OktoPOS Solutions GmbH

Abschlussprüfung Winter 2020/21

Fachinformatiker für Anwendungsentwicklung

Projektdokumentation

Hamburg, den 10.12.2020

Anbindung der OktoPOS Kassensoftware an den internen Übersetzungsdienst

Prüfungsteilnehmer	Ausbildungsbetrieb
Philipp Jetzlaff Kielort 19c 22850 Norderstedt philipp.jetzlaff@white-paper-media.de Prüflingsnummer: 131 54050	OktoPOS Solutions GmbH Große Elbstraße 212 22767 Hamburg Ausbilder: Stefan Marksteiner stefan.marksteiner@white-paper-media.de 040 4665657-66

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Glossar

Begriff	Definition
Annotations	Markierungen im Quellcode für die ORM Konfiguration.
Buildserver	Server für die kontinuierliche Integration von Software.
Caching	Zwischenspeichern von Daten.
Classpath	Verzeichnis in dem die Java-Class-Dateien liegen.
Client	In der Client-Server-Architektur entspricht der Client einem Nutzer.
Dashboard	Übersicht.
Datenbankschema	Aufbau der Relationen in einer Relationalen Datenbank.
Deployment	Ausliefern von Software.
DDL	Befehle zur Definition von Tabellen und anderer Datenstrukturen.
Dependency Injection	Entwurfsmuster, welches die Abhängigkeiten eine Objekts zur Laufzeit reglementiert.
DML	Befehle zur Datenmanipulation und Datenabfrage.
Entität	Ein eindeutig bestimmbares Objekt, über das die Datenbank Informationen speichern soll.
HashMap	Eine Datenstruktur in der Informatik um Werte einem eindeutigen Key zugewiesen werden können.
Interface	Schnittstellen, entweder für weitere Programme oder Benutzer.
JSON	JavaScript Object Notation.
Langauagelevel	Die Version einer Programmiersprache.
Multilingualität	Die Verwendung von mehreren Sprachen.
ORM	Objektrelation-Mapping. Eine Technik in der Softwareentwicklung um Objekte in Entitäten umzuwandeln.
POS-System	Point-Of-Sales.
Property-key	Schlüssel zur Identifizierung eines Wertes innerhalb einer Datei.
Property-Pair	Kombination aus einem Property-Key und einer Property-Value. Z.B. foo=bar.
Property-Value	Wert, welcher einem Schlüssel in einer Datei zugewiesen ist.

Begriff	Definition
Stylesheet	Dokument um den Style von Klassen innerhalb eines HTML Dokuments zu beschreiben.
Token	Platzhalter.
Webservice	Stellt eine Schnittstelle für die Kommunikation über Rechnernetze zur Verfügung.

2 Einleitung

Im Rahmen der IHK-Abschlussprüfung für den Ausbildungsberuf Fachinformatiker für Anwendungsentwicklung wurde diese Projektdokumentation angefertigt. Sie dokumentiert den Ablauf und die Herangehensweise welche zur Lösung der, im Vorfeld von dem zuständigen Ausbilder definierten, Projektanforderungen beigetragen haben. Der Ausbildungsbetrieb OktoPOS Solutions GmbH ist ein mittelständisches Unternehmen mit Hauptsitz in Hamburg. Hauptdienstleistung der OktoPOS Solutions GmbH sind das POS-System OktoPOS Cash und das Personalmanagementsystem OktoCareer.

2.1 Projektbeschreibung

Das von der OktoPOS Solutions entwickelte Produkt OktoPOS Cash ist ein, im internationalen Raum, genutztes POS System. Für eine anwenderfreundliche Nutzung ist die gesamte Textausgabe des Front-End in diversen Sprachen verfügbar. Zur Realisierung der multilingualen Textausgabe werden für die angezeigten Texte Platzhalten (Tokens) im Kassencode verwendet. Die anzuzeigenden Texte in den jeweiligen Sprachen werden als Schlüssel-Werte-Paar in den Übersetzungsdateien hinterlegt, wobei der im Kassencode verwendete Token als Schlüssel verwendet wird. Für jede unterstützte Sprache gibt es genau eine Übersetzungsdatei. Die Erweiter- und Wartbarkeit der Übersetzungsdateien sind nach aktuellem Stand stark optimierungsbedürftig. Es gibt keine Garantie dafür, dass jede Datei dieselbe Anzahl an Tokens beinhaltet bzw. es gibt keinen direkten Überblick über den Übersetzungsstand der Dateien. Es kann vorkommen, dass die Übersetzungen von unternehmensfremden Personal angefertigt, die mit der Struktur solcher Dateien nicht vertraut sind. Dadurch verlängert sich die Einarbeitungszeit der Übersetzer und kostet nicht notwendige Ressourcen.
Neben der unwirtschaftlichen Nutzung von Ressourcen, ist der Stand der Übersetzung innerhalb des Kassensystems abhängigkeit von der Releaseversion der Kassensoftware. Übersetzungs- sowie Rechtschreibfehler in den Übersetzungen können erst nach Release einer neuen Kassenversion behoben werden.
In der Abteilung für das Personalmanagementsystem OktoCareer wird nach einem ähnlichen Prinzip die Mulitlingualität des Front-Ends unterstützt. Dabei werden allerdings die genutzten Tokens über den unternehmensinternen Übersetzungsdienst TranslationService abgebildet und in einem benutzerfreundlichen Front-End übersetzt und verwaltet. Mit der Anbindung der Kassensoftware an den unternehmensinternen Übersetzungsdienst TranslationService sollen die oben genannten Schwächen im Übersetzungsworkflow und dem Management der Übersetzungsdateien behoben werden. Außerdem soll über ein zusätzlichen Dienst der aktuellste Stand an Übersetzungen im Livebetrieb zur Verfügung gestellt werden können.

2.2 Projektziel

Ziel des Projektes ist es, durch die Anbindung der Kassensoftware an den unternehmensinternen TranslationService, den Prozess der Übersetzung von dem Releaseprozess zu entkoppeln und Versionsupdates der Übersetzungsdateien im Livebetrieb zu ermöglichen. Im Rahmen des Projektes soll die Integrität des Kassencodes erhalten bleiben. Daher ist es notwendig den Updateprozess der Übersetzungsdateien in einen externen Dienst auszulagern.
Der Transaltionservice ist zu so zu erweitern, dass Tokens und Übersetzungen in Abhängigkeit zu ihrer Version empfangen bzw. bereitgestellt werden können. Nach aktuellem Stand, ist der TranslationService nicht in der Lage, Tokens und Übersetzungen zu versionieren. Das Anlegen einer neuen Version erfolgt von außen über eine zu schaffende Schnittstelle. Im Front-End des TranslationService, soll die Möglichkeit gegeben werden, die Tokens und den Stand der Übersetzung in Abhängigkeit zu ihrer Version anzeigen zu lassen.
Ziel ist es durch eine Versionierung der Tokens und Transaltions eine Kassenversion kompatible Version der Übersetzungsdateien zu exportieren. Dabei können die Anzahl der Tokens, sowie die mögliche Übersetzung einer Textstelle in Versionen unterscheiden.
Da das Erstellen der Schnittstellen, die Implementierung der neuen Anwendung und das Testen der Komponenten die veranschlagten 70 Stunden in Anspruch nehmen, werden das Deployment der Minianwendung sowie die Anpassung des Deploymentprozesses der Kassensoftware an eine andere Abteilung ausgelagert, die in der Kostenaufstellung mit aufgenommen werden.

2.3 Projektumfeld

Das Projekt ist ein Auftrag der Entwicklungsabteilung der Kassensoftware OktoPOS Cash. Das Java gestützte POS System nutzt Übersetzungsdateien um Textstellen im Front-End in verschiedenen Sprachen darzustellen. Während der Entwicklung an der Kassesoftware, speziell beim Implementieren von Features, werden in machen Fällen neue Übersetzungstokens eingeführt. Der Entwickler muss dabei den Token in jede einzelne Datei schreiben. Die Übersetzer der Abteilung OktoCareer nutzen den TranslationService für die Übersetzungen an ihrem Personalmanagementsystem. Der TranslationSerivce bietet die Datenstruktur, welche mit geringen Aufwand erweitert werden kann um die neuen Anforderungen zu erfüllen.

2.4 Projektabgrenzung

Die grundlegende Struktur des TranslationService ist bereits implementiert und wird produktiv im Unternehmen genutzt. Das Projekt beschränkt sich hinsichtlich der Arbeiten an dem Transaltionservice nur auf das Hinzufügen der neuen Schnittstellen, dem Erweitern der Datenbank und den daraus resultierenden Änderungen der vorhandenen Entitäten und Anpassungen im Front-End. Die beschränkte Dauer, die für dieses Projekt zur Verfügung gestellt wurde, hat zur Folge, dass Teile zur Realisierung des Gesamtprojektes an andere Abteilungen ausgelagert werden mussten. Dazu gehört neben dem GitWebhook, welcher die Übersetzungsdateien der Kasse an den Übersetzungsdienst übergibt, auch das Deployment des neuen Dienstes.

3 Projektplanung

3.1 Entwicklungsprozess

Im Vorfeld der Planung wurde von dem Autor ein Entwicklungsprozess festgelegt. Der Entwicklungsprozess legt fest, in welcher Reihenfolge die notwendigen Projektphase geplant und bearbeitet werden können. Der Autor hat sich in Absprache mit den Projektleitern der betreffenden Projekte auf das erweiterte Wasserfallmodell geeinigt.
Ergänzend zu den iterativen Aspekten des Wasserfallmodells erlaubt das erweiterte Wasserfallmodell einen schrittweisen Rückgang zu vorhergehenden Phasen, sofern in der aktuellen Phase ein Fehler aufgetreten ist.
Die Abbildung 1 zeigt das genutzte Wasserfallmodell.

