Hochschule für angewandte Wissenschaften München
Fakultät 03 – Maschinenbau, Fahrzeugtechnik und Flugzeugtechnik

Hochschule München University of Applied Sciences

Wissenschaftliche Arbeit zur Erlangung des Grades
Bachelor of Science (B.Sc)

Lieferantenmanagement und Kostenkalkulator

Supplier management and cost calculator

Verfasser: Florian Wandl
Matrikelnummer: 32175518
Studiengang: Fahrzeugtechnik
E-Mail-Adresse: florian.wandl@hm.edu
Betreuer: Herr Dipl.-Ing. Armin Rohnen LbA
Abgabetermin: 14.02.2023

Eigenständigkeitserklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht. Außerdem bestätige ich, dass diese Arbeit bisher weder ganz noch in Teilen als Prüfungsleistung vorgelegt oder veröffentlicht wurde.

Reichertsheim, 14.02.2023
Ort, Datum


Unterschrift

Abstract

Im Projekt der Glasboilermaschine sind viele unterschiedliche Parteien beteiligt und somit ist eine hohe Anzahl an Daten entstanden, wodurch der Überblick über die Kosten einer Maschine gelitten hat. Die Kalkulation von Gesamtpreisen der Maschinen ist aufgrund der wachsenden Komplexität nicht mehr in einer Excel-Liste möglich. Deshalb wird im Rahmen dieser Bachelorarbeit ein Datenbankmodell entwickelt und daraus eine Datenbank auf Basis von MySQL erstellt, welche alle Bauteildaten speichert. Dazugehörend wird eine MATLAB-GUI entwickelt, welche die Daten in die Datenbank aufnimmt, Teilenummern generiert, Maschinenpreise automatisch in Losgrößen berechnet und die zugehörigen Bestelllisten erstellt.

In the glass boiler machine project, many different parties are involved and thus a high number of data has been generated, which has made the overview of the costs of a machine suffer. The calculation of total prices of the machines is no longer possible in an Excel list due to the growing complexity. Therefore, in the context of this bachelor thesis, a database model is developed and from it a database based on MySQL is created, which stores all component data. In addition, a MATLAB-GUI is developed, which takes the data into the database, generates part numbers, calculates machine prices automatically in lot sizes and creates the corresponding order lists.

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abkürzung	Beschreibung
DB	Database (dt. Datenbank)
FK	Foreign key (dt. Fremdschlüssel)
GUI	Graphical User Interface (dt. grafische Benutzeroberfläche)
ID	Identifier (dt. Identifikationsnummer)
JDBC	Java Database Connectivity (dt. Java Datenbankverbindungsfähigkeit)
ODBC	Open Database Connectivity (dt. offene Datenbankverbindungsfähigkeit)
PK	Primary key (dt. Primärschlüssel)
RDBMS	Relationales Datenbank-Management-System
SQL	Structured Query Language (Datenbanksprache)

1 Einleitung

Beim Projekt der Glasboilermaschine handelt es sich um eine neuartige Kaffeemaschine, welche unter Leitung von Herrn Dipl.-Ing. A. Rohnen stetig weiterentwickelt wird. Im Projekt sind viele Studierende im Rahmen von Abschluss- und Projektarbeiten beteiligt, wodurch die Anzahl an Datensätzen gestiegen ist und auch weiterhin wachsen wird. Vor Beginn dieser Arbeit wurde bereits ein Teilenummernsystem festgelegt, um die Übersicht behalten zu können, welches allerdings noch nicht vollumfänglich umgesetzt ist. Außerdem sind durch die unterschiedliche Bearbeitung von Baugruppen und Komponenten durch andere Parteien auch die zugehörigen Stücklisten nicht einheitlich. Bisher wurde zur Preiskalkulation der Maschine eine Excelliste genutzt, welche mit der zunehmenden Komplexität nicht mehr zielführend ist. Zur Verstreuung der Daten kommen die Besonderheiten des Lieferantenmanagements hinzu, welche einen Einfluss auf die Preiskalkulation haben. So sollen im Kostenkalkulator die Faktoren Rabatt, Lieferkosten, Mindestbestellmengen und Mindestbestellsummen mit einbezogen werden. Außerdem gibt es für manche Bauteile mehrere Lieferanten und es muss immer das wirtschaftlichere Bauteil gewählt werden.

Deshalb wird für die Aufgabe des Lieferantenmanagements und Kostenkalkulators eine Datenbank über die Plattform MySQL erstellt, in welcher alle vorhandenen und zukünftigen Bauteile einpflegt werden sollen. Die Datenbank soll später auf einen Server gehostet werden, sodass auch von außerhalb darauf zugegriffen werden kann. Für die weitere Umsetzung des Themas wird eine MATLAB-GUI entwickelt, welche automatisch Teilenummern generiert und die Daten in die Datenbank aufnimmt. So können Teilenummern in der Konstruktionsphase generiert und vergeben werden und die Daten für jedes einzelne Bauteil sind zentral gespeichert. In der GUI wird ebenfalls ein Kostenkalkulations-Tool erstellt, welches automatisch die Preise einer Maschine in vordefinierten Losgrößen bestimmt. Bei den Losgrößen handelt es sich um Kleinserien. Für die Kalkulation werden alle oben genannten Einflussfaktoren mit einbezogen. Zusätzlich wird auch die Möglichkeit gegeben, die zugehörigen Bestelllisten automatisch für die gewählte Kalkulation zu erzeugen. Zur Abrundung der MATLAB-GUI wird die Funktion vorgesehen, Datensätze direkt in der Datenbank manipulieren zu können. Das Projekt wird im Augenmerk auf die Glasboilermaschine entwickelt, soll aber auch für die Labormaschine anwendbar sein.

Für die Umsetzung des Lieferantenmanagements und Kostenkalkulators ist also eine Datenbank und eine MATLAB-GUI nötigt. Die MATLAB-GUI besteht hierbei aus drei Teilen, dem Teilenummerngenerator, dem Kostenkalkulator und dem Datenbank Editor. Daraus lassen sich die vier Schwerpunkte des Themas ableiten, die in den folgenden Kapiteln strukturiert dokumentiert werden.

2 Entwicklung eines Datenbankmodells

Ein Datenbankmodell dient als Grundlage für die Organisation und Strukturierung von Daten in einer Datenbank. Es beschreibt die Entitäten und Beziehungen zwischen diesen sowie deren Attribute und Schlüssel. Eine ordnungsgemäß erstellte und optimierte Datenbankmodellierung kann nicht nur die Datenintegrität sicherstellen, sondern auch die Leistung und Skalierbarkeit der Datenbank verbessern. Für das Thema Lieferantenmanagement und Kostenkalkulator ist es notwendig, ein Datenbankmodell zu erarbeiteten, welches zentral alle Informationen bzgl. der Bauteile und Lieferanten enthält. Dabei ist auf die Einführung eines bestehenden Teilenummernsystems zu achten, welche das Grundgerüst des Modells vorgibt. Das Datenbankmodell soll mittels der Open Source Software MySQL erstellt werden und später auf einem Server laufen. Ziel und Aufgabe der Datenbank ist es, dass über eine MATLAB-GUI Bauteile eingearbeitet werden können, welche dann automatisch eine zugehörige Teilenummer erhalten sowie das Bestimmen von Maschinenpreisen über einen Kostenkalkulator. Dabei gilt es besonders, auf die Datenintegrität zu achten, damit der Datenaustausch zwischen MATLAB und MySQL reibungslos funktioniert. Primär liegt der Fokus der Datenbank zunächst auf der Glasboilermaschine, diese soll jedoch auch für die Labormaschine funktionieren und darauf ausgelegt werden.

2.1 Einführung in Datenbanken

Es gibt viele verschiedene Arten von Datenbanken, die für unterschiedliche Anwendungen verwendet werden können. In dieser Dokumentation werden die vier häufigsten Typen von Datenbanken beschrieben: relationale, dokumentenorientierte, Schlüssel-Werte und graphbasierte Datenbanken [vgl.1, 2]. Dabei ist zu beachten, dass es sich nur bei der relationalen um eine reine SQL-Datenbank handelt. Bei den anderen Typen handelt es sich um sogenannte NoSQL-Datenbanken, welche kein festgelegtes Tabellenschema verfolgen. Dabei wird bewusst versucht Joins, zu vermeiden. Ein Join bildet aus den Datensätzen zweier Tabellen eine Ergebnistabelle.

- Relationale Datenbank:
  Bei dieser Art von Datenbank werden die Daten in verschiedenen Tabellen gespeichert. Jede Tabelle hat einen primären Schlüssel, der verwendet wird, um Datensätze eindeutig zu identifizieren und mehrere Spalten, die verschiedene Attribute der Datensätze beschreiben. Tabellen können miteinander verknüpft werden, um Daten aus mehreren Tabellen abzufragen. Relationale Datenbanken sind eine sehr gebräuchliche Art von Datenbanken. Sie sind sehr gut für die Speicherung von strukturierten und vorhersehbaren Daten geeignet und bieten eine sehr gute Unterstützung für die ACID-Eigenschaften (Atomizität, Konsistenz, Isolation und Dauerhaftigkeit). Sie haben in den letzten Jahrzehnten eine breite Akzeptanz gefunden und werden in vielen Anwendungen und Branchen eingesetzt, von kleinen bis hin zu großen Unternehmensanwendungen, weil sie eine sichere, skalierbare und flexible Möglichkeit zur Verwaltung von Daten bieten. Beispiele für relationale Datenbanken sind MySQL, PostgreSQL und Microsoft SQL Server.

- Dokumentenorientierte Datenbank:
  Die Speicherung und Verwaltung von Daten findet in dieser Form mittels Dokumenten statt, welche im Format JSON oder BSON gespeichert werden. Jedes Dokument hat eine eindeutige Identifikationsnummer und kann mehrere Felder haben, die verschiedene Attribute der Daten beschreiben. Dokumentenorientierte Datenbanken sind gut geeignet für Anwendungen, bei denen Daten häufig geändert werden und nicht in einer festen Struktur gespeichert werden müssen.

- Schlüssel-Werte-Datenbank:
  Die Ablage von Daten erfolgt innerhalb dieser Datenbanken mit einem Schlüssel, der auf einen bestimmten Wert verweist. Jeder Schlüssel muss eindeutig sein. Schlüssel-Werte-Datenbanken sind sehr schnell und einfach zu verwenden, aber sie haben in der Regel keine Abfragemöglichkeiten wie relationale Datenbanken.

- Graphdatenbank:
  In dieser Form der Datenbank werden die Informationen in Form von Knoten und Kanten gespeichert. Knoten repräsentieren Entitäten oder Objekte in der Datenbank und haben einen Typ sowie mehrere Felder, die verschiedene Attribute der Entität beschreiben. Kanten repräsentieren Beziehungen oder Verbindungen zwischen den Knoten. Dieser Ansatz ermöglicht es, komplexe Beziehungen zwischen Daten in einer natürlichen und intuitiven Weise darzustellen und abzufragen.

Betrachtet man nun die zu verarbeitenden Daten für das Lieferantenmanagement und den Kostenkalkulator, ist eine relationale Datenbank die einzig sinnhafte Lösung. Die Erweiterung der Daten, z. B. durch ein neues Teil, soll hierbei zu jeder Zeit möglich sein und in Korrelation zu bereits existierenden Datensätzen stehen. Dies ist in einer relationalen Datenbank möglich durch das Hinzufügen von Zeilen in den entsprechenden Tabellen, unter Einbindung der richtigen Schlüssel. Ein weiteres Ausschlusskriterium für die anderen Datenbanktypen sind die benötigten Datenabfragen für den Kostenkalkulator, welche eine klassische SQL-Anwendung darstellen.

2.1.1 Grundlagen SQL

SQL (Structured Query Language) ist eine standardisierte Sprache, die verwendet wird, um Daten in relationalen Datenbanken zu speichern, abzufragen und zu verwalten [vgl. 1]. Eine SQL-Datenbank besteht aus Tabellen, die in Spalten (Felder) und Zeilen (Datensätze) unterteilt sind und die miteinander durch bestimmte Schlüsselfelder verknüpft sind. Um mit Daten in einer SQL-Datenbank zu arbeiten, gibt es verschiedene SQL-Befehle, die verwendet werden können, wie z.B. SELECT, INSERT, UPDATE und DELETE. Mit diesen Befehlen können Daten abgefragt, eingefügt, aktualisiert oder gelöscht werden. Eine gute Kenntnis der Datenbankarchitektur und der SQL-Syntax ist erforderlich, um effektiv mit Daten in einer SQL-Datenbank arbeiten zu können. Durch SQL wird es ermöglicht, Daten auf eine einfache Art und Weise abzufragen und zu manipulieren, unabhängig von der Größe oder Struktur der Datenbank.

2.1.2 Welche Rolle spielt MySQL

Bei MySQL handelt es sich um ein relationales Datenbank-Management-System (RDBMS), das verwendet wird, um Daten in einer strukturierten Form zu speichern und zu verwalten [vgl. 5]. Besonders die leichte Zusammenarbeit zwischen MATLAB und MySQL sind für die Datenbank von Vorteil. Zusätzlich handelt es sich um ein Open-Source-System, was bedeutet, dass es kostenlos zur Verfügung steht und von einer großen Community von Entwicklern weiterentwickelt wird. Es ist sehr leistungsfähig, skalierbar und zuverlässig und kann daher auch für große und komplexe Anwendungen verwendet werden. Außerdem unterstützt MySQL viele Programmiersprachen, darunter Java, Python, C++ und PHP. Zudem ist MySQL eine der am häufigsten verwendeten Datenbanken für Web-Anwendungen und wird von vielen großen Unternehmen und Organisationen verwendet. Es hat auch eine große Unterstützung von der Entwicklergemeinschaft und es gibt viele Tools und Bibliotheken, die es erleichtern, mit MySQL zu arbeiten. Für die App selbst wird jedoch nicht das Programm MySQL an sich benötigt. Die Datenbank soll nur über die MySQL-Server Anwendung gehostet werden. Die eigentlichen SQL-Befehle werden in MATLAB geschrieben und deswegen ist eine Kompatibilität von größter Bedeutung und MySQL bestens für eine MATLAB Anwendung geeignet.

2.2 Anforderungen an das Datenbankmodell

Generell lassen sich die Daten in drei Teilbereiche aufteilen, für welche diese benötigt werden. Eine der umfangreichsten Anforderungen ist die Umsetzung des Teilenummernsystems. Dieses setzt sich aus den folgenden Elementen zusammen:

Modul - Baugruppe - Komponente - Teil - Version - Typ - Lieferant - Status - Datum

Die einzelnen Segmente werden hierbei mit einem Bindestrich voneinander getrennt. Generell gilt, dass sich jedes Teil mindestens einem Modul und einer Baugruppe zuordnen lässt. Pro Maschine gibt es sieben Module, welchen bereits die logischen Nummern wie folgt zugewiesen sind:

1 - Boiler
2 - Tank
3 - Brühturm
4 - Lanzen
5 - Abtropfbereich
6 - Bodenplatte
7 – Unterbau

Für die Baugruppe beginnt die Nummer zählend mit der logischen eins. Eine Komponente beschreibt den sinnvollen Zusammenschluss von mehreren Bauteilen in einer Baugruppe. Die Komponentennummer beginnt ebenfalls mit der eins oder ist null, falls es keine Komponentenzuordnung gibt. Die Versionsnummer stellt den aktuellen Stand des Bauteils dar und ist ebenfalls eine Ganzzahl.

