

|  |
| --- |
| **Projektteam** |
| Bresemler, Eduard |
| Cavallaro, Angelo |
| Gröne, Adrian |
| Kasarca, Hüseyin |
| Kinzelmann, Daniel |

Inhaltsverzeichnis

[Abkürzungsverzeichnis III](#_Toc33034987)

[Abbildungsverzeichnis IV](#_Toc33034988)

[Tabellenverzeichnis V](#_Toc33034989)

[1 Vorstellung des Projekt-Teams 1](#_Toc33034990)

[2 SAPlexa – Die betriebliche Sprachassistenz für SAP 2](#_Toc33034991)

[2.1 Beschreibung von SAPlexa 2](#_Toc33034992)

[2.2 Technische Anforderungen an SAPlexa 2](#_Toc33034993)

[2.3 Einsatzbereich der Applikation 2](#_Toc33034994)

[3 Organisation des Projekts 3](#_Toc33034995)

[3.1 Projektbezogene Anwendung von SCRUM 3](#_Toc33034996)

[3.2 Zeitmanagement 3](#_Toc33034997)

[4 Meilensteine der (agilen) Softwareentwicklung 4](#_Toc33034998)

[4.1 Konzeption des Front-Ends 4](#_Toc33034999)

[4.1.1 Entwurf der Menüführung 5](#_Toc33035000)

[4.1.2 Ergonomie und Erprobung von Schlüsselbegriffen 9](#_Toc33035001)

[4.1.3 Informationsbeschaffung bei der Groz-Beckert KG 13](#_Toc33035002)

[4.2 Konzeption des Back-Ends 14](#_Toc33035003)

[4.2.1 Java-Perspektive 14](#_Toc33035004)

[4.2.2 SAP-Perspektive 17](#_Toc33035005)

[4.2.3 SAP Java Connector – Die Schnittstelle 21](#_Toc33035006)

[5 Zukünftige Optimierungs- und Erweiterungsmöglichkeiten 23](#_Toc33035007)

[6 Reflexion und Fazit 24](#_Toc33035008)

[Glossar 25](#_Toc33035009)

[Literaturverzeichnis 26](#_Toc33035010)

[Anhangsverzeichnis 27](#_Toc33035011)

[Anhang (Quellcodes) 28](#_Toc33035012)

# Abkürzungsverzeichnis

|  |  |
| --- | --- |
| BAPI | Business Application Programming Interface |
| BEDAT | Komponententyp: Bestelldatum |
| CHAR | Datentyp: Zeichenfolge |
| DATS | Datentyp: Datum im Format JJJJMMDD |
| EBELN | Komponententyp: Einkaufsbelegnummer |
| EBELP | Komponententyp: Einkaufsbelegposition |
| EKET | SAP-Datenbanktabelle: Lieferplaneinteilung |
| EKKO | SAP-Datenbanktabelle: Einkaufsbelegkopf |
| EKPO | SAP-Datenbanktabelle: Einkaufsbelegposition |
| i.d.R. | in der Regel |
| LGORT | Komponententyp: Lagerort |
| LIFNR | Komponententyp: Lieferantennummer |
| MATNR | Komponententyp: Materialnummer |
| MENGE | Komponententyp: Bestellmenge zur Position |
| NUMC | Datentyp: Numerischer Text |
| o.g. | oben genannt |
| QUAN | Datentyp: Mengenfeld |
| TXZ01 | Komponententyp: Material – Beschreibung |
| WEMNG | Komponententyp: Wareneingangsmenge (Gelieferte Menge) |

# Abbildungsverzeichnis

[Abbildung 5.1 Hauptmenü SAPlexa 5](#_Toc33035013)

[Abbildung 5.2: Hauptmenü SAPlexa - Bestellnummer übergeben 6](#_Toc33035014)

[Abbildung 5.3: MIGO-Übersicht SAPlexa 6](#_Toc33035015)

[Abbildung 5.4: MIGO-Übersicht SAPlexa - Bild anzeigen 7](#_Toc33035016)

[Abbildung 5.5: MIGO-Übersicht SAPlexa – Materialbeleg erzeugen 8](#_Toc33035017)

[Abbildung 5.6 Kardinalitäten zwischen EKKO, EKPO und EKET 17](#_Toc33035018)

[Abbildung 5.7 Tabellentypen - Strukturen - Datentypen 18](#_Toc33035019)

[Abbildung 5.8 Quellcode ZE268\_GETPROPOSALLIST 20](#_Toc33035020)

[Abbildung 5.9 Java-seitiger Funktionsaufruf auf den Baustein ZE268\_GETPROPOSALLIST 21](#_Toc33035021)

# Tabellenverzeichnis

[Tabelle 5.1 Relevante Datenbanktabellen in HANA 17](#_Toc33035022)

[Tabelle 5.3 Definierte RFC-Funktionsbausteine in SAP 19](#_Toc33035023)

# Vorstellung des Projekt-Teams

# SAPlexa – Die betriebliche Sprachassistenz für SAP

## Beschreibung von SAPlexa

(Wird noch überarbeitet)

SAPlexa- die sprachgesteuerte Applikation, die SAP unterstützt.

Im Rahmen des Projektstudiums wurde die Aufgabe gestellt eine Applikation zu entwickeln, die mit SAP kompatibel ist.

SAPlexa soll den Lagerarbeitern in aller erster Linie durch die Sprachsteuerung eine Erleichterung beim Wareneingang bringen. Durch diese Applikation muss der Lagerist nicht mit schmutzigen Fingern am Display touchen, sondern kann per einfachen Kommandos die App bedienen und den Wareneingang verbuchen. Des Weiteren kann er alle relevanten Informationen über die eingegangenen Bestellungen einsehen und darüber verfügen.

## Technische Anforderungen an SAPlexa

Was soll die App können?

Wobei unterstütz die App?

Hintergrundgeräusche

Oberfläche ?

## Einsatzbereich der Applikation

Wo soll sie eingesetzt werden?

Warum soll die dort eingesetzt werden?

**ADRIAN**

# Organisation des Projekts

## Projektbezogene Anwendung von SCRUM

**EDUARD**

Zuteilungen und reflektierte Anwendung auf unser Projekt

Zuteilungen = AUfgabnezuordnung pro Projektmitglied erwähnen

Wie wird die App entwickelt/ Fortschritte im Sprint?

Sprint und BackLog-Listen anfertigen

## Zeitmanagement

Das Zeitmanagement im Team wurde nach der im SCRUM-Prinzip beschriebenen Vorgehensweise strukturiert und organisiert. Somit ergaben sich in nahezu regelmäßigen Abständen definierte Zeitblöcke, in denen ausgewählte Ziele hinsichtlich der Implementierung oder auch der Recherchearbeit verfolgt wurden. Auf Grundlage einer realistischen Aufwandsabschätzung anhand eines Punktesystems konnte die termingerechte Fertigstellung in den allermeisten Fällen garantiert werden. Die Vordefinition einer konkreten Zielerreichung (Definition-Of-Done) stellte ein zentrales Merkmal der Organisation dar.