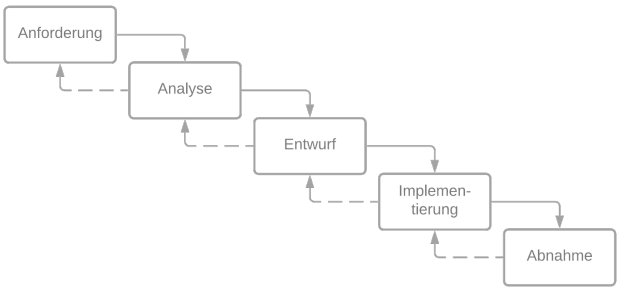

Abbildung 1: Erweitertes Wasserfallmodell

Im weiteren soll der Entwicklungsprozess durch das Test-Driven-Development-Prinzip erweitert werden.

3.1.1 Test Driven Developtment

Das Test-Driven-Development-Prinzip wird durch die drei folgenden Schritte beschrieben.

- **Tests erstellen:** Zu Beginn der Implementierung werden Tests erstellt, welche die Funktionalität der neuen Funktion verifizieren sollen.
- **Implementierung:** Die Geschäftslogik hinter den Funktionen implementieren, bis die Tests erfolgreich durchlaufen werden.
- **Refactoring:** Nachdem die Geschäftslogik implementiert wurde, kann der Entwickler seinen Code noch optimieren.

3.2 Projektphasen

Durch den gewählten Entwicklungsprozess aus Abschnitt 3.1 wurden die zu verfügung stehenden 70 Entwicklerstunden auf die verschiedenen Projektphasen aufgeteilt. Die Zeiteinteilung sowie die einzelne Projektphasen wurden für einen groben Überblick in der Tabelle 1 zusammengefasst. Eine genaue Zeitplanung inklusive der Aufgaben in jeder einzelnen Phase können aus dem Anhang A.1 entnommen werden.

Projektphase	Stunden
Analysephase	7h
Entwurfsphase	11h
Implementierungsphase	35h
Abnahme und Testphase	3h
Dokumentation	14h
Gesamtstunden	70h

Tabelle 1: Grobe Zeitplanung

3.3 Ressourcenplanung

Das Projekt wurde auf einem, von dem Ausbildungsbetrieb zur Verfügung gestellten, Windows Surface Book geplant, bearbeitet und getestet. Dabei wurde als Entwicklungsumgebung für den TransaltionService sowie für den Java Dienst die Software IntelliJ Ultimate von der Firma JetBrains verwendet. Die Grundlage der Daten für den TranslationSerivce sind in einer MySql Datenbank gespeichert. Zur Überprüfung der korrekten Anwendung des ORM Frameworks Doctrine hinsichtlich DDL, DCL und DML, wurden die Ergebnisse über die Software MySQL Workbench mittels SQL Abfragen validiert. Neben den automatisierten Tests der Schnittstellen mit PHPUnit, wurden regelmäßig mit der Software Postman Anfragen an die REST Schnittstellen geschickt. Für die, in Latex erstellte, Dokumentation notwendigen Diagramme wurden über den Onlinedienst Lucidchart angefertigt. Eine genaue Übersicht aller verwendeten Ressourcen ist im Anhang A.6 Genutzte Ressourcen zu finden.

4 Analysephase

4.1 Ist-Analyse

In der nachfolgenden Analyse wird das Kassensystem OktoPOS Cash und der Übersetzungsdienst TranslationService in ihrer ursprünglichen Verfassung beschrieben.

4.1.1 OkotoPOS Cash

Wie in den Abschnitten 2.1 und 2.3 beschrieben unterstützt das Kassensystem OktoPOS Cash ein multilinguales Front-End. Um das zu gewährleisten, werden im Quellcode der Kasse sog. Übersetzungstokens verwendet. Für jede unterstützte Sprache ist eine Übersetzungsdatei im Property-Format in der

Ressourcen der Kassesoftware hinterlegt. Im Normalfall sollte jede Datei das gleiche Set an Tokens mit der jeweiligen Übersetzung als Schlüssel-Werte-Paar beinhalten. Bei der Einführung eines neuen Tokens hat der Entwickler die Aufgabe den neuen Token in eine dieser Dateien zu überführen und eine exemplarische Übersetzung anzulegen. Da es im Unternehmen gängig ist, die Tokens nur in eine bzw. maximal zwei Dateien zu überführen, wird die Aufgabe der Wartung und Pflege an die Übersetzungsabteilung delegiert. Neben der trivialen und zeitaufwändigen Arbeit die einzelnen Dateien auf den gleichen Stand zu halten, kann es auf Grund von diversen Faktoren passieren, dass der Stand der Dateien divergiert.
Bisher gibt es keine benutzerfreundliche Oberfläche, um Übersetzungen für die einzelnen Tokens anzufertigen bzw. auch keine direkte Referenz auf die Textstelle an denen der Token eingesetzt wird. Der Übersetzer hat die Aufgabe, sich aus geeigneten Quellen (Ticketsystem, andere Übersetzungsdatei, Entwickler fragen), eine geeignete Übersetzung zu beschaffen. Es gibt allerdings auch Fälle, bei denen keine Übersetzung für einen Token erstellt wurde. Dann hat der Übersetzer die Aufgabe, über das Lesen von Quellcode die konkrete Textstelle im Front-End zu finden.
Im aktuellen Workflow werden die Übersetzungsdatein im Betazustand einer neuen Minor-Version an die Abteilung für Übersetzung übergeben. Nach Fertigstellung der Übersetzung werden die überarbeiteten Dateien an den Projekteigentümer der Kassesoftware übergeben, welche er dann mit den alten Übersetzungsdateien zusammenführt. Dadurch kann nur noch indirekt über einen Release-Branch auf die verwendeten Übersetzungen geschlossen werden und nicht über einen eigenen Versionsstand der Übersetzungsdateien. Außerdem hat der Workflow zur Folge, dass Änderungen an den Übersetzungsdateien immer ein Versionsupdate der Kasse mit sich bringen.

4.1.2 TranslationService

Für die Übersetzungen in einem anderen System des Unternehmens wurde der webbasierte Transaltionservice entwickelt. Dieser hat die Aufgabe über Http-Schnittstellen, Tokens zu importieren bzw. zu exportieren. In einer benutzerfreundlicher Übersicht hat der Übersetzer die Möglichkeit sich nicht übersetzte Tokens für seine Sprache anzeigen zu lassen. Neben der Übersicht hat jeder Token, falls vorhanden, eine exemplarische Übersetzung für jede Sprache, in der Token bereits übersetzt wurde. Der TranslationService bietet verschiedene Möglichkeiten, die Informationen für Tokens anzeigen zu lassen. Das Komponentendiagramm im Anhang A.2 stellt die Beziehungen zwischen den einzelnen Systemkomponenten dar.
Die Daten des Übersetzungsdiensts sind in einer MySql Datenbank gespeichert. Das ERD im Anhang A.4 zeigt das Datenbankschema in der ursprünglichen Form. Mit dem ERD wird die Planung der Anpassungen an der Datenbankstruktur erleichtert.
Nach aktuellem Stand ist es nicht möglich, die einzelnen Tokens und deren Übersetzung über eine Version abzubilden. Das hat zur Folge, dass die Änderungen an den Übersetzungen final sind.

4.2 Soll-Analyse

Durch die Anbindung des Kassensystems an den Transaltionservice soll die Pflege und Wartung der Übersetzungsdateien im Kassensystem nahezu automatisiert werden. Die Übersetzungen für die einzelnen Tokens sollen über eine neue Schnittstelle am TranslationService in Abhängigkeit zu ihrer Version bereitgestellt werden. Es ist ein Dienst zu erstellen, welcher die vom TranslationService bereitgestellten Daten an der Schnittstelle anfragt und die Übersetzungsdateien im Livebetrieb aktualisieren kann. Änderungen an Übersetzungen, auf Ebene des TranslationService, werden dann durch den Neustart der Kasse übernommen. Durch die neue Anbindung soll der Übersetzungsprozess vollständig von dem Deployment der Kasse entkoppelt werden. Damit die Übersetzungen in Abhängigkeit zu ihrer Version erstellt bzw. bereit gestellt werden können, ist es notwendig den TranslationService hinsichtlich einer Versionierung von Tokens und Translations anzupassen. Die dafür notwendigen Änderungen sind auf Datenbank-, Schnittstellen- und Front-End-Ebene auszuführen.

4.3 Wirtschaftlichkeitsanalyse

4.3.1 ”Make-Or-Buy”-Entscheidung

Da es sich bei dem Projekt um ein internes Feature mit Berührungspunkten an unternehmensinternen Projekten handelt, lässt sich auf dem Markt keine alternative Lösung finden. Aus diesem Grund hat man sich dazu entschlossen, das Feature in Eigenentwicklung

4.3.2 Kostenaufstellung

In der Tabelle 2 werden die Kosten, die für dieses Projekt aufgebracht wurden, aufgelistet. Entwicklerstunden eines Auszubildenden werden mit 40€ pro Stunde berechnet, während ausgelernte Anwendungsentwickler mit einem Stundensatz von 120€ veranschlagt werden. In den Kosten ist der Gemeinkostenzuschlag bereits eingerechnet.

Vorgang	Mitarbeiter	Zeit(h)/Mitarbeiter	Personal/h	Gesamt
Entwicklungskosten	1x Auszubildender	70h	40€	2800€
Planungsmeeting	2x Anwendungsentwickler	4h	120€	960€
Codereview	1x Anwendungsentwickler	1h	120€	120€
Abnahme	2x Anwendungsentwickler	0.5h	120€	120€
Projektkosten gesamt				4000€

Tabelle 2: Kostenaufstellung

4.3.3 Amortisationsdauer

Das Projekt enthält neben den monetäre Vorteilen auch nicht-monetäre Vorteile. Daher ist eine genaue Amortisationsdauer nicht festzulegen. Dennoch kann anhand der monetäre Vorteile davon ausgegangen werden, dass sich die Anpassungen zu einem späteren Zeitpunkt positiv auf den Gewinn des Unternehmens auswirken.