Für die Segmente Typ und Status sind folgende Buchstaben vorgesehen:

Tabelle 1: Legende Typ und Status

O - ohne Typenzuordnung	O - ohne Status
N - Normteil	E - Entwicklung
K - Konstruktionsteil	F - Freigegeben
	S - Serienteil

Das Datum ist im ISO-Format JJJJMMTT vorgesehen. Der Lieferant wird aus der Liste aller Lieferanten mit der entsprechenden Lieferantennummer ausgewählt. Zurzeit sind ca. 40 unterschiedliche Lieferanten im Projekt involviert.

Als konkretes Beispiel wird die Teilenummern des STM32F411 Nucleo (Microcontrollerboard) erläutert. Die Teilenummer hierfür lautet:

7 - 1 - 1 - 1 - 0 - O - 26 - F – 20210511

und schlüsselt sich nach der oben beschriebenen Logik wie folgt auf:

Modul Unterbau – 1. Baugruppe (Elektronik) – 1. Komponente (Basisplatine) – 1. Bauteil (STM32F411) - keine Version - ohne Typzuordnung (weder Normteil noch Konstruktionsteil) - Lieferant 26 (mouser.de) - Freigegeben am 11. Mai 2021

Mit der entsprechenden Teilenummer lässt sich jedes Teil eindeutig entschlüsseln. Zusätzlich zu dieser besitzt jedes Bauteil auch eine eindeutige textliche Beschreibung bzw. Namen. Ein Problem hierbei stellen allerdings Norm- bzw. Kaufteile da. Eine Schraube kann in unterschiedlichen Modulen als auch Baugruppen vorkommen und wäre somit entweder nur einmal mit der richtigen Nummer zugeordnet oder müsste mehrfach angelegt werden. Um dies zu umgehen wurde entschieden, eigene Module für diese Teile einzufügen, welche im Nummernkreis nach ganz hinten gestellt werden und mit der 999 beginnen. Somit ist auf den ersten Blick erkennbar, um welches Teil es sich handelt und es kann in mehreren Baugruppen mit derselben Teilenummer verwendet werden.

Der zweite Teilbereich für die benötigten Daten ist der Kostenkalkulator. Dieser benötigt weiterführende Informationen zu den Teilenummern. Besonders Preise und Stückzahlen sind hierfür elementar. Eine Herausforderung an das Datenmodell ist hierbei, dass es für manche Teile mehrere Lieferanten geben kann. Der Kostenkalkulator braucht hierfür alle Informationen, um je nach Situation intelligent das günstigere der Teile zu wählen. Außerdem gibt es für manche Lieferanten Lieferkosten, Mindestbestellsummen und Rabatte, welche sich ebenfalls auf die Kalkulation auswirken. Für die Preise von Bauteilen sind die Anfragengrößen von eins, zehn und einhundert vorgesehen.
Da der Kostenkalkulator nicht nur den Endpreis der Maschine bestimmen soll, sondern auch automatisch die zugehörigen Bestelllisten generiert, werden hierfür weitere Informationen wie die zugehörige Bestellnummer und alle benötigten Kontaktdaten des jeweiligen Lieferanten benötigt.

Der letzte Teilbereich der Daten sind jene, die nicht direkt Anwendung für den Teilenummerngenerator und Kostenkalkulator finden. Diese sollen trotzdem in die Datenbank mit eingearbeitet werden, um eine spätere Verarbeitung zu ermöglichen. Sie machen den kleinsten Teil in der Datenbank aus. Die zusätzlichen Informationen umfassen das Material, das Gewicht, die Produktions- und Nachbehandlungsmethode sowie die zugehörigen Produktions-, Nachbehandlungs- und Nebenkosten eines jeden Bauteils.

2.3 Datentypen in einer MySQL-Datenbank

MySQL ist ein relationales Datenbankverwaltungssystem (RDBMS) und unterstützt eine Vielzahl von Datentypen für die Definition von Spalten in einer Tabelle. Im Folgenden ist eine Übersicht der wichtigsten und gängigsten Datentypen in MySQL mit einer dazugehörigen Beschreibung dargestellt [vgl. 4, 5]:

- Ganzzahlen:
  - TINYINT: ganzzahliger Wert im Bereich von -128 bis 127
  - SMALLINT: ganzzahliger Wert im Bereich von -32768 bis 32767
  - MEDIUMINT: ganzzahliger Wert im Bereich von -8388608 bis 8388607
  - INT oder INTEGER: ganzzahliger Wert im Bereich von -2147483648 bis 2147483647
  - BIGINT: ganzzahliger Wert im Bereich von -9223372036854775808 bis 9223372036854775807

- Fließkommazahlen:
  - FLOAT: Ein Gleitkommawert mit bis zu 24 signifikanten Ziffern
  - DOUBLE: Ein Gleitkommawert mit bis zu 53 signifikanten Ziffern
  - DECIMAL: Ein genauer Dezimalwert mit einer festgelegten Anzahl von Nachkommastellen

- Zeichenketten:
  - CHAR: Eine Zeichenkette mit einer festen Länge von bis zu 255 Zeichen
  - VARCHAR: Eine Zeichenkette mit einer variablen Länge von bis zu 65535 Zeichen
  - TEXT: Eine Zeichenkette mit einer variablen Länge von bis zu 4 GB

- Datum/Zeit:
  - DATE: Datum im Format YYYY-MM-DD
  - DATETIME: Datum und Uhrzeit im Format YYYY-MM-DD HH:MM:SS
  - TIMESTAMP: Datum und Uhrzeit im Format YYYY-MM-DD HH:MM:SS (wird automatisch aktualisiert wenn sich die Daten ändern)
  - YEAR: Jahreszahl im Format YYYY oder YY

- Wahr/Falsch (Boolean):
  - BOOLEAN oder BOOL

Angesichts der aufkommenden Daten im angewandten Thema Lieferantenmanagement und Kostenkalkulator werden die Datentypen VARCHAR für alle Beschreibungen und auch für die Teilenummer verwendet. DECIMAL wird für alle Preise und INT für alle sonstigen Zahlen verwendet, besonders für Identifikationsnummern und Stückzahlen. Damit die Währung Euro richtig verarbeitet werden kann, wird der entsprechende Datentyp auf DECIMAL(10,2) festgelegt. Übersetzt bedeutet dies eine Zahl mit zwei Nachkommastellen und zehn Stellen gesamt. Da sich der Gesamtpreis einer Maschine voraussichtlich im fünfstelligen Bereich beweget, ist dies als ausreichend zu betrachten. Für den Datentyp für Zeichen wird VARCHAR(45) festgelegt. Es lassen ich somit Zeichenketten bis zu einer Länge von 45 aufeinanderfolgenden Zeichen speichern.

2.4 Beziehungen innerhalb einer Datenbank

Eine wichtige Funktionalität von Datenbanken ist die Möglichkeit, Beziehungen zwischen den gespeicherten Daten zu definieren und zu verwalten. Es gibt verschiedene Datenbankmanagementsysteme die unterschiedlich implementiert sind, aber die Konzepte von Beziehungen und Fremdschlüssel sind überall ähnlich.

Die häufigsten Beziehungen innerhalb Datenbanken sind [vgl. 1, 2]:

- Eins-zu-eins-Beziehungen (1:1), bei denen jeder Datensatz in einer Tabelle genau einem Datensatz in einer anderen Tabelle zugeordnet ist.
- Eins-zu-viele-Beziehungen (1:n), bei denen ein Datensatz in einer Tabelle mehreren Datensätzen in einer anderen Tabelle zugeordnet sein kann.
- Viele-zu-viele-Beziehungen (n:m), bei denen ein Datensatz in einer Tabelle mehreren Datensätzen in einer anderen Tabelle zugeordnet sein kann und umgekehrt.

Mit diesen drei Beziehungen lassen sich alle Bedingungen direkt oder indirekt auflösen. Um Beziehungen in einer Datenbank zu verwalten, verwendet man in der Regel Fremdschlüssel (engl. "foreign key"). Ein Fremdschlüssel ist ein Attribut in einer Tabelle, das einen Bezug zu einem Primärschlüssel (engl. "primary key") in einer anderen Tabelle enthält. Auf diese Weise werden die Beziehungen zwischen den Tabellen dargestellt.

Der Primärschlüssel wird durch eine eindeutige ID (engl. "identifier") festgelegt. Diese ID wird in der Regel durch ein numerisches Attribut realisiert, das automatisch von der Datenbank erstellt wird, wenn ein neuer Datensatz hinzugefügt wird. Sie dient dazu, jeden Datensatz eindeutig zu identifizieren und sicherzustellen, dass keine duplizierten Datensätze in der Tabelle vorhanden sind.
Es gilt zu beachten, dass Primärschlüssel im Gegensatz zu einem Fremdschlüssel immer eindeutig sein müssen, dennoch dürfen beide niemals NULL bzw. leer sein. Wenn der Primär- oder Fremdschlüssel eines Datensatzes geändert wird, ändert sich auch die Beziehung zu den Datensätzen, daher sollte man eine entsprechende Änderung mit Bedacht umsetzen. Bestenfalls sollte es niemals zu solch einer Änderung kommen.

Eine Eins-zu-viele-Beziehung besteht zum Beispiel zwischen einem Modul und Baugruppen. Ein Modul kann mehrere Baugruppen enthalten, doch jede Baugruppe lässt sich nur genau einem Modul zuordnen. Es entstehen also zwei Tabellen daraus mit den Namen Modul und Baugruppe. Folglich stellt der Fremdschlüssel in der Modultabelle, der auf den Primärschlüssel der Baugruppentabelle verweist, diese Beziehung dar.

Eine Besonderheit stellt die viele-zu-viele-Beziehung (engl. "many-to-many relationship") dar. Diese ist eine Beziehung zwischen zwei Tabellen, bei der ein Datensatz in der einen Tabelle mehreren Datensätzen in der anderen Tabelle zugeordnet sein kann und umgekehrt. Bei den hier vorliegenden Daten kommt dies in der Beziehung zwischen Teilen und Lieferanten vor. Ein Lieferant kann viele Teile liefern und ein Teil kann mehrere Lieferanten haben, auch aufgrund unterschiedlicher Versionen und Stände. Aus diesem Grund kann die viele-zu-viele-Beziehung nicht direkt durch einen Fremdschlüssel in einer der beiden Tabellen aufgelöst werden. Stattdessen muss man eine dritte Tabelle erstellen, die die Beziehungen zwischen den Tabellen "Teil" und "Lieferant" verwaltet. Diese Tabelle nennt man "Junction-Tabelle" oder "Brückentabelle" und wird meistens nach einer Verbindung der beiden aufgelösten Tabellen benannt.

Die Junction-Tabelle enthält Fremdschlüssel, die auf die Primärschlüssel der Teil- und der Lieferant-Tabelle verweisen. Jeder Datensatz in der Junction-Tabelle repräsentiert eine Zuordnung zwischen einem Teil und einem Lieferanten. Beim Einfügen von Daten wird jeweils der Primärschlüssel der Teil- und Lieferant-Tabellen in die Junction-Tabelle eingefügt. So kann man die Beziehungen verwalten.

2.5 Datenbankstruktur

Wendet man nun die Beziehung auf die Zuordnung eines Teils, an ergibt sich der folgende Verlauf. Eine Komponente kann mehrere Bauteile beinhalten, ist selbst aber nur in einer Baugruppe enthalten. Die Beziehung zwischen Baugruppen, wie bereits in einem vorherigen Absatz beschrieben, verhält sich genauso. Völlig analog dazu ist die übergeordnete Maschinenebene, für welche es die Unterteilung zwischen Labor-, Glasboilermaschine und Gleichteil gibt. Hierbei handelt es sich also um Eins-zu-viele-Beziehungen, welche in Tabelle 2 mit den entsprechenden Spalten und zugehörigen Datentypen aufgelöst werden:

Tabelle 2: DB Tabelle Maschine, Modul, Baugruppe und Komponente

Maschine	
id_maschine (PK)	INT
beschreibung_maschine	VARCHAR(45)

Modul	
id_maschine (FK)	INT
id_modul (PK)	INT
beschreibung_modul	VARCHAR(45)

Baugruppe	
id_modul (FK)	INT
id_baugruppe (PK)	INT
beschreibung_baugruppe	VARCHAR(45)

Komponente	
id_baugruppe (FK)	INT
id_komponente (PK)	INT
beschreibung_komponente	VARCHAR(45)

Die Kürzel in den Tabellen FK und PK stehen jeweils für den Fremd- und Primärschlüssel. Die Tabelle Maschine besitzt keinen Fremdschlüssel, da es für die enthaltenen Elemente keine übergeordnete Zuweisung gibt und die Maschine somit die höchste Ebene darstellt.

Für die Zusatzkosten ergibt sich eine Eins-zu-eins-Beziehung, da diese nur einem bestimmten Teil angehören. Der Terminus Zusatzkosten bezieht sich hierbei nur auf Rüstkosten, Einrichtungskosten und dergleichen. Diese bestehen für ein bestimmtes Teil bei nur diesem Lieferanten zu den vereinbarten Bedingungen. Deshalb wird als Fremdschlüssel die eindeutige Teilenummer verwendet. Für die Kosten gilt zu beachten, dass es einmalige und wiederkehrende gibt. Diese Information wird zusätzlich benötigt und hierfür wird ein Erledigungsdatum vorgesehen. So ist die Information auch zu einem späteren Zeitpunkt noch nachvollziehbar, z. B. wann die Einrichtungskosten beim Lieferanten getätigt wurden. Zu den bauteilspezifischen Zusatzkosten sind ebenfalls noch allgemeine Zusatzkosten vorgesehen, welche bei jeder Maschine anfallen. Hierunter fallen die Lohnkosten. Da diese keine Schlüssel besitzen, kann die Teilenummer nicht als Primärschlüssel eingesetzt werden. Stattdessen kommt eine durchlaufende Nummer als Identifier zum Einsatz. Die Zusatzkostentabelle ist in Tabelle 3 dargestellt.

