letzten Demonstration von Kapitel [4.2: Implementierung des Webservers](#), benötigen wir auch hier als erstes die Definition des Go Package `main` in unserem Go Quellcode. Was anschließend folgt, sind natürlich die üblichen Go Packages die in den Go Quellcode importiert werden. Für diese Demonstration werden die beiden Go Packages, `fmt` und `encoding/xml` benötigt. Das Package `fmt` wurde bereits in der Tabelle [4: Go-Webserver-Packages](#) ausführlich vorgestellt. Das neue Package „`encoding/xml`“ hat die Aufgabe einen einfachen XML 1.0 Parser in den Go Quellcode zu implementieren. Dieser XML Parser ist zudem auch in der Lage XML Namensräume zu berücksichtigen.

Nachdem nun die erforderlichen Packages in den Go Quellcode implementiert wurden, folgt nun die Definierung einer Struktur mit den Namen „`pruefing`“. Diese Struktur umfasst insgesamt sieben Variablen vom Datentyp `String` mit folgenden Bezeichnungen:

- `Prueflingsnummer`
- `IHK_Name`
- `Name`
- `Alter`
- `Ort`
- `Ausbildungsberuf`
- `Firma`

Jede der Variablen bezieht sich auf einen entsprechenden XML Tag, der in der Struktur ebenfalls seine Bezeichnung erhält. Die Struktur dient also dazu die XML Tags und deren Eigenschaften festzulegen.

Nun folgt die die Hauptfunktion unseres Go Programms, die mit `func main()` definiert wird. Innerhalb der spitzen Klammern (`{}`) wird zuerst eine Variable mit dem Namen „`pruefing`“ deklariert. Als Zuweisung erhält die Variable einen Pointer, der auf die Struktur `pruefing` verweist. Innerhalb der entsprechenden Klammern bekommen alle Variablen einen Wert als Muster vom Datentyp `String` zugewiesen. Das bedeutet, dass nach der Erstellung der XML alle Tags die Musterwerte enthalten.

Um die XML nun zu generieren, wird eine zusätzliche Variable `xml` deklariert. Da wir in dieser Demonstration keine Fehlerbehandlung benötigen, wird auf die Deklaration einer zweiten Variable verzichtet. Als Zuweisung erhält die Variable den Funktionsaufruf `MarshalIndent` aus dem Package `encoding/xml`.

Die Funktion `MarshalIndent` sorgt dafür, dass jedes XML Element in einer neu eingerückten Zeile beginnt. Zudem wird je nach Art der Verschachtelung der Einzug angepasst. Als Parameter erhält sie die Variable „`pruefing`“ als Interface, einen leeren Prefix und den Indent als Leerzeichen.

4. Projektumsetzung

Nun ist innerhalb der Variable `xml` der entsprechende XML Quellcode erstellt worden. Um ihn jedoch auf der CLI als Ergebnis ausgeben zu können muss als letztes die Funktion `Println` aus dem Package `fmt` aufgerufen werden. Als Parameter wird die Funktion `String` aufgerufen. Diese Funktion wandelt eingehende Bytes einer Variable oder eines Arrays/Slices in einem String um. In dem Fall werden die in der Variable hinterlegten Bytes in einen String umgewandelt, der anschließend mit der Funktion `Println` ausgegeben wird.

Nach diesem Schritt ist der Go Quellcode fertiggestellt und kann vom Go Compiler übersetzt werden und anschließend ausgeführt werden. Um den Go Quellcode innerhalb des Go Playgrounds übersetzen zu können, muss der Button „Run“ an der rechten Unterseite des Textfeldes betätigt werden. Das Kompilieren (übersetzen) kann je nach Komplexität des Quellcodes dauern. Nach dem Kompilieren wird der XML Quellcode unten als Ergebnis angezeigt. Der generierte Go Quellcode und die entsprechende Ausgabe dieser Demonstration kann visuell im Anhang [A.7: Go XML Demonstration](#) auf der Seite [vii](#) betrachtet werden.

Nachdem die Demonstration nun abgeschlossen ist, wenden wir uns nun wieder dem Umsetzungsprozess des Abschlussprojektes zu:

Strukturell ist die CAMT053-Datei, die jeweils die Daten eines Kontoauszuges enthält in zwei XML-Tags nach den Richtlinien des CAMT Verfahrens gegliedert. Es handelt sich dabei um die Tags `Group Header (GrpHdr)` und `Statement (Stmt)`. Der Tag mit den Namen `Group Header (GrpHdr)` beinhaltet lediglich die Adresse des Kontoinhabers, zudem die Message ID und das Datum an dem der Kontoauszug erstellt wurde. Der Tag `Statement (Stmt)` hingegen umfasst die getätigten Umsätze, Saldo und viele weitere Informationen, die der Kontoinhaber für den Monat durchgeführt hat. Um einen Überblick über die Struktur zu erhalten, ist ein kleiner Ausschnitt einer XML-Datei im Anhang auf der Seite darstellt. Aufgrund der strengen Datenschutz Richtlinien, sind alle Werte, die sich zwischen den XML-Tags befinden, unkenntlich gemacht worden. Da der Webservice dieses Projektes externe XML Dateien einlesen soll, muss der Nutzer mit Hilfe eines Eingabefeldes individuell die Namen der XML Dateien angeben können.

Die Umsetzung erfolgte hierbei in zwei Etappen. Als erstes wurde innerhalb des HTML Quellcodes ein Formular mit HTML Forms integriert, dass als Eingabetyp Text einliest und dementsprechend mit einem Button übermittelt. Innerhalb des Go Quellcodes für den Webservice wurden zwei zusätzliche Funktionen mit den Namen `getxmlfile` und `readxmlfile` definiert.

Die Funktion `getxmlfile` sorgt dafür, dass der eingegebenen Textwert des Formulars „XMLFile“ als String abgespeichert und als Rückgabewert an die Funktion zurückgegeben wird. Dieser Rückgabewert, wurde anschließend an die Funktion `readxmlfile` als Parameter übergeben. Da es sich hierbei um den Namen der XML Datei handelt, muss der Webservice auch die entsprechende XML-Datei aus einem Verzeichnis herausuchen und einlesen. Dazu wurde der String, der den Dateinamen beinhaltet, erweitert.

Durch den Aufruf der Funktion `Abs()` aus dem Package `path/filepath` konnte die genaue Position der XML-Datei als Dateipfad bestimmt werden. Dadurch können die XML-Dateien direkt mit der Funktion `Open()` anschließend in das Programm eingeladen werden. Um den Inhalt der XML-Datei im Browser zu parsen, kommt nun das spezielle Package `xquery` zum Einsatz. Das Package selbst, dass hierbei mit XPath Expressions arbeitet gibt es in zwei folgenden Variationen:

1. `htmlquery`
2. `xmlquery`

Im Falle unseres Projektes benötigen wir daher die zweite Variante `xmlquery` und stellt dem Nutzer eine Vielzahl von nützlichen Funktionen bereit, um die Ausgabe von XML-Dateien zu vereinfachen. Die Funktion `FindOne()` beispielsweise sucht den entsprechenden Knoten (Node) heraus, der einem spezialisierten XPath Ausdruck (Expression) entspricht. Anschließend wird das erste Element davon zurückgegeben.

Genau diese Funktion kam hierbei in der Funktion `readxmlfile()` zum Einsatz, um das erste Element der Tags `GrpHdr` und `Stmt` zu ermitteln. Um sowohl die inneren Tags, als auch deren Werte zugreifen und ausgeben zu können, wurde zudem eine Funktion mit dem Namen `printtagvalues()` definiert, das als Parameter das erste

4. Projektumsetzung

gefundenen Element der zwei Tags übergibt. Innerhalb dieser Funktion wurde mit Hilfe einer Zählschleife jedes einzelne unter Tag durchlaufen. Zudem wurden mit Funktionen aus dem Package regexp (Regular Expressions) verwendet, um unnötige Zeichen aus dem generierten String zu entfernen. Anschließend wurde der String in zwei weitere aufgeteilt, die wiederum sowohl den Tagnamen als auch den Wert enthalten. Zum Schluss wurden beide Strings über den Funktionsaufruf von `printmap()` als Parameter übergeben, wo diese schließlich als Map umgewandelt wurden und mit `Fprintf()` auf dem Browser ausgegeben wurden. Der entsprechende Go Quellcode kann im Anhang [A.11: main.go \(Go-Quellcode\)](#) auf der Seite [xi](#) als Einblick entnommen werden.

4.4 Parsen der XML-Daten mit CSS aufbereiten

Um die Ausgabe der CAMT053-Datei auf dem Browser visuell aufzubessern, wurde zuerst mit der Implementierung eines Fileservers begonnen. Zweck des Fileservers ist es lediglich eine CSS Datei in den Go Quellcode des Webservices einzubinden. Daher wurde innerhalb der Funktion `main` die `http` Funktion `Handle` hinzugefügt, die zusätzlich mit den Funktionsaufrufen von `http.StripPrefix()` und `http.FileServer()` den entsprechenden Fileserver erzeugt.

Die jeweiligen CSS Dateien befinden sich in dem Verzeichnis `static`, dass zuvor in der Funktion `Handle` mit den entsprechenden Funktionsaufrufen festgelegt wurde. Dadurch war es nun möglich ein ansprechendes Layout für die XML Ausgabe im Browser mit CSS zu gestalten. CSS ist eine Stil- und Formatierungssprache, die mit den entsprechenden HTML-Elementen verknüpft wird und somit die Formatierung im Browser umsetzt. CSS kann somit mehrere Design spezifische Aufgaben zur Gestaltung einer Webseite übernehmen, dazu gehört auch das Hinzufügen von farbigen Hintergründen, Schriftauszeichnungen, Ränder, Seitenränder. Links und Bildern. Die entsprechenden Auszüge aus den Go/HTML und CSS Quellcode für den CAMT053 – Webservice können in den jeweiligen Anhängen: [A.12: webseite.go \(Go-Quellcode\)](#), [A.13: Webseite.css \(CSS-Quellcode\)](#), [A.14: Kontoauszugs-View.go \(Go-Quellcode\)](#), [A.15: Kontoauszugs-View.css \(CSS-Quellcode\)](#), [A.16: XML-View.go \(Go-Quellcode\)](#) und [A.17: XML-View.css \(CSS-Quellcode\)](#) als Einblick betrachtet werden.