4.3.4 Monetäre Vorteile

Nach Abschluss des Projektes werden die Übersetzungsdateien durch den Übersetzungsdienst generiert, die zeitaufwändige Wartung und Pflege entfällt. Durch die benutzerfreundliche Oberfläche des Übersetzungsdiensts müssen externe Übersetzer nicht mehr in die Struktur und Funktionsweise der Übersetzungsdateien eingebarbeitet werden. Die dadurch entstandene Zeitersparnis kann als zusätzlicher Gewinn für das Unternehmen gerechnet werden.

4.3.5 Nicht-monetäre Vorteile

Als Softwarehersteller ist der Kundensupport und die Kundenzufriedenheit ein wichtiger Bestandteil der wettbewerbsfähigkeit. Auch wenn mit dem Liveupdate der Übersetzungen im Kassensystem und die Versionierung der Tokens und Übersetzungen im TranslationService kann keine konkreten Gewinne berechnen lassen, ergeben sich dennoch Voreile, die sich indirekt positiv auf den Unternehmensgewinn auswirken.

4.4 Anwendungsfälle

Die Tabelle 3 zeigt, welche Stakeholder von der Anbindung des Kassensystems an den TranslationService betroffen sind. Das Use-Case-Diagramm im Anhang A.3 verdeutlicht die direkten Berührungpunkte der Stakeholder mit den Systemen. Da es sich bei dem Projekt um ein internes Feature handelt, wird der Stakeholder ”Kunde”nicht mit aufgeführt.

Stakeholder	Aufgabenbereich
Übersetzer	Fertig Übersetzungen am Transaltionservice an.
Entwickler	Legt im Rahmen eines neuen Features einen neuen Übersetzungstoken an.

Tabelle 3: Stakeholder

4.5 Lastenheft

Zum Ende der Analyse wurde in Kooperation mit den Verantwortlichen aus den zuständigen Abteilungen ein Lastenheft angefertigt. Dieses beschreibt die funktionalen und nicht-funktionalen Anforderungen an das Projekt. Der Abschnitt über die funktionalen Anforderungen ist im Anhang A.7.

5 Entwurfsphase

5.1 Zielplattform

Das Ziel des Projektes ist, wie im Abschnitt Projektbeschreibung beschrieben, die Kommunikation zwischen zwei bestehenden System zu ermöglichen. Nach Angaben des Auftraggebers aus dem Lastenheft soll die Integrität des Kassencodes bestehen bleiben. Um das zu gewährleisten werden die notwendigen Programmschritte in einen eigenenständigen Dienst ausgelagert.
Als Anforderung an den Dienst soll die Möglichkeit bestehen, diesen zu einem späteren Zeitpunkt in den Kassencode zu integrieren. Daraus entsteht die Vorgabe, das Modul auf dem gleichen Java Languagelevel des Kassencodes zu programmieren. Zum aktuellen Zeitpunkt wird die Kasse über das Build-Management-Tool Ant gebaut. Basierend auf der Anforderung des Auftraggebers wird in naher Zukunft auf das Build-Management-Tool Gradle umgestellt. Für eine erleichterte Integration des neuen Moduls, soll das neue Projekt bereits den Gradle Buildprozess unterstützen.
Der TranslationService ist in der Programmiersprache PHP auf dem Languagelevel 7.2 programmiert. Es wurde bei der Entwicklung des TranslationService darauf geachtete, dass der Service in allen gängigen Webbrowsern[1] funktionsfähig ist. Für die Darstellung der Daten wird HTML 5.2 verwendet. Für eine klare, ansprechende und benutzerfreundliche Darstellung der Daten wurde das, in HTML 5.2 beschriebene, Front-End mit dem Stylesheet von Twitter Bootstrap verbessert.
Da es sich bei den Systemen um produktive Systeme handelt, sind die zu benutzenden Programmierparadigmen und Programmiersprachen vorgegeben. Das Aufstellen einer Nutzwertanalyse ist daher obsolet.

5.2 Architektur

Der TransaltionSerivce basiert auf dem Model-View-Controller (MVC)-Architekturmuster. Demnach werden die zugrunde liegenden Daten in einer Datenbank gespeichert, während die View für die Darstellung der Daten verantwortlich ist. Über Controller wird eine bidirektionale Kommunikationschicht geschaffen, um Daten zu manipulieren bzw. Daten anzuzeigen. Durch die Entkopplung, welche durch das MVC-Pattern erreicht wird, wird die Erweiter- und Wartbarkeit signifikant erleichtert.
Die Darstellung der Daten im TranslationService wird durch die Template-Engine Twig realisiert. Dabei werden die Daten über Action Klassen der Engine zur Verfügung gestellt. Die Twig-Engine verwendet die Daten, um variable Felder mit Daten zu beschreiben. Die Action-Klassen repräsentieren im MVC-Pattern die Controller. Action-Klassen werden üblicherweise über vordefinierte Routen aufgerufen und verarbeiten die mitgesendeten Daten auf Grundlage konzipierten Aufgabe. In dieser webbasierten Anwendung werden die genutzten Daten in einer Datenbank persistiert. Die Modelle der Daten sind als Entitäten in einer Datenbank definiert. Das Object-Relation Mapping (ORM)-Framework Doctrine automatisiert dabei die Aufgabe, aus definierten Objekten die korrekte DDL zu generieren und auf der Datenbank auszuführen.
Die aktuell verwendeten Schnittstellen des TranslationService sind bereits nach der RESTful-Architektur designed. RESTful steht für eine zustandslose Kommunikation zwischen Client und Server. Durch cachingfähige Daten können Client-Server-Interaktionen optimiert werden. Die neuen Schnittstellen, sollen

[1] Mozilla Firefox, Google Chrome, Micrsoft Edge, Safari, Opera

ebenfalls nach der RESTful-Architektur implementiert werden. Der Translationupdater ist der neue Dienst welcher das Kassensystem um eine Subroutine erweitern soll. Die Subroutine beschafft Daten von einem autarkem System, verarbeitet diese und injiziert die verarbeiten Daten in das bestehende Kassensystem.

5.3 Benutzeroberfläche

Änderungen an der Benutzeroberfläche des TranslationService werden über die Template-Engine Twig durchgeführt. Dabei wird die Oberfläche durch die Auszeichnungssprache HTML beschrieben. Twig bietet die Möglichkeit HTML Formulare durch algorithmische Grundstrukturen zu erweitern. Dabei ist es möglich, Iterationen und bedingte Anweisungen direkt im HTML Formular einzubinden.

5.4 Datengrundlage

Wie im Abschnitt 5.2 beschrieben, persistiert der TranslationService seine Daten nach dem Datenbankschema A.4, in einer MySql Datenbank. Die REST Schnittstellen im TransaltionService verwenden das für REST typische Format JSON, um Daten mit Clients auszutauschen. Die neu zu entwicklenden Schnittstellen sollen sich der bestehenden Struktur anpassen und dasselbe Format für den Datenaustausch mit den Clients verwenden.
Als Datengrundlage für die Übersetzungsdateien nutzt das Kassensystem Dateien im Property-Format. Der Translationupdater hat somit auch die Aufgabe, die empfangenen Daten aus dem Json-Format in das, von der Kasssensoftware, lesbare Property-Format zu konvertieren.

5.5 RESTful APIs

Im Abschnitt TranslationService ist beschrieben, dass die bestehenden Schnittstellen nach der REST-Architektur designed wurden. Für eine stringente Implementierung werden neue Schnittstellen ebenfalls nach dem RESTful-Architekturmuster designed. Die Kommunikation zwischen einem Client und einer REST Schnittstelle basiert auf dem HTTP-Request-Response Prinzip. Dabei ist der HTTP-Request die Anfrage des HTTP-Client an den Server. Der Server verarbeitet die Anfrage und schickt den HTTP-Response an den Client zurück. Die Abbildung 2 soll das Prinzip visualisieren. Im Gegensatz zu den

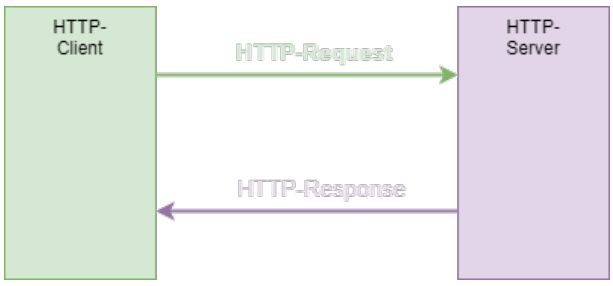

Abbildung 2: Http-Request-Response-Prinzip

ursprünglichen undokumentierten Schnittstellen, hat der Auftraggeber im Lastenheft darum gebten, die neuen Schnittstellen zu dokumentieren.
Als Dokumentationswerkzeug für RESTful APIs wird in diesem Projekt Swagger verwendet. Ein Ausschnitt der, im Planungsmeeting entstandenen, Schnittstellen ist im Anhang 5.5. Schnittstellendokumentation Auszug zu finden. Die neuen Schnittstellen wurden in einem Planungsmeeting mit dem zuständigen Projekteigentümer des TranslationServices geplant.

5.6 GitWebhook

Für die Versionsverwaltung nutzt das Unternehmen den Webanwendung GitLab. GitLab unterstützt die Möglichkeit einen sog. WebHooks einzurichten.
Ein WebHook führt anhand eines bestimmten Ereignis, dir vorgegebenen Anweisungen aus. Für die Anbindung der Kassensoftware an den TranslationSerivce, sollte ein WebHook eingerichtet werden,

welcher auf ein Merge-Ereignis reagiert. Dabei sollte Änderungen innerhalb der Übersetzungsdateien erkannt und an den TransaltionService weitergeleitet werden.