Tabelle 3: DB Tabelle Zusatzkosten

Zusatzkosten	
zusatzkosten_lfdnr (PK)	INT
betrag	DECIMAL(10,2)
beschreibung_zusatzkosten	VARCHAR(45)
teile_nr (FK)	VARCHAR(45)
einmalkosten	INT
erledigt_am	INT

Eigentlich handelt es sich bei der Zuordnung von Teil zu Lieferanten um eine viele-zu-viele-Beziehung, die mit einer Brückentabelle aufgelöst werden muss. Diese Brückentabelle würde nur die Teile-ID und Lieferanten-ID enthalten. Allerdings kommt in der Teiletabelle die Teile-ID mehrfach vor, da es z. B. mehrere gleiche Teile mit unterschiedlichen Versionsständen gibt, die prinzipiell das gleiche Teil darstellen und deshalb soll der entsprechende Identifier auch gleich bleiben. Die Brückentabelle würde also in diesem Fall nur die Zuordnung ermöglichen, welches Teil von welchem Lieferanten stammt. Dabei könnte aber ein Teil von mehreren Lieferanten stammen, wodurch die Differenzierung des richtigen Teils nicht möglich ist. Für die Maschine wird immer nur ein Teil von einem Lieferanten bezogen, auch wenn mehrere möglich wären. Diese Komplikation wird vielmehr durch eine 1:1:n als eine n:m Beziehung gelöst. Dazu wird die Teilenummer als Hilfe genommen, von der es mehrere unterschiedliche für ein Bauteil bzw. eine Teile-ID

geben kann. Diese enthält die spezifische Verknüpfung vom Bauteil zum Lieferanten und entzerrt die Daten somit. Die richtige Auswahl bzgl. der Teilenummer trifft in dem Fall der Kostenkalkulator selbst, da die richtige Version und ggf. situationsbedingt das günstigere Teil ausgewählt wird. Für die reine Datenbanksprache SQL bedeutet das, dass die Brückentabelle theoretisch in die Teiletabelle integriert werden könnte. Aufgrund der Arbeitsweise des Kostenkalkulators und der Art, wie die Daten in MATLAB eingegeben werden, macht dies jedoch keinen Sinn. Aufgelöst enthält die Teiletabelle prinzipiell nur die Zuordnung der Teile-ID zu den verschiedenen Teilenummern. Die anderen Daten der Tabelle sind vom Informationsgehalt in der Teilenummer bereits integriert und finden keine direkte Anwendung. Sie dienen also nur als erleichterte Informationsquelle für spätere Datenabfragen. Die Aufgabe der Brückentabelle wird somit verschoben.
Die Teiletabelle dient als Informationsträger für alle Daten, die zur Teilenummerngenerierung benötigt werden ausgenommen der Lieferanten-ID. Die Komponenten- und Teile-ID sind hierfür der Fremdschlüssel, während die Teilenummer als Primärschlüssel fungiert, da diese immer eindeutig ist. Die Lieferantentabelle hingegen enthält alle Lieferanten mit den zugehörigen Informationen wie Name, Lieferbedingungen, Anschrift und Kontaktdaten. Jeder Lieferant erhält eine eindeutige numerische ID. Für die benötigte Brückentabelle wird die Beziehung nun mit den beiden Schlüsseln Teilenummer und Lieferanten-ID aufgelöst. An Informationen enthält diese alle Daten bzgl. des Bauteils, darunter fallen Mindestbestellmenge, Material, Gewicht, Preise, Fertigungsmethoden und – kosten. Die drei Tabellen sind mit den entsprechenden Spaltennamen und zugehörigen Datentypen in Tabelle 5-6 dargestellt.

Tabelle 5: DB Tabelle Teil

Teil	
id_teil (FK)	INT
id_komponente (FK)	INT
beschreibung_teil	VARCHAR(45)
typ	VARCHAR(45)
status	VARCHAR(45)
version	INT
datum	INT
teile_nr (PK)	VARCHAR(45)

Tabelle 4: DB Tabelle Teil_Lieferant

Teil_Lieferant	
teile_nr (PK)	VARCHAR(45)
id_lieferant (FK)	INT
mindestbestellmenge	INT
bestellnummer	VARCHAR(45)
material	VARCHAR(45)
gewicht	INT
produktionsmethode	VARCHAR(45)
nachbehandlungsmethode	VARCHAR(45)
produktionskosten	DECIMAL(10,2)
nachbehandlungskosten	DECIMAL(10,2)
nebenkosten	DECIMAL(10,2)
stueckpreis_1	DECIMAL(10,2)
stueckpreis_10	DECIMAL(10,2)
stueckpreis_100	DECIMAL(10,2)

Tabelle 6: DB Tabelle Lieferant

Lieferant	
id_lieferant (PK)	INT
lieferant_name	VARCHAR(45)
mindestbestellsumme	DECIMAL(10,2)
lieferkosten	DECIMAL(10,2)
lieferkosten_frei_ab	DECIMAL(10,2)
rabatt_in_prozent	DECIMAL(10,2)
rabatt_ab	DECIMAL(10,2)
adresszeile_1	VARCHAR(45)
adresszeile_2	VARCHAR(45)
adresszeile_3	VARCHAR(45)
adresszeile_4	VARCHAR(45)
internetadresse	VARCHAR(45)
email	VARCHAR(45)
telefon	VARCHAR(45)
ansprechpartner	VARCHAR(45)

Verständlich und nachvollziehbar wird die Beziehung zwischen den vorher genannten Tabellen vermutlich erst mit der Stücklistentabelle. Diese ist für die Kalkulation zwingend für jede Maschine einzeln erforderlich. Sie enthält die Kerninformation, welche Bauteile in welcher Quantität in der jeweiligen Maschine verbaut sind. Deshalb enthält sie die Teile-ID und die jeweilige Stückzahl. Da die Preise pro Baugruppe ausgegeben werden sollen und ein Teil, z.B. ein Normteil, in mehreren Baugruppen vorkommen kann, ist zusätzlich die Baugruppen-ID enthalten. Da die beiden Identifier nicht eindeutig sind, muss zusätzlich eine eigene Listen-ID eingeführt werden. Diese ist zu vergleichen mit einer Positionsnummer in einer Strukturstückliste. Die Stückliste ist in Tabelle 7 dargestellt und muss in der Datenbank, wie bereits erwähnt, zweifach vorhanden sein, einmal für die Glasboiler- und einmal für die Labormaschine.

Tabelle 7: DB Tabelle Stückliste

Stückliste Glasboiler oder Stückliste Labormaschine	
id_list (PK)	INT
id_teil (FK)	INT
id_baugruppe (FK)	INT
anzahl	INT

Der Kostenkalkulator stellt die Beziehung zwischen den Teile-IDs in der Stückliste und den zugehörigen Teilen in der Teiletabelle her. Die Baugruppen-ID dient hierbei als Zuordnung für die jeweilige Baugruppe. Die gematchten Teile in den Teiletabellen werden nach den jeweiligen Auswahlkriterien verglichen und je nach Situation werden eine oder mehrere weiterverarbeitet. Mit der jeweils eindeutigen Teilenummer können die Bauteildaten und auch der Lieferant abgefragt werden. Der Prozess ist in Abbildung 1 in einem vereinfachten Flussdiagramm mit den Beziehungen und den jeweiligen Schlüsseln zwischen den Tabellen dargestellt.

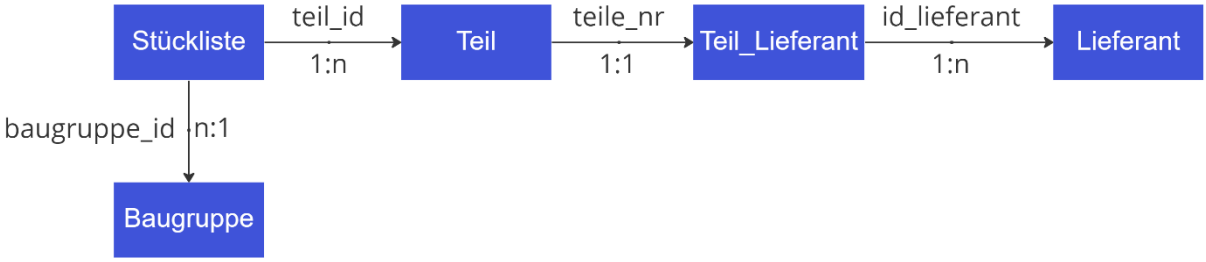

Abbildung 1: Zusammenhang Kostenkalkulator-Tabellen

Das durch die vorherigen Abschnitte beschriebene Datenbankmodell wurde mit einem gängigen Datenbankdesigntool erstellt und befindet sich im Anhang der Dokumentation [s. Anhang Nr. 1]. In dieser Grafik sind die Tabellen mit den zugehörigen Spalten und die zugehörigen Verbindungen zwischen den Tabellen noch einmal ausführlich dargestellt.

2.6 Schnittstelle zwischen Datenbank und MATLAB

Um eine Verbindung zwischen dem MySQL Server und MATLAB herzustellen bietet MATLAB unterschiedliche Optionen an [vgl. 5]. MATLAB besitzt eine eigene integrierte Datenbank Toolbox, über welche die Verbindung einmalig eingerichtet werden kann und dann jedes darauffolgende mal über die Verbindungsoption geöffnet werden kann. Mathworks empfiehlt, diese Option zu verwenden, um auf Datenbanken zuzugreifen, allerdings ist dies in der GUI wenig praktikabel. Jeder neue Nutzer müsste dies beim ersten Starten der APP einrichten und das Passwort für den Datenbankzugang müsste veröffentlicht werden. Zu dieser Option muss, wie bei anderen SQL-Anwendungen auch, die Verbindung über einen sogenannten Connectionstring hergestellt werden. Für die Verbindung stehen in MATLAB die zwei Funktionen "mysql" und "database" zur Verfügung. Das Ergebnis beider ist ein connection object und somit gleich. Allerdings unterscheidet sich die Art und Weise, wie diese Verbindung in MATLAB aufgebaut wird. Während bei der Funktion mysql ein eigener Verbindungstyp (MySQL native interface database connection) verwendet wird, kann bei der Funktion database sowohl eine Open Database Connectivity (ODBC) als auch eine Java Database Connectivity (JDBD) spezifiziert werden.

2.6.1 Unterschied zwischen JDBC und ODBC

Bei JDBC und ODBC handelt es sich um Schnittstellen, welche entwickelt wurden, um auf relationale Datenbanken zugreifen zu können. Sie haben jedoch einige wichtige Unterschiede in Bezug auf ihre Architektur, Plattformunabhängigkeit, ihren Treiber und ihre Leistung.

Architektur: JDBC ist eine Java-basierte Schnittstelle, die auf Java-Programme abzielt. Es stellt Java-Anwendungen ein Application Programming Interface (API) zur Verfügung, die es ermöglicht, direkt auf relationale Datenbanken zugreifen zu können. ODBC hingegen ist eine C-basierte Schnittstelle, die auf Anwendungen abzielt, die in C oder C++ geschrieben wurden. Es bietet eine API, die es Anwendungen ermöglicht, über eine C-Schnittstelle auf relationale Datenbanken zuzugreifen.

Plattformunabhängigkeit: JDBC ist plattformunabhängig und kann auf jeder Plattform verwendet werden, auf der eine Java Virtual Machine ausgeführt wird. Dadurch wird ermöglicht, dass Java-Anwendungen auf eine Vielzahl von Datenbanken und Plattformen zuzugreifen können. ODBC hingegen ist plattformabhängig und kann nur auf Windows-Plattformen verwendet werden. Es ermöglicht C/C++-Anwendungen, auf eine Vielzahl von Datenbanken und Plattformen zuzugreifen, aber nur unter Windows.

Treiber: JDBC-Treiber müssen für jede Datenbank separat erworben werden. Diese sind in der Regel direkt von den Datenbankherstellern zu beziehen. Dies bedeutet, dass für jede Datenbank, auf die man zugreifen möchte, ein entsprechender JDBC-Treiber benötigt wird. ODBC-Treiber sind in der Regel für mehrere Datenbanken in einem Paket erhältlich und sind in der Regel von Drittanbietern beziehbar. Dies bedeutet, dass ein ODBC-Treiber, der mehrere Datenbanken unterstützt, in der Regel genügt.

Leistung: Da JDBC-Treiber in der Regel direkt mit der Datenbank gekoppelt sind, kann die Leistung im Vergleich zu ODBC-Treibern höher sein. ODBC-Treiber funktionieren in der Regel als Middleware und die Leistung hängt somit mit dem Betriebssystem und der Hardware zusammen.

2.6.2 Verbindungsaufbau und Connectionstring

Im Folgenden wird jeweils ein Beispiel für jede Verbindungsmöglichkeit aufgezeigt, wie es im MATLAB Programmcode aussehen könnte. Die serverzugehörigen Werte müssen für eine erfolgreiche Verbindung spezifisch angepasst werden.

Beispiel für eine Datenbankverbindung mittels der Funktion mysql:

```
username =  'root'
password = '*******'
conn = mysql(username, password, 'Server', "localhost", ...
      DatabaseName',"geschmacksachekaffee",'PortNumber', 3306)
```

2.6.1

Beispiel für eine Datenbankverbindung mittels der Funktion database (OBDC):

```
databasename = "geschmacksachekaffee";
username = "username";
password = "******";
conn = database(databasename, username, password,'Vendor', 'MySQL',  ...
                'Server', 'dbtb01', 'PortNumber', 3306)
```

2.6.2

Beispiel für eine Datenbankverbindung mittels der Funktion database (JDBC):

```
databasename = "geschmacksachekaffee";
username = "username";
password = "******";
driver = 'com.mysql.cj.jdbc.Driver'
url = 'jdbc:z1MySQL://root:3306/geschmacksachekaffe'
conn = database(databasename, username, password, driver, url)
```

2.6.3

Zu beachten gilt, dass die Werte für Server und Port abweichen können und entsprechend angepasst werden müssen. Der Username und das zugehörige Passwort müssen natürlich auch entsprechend angepasst werden. Anstatt "localhost" ist für eine Datenbank, welche auf einem separaten Rechner läuft, die entsprechende IP-Adresse einzutragen. Bei einer JDBC werden die Werte username, port und databasename in die url eingefügt. Zusätzlich wird abweichend zu den anderen beiden ein spezieller Driver benötigt. Die URL und der Driver sind für verschiedene Versionen von MySQL abweichend. Das obige Beispiel bezieht sich auf eine MySQL Version von 8.0 und höher. Eine JDBC wird von MATLAB und MySQL unter Windows aufgrund der Einfachheit der ODBC nicht empfohlen. Soll die APP bzw. die Datenbankverbindung auch auf weiteren Betriebssystemen laufen, respektive IOS und Linux, muss die Verbindung über JDBC erfolgen.

Um zu testen, ob eine erfolgreiche Verbindung besteht, kann die MATLAB Funktion "isopen" benutzt werden. Diese gibt eine logische eins zurück, wenn die Datenbankverbindung besteht und eine entsprechende null, falls dies nicht der Fall sein sollte. Das folgende Beispiel zeigt die Funktionen und ihre benötigten Variablen:

```
i = isopen(conn)
```

2.6.4

Das Verbindungsobjekt "conn" ist in MATLAB in den Properties einzufügen. Dies ist nötig, um innerhalb der gesamten APP auf die Variable bzw. Datenbankverbindung zugreifen zu können.

2.7 Arbeiten mit Daten

Wie im Kapitel Grundlagen SQL bereits beschrieben, gibt es mehrere Befehle über welche die Daten entsprechend aufgerufen, gespeichert, geändert und gelöscht werden können. Diese Befehle wurden von der Funktionsweise 1:1 in MATLAB übernommen, sind jedoch teilweise abweichend bezeichnet. Jede dieser Funktionen wird im Folgenden einzeln beschrieben und ein entsprechendes Beispiel mit aufgeführt. Für alle Befehle ist es elementar, die entsprechende Datenbank anzugeben, auf welche zugegriffen werden soll. Dies geschieht durch die Variable conn, welche wie bereits beschrieben eine Datenbankverbindung darstellt.

2.7.1 Abrufen

Das Lesen und Anzeigen von Daten ist essenziel. Dabei sollte sich stets Gedanken gemacht werden, welche Daten aufgerufen werden sollen. Es besteht die Möglichkeit alle Daten einer entsprechenden Tabelle anzuzeigen oder nur Datensätze mit entsprechenden Kriterien. Um eine ganze Tabelle einer SQL-Datenbank in MATLAB aufzurufen, gibt es den Befehl "sqlread".