Als Dead-Line für das Projekt lässt sich terminlich die Vorstellung der Projekte am Dienstag, den 14. Januar 2020, nennen. Eine funktionierende und zuverlässige Sprachassistenz zur erfolgreichen Buchung eines (fiktiven) Wareneingangs stellte für die Vorstellung des Projekts das Hauptziel dar. Als Startzeitpunkt für das Projekt lässt sich rückblickend der Dienstag, den 15. Oktober 2019, identifizieren. Der Zeitrahmen umfasst demnach ganze 14 Wochen.

| **Jahr** | **KW** | **Zeitraum** |
| --- | --- | --- |
| Tätigkeit |
| **2019** | **42** | **14. Oktober – 20. Oktober** |
| 15. Oktober   Sprint Meeting 1°:  - Festlegung des Projektziels  - Besprechung der Projekt-Organisation  - Rollenverteilung |
| **43** | **21. Oktober – 27. Oktober** |
| 22. Oktober   Sprint Meeting 2°:  - Prozedur einer Wareneingangsbuchung im SAP wiederholen  - Besprechung von Beispielansagen für die Sprachassistenz  - Vorentscheidungen und Recherche |
| **44** | **28. Oktober – 03. November** |
| Recherche und Auswahl einer S2T API  Implementierungs- und Testphase |
| **45** | **04. November – 10. November** |
| Implementierungs- und Testphase |
| **46** | **11. November – 17. November** |
| 12. November   Sprint Meeting 3°:  - Feststellung der Zielerreichung (Review)  - Festlegung weiterer Projektziele im Backlog |
| **47** | **18. November – 24. November** |
| Vorbereitung zur Exkursion bei Groz-Beckert  Implementierungs- und Testphase |
| **48** | **25. November – 01. Dezember** |
| 26. November   Informationsbeschaffung:  Exkursion bei Groz-Beckert KG |
| **49** | **02. Dezember – 08. Dezember** |
| 03. Dezember  Internes Team-Meeting:  Nachbereitung der gewonnen Erkenntnisse aus der Exkursion |
| **50** | **09. Dezember – 15. Dezember** |
| 10. Dezember   Sprint Meeting 4°:  - Besprechung von Verbesserungspotenzialen der GUI und UX  - Beschränkung der visualisierten GUI-Inhalte aus Relevanzgründen |
| **51** | **16.Dezember – 22. Dezember** |
| Auswahl und Erprobung optimierter Keywords  Implementierungs- und Testphase |
| **52** | **23. Dezember – 29. Dezember** |
| Weihnachtsferien  Implementierungs- und Testphase |
| **2020** | **1** | **30. Dezember – 05. Januar** |
| Weihnachtsferien  Implementierungs- und Testphase |
| **2** | **06. Januar – 12. Januar** |
| Besprechung der Präsentationsinhalte der kommenden Projekt-Vorstellung  Finale Implementierungs- und Testphase |
| **3** | **13. Januar – 19. Januar** |
| 14. Januar  Dead-Line:  Vorstellung des Projekts |

In der obigen Zeitabfolge sind nur Meetings aufgeführt, die durch physische Anwesenheit an der Hochschule stattfanden. Virtuelle Meetings, die im Fortgang des Projektfortschritts immer relevanter wurden, sind nicht visualisiert.

# Meilensteine der (agilen) Softwareentwicklung

## Konzeption des Front-Ends

Das Front-End stellt die visuelle Kommunikation zwischen dem Lageristen und SAPlexa dar. Im Fall SAPlexa besteht das Front-End aus einer GUI**[[1]](#footnote-1)**, die als graphische Schnittstelle zwischen Mensch und dem technischem System agiert. Dabei ist es einem Lageristen nicht von großer Bedeutung wie eine GUI selbst programmiert ist, sondern mehr wie diese optisch aufgebaut ist. Man spricht hierbei von der *Nutzersicht der GUI*, die man anhand folgender Fragestellungen näher beschreiben kann:

* **Werden alle Elemente auf der Oberfläche erkannt?**

Sind die Kontraste dieser Elemente stark genug? Werden die Oberflächenbereiche deutlich genug voneinander abgesetzt? Sind alle Beschriftungen lesbar?

* **Sind die Abläufe in meinem Arbeitsumfeld logisch und folgerichtig?**

Sind diese Abläufe schnell zu lernen (Lernbarkeit)? Ist die Oberfläche selbsterklärend?

* **Ist die Oberfläche konsistent in Aussehen und Verhalten?**
* **Wie schnell kann ich mit der GUI arbeiten?**
* **Ist die Oberfläche auf mein Arbeitsumfeld angepasst?**

Um Antworten auf diese Fragestellungen zu finden, ist es wichtig sich in die Lage eines Lageristen zu versetzen. Es ist von großer Bedeutung zu erkennen was dem Lageristen für die Buchung eines Wareneingangs wichtig ist. Welche Informationen werden benötigt und auf welche Informationen kann verzichtet werden um die Oberfläche ansprechend und überschaubar zu halten.

Ein weiteres wichtiges Merkmal, das leicht in Vergessenheit gerät, ist die Wahl der Schriftgröße. Da ein Lagerist auch mal mit größeren Lieferungen rechnen muss und dadurch etwas weiter weg vom Bildschirm steht, ist eine angemessene große Schriftgröße ein wichtiges Merkmal um eine hohe Nutzerfreundlichkeit zu garantieren.

### Entwurf der Menüführung

Rückblickend auf die Nutzersicht der graphischen Oberfläche, betrachten wir in den folgenden Absätzen wie der Lagerist durch die GUI geführt wird. Wir nehmen an, dass eine Lieferung beim Lageristen ankommt und nun über SAPlexa der Prozess zur Wareneingangsbuchung startet.

Die Ablauflogik und Menüführung wird allem Voran in der nachstehenden Abbildung visualisiert.



Abbildung 4.1 Menüführung als Ablauflogik

Beim Start von SAPlexa wird zunächst das Hauptmenü angezeigt. Das Hauptmenü besteht dabei aus einer Liste aller Bestellungen. Anhand des farblich markierten Bereiches erkennt der Lagerist auf welche Kommandos SAPlexa lauscht.

Ein Bild, das Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung 4.2 Hauptmenü SAPlexa

Der Lagerist startet nun über das Kommando **„HANA“** die Spracheingabe und kann die Bestellnummer auf dem Lieferschein an SAPlexa übergeben. Dabei wird die aktuell übergebene Ziffer der Bestellnummer im oberen linken Bereich angezeigt. Wenn die Bestellnummer übergeben wurde, bestätigt man die Eingabe über das Kommando **„OKAY“.**

Dem Lageristen stehen für die Übergabe der Bestellnummer nun zwei Optionen zur Wahl. Der Lagerist hat die Möglichkeit nur einen Teil der Bestellnummer an SAPlexa zu übergeben. Anschließend kann nach der Bestätigung über das Kommando **„OKAY“** aus einer Liste, bestehend aus mehreren Treffern der übergebenen Ziffernfolge, die richtige Bestellnummer selektiert werden. Dazu muss das Kommando **„SELECT“** in Kombination mit der Voice-ID-Spaltennummer an SAPlexa übergeben werden und die Auswahl über das Kommando **„OKAY“** bestätigen.

Ein Bild, das Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung 4.3: Hauptmenü SAPlexa - Bestellnummer übergeben

Die zweite Möglichkeit besteht darin die ganze Bestellnummer an SAPlexa zu übergeben und dies direkt über das Kommando **„OKAY“** zu bestätigen. Somit gelangt man ohne Umwege in die MIGO-Übersicht von SAPlexa.

Ein Bild, das Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung 4.4: MIGO-Übersicht SAPlexa

Die MIGO-Übersicht enthält alle wichtigen Informationen für den Lageristen bezüglich einer Lieferung. Im oberen Bereich der MIGO-Übersicht werden die allgemeinen Daten zur Lieferung und die Details über den Lieferanten angezeigt. Dadurch kann der Lagerist die Lieferung schnell zuordnen.