5.7 Testcases

Das gesamte Projekt wurde nach dem Prinzip Test-driven entwickelt. Das bedeutet, dass vorangehend zur Implementierung Tests erstellt werden. Die Tests werden auf Grund der fehlenden Funktionalität zu Beginn fehlschlagen. Test-Driven bietet den Vorteil, dass der Entwickler während der Implementierung von Funktionalitäten ein direktes Feedback bekommt, ob das gewünschte Ergebnis erreicht wurde bzw. diverse Fehlerfälle aus dem Test abgedeckt wurden.
Damit Tests, die Datenbankzugriffe beinhalten, nicht beeinflusst werden können, werden die Tests auf einer leeren Testdatenbank ausgeführt. Um eine leere Testdatenbank zu garantieren, wird vor und nach jedem Test die gesamte Datenbank bereinigt. Für den Test werden ausschließlich im Test definierte Daten verwendet. Die Tests für den TranslationService werden mit dem PHP Framework PHPUnit erstellt und regelmäßig ausgeführt. Der Translationupdater nutzt das Testframework JUnit und führt seine Tests über das Gradle Buildscript während des Buildprozesses aus.

5.8 Pflichtenheft

Zum Ende der Entwurfsphase wurden die Entwürfe in einem Pflichtenheft zusammengefasst. Das Pflichtenheft beschreibt auf Basis des Lastenheftes die geplante Umsetzung der Anforderungen.
Das Pflichtenheft wurde von den betroffenen Abteilungen genehmigt und dient als Leitfaden für die Realisierung des Projektes. Im Anhang A.9 ist ein Auszug des erstellten Pflichtenheft zu finden.

6 Implementierungsphase

6.1 Iterationsplanung

Für eine strukturierte Implementierung wurde ein Iterationsplan erstellt. Dieser gibt dem Entwickler eine Überblick über die erforderlichen Schritte zur Fertiggestellung des neuen Features. Da es sich um zwei unterschiedliche Systeme handelt, wurden zwei Iterationspläne erstellt. Der Iterationsplan im Anhang A.10 beschreibt den Ablauf der Implementierung für den TranslationService, während der Iterationplan A.11 den Ablauf für den Translationupdater beschreibt.

6.2 TranslationService

6.2.1 Erweiterung der bestehenden Datenstruktur

Als Grundlage für die weiterführende Programmierung wurde zu Beginn der Implementierung mit der Erweiterung der Datenstruktur begonnen. Die daraus resultierenden Änderungen an dem Datenbankschema können aus dem Anhang A.5 entnommen werden. Um die geforderte Versionierung im TranslationService abbilden zu können, musste die neue Entität `Version` in das bestehende Datenbankschema eingebunden werden. Das ORM Mapping Tool Doctrine kann über Annotations PHP-Klassen in ein Datenbankschema überführen.
Dabei werden Klassen, die in der Datenbank eine Entität repräsentieren sollen, mit der Annotation `@ORM\Table` markiert. Listing 1 zeigt, wie die Klasse `Version.php` als Entität für Doctrine markiert wird.

```
@ORM\Table (name="version")
```

Listing 1: Annotation für eine Entität

Attribute werden für Doctrine ebenfalls über Annotations markiert. Dabei wird die Annotation über das Attribut der Klasse geschrieben.

```
@ORM\Column (type="string", nullable=false)
```

Listing 2: Annotation für ein Attribut

Dabei können neben dem Datentyp auch weitere Parameter wie zum Beispiel `nullable=false` gesetzt werden. `Nullable` sagt aus, das der Wert in der Spalte nicht null sein darf.
Aus dem finalen ERD (Anhang A.5) sind neben der neuen Basisentität noch die Relationstabellen Token_Version und Translation_Version zu erkennen. Relationstabellen werden von Doctrine generiert, wenn zwei Tabellen in einer Many-To-Many Beziehung stehen. In der objektorientierten Programmierung werden Many-To-Many Beziehungen über Listen abgebildet. Dabei haben die beiden Objekte eine Liste von Instanzen des jeweiligen anderen Objektes als Attribut.
Damit das verwendente ORM Tool die Relationstabellen für die betroffenen Entitäten erstellen kann, werden weitere Annotations benötigt. Listing B.1 zeigt an dem Beispiel Token_Version die Umsetzung einer M:N Beziehung über Doctrine Annotations.

6.2.2 Erweitern der Schnittstellen

Anhand der Schnittstellendefinition A.8 wurden die neuen Schnittstellen im TranslationService implementiert. Der TranslationService wurde auf Basis des Slim Frameworks aufgebaut. Über das PHP-File `Bootstrap.php` werden mithilfe der Klasse `TranslationServiceAppBuilder.php` die Routen für die Schnittstellen in der TranslationService App definiert. Neue Routen werden anhand eine simplen Schemas als Route für die App definiert. Die Definition der Routen innerhalb des Slim Frameworks ist wie folgt aufgebaut:

```
$app->post('/api/v1/{foo}/list',Foo::class)
```

Listing 3: Beispielroute

Befehl	Erläuterung
$app	Konfigurationsklasse für die Transaltionservice Applikation.
post	Definition des HTTP Request für die Route. (GET, POST, PUT usw.)
´/api/{foo}/list´	Definition der Route. Variable Parameter werden in {} dargestellt.
Foo::class	Die genutzte Action Klasse die aufgerufen werden soll.

Tabelle 4: Übersicht Routenregistrierung in Slim

Die Tabelle 4 erklärt die Bestandteile der Routenregistrierung aus Listing 3. Im Listing B.2 wird die Umsetzung einer Routenregistrierung anhand von zwei, im Rahmen des Projektes erstellten, Routen deutlich gemacht.
Für die neuen Routen wurden dementsprechen neue Action-Klassen entwickelt die die jeweiligen Anforderung an die Routen erfüllt.
Über das verwendente PHP Framework

```
"php-di/slim-bridge": "^1.1.2"
```

können alle im Service verwendeten Abhängigkeiten in einem Container abgelegt und per Dependency Injection in den verschiedenen Klassen genutzt werden. In den meisten Fällen haben die Action-Klassen Abhängigkeiten zu den verschiedenen Repositoires. Die Abhängigkeiten werden in den Konstruktoren der Actionklasse definiert. Durch die Dependency Injection werden bei Aufruf die definierten Abhängigkeit geladen. Dependency Injection hat den weiteren Vorteil, dass zur Laufzeit auf der gleichen Instanz eines Objektes gearbeitet wird.
Beim Aufruf einer Action-Klasse wird die Funktion `__invoke` mit den Parameter `Slim/Request` und `Slim/Response` aufgerufen. Innerhalb der `__invoke` wird die eigentliche Logik der Action-Klasse ausgeführt. Über die Referenz auf `Slim/Request`-Objekt kann der Entwickler auf die im Request stehenden Daten zugreifen.
Der Parameter `Response` ist eine Referenz auf das Responseobjekt des Routenaufrufes. Dieser wird so manipuliert, dass die Applikation die korrekten Informationen an den aufrufenden Client zurückgibt. Für den Response stehen mehrere Möglichkeiten zur Verfügung, um die Informationen in ein eindeutig lesbares und serialisierbares Ergebnis zu ”verpacken”. In den meisten Fällen wird für den Datenaustausch über RESTful designte Schnittstellen das JSON-Format genutzt. Neben der Information, ob die angeforderte Operation erfolgreich (success) war, wird eine Message und optional auch noch weitere Daten an den Client zurückgegeben. Da es sich um eine Referenz handelt, muss das Responseobjekt nicht neu erzeugt werden. Es werden neue Attribute hinzugefügt oder geändert.

Der Ablauf von `__invoke` Funktionen der Action-Klasse wird in dem Kapitel Geschäftslogik 6.2.3 genauer erläutert. Während des Projektes wurden alle im Anhang A.8 definierten Routen nach dem in diesem Abschnitt beschriebenen Prinzip implementiert.

6.2.3 Geschäftslogik

In dem Abschnitt 6.2.2 wurde die Erweiterung der Schnittstellen beschrieben, wie die einzelnen Routen mit dem Slim-Framework angelegt und mit einer Geschäftslogik verknüpft werden. Dieser Abschnitt beschreibt den Umgang mit den mitgesendeten Daten hinsichtlich der generellen Routine, die der Bearbeiter genutzt hat, um vorhersehbare Fehler abzufangen und das geforderte Ergebnis zu erreichen.
Wie im Abschnitt 6.2.2 beschrieben, hat jede Route genau eine eigene Action-Klasse, alternativ auch Controller genannt, die wiederrum genau eine `__invoke`-Funktion besitzt. Listing 8 und Listing 9 zeigen zwei unterschiedliche Implementierungen der `__invoke`-Funktionen. Die Implementierung der `__invoke`-Funktionen richtet sich nach den Vorgaben der Schnittstellendokumentation im Anhang A.8. Eine `__invoke`-Funktion beginnt damit, die mitgesendeten Daten und Routenparameter in geeignete Variablen zu überführen. Um zu verhindern, dass mit fehlenden, falschen oder kaputten Daten gearbeitet wird, werden vor zuerst die - für die Schnittstelle definierten - Fehlerfälle abgefangen und mit dem, in der Schnittstellendokumentation A.8 definierten, Response an den Client zurückgegeben.
Nach Ausschluss aller möglichen Fehlerfälle, können die validen Daten verarbeitet werden. Die Erweiterung des TranslationService beinhaltet drei Arten von Schnittstellen - Hinzufügen, Ändern und Abfragen von Daten. Die drei genannten Typen der Datenverarbeitung greifen auf die geeigneten Funktionen der Entityrepositories zu. Um die Rechenleistung des Servers zu entlasten, wird die Anzahl der Datenbankoperationen so gering wie möglich gehalten. Um das zu erreichen bietet der Entitymanager von Doctrine die Möglichkeit alle Änderungen erst in einem Cache zu persistieren. Persistierte Änderungen werden über die Funktion `$this->_em->flush` des Entitymanager in die Datenbank geschrieben. Als Anwendungsfall gelten Massenimporte von Tokens. Die Tokens werden aus dem Request ausgelesen, validiert, als konkretes Objekt erstellt und in einem Cache hinterlegt. Erst nach der vollständigen Iteration über die gelieferten Tokens werden die Änderungen in der Datenbank gespeichert.
In dem Abschnitt 5.3 wird der grundlegende Aufbau der Benutzeroberfläche des TranslationService erläutert. Die Anforderungen des Auftraggebers hinsichtlich der Erweiterungen der Benutzeroberfläche beschränken sich auf einen weiteren Filter, der die Translation und Tokenübersicht hinsichtlich ihrer Version sortiert. Dabei hat sich der Projektbearbeiter an den bestehenden Style der Benutzeroberfläche gehalten. Im Anhang A.12 ist ein Vorher-Nachher-Vergleich zu sehen. Für den neuen Filter wurde ein neuer Dropdown-Button an das bestehende Filtermenu angefügt. Das Dropdown besteht aus den einzelnen Versionen die bereits im System angelegt wurden. Die Daten werden dem Front-End über die Twig-Engine bereitgestellt. Der Klick auf ein Dropdown-Item löst die