Da der Autor zudem auch eine hohe Begeisterung und Leidenschaft in der kreativen und gestalterischen Arbeit in anderen seiner Projekte investiert, war die Gestaltung des entsprechenden Layouts für den CAMT053 – Webservice mit wenig Aufwand abgeschlossen. Auch wenn für das Layout keine designtechnischen Anforderungen seitens des Auftraggebers existierten, wurde es dennoch an die Designphilosophie der a.b.s Rechenzentrum GmbH angepasst. Das fertige Layout für den CAMT053 – Webservice kann in den folgenden Anhängen [A.18: Browseransicht: Hauptseite + Eingabefeld](#), [A.19: Browseransicht: Kontoauszugs-View](#) und [A.20: Browseransicht: XML-View](#) auf den Seiten [xix](#) sowie [xx](#) entnommen werden.

4.5 Implementierung der DOM

Sowohl jede CAMT053-Datei, die in dem CAMT053 – Webservice eingelesen wird, beinhaltet mehrere Objekte. Diese Objekte werden in HTML als auch in XML in Form einer Baumstruktur von mehreren Tags repräsentiert. Daher besitzt jeder Tag entweder einen Wert oder mehrere darauffolgende Tags. Für den Autor waren hierbei zwei mögliche Optionen gegeben, wie er die DOM in den Webservice implementiert. Die erste Option war es mit dem Package `XMLQuery` jeden einzelnen Tag zu durchlaufen und sowohl deren Werte als auch Tagnamen in einer Map abzuspeichern. Mit dieser Methode wäre es daher möglich über die jeweiligen Werte über die Map direkt mit Angabe der Tagnamen zu beziehen.

Die zweite Optionen hingegen soll, nach dem erfolgreichen Durchlaufen der Tags, die Werte in ein Array/Slice vom Datentyp `String` abspeichern. Da Arrays unterschiedliche Längen haben können und deren Daten auf unterschiedlichen Elementen aufgeteilt sind, muss man die Array-Elemente (Index) genauer auf die Inhalte abfragen. Die Inhalte werden selbstverständlich für die Darstellung im Browser auch in einer Map abgelegt. Aufgrund von zunehmenden Zeitmangel in der aktuellen Situation mit dem Coronavirus entschied sich der Autor für die Darstellung der Kontoauszüge die zweite Option zu wählen.

Zwar war dem Autor bewusst, dass diese Methode nicht perfekt ist, dennoch ließ sich damit ein einfaches Muster für die Ansicht Kontoauszüge des CAMT053 – Webservice erstellen. Das Muster enthält lediglich die

5. Projektabschluss

notwendigen Daten eines Kontoauszuges mit der IBAN, BIC, BLZ, die jeweilige Kontonummer und einen einfachen Umsatz/Saldo und kann als Anhang [A.19: Browseransicht: Kontoauszugs-View](#) auf der Seite [xix](#) betrachtet werden. Neben der Ansicht des Kontoauszugs bietet der Webservice auch die Möglichkeit an alle Daten aus der CAMT053 Datei in Form einer XML Ansicht auf dem Browser formatiert auszugeben. Hierbei werden lediglich die Daten aus den zwei Hauptelementen (Tags): GrpHdr und Stmt als Liste dargestellt. Diese zusätzliche Ansicht soll dazu dienen bei Bedarf weitere Daten der Buchhaltung der a.b.s. Rechenzentrum GmbH zur Verfügung zu stellen. Die XML- Ansicht kann im Anhang [A.20: Browseransicht: XML-View](#) entnommen werden. Die jeweiligen Daten, die im Browser bei beiden Anhängen ausgegeben werden, sind aufgrund der strengen Datenschutzrichtlinien unkenntlich gemacht worden.

Da die Umsetzung der Ansicht des Kontoauszugs aufgrund der Coronakrise nicht mehr fertig gestellt werden konnte und daher nicht vollständig ist, werden die zukünftigen Schritte im Kapitel [7.2: Ausblick](#) näher erläutert.

4.6 Testverfahren

Während der Umsetzung kamen mehrere Methoden zum Testen der Qualitätsicherung des CAMT053 Webservice zum Einsatz. Beispielsweise um die Funktionalität von implementierten Funktionen oder Abläufen zu testen, wurden zusätzliche `fmt.Println()` Aufrufe, die Test auf der Kommandozeile (CLI) ausgeben, in den jeweiligen Stellen im Go-Quellcode eingebaut. Somit sollte geprüft werden, ob eine neue Funktion auch tatsächlich gestartet oder ob ein Prozess auch berücksichtigt und abgearbeitet wurde. Die Tests wurden daher nach dem Prinzip des White-Box-Test Verfahrens durchgeführt, dass hierbei jeden Abschnitt eines Programms auf die interne Funktionsweise hin überprüft. Da dieses Testverfahren auch mit einem sehr geringen Aufwand in der Organisation verbunden ist, können Tests daher an jede beliebige Stelle des Quellcodes durchgeführt werden.

Um Prozesse wie die Eingabe der CAMT053-Datei auf mögliche falsche Eingaben zu prüfen, wurden auch dementsprechende Fehlerbehandlungen in den Go-Quellcode implementiert, die dem Anwender mit einer jeweiligen Fehlermeldung in Kenntnis setzt. Andere Fehlerbehandlungen sind hingegen auf interne Prozesse beschränkt, die nur auf der Kommandozeile (CLI) für andere Entwickler relevant sind und ausgegeben werden.

Natürlich war es auch ratsam die Variablen und Arrays auf deren Werte zu überprüfen, bevor der Autor weiteren Implementierungen neuer Funktionen für den Webservice fortsetzte. Beispielsweise wurde geprüft, ob die Arrays `grpHdrarr` und `stmtarr` auch die jeweiligen Daten aus CAMT053 Datei nach dem Durchlaufen auch abgespeichert haben. Die Prüfung wurde lediglich mit dem Durchlaufen der beiden Arrays mit einer Zählschleife durchgeführt, die jeweils den Array Index an einer bestimmten Stelle ausgegeben hat. Dadurch war es möglich die kompletten Arrays zuerst auf der CLI als Zwischenstand ausgeben zu können. Später wurde diese Ausgabe dafür verwendet die XML Ansicht des Webservice zu erstellen.

5. Projektabschluss

5.1 Projektanbahnung

Nach der allgemeinen Fertigstellung des Webservices, konnte dieser der Geschäftsleitung vorgelegt werden. Aufgrund der Durchführung von regelmäßigen Meetings konnte sich die Geschäftsleitung jederzeit über den aktuellen Stand des Webservices informieren. Dadurch war nach der Endabnahme der aktuelle Stand des Webservice bereits vertraut. Demzufolge waren jederzeit Anregungen sowie Kritik seitens der Geschäftsleitung frühzeitig möglich. Generell stellt der Webservice die gewünschten Daten im Browser bereits zur Verfügung, nur die Formatierung und die visuelle Darstellung muss im nachhinein noch angepasst werden.

5.2 Einführung

Da jeder Webservice in der a.b.s Rechenzentrum GmbH in einem internen Server abgelegt wird, kann jeder Mitarbeiter diese ausführen. Dies wurde natürlich auch für den vorliegenden Webservice ermöglicht. Er steht nicht nur als ausführbare Datei im Server bereit, sondern auch der Go Quellcode kann von jedem Entwickler aufgerufen und bei Bedarf bearbeitet werden. Die Geschäftsleitung selbst führt Anwendungen in der Regel über

6. Dokumentation

die Kommandozeile aus. Da der Webservice darauf ausgelegt ist, kann dieser über die Kommandozeile direkt ausgeführt werden. Dadaurch ist keine zusätzliche Installation als Desktopanwendung notwendig.

6. Dokumentation

Die vorliegende Dokumentation des CAMT053 – Webservice besteht aus drei Teilen: der Projekt-, der Benutzer- und der Entwicklerdokumentation. Die Projektdokumentation beschreibt lediglich die einzelnen Phasen, die während des Projektes durch den Autor durchgeführt wurden. Die Benutzerdokumentation stellt Informationen über die Funktionweise und der Struktur der Software zur Verfügung. Es soll dazu dienen mögliche Fragen des Anwenders zu klären und soll zudem als Hilfestellung für die Einarbeitung neuer Mitarbeiter zu Verfügung stehen. Um einen Einblick in die Benutzerdokumentation zu bekommen, ist im Anhang [A.22: Benutzerdokumentation \(Auszug\)](#) auf der Seite [xxiv](#) ein Auszug zu finden.

Die Entwicklerdokumentation soll wiederum als ein umfassendes Nachschlagewerk für Entwickler dienen, die sich mit dem Quellcode der vorliegenden Software beschäftigen oder einarbeiten wollen. Im Falle des vorliegenden Projektes wurde die Entwicklerdokumentation als eine klassische Read Me Datei im Textformat verfasst. Sie beinhaltet hierfür detaillierte Informationen und Erläuterungen über die eingesetzten Funktionen und deren Attribute. Den entsprechenden Auszug zu der Entwicklerdokumentation ist im Anhang [A.23: Entwicklerdokumentation \(Read me\)](#) auf der Seite [xxv](#) zu entnehmen.

7. Projektfazit

Durch das vorliegende Projekt konnte der Autor wertvolle Erfahrungen bezüglich der Planung und Umsetzung komplexer Programmierprojekten sammeln. Auch wenn sich der Autor selbst sich noch als Anfänger im Bereich der Programmierung sieht, konnte er dennoch nützliche Erfahrungen, Vorgehensweisen und Funktionen erwerben die für die Umsetzung einer Software relevant sind. Besonders die Implementierung als auch Verarbeitung einer komplexen CAMT053 Datei waren für den Autor eine sehr spannende Erfahrung, wodurch auch neue Erkenntnisse erworben werden konnten.

Allerdings zeigte das Projekt auch dem Autor die Wichtigkeit der Kommunikation zu den Mitarbeitern und der Geschäftsleitung. Im Allgemeinen war die Kommunikation zwischen dem Autor und den Mitarbeitern sowie der Geschäftsleitung jederzeit gegeben, doch aufgrund des bestehenden Ehrgeizes des Autors an dem Projekt selbstständig zu arbeiten, nahm er eventuelle Hilfestellungen bei einem aufkommenden Problem erst nach eigenem Erproben von Ideen und möglichen Lösungen in Anspruch.

Nach Aufkommen des Coronavirus seit Januar 2020 änderten sich zudem auch viele organisatorische Aspekte in der Firma. Um möglichst viele Mitarbeiter vor einer Infektion zu schützen, wurden viele angewiesen im Homeoffice zu arbeiten. Diese Anpassung galt auch für den Autor selbst.

Da er allerdings auch übers das Heimbüro an dem vorliegenden Projekt arbeiten konnte, war die Umstellung kein Problem. Neben der Umsetzung des Webservice mit Go und den dazugehörigen Dokumenten konnte sich der Autor auch kreativ an dem Projekt mit zusätzlichen Erfahrungen bereichern. Besonders die Arbeit mit CSS und HTML zur Erstellung des Layouts des Webservice war ebenfalls eine sehr erfüllende Tätigkeit.