Die allgemeine Darstellung der Funktion lautet:

```
data = sqlread(conn, tablename)
```
2.7.1

Mit spezifisch eingesetzten Werten sieht die Abfrage wie folgt aus:

```
data = sqlread(conn, 'geschmacksachekaffee.lieferant')
```
2.7.2

Der ausgegebene Wert ist eine Duplizierung der Tabelle Lieferant aus der Datenbank mit dem Namen Geschmacksachekaffee mit allen Zelleninhalten und Spaltennamen.
Um Daten vorgefiltert abzurufen, steht die Funktion "select" zur Verfügung. Mit dieser lassen sich aus einer Kombination mit dem SQL-Befehl SELECT und der Eingrenzung WHERE oder auch LIKE entsprechende Abfragen durchführen. Die allgemeine Darstellung lautet:

```
data = select(conn, selectquery)
```
2.7.3

Für die SQL-Query können alle gängigen Bedingungen von SQL verwendet werden. Mit der WHERE-Bedingung können die gängigen Operatoren (gleich, größer, kleiner, größer gleich und kleiner gleich) realisiert werden. Zusätzlich zu diesen Optionen besteht auch die Möglichkeit, ein Ungleich bzw. entspricht nicht mit den Symbolen <> oder != zu verwirklichen. Ein konkretes Beispiel um nur die Zeile des Lieferanten abzurufen mit der ID 17, lautet wie folgt:

```
selectquery = 'SELECT * FROM geschmacksache.lieferant WHERE id_lieferant = 17'
```
2.7.4

Falls die Variable für die Abfrage keine numerische sondern ein String sein sollte, muss dieser in Singlequotes gesetzt werden. Dabei gilt zu beachten, dass der Singlequote an sich in MATLAB mehrere Funktionen besitzt. Unter anderem gehören dazu transponieren, Zeichen-Array erstellen und weitere. Deshalb muss dieser in einem String wiederholt eingegeben werden um als eigenes Element in einem String verwendet werden zu können. Eine Query mit einem String, um die entsprechenden Zeilen mit dem Lieferantenamen Xometry abzufragen, würde also wie folgt aussehen:

```
selectquery = 'SELECT * FROM geschmacksache.lieferant WHERE ...
               lieferant_name = 'Xometry''
```
2.7.5

Nachdem der SELECT-Befehl als Zeichenkette erstellt wurde, muss dieser über die oben genannte Funktion lediglich noch ausgeführt werden. Das erhaltene Element ist eine Tabelle, deren Zeile die übereinstimmenden Elemente sind. Mit der Option AND oder OR lassen sich weitere Optionen hinzufügen, die folglich alle oder nur eine zutreffen müssen. Das folgende Beispiel würde alle Zeilen abrufen, in denen die Lieferkosten größer null und der String "stein" im Lieferantennamen oder Ansprechpartner auftritt. Die Prozentzeichen stehen hierbei für Platzhalter.

```
selectquery = 'SELECT * FROM geschmacksache.lieferant WHERE lieferantname
      LIKE '%stein%' OR ansprechpartner LIKE '%stein%' AND lieferkosten > 0'
```
2.7.6

2.7.2 Speichern

Das Speichern von Daten erfolgt anstatt des SQL-INSERT-Befehls mit der eigenen MATLAB Funktion "sqlwrite". Die Daten werden hierbei als neue Zeile in einer bestehenden Tabelle hinzugefügt. Hierbei ist wichtig, dass die übergebenen Daten im Format Tabelle sind und in Anzahl sowie Bezeichnungen der Spalten exakt gleich sind wie in der Zieltabelle der Datenbank. Auch der Datentyp der einzelnen Werte muss mit den in MySQL definierten Datentypen übereinstimmen.

Die übergebenen Daten bzw. die Variable "data" muss von vornherein dem Datentyp Tabelle entsprechen. Entweder sind die Daten so direkt bei der Eingabe aufzunehmen oder können über die Funktionen table, array2table und cell2table mit den entsprechenden Parametern umgewandelt werden.

```
sqlwrite(conn, tablename, data)
```

2.7.7

2.7.3 Aktualisieren

Das Aktualisieren von Daten ist im Prozess ähnlich zum Speichern. Allerdings ist aus der Logik heraus eine Einschränkung zu treffen, welche Datensätze entsprechend überschrieben werden sollen. In MATLAB steht hierfür die Funktion "update" zur Verfügung.

```
update(conn, tablename, colnames, data, whereclause)
```

2.7.8

Der "whereclause" schränkt die zu überschreibenden Daten entsprechend ein. Die Variable "data" enthält die zu speichernden Daten. Wichtig hierbei ist, dass die Daten das vorgegebene Dateiformat der Datenbank in der entsprechenden Spalte haben müssen. Zu guter Letzt wird noch die Zuordnung der zu speichernden Daten mit den entsprechenden Spalten benötigt, in welchen diese angepasst werden sollen. Hierfür wird die Variable "colnames" verwendet, welche die genauen Bezeichnungen der Spalten wie in der Datenbank enthalten müssen. Die Daten und Spaltennamen müssen immer als Datentyp cell definiert und konform sein. Das heißt die Anzahl an Elementen von Daten muss genau gleich und in der entsprechenden Reihenfolge der Spaltennamen sein. Ein spezifisch angewandtes Beispiel, um für den Lieferanten mit der ID 17 die Lieferkosten auf 5 und Mindestbestellsumme auf 500 in der Tabelle Lieferant zu ändern, lautet:

```
colnames = {'mindestbestellsumme', 'lieferkosten'};
data = {500, 5};
whereclause = 'WHERE id_lieferant = 17';

update(conn, 'geschmacksachekaffee.lieferant', colnames, data, whereclause)
```

2.7.9

2.7.4 Löschen

Das Löschen von Datensätzen innerhalb einer Datenbank sollte in einem Produktivsystem keine Anwendung finden. Grund hierfür ist, dass die Löschung endgültig und ohne Möglichkeit auf Nachverfolgung geschieht. Das bedeutet, dass in den meisten Fällen nur schwer nachvollzogen werden kann, welche Daten von wem und warum gelöscht wurden. Ein gängiger Workaround ist ein sogenannter Löschvermerk. Datensätze, die mit einem Löschvermerk versehen werden, werden durch einen entsprechenden Schlüssel nicht mehr länger im Frontend betrachtet. Die Löschung von Daten ist in der Datenbank des Lieferantenmanagements und Kostenkalkulators nicht vorgesehen. Zur Vollständigkeit wird diese dennoch kurz aufgeführt und erläutert. Um Daten aus einer Datenbank zu löschen, wird in MATLAB der Befehl "execute" in Verbindung mit einer SQL-Bedingung verwendet. Die Funktion dient im Allgemeinen dazu, eine SQL-Query auszuführen. Es lassen sich damit alle SQL spezifischen Funktionen ausführen. Um zum Beispiel den Lieferanten mit der ID 17 aus der Tabelle Lieferant zu löschen, sieht die Funktion folgendermaßen aus:

```
sqlquery= 'DELETE * FROM geschmacksache.lieferant WHERE id_lieferant = 17'
execute(conn, sqlquery)
```

2.7.10

3 Teilenummerngenerator

Der Teilenummerngenerator ist ein Werkzeug, welches ermöglicht, automatisiert eindeutige Identifikationsnummern für Bauteile der Glasboiler- und Labormaschine zu erzeugen. Gegebenenfalls kann dieser auch um weitere Maschinen ergänzt und skaliert werden. Die Teilenummer ist eine eindeutige Referenz, die es ermöglicht ein Bauteil mit den zugehörigen Datensätzen zu verknüpfen. Dabei gilt es, für die Teilenummern das vorgesehene Teilenummernsystem einzuführen. Dieses gibt vor, wie die Nummer aufgebaut ist. Die Teilenummer ist ebenfalls fundamental für Verknüpfungen innerhalb der Datenbank. Der Teilenummerngenerator wird primär zur Erstellung von neuen Bauteilen in der Datenbank und die einhergehende Erzeugung der Teilenummer verwendet. Weitere Funktionen des Teilenummerngenerators sind das Anlegen von Baugruppen und Komponenten sowie das Bearbeiten und Anlegen von neueren Versionen bereits vorhandener Bauteile. Dabei ist das Bearbeiten von Bauteilen nur für Bauteile vorgesehen, die ohne den Lieferanten zu kennen angelegt wurden. So kann die Teilenummer in der Konstruktionsphase bereits vergeben werden und die Lieferantendaten zu einem späteren Zeitpunkt nachgetragen werden. Für die Teilenummerngenerierung wird ein eignes Array in der Anwendung verwendet, welches die neun verschiedenen Elemente der Teilenummern enthält und zusätzlich als zehntes Element die Maschinennummer in der ersten Position erhält. Die Teilenummernstruktur wurde bereits bei der Erstellung des Datenmodells genauer erläutert. Aus den ersten fünf Positionen des Arrays lässt sich die Teile-ID erzeugen. Diese Positionen werden durch die entsprechende Auswahl in der GUI aufgegriffen, während die restlichen durch Eingaben ergänzt werden.

3.1 GUI-Teilenummerngenerator

Im linken Teil der App, wie in Abbildung 2 dargestellt, befinden sich alle ausführbaren Funktionen der App. Jedes der Bedienungselemente startet eine andere Funktion und ändert den jeweiligen Ablauf der Anwendung. Im rechten Teil der App befinden sich mehrere Dropdown-Menüs, mit welchen durch die jeweiligen Anwendungen navigiert werden kann. Diese werden nur für die Anlage neuer Elemente benötigt. Im unteren Bereich befinden sich die jeweiligen Eingabefelder und mittig ein Ausgabefenster für die erzeugte Teilenummer. Die Eingabefelder sind logisch unterteilt in zwei Tabellen. Eine ist für die Grunddaten zuständig und ist folglich etwas kompakter und enthält nur die jeweiligen fundamentalen Informationen. Die größere der beiden ist die Zusatzdatentabelle, welche alle weiterführenden Informationen für Bauteile enthält. Neben den Eingabefeldern befindet sich außerdem der Speicherbefehl, durch welchen die eingegebenen Informationen in der Datenbank angelegt werden. Die Callbacks des Teilenummerngenerators sind ineinander verschachtelt. So werden die Dropdown-Menüs nur für die Anlage neuer Elemente benötigt und nicht alle Eingabefelder der unteren Tabellen dabei gebraucht. Im oberen rechten Eck der App befindet sich zusätzlich eine Statuslampe, welche den aktuellen Verbindungsstatus zur Datenbank darstellt. Bei einer vorhandenen Verbindung leuchtet diese grün. Sollte keine aktive Verbindung vorliegen leuchtet diese rot und es kann nicht mit der App gearbeitet werden, da alle Funktionen auf die Datenbank zugreifen und eine aktive Verbindung dafür zwingend erforderlich ist.

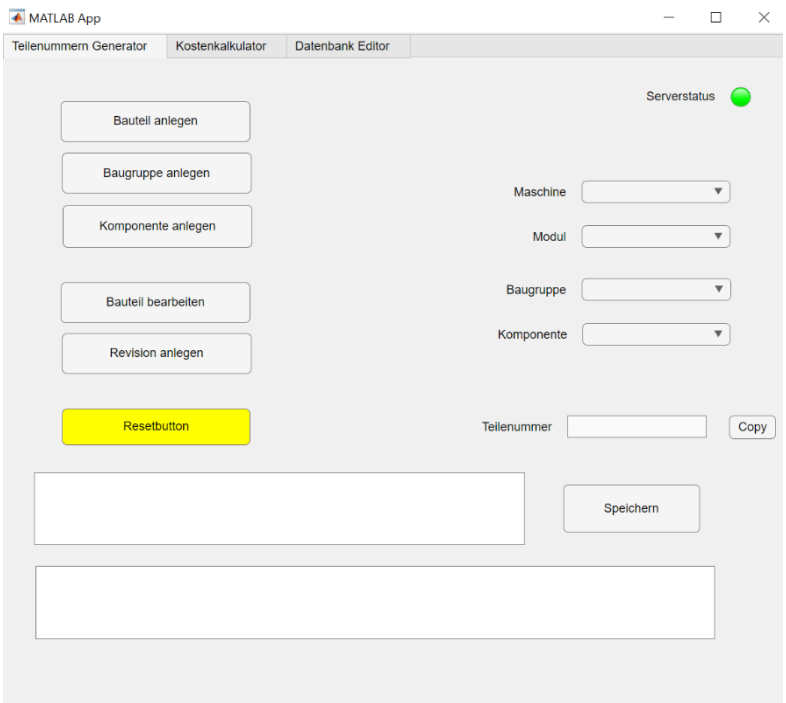

Abbildung 2: GUI-Teilenummerngenerator

3.2 Zusammenspiel der GUI-Elemente

Da in einer MATLAB-App mit sogenannten Callbacks gearbeitet wird, welche sich unter den GUI-Elementen befinden, ist das Zusammenspiel dieser elementar. Einige der eingesetzten Elemente haben mehre Aufgaben, welche von der ausgewählten Funktion abhängig sind. So werden zum Beispiel für die Funktionen Bauteil, Baugruppe und Komponente anlegen die Dropdown-Menüs mehrfach verwendet. Ebenso wird für das Anlegen eines neuen Bauteils die Zusatzkostentabelle verwendet, welche auch für eine Revision und Bauteil bearbeiten verwendet wird. In Abbildung 3 ist der Zusammenhang für die Funktionen zu den GUI-Elementen in einem Flussdiagramm sinnhaft dargestellt. Die verwendeten Farben der Linien suggestieren das jeweilige Zusammenspiel der Elemente. Dabei sind im Ausgangs- bzw. oberen Bereich der Grafik die Namen der jeweiligen Funktion gleich dem des jeweiligen Buttons in der GUI.