Im Zentrum der MIGO-Übersicht werden alle Bestellpositionen zu der selektierten Lieferung aufgelistet. Dabei wird neben den allgemeinen Informationen wie dem gelieferten Material und der Produktbeschreibung auch die gelieferte Menge, die bestellte Menge und der Lagerort angezeigt. Dadurch kann der Lagerist auf einem Blick erkennen ob das richtige Produkt geliefert wurde und ob die gelieferte Menge der bestellten Menge entspricht. Zusätzlich bietet SAPlexa noch die Möglichkeit per Doppelklick ein Bild zu der ausgewählten Bestellposition anzuzeigen um das Produkt schneller zu identifizieren.

Ein Bild, das Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung 4.5: MIGO-Übersicht SAPlexa - Bild anzeigen

Nach der Überprüfung der Bestellpositionen kann diese nun gebucht werden. Im unteren Bereich der MIGO-Übersicht wird angezeigt auf welches Kommando SAPlexa lauscht um einen Wareneingang zu buchen. Ist nur eine Bestellposition aufgelistet, kann über das Kommando **„BOOK ORDER“** die Bestellposition gebucht werden. Falls mehrere Bestellpositionen vorhanden sind, müssen diese explizit über das Kommando **„SELECT“** in Kombination mit der Spaltennummer an SAPlexa übergeben werden und mit dem Kommando **„OKAY“** bestätigen.

Hat der Lagerist die gewünschte Bestellposition gebucht, wird ein Materialbeleg zu der Buchung erzeugt und angezeigt.

Ein Bild, das Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung 4.6: MIGO-Übersicht SAPlexa – Materialbeleg erzeugen

### Ergonomie und Erprobung von Schlüsselbegriffen

Ergonomie

Unter Ergonomie versteht man die wechselseitige Anpassung zwischen Menschen und ihren Arbeitsbedingungen. Auch in der Softwareentwicklung sollte auf die Ergonomie der Anwendung geachtet werden. Dabei sollte man folgende Kriterien/Eigenschaften im Blick behalten:

|  |  |
| --- | --- |
| Attribut | Beschreibung |
| Konsistenz | Der Nutzer sollte auf Bekannte Muster und Funktionen vertrauen können. |
| Verfügbarkeit | Features und Funktionen sollten nicht den Fluss der Software blockieren. |
| Verständlichkeit | Ähnliche Funktionen/Tools sollten in gleichen Sektionen zu finden sein und lassen sich auch ähnlich bedienen. |
| Automatisierung wiederholter Aufgaben | Repetitive Aufgaben sollten automatisiert werden oder optional automatisierbar sein. |
| Umgehende Rückmeldung | Der Nutzer sollte sowohl über fehlgeschlagene als auch durch erfolgreiche Aktionen informiert werden. (Fehler sollten „aufdringlich“ gezeigt werden z.B. Popup, Erfolge nur „nebenbei“ z.B. Statusziele) |
| Selbsterklärung | Durch Namen oder visuelle Elemente sollte sich der Sinn bzw. die Auswirkung einer Funktion erahnen lassen. |
| Anpassbarkeit | Der Nutzer sollte Optionen nach persönlicher Präferenz einstellen können |
| Fehlertoleranz | Fehler sollten vom Verursacher rückgängig gemacht bzw. korrigiert werden können |
| Erwartungskonformität | Funktionen sollten korrekt betitelt werden und nicht unvorhersehbare Auswirkungen mit sich bringen |
| Höflichkeit | Der Nutzer sollte auf Fehler, bei Aufforderungen oder einfachen Statusmeldungen immer Höflich auf seine Situation aufmerksam gemacht werden. |

Gerade bei Benutzereingaben sollte darauf geachtet werden, dass es zu möglichst wenig Missverständnissen kommen kann und die Eingaben für die vorgesehene Auftrittsumstände angemessen ausgewählt werden. Ferner sollte ebenfalls überlegt werden wie Komplex die Nutzereingaben sein können, sollen und dürfen. Generell sollten häufig auftretende Eingaben möglichst kurzgehalten werden, um Zeit und Aufwand bei der Eingabe niedrig zu halten. Beim Fall der Spracherkennung sollte ein gesunder Kompromiss zwischen für den Arbeiter angenehm aufzusagen als auch von der Spracherkennungssoftware schnell und zuverlässig erkennbar sein.

Erprobung von Schlüsselbegriffen

Bei der Erprobung von Schlüsselbegriffen sollten die oben genannten Punkte der Ergonomie im Hinterkopf behalten werden. Jedoch spielt bei den Schlüsselbegriffen die zuverlässige Erkennung in Speech–to–Text Software eine wichtigere Rolle. Dabei sollten die Begriffe sorgfältig ausgewählt werden. Um die Fehlerquote zu senken hilft es, sich vorab Gedanken zu machen was der S2T-Sw helfen könnte Wörter besser zu differenzieren. Jedoch darf ein Erproben der Begriffe in einem Umfeld, welches dem zukünftigen Einsatzgebiet ähnelt, nicht vernachlässigt werden. Dabei muss sowohl die Fehlerquote als auch die Ergonomie betrachtet werden, wobei eine niedrige Fehlerquote automatisch die Ergonomie verbessert.

Im Folgenden wird am Beispiel SAPlexa die Erprobung und das Ersetzen der Schlüsselbegriffe erklärt.

Zum Beginn der Entwicklung wurden die Schlüsselbegriffe „Start Number“ und „End Number“ genutzt, um das zusammensetzen einer neuen Ziffernkette zu Beginnen bzw. zu Beenden. Schnell wurde dabei klar, dass das Wort „Number“ weder zur besseren Erkennung noch zur Ergonomie beiträgt und wurde damit schnell verworfen. Damit waren die neuen ergonomischeren Schlüsselwörter „Start“ und „Stop“ (jeweils englisch ausgesprochen, da ein englisches Spracherkennungsmodell verwendet wurde).

In den im Entwicklungsprozess üblichen Unittest fiel eine häufige Fehlerkennung der beiden Worte auf, was wiederum die Ergonomie der Sprachbedienung durch eine hohe Fehlerquote verschlechterte. Beim genaueren Betrachten der Begriffe über eine Audioverarbeitungssoftware ist zu erkennen, dass die Wörter sich in Verschiedenen Audiovisuellen Formen kaum unterscheiden.

In den untenstehenden Tabellen sind zwei verschiedene Arten zur Visualisierung von Audiodaten.

Die Bilder obige Zeile der Tabellen zeigt ein sogenanntes Spektrogramm oder speziell bei Audiodaten auch Sonagramm genannt, welches die Energiedichte der Daten in bestimmten Frequenzbereichen über einen Zeitraum aufzeigt. Die darauffolgende Zeile zeigt die eher übliche Waveform (Wellenform).