Das Schreiben der Dokumentation selbst war aufgrund der Kenntnisse einer zuvor geschriebenen Facharbeit von der Formulierung und vom Umfang her kein Problem. Ausserdem war es zum Teil aufwendig die entsprechenden Quellcode Segmente, Diagramme und Informationen zusammenzutragen.

Dennoch ist dies eine nützliche Erfahrung, um in zukünftigen Projekten weitere Vorgehensweisen seitens des Autors zu entwickeln und die bestehenden zu verbessern. Zusammenfassend lässt sich folgendes als Fazit definieren: Die Realisierung des vorliegenden Projektes war nicht nur eine umfassende und größere Erfahrung für den Autor, sondern gleichzeitig eine Bereicherung für die a.b.s. Rechenzentrum GmbH.

7. Projektfazit

7.1 Soll/Ist Vergleich

Wenn das IHK-Abschlussprojekt mit Hilfe einer rückblickenden Betrachtung unterzogen wird, können viele Anforderungen, die aus dem Auszug des Lastenheftes im Anhang [A.4: Lastenheft \(Auszug\)](#) auf der Seite [iv](#) definiert sind, als erfüllt gelten. Das Projekt hat hierbei einen Webservice hervorgebracht, der nicht nur in der neuen Programmiersprache Go programmiert wurde, sondern auch in der Lage ist einen Webserver zu etablieren, eine CAMT053-Datei einzulesen, zu verarbeiten und den Inhalt entweder als Kontoauszug oder als Ausgabe auf dem Browser anzeigen zu lassen. Die Formatierung und die visuelle Darstellung mit CSS ist ebenfalls eingebunden worden und gilt daher als abgeschlossen. Die Ansicht der Kontoauszüge muss allerdings noch überarbeitet, erweitert und verbessert werden.

7.2 Ausblick

Wie bereits in [Kapitel 4.5: Implementierung der DOM](#) und [7.1 Soll/Ist Vergleich](#) erwähnt, ist die Ansicht des Kontoauszuges des CAMT053 – Webservice nicht vollständig. Daher sollte der Webservice wie folgt nochmal überarbeitet beziehungsweise erweitert werden:

Die Umsetzung soll demnach durch das Durchlaufen der Tags sowie Untertags der CAMT053-Datei mit dem Package `xmlquery` mit Hilfe von mehreren Zählschleifen erfolgen. Die Schleifen sollen lediglich sowohl jeden Tag als auch Untertag durchlaufen. Zudem sollen der Tagname und der Wert des jeweiligen Tags in einer Variable oder in einem Array/Slice abgespeichert werden. Die hierfür verwendete Map soll anders als im aktuellen Stand nur den Tagnamen und den Wert enthalten und in der HTML an der entsprechenden Stelle ausgegeben werden.

Daher müssen beide Werte auf der Map entsprechend zugewiesen werden. Das Ziel ist es nicht nur wie im vorliegendem Stand die Kontonummer, IBAN, BIC, BLZ und einen beliebigen Umsatz oder Saldo anzuzeigen, sondern es sollen auch alle in der CAMT053 aufgeführten Umsätze aufgelistet werden. Daher muss die entsprechende Tabelle in den jeweiligen HTML Quellcode ebenfalls mit neuen Einträgen erweitert werden. Bezüglich der Eingabe der CAMT053 Datei gab es auch Überlegungen diese mit der Angabe von zwei CAMT053 Dateien die Eingabe eines jeweiligen Zeitraums dieser beiden Kontoauszüge zu integrieren.

Dadurch soll es möglich sein alle durchgeführten Umsätze und Salden, die innerhalb eines Zeitraums getätigt wurden, auf diese Weise aufzulisten. Momentan unterstützt der CAMT053 – Webservice nur die Eingabe einer einzelnen CAMT053 Datei. Der Webservice hat daher ein großes Spektrum mit zusätzlichen Features in Zukunft ausgebaut zu werden.

Literaturverzeichnis

Fachliteratur:

Donovan und Kernighan 2015

Donovan, A. ; Kernighan, B.: The Go Programming Language. Addison-Wesley Professional, 2015. - ISBN 978-0134190440

Ehlert 2019

Ehlert, A. Basiswissen IT-Berufe / Anwendungsentwicklung in Theorie und Praxis. Bildungsvlg EINS, 2019. - ISBN 978-3-427-01606-9

Internetquellen:

Informationen über XML und der DOM

Hery-Moßmann, Nicole: „XML – was ist das? Einfach erklärt“, unter:

https://praxistipps.chip.de/xml-was-ist-das-einfach-erklart_47836 (Abgerufen: 15. Mai 2020)

https://de.wikipedia.org/w/index.php?title=Document_Object_Model&oldid=189136298 (Abgerufen: 27. Mai 2020, 15:24 UTC)

Informationen über Go und A Tour of Go

[https://de.wikipedia.org/w/index.php?title=Go_\(Programmiersprache\)&oldid=199409542](https://de.wikipedia.org/w/index.php?title=Go_(Programmiersprache)&oldid=199409542) (Abgerufen: 27. Mai 2020, 13:30 UTC)

<https://golang.org/> (Abgerufen: 22. Mai 2020, 16:15 UTC)

<https://golang.org/doc/> (Abgerufen: 28. Mai 2020, 14:28 UTC)

<https://tour.golang.org/welcome/1> (Abgerufen: 28. Mai 2020, 14:00 UTC)

Informationen über die DSGVO

<https://dsgvo-gesetz.de/> (Abgerufen: 05. Mai 2020, 11:45 UTC)

Informationen über das CAMT Format (SEPA CAMT.053)

<https://www.hettwer-beratung.de/sepa-spezialwissen/sepa-technische-anforderungen/camt-format-camt-053/> (Abgerufen: 13. April 2020, 14:33 UTC)

Informationen über SVN und Git

<http://svnbook.red-bean.com/de/1.6/svn.intro.whatis.html> (Abgerufen: 06. März 2020, 14:08 UTC)

<https://entwickler.de/online/windowsdeveloper/git-versionskontrolle-tutorial-579885757.html> (Abgerufen: 04. März 2020, 12:15 UTC)

Informationen über Agile Softwareentwicklung

<https://www.dev-insider.de/was-ist-agile-softwareentwicklung-a-569187/> (Abgerufen: 17. April 2020, 12:25 UTC)

<https://www.business-wissen.de/artikel/agiles-projektmanagement-so-funktioniert-scrum/> (Abgerufen: 10 April 2020, 14:45 UTC)

Literaturverzeichnis

Informationen über White-Box-Testverfahren

<https://www.software-testing.academy/white-box-test.html> (Abgerufen: 29. Mai 2020, 13:00 UTC)

<https://de.wikipedia.org/wiki/White-Box-Test> (Abgerufen: 29. Mai 2020, 12:00 UTC)

Informationen über CSS

<https://wiki.selfhtml.org/wiki/CSS> (Abgerufen: 27. Mai 2020, 14:50 UTC)

Informationen über Pseudocode

<https://www.wikiwand.com/de/Pseudocode> (Abgerufen: 28. Mai 2020, 16:00 UTC)

A Anhang

A.1 Zeitplanung im Detail

Projektanalyse		6 h
1. Definieren der Projektphasen, Ressourcenplanung und des Entwicklungsprozesses	1 h	
2. Ist-Analyse durchführen (Use-Cases ermitteln, Use Case-Diagramm, Fachgespräche)	2 h	
3. Durchführung der Wirtschaftlichkeits- und Nutzwertanalyse	2 h	
4. Unterstützung der Geschäftsleitung bei der Erstellung des Lastenheftes	1 h	
Projektplanung		11 h
1. Erstellung eines Programmablaufplanes (PAP)	1 h	
2. Erstellung eines Pseudocodes für die Funktion readxmlfile() als Beispiel	1 h	
3. Projektverzeichnis in Github erstellen	0,5 h	
4. Erstellung eines Pflichtenhefts	8,5 h	
Projektumsetzung		39 h
1. Implementierung des Go Webservers, Cookies und allgemeinen Go Syntax	15 h	
2. Implementierung einer einfachen Benutzeroberfläche mit HTML	1 h	
3. Implementierung Eingabevalidierung für die XML-Dateien	1 h	
3.1 Extrahierung der entsprechenden XML-Daten implementieren	4 h	
3.2 Implementierung von xquery und xmlquery Funktionen	2 h	
3.3 Implementierung von RegExp (XML-Daten filtern in Tagnamen und Werte)	2 h	
3.2 Ausgabe der gefilterten XML-Daten auf dem Browser implementieren	2 h	
3.3 Implementierung der entsprechenden Fehlerbehandlungen	2 h	
4. Aufbereitung der XML-Daten Ausgabe mit CSS implementieren	1 h	
5. Implementierung der DOM	8 h	
6. Durchführung von Testfällen, inklusive Testverfahren	1 h	
Projektabschluss		5 h
Abnahme durch die Geschäftsleitung	1 h	
Installation und Einrichten des Webservers	4 h	
Projektdokumentationen		9 h
1. Erstellen der Projektdokumentation	7 h	
2. Erstellen der Entwicklerdokumentation	1 h	
3. Erstellen des Benutzerhandbuchs	1 h	
Gesamt		70 h

Anhang

A.2 Eingesetzte Ressourcen

Hardware

- Büroarbeitsplatz
- Heimarbeitsplatz (Coronakrise)

Software

- | | | |
|----------------------|---|---|
| • macOS | - | Betriebssystem |
| • Windows 10 Pro | - | Betriebssystem |
| • Vi Editor | - | Text Editor |
| • Notepad ++ | - | Text Editor |
| • Git/SVN | - | Versionsverwaltung |
| • Google Chrome | - | Webbrowser |
| • Mozilla Firefox | - | Webbrowser |
| • Microsoft Edge | - | Webbrowser |
| • Safari | - | Webbrowser |
| • Teamviewer | - | Fernwartungssoftware |
| • Dia | - | Diagramm, Zeichen und Illustrationsprogramm |
| • Word | - | Textverarbeitungsprogramm |
| • Confluence | - | Wikisoftware |
| • Jira | - | Webanwendung zur Projektverwaltung |
| • Ubuntu 20.04 LTS | - | Linux-Distribution für Windows |
| • Windows Powershell | - | Befehlszeilenshell |
| • PaintNet | - | Bildbearbeitungsprogramm |

Personal

- | | | |
|-----------------------------|---|--|
| • Geschäftsleitung | - | Festlegung der Anforderungen und Abnahme |
| • Buchhaltung | - | Abnahme und Verwendung |
| • Autor/Entwickler | - | Umsetzung des Projektes |
| • Mitarbeiter/Programmierer | - | Code-Review |

A.3 Programmablaufplan (PAP)

In diesem Anhang wird der Prozess des entwickelten Webservices in einem Programmablaufplan verdeutlicht.