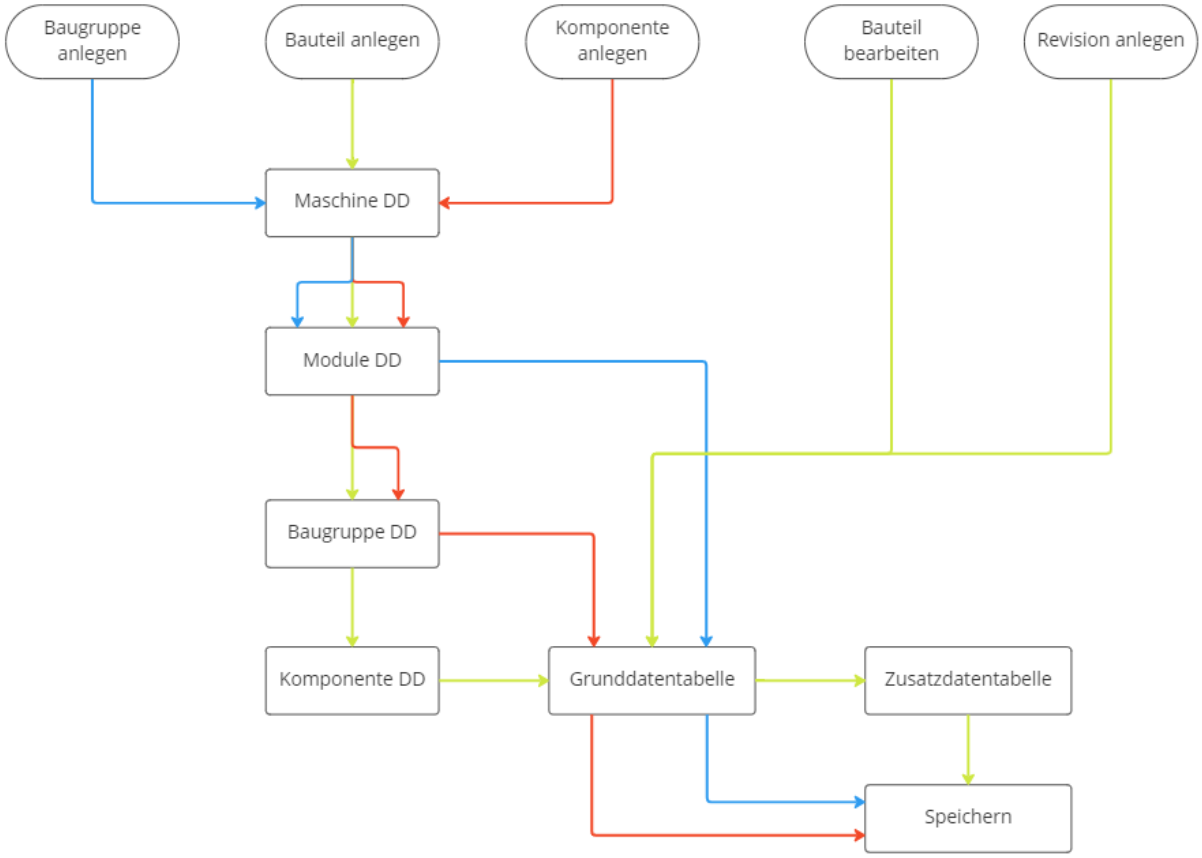

Abbildung 3: Zusammenspiel der GUI-Elemente des Teilenummerngenerators

Aus dem Diagramm erkennt man, dass die Funktionen für das Anlegen einer Baugruppe und Komponente nicht alle Dropdown-Menüs durchlaufen. Der Grund hierfür ist die Teilenummernstruktur. Eine Komponente kann einer Baugruppe zugeordnet werden und eine Baugruppe einer Maschine. Nur ein Bauteil durchläuft alle Menüpunkte, da diese von der Struktur einer Maschine, einem Modul, einer Baugruppe und ggf. einer Komponente zugeordnet werden kann. Außerdem wird für beide Funktionen die Zusatzdatentabelle nicht benötigt. Für das Bearbeiten und Anlegen neuer Versionen von Bauteilen werden die Dropdown-Menüs überhaupt nicht benötigt, da die Zuordnung des Bauteils schon festgelegt ist. Für beide werden jedoch beide Eingabetabellen benötigt und in der Grunddatentabelle wird vor dem Bearbeiten des jeweiligen Teils zunächst die zugehörige Teile-ID abgefragt, wodurch das jeweilige Bauteil mit den zugehörigen Daten abgefragt wird.
Der Aufbau des Programmcodes ist wie im obigen Abschnitt bereits beschrieben nicht linear und in sich komplex. Aufgrund der Wiederverwendung mehrere Elemente werden die einzelnen Elemente in Gruppierungen erklärt und Abzweigungspfade detailliert beschrieben. Dadurch wird eine unnötige Duplizierung des Programmcodes innerhalb der folgenden Seiten vermieden.

3.3 Schaltflächen (Buttons)

Durch die Schaltflächen im linken Bereich der MATLAB-GUI wird die jeweilige Funktion gestartet und der Appablauf grundlegend geändert. Dies erfolgt durch eine unterschiedliche Programmcodeübergabe. Aufgrund der erhöhten Komplexität und Unterscheidung wird das Speichern der Daten in einem eigenen Kapitel für alle Funktionen bearbeitet, obwohl dieser ebenfalls zu den Schalflächen zählt. Im Gegensatz dazu steht der Copybutton, welcher aufgrund der Simplizität nicht in diesem Kapitel erläutert wird, sondern unter sonstige Funktionen fällt.

3.3.1 Bauteil anlegen

Das Anlegen eines neuen Bauteils und die mit einhergehende Zuordnung der Teilenummer stellt die Hauptfunktion des Teilenummerngenerators dar. Beim Drücken des Buttons werden mehrere Befehle ausgeführt und die Funktionswahl zu eins gesetzt. Der erste Schritt ist das Importieren der verschiedenen Maschinen aus der Datenbank. Hierfür wird die Tabelle über einen Select-Befehl abgefragt und die Spalte, in welcher die Beschreibung der jeweiligen Maschine steht, als Auswahlmöglichkeit für das Maschinen Dropdown-Menü vorgesehen. Zu den Maschinenbezeichnungen aus der Datenbank wird zusätzlich die Option "Bitte Maschine wählen" als erste Auswahl vorgesehen und übergeben. Dieser Schritt ist nötig, da ein Dropdown-Element in MATLAB über eine Variable geändert Funktion läuft und der Wert somit erst geändert werden muss, um eine Auswirkung zu erzeugen. Anders ausgedrückt passiert die Ausführung des Programmcodes erst dann, wenn die Auswahl im Menü geändert wird. Der folgende Programmabschnitt ist für die Funktionen Baugruppe und Komponente anlegen identisch, bis auf die Funktionszuweisung in der ersten Zeile. Dort variiert die Zahl entsprechend der Auswahl.

```
app.funktion = 1;
selectquery = 'SELECT * FROM geschmacksachekaffee.maschine';
app.abfrage = select(app.conn, selectquery);

dropDownOptions = table2array(app.abfrage(:,2)).';
dropDownOptions = [{'Bitte Maschine Wahlen'}, dropDownOptions];
app.DropDown.Items = dropDownOptions;
```

3.3.1

Zu den Menüauswahlmöglichkeiten werden außerdem die Tabellen für die Eingabe vorbereitet. Dabei stehen zwei Tabellen zur Verfügung. Die Grunddatentabelle, welche alle Informationen bzgl. der Teilenummer außer den Lieferanten enthält und die Zusatztabelle. Diese enthält alle anderen Informationen, die für die Datenbank vorgesehen sind.
Für die Grunddatentabelle werden zunächst die Spaltennamen für die UI-Tabelle in der App festgelegt. Für die Spalten Typ und Status werden die Eingabemöglichkeiten eingegrenzt, indem die Werte als Dateiformat vorgegeben werden. Dadurch entsteht in der entsprechenden Zelle der MATLAB-Tabelle eine Dropdown-Liste mit den vorgegebenen Auswahlmöglichkeiten. Somit wird verhindert, dass der Benutzer falsche oder ungültige Werte eintragen kann. Als letztes wird die Bearbeitung für die verschiedenen Spalten eingegrenzt. Hierfür werden die Teile-ID und Version nicht zur Bearbeitung freigeschaltet und die Felder gesperrt.

```
app.Grunddaten.ColumnName = {'TeileID', 'Beschreibung', 'Typ', ...
                             'Status', 'Version', 'Datum'};

app.Grunddaten.ColumnFormat = {[] [] [] [] [] []};
app.Grunddaten.ColumnFormat(3) = {{'O - ohne Typzuordnung', 'N - ...
                                   Normteil', 'K - Konstruktionsteil'}};

app.Grunddaten.ColumnFormat(4)= {{'O - ohne Status', 'E - Entwicklung',...
                                  'F - Freigegeben', 'S - Serienteil'}};

app.Grunddaten.ColumnEditable = [false true true true false true];
```

3.3.2

Der Ablauf für die Zusatzkostentabelle ist vergleichbar mit der Grunddatentabelle. Für diese Tabelle sind alle Spalten bzw. Felder zur Bearbeitung freigegeben. Damit das Zahlenformat in der Ausgabe richtig dargestellt wird, muss das Spaltenformat auf "bank" eingestellt werden. Dabei handelt es sich um einen buchhalterischen Dateityp, der Zahlen mit zwei Nachkommastellen ausgibt. Da die Kosten in Euro auf zwei

Nachkommastellen eingegeben werden sollen, ist dies hier zwingend nötig. Falls der Dateityp nicht explizit definiert wird, arbeitet MATLAB automatisch mit der Grundeinstellung von vier Nachkommastellen. Ein weiterer Vorteil des Dateityps ist, dass dieser kein Komma als Dezimaltrennzeichen zulässt. Das ist ebenfalls wichtig, da als Dezimaltrennzeichen von MATLAB und MySQL nur ein Punkt und kein Komma akzeptiert wird. Sollte dennoch eine kommagetrennte Dezimalzahl eingegeben werden, wird der Wert so automatisch zu "NaN" und als leeres Feld bzw. Null behandelt. Für den Lieferanten soll die Auswahlmöglichkeit wieder eingegrenzt werden. Dafür werden aus der Datenbank alle Lieferanten aus der zugehörigen Lieferantentabelle abgefragt. Anschließend wird die Lieferant-ID mit der Lieferantbezeichnung mit einem Bindestrich zusammengefügt und als Spaltenformat übergeben. Für das Zusammenfügen wird die Funktion "strcat" verwendet. Mit dieser lassen sich alle Zellen aus einem Array mit einem definierten Abgrenzer (engl. Delimiter) verbinden. Anschließend muss das Array noch transponiert werden, da dieses in einer horizontalen Dimension ist, aber in einer vertikalen Ausdehnung benötigt wird. In MATLAB kann dies durch ein einfaches Transponierenzeichen umgesetzt werden, wie in der letzten Programmcodezeile des folgenden Abschnitts.

```
app.Zusatzdaten.ColumnName = {'Lieferant', 'Mindestbestellmenge', ...
 'Bestellnummer', 'Material', 'Gewicht [g]', 'Produktionsmethode', ...
 'Nachbehandlungsmethode','Produktionskosten','Nachbehandlungskosten', ...
 'Nebenkosten', 'Stückpreis (1)', 'Stückpreis (10)', 'Stückpreis (100)'};

app.Zusatzdaten.ColumnEditable = [true true true true true true true ...
                                  true true true true true true];

app.Zusatzdaten.ColumnFormat = {[] [] [] [] [] [] [] ...
                                'bank' 'bank' 'bank' 'bank' 'bank' 'bank'};

lieferanten = sqlread(app.conn, 'geschmacksachekaffee.lieferant');
auswahl = strcat(num2str(lieferanten{:,1}), {' - '}, lieferanten{:,2});
app.Zusatzdaten.ColumnFormat(1) = {auswahl'};
```

3.3.3

3.3.2 Baugruppe anlegen

Beim Drücken dieses Buttons wird zunächst die Auswahlfunktion auf zwei gesetzt und die Optionen für das Maschinen Dropdown-Menü wie bereits beschrieben übergeben. Zusätzlich werden die Spalten der Grunddatentabelle benannt und nur die Beschreibung zur Bearbeitung freigegeben. Zusatzdaten werden hierfür nicht benötigt.

```
app.funktion = 2;

%Auswahlmöglichkeiten Dropdown-Menü Maschine (siehe Programcode 3.3.1)

app.Grunddaten.ColumnName = {'BaugruppenID', 'ModulID', 'Beschreibung'};
app.Grunddaten.ColumnEditable = [false false true];
```

3.3.4

3.3.3 Komponente anlegen

Der Programmcode für diese Funktion ist wieder identisch zu dem der Baugruppe, bis auf die Spaltenbenennung.

```
app.funktion = 3;

%Auswahlmöglichkeiten Dropdown-Menü Maschine (siehe Programcode 3.3.1)

app.Grunddaten.ColumnName= {'KomponentenID','BaugruppenID',Beschreibung'};
app.Grunddaten.ColumnEditable = [false false true];
```

3.3.5

3.3.4 Revision anlegen

Für das Anlegen einer neuen Bauteilversion werden die Abfragen durch die Dropdown-Menüs nicht benötigt. Stattdessen bedarf es der Teile-ID eines bereits angelegten Bauteils, für das eine neue Version angelegt werden soll. Dazu wird die erste Zelle der Grunddatentabelle zur Bearbeitung freigegeben und mit einer Eingabeaufforderung versehen. Da die restlichen Zellen der Tabelle leer sind, wird nur diese Zelle für die gesamte Tabelle ausgegeben. Zur Unterscheidung zu den anderen Funktionen wird die App-Funktion auf vier und die App-Eingabe auf eins gesetzt. Die Zusatzdatentabelle wird analog erstellt wie bei einer Bauteilanlage.

```
app.funktion = 4;
app.eingabe = 1;

app.Grunddaten.ColumnName = {'TeileID', 'Beschreibung', 'Typ', ...
                             'Status', 'Version', 'Datum'};
app.Grunddaten.ColumnFormat = {[] [] [] [] [] []};
app.Grunddaten.ColumnFormat(3) = {{'O - ohne Typzuordnung', 'N – ...
                                   Normteil', 'K - Konstruktionsteil'}};
app.Grunddaten.ColumnFormat(4) = {{'O - ohne Status', 'E -
Entwicklung',...
                                  'F - Freigegeben', 'S - Serienteil'}};
app.Grunddaten.ColumnEditable = [true true true true true true];


app.Grunddaten.Data{1,1} = 'Bitte TeileID eingeben';

app.Zusatzdaten.ColumnName = {'Lieferant', 'Mindestbestellmenge', ...
 'Bestellnummer' 'Material', 'Gewicht [g]', 'Produktionsmethode', ...
 'Nachbehandlungsmethode', 'Produktionskosten',
'Nachbehandlungskosten',...
 'Nebenkosten', 'Stückpreis (1)', 'Stückpreis (10)', 'Stückpreis (100)'};

app.Zusatzdaten.ColumnEditable = [true true true true true true ...
                                  true true true true true true true];
app.Zusatzdaten.ColumnFormat = {[] [] [] [] [] [] []...
                                'bank' 'bank' 'bank' 'bank' 'bank'
'bank'};

lieferanten = sqlread(app.conn, 'geschmacksachekaffee.lieferant');
auswahl = strcat(num2str(lieferanten{:,1}), {' - '}, lieferanten{:,2});
app.Zusatzdaten.ColumnFormat(1) = {auswahl'};
```

3.3.7

3.3.5 Bauteil bearbeiten

Der Programmcode für diesen Button ist nahezu identisch zu Revision anlegen, der einzige Unterschied besteht in der Funktionswahl. Diese Variable wird hierfür auf fünf gesetzt. Da der sonstige Programmcode inklusive der verwendeten Variablen eine hundertprozentige Übereinstimmung hat, wird dieser nicht nochmal explizit abgedruckt. Die Gleichheit beruht darauf, dass beide Funktionen eine starke Ähnlichkeit aufweisen. Der Unterschied liegt im Speichern der Daten. Für eine Revision werden diese neu angelegt, während beim Bearbeiten vorhandene Daten ersetzt werden.

```
app.funktion = 5;
%Identisch zu Revision anlegen (siehe Programmcode 3.3.7)
```

3.3.8

3.4 Dropdown-Menüs

In der MATLAB-GUI sind vier Dropdown-Menüs vorgesehen, welche zur Auswahl der Maschine, des Moduls, der Baugruppe und der Komponente dienen. Diese werden nur für das Anlegen neuer Bauteile, Baugruppen oder Komponenten benötigt, um die Zuordnung zu treffen. Dabei wird nicht jedes Dropdown für jede Funktion benötigt und es entstehen Abzweigungen in den Callbacks.