6.3 Translationupdater

Die zweite Iterationsphase des Projektes behandelt die Implementierung des Translationupdaters. Der Translationupdater ist ein Dienst zum Aktualisieren der Übersetzungsdateien der Kassensoftware. Die Routine für den Translationupdater besteht aus drei Teilen - Beschaffung, Verarbeitung, Injektion. Die Routine des Translationupdaters wird nach Vorgaben des erstellten Aktivitätsdiagramm (Anhang A.13) implementiert. Da es sich um einen Dienst für das Kassensystem handelt, sind bestimmte Parameter über eine Konfigurationsdatei steuerbar. Die Tabelle 5 beschreibt die konfigurierbaren Parameter des neuen Dienstes. Die Konfiguration des Translationupdater wird über die Klasse `SystemConfig.java` realisiert.

Parameter	Erläuterung
baseUrl	Bestimmt die URL über die der TranslationService erreichbar ist
languages	Eine Liste von Sprachen die aktualisiert werden sollen
version	Gibt die Version der Übersetzungen an

Tabelle 5: Konfigurationsparameter

Diese lädt die einzelnen Properties aus der `configuration.ini` (siehe Listing B.3) und überführt die Werte der Parameter in Attribute der Klasse. Über eine Instanz der `SystemConfig.java` können via `Getter`-Methoden auf die einzelnen Werte zugegriffen werden. Da es sich bei der Konfiguration um eine

eindeutige, zur Laufzeit unveränderbare Klasse handelt, wurde die `SystemConfig.java` als Singleton-Pattern implementiert.

Die Singletonimplementation stellt sicher, dass von einer Klasse genau ein Objekt erzeugt wird. Da eine Konfiguration normalerweise an verschiedene Stellen im Code verwendet wird, kann auch der Vorteil der globale Erreichbarkeit eines Singleton genutzt werden. Listing B.4 zeigt anhand der `SystemConfig.java` wie eine Singletonimplementation umgesetzt wird.
Als weitere Utility wurde das `Printer`-Interface geschaffen. Das `Printer`-Interface abstrahiert die in der Tabelle 6 gelisteten Methoden um unterschiedliche Implementierung durch verschiedene Printer zuzulassen. Eine konkrete Implementierung des `Printer`-Interfaces ist die Klasse `ConsolePrinter.java`.

Signatur	Beschreibung
printProgressMessage(int progress, int max, String toWrite)	Gibt den aktuellen Progress inklusive einer Nachricht aus
print(String message)	Gibt Nachricht aus.
printError(String error)	Gibt einen Error aus.
printWarning(String)	Gibt eine Warnung aus.

Tabelle 6: Printer Interface

Der `ConsolePriner` soll die Nachrichten über die Kommandozeile an den Nutzer weitergeben. Durch das Interface `Printer.java` besteht die Möglichkeiten zu einem anderen Zeitpunkt weitere Ausgabeformen zuzulassen. Der erste Schritt der eigentlichen Routine, ist die Beschaffung der geforderten Daten von einem autarken System. Der TranslationService stellt die versionierten Übersetzungen über die Schnittstelle (Listing 4)

```
'/api/v1/version/{version}/translations/{language}
```

Listing 4: Export Translations

Die Schnittstelle erwartet die Parameter `version` und `language`. Für den Parameter `version` ist der Wert aus der `SystemConfig.java` zu verwenden. In den Anforderungen aus dem Lastenheft an den Translationupdater beschreibt der Auftraggeber, dass die Aktualisierung von mehreren Übersetzungsdateien innerhalb eines Programmaufrufes möglich sein muss. Die Konfiguration stellt dafür den überladbaren Parameter `languages` bereit. Die `SystemConfig.java` gibt die konfigurierten Sprachen mit einem `Getter` als Liste von Sprachen zurück. Die in Listing 4 beschriebene HTTP-Schnittstelle stellt nur die Übersetzungen einer Sprache bereit. Daher muss die Schnittstelle für jede einzelne Sprache neu angesprochen werden. Die Routine zur Datenbeschaffung ist als Nassi-Shneiderman-Diagramm im Anhang A.14 beschrieben.

Der notwendige HTTP-GET-Request wird mithilfe der externe Java Library `org.apache.httpcomponents` erstellt und ausgeführt. Der Methodenaufruf `CloseableHttpClient::execute` gibt eine Referenz auf das Responseobjekt zurück, mit dem die ”Antwort” des TranslationService ausgelesen werden kann. Die Schnittstellendokumentation A.8 beschreibt den Response bei einer erfolgreichen Anfrage. Mit der externen Library `com.google.gson` werden die Array-elmente des Json in ein Array von Objekten des Elementtypen `TransferObject` überführt. Zur weiteren Verwendung der Daten, wird das Array in eine ArrayList umgewandelt und mit dem jeweiligen Sprachenkürzel als Key in einer HashMap hinterlegt.

Im nächsten Schritt der Routine, werden die bereitgestellten Daten in das Property-Format konvertiert und temporär als Datei im Dateisystem abgelegt. Der Dateiname setzt sich aus dem prefix `translation`, dem Ländercode z.B. `De` und der Dateiendung `.properties` zusammen.

Der letzte Schritt injiziert die Übersetzungsdateien in die `.jar` er Kassensoftware. Der schematische Aufbau der Kassensoftware unterliegt laut der Betriebsordnung der OktoPOS Solutions GmbH der Geheimhaltung. Daher ist es dem Autor dieses Dokumentation nicht gestattet, konkrete Informationen über den Zielpfad der Injektion anzugeben. Das Updaten einer .jar-Datei ist innerhalb einer Java-Anwendung nur über externe Hilfsmittel möglich. Im folgenden wird ein ”Workaround” für die eigentliche Injektion beschrieben.
Der `CashDeskInjector.java` sucht innerhalb seines Dateiverzeichnis nach der konkreten .jar der Kassensoftware. Wird diese nicht gefunden, wird eine Warnung an den Benutzer ausgegeben und das Programm beendet. Dabei werden die im zweiten Programmschritt erstellten Übersetzungsdateien aus dem Dateisystem gelöscht. In dem anderen Fall wird ein CMD-Befehl generiert, der die Übersetzungsdatei

im Klassenpfad der Kasse durch die neue Übersetzungsdatei ersetzt. Listing 5 beschreibt den grundlegenden Aufbau des CMD-Befehls der über den Java Befehl `Runtime.getRuntime().exec(String command)` ausgeführt wird.

```
jar -uf foobar.jar de/foo/resources/translationDe.properties
```

Listing 5: Update Jar

Nach erfolgreicher Injektion der Übersetzungsdatei, wird diese aus dem Dateisystem entfernt. Nach vollständigen Injektion aller Übersetzungsdateien, wird eine Nachricht an den Nutzer ausgegeben und das Programm beendet.

7 Abnahme- und Einführungsphase

7.1 Abnahme durch den Fachbereich

Die Änderungen am TranslationService wurden nach Fertigstellung dem zuständigen Projekteigentümer vorgelegt. Die Anforderungen und Lösungsansetze wurden in den Meetings regelmäßig besprochen, daher war der Großteil der Änderungen bereits bekannt und die Abnahme konnte innerhalb kurzer Zeit durchgeführt werden. Der neu entwickelte Dienst Translationupdater wurde durch den Projekteigentümer der Kassensoftware abgenommen. Dabei wurden Verbesserungsvorschläge bezüglich des iterativen Ablaufes der Routine angemerkt und für den Ausblick 8.3 notiert.

7.2 Deployment

Nach den Vorgaben des Auftraggebers soll der neue Dienst über den Buildprozess Gradle gebaut werden. Daher wurde das Projekt im Vorfeld über das `Gradle Init Plugin` als Gradle Projekt angelegt. Im Nachgang wurde das Gradlescipt um die unternehmensinternen Buildparameter erweitert. Für das Deployment des neuen Dienstes wurde ein neuer Task auf dem unternehmensinternen Buildserver angelegt.
Da es bei dem TranslationService zu Änderungen an bestehenden Entitäten gekommen ist, müssen bestehende Livedaten des Service über ein SQL Script auf das neue Datenbankschema angepasst werden. Damit ein problemloses Update des Livesystems durchgeführt werden kann, muss mit den Kunden der OktoPOS Solutions GmbH ein geeigneter Termin kommuniziert werden. Die verschiedenen Faktoren führen dazu, dass die Anbindung des Kassensystems an den TranslationService erst zum Februar des Jahres 2021 durchgeführt werden kann.

7.3 Schulung der betroffenen Stakeholder

Durch die Anbindung der Kassensoftware an den bestehenden TranslationSerivce ändert sich der Workflow zum Erstellen von Übersetzungen. Die betroffenen Stakeholder aus dem Abschnitt 4.4 werden durch eine Präsentation über die Änderungen in Kenntnis gesetzt. Der zuständige Leiter der Übersetzungsabteilung bekommt zusätzlich eine Einweisung in die Arbeit mit versionierten Übersetzungen.