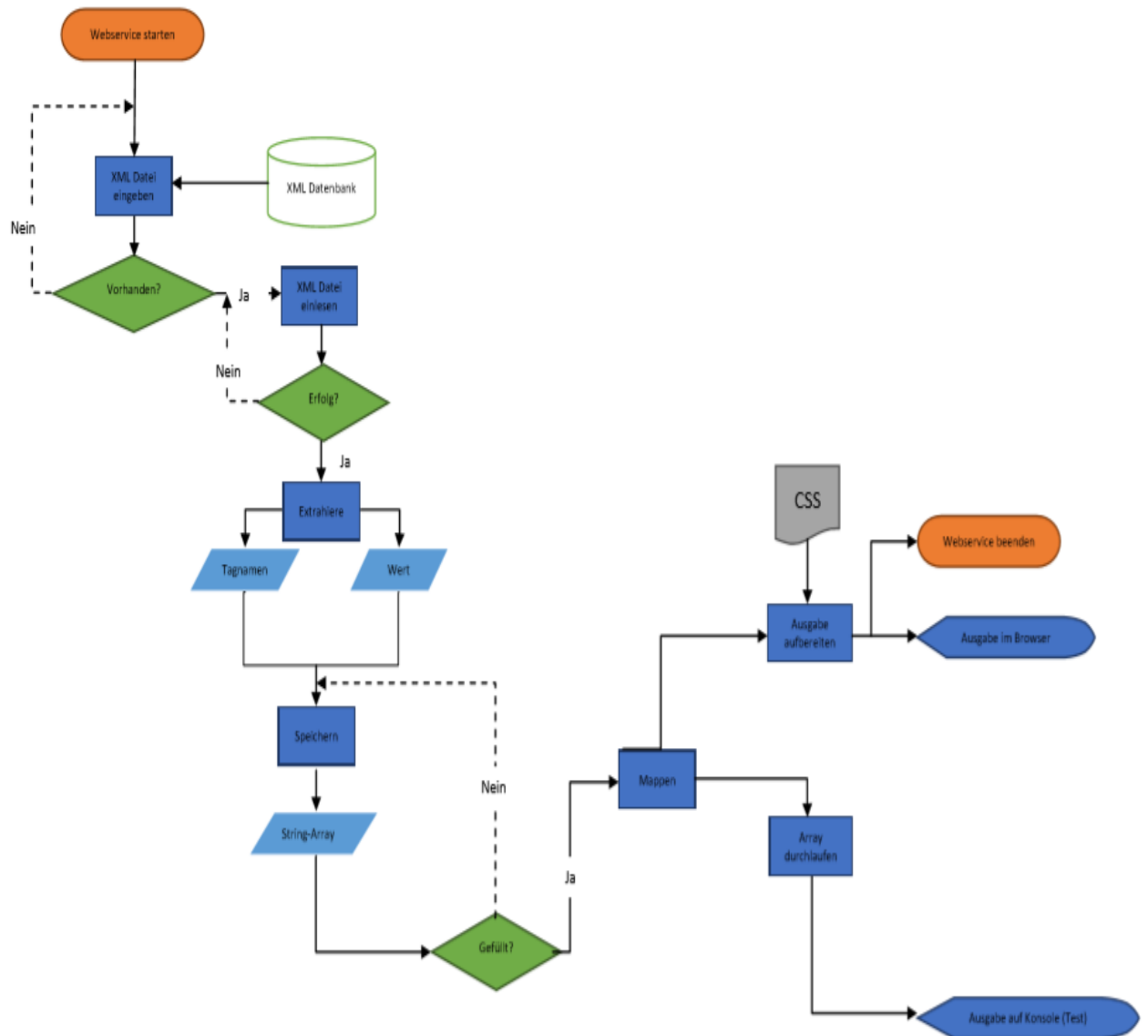


Abbildung 1: Programmablaufplan

Anhang

A.4 Lastenheft (Auszug)

Im folgenden Auszug werden alle definierten Anforderungen aus dem Lastenheft aufgelistet, die der Webservice nach Fertigstellung erfüllen muss.

Anforderungen

Folgende Anforderungen müssen vom Webservice erfüllt werden:

- Ziegruppe des Webservice ist die Buchhaltung des Unternehmens a.b.s. Rechenzentrum GmbH
- Der Webservice muss mit der Programmiersprache Go umgesetzt werden.
- Der Webservice muss ein Eingabefeld für das Angeben einer CAMT053 Datei bereitstellen.
- Der Webservice muss die Inhalte eines Kontoauszuges wie IBAN, ggf. BIC, Auszugsdatum, Saldo, Kontoumsätze, Verwendungszweck, Betrag, Buchungsdatum darstellen.
- Der Webservice muss für die Formatierung der Daten CSS verwenden.
- Der Webservice muss die Darstellung in tabellarischer Form zur Verfügung stellen.
- Such-/Druckfunktionen werden vom Browser zur Verfügung gestellt.
- Die für den eingebetteten Webserver (Webdienst) der Anwendung bestehende Zugriffsschutz/Zugangskontrolle der Anwender wird vorerst via internen Unternehmens-Firewall gelöst. Eine Nutzerverwaltung wird zu einem späteren Zeitpunkt realisiert.
- Die Bereitstellung der Anwendung soll nach erfolgreichen Abschluss der Projektarbeit erfolgen.
- Die Abnahme soll durch Mitarbeiter der Buchhaltung durchgeführt werden.
- Die Freigabe der Anwendung soll nach erfolgreichem Test im Parallelbetrieb mit der vorhandenen Lösung erfolgen.

[...]

Anhang

A.5 Pflichtenheft (Auszug)

In dem folgendem Auszug werden alle geplanten Umsetzungsschritte die auf Basis der im Lastenheft festgelegten Anforderungen basieren aufgelistet und dementsprechend beschrieben.

Umsetzung der Anforderungen

- Die Programmierung des Webservers für dieses Projektes soll mit Hilfe des Go Packages net/http erfolgen. Als Vorlage bzw. Hilfestellung wird hierbei der Quellcode aus einem vorherigen abgeschlossenen Projektes verwendet, um diesen Arbeitsschritt zu beschleunigen.
- Der Webserver soll zudem mit dem Go Package html/template ein dementsprechend festgelegtes grundlegendes HTML- Layout zur Verfügung stellen.
- Die Eingabe der CAMT053-Datei soll hierbei in Form eines Eingabefeldes erfolgen. Daher wird bei der Umsetzung der grundlegende HTML-Quellcode mit zusätzlichen Formular Eigenschaften und Funktionen von html.Forms erweitert.
- Die Formeingabe soll zudem nur die Eingabe des Namens der CAMT053-Datei umfassen. Daher wird die CAMT053-Datei anschließend durch den Webservice gezielt aus einem Verzeichnis oder einer Datenbank herausgesucht.
- Für das Parsen der Inhalte, soll auf die xmlquery Variante des Go Packages xquery zugegriffen werden. Dazu wird die CAMT053-Datei gezielt auf Grphdr (Group Header) und Stmt (Statement) durch den Webservice durchlaufen, wodurch die entsprechenden Daten extrahiert werden.
- Die Extrahierung der Daten soll zudem mit dem Package regexp vereinfacht werden.
- Die Daten sollen anschließend in Form einer Map zusammengefasst werden und zudem mit dem Go Package fmt auf dem Browser in jeweils zwei Ansichten als Kontoauszug und einfache Inhaltsausgabe ausgegeben werden.
- Die Ausgabe soll dann mit CSS visuell auf Basis eines mitgelieferten Kontoauszuges im PDF-Formates angepasst und dementsprechend verbessert werden.

[...]

Anhang

A.7 Go XML Demonstration

sandbox.go

Imports off Syntax off

```

1 package main
2
3 import (
4     "encoding/xml"
5     "fmt"
6 )
7
8 type pruefling struct {
9     Prueflingsnummer string `xml:"prf_nr"`
10    IHK_Name          string `xml:"ihk_name"`
11    Name              string `xml:"name"`
12    Alter             string `xml:"alter"`
13    Ort               string `xml:"ort"`
14    Ausbildungsberuf string `xml:"aberuf"`
15    Firma             string `xml:"firma"`
16 }
17
18 func main() {
19     pruefling := &pruefling{
20         Prueflingsnummer: "100000",
21         IHK_Name:          "IHK Musterstadt",
22         Name:              "Max",
23         Alter:             "27",
24         Ort:               "Musterstadt",
25         Ausbildungsberuf: "Fachinformatiker Anwendungsentwicklung",
26         Firma:             "Muster GmbH",
27     }
28
29     xml, _ := xml.MarshalIndent(pruefling, "", " ")
30
31     fmt.Println(string(xml))
32
33 }
34

```

Reset Format Run

```

<pruefling>
  <prf_nr>100000</prf_nr>
  <ihk_name>IHK Musterstadt</ihk_name>
  <name>Max</name>
  <alter>27</alter>
  <ort>Musterstadt</ort>
  <aberuf>Fachinformatiker Anwendungsentwicklung</aberuf>
  <firma>Muster GmbH</firma>
</pruefling>

Program exited.