3.4.1 Maschine Dropdown-Menü

Damit die ungültige Dropdown Option "Bitte Maschine wählen" nicht selektiert werden kann, wird diese in einer Bedingung abgefragt und somit ausgeschlossen. Hierfür wird die Funktion "strcmpi" verwendet. Damit wird bewirkt, dass der Programmcode nur abläuft, wenn eine valide Auswahl getroffen wurde. Danach werden für die gewählte Maschine die vorhandenen Module aus der Datenbank abgefragt. Die Bezeichnungen der Module werden anschließend an das Modul Dropdown-Men mit der Erweiterung der Option "Bitte Modul Wählen" weitergegeben. Zusätzlich wird in der Funktion die entsprechende Maschinen-ID als erstes Element in das Teilenummern-Array eingesetzt.

```
if strcmpi(app.DropDown.Value,'Bitte Maschine Wahlen') == 0
   sqlvalue = find(strcmp(app.abfrage{:, 2}, value));
   app.teilenummer{1,1} = sqlvalue;

   selectquery = ['SELECT * FROM geschmacksachekaffee.modul WHERE ...
   id_maschine = ', num2str(sqlvalue)];
   app.modulabfrage = select(app.conn, selectquery);

   dropDownOptions = table2array(app.modulabfrage(:,3)).';
   dropDownOptions = [{'Bitte Modul Wählen'}, dropDownOptions];
   app.ModulDropDown.Items = dropDownOptions;
end
```

3.4.1

3.4.2 Modul Dropdown-Menü

Die Auswahlmöglichkeiten für die Module werden mit dem vorherigen Programmabschnitt übergeben. Hierbei wird ebenfalls die vorausgefüllte Option in einer Bedingung abgefragt und somit gesperrt. Für dieses Menü gibt es allerdings die erste große Unterscheidung. Sollte die Funktion Baugruppe anlegen gewählt sein, wird nicht die Auswahl für das nächste Dropdown-Menü erzeugt, sondern die Eingabe der Grunddaten für die anzulegende Baugruppe vorbereitet. Im folgenden Programmcode wird die ausgewählte Variable im Dropdown-Menü mit der Modulabfrage verglichen. Dadurch können über den zugehörigen Identifier die untergeordneten Baugruppen ermittelt werden. Außerdem wird die Modul-ID um die Maschinen-ID gekürzt, um die reine Modulnummer zu erhalten. Diese wird im Teilenummern-Array abgelegt.

```
if strcmpi(app.ModulDropDown.Value,'Bitte Modul Wahlen') == 0
 sqlvalue2= find(strcmp(app.modulabfrage{:, 3}, app.ModulDropDown.Value));
 sqlvalue2= table2array(app.modulabfrage(sqlvalue2, 1));
 deletlength = strlength(string(app.teilenummer {1,1}));
 app.teilenummer{1,2} = eraseBetween(num2str(sqlvalue2), 1, deletlength);

 selectquery = ['SELECT * FROM geschmacksachekaffee.baugruppe WHERE ...
                  id_modul = ', num2str(sqlvalue2)];
 app.baugruppenabfrage = select(app.conn, selectquery);

 if app.funktion == 2
     %Unterscheidung für Anlage einer Baugruppe (siehe Programmcode 3.4.3)
 else
     dropDownOptions2 = table2array(app.baugruppenabfrage(:,3)).';
     dropDownOptions2 = [{'Bitte Baugruppe Wahlen'}, dropDownOptions2];
     app.DropDown_3.Items = dropDownOptions2;
 end
end
```

3.4.2

Der folgende Programmcode behandelt die Unterscheidung für das Anlegen einer Baugruppe und ist im vorherigen Abschnitt aus der bedingten Anweisung (engl. if statment) ausgeschnitten. Zunächst wird durch eine weitere bedingte Anweisung die höchste vorhandene Baugruppen-ID ermittelt. Sollte es noch keine Baugruppe in dem ausgewählten Modul geben, ist diese null. Daraufhin wird diese aufgetrennt, um nur die Ziffer für die Baugruppe um eins erhöhen zu können. Grund hierfür ist der Umschlag von neun auf zehn. Als Beispiel wäre die Baugruppen-ID 129, was übersetzt erste Maschine, zweites Modul und neunte Baugruppe bedeutet, die nächste Baugruppen-ID sonst die 130. Die Bedeutung geht hierbei verloren, da die nächste gültige ID die 1210 sein muss. Danach werden die Modul-ID und die neu erzeugte Baugruppen-ID in die Grunddatentabelle übergeben. Die Beschreibung wird zur Bearbeitung leer gelassen.

```
if isempty(app.baugruppenabfrage) == 1
   maxbaugruppen_id = 0;
else
   maxbaugruppen_id = max(app.baugruppenabfrage.id_baugruppe);
end
baugruppen_id_next = num2str(str2num(erase(num2str(maxbaugruppen_id), ...
                                           num2str(sqlvalue2)))+1);
baugruppen_id = str2num(strcat(num2str(sqlvalue2), baugruppen_id_next));
app.Grunddaten.Data{1,1} = baugruppen_id;
app.Grunddaten.Data{1,2} = sqlvalue2;
app.Grunddaten.Data{1,3} = ' ';
```

3.4.3

3.4.3 Baugruppe Dropdown-Menü

Für das Baugruppen Dropdown-Menü gilt derselbe Ansatz wie für das Modul Dropdown. Über die ausgewählte Variable wird die Komponenten-ID ermittelt, mit welcher die zugehörigen Komponenten abgefragt und an das nächste Dropdown-Menü übergeben werden könne, insofern nicht die Option Komponente anlegen gewählt wurde. Wird eine Komponente angelegt, wird hierfür wieder durch eine bedingte Anweisung unterschieden. Die nächste Ziffer der Teilenummer wird durch das Löschen der bereits ermittelten Ziffern für Maschine und Modul von der Baugruppen-ID erstellt und abermals in das Teilenummern-Array übertragen.

```
if strcmpi(app.BaugruppeDropDown.Value,'Bitte Baugruppe Wahlen') == 0

  sqlvalue3 = find(strcmp(app.baugruppenabfrage{:, 3}, ...
                  app.BaugruppeDropDown.Value));
  sqlvalue3 = table2array(app.baugruppenabfrage(sqlvalue3, 1));
  app.teileID1 = sqlvalue3;

  hilf = string(app.teilenummer(1,1)) + string(app.teilenummer(1,2));
  deletlength = strlength(hilf);
  app.teilenummer{1,3} = eraseBetween(num2str(sqlvalue3), 1, deletlength);

  selectquery = ['SELECT * FROM geschmacksachekaffee.komponente WHERE ...
  id_baugruppe = ', num2str(sqlvalue3)];
  app.komponentenabfrage = select(app.conn, selectquery);

  if app.funktion == 3
     %Unterscheidung für Anlage einer Komponente (s. Programmcode 3.4.5)
  else
     dropDownOptions3 = table2array(app.komponentenabfrage(:,3)).';
     dropDownOptions3 = [{'Bitte Komponente Wählen'}, ...
                         'Keine Komponente', dropDownOptions3];
     app.DropDown_4.Items = dropDownOptions3;
  end
end
```

3.4.4

Der Unterscheidungsfall für das Anlegen einer Komponente ist bis auf die geänderten Variablen identisch zum Anlegen einer Baugruppe, wie in Programmabschnitt 3.4.3 beschrieben. Auf eine ausschweifende Erklärung wird aus diesem Grund verzichtet, die Programmcodes aber der Vollständigkeit halber abgebildet.

```
if isempty(app.komponentenabfrage) == 1
   maxkomponenten_id = 0;
else
   maxkomponenten_id = max(app.komponentenabfrage.id_komponente);
end
komponenten_id_next = num2str(str2num(erase(num2str ...
                              (maxkomponenten_id), num2str(sqlvalue3)))+1);

komponenten_id = str2num(strcat(num2str(sqlvalue3), komponenten_id_next));
app.Grunddaten.Data{1,1} = komponenten_id;
app.Grunddaten.Data{1,2} = sqlvalue3;
app.Grunddaten.Data{1,3} = ' ';
```

3.4.5

3.4.4 Komponente Dropdown-Menü

Für dieses Dropdown-Menü gibt es keine Unterscheidung mehr, da nur die Bauteilanlage dieses durchläuft. Allerdings werden hinter diesen Callback die Informationen für die Grunddatentabelle aufbereitet und an diese übergeben. Der Programmcode ist wieder in einer bedingten Anweisung eingebettet, um die nicht gültige Auswahl herauszufiltern. In einer weiteren Bedingung wird die Komponenten-ID mit einer Nullappendex erzeugt, falls die Variable den Wert "Keine Komponente" besitzt. Ansonsten wird die Komponenten-ID über die Löschung der bereits ermittelten Ziffern ermittelt, sodass nur die Ziffer für die zugehörige Komponente bleibt.

```
if strcmpi(app.KomponenteDropDown.Value,'Bitte Komponente Wählen') == 0

   if strcmpi(app.KomponenteDropDown.Value,'Keine Komponente')
      app.teileID = app.teileID1*10;
      app.teilenummer{1,4} = 0;
   else
      sqlvalue4 = find(strcmp(app.komponentenabfrage{:, 3}, ...
                  pp.KomponenteDropDown.Value));
      sqlvalue4 = table2array(app.komponentenabfrage(sqlvalue4, 1));
      app.teileID = sqlvalue4;
      hilf= string(app.teilenummer(1,1)) + string(app.teilenummer(1,2))...
                   + string(app.teilenummer(1,3));
      deletlength = strlength(hilf);
      app.teilenummer{1,4} = eraseBetween(num2str(sqlvalue4), ...
                                          1, deletlength);
   end
   % Daten aufbereiten und übergeben (s. Programmcode 3.47, 3.4.8 und 3.4.9)
end
```

3.4.6

Damit die Teile-ID bestimmt werden kann, muss zuerst ein Abgleich stattfinden, ob bereits Bauteile in der entsprechenden Komponente existieren. Dafür werden die zugehörigen Bauteile mit der Komponenten-ID über folgenden Programmcode abgefragt. Danach wird über eine bedingte Anweisung die höchste Teile-ID ermittelt. Falls die Abfrage leer ist, sprich keine Bauteile der Komponente angehören, ist diese Null.

```
hilf2 = string(app.teilenummer(1,1)) + string(app.teilenummer(1,2) + ...
string(app.teilenummer(1,3)) + string(app.teilenummer(1,4));

selectquery = ['SELECT * FROM geschmacksachekaffee.teil WHERE ...
               id_komponente = ', num2str(app.teileID)];

teileabfrage = select(app.conn, selectquery);

if isempty(teileabfrage) == 1
   maxteile_id = 0;
else
   maxteile_id = max(teileabfrage.id_teil);
end
```

3.4.7

Durch das Entfernen der Komponenten-ID von der Teile-ID, kann die Ziffer zählend um eins erhöht werden. Dies geschieht im folgenden Programmabschnitt, in welchem diese auch in das Teilenummern-Array übergeben wird und für die Weiterverarbeitung aufbereitet wird.

```
teil_id = num2str(str2num(erase(num2str(maxteile_id), hilf2))+1);
app.teilenummer{1,5} = teil_id;
app.teileID = str2num(strcat(hilf2, teil_id));
app.komponente_id = hilf2;
```

3.4.8

Im Anschluss wird für die Grunddatentabelle die Teile-ID, das aktuelle Datum und die Version mit eins übergeben. Zum Ermitteln steht in MATLAB die Funktion "datetime" zur Verfügung, welche auf das geforderte Format angepasst wird. Für die Zusatzdaten wird ein leeres Array übergeben und somit die Bearbeitung freigeschaltet. Außerdem wird in die erste Zelle eine Eingabeaufforderung übergeben.

```
date = yyyymmdd(datetime("today"));

app.Grunddaten.Data{1,1} = app.teileID;
app.Grunddaten.Data{1,5} = 1;
app.Grunddaten.Data{1,6} = date;

app.Zusatzdaten.Data = app.zusatzdaten;
app.Zusatzdaten.Data{1,1} = 'Bitte Lieferant Auswählen';
```

3.4.9

3.5 Eingabetabellen

In der MATLAB-GUI sind zwei Eingabetabellen vorgesehen. Da nicht für jede Funktion der App beide gebraucht werden, werden jeweils nur die benötigte(n) über die entsprechenden Knöpfe aufgerufen. Bezüglich der Datenstruktur enthält die Grunddatentabelle nur die Daten für die Teilenummer und die Bauteilbeschreibung. Zusätzlich wird diese als Eingabefeld für die Teil-ID bei Bauteilrevisionen oder –bearbeitung verwendet. Die Zusatzdatentabelle enthält alle zugehörigen Bauteilinformationen inklusive des Lieferanten. Zu den Eingabetabellen existiert noch das Ausgabefeld für die Teilenummer. Dort wird diese entsprechend ausgegeben, sobald alle benötigten Informationen in den Eingabetabellen vorhanden sind.

3.5.1 Grunddatentabelle

Für die Grunddatentabelle werden die Daten aus der UI-Table bei jeder Änderung in eine eigene Tabelle eingefügt. Außerdem ist die Funktion der Tabelle in einen Switch Case unterteilt, wobei ein Fall für das Anlegen eines Bauteils zuständig ist und der andere für Revisionen und Bearbeiten von Bauteilen. Ein sonstiger Fall wird hierfür nicht benötigt, da für die anderen Funktionen nur die erste Programmcodezeile des folgenden Ausschnitts benötigt wird.

```
app.bauteilgrunddaten = app.Grunddaten.Data;

switch app.funktion
       case 1
       % Bauteilanlegen (siehe Programmcode 3.5.2 und 3.5.3)
       case {4,5}
       % Revision und Bauteil bearbeiten (siehe Programmcode 3.5.4 bis 3.5.8)
end
```

3.5.1

Für die Dateneintragung bei einer Bauteilanlage werden die entsprechenden Felder an das Teilenummern-Array übertragen. Dies sind die Version, Datum, Status und Typ, allerdings werden für den Status und den Typ nur die bezeichnenden Buchstaben übernommen und die Beschreibung entfernt. Hierzu wird die Funktion "strtok", mit der sich ein String nach einem beliebigen Zeichen auftrennen lässt verwendet. In diesem Fall wird der Bindestrich benutzt.

```
app.teilenummer{1,6} = app.bauteilgrunddaten{1,5};
app.teilenummer{1,7} = strtok(app.bauteilgrunddaten{1,3}, {' - '});
app.teilenummer{1,9} = strtok(app.bauteilgrunddaten{1,4}, {' - '});
app.teilenummer{1,10} = app.bauteilgrunddaten{1,6};
```

3.5.2

Im Anschluss wird durch eine bedingte Anweisung mit der Funktion "isempty" gegengeprüft, ob bereits alle benötigten Daten für die Teilenummer eingegeben wurden. Zum Zeitpunkt des untenstehenden Programmcodes sind das die Felder Lieferant, Typ und Status. Falls dem so ist wird die Teilenummer aus dem Array mit der Funktion "strjoin" mit jeweils einem Bindestrich zusammengefügt und an das dafür vorgesehene Ausgabefeld übergeben.

```
if isempty(app.bauteilgrunddaten{1,3}) == 1 || ...
   isempty(app.bauteilgrunddaten{1,4}) == 1||isempty(app.teilenummer{1,8})

   app.TeilenummerEditField.Value = app.TeilenummerEditField.Value;

else

   hilf_teilenummer = app.teilenummer;
   hilf_teilenummer(:,1) = [];
   app.TeilenummerEditField.Value = strjoin(string(hilf_teilenummer),'-');

end
```