8 Retroperspektive

8.1 IST-SOLL-Vergleich

Im Rahmen des Projektes wurden zwei der drei Kernziele erreicht. Während der Planung des Projektes wurden von dem Autor Entscheidungen außerhalb seines Zuständigkeitsbereiches getroffen. Durch die Missachtung der Zuständigkeit, mussten die drei geplanten Schnittstellen verworfen werden und wurden durch die sieben neuen ersetzt. Dabei entstand eine eklatante Diskrepanz zwischen der veranschlagten und genutzten Zeit für die Implementierung der Schnittstellen. Der Autor hat in Absprache mit den Verantwortlichen den GitWebhook an eine weitere Abteilung ausgelagert.
Ansonsten wurden die Vorgaben aus dem Lastenheft des Auftraggebers erfolgreich ausgeführt.

8.2 Lessons Learned

Durch das Abschlussprojekt hat der Autor viele wertvolle Erfahrungen im Bereich der Projektplanung und Durchführung sammeln können. Speziell die fehlende Kommunikation mit einem der beiden Fachbereiche hat zu einem Fehler geführt, der von dem Bearbeiter in der Zukunft nicht noch einmal gemacht wird.
Neben den bereits umfangreichen Kenntnissen in der Programmiersprache Java konnte der Autor weitere Fähigkeiten im Bereich der Webentwicklung mit PHP und den dazugehörigen Frameworks erwerben.

8.3 Ausblick

Während der Bearbeitung des Projektes sind bereits weitere Vorschläge für die Verbesserung der TranslationService Anbindung besprochen worden. Wie in dem Abschnitt 5.1 Zielplattform angeschnitten, soll der Translationupdater später in den Kassencode als Modul integriert werden. Dabei soll die jetzt über eine neue Konfigurationsdatei einstellbaren Parameter durch das Kassensystem vorgegeben werden.
Im Abschnitt 7.1 hat der zuständige Reviewer die Anmerkung gemacht, die aktuell drei verwendeten Schleifen für den Updateprozess auf eine Schleife zu kürzen.
Der TranslationService bietet noch die Möglichkeit für die verwendenten Tokens eine Position (Datei + Zeile) anzugeben. Für die Zukunft ist geplant, die genutzten Tokens innerhalb des Kassencodes zu lokalisieren und die Daten im Service zu updaten. Dafür wurde ebenfalls der Vorschlag in Betracht gezogen, die Position des Übersetzungstokens zu versionieren.

A Anhang

A.1 Detaillierte Zeitplanung

	Analysephase		**7h**
1.	Durchführung der IST-Analyse	1h	
2.	Durchführung der Soll-Analyse	2h	
3.	Durchführung der Wirtschaftlichkeitsanalyse	1h	
4.	Erstellung des Lastenheftes	3h	
	Entwurfsphase		**11h**
1.	Zielplattform festlegen	1h	
2.	Architektur festlegen	1h	
3.	Erweiterung der Datenbank entwerfen	1h	
4.	Neue Schnittstellen planen und beschreiben	2h	
5.	Geschäftslogik für den TranslationService planen	2h	
6.	Anpassungen am Front-End planen	1h	
7.	Translationupdater planen (Aktivitätsdiagramm)	2h	
8.	Erstellen des Pflichtenheftes	1h	
	Implementierungsphase		**35h**
1.	Erweiterung des bestehen Datenbankmodelles	1h	
2.	Implementieren der PHPUnit Tests für die geplanten Schnittstellen	5h	
3.	Implementieren der neuen Schnittstellen	12h	
4.	Anpassen der bestehenden Benutzeroberfläche	2h	
5.	Implementieren der Routine nach dem Aktivitätsdiagramm	2h	
5.	Erstellen der Unittests für die einzelnen Komponenten des Java Dienstes	4h	
6.	Implementieren der Geschäftslogik für den Downloadmanager	2h	
7.	Implementieren der Geschäftslogik für den TranslationConvertors	2h	
8.	Implementieren der Geschäftslogik für den CashDeskInjector	3h	
9.	Erstellen des GitWebhooks zum Senden der verwendeten Tokens an den Service	2	
	Abnahme- und Testphase		**7h**
1.	Testen der Schnittstellen mit Postman	3h	
2.	Abnahme des TranslationService durch den zuständigen Anwendungsentwickler	1h	
3.	Testen der Funktionalität des Translationupdater	2h	
4.	Abnahme des Translationupdater durch den zuständigen Anwendungsentwickler	1h	
	Dokumentation		**14h**
1.	Erstellen der Projektdokumentation	12h	
2.	Erstellen der Entwicklerdokumentation	2h	
	Gesamt		**70h**