```

Abbildung 3: Go XML Demonstration (Quellcode)

A.8 Grundlegender Go Webserver (Beispiel)

```

1 package main
2
3 import (
4     "fmt"
5     "net/http"
6     "html/template"
7 )
8
9 func main() {
10     http.HandleFunc("/", handler)
11     log.Fatal(http.ListenAndServe(":3000", nil))
12 }
13
14 func handler(w http.ResponseWriter, r *http.Request) {
15     fmt.Fprintf(w, "This is a basic Go Webserver")
16 }
17
18

```

Abbildung 4: Go grundlegender Webserver (main.go)

```

1 package main
2
3 import (
4     "fmt"
5     "net/http"
6     "html/template"
7 )
8
9 func main() {
10     http.HandleFunc("/", handler)
11     log.Fatal(http.ListenAndServe(":3000", nil))
12 }
13
14 func handler(w http.ResponseWriter, r *http.Request) {
15     tmpl, err := template.New("Webseite").Parse(webseite)
16     err = tmpl.Execute(w, nil)
17
18     if err != nil {
19         http.Error(w, err.Error(), http.StatusInternalServerError)
20     }
21 }
22

```

Abbildung 5: Go grundlegender Webserver (main.go mit HTML Template)

```

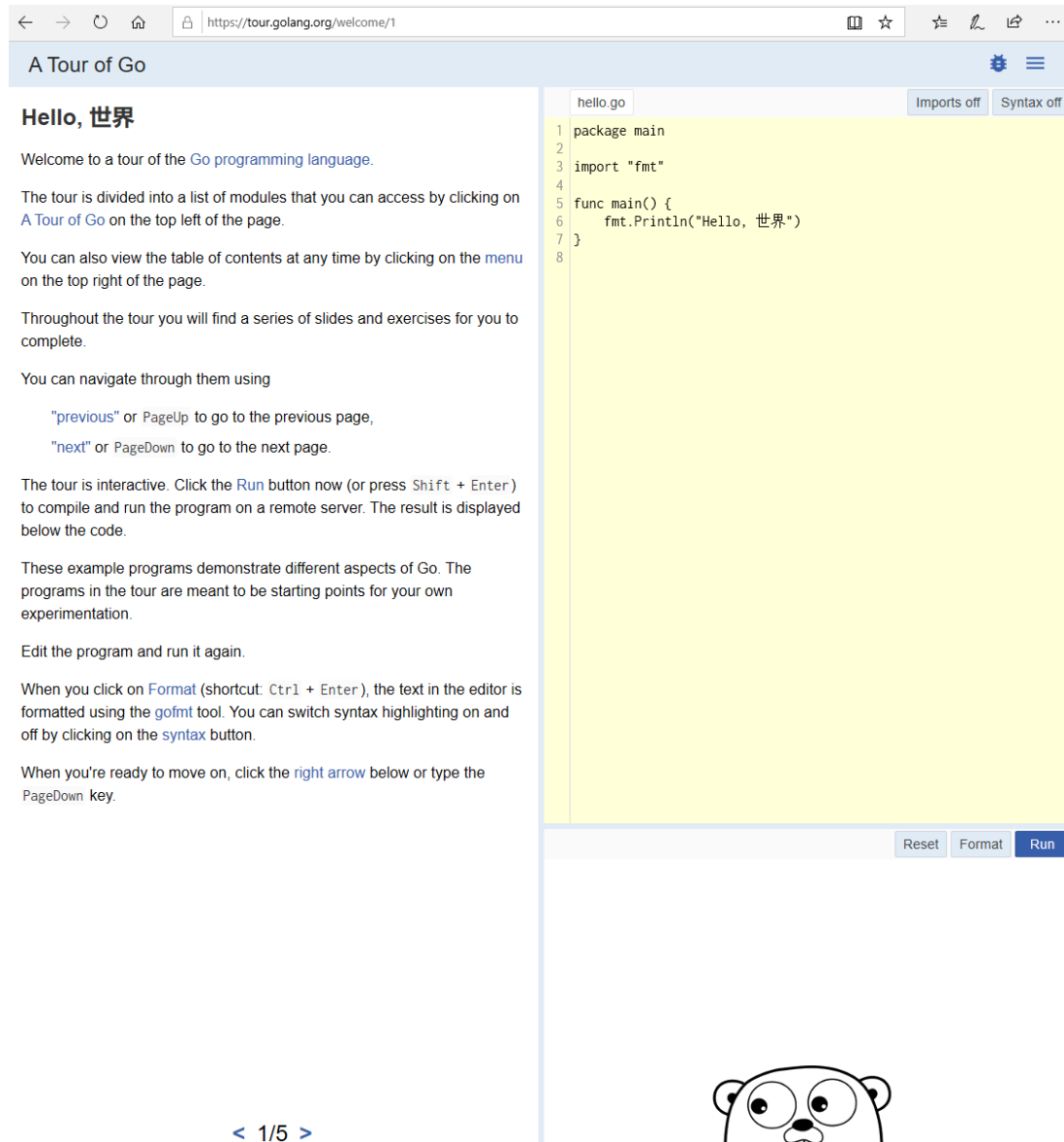
1 package main
2
3 const webseite = `
4     <html>
5         <center>
6         <head>
7             <center>
8                 <title>Grundlegender Go Webserver</title>
9                 <h1 style = "color:Black ; font-family:Arial">Grundlegender Go Webserver</h1>
10            </center>
11        </head>
12        <body>
13            <center>
14                <p>Dieser Webserver dient als Demonstration</p>
15            </center>
16        </body>
17    </html>
18 `
19
20

```

Abbildung 6: Go grundlegender Webserver (webseite.go)

Anhang

A.9 A Tour of Go



Hello, 世界

Welcome to a tour of the [Go programming language](#).

The tour is divided into a list of modules that you can access by clicking on [A Tour of Go](#) on the top left of the page.

You can also view the table of contents at any time by clicking on the [menu](#) on the top right of the page.

Throughout the tour you will find a series of slides and exercises for you to complete.

You can navigate through them using

- "previous" or PageUp to go to the previous page,
- "next" or PageDown to go to the next page.

The tour is interactive. Click the [Run](#) button now (or press Shift + Enter) to compile and run the program on a remote server. The result is displayed below the code.

These example programs demonstrate different aspects of Go. The programs in the tour are meant to be starting points for your own experimentation.

Edit the program and run it again.

When you click on [Format](#) (shortcut: Ctrl + Enter), the text in the editor is formatted using the [gofmt](#) tool. You can switch syntax highlighting on and off by clicking on the [syntax](#) button.

When you're ready to move on, click the [right arrow](#) below or type the PageDown key.

hello.go Imports off Syntax off

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello, 世界")
7 }
8
```

Reset Format Run

< 1/5 >




Abbildung 7: Willkommensseite zu A Tour of Go

Anhang

A.10 Pseudocode (Beispiel)

```
Algorithmus readxmlfile(check, filename, r, w)

Ausgabe: Gebe Test2 auf der CMD aus (Kontrolle zum Funktionsstart)

Deklariere Variablen pwd, err und Zuweisung des Dateipfades der XML Datei

wenn Dateipfad ist falsch dann

    Ausgabe: XML Datei nicht gefunden

    beende

wenn_ende

Ausgabe: Dateipfad

Deklariere Variablen getxmlcontent, _ und öffne XML Datei

Deklariere parsexmlcontent, err und parse die XML Datei

wenn Parsen ist Fehlgeschlagen dann

    Ausgabe: Parsen der XML Datei ist fehlgeschlagen

    beende

wenn_ende

Deklariere Variable grphdr und suche das erste Element (MsgId) in GrpHdr

Deklariere Array grphdrarr und Aufruf des Algorithmus getxmlvalues(grphdr, r, w)

Deklariere Variable stmt und suche das erste Element (Id) in Stmt

Deklariere Array stmtarr und Aufruf des Algorithmus getxmlvalues(stmt, r, w)

Ausgabe: Gebe das Array stmtarr auf der CMD aus (Kontrolle)

wenn Check ist true dann

    Aufruf des Algorithmus printauszugscontent(w, grphdrarr, stmtarr)

sonst

    Aufruf des Algorithmus printxmlcontent(w, grphdrarr, stmtarr)

    beende

wenn_ende

Gebe den Rückgabewert nil zurück

Ende
```

Anhang

A.11 main.go-Quellcode

```

1  /*
2  #####
3  Projekt: CAMT053-Webservice
4  Autor: Marvin Viedt
5  Copyright: a.b.s. Rechenzentrum GmbH
6  Version: 1.0 (still WIP)
7  Release: Sommer 2020
8  #####
9  */
10
11 package main
12
13 import (
14     "fmt"
15     "log"
16     "errors"
17     "net/http"
18     "html/template"
19     "time"
20     "strings"
21     "regexp"
22     "os"
23     "github.com/antchfx/xquery/xml"
24     "path/filepath"
25 )
26
27 var (
28     Map = make(map[string]string)
29 )
30
31 func main() {
32     log.Println("Server wird gestartet")
33
34     http.HandleFunc("/", handler)
35     http.HandleFunc("/Camt053/", kauszughandler)
36
37     http.Handle("/static/", http.StripPrefix("/static/", http.FileServer(http.Dir("static"))))
38
39     err := (http.ListenAndServe(":3000", nil))
40
41     log.Fatal(err)
42 }
43
44 func handler(w http.ResponseWriter, r *http.Request) {
45     t, err := template.New("Webservice").Parse(webseite)
46     fmt.Println("CAMT053-Webservice wird ausgeführt")
47
48     expiration := time.Now().Add(10 * time.Second)
49     cookie := http.Cookie{Name: "Session", Value: "Session", Expires: expiration}
50     http.SetCookie(w, &cookie)
51
52     if err != nil {
53         http.Error(w, err.Error(), http.StatusInternalServerError)
54     }
55
56     err = t.Execute(w, nil)
57
58     if err != nil {
59         http.Error(w, err.Error(), http.StatusInternalServerError)
60     }
61 }
62
63 func kauszughandler (w http.ResponseWriter, r *http.Request) {
64
65     if r.Method != http.MethodPost {
66         http.Error(w, "Das ist nicht gültig", http.StatusMethodNotAllowed)
67         return
68     }
69
70     formvalue, err := getformvalue(r)
71
72     if err != nil {
73         http.Error(w, fmt.Sprintf("Formeingabe(n) konnte(n) nicht geladen oder bearbeitet werden: %s", err), http.StatusBadRequest)
74         return
75     }
76
77     if r.URL.Path == "/Camt053/Kontoauszugs-View" {

```

Anhang

```

79     if r.URL.Path == "/Camt053/Kontoauszugs-View" {
80         var kontoauszug = true
81         fmt.Println("Test 1")
82         err = readxmlfile(kontoauszug, formvalue, r, w)
83         fmt.Println("\nTest 3")
84         if err != nil {
85             http.Error(w, "Einlesen der XML Datei ist fehlgeschlagen", http.StatusBadRequest)
86             return
87         }
88     }
89
90     if r.URL.Path == "/Camt053/XML-View" {
91         var kontoauszug = false
92         fmt.Println("Test 1")
93         err = readxmlfile(kontoauszug, formvalue, r, w)
94         fmt.Println("\nTest 3")
95         if err != nil {
96             http.Error(w, "Einlesen der XML Datei ist fehlgeschlagen", http.StatusBadRequest)
97             return
98         }
99     }
100    http.Redirect(w, r, "/", http.StatusFound)
101 }
102
103 func getformvalue(r *http.Request) (string, error) {
104     err := r.ParseForm()
105
106     if err != nil {
107         log.Println("Error:", err)
108     }
109
110     formvalue := r.FormValue("XMLFile")
111     fmt.Println(formvalue)
112
113     if err != nil {
114         errors.New(fmt.Sprintf("Parsen des Eingabefeldes XMLFile fehlgeschlagen: %s", err))
115     }
116
117     return formvalue, nil
118 }
119
120 func checkstring(s string) bool {
121
122     if (s >= "A" && s <= "Z") || (s >= "0" && s <= "9") || (s == "\n") {
123         return true
124     }
125
126     return false
127 }
128
129 func gettagname(get_tag string, w http.ResponseWriter) string {
130
131     value := regexp.MustCompile("(<.*?>))")
132
133     tags := value.FindString(get_tag)
134     rmrightbracket := strings.ReplaceAll(tags, ">", "")
135     tagname := strings.ReplaceAll(rmrightbracket, "<", "")
136
137     return tagname
138 }
139
140 func getxmlvalues(xmlcontent *xmlquery.Node, r *http.Request, w http.ResponseWriter) [200]string {
141
142     var i int
143     var xmlarr [200]string
144     for ; xmlcontent != nil; xmlcontent = xmlcontent.NextSibling {
145
146         get_xmlcontent := xmlcontent.InnerText()
147         get_xmltag := xmlcontent.OutputXML(true)
148
149         trim_xmlcontent := strings.Trim(get_xmlcontent, "\t \n")
150         rmnewlines := regexp.MustCompile("\n\n\n")
151         content_rmnewlines := rmnewlines.ReplaceAllString(trim_xmlcontent, "")
152         rmtab := regexp.MustCompile("\t")
153         content_rmtab := rmtab.ReplaceAllString(content_rmnewlines, "")
154         rmspaces2 := regexp.MustCompile("\n\n")
155         content_rmspaces2 := rmspaces2.ReplaceAllString(content_rmtab, "\n")
156         rmspaces3 := regexp.MustCompile("\n\n\n")
157         content_rmspaces3 := rmspaces3.ReplaceAllString(content_rmspaces2, "\n")

```

Anhang