Die Folgende Tabelle stellt die bei SAPlexa sich als problematisch herausgestellten Begriffe gegenüber.

|  |  |
| --- | --- |
| START | STOP |
|  |  |
|  |  |

Die Aufnahmen wurden mit einem für den Privatgebrauch sehr gutem Mikrophon in einer ruhigen Umgebung aufgenommen. Somit sind wenig Verunreinigungen wie Hintergrundrauschen in den Audiosamples enthalten. Es sind zwar Unterschiede zu erkennen, jedoch sind diese eher über den Zeitraum zu erkennen und können bei weniger optimalen Bedingungen sowie anderer Aussprache noch weniger erkennbare Unterschiede aufweisen.

|  |  |
| --- | --- |
| HANA | OKAY |
|  |  |
|  |  |

In der obigen Tabelle stehen sich die „neuen“ ausgetauschten Schlüsselbegriffe „Hana“ und „Okay“ der Anwendung gegenüber. Es sind direkt mehrere Unterschiede zu erkennen, was sich auch in der Fehlerkennungsrate der Speech-to-Text-Anwendung wiederspiegelt. Die Begriffe sind recht kurz geblieben, was sich zusammen mit der geringeren Fehlerquote positiv auf die Ergonomie der Applikation auswirkt. Eventuell könnten Fehlerkennung durch Wörter mit drei bis fünf Silben nochmals gesenkt werden.

### Informationsbeschaffung bei der Groz-Beckert KG

Warum diese Exkursion?

Welche Einblicke?

**Eduard**

## Konzeption des Back-Ends

### Java-Perspektive

#### Auswahl der Speech-to-Text API – CMUSphinx

Bevor mit der Erstellung der SAP Sprachsteuerung begonnen werden konnte, mussten zuerst ein grober Überblick über die zur Verfügung stehenden Speech-to-Text APIs verschafft werden, um eine passende Speech-to-Text API auszuwählen, die die Ansprüche an das Projekt erfüllt. Nach etwas Recherche wurden schon ein paar mögliche Kandidaten gefunden. Es wurde nun versucht diese miteinander zu vergleichen, um eine Speech-to-Text API zu finden die für das Projekt verwendet werden kann. Folgende Speech-to-Text APIs haben wurden etwas genauer betrachtet:

„Google Speech-to-Text API“ über die Google Cloud. Die Vorteile dieser Speech-to-Text Lösung sind eine hohe Genauigkeit und dass sie eine der geringsten Fehlerquoten hat, außerdem erkennt sie viele verschiedene Sprachen schon Standartmäßig. Leider kostet die Nutzung der Speech-to-Text API Geld da sie wie oben schon erwähnt über die Google Cloud läuft. Auch wäre der Datenschutz durch die Cloud Lösung vermutlich etwas schwerer zu bewerkstelligen.

Auch Microsoft hat eine Speech-to-Text API auf dem Markt, mit „Microsoft Cognitive Services“. Diese Speech-to-Text API hat jedoch so ziemlich die gleichen Vorteile aber auch Probleme wie die von Google bereitgestellten Lösung, da sie auch übe die Cloud arbeitet und somit Geld kostet.

Durch die Betrachtung der beiden oben genannten möglichen Lösungen entstand die Erkenntnis, dass eine Cloud Lösung nicht optimal ist, da diese immer mit kosten verbunden sind. Es wäre besser eine offline Speech-to-Text API zu verwenden, da so auch ein Internet zwang entfällt. Es wurde bei der weiteren suche nun nach offline Varianten gesucht und auch 3 weitere Kandidaten gefunden.

„IBM Watson“ und „Speechmatics“ sind oft benutze offline Speech-to-Text APIs, jedoch kosten beide leider auch Geld, was bei einer Professionellen Anwendung kein großes Problem darstellt, da diese Varianten oft eine sehr gute Qualität aufweisen, einen guten Support bieten und auch regelmäßig geupdatet werden. Bei einem Projektstudium jedoch nicht wirklich infrage kommen.

Eine weitere offline Speech-to-Text API ist die Open Source Variante „CMUSphinx“, welche von der „Carnegie Mellon University (CMU)“ veröffentlich wurde und auch regelmäßig mit Updates versorgt wird. Sie wird oft bei Projekten an Universitäten und allgemein zu Lehrzwecken eingesetzt, da sie eine Kostenlose offline Lösung, eine gute Qualität mit geringer Fehlerquote, die Dokumentation gut und ausführlich ist und viele Tutorials zu ihrer Nutzung im Internet zur Verfügung stehen. Außerdem hatten 2 Personen in der Projektgruppe schon etwas Erfahrung mit „CMUSphinx“ da sie bei dem „Makeathon“ and der Hochschule eingesetzt wurde und die beiden Gruppenmitglieder dort teilgenommen hatten.

Nach der Sichtung dieser verscheiden Speech-to-Text APIs wurde sich für „CMUSphinx“ als Speech-to-Text API entschieden, da diese Variante Kostenlos ist, eine gute Dokumentation besitzt und die bereits (etwas) vorhandene Erfahrung praktisch ist.

#### Speech-to-Text - Implementierung

Bevor mit der eigentlichen Programmierung gestartet werden konnte, wurden zuerst ein paar Online Tutorial Videos angeschaut, um sich einen Überblick über die Sprachsteuerung zu verschaffen. Nachdem man sich mit den Grundfunktionen der Sprachsteuerung und der Bibliotheken auseinandergesetzt hat. Anhand von ein paar Dokumentationen zur Verwendung von CMUSphinx, wurde dann das erste Grundgerüst der Sprachsteuerung programmiert.

Vor dem weiteren Programmieren musste noch eine weitere Ramen Bedingung geklärt werden und zwar ob man eine gesamte Sprache erkennen möchte, also z.B. Deutsch oder Englisch, oder ob man mithilfe einer sogenannten „Grammatik“ nur ausgewählte Wörter erkennen möchte. In einer Grammatik kann man die Wörter, die man erkennt haben möchte reinschreiben, um False-Positives zu vermeiden, indem man die zu erkennende „Grammatik“ eingrenzt. Im rahmen unseres Projektes wurde sich auf die Wörter:

**HANA:** Wird zum Start der Spracherkennung benutzt, nach dem die Anwendung das Schlüsselwort „HANA“ erkennt, beginnt sie nun nur noch auf die folgenden beschrieben Ziffern zu reagieren.

**Zero; one; two; three; four; five; six; seven; eight; nine:** Die Spracherkennung erkennt die gesagten einzelnen Ziffern und fügt sie in die Anwendungslogik ein und zeigt sie auch in der GUI auf.

**Select:** „Select“ wählt die mit dem Schlüsselwort verbundene Eingabe aus, um weiter im Menüfluss voranzuschreiten.

**Okay:** Mit dem Schlüsselwort „Okay“ kann man eine Eingabe bestätigen und zum nächsten Menu bzw. zur nächsten Eingabe voranschreiten.

**Book Order:** Mit diesem Schlüsselbegriff wird am Ende der Wareneingang im SAP System gebucht.

Es wurden bewusst nur eine kleine Menge an Schlüsselworten angelegt, um eine Einfachere Bedienung zu gewährleisten. So wie in 5.1.2 beschrieben wurde, haben wir die Schlüsselworte so angepasst das sie von der Spracherkennung gut voneinander unterschieden werden können.

Im folgenden Absatz wird der Ablauf der Spracherkennung beschrieben.

Am Anfang des Spracherkennungszyklus wartet die Anwendung auf den Begriff „HANA“ um eine neue Suche nach Bestellnummern zu initialisieren. Nach der erfolgreichen Erkennung des Schlüsselwortes können nun die Ziffern der Bestellnummer genannt werden. Die Anwendungslogik erkennt die Ziffern und wandelt sie in integer Zahlen um, um sie verarbeiten und auf der GUI anzeigen zu können. Nach jeder erkannten Zahl werden die Passenden Bestellnummern auf dem GUI angezeigt, damit der Benutzer bei mehreren Angezeigten Bestellnummern gegebenenfalls die passende Nummer mit dem Befehl „Select“ auswählen kann. Nach Bestätigung der richtigen Bestellnummer wird die in 5.1.1 beschriebene Detailansicht aufgerufen. Daraufhin kann der gesamte Wareneingang mit „Book Order“ gebucht werden.