Tabelle 7: Detaillierte Zeiteinteilung

A.2 Komponentendiagramm TranslationService

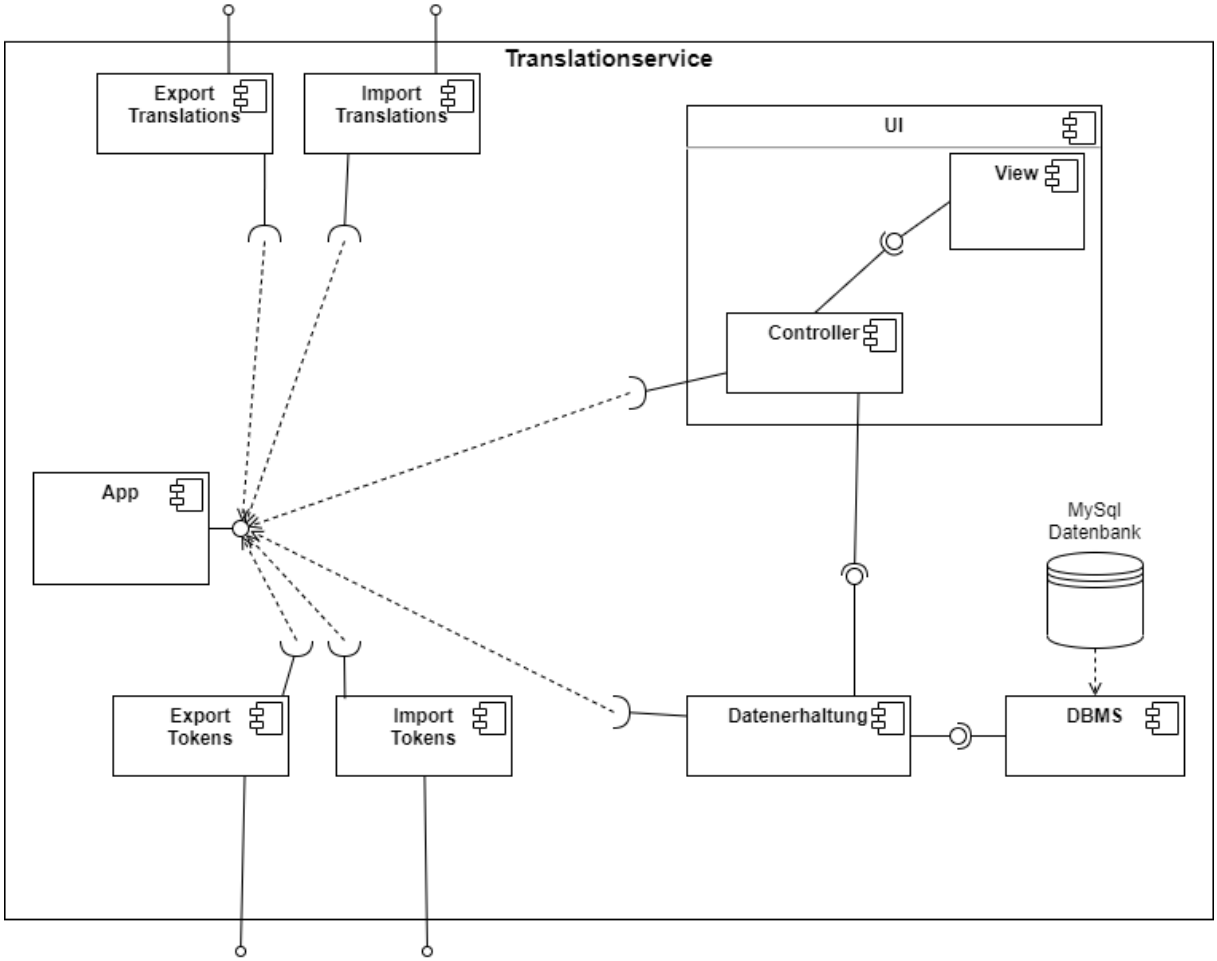

Abbildung 3: Komponentendiagramm TranslationService

A.3 Use-Case-Diagramm

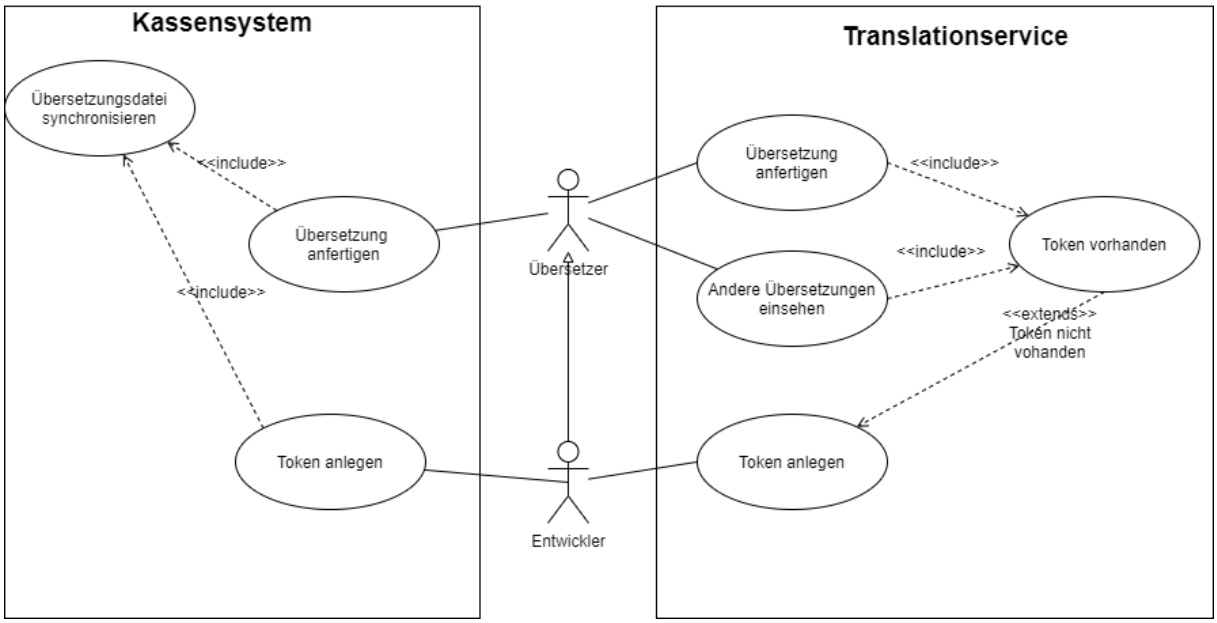

Abbildung 4: Use Case Diagramm Übersetzungen

A.4 Datenbankstruktur Urfassung

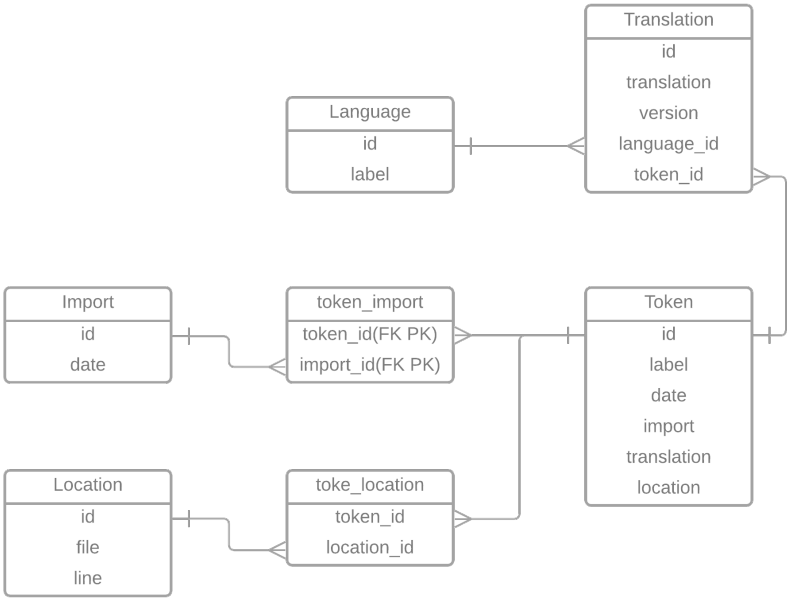

Abbildung 5: ERD im IST Zustand

A.5 Finale Datenbankstruktur

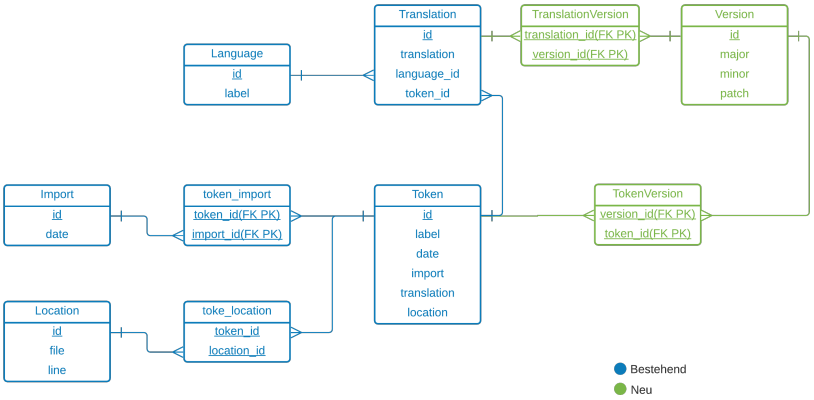

Abbildung 6: ERD im Soll Zustand

A.6 Ressourcen und Technologien

Hardware	
	Microsoft Surface Book - Entwicklungsrechner
Software	
	Microsoft Windows 10 - Betriebssystem
	JetBrains IntelliJ Ultimate 2020.2 - Entwicklungsumgebung für den TranslationService und Translationupdater
	MySql Workbench 8.0 CE - Datenbankmanagement und SQL Abfragen
	Postman 7.36.0 - Erstellen von HTTP Requests
	Laragon 4.0.16 - Lokale Webserverumgebung
	TexLive V. 2020 - Distribution des Textsatzsystem LaTeX
	Visual Studio code - Editor zum Erstellen der Dokumentation in LaTeX
	Lucidchart - Webanwendung zum Erstellen von UMl-Diagrammen und Grafiken
	Draw.io - Webanwendung zum Erstellen von UMl-Diagrammen und Grafiken
Personal	
	Entwickler - Umsetzung des Projektes
	Projektverantwortlicher TranslationSerivce - Planungsmeeting und Codereview
	Projektverantwortlicher Kassensystem - Planungsmeeting und Codereview

Tabelle 8: Genutzte Ressourcen

A.7 Lastenheft

Im folgenden Auszug aus dem Lastenheft werden die Anforderungen definiert, die die zu entwickelnde Anwendung erfüllen muss.

Lastenheft

- Der TranslationService muss die Möglichkeit bieten Tokens und Translations versioniert abzuspeichern. Dabei gilt: Ein Token kann in mehreren Versionen enthalten sowie eine Version kann aus mehreren Token bestehen. Gleiches gilt für die Translations.
- Es muss eine neue Schnittstelle geschaffen werden, die es ermöglicht von außen eine neue Version anzulegen.
- Es müssen neue Schnittstelle geschaffen werden, die es ermöglichen Tokens und Translations einer Version hinzuzufügen.
- Für einen ”Massenimport” an Translations und Tokens sollen die jeweils neuen Schnittstellen einen Diff zu einer gewählten Version erstellen. Dabei sollen nicht verwendete Tokens/Translations aus der Version entfernt werden.
- Es müssen neue Schnittstelle geschaffen werden, welche den Export von Tokens und Translations in Abhängigkeit zu einer Version ermöglichen.
- Die Translation- und Tokenübersicht soll einen neues Dropdown-Menu erhalten, mit dem die Tokens/Translations anhand der Version gefiltert werden können.
- Das Übersetzungsdateien innerhalb des Kassensystems sollen im Livebetrieb aktualisiert werden.
- Im Rahmen des Projektes ist die Integrität des Kassencodes zu erhalten. Daher soll das Update der Übersetzungsdateien über einen externen Dienst durchgeführt werden.
- Der externe Dienst soll so vorbereitet werden, dass er ohne weiteres in den Kassencode integriert werden kann.
- Neu angelegte Tokens sollen den TranslationService gesendet werden, sobald ein neues Feature zum Testen freigegeben wurde.
- Das Projekt soll über den Buildprozess Gradle gebaut werden.

[...]

A.8 Schnittstellendokumentation Auszug

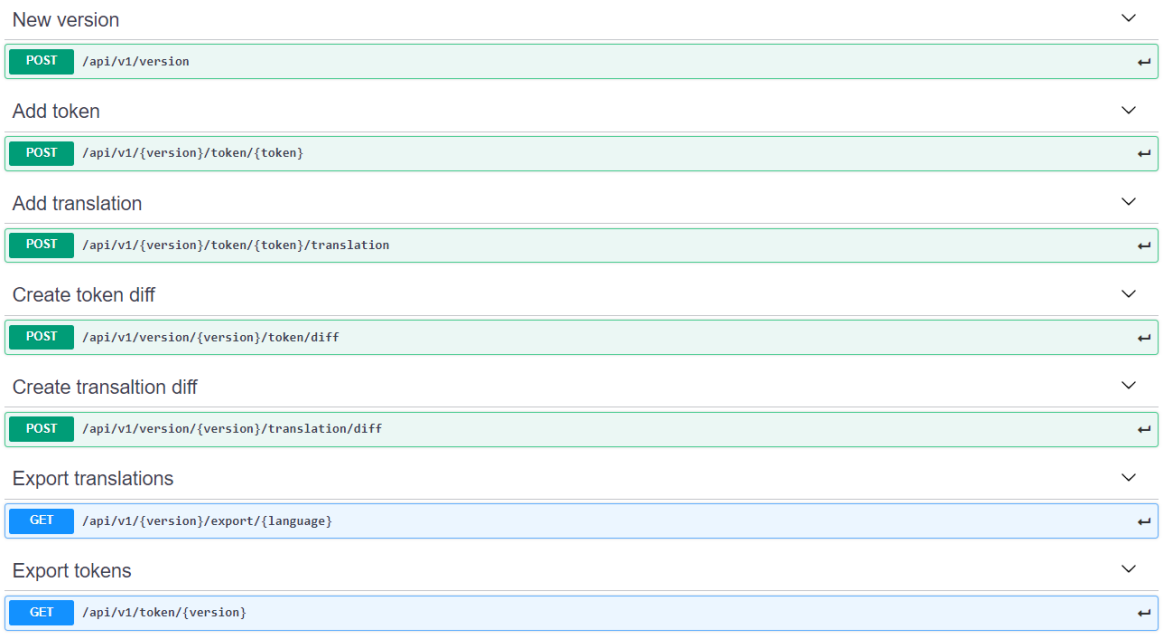

Abbildung 7: Übersicht der geplanten Schnittstellen

A.9 Pflichtenheft

In folgendem Auszug aus dem Pflichtenheft wird die geplante Umsetzung der im Lastenheft definierten Anforderungen beschrieben:

Umsetzung der Anforderungen

- Damit Tokens und Transaltions versioniert werden können ist eine weitere Entität in der Datenbank notwendig. Ohne weitere Relationstabellen zwischen den Version und Translation bzw. Version und Token würde das Datenbankschema nicht mehr in er 3ten Normalform sein.
- Diverse neue Schnittstellen werden nach der Schnittstellendokumentation basierend auf der RESTful Architektur designed und implementiert.
- Die Anpassungen am Front-End werden auf Basis des bisherigen Styles des TranslationService ausgeführt.
- Der neue Dienst zum Updaten der Übersetzungsdateien wird in der gleichen Java Version entwickelt in dem auch die Kasse entwickelt wurde.
- Damit ein neues Feature von der Qualitätsmanagement-Abteilung getestet werden kann, wird der Feature-Branch in den Masterbranch gemerged. Um die Vorgabe zu erfüllen die neuen Tokens an den TranslationService zu übergeben, wird zum Zeitpunkt des Merges der GitWebhook ausgelöst und sendet die neuen Token an den Service.