```

157     content_rmspaces3 := rmspaces3.ReplaceAllString(content_rmspaces2, "\n")
158
159     check := checkstring(content_rmspaces3)
160
161     if check == true {
162
163         //An dieser Stelle alle Elemente und vllt der Tag Ausgabe soll die Zuweisung sein.
164
165         tagname := gettagname(get_xmltag, w)
166         tagvalue := content_rmspaces3
167
168         Map["tagname"] = tagname
169
170         //fmt.Fprintf(w, "%s\n", tagname)
171         //fmt.Fprintf(w, "%s\n", tagvalue)
172
173
174         xmlarr[i] = tagvalue
175         i++
176
177     }
178 }
179
180 //fmt.Fprintf(w, "%s\n", xmlarr)
181
182 return xmlarr
183 }
184
185 func readxmlfile(check bool, filename string, r *http.Request, w http.ResponseWriter) error {
186
187     fmt.Println("Test 2\n")
188
189     pwd, err := filepath.Abs("./XML/" + filename + ".xml")
190     if err != nil {
191         fmt.Println("Die XML Datei konnte nicht gefunden werden: %s", err)
192         return err
193     }
194
195     fmt.Println("Der Dateipfad ist: %s\n", pwd)
196     getxmlcontent, _ := os.Open(pwd)
197
198     parsexmlcontent, err := xmlquery.Parse(getxmlcontent)
199     if err != nil {
200         fmt.Println("Öffnen der XML Datei ist fehlgeschlagen: %s", err)
201         return err
202     }
203
204     fmt.Println("-----")
205
206     grphdr := xmlquery.FindOne(parsexmlcontent, "//BkToCstmrStmt/GrpHdr/MsgId")
207     grphdrarr := getxmlvalues(grphdr, r, w)
208
209     stmt := xmlquery.FindOne(parsexmlcontent, "//BkToCstmrStmt/Stmt/Id")
210     stmtarr := getxmlvalues(stmt, r, w)
211
212     //fmt.Fprintf(w, "%s\n", Map)
213     fmt.Println("%s\n", stmtarr)
214
215     if check == true {
216         printauszugcontent(w, grphdrarr, stmtarr)
217     }
218     if check == false {
219         printxmlcontent(w, grphdrarr, stmtarr)
220     }
221
222     return nil
223 }
224
225 func printauszugcontent(w http.ResponseWriter, grphdrarr [200]string, stmtarr [200]string) {
226     t := template.Must(template.New("Kontoauszugs-View").Parse(KontoauszugsView))
227
228     Nm := strings.Fields(grphdrarr[2])
229     IBAN := strings.Fields(stmtarr[4])
230     stmt1 := strings.Fields(stmtarr[6])
231
232     stmt2 := strings.Fields(stmtarr[8])
233
234     //fmt.Fprintf(w, "%s\n", IBAN)
235

```

Anhang

```

235     Map["MsgId"] = grphdrrarr[0]
236     Map["CreDtIm"] = grphdrrarr[1]
237     Map["Nm"] = Nm[0] + " " + Nm[1] + " " + Nm[2] + " " + Nm[3]
238
239     Map["IBAN"] = IBAN[0]
240
241     CreDtIm := grphdrrarr[1]
242
243     Map["CreateDate"] = getdate(CreDtIm)
244     Map["Buchdt"] = getdate(stmt2[3])
245     Map["valdt"] = getdate(stmt2[4])
246
247     kntnr := IBAN[0]
248     Map["Kntnr"] = string(kntnr[len(kntnr)-7:])
249
250     blz := strings.Fields(strings.Trim(IBAN[0], "DE25 7404000"))
251     Map["BLZ"] = blz[0]
252
253     Map["BIC"] = IBAN[12]
254     Map["Kntst"] = stmt1[1]
255
256     Map["ums"] = stmt2[0]
257
258     Map["zweck"] = strings.ToUpper(stmt2[len(stmt2)-1])
259     Map["zwecknm"] += stmt2[41] + " " + stmt2[42] + " " + stmt2[43]
260     for i := 47; i < len(stmt2)-1; i++ {
261         Map["info"] += " " + stmt2[i]
262     }
263     Map["kndref"] = stmt2[11]
264     Map["manref"] = stmt2[13]
265
266     err := t.Execute(w, Map)
267     if err != nil {
268         http.Error(w, err.Error(), http.StatusInternalServerError)
269     }
270
271     Map = make(map[string]string)
272 }
273
274 func printxmlcontent(w http.ResponseWriter, grphdrrarr [200]string, stmtarr [200]string) {
275     t := template.Must(template.New("XML-View").Parse(XMLView))
276
277     var grphdrrcontent string
278     var stmtcontent string
279     var newgrphdrrarr []string
280     var newstmtarr []string
281
282     newgrphdrrarr = remove_empty_value (grphdrrarr)
283
284     for i:=0; i < len(newgrphdrrarr); i++ {
285         grphdrrcontent += "\n" + newgrphdrrarr[i]
286     }
287
288     newstmtarr = remove_empty_value (stmtarr)
289
290     for i:=0; i < len(newstmtarr); i++ {
291         stmtcontent += "\n" + newstmtarr[i]
292     }
293
294     Map["grphdrr"] = grphdrrcontent
295     Map["stmt"] = stmtcontent
296
297     err := t.Execute(w, Map)
298     if err != nil {
299         http.Error(w, err.Error(), http.StatusInternalServerError)
300     }
301 }
302
303 func remove_empty_value (arr [200]string) []string {
304     var newarr []string
305     for _, str := range arr {
306         if str != "" {
307             newarr = append(newarr, str)
308         }
309     }
310     return newarr
311 }
312
313 }

```

```

314 func getdate(datearr string) string {
315     getdate := string(datearr[0:10])
316     layout := "2006-01-02"
317     resdate, _ := time.Parse(layout, getdate)
318     dateString := resdate.Format("2.01.2006");
319
320     return dateString
321 }
322
323 }
324
325

```

A.12 webseite.go (Go-Quellcode)

```

1 package main
2
3 const webseite = `
4 <html>
5   <center>
6     <head>
7       <div class="header-container">
8         <center>
9           <link rel="stylesheet" type="text/css" href="/static/css/Webseite.css">
10          <title>CAMT053-Webservice</title>
11          <h1>CAMT053 - Webservice</h1>
12        </center>
13      </div>
14    </head>
15    <body>
16      <br></br>
17      <div class="container">
18        <form class="text-center" action="/CAMT053" method="post">
19          <fieldset>
20            <legend>Bitte die XML Datei angeben:</legend>
21            <label for="XMLFile">XML Datei:</label>
22            <input id="XMLFile" name="XMLFile" type="text" size="60" placeholder="Dateiname">
23          </fieldset>
24          <br></br>
25          <button type="submit" name="Einlesen" formaction="/Camt053/Kontoauszugs-View">Kontoauszug anzeigen</button>
26          <button type="submit" name="Einlesen" formaction="/Camt053/XML-View">XML anzeigen</button>
27        </form>
28      </div>
29    </body>
30  </center>
31 </html>
32 `

```

A.13 Webseite.css (css-Quellcode)

```

1 .header-container {
2   width: 100%;
3   border: 3px solid C42A70;
4   margin-left: auto;
5   margin-right: auto;
6 }
7
8 .container {
9   width: 100%;
10  margin-left: auto;
11  margin-right: auto;
12 }
13
14 .box {
15   float: left;
16   padding: 20px;
17   background: #eee;
18   margin-left: auto;
19   margin-right: auto;
20   box-sizing: border-box;
21   width: 30%;
22 }
23
24 .box3 {
25   float: right;
26   padding: 20px;
27   background: #eee;
28   margin-left: auto;
29   margin-right: auto;
30   box-sizing: border-box;
31   width: 70%;
32 }
33
34 button {
35   float: center;
36   width: 300px;
37   height: 40px;
38   font-weight: bold;
39   font-size: 115%;
40   color: darkmagenta
41 }
42
43 legend {
44   color: C42A70;
45   font-family: Arial;
46 }
47
48 fieldset {
49   border: 1px solid C42A70;
50   font-family: Arial;
51 }
52
53 label {
54   color: black;
55   font-style: Bold;
56   font-size: 83%;
57   font-family: Arial;
58 }
59
60 h1 {
61   color: C42A70;
62   font-family: Arial;
63 }
64
65 h4 {
66   color: C42A70;
67   font-family: Arial;
68 }
69
70 body {
71   background-color: C189A3;
72 }
73
74 p {
75   text-align: left;
76   font-family: Arial;
77 }
78

```

A.14 Kontoauszugs-View.go (Go-Quellcode)