//TO DO: Wenn nicht woanders vertreten, noch mehr auf nutzten von Spracherkennung eingehen

#### (Auswahl der) Java Libraries – SWT

**HÜSO**

#### Eingehen von technischer Schuld

Ein Punkt, der leider aus Zeittechnischer perspektive nicht bewerkstelligt werden konnte, ist das sogenannte „Training“ (oder was auch möglich wäre eine „acoustic model adaptation“) der Sprachsteuerung. „CMUSphinx“ bietet nämlich die Möglichkeit mithilfe von „.wav“ Dateien „Trainiert“ zu werden, also z.B. Spezielle Wörter wie z.B. „SAPLEXA“ zu lernen, da dieses ja kein gewöhnliches Wort einer bekannten Sprache ist oder eine ganze neue Sprache zu lernen falls diese noch nicht Standard mäßig von „CMUSphinx“ unterstützt wird. Es ist auch möglich einen Dialekt zu lernen, was z.B. bei „Groz Beckert“ (auf der Alb, wo manchmal auch nicht unbedingt hochdeutsch gesprochen wird) ein Vorteil oder sogar eine Voraussetzung sein könnte. Die .wav Dateien können dann z.B. die Aufnahmen der Stimme einer Person enthalten, die ein Bestimmtes oder mehrerer Bestimmte Worte (oder auch Sätze bzw. Längere Texte für eine neue Sprache) wiederholen um die Stimme bzw. den Dialekt oder eine neue Sprache zu erlenen. Das Trainieren von CMUSphinx ist jedoch nicht einfach und erfordert auch, wenn man es richtig machen möchte, viele Dateien und aufgenommene Stimmen von Personen. Das trainieren an sich ist auch etwas komplizierter und hätte leider in dem Projektzeitraum keinen Platz mehr gefunden.

Ein weiterer Punkt, der leider nicht mehr in den Zeitlichen Ablauf untergebracht werden konnten, war der Abbruch einer Buchung im GUI oder ein Sprung zurück auf die Vorherige Seite. Dies wäre natürlich noch ein sehr wichtiger Punkt der unerlässlich für die Menüführung ist, da ein Abbruch einer Buchung oder eine Korrektur eine Eingabe oft gebraucht wird. Bei einer Sprachsteuerung kann es vorkommen, vor Allem wenn sie in einem Bereich eingesetzt wird, in dem es viele Hintergrundgeräusche gibt.

ERWEITERN EVENTUELL?

ANGE

### SAP-Perspektive

Als angemeldeter SAP-Benutzer erfolgt die Wareneingangsbuchung über den Transaktionsbefehl MIGO. In unserem Fall handelt es sich um eine Wareneingangsbuchung zu einer zuvor elektronisch hinterlegten und getätigten Bestellung beim Lieferanten. Unter Angabe der Bestellnummer kann somit im Anschluss die Buchung aller gelieferten Positionen vorgenommen werden. Das anschließende Unterkapitel soll diesen Vorgang aus Sicht der Datenbank-Transaktionen erläutern.

#### Technischer Vorgang einer Wareneingangsbuchung im SAP

Im Hintergrund des o.g. Buchungsprozesses spielen in SAP HANA 3 systemeigene Datenbanktabellen eine wesentliche Rolle. Diese sind im Folgenden mit deren Funktionsbezeichnungen aufgeführt.

|  |  |
| --- | --- |
| Name | **Kurzbeschreibung der Datenbanktabelle** |
| Informationsgehalt |
| **EKKO** | **Einkaufsbelegkopf** |
| Die EKKO-Tabelle enthält übergeordnete Informationen zu den im System hinterlegten Einkaufsbelegen. Derartige Informationen können beispielsweise Belegnummern, Zeitstempel zum Anlagezeitpunkt, Lieferantennummer oder auch der Buchungskreis selbst sein. |
| **EKPO** | **Einkaufsbelegposition** |
| Die EKPO-Tabelle enthält im Vergleich zur EKKO-Tabelle nähere Informationen zu den im Einkaufsbeleg hinterlegten Positionen. Hierin werden Informationen zu Materialnummern, Lagerort, Werk oder auch Bestellmenge gelistet. |
| **EKET** | **Lieferplaneinteilungen** |
| Als Nachweis für den Warenverkehr dient die Datenbanktabelle EKET. Im Rahmen dieses Projekts steht die darin enthaltene Information zu bereits gelieferten Positionen im Mittelpunkt. Damit wird u. A. die Verwaltung von Teillieferungen ermöglicht. |

Tabelle 5.1 Relevante Datenbanktabellen in HANA

Der Bestellbeleg beinhaltet i.d.R. mindestens eine Bestellposition. Hierbei besteht die Möglichkeit den Wareneingang zu den einzelnen Bestellpositionen sowohl nach Eintreffen einer einmaligen, vollständigen Lieferung, als auch nach Eintreffen mehrmaliger Teillieferungen zu verbuchen. Die Beziehungen zwischen den o.g. Tabellen lassen sich demnach als folgende Kardinalitäten beschreiben:



Abbildung 4.7 Kardinalitäten zwischen EKKO, EKPO und EKET[[2]](#footnote-2)

Der Zugriff auf die genannten Datenbanktabellen soll über sog. Funktionsbausteine erfolgen, die von außerhalb des SAP-Systems zugänglich sind. Um die Funktionsweise dieser Funktionsbausteine besser zu illustrieren, werden im nächsten Unterkapitel zunächst die relevanten Datenstrukturen aus dem SAP System vorgestellt, welche für die Datenübermittlung verwendet werden sollen.

#### Relevante Datenstrukturen im ABAP Dictionary

Als relevante Datenstrukturen werden im Rahmen dieser Ausarbeitung derartige Strukturen verstanden, die im Rahmen der Datenübermittlung zwischen dem SAP-System und der GUI-Anwendung verwendet werden. Dazu gehören im Wesentlichen die (Zeilen-)Strukturen als solches und die Tabellentypen. Diese können im ABAP Dictionary unter der Transaktion SE11 neu definiert und systemintern in der ABAP Programmierung verwendet werden. Durch die Definition neuer, flacher Datenstrukturen soll die Datenverarbeitung effizienter, zweckmäßiger und übersichtlicher gestaltet werden. Nicht zuletzt können sich daraus Performance-Verbesserungen ergeben, die mit der Vermeidung von entbehrlichen Datenabfragen und -übertragungen einhergehen. Aus diesem Grund wurde bewusst auf bereits vordefinierte Systemstrukturen verzichtet.

Tabellentypen können sich auf eine vordefinierte Struktur beziehen, welche die Zeilenstruktur für den Tabellentyp vorgibt. Programminterne ABAP-Objekte, die demnach auf einen Tabellentypen verweisen, sind interne Tabellen des entsprechenden Zeilentyps aus der Struktur.[[3]](#footnote-3) Die anschließende Abbildung soll die Beziehungen zwischen den einzelnen Elementen und deren Bezug zu vordefinierten Datentypen visualisieren. Dabei ist noch hinzuzufügen, dass im Sinne der Übersichtlichkeit alle referenzierten Komponententypen (in der Abbildung als Spalte bezeichnet), auf welche sich die einzelnen Komponenten der Struktur primär beziehen, ausgeblendet wurden. Stattdessen wird der zugrundeliegende Datentyp visualisiert. Die vollständigen Bezeichnungen zu den Akronymen aus den Typdefinitionen können im Abkürzungsverzeichnis nachgeschlagen werden.