[...]

A.10 Iterationsplan TranslationService

- Erweitern des Datenbankschemas
- Hinzufügen der neuen Schnittstellen
- Implementieren der Geschäftslogik für die Schnittstellen
- Anpassen des Front-Ends

A.11 Iterationsplan Translationupdater

- Projekt als Gradlebuild einrichten
- Klassen für den Downloadmanager erstellen und Geschäftslogik implementieren
- Klassen für den FileConvertor erstellen und Geschäftslogik implementieren
- Klassen für den CashDeskInjector erstellen und Geschäftslogik implementieren
- Implementieren der Geschäftslogik nach Aktivitätsdiagramm

A.12 TranslationService Ui

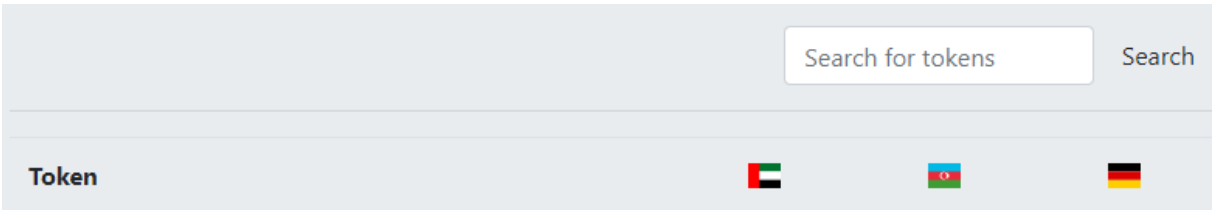

Abbildung 8: Ehemalige Tokenübersicht

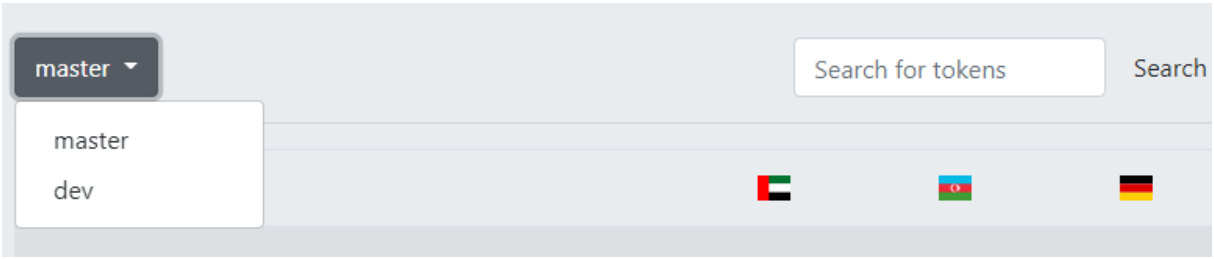

Abbildung 9: Neue Tokenübersicht

A.13 Aktivitätsdiagramm Translationupdater

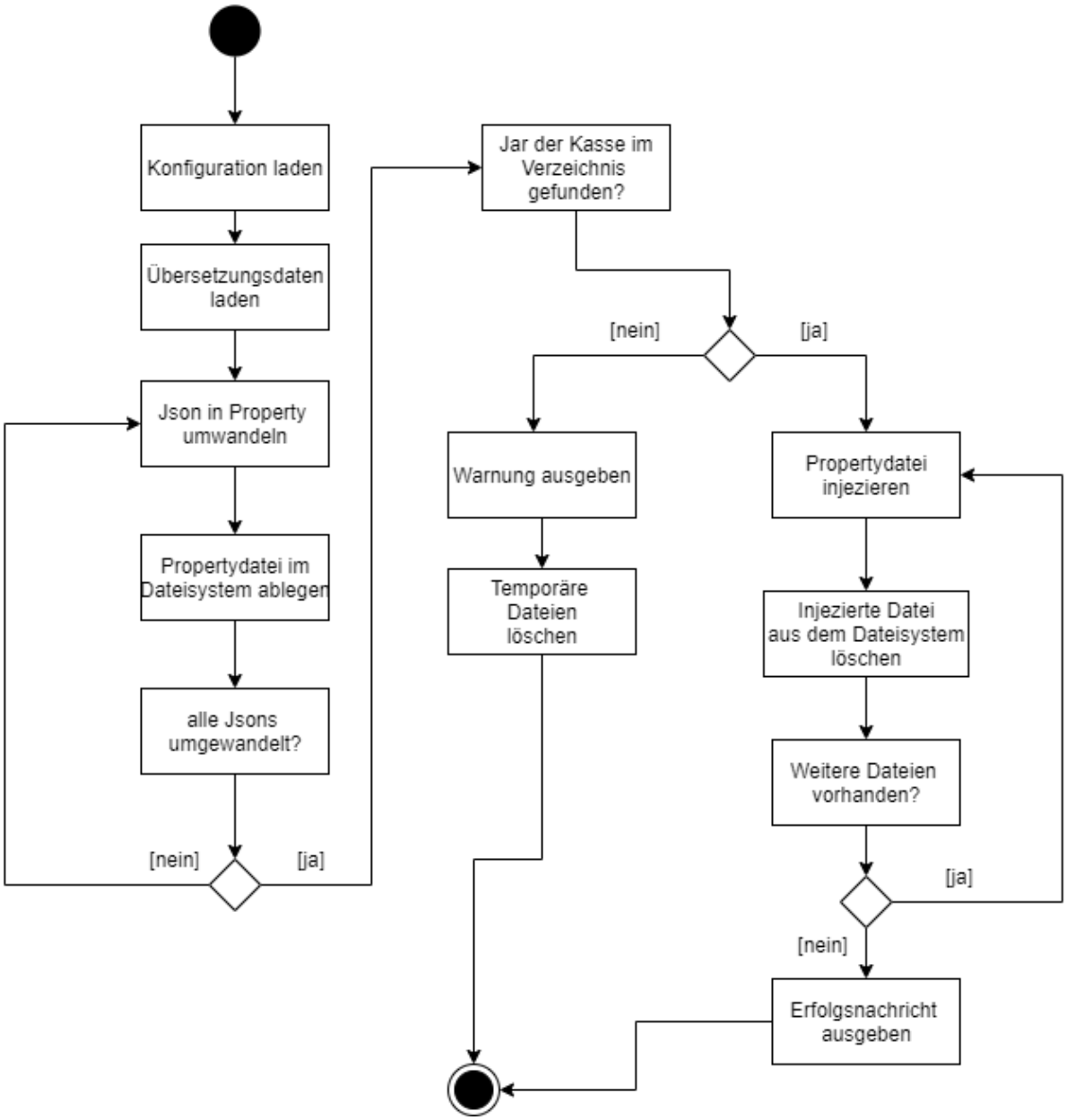

Abbildung 10: Aktivitätsdiagramm Translationupdater

A.14 Datenbeschaffung

Map<Stirng,List> loadTransaltions()

```
HttpClient httpClient = HttpClients.createDefault()
Map<String, List<TransferObject>> result = new HashMap≠()
int max = this.systemConfig.getLangiages().size()
int current = 1
foreach this.systemConfig.getLanguages as Languages
    printer.printProgressMessage(current,max,"Start downloading translation for " + language)
    url = this.createUrl(language)
    request = new HttpGet(url)
    response = httpClient.execute(request)
    result.put(language, this.parseEntity(response.getEntity())
    printer.printProgressMessage(current, max, "downloaded translation for" + language)
return result
```

Abbildung 11: Nassi-Shneiderman Diagramm Datenbeschaffung

B Listings

B.1 Many-To-Many Annotation

Token.php

```
/**
 * @var Version[]|ArrayCollection
 * @ORM\ManyToMany (targetEntity="OktoCareer\TranslationService\
    Version\Version", inversedBy="tokens")
 * @ORM\JoinTable (name="token_version",
 *   joinColumns = {
 *    @ORM\JoinColumn (name="token_id", referencedColumnName="id")
 * },
 *   inverseJoinColumns={
 *    @ORM\JoinColumn (name="version_id", referencedColumnName="id")
 * }
 *     )
 */
 private $version;
```

Listing 6: ManyToMany mit Doctrine

Version.php

```
/**
 * @var Token[]|ArrayCollection
 * @ORM\ManyToMany (targetEntity="OktoCareer\TranslationService\
    Token\Token",mappedBy="version")
 */
 private $tokens;
```

Listing 7: Many To Many mit Doctrine

B.2 Routing

HTTP GET Route:

```
$app->get('/api/v1/version/{version}/translations/{language}'
  ,ExportTranslationAction::class);
```

Listing 8: Erstellen einer HTTP GET Route

HTTP POST Route:

```
$app->post('/api/v1/version/{version}/diff/translations/{language}'
  ,CreateTranslationDiff::class);
```

Listing 9: Erstellen einer HTTP POST Route

B.3 Configuration.ini

```
[Webservice]
  # Defines the Url to the TranslationService
  baseUrl=http://localhost

  [Cashdesk Properties]
  # Defines the language-files to update
  # e.g fr,en,en-UK,it,sv,es,ja,tr
  languages=de,fr,en

  # Defines the version to download
  version=dev
```

Listing 10: Verwendete Konfigurationsdatei

B.4 Singleton

```
private static SystemConfig instance;
private Properties properties;

private String baseUrl;
private String languageList;
private String version;

public static SystemConfig getInstance() {
    if(instance == null) {
        instance = new SystemConfig();
    }
    return instance;
}

private SystemConfig(){
    try {
        loadProperties();
        loadConfiguration();
    } catch (IOException e) {
        System.out.println("Configuration file not found");
    }
}
```

Listing 11: Singlton in Java