3.5.3

Für das Bearbeiten von Bauteilen und Anlegen von neuen Versionen werden erst durch die Eingabe der Teile-ID die entsprechenden Daten aus der Datenbank abgerufen und in die Eingabetabellen geladen. Um dies zu ermöglichen, ist der Programmcode in zwei Anweisungen eingebunden. Die erste ist dafür zuständig, dass das Abfragen der Daten nur einmal abläuft. Die zweite sorgt dafür, dass sich der gedrückte Knopf rot einfärbt und eine Ausgabemeldung kommt, falls kein Datensatz zur eingegebenen Nummer existiert. Dazu wird der Datensatz über die ID abgefragt und anhand der höchsten Revision ausgewählt. Falls dieser keine Zeile enthalten sollte, wird durch eine weitere bedingte Anweisung, je nach gewählter Funktion, der zugehörige Button rot eingefärbt und die Eingabevariable zurückgesetzt. Dadurch kann die Bedingung so lange ablaufen, bis eine gültige Teile-ID eingegeben wurde.

```
if app.eingabe == 1

   app.eingabe = 2;
   sqlvalue_rev = app.Grunddaten.Data{1,1};
   selectquery = ['SELECT * FROM geschmacksachekaffee.teil WHERE  ...
                  id_teil = ', sqlvalue_rev];
   app.revisionsabfrage = select(app.conn, selectquery)
   topversion = max(app.revisionsabfrage.version);

   work_data = table2cell(app.revisionsabfrage(topversion,:));

   if height(work_data) ~= 0
       %Datenverarbeitung (siehe Programmcode 3.5.5 bis 3.5.8)
   else
       if app.funktion == 4
          app.RevisionanlegenButton.BackgroundColor = 'r';
       else
          app.BauteilBearbeiten.BackgroundColor = 'r';
       end
       app.Grunddaten.Data{1,1} = 'Teile ID nicht gefunden';
       app.eingabe = 1;
   end

else
   %Dateneingabe (siehe Programmcode 3.5.2)
end
```

3.5.4

Zur Datenverarbeitung wird zunächst die Teilenummer aus der Datenbank aufgetrennt und in das Teilenummern-Array eingefügt. Die alte Teilenummer wird hierbei zwischengespeichert, da diese später zum Aktualisieren der Bauteildaten benötigt wird. Ebenfalls werden die Komponenten-ID und Teile-ID zwischengespeichert. Aus der Datenabfrage werden final noch die Grunddaten von überflüssigen Spalten bereinigt, bevor diese an die Tabelle übertragen werden.

```
app.teilenummer_alt = work_data{1,8};
app.teilenummer = strsplit(work_data{1,8}, '-');
app.teileID = work_data (1,1);
app.komponente_id = work_data(1,2);
work_data(:, 2) = [];
work_data(:, 7) = [];


app.bauteilgrunddaten = work_data;
app.Grunddaten.Data = work_data;
```

3.5.5

Unmittelbar danach werden die Grunddaten noch geringfügig angepasst, falls die gewählte Funktion das Anlegen einer Revision ist. Dies wird durch eine bedingte Anweisung umgesetzt. Hierbei wird das Datum durch das aktuelle ersetzt und die Version automatisch um eins erhöht. Die Teilenummer wird ebenfalls neu erzeugt, da sich die Grundlage dafür verändert hat. In beiden Fällen muss die Bearbeitung der Zellen der Grunddatentabelle noch angepasst werden, da ab diesem Moment die erste Zelle nicht mehr zur Bearbeitung freigegeben sein darf. In dieser Zeile wird die unveränderliche Teile-ID abgebildet.

```
if app.funktion == 4
   app.Grunddaten.Data{1,5} = app.Grunddaten.Data{1,5} +1;
   app.Grunddaten.Data{1,6} = yyyymmdd(datetime("today"));
   app.teilenummer{1,6} = app.Grunddaten.Data{1,5};

   hilf_teilenummer = app.teilenummer;
   hilf_teilenummer(:,1) = [];

   app.TeilenummerEditField.Value = strjoin(string(hilf_teilenummer),'-');
   app.Grunddaten.ColumnEditable = [false true true true true true];

else
   app.Grunddaten.ColumnEditable = [false true true true false true];
end
```

3.5.6

Die Grunddaten sind hiermit vollständig in die dafür vorgesehene Eingabetabelle übertragen. Der folgende Programmabschnitt ist für die Zusatzdaten zuständig. Diese müssen über die für die Grunddaten ermittelte Teilenummer abgefragt werden.

```
sqlvalue = strjoin(['''',app.revisionsabfrage{topversion,8} ,'''']);
sqlvalue = sqlvalue(~isspace(sqlvalue));
selectquery = ['SELECT * FROM geschmacksachekaffee.teil_lieferant WHERE...
                teile_nr = ', sqlvalue];
sql_abfrage = select(app.conn, selectquery);

work_data2 = sql_abfrage(1,:);
work_data2(:, 1) = [];
app.Zusatzdaten.Data = table2cell(work_data2);
```

3.5.7

Im Code wird eine zusätzliche Bedingung eingebaut, die prüft, ob Zusatzdaten für das Bauteil vorhanden sind. Falls keine vorhanden sein sollten, wird eine Fehlermeldung ausgegeben. Ansonsten wird das Eingabefeld für den Lieferanten an die vorgegebene Darstellung angepasst. In der Datenbank ist nur die zugehörige ID hinterlegt, aber im Feld wird die ID mit der Beschreibung gekoppelt. Dazu wird die zugehörige Beschreibung aus der Datenbank abgefragt und verbunden mit der ID in das Lieferantenfeld der Zusatzdatentabelle übertragen.

```
if height(work_data2) ~= 0

   sqlvalue_lieferant = app.Zusatzdaten.Data{1,1};
   selectquery = ['SELECT * FROM geschmacksachekaffee.lieferant WHERE ...
                  id_lieferant = ', num2str(sqlvalue_lieferant)];
   lieferant_alt = select(app.conn, selectquery);
   app.Zusatzdaten.Data(1,1) = strcat(num2str(lieferant_alt{1,1}), ...
                               {' - '}, lieferant_alt{1,2});
Else

   app.RevisionanlegenButton.BackgroundColor = 'r';

   app.Grunddaten.Data{1,1} = 'unexpected error no match of teile ...
                               ID to zusatzdaten';
end
```

3.5.8

Mit diesen Zeilen ist der Programmcode für die Grunddatentabelle vollständig.

3.5.2 Zusatzdatentabelle

Diese Tabelle ist eine reine Eingabetabelle und bis auf die Lieferantenauswahl beeinflusst diese keine anderen Elemente der MATLAB-GUI. Die Daten werden in einer eigenen Tabelle und das Lieferantenfeld in einer eigenen Zelle eingefügt. In einer Anweisung wird für eine valide Auswahl des Lieferanten die zugehörige ID in das Teilenummern-Array eingefügt. Dies wird mit dem Vergleich der Vorauswahl erreicht. Außerdem wird durch eine zweite bedingte Anweisung festgestellt, ob die Option Lieferant unbekannt gewählt wurde. Falls dem so ist, werden alle davon abhängigen Felder vorausgefüllt. Das bedeutet, Preise werden null und die Bestellnummer auf "TBD" gesetzt, da diese Informationen ohne eine Bauteilanfrage bei einem Lieferanten noch nicht vorliegen.

```
app.zusatzdaten = app.Zusatzdaten.Data;
lieferantnr{1,1} =  app.zusatzdaten(1,1);

if strcmpi(lieferantnr{1,1},'Bitte Lieferant Auswählen') == 0
   app.teilenummer{1,8} = str2double(strtok(lieferantnr{1,1}, {' '}));

   if strcmpi(cell2mat(lieferantnr{1,1}),' 0 - Unbestimmt')
      app.Zusatzdaten.Data{1,2}  = 0;
      app.Zusatzdaten.Data{1,3}  = 'TBD';
      app.Zusatzdaten.Data{1,8}  = 0;
      app.Zusatzdaten.Data{1,9}  = 0;
      app.Zusatzdaten.Data{1,10} = 0;
      app.Zusatzdaten.Data{1,11} = 0;
      app.Zusatzdaten.Data{1,12} = 0;
      app.Zusatzdaten.Data{1,13} = 0;

   end
end
```

3.5.9

In der Zusatzdatentabelle ist ebenfalls eine Bedingung für die Erzeugung der Teilenummer enthalten. Diese wird hier erzeugt, wenn alle Felder des Teilenummern-Arrays befüllt sind. Dafür wird die Funktion "isempty" in Verbindung mit "cellfun" auf das Array angewandt.

```
if sum(cellfun('isempty', app.teilenummer)) < 0
  hilf_teilenummer = app.teilenummer;
  hilf_teilenummer(:,1) = [];
  app.TeilenummerEditField.Value = strjoin(string(hilf_teilenummer),'-');

else
  app.TeilenummerEditField.Value = app.TeilenummerEditField.Value;
end
```

3.5.10

Die Erzeugung der Teilenummer muss für die Funktionen Revision und Bauteil bearbeiten noch einmal separat auf einer anderen Grundlage erzeugt werden. Grund hierfür ist die Bearbeitungsreihenfolge der Daten und ggf. werden die Grunddaten nicht manuell angepasst, aber das Datum und die Version ändern sich trotzdem automatisch. Hierfür werden im Folgenden zwei bedingte Anweisungen verwendet.

```
if app.funktion == 4 || app.funktion == 5
 if isempty(app.bauteilgrunddaten{1,3}) == 0 && ...
              isempty(app.bauteilgrunddaten{1,4}) == 0 && ...
              isempty(app.teilenummer{1,8}) == 0
    hilf_teilenummer = app.teilenummer;
    hilf_teilenummer(:,1) = [];
    app.TeilenummerEditField.Value = strjoin(string ...
                                          (hilf_teilenummer), '-');
    app.bauteilgrunddaten = app.Grunddaten.Data;
 end
end
```

3.5.11

3.6 Speichern Funktion

Die Speicherfunktion ist vielseitig und aufgrund der Datenstruktur in fünf unterschiedliche Fälle unterteilt. Jeder Fall ist für eine andere Funktion des Teilenummerngenerators zuständig. In MATLAB wird dies durch einen Switch Case mit der Variable Funktion realisiert. Falls die Variable der Funktion keine gültige Auswahl bzw. keinen gültigen Fall darstellt, wird die Variable zurück auf null gesetzt und der Speichern-Button rot dargestellt. Dies ist zum Beispiel der Fall, wenn der Knopf gedrückt wird, ohne zuerst eine andere Funktion gewählt zu haben. Ein ungewolltes vorzeitiges Speichern wird hierdurch verhindert. Der Programmcode gestaltet sich dafür wie folgt.

```
switch app.funktion

        case 1
        % Bauteil anlegen (siehe Programmcode 3.6.2 bis 3.6.5)

        case 2
        % Baugruppe anlegen

        case 3
        % Komponente anlegen

        case 4
        % Revision anlegen

        case 5
        % Bauteil bearbeiten

        otherwise

          app.funktion = 0;
          app.SpeichernButton.BackgroundColor = 'r';

end
```

3.6.1

Damit Daten nicht unvollständig in der Datenbank angelegt werden können, müssen die Eingabetabellen auf leere Zellen geprüft werden. Hierfür reicht eine einfache "isempty" Funktion in MATLAB nicht aus, da über die Tabellen mit einem Datenarray gearbeitet wird. Deshalb wird zusätzlich die Funktion "cellfun" benötigt, welche eine Funktion auf jede Zelle innerhalb eines Arrays anwendet. Die Summe der leeren Zellen wird mit Null verglichen. Sollten eine oder mehrere Zellen leer sein, ist ein Speichern der Daten nicht möglich und der Speichern-Button wird rot dargestellt. Dieser Abgleich findet in den ersten drei Fällen der Speicherfunktion mit Eingrenzung auf die verwendeten Tabellen Anwendung.

```
if sum(cellfun('isempty', app.Zusatzdaten.Data)) > 0 || ...
   sum(cellfun('isempty', app.Grunddaten.Data)) > 0

   app.SpeichernButton.BackgroundColor = 'r';

else

% Daten speichern (siehe Programmcode 3.6.4 bis 3.6.10)

end
```

3.6.2

Bei erfolgreichem Speichern wird der Speichern-Button als Feedback für den Benutzer grün dargestellt. Dies wird durch das Anhängen folgender Programmzeile am Ende jeden Falles erreicht.

```
app.SpeichernButton.BackgroundColor = 'g';
```

3.6.3

Für ein neues Bauteil werden die Grund- und Zusatzdaten getrennt voneinander in der Datenbank angelegt. Für das Anlegen von neuen Datensätzen wird in MATLAB die Funktion "sqlwrite" benutzt. Bevor die Daten allerdings gespeichert werden können, muss die Grunddatentabelle, wie sie in der GUI steht, in den Spalten noch an die Teiletabelle der Datenbank angepasst werden. Dazu muss an zweiter Stelle noch die Komponenten-ID eingeschoben und am Ende die Teilenummer eingefügt werden. Für die Komponenten-ID wird ein temporäres Array erzeugt, welches als erstes Element die Teile-ID und als zweites die Komponenten-ID erhält. Für die Teilenummern wird ein einzelnes Array erzeugt und dieses mit "strjoin" zusammengefügt. Die Teile-ID wird anschließend aus der Grunddatentabelle entfernt und die drei Arrays miteinander verkettet. Für das Speichern muss das Array noch in eine Tabelle mit den jeweiligen Spaltenköpfen der Datenbanktabelle umgewandelt werden. Die Position der Daten muss hierbei mit der Position des Spaltennamens übereinstimmen.

```
% Bedingung zum Abfragen von leeren Zellen (siehe Programmcode 3.6.2)
else
% Grunddatenspeichern
hilf_komponente_id{1,1} = app.teileID;
hilf_komponente_id{1,2} = str2num(app.komponente_id);
hilf_teilenr{1,1} = strjoin(string(app.teilenummer), '-');
app.bauteilgrunddaten(:, 1) = [];

data_combined = horzcat(hilf_komponente_id, app.bauteilgrunddaten, ...
                        hilf_teilenr);

saving_data = cell2table(data_combined, 'VariableNames',["id_teil" ...
                         "id_komponente" "beschreibung_teil" "typ" ...
                         "status" "version" "datum" "teile_nr"]);

sqlwrite(app.conn, 'geschmacksachekaffee.teil', saving_data);

%Zusatzdatenspeichern (siehe Programmcode 3.6.5)
```