Abbildung 4.8 Tabellentypen - Strukturen - Datentypen

Mithilfe dieser erzeugten Strukturen werden im nächsten Schritt die Ausgabe-Parameter der Funktionsbausteine definiert.

|  |
| --- |
| Unter Verwendung von Mengen- oder auch Währungskomponenten muss zusätzlich ein Referenzfeld als Einheitenschlüssel hinterlegt werden, welches die Mengen- bzw. Währungseinheit spezifiziert. Dies wird in der Einzelfeldpflege der Strukturkomponente vorgenommen.[[4]](#footnote-4) |

#### Implementierung der Funktionsbausteine

Im SAP ABAP Kontext dienen Funktionsbausteine zur Kapselung bzw. Auslagerung von Programmcode, um einen globale Zugriff auf dessen Funktionalität im SAP-System zu gewährleisten.[[5]](#footnote-5) Unterschieden wird dabei ein folgende Arten von Funktionsbausteinen:

* Normaler Funktionsbaustein
* Remote fähiger Funktionsbaustein – (Remote Function Call = RFC)
* Verbuchungsbaustein
  + Start sofort
  + Start sofort – nicht nachverbuchbar
  + Start verzögert
  + Sammellauf

Wie die jeweiligen Bezeichnungen richtig vermuten lassen, wird der Zugriff auf die relevanten Informationen aus Sicht eines Fremdsystems (hier: JAVA-GUI Programm) nur über den o.g. RFC-Baustein möglich, da diese von außerhalb zugänglich sind und damit eine wichtige Voraussetzung für eine Schnittstelle zwischen SAP und JAVA erfüllen. Zu den übrigen Bausteinarten lässt sich vereinfacht darstellen, dass diese keinen Zugriff aus Fremdsystemen gewähren und sich demnach nicht für unser Projekt eignen. Normale Funktionsbausteine stehen primär für die Kapselung von Programmcode, wie z.B. einem Einheiten-Umrechner, wohingegen Verbuchungsbausteine die Manipulation von Datenbanktabellen fokussieren und deren eigenen SAP-seitigen Prüfmechanismus (Logical Unit of Work, kurz: LUW) zur Wahrung der Datenkonsistenz auslösen.[[6]](#footnote-6)

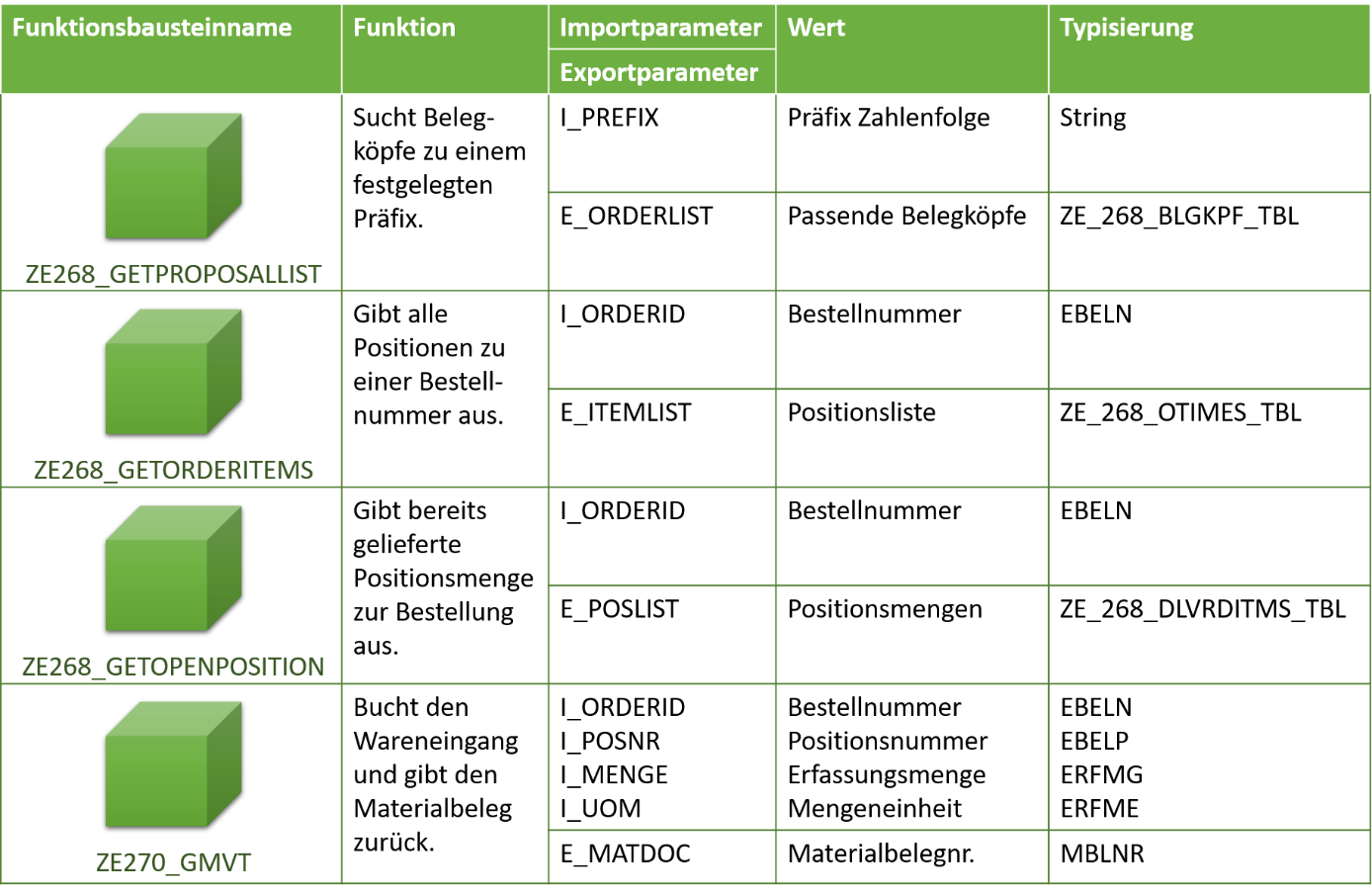
**

Tabelle 5.3 Definierte RFC-Funktionsbausteine in SAP

Funktionsbausteine bieten grundsätzlich die Möglichkeit zur Pflege von optionalen Import- bzw. Export-Parametern für den Datenaustausch mit dem aufrufenden Programm. Das SAP-System selbst stellt zahlreiche Funktionsbausteine, wie z.B. die BAPI Bausteine, zur sofortigen Verwendung bereit. Da sich die Quellcode-Programmierung der Funktionsbausteine in unserem Fall überwiegend auf relativ simple Open-SQL-Statements beschränkt, wurden für das SAPlexa-Projekt eigene, zweckorientierte Funktionsbausteine für den Informationsaustausch definiert. Die RFC-Bausteine sind in programmlogischer Reihenfolge in obiger Tabelle 5.3 visualisiert. Den neu hinzukommenden Typisierungen ERFMG (MENGE), ERFME (Mengeneinheitsschlüssel bzw. Referenzfeld zu ERFMG) und MBLNR (EBELN) liegt jeweils dieselbe Datenstruktur der in Klammer stehenden Datenelementen zugrunde.