```

1 package main
2
3 const KontoauszugsView = `
4     <html>
5         <center>
6         <head>
7             <div class="header-container">
8                 <link rel="stylesheet" type="text/css" href="/static/css/Kontoauszugs-View.css">
9                 <center>
10                    <title>CAMT053 - Webservice</title>
11                    <h1>CAMT053 - Webservice</h1>
12                </center>
13            </div>
14        </head>
15        </center>
16        <body>
17            <h4>Kontoinformationen</h4>
18            <div class="kontoinfo-container">
19                <div class="box2">
20                    <h4>Konto-Nr.</h4>
21                    <p>{{.Kntnr}}</p>
22                    <h4>BLZ</h4>
23                    <p>{{.BLZ}}</p>
24                </div>
25                <div class="box3">
26                    <h4>IBAN</h4>
27                    <p>{{.IBAN}}</p>
28                    <h4>BIC</h4>
29                    <p>{{.BIC}}</p>
30                </div>
31                <div class="box4">
32                    <h3>Kontostand am {{.CreateDate}}</h3>
33                </div>
34                <div class="box4">
35                    <h3>{{.Kntst}} EUR</h3>
36                </div>
37            <div style="clear:both;"></div>
38        </div>
39        <h4>Auswahlkriterien</h4>
40        <div class="kriterien-container">
41            <p>{{.CreateDate}}</p>
42        </div>
43        <h4>Umsätze im gewählten Zeitraum</h4>
44        <div class="table-container">
45            <table>
46                <tr>
47                    <th>Buchung</th>
48                    <th>Valuta</th>
49                    <th>Verwendungszweck</th>
50                    <th>Betrag</th>
51                </tr>
52                <tr>
53                    <td>{{.buchdt}}</td>
54                    <td>{{.valdt}}</td>
55                    <td>
56                        <b>{{.zweck}}</b>
57                        <br>
58                        {{.zwecknm}}
59                        <br>
60                        {{.info}}
61                        <br>
62                        KUNDENREFERENZ {{.kndref}}
63                        <br>
64                        MANDATSREFERENZ {{.manref}}
65                    </td>
66                    <td>{{.ums}} EUR</td>
67                </tr>
68            </table>
69        </div>
70        <br><br>
71        <center>
72            <form class="text-center" action="/Camt053/Kontoauszugs-View" method="post">
73                <button type="submit" name="Zurück" formaction="/">Zurück</button>
74            </form>
75        </center>
76    </body>
77 </html>
78 `

```

Anhang

A.15 Kontoauszugs-View.css (css-Quellcode)

```

1  .header-container {
2      width: 100%;
3      border: 3px solid C42A70;
4      margin-left: auto;
5      margin-right: auto;
6  }
7
8  .kontoinfo-container {
9      width: 100%;
10     border: 3px solid C42A70;
11     background-color: #eee;
12     margin-left: auto;
13     margin-right: auto;
14 }
15
16 .kriterien-container {
17     width: 100%;
18     border: 3px solid C42A70;
19     margin-left: auto;
20     margin-right: auto;
21 }
22
23 .table-container {
24     width: 100%;
25     border: 3px solid C42A70;
26     margin-left: auto;
27     margin-right: auto;
28 }
29
30 .box2 {
31     float: left;
32     padding: 20px;
33     background: #eee;
34     margin-left: auto;
35     margin-right: auto;
36     box-sizing: border-box;
37     width: 30%;
38 }
39
40 .box3 {
41     float: right;
42     padding: 20px;
43     background: #eee;
44     margin-left: auto;
45     margin-right: auto;
46     box-sizing: border-box;
47     width: 70%;
48 }
49
50 .box4 {
51     float: left;
52     padding: 20px;
53     background: #eee;
54 }
55
56 h1 {
57     color: C42A70;
58     font-family: Arial;
59 }
60
61 h3 {
62     font-family: Arial;
63 }
64
65 h4 {
66     color: C42A70;
67     font-family: Arial;
68 }
69
70 body {
71     background-color: C189A3;
72 }
73
74 button {

```

```

73
74 button {
75     float: center;
76     width: 200px;
77     height: 40px;
78     font-weight: bold;
79     font-size: 115%;
80     color: darkmagenta;
81 }
82
83 table {
84     width: 100%;
85 }
86
87 table, th, td {
88     border: 1px solid C42A70;
89     font-family: Arial;
90     background: #eee;
91 }
92
93 th {
94     background-color: BFAAB2;
95     color: #eee;
96 }
97
98 p {
99     text-align: left;
100    font-family: Arial;
101 }
102

```


A.16 XML-View.go (Go-Quellcode)

```

1 package main
2
3 const XMLView = `
4     <html>
5         <center>
6             <head>
7                 <div class="header-container">
8                     <link rel="stylesheet" type="text/css" href="/static/css/XML-View.css">
9                     <center>
10                        <title>CAMT053 - Webservice</title>
11                        <h1>CAMT053 - Webservice</h1>
12                    </center>
13                </div>
14            </head>
15            </center>
16            <body>
17                <h4>XML-View</h4>
18                <div class="xmlview-container">
19                    <h4>GRPHDR:</h4>
20                    <pre>{{.grphdr}}</pre>
21                    <br>
22                    <h4>STMT:</h4>
23                    <pre>{{.stmt}}</pre>
24                </div>
25                <br>
26                <center>
27                    <form class="text-center" action="/Camt053/XML-View" method="post">
28                        <button type="submit" name="Zurück" formaction="/">Zurück</button>
29                    </form>
30                </center>
31            </body>
32        </html>
33 `

```

A.17 XML-View.css (css-Quellcode)

```

1
2 .header-container {
3     width: 100%;
4     border: 3px solid C42A70;
5     margin-left: auto;
6     margin-right: auto;
7 }
8
9 .xmlview-container {
10     width: 100%;
11     border: 3px solid C42A70;
12     background-color: #eee;
13     margin-left: auto;
14     margin-right: auto;
15 }
16
17 h1 {
18     color: C42A70;
19     font-family: Arial;
20 }
21
22 h3 {
23     font-family: Arial;
24 }
25
26 h4 {
27     color: C42A70;
28     font-family: Arial;
29 }
30
31 body {
32     background-color: C189A3;
33 }
34
35 button {
36     float: center;
37     width: 200px;
38     height: 40px;
39     font-weight: bold;
40     font-size: 115%;
41     color: darkmagenta;
42 }
43
44 p {
45     text-align: left;
46     font-family: Arial;
47 }
48

```

A.18 Browseransicht: Hauptseite + Eingabefeld

CAMT053 - Webservice

— Bitte die XML Datei angeben: —
 XML Datei:

Kontoauszug anzeigen

XML anzeigen

A.19 Browseransicht: Kontoauszugs-View

CAMT053 - Webservice

Kontoinformationen

Konto-Nr.

IBAN

BLZ

BIC

Kontostand am**EUR**

Auswahlkriterien

Umsätze im gewählten Zeitraum

Buchung	Valuta	Verwendungszweck	Betrag

Zurück

A.20 Browseransicht: XML-View

CAMT053 - Webservice

XML-View

GRPHDR:

STMT:

Zurück

A.21 CAMT053 Datei (Auszug)

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <Document xmlns="urn:iso:std:iso:2002:tech:xsd:camt.053.001.02" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
3    <BkToCstmRstmt>
4      <GrpHdr>
5        <MsgId>[REDACTED]</MsgId>
6        <CreDtTm>[REDACTED]</CreDtTm>
7        <MsgRcpt>
8          <Nm>[REDACTED]</Nm>
9          <PstlAdr>
10             <AdrLine>[REDACTED]</AdrLine>
11             <AdrLine>[REDACTED]</AdrLine>
12          </PstlAdr>
13        </MsgRcpt>
14        <MsgPgntn>
15          <PgNb>[REDACTED]</PgNb>
16          <LastPgInd>[REDACTED]</LastPgInd>
17        </MsgPgntn>
18      </GrpHdr>
19      <Stmt>
20        <Id>[REDACTED]</Id>
21        <ElectrncSegNb>[REDACTED]</ElectrncSegNb>
22        <CreDtTm>[REDACTED]</CreDtTm>
23        <FrToDt>
24          <FrDtTm>[REDACTED]</FrDtTm>
25          <ToDtTm>[REDACTED]</ToDtTm>
26        </FrToDt>
27        <Acct>
28          <Id>
29            <IBAN>[REDACTED]</IBAN>
30          </Id>
31          <Ccy>[REDACTED]</Ccy>
32          <Ownr>
33            <Nm>[REDACTED]</Nm>
34            <PstlAdr>
35              <AdrLine>[REDACTED]</AdrLine>
36              <AdrLine>[REDACTED]</AdrLine>
37            </PstlAdr>
38          </Ownr>
39          <Svcr>
40            <FinInstnId>
41              <BIC>[REDACTED]</BIC>
42              <Nm>[REDACTED]</Nm>
43              <Othr>
44                <Id>[REDACTED]</Id>
45                <Issr>[REDACTED]</Issr>
46              </Othr>
47            </FinInstnId>
48          </Svcr>
49        </Acct>
50        <Bal>
51          <Tp>
52            <CdOrPrtry>
53              <Cd>[REDACTED]</Cd>
54            </CdOrPrtry>
55          </Tp>
56          <Amt Ccy="[REDACTED]">[REDACTED]</Amt>
57          <CdtDbtInd>[REDACTED]</CdtDbtInd>
58          <Dt>
59            <Dt>[REDACTED]</Dt>
60          </Dt>
61        </Bal>
62        <Bal>
63          <Tp>
64            <CdOrPrtry>
65              <Cd>[REDACTED]</Cd>
66            </CdOrPrtry>

```

Anhang

```

67      </Tp>
68      <Amt Ccy=███></Amt>
69      <CdtDdtInd>███</CdtDdtInd>
70      <Dt>███</Dt>
71      <Dt>███</Dt>
72      </Dt>
73      </Bal>
74      <Bal>
75      <Tp>
76      <CdOrPrtry>
77      <Cd>███</Cd>
78      </CdOrPrtry>
79      </Tp>
80      <Amt Ccy=███></Amt>
81      <CdtDdtInd>███</CdtDdtInd>
82      <Dt>███</Dt>
83      <Dt>███</Dt>
84      </Dt>
85      </Bal>
86      <Ntry>
87      <Amt Ccy=███></Amt>
88      <CdtDdtInd>███</CdtDdtInd>
89      <Sts>███</Sts>
90      <BookgDt>
91      <Dt>███</Dt>
92      </BookgDt>
93      <ValDt>
94      <Dt>███</Dt>
95      </ValDt>
96      <AcctSvcrRef>███</AcctSvcrRef>
97      <BkTxCd>
98      <Domn>
99      <Cd>███</Cd>
100     <Fmly>
101     <Cd>███</Cd>
102     <SubFmlyCd>███</SubFmlyCd>
103     </Fmly>
104     </Domn>
105     <Prtry>
106     <Cd>███</Cd>
107     <Issr>███</Issr>
108     </Prtry>
109     </BkTxCd>
110     <NtryDtls>
111     <TxDtls>
112     <Refs>
113     <EndToEndId>███</EndToEndId>
114     <TxId>███</TxId>
115     <MndtId>███</MndtId>
116     <ClrSysRef>███</ClrSysRef>
117     </Refs>
118     <BkTxCd>
119     <Domn>
120     <Cd>███</Cd>
121     <Fmly>
122     <Cd>███</Cd>
123     <SubFmlyCd>███</SubFmlyCd>
124     </Fmly>
125     </Domn>
126     <Prtry>
127     <Cd>███</Cd>
128     <Issr>███</Issr>
129     </Prtry>
130     </BkTxCd>
131     <RltdPties>
132     <Dptr>