Abbildung 4.9 Quellcode ZE268\_GETPROPOSALLIST

Um die Nutzung der Sprachsteuerung, und damit einhergehend die Eingabe der Bestellnummer, ergonomisch zu halten, wurde der Suchmechanismus nach Bestellbelegen im betroffenen Funktionsbaustein mit zwei Wildcards (siehe: %-Symbole) versehen. Damit kann der Benutzer in Kürze zu seinem gewünschten Beleg gelangen ohne dabei eine vorgegebene Abfragestruktur einhalten zu müssen. Der Vorteil hieraus liegt in der Zusammensetzung bzw. Nummernkreisstruktur der Bestellnummer. Bei einer feststehenden Nummernlänge für Bestellbelege von 10 Ziffern kann damit erreicht werden, dass lediglich eine überschaubare Anzahl an Endziffern diktiert werden müssen.

Die Ablauflogik der JAVA Programmierung bewirkt also, dass nach jeder schrittweisen Zahleneingabe bzw.-ansage durch den Benutzer eine neue Anfrage an das SAP System gesendet. Die hierfür zuständige Open SQL Abfrage beinhaltet dabei die gesuchte Zahlenfolge als String-Parameter und übergibt alle Einträge aus der EKKO-Tabelle an das aufrufende JAVA Programm, welche die entsprechende Zahlenfolge beinhalten. Die Programmierung der SAP-Schnittstelle steht im nachfolgenden Unterkapitel im Fokus.

|  |
| --- |
| **Hinweis** Alle vollständigen Quellcodes zu den Programmierungen aller Bestandteile dieses Projekts sind im Anhang aufgeführt. |

### SAP Java Connector – Die Schnittstelle

Die Implementierung einer Schnittstelle zwischen Java und SAP wird durch den SAP-eigenen Java Connector realisiert. Für das Java Programm wird sowohl die physische JAR-Bibliothek des Java Connectors, als auch ein gültiges Benutzerkonto zur Anmeldung am SAP System benötigt. Alle zur Anmeldung benötigten Benutzer- und Serverinformationen werden in einer ausgelagerten JCoDestination-Datei in UTF-8 formatierter Textform gelistet.

Das folgende Abbild der Textdatei zeigt die zugrundeliegende Konfigurationen für das erfolgreiche Verbinden des Java Connectors mit unserem gehosteten SAP-Server aus Magdeburg. Unter anderem werden darin Sprache, Mandant, Anmeldedaten des Users und Serverspezifikation festgelegt.

jco.client.lang=de

jco.client.client=202

jco.client.passwd=SAPLEXA

jco.client.user=RFC\_SAPLEXA

jco.client.sysnr=95

jco.client.ashost=/H/cloud.ucc.ovgu.de/S/3299/H/a95z.2.ucc.md

jco.destination.peak\_limit=10

jco.destination.pool\_capacity=3

Nach einem erfolgreichen Ping-Test und Verbindungsaufbau stehen nun für das Ausführen der im SAP-System definierten Funktionsbausteine zahlreiche Java-Funktionen zur Verfügung. Im Rahmen dieser Ausarbeitung wird dabei nur auf die für dieses Projekt praktisch relevanten Funktionen eingegangen. Unter der Annahme, die Variable repository sei vom Typ JCoRepository, function vom Typ JCoFunction und dvom Typ JCoDestination, kann ein möglicher Java-seitiger Funktionsaufruf wie folgt aussehen.



Abbildung 4.10 Java-seitiger Funktionsaufruf auf den Baustein ZE268\_GETPROPOSALLIST

Die Funktionsweise des Zugriffs auf einen definierten Funktionsbaustein aus SAP sollte in obiger Abbildung selbsterklärend sein. Um die Datenströme zu klassifizieren werden die eigens hierfür kreierten Objektklassen „Order“ und „Item“ mit deren zugehörigen Felder instanziiert. Durch das schleifenweise Abfragen der Ergebniszeilen aus der exportierten (hier) internen Tabelle des Funktionsbausteins werden die einzelnen Objektinstanzen erzeugt, gefüllt und einer Collection zugewiesen, um die darauf anschließende Datenverarbeitung zu strukturieren.

Die tatsächliche Buchung des Wareneingangs greift erstmalig auf vordefinierte Bausteine in SAP zurück, den sogenannten BAPI-Bausteinen (Business Application Programming Interface). Durch die Verwendung der Bausteine können weitere für die Buchung benötigte Bestellinformationen abgefragt werden. Die Programmierung aus dem Funktionsbaustein ZE270\_GMVT nutzt die nachstehenden BAPI-Funktionen.

* ***BAPI\_PO\_GETDETAIL***Dieser BAPI Baustein ermöglicht die Ausgabe einer detaillierten Auflistung von Bestellpositionen, die innerhalb einer ausgewählten Bestellung aufgeführt sind. ***[[7]](#footnote-7)*** Als notwendiger Importparameter hierfür ist die Belegnummer zu erwähnen.
* ***BAPI\_GOODSMVT\_CREATE***Bei diesem Baustein handelt es sich um ein universelles BAP-Interface, welches die Buchung einer Warenbewegung auf Datenbankebene vornimmt.[[8]](#footnote-8)
* ***BAPI\_TRANSACTION\_COMMIT***Die Manipulation von Daten durch BAPI-Bausteine erfordert eine abschließende COMMIT Anweisung, um alle vorgenommen Änderungen final zu speichern. Grundsätzlich hängt dies mit der zuvor erwähnten LUW (SAP-Datenbankmechanismus zur Wahrung der Datenkonsistenz).[[9]](#footnote-9) Die Ausführung dieses BAPI-Bausteins umfasst dabei mehr als eine simple inline Commit-Work Anweisung. Weitergehende Details hierzu werden aus Gründen unzureichender Relevanz im Rahmen dieser Ausarbeitung ausgeblendet.

Als Export-Parameter übergibt der ZE270\_GMVT Baustein den gebuchten Materialbeleg als numerischen String an das aufrufende Programm zurück. Geeignete Prüfmechanismen vor der Buchung eines Falschbelegs werden im aufrufenden Programm (hier: Java Programm) implementiert.

# Zukünftige Optimierungs- und Erweiterungsmöglichkeiten

Für die Zukunft, was können kommende Semester daran verbessern oder wo weiterarbeiten.

**HÜSO**

# Reflexion und Fazit

**DANI ? ODER GEMEINSAM**

# Literaturverzeichnis

**BAPI\_GOODSMVT\_CREATE** [Online] / Verf. Wiki SAP // Goods Movements with BAPI. - 06. Januar 2020. - https://wiki.scn.sap.com/wiki/display/ERPSCM/Goods+Movements+with+BAPI.

**BAPI\_PO\_GETDETAIL** [Online] // CONSULT. - 06. Januar 2020. - https://www.consolut.com/s/sap-ides-zugriff/d/e/doc/E-BAPI\_PO\_GETDETAIL/.

**BAPI\_TRANSACTION\_COMMIT** [Online] / Verf. Wiki SAP. - 06. Januar 2020. - https://wiki.scn.sap.com/wiki/display/ABAP/BAPI\_TRANSACTION\_COMMIT+versus+COMMIT+WORK.

**Eclipse User Interface Guidelines** [Online] / Verf. al. N. Edgar et // Official Eclipse Wiki. - 2016. - 02.. Februar 2020. - http://wiki.eclipse.org/User\_Interface\_Guidelines.

**EKKO-Tabelle** [Online] / Verf. Böselager Gerrit // ERP-Yourself. - 26. Dezember 2019. - https://www.erpyourself.net/de/sap-tabellen/EKKO.html.

**Funktionsbausteine** [Online] / Verf. SAP. - 26. Dezember 2019. - https://help.sap.com/doc/abapdocu\_751\_index\_htm/7.51/de-DE/abenabap\_functions.htm.

**Java-Entwicklung mit Eclipse 3.2** [Buch] / Verf. Daum B.. - Heidelberg : D.Punkt Verlag, 2006. - Bd. 4. Edition.

**Mengenfelder** [Online] / Verf. SAP. - 26. Dezember 2019. - https://help.sap.com/doc/abapdocu\_752\_index\_htm/7.52/de-de/abenddic\_quantity\_field.htm.

**Mobile Informationssysteme** [Online] / Verf. Becker Peter // Fachhochschule Rhein-Sieg. - 2011. - 03.. Februar 2020. - http://www2.inf.h-brs.de/~pbecke2m/mobis2/script.pdf.

**Professional Java Native Interfaces with SWT/JFace** [Buch] / Verf. Guojie Jackwind Li. - [s.l.] : Wrox, 2009. - Bd. 1. Edition.

**Tabellentypen** [Online] / Verf. SAP. - 26. Dezember 2019. - https://help.sap.com/doc/abapdocu\_751\_index\_html/7.51/de-DE/abenddic\_table\_types.htm.

**Verbuchungsbausteine** [Online] / Verf. SAP. - 26. Dezember 2019. - https://help.sap.com/doc/abapdocu\_751\_index\_htm/7.51/de-DE/abenupdate\_function\_module\_glosry.htm.

# Anhangsverzeichnis

# Anhang (Quellcodes)

Code 1 Funktionsbaustein ZE268\_GETOPENPOSITION

FUNCTION ZE268\_GETOPENPOSITION.  
\*"---------------------------------------------------------------  
\*"\*"Lokale Schnittstelle:  
\*"  IMPORTING  
\*"     VALUE(I\_ORDERID) TYPE  EBELN  
\*"  EXPORTING  
\*"     VALUE(E\_POSLIST) TYPE  ZE\_268\_DLVRDITMS\_TBL  
\*"---------------------------------------------------------------  
  
SELECT EKET~WEMNG FROM EKET INTO CORRESPONDING FIELDS OF TABLE E\_POSLIST WHERE EKET~EBELN = I\_ORDERID.  
  
  
ENDFUNCTION.

Code 2 Funktionsbaustein ZE268\_GETORDERITEMS

FUNCTION ZE268\_GETORDERITEMS.  
\*"---------------------------------------------------------------  
\*"\*"Lokale Schnittstelle:  
\*"  IMPORTING  
\*"     VALUE(I\_ORDERID) TYPE  EBELN  
\*"  EXPORTING  
\*"     VALUE(E\_ITEMLIST) TYPE  ZE\_268\_OITEMS\_TBL  
\*"---------------------------------------------------------------  
\*\* Retrieve Order Positions

SELECT \* FROM EKPO INTO CORRESPONDING FIELDS OF TABLE E\_ITEMLIST WHERE EKPO~EBELN = I\_ORDERID.  
  
  
ENDFUNCTION.

Code 3 Funktionsbaustein ZE268\_GETPROPOSALLIST

FUNCTION ZE268\_GETPROPOSALLIST .  
\*"---------------------------------------------------------------  
\*"\*"Lokale Schnittstelle:  
\*"  IMPORTING  
\*"     VALUE(I\_PREFIX) TYPE  STRING  
\*"  EXPORTING  
\*"     VALUE(E\_ORDERLIST) TYPE  ZE\_268\_BLGKPF\_TBL  
\*"---------------------------------------------------------------  
  
  
  DATA L\_PREFIX TYPE STRING.  
  
  
  CONCATENATE '%' I\_PREFIX '%' INTO L\_PREFIX.  
  SELECT EBELN, LIFNR, BEDAT FROM EKKO  
    INTO CORRESPONDING FIELDS OF TABLE @E\_ORDERLIST  
    WHERE EKKO~EBELN LIKE @L\_PREFIX.  
  
  
ENDFUNCTION.

Code 4 Funktionsbaustein ZE270\_GMVT

FUNCTION ZE270\_GMVT.  
\*"----------------------------------------------------------------------  
\*"\*"Lokale Schnittstelle:  
\*"  IMPORTING  
\*"     VALUE(I\_ORDERID) TYPE  EBELN  
\*"     VALUE(I\_POSNR) TYPE  EBELP  
\*"     VALUE(I\_MENGE) TYPE  ERFMG  
\*"     VALUE(I\_UOM) TYPE  ERFME  
\*"  EXPORTING  
\*"     VALUE(E\_MATDOC) TYPE  MBLNR  
\*"----------------------------------------------------------------------  
  
Data: l\_goodsmvt\_header type BAPI2017\_GM\_HEAD\_01,  
      lt\_goodsmvt\_item type TABLE OF BAPI2017\_GM\_ITEM\_CREATE   
 WITH HEADER LINE,  
      lt\_return type TABLE OF BAPIRET2,  
      lt\_items type TABLE OF BAPIEKPO.  
  
CALL FUNCTION 'BAPI\_PO\_GETDETAIL'  
  EXPORTING  
    PURCHASEORDER                   = i\_orderid  
  TABLES  
   PO\_ITEMS                         = lt\_items  
    .  
  
l\_goodsmvt\_header-pstng\_date = sy-datum.  
l\_goodsmvt\_header-doc\_date = sy-datum.  
lt\_goodsmvt\_item-entry\_qnt = I\_MENGE.  
lt\_goodsmvt\_item-entry\_uom = I\_UOM.  
lt\_goodsmvt\_item-move\_type = '101'.      
lt\_goodsmvt\_item-plant = 'HD00'.  
lt\_goodsmvt\_item-stge\_loc = 'RM00'.  
lt\_goodsmvt\_item-material = 'R1-002'.  
lt\_goodsmvt\_item-po\_number = I\_ORDERID.  
lt\_goodsmvt\_item-po\_item = I\_POSNR.  
lt\_goodsmvt\_item-mvt\_ind = 'B'.    
  
APPEND lt\_goodsmvt\_item.  
  
  
CALL FUNCTION 'BAPI\_GOODSMVT\_CREATE'  
  EXPORTING  
    GOODSMVT\_HEADER               = l\_goodsmvt\_header  
    GOODSMVT\_CODE                 = '01'  
  IMPORTING  
    MATERIALDOCUMENT              = E\_MATDOC  
  TABLES  
    GOODSMVT\_ITEM                 = lt\_goodsmvt\_item  
    RETURN                        = lt\_return

.  
  
CALL FUNCTION 'BAPI\_TRANSACTION\_COMMIT'.  
  
ENDFUNCTION.

1. GUI: **G**raphical **U**ser **I**nterface [↑](#footnote-ref-1)
2. EKKO-Tabelle, Böselager. [↑](#footnote-ref-2)
3. Tabellentypen, SAP. [↑](#footnote-ref-3)
4. Mengenfelder, SAP. [↑](#footnote-ref-4)
5. Funktionsbausteine, SAP. [↑](#footnote-ref-5)
6. Verbuchungsfunktionsbaustein, SAP. [↑](#footnote-ref-6)
7. BAPI\_PO\_GETDETAIL, CONSULT. [↑](#footnote-ref-7)
8. BAPI\_GOODSMVT\_CREATE, Wiki SAP. [↑](#footnote-ref-8)
9. BAPI\_TRANSACTION\_COMMIT, Wiki SAP. [↑](#footnote-ref-9)