```

Anhang

```

132 <Dbtr>
133   <Nm> [REDACTED] </Nm>
134   <PstlAdr>
135     <Ctry> [REDACTED] </Ctry>
136     <AdrLine> [REDACTED] </AdrLine>
137     <AdrLine> [REDACTED] </AdrLine>
138   </PstlAdr>
139 </Dbtr>
140 <DbtrAcct>
141   <Id>
142     <IBAN> [REDACTED] </IBAN>
143   </Id>
144 </DbtrAcct>
145 <Cdtr>
146   <Nm> [REDACTED] </Nm>
147   <PstlAdr>
148     <Ctry> [REDACTED] </Ctry>
149     <AdrLine> [REDACTED] </AdrLine>
150     <AdrLine> [REDACTED] </AdrLine>
151   </PstlAdr>
152   <Id>
153     <PrvtId>
154       <Othr>
155         <Id> [REDACTED] </Id>
156       </Othr>
157     </PrvtId>
158   </Id>
159 </Cdtr>
160 <CdtrAcct>
161   <Id>
162     <IBAN> [REDACTED] </IBAN>
163   </Id>
164 </CdtrAcct>
165 <UltmtCdtr>
166   <Nm> [REDACTED] </Nm>
167 </UltmtCdtr>
168 </RltdPties>
169 <RltdAgts>
170   <DbtrAgt>
171     <FinInstnId>
172       <BIC> [REDACTED] </BIC>
173     </FinInstnId>
174   </DbtrAgt>
175   <CdtrAgt>
176     <FinInstnId>
177       <BIC> [REDACTED] </BIC>
178     </FinInstnId>
179   </CdtrAgt>
180 </RltdAgts>
181 <Purp>
182   <Cd> [REDACTED] </Cd>
183 </Purp>
184 <RmtInf>
185   <Ustrd> [REDACTED] </Ustrd>
186 </RmtInf>
187 </TxDtls>
188 </WtryDtls>
189 <AddtlNtryInf> [REDACTED] </AddtlNtryInf>
190 </Ntry>
191 <Ntry>
192   <Amt Ccy= [REDACTED] </Amt>
193   <CdtDbtInd> [REDACTED] </CdtDbtInd>
194   <Sts> [REDACTED] </Sts>
195   <BookgDt>
196     <Dt> [REDACTED] </Dt>
197 </BookgDt>


```

A.22 Benutzerdokumentation (Auszug)

Der folgende Auszug aus der Benutzerdokumentation werden die jeweils einzelnen Schritte erläutert, die bei der Bedienung des CAMT053 - Webservice relevant sind.

Schritt 1: XML-Datei importieren

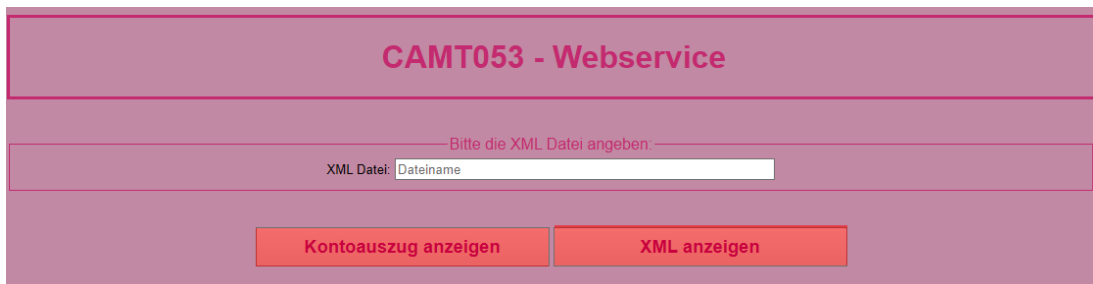
Um eine CAMT053-Datei in den Webservice importieren zu können, muss als erstes der Name der XML-Datei in das Eingabefeld eingegeben werden. Die Endung .xml ist bei der Eingabe nicht erforderlich.



The screenshot shows the 'CAMT053 - Webservice' interface. At the top, there is a header bar with the title 'CAMT053 - Webservice'. Below the header, there is a form area. Inside the form, there is a label 'Bitte die XML Datei angeben:' followed by a text input field labeled 'XML Datei:' with the placeholder text 'Dateiname'. Below the input field, there are two buttons: 'Kontoauszug anzeigen' and 'XML anzeigen'.

Schritt 2: Eingabe bestätigen

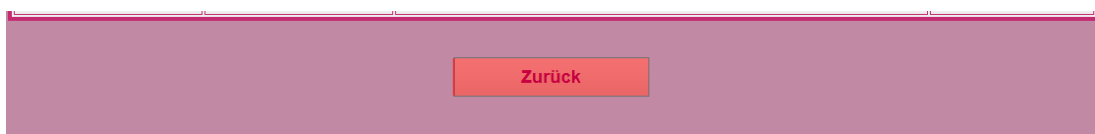
Nachdem die Eingabe der CAMT053-Datei abgeschlossen ist kann nun mit dem egl importieren begonnen werden. Dafür kann der unter dem Eingabefeld angegebene Button mit dem Namen: „XML anzeigen“ betätigt werden um die Weiterverarbeitung und die abschließende Ausgabe der Inhalte der CAMT053-Datei im Browser zu generieren. Möchte man nur die notwendigen Daten eines Kontoauszuges wie IBAN, BIC, Salden und Umsätze anzeigen, ist dafür der Button „Kontoauszug anzeigen zu betätigen“.



This screenshot is identical to the one in Schritt 1, showing the 'CAMT053 - Webservice' interface with the input field for the XML file name and the buttons 'Kontoauszug anzeigen' and 'XML anzeigen'.

Schritt 3: Neue Eingabe?

Möchte man als Anwender eine neue Eingabe durchführen ist unterhalb der generierten Ausgabe ein weiterer Button mit dem Namen: „Zurück“ zu finden. Nach Betätigung des Buttons kommt man zurück zur Startseite wo man eine neue Eingabe durchführen kann.



The screenshot shows a single button labeled 'Zurück' centered on a light blue background.

A.23 Entwicklerdokumentation (Read Me)

```
#####
# Titel:      Entwicklerdokumentation (Read Me)
# Programm:   CAMT053 - Webservice (Version 1.0)
# Autor:      Marvin Viedt
# Copyright:  ©a.b.s. Rechenzentrum GmbH
# Erstelldatum: 29.05.2020
#####

Einführung:

Willkommen zum CAMT053 - Webservice

Dieses Tool wurde für die a.b.s. Rechenzentrum GmbH als Resultat eines Abschlussprojektes einer Ausbildung erstellt.
Sinn und Zweck dieses Tools ist es CAMT053 Datei einzulesen und die darin enthaltenden Daten im Browser darzustellen.

Der CAMT053 Webservice bietet daher zwei Arten von Ansichten an:

    > Eine Kontoauszugs Ansicht
    > Eine XML Ansicht

#####

Installation:

Um den Webservice auf den lokalen Rechner zu installieren, muss zuerst das Verzeichnis camt053-Webservice entweder auf den Desktop/Verzeichnis kopiert werden.

Der Webservice muss dann über den Befehl ./camt053-Webservice in der Komandozeile (CLI) oder mit doppel klick auf der Ausführbaredatei (.exe) gestartet werden.

Zusätzliche Argumente werden nicht bei der Eingabe des Befehls benötigt.

#####

Ordnerstruktur:

Innerhalb des Verzeichnisses camt053-Webservice befinden sich folgende relevante .go Dateien:

main.go, webseite.go, XML-View.go und Kontoauszugs-View.go

Diese Dateien beinhalten lediglich den jeweiligen Go-Quellcode für den Webservice

In den anderen Dateien befindet sich hauptsächlich unbenutzter Code, der während der Entwicklung des Webservices archiviert wurde.

Die CSS Dateien sind im Verzeichnis static zu finden.

Einige Test CAMT053 Dateien sind im XML Verzeichnis zu finden.

#####

Die Funktionen in Main.go:

Folgende Funktionen die innerhalb der main.go Datei relevant sind:

main:
Hauptfunktion des Programms.
Hier wird mitgeteilt, dass der Server gestartet wurde und mit dem Aufruf der Handlefunc und Handler Methoden wird der Webserver und die Webseiten etabliert.
Zudem werden die entsprechenden Templates für die Layouts der Webseiten geladen.

handler:
Darstellung und Etablierung der Willkommenseite mit dem Eingabefeld
Etablierung eines Cookies

kauszughandler:
Etablierung und Festlegung der beiden Browseransichten für die CAMT053 Datei

getformvalue:
Einlesen des Eingabewertes

checkstring:
Überprüfung eines Strings ob dieser nur Zahlen, Buchstaben und Newlines enthält.

gettagname:
Ermitteln und Speicherung eines Tagnamens

getxmlvalues:
Ermitteln und Speicherung der Tagnamen und Werte der CAMT053 Datei

readxmlfile:
Öffnen und Lesen der CAMT053 Datei
Festlegung der Startpunkte für den GrpHdr und Stmt Tag zum Durchlaufen der Werte
Speicherung ermittelten Werte in String Arrays
Prüfen welche Ansicht angegeben wurde.

printauszugscontent:
Speicherung der notwendigen Werte für die Kontoauszugs Ansicht in die Map
Etablierung des Layouts für die Kontoauszugs Ansicht

printxmlcontent:
Durchlaufen der Arrays für GrpHdr und Stmt für die anschließende Ausgabe
Etablierung des Layouts für die XML Ansicht

remove_empty_value:
Durchlaufen eines String Arrays
Entfernen von Leeren Werten aus einem String Array

getdate:
Umwandlung des Formats für das Datum (2019-03-25 zu 25.03.2019)

#####

#####

Todo:
Gilt für die Methoden readxmlfile und printauszugscontent
Durchlaufen der Untertags von GrpHdr und Stmt
Ermitteln der Tagnamen und Werte mit mehreren Schleifen.
Speicherung der Tagnamen und Werte in einer Map (Tagname, Tagvalue)

#####
```


Abnahmeprotokoll

Projekt:

Zeitraum:

- Die Abnahme war erfolgreich
- Die Abnahme war nicht erfolgreich
Folgende arbeiten sind noch auszuführen:

- Bemerkungen/Sonstiges:

Ausgeführte Schritte	Abnahme erfolgreich
Erstellung der Software	<input type="checkbox"/>
Installation der Software	<input type="checkbox"/>
.....	<input type="checkbox"/>

Datum, Unterschrift

Auftraggeber

Datum, Unterschrift

Auftragnehmer