3.6.4

Das Speichern der Zusatzdaten erfolgt mit dem gleichen Schema der Grunddaten. Für die Vollständigkeit der Tabelle wird noch die Lieferanten-ID und die Teilenummer benötigt. Diese werden hierfür auch über ein Hilfsarray eingefügt. Eine Besonderheit tritt bei der Mindestbestellmenge und bei dem Gewicht auf, diese werden zwar als Zahl in die UI-Tabelle eingegeben, aber in MATLAB frequentativ als String erkannt. Um diese potenzielle Fehlerquelle zu umgehen, wird der Wert mit der Funktion "str2num" umgewandelt.

```
hilf_teilenr_id{1,1} = app.teilenummer{1,8};
hilf_teilenr_id{1,2} = strjoin(string(app.teilenummer), '-');
app.zusatzdaten(:, 1) = [];

%Mindestbestellmenge und Gewicht in Zahlenformat umwandeln
if app.zusatzdaten{1,1} ~= 0
   app.zusatzdaten{1,1} = str2num(app.zusatzdaten{1,1});
end
app.zusatzdaten{1,4} = str2num(app.zusatzdaten{1,4});

data_combined2 = horzcat(hilf_teilenr_id, app.zusatzdaten);
saving_data2 = cell2table(data_combined2, VariableNames', ...
   ["id_lieferant" "teile_nr" "mindestbestellmenge" "bestellnummer" ...
    "material" "gewicht" "produktionsmethode" "nachbehandlungsmethode" ...
    "produktionskosten" "nachbehandlungskosten" "nebenkosten"...
    "stueckpreis_1" "stueckpreis_10" "stueckpreis_100"]);

sqlwrite(app.conn, 'geschmacksachekaffee.teil_lieferant', saving_data2)
app.SpeichernButton.BackgroundColor = 'g';
end
```

3.6.5

Das Speichern bzw. Anlegen von einer Baugruppe und Komponente stellt einen stark vereinfachten Prozess einer Bauteilanlage dar. Hierfür sind nur gekürzte Grunddaten vorhanden. Das Prüfen auf leere Zeilen und

Speichern muss also nur für die Grunddatentabelle erfolgen. Beide Funktionen sind zueinander analog und es unterscheiden sich nur die Variablennamen. Aufgrund der Einfachheit wird auf eine ausführlichere Beschreibung in diesem Fall verzichtet. Der Programmcode für das Speichern einer Baugruppe gestaltet sich wie folgt.

```
% Bedingung zum Abfragen von leeren Zellen (s. 3.6.2 nur Grunddaten)
else
% Baugruppe speichern
data_combined = app.bauteilgrunddaten;
saving_data = cell2table(data_combined, 'VariableNames', ...
              ["id_baugruppe" "id_modul" "beschreibung_baugruppe"]);

sqlwrite(app.conn, 'geschmacksachekaffee.baugruppe', saving_data);
app.SpeichernButton.BackgroundColor = 'g';
end
```

3.6.6

Analog dazu sieht der Programmcode auf eine Komponente eingepasst somit wie folgt aus.

```
% Bedingung zum Abfragen von leeren Zellen (s. 3.6.2 nur Grunddaten)
else
data_combined = app.bauteilgrunddaten;
saving_data = cell2table(data_combined,VariableNames', ...
          ["id_komponente" "id_baugruppe" "beschreibung_komponente"]);

sqlwrite(app.conn, 'geschmacksachekaffee.komponente', saving_data);
app.SpeichernButton.BackgroundColor = 'g';
end
```

3.6.7

Der vierte Fall der Speicherfunktion ist für das Anlegen neuer Versionen von bereits vorhandenen Bauteilen zuständig. Dieser unterscheidet sich kaum zum ersten Fall. Hierbei können allerdings kleine Feinheiten weggelassen werden, um den Programmcode zu kürzen. Die Umwandlung in ein Zahlenformat ist nicht nötig, da die Daten bereits richtig aus der Datenbank importiert werden. Auf die Abfrage nach leeren Zellen kann ebenfalls verzichtet werden. Grund hierfür ist, dass das Teil den ersten Fall bereits durchlaufen haben muss, um in der Datenbank zu existieren. Obwohl der Programmcode vollständig aus dem Code für das Bauteil anlegen abgeleitet werden könnte, wird dieser aus Gründen der Vollständigkeit dennoch abgedruckt.

```
% Grunddatenspeichern
hilf_komponente_id(1,1) = app.teileID;
hilf_komponente_id(1,2) = app.komponente_id;
hilf_teilenr{1,1} = cellstr(strjoin(string(app.teilenummer), '-'));
app.bauteilgrunddaten(:, 1) = [];
data_combined = horzcat(hilf_komponente_id, app.bauteilgrunddaten, ...
                        hilf_teilenr);
saving_data = cell2table(data_combined, 'VariableNames',["id_teil" ...
                         "id_komponente" "beschreibung_teil" "typ" ...
                         "status" "version" "datum" "teile_nr"]);
sqlwrite(app.conn, 'geschmacksachekaffee.teil', saving_data)
%Zusatzdatenspeichern
hilf_teilenr_id{1,1} = app.teilenummer{1,8};
hilf_teilenr_id{1,2} = strjoin(string(app.teilenummer), '-');
app.zusatzdaten(:, 1) = [];

data_combined2 = horzcat(hilf_teilenr_id, app.zusatzdaten);
saving_data2 = cell2table(data_combined2, 'VariableNames', ...
  ["id_lieferant" "teile_nr" "mindestbestellmenge"
  "bestellnummer" "material" "gewicht" "produktionsmethode" ...
  "nachbehandlungsmethode" "produktionskosten" "nachbehandlungskosten" ...
   "nebenkosten" "stueckpreis_1" "stueckpreis_10" "stueckpreis_100"]);

sqlwrite(app.conn, 'geschmacksachekaffee.teil_lieferant', saving_data2)
app.SpeichernButton.BackgroundColor = 'g';
```

3.6.8

Als letztes wird nun der Fall für das Bearbeiten von bereits vorhandenen Bauteilen betrachtet. Hierbei gibt es einen großen Unterschied zu den vorherigen Fällen. Während bei diesen die Daten neu in die Datenbank aufgenommen wurden, werden im fünften Fall die Daten aktualisiert bzw. ersetzt. Hierfür wird die Funktion "update" anstatt "sqlwrite" für das Aktualisieren von Datenbanken verwendet. Das Prüfen auf leere Zellen kann hierbei ebenfalls vernachlässigt werden, da die Ursprungsdaten aus der Datenbank importiert werden. Für das Ersetzten von vorhandenen Daten wird eine entsprechende Eingrenzung benötigt, um die zu ersetzenden Daten eindeutig zu identifizieren. Im Programmcode wird dies durch die Variable "whereclause" realisiert, welche als Wert die Teilenummer besitzt. Der Aufbau der Speicherdaten ist hierbei analog zum vorherigen Abschnitt, allerdings werden diese nicht in eine Tabelle umgewandelt, sondern als Array behandelt. Der Datentyp der Teilenummer und des Datums muss auf double geändert werden, da mit einem Array anstatt einer Tabelle gearbeitet wird. In einem zweiten Array werden die zugehörigen Spaltennamen behandelt, die Position der Daten muss hierbei mit den Spalten übereinstimmen. Die Teile-ID wird hierbei weggelassen, da eine Änderung dieser nicht vorgesehen ist.

```
% Grunddatenspeichern
hilf_teilenr{1,1} = strjoin(string(app.teilenummer), '-');

app.bauteilgrunddaten(:, 1) = [];
app.bauteilgrunddaten(:, 4) = [];

data_combined = horzcat(app.bauteilgrunddaten, hilf_teilenr);
data_combined{1,4} = double(data_combined{1,4});
data_combined{1,5} = cell2mat(data_combined{1,5});

colnames = {'beschreibung_teil', 'typ', 'status', 'datum', 'teile_nr'};

whereclause = strjoin(string(['WHERE teile_nr = ','''', ...
                              app.teilenummer_alt, '''']));

update(app.conn, 'geschmacksachekaffee.teil', colnames, ...
        data_combined, whereclause)

%Zusatzdatenspeichern (siehe Programmcode 3.6.10)
```

3.6.9

Für das Speichern der Zusatzdaten wird derselbe "whereclause" wie für die Grunddaten verwendet. Im Zusatzdaten-Array müssen die Werte für die Teilenummer, Mindestbestellmenge und Gewicht ebenfalls angepasst werden.

```
hilf_teilenr_id{1,1} = app.teilenummer{1,8};
hilf_teilenr_id{1,2} = strjoin(string(app.teilenummer), '-');
app.zusatzdaten(:, 1) = [];

data_combined2 = horzcat(hilf_teilenr_id, app.zusatzdaten);
data_combined2{1,2} = cell2mat(data_combined2{1,2});
data_combined2{1,3} = double(data_combined2{1,3});
data_combined2{1,6} = double(data_combined2{1,6});

colnames2 = {'id_lieferant', 'teile_nr', 'mindestbestellmenge', ...
 'bestellnummer' 'material', 'gewicht', 'produktionsmethode', ...
 'nachbehandlungsmethode','produktionskosten','nachbehandlungskosten',...
 'nebenkosten', 'stueckpreis_1', 'stueckpreis_10', 'stueckpreis_100'};

update(app.conn, 'geschmacksachekaffee.teil_lieferant', colnames2, ...
       data_combined2, whereclause)

app.SpeichernButton.BackgroundColor = 'g';
```

3.6.10

3.7 Sonstige Funktionen

In diesem Kapitel werden alle kleineren Funktionen der GUI erläutert, welche nicht zwingend einer aktiven Ausführung benötigen.

3.7.1 Serverstatus

Beim Starten der App wird eine Verbindung zur Datenbank aufgebaut, diese muss während der gesamten Bearbeitungszeit in der App geöffnet bleiben. Ohne die Verbindung kann mit der GUI nicht gearbeitet werden. Deshalb befindet sich im oberen rechten Eck der App eine Statusleuchte. Durch ein Startup-Callback wird der Serverstatus über die "isopen" Funktion abgefragt, bevor die GUI sich vollständig aufgebaut hat. Durch eine bedingte Anweisung wird bei erfolgreicher Verbindung das Lämpchen grün und bei gescheiterter Verbindung rot.

```
i = isopen(app.conn);

if i == 1
    app.Serverstatus.Color = 'green';
else
    app.Serverstatus.Color = 'red';
end
```

3.7.1

3.7.2 Speichern-Button sperren

Zusätzlich zum Serverstatus befindet sich eine Programmzeile in der Startup-Funktion, welche die Variable für den Speichern-Button auf null setzt. Somit kann dieser erst ausgeführt werden, wenn dies durch einen anderen Programmabschnitt vorgesehen ist.

```
app.funktion = 0;
```

3.7.2

3.7.3 Reset-Button

Mit dem Reset-Button wird die App geschlossen und neu gestartet. Für das Schließen der App wird die "delete" Funktion verwendet. Durch diese Funktion wird das gesamte Userinterface gelöscht, was einem Schließen der App entspricht. Diese kann anschließend erneut gestartet werden, in dem man den MATLAB Programmname aufruft. Zusätzlich wird die Datenbankverbindung über die "close" Funktion getrennt. In MATLB werden folglich für die Umsetzung des Befehls drei Programmcodezeilen benötigt.

```
close(app.conn)
delete(app.UIFigure);
Kosten_und_Lieferantenmanagement;
```

3.7.3

3.7.4 Copy Button

Der Knopf mit dem Namen Copy ermöglicht es, die Teilenummern in das Benutzer-Clipboard zu kopieren. Dadurch kann die Teilenummer überall durch die Eingabe Einfügen am Computer eingefügt werden. In MATLAB wird für die Umsetzung die Funktion "clipboard" benötigt.

```
clipboard('copy', app.TeilenummerEditField.Value)
```

3.7.4

3.7.5 Close Request

Mit dem Starten der App wird eine aktive Datenbankverbindung erstellt. Diese muss beim Beenden der App geschlossen werden. Die Handlung des Schließens lässt sich in MATLAB durch den Callback "CloseRequest" abfangen. Für das Trennen der Verbindung wird die "close" Funktion verwendet.

```
close(app.conn)
delete(app)
```

3.7.5

4 Kostenkalkulator

Ein Kostenkalkulator ist ein wichtiges Werkzeug, um die Kosten von Projekten oder Produkten zu berechnen und zu vergleichen. Es ermöglicht Unternehmen, die Ausgaben für Material, Arbeitszeiten und andere Faktoren zu schätzen, um einen realistischen Preis für ihre Angebote zu berechnen. Ein Kostenkalkulator kann auch dazu verwendet werden, die Wirtschaftlichkeit von Projekten zu überprüfen und Entscheidungen über die Fortführung oder Beendigung von Projekten zu treffen.
Die Daten für die Preisbestimmungen sind bisher in mehreren Excellisten abgelegt. Mit zunehmender Teileanzahl und Komplexität der Maschine hat die Übersichtlichkeit stark gelitten. Es gibt keine einfache Möglichkeit mehr, mit wenig Aufwand den aktuellen Preis einer Maschine zu bestimmen. Deshalb ist der Kostenkalkulator von größter Bedeutung für das Projekt der Glasboilermaschine, da es dadurch ermöglicht wird, den aktuellen Preis einer Maschine in wenigen Augenblicken zu bestimmen. Außerdem ist die kalkulatorische Abbildung der Einflussfaktoren bzgl. des Preises in Excel nicht mehr umsetzbar. Der Kostenkalkulator soll automatisiert die Preise für eine Maschine in der Prototypfertigung, sowie in der Serienfertigung mit verschiedenen Losgrößen bestimmen. Die definierten Losgrößen sind hierfür eins, fünf, zehn und zwanzig und sind somit als Kleinserie einzuordnen. Wichtig ist es hierfür, die unterschiedlichen Lieferbedienungen der verschiedenen Teile und Lieferanten zu beachten. Für manche Teile gibt es Mindestbestellmengen, unterschiedliche Staffelpreise und Einrichtungs- bzw. Rüstkosten. Bei Lieferanten kann es Mindestbestellsummen, Lieferkosten und Rabatte geben. Eine weitere Herausforderung stellen Mehrfachlieferanten dar. Dabei kann ein Bauteil bei unterschiedlichen Lieferanten zu unterschiedlichen Konditionen bezogen werden. Hierfür muss der Kostenkalkulator situationsbedingt das wirtschaftlichere Bauteil auswählen. Zu dem Gesamtpreis pro Maschine sollen auch die dadurch entstehenden Zusatzkosten ausgewiesen werden. Ebenso sollen die Preise für jede Baugruppe einzeln abgebildet werden. Zur Preisbestimmung sollen unmittelbar die Bestelllisten für die entsprechend gewählte Losgröße erzeugt werden. Diese sollen alle Bauteile und dazugehörigen Informationen enthalten, um mit diesen unmittelbar eine Bestellung für die entsprechende Maschine in der zugehörigen Losgröße auszulösen.

4.1 GUI-Kostenkalkulator

Der Kostenkalkulator ist im zweiten Tab der MATLAB-App zu finden und umfasst drei Dropdowns, zwei Buttons und eine Tabelle. In den Dropdowns können die Eingaben getroffen werden, welche Maschine, welcher Status und welche Losgröße für die Kalkulation verwendet werden. Die Buttons übergeben die jeweilige Funktionsaufforderung zum Kalkulieren der Preise oder Erstellen der Bestelllisten. Im Zentrum der GUI befindet sich die Tabelle, in der die Preise nach der Kalkulation übersichtlich ausgegeben werden. Dabei werden die Preise pro Baugruppe ausgewiesen und dem übergeordneten Modul zugeordnet. Danach folgt eine Zwischensumme, welche sich aus der Summe aller Baugruppenpreise berechnet. Nach dieser werden die entstehenden Zusatzkosten und die Lieferkosten ausgegeben. Diese Kosten werden noch umgelegt auf eine einzelne Maschine und ergeben mit der Zwischensumme den Gesamtpreis, welcher sich im unteren rechten Ende der Tabelle befindet. Die Ausgabetabelle ist beispielhaft mit Testdaten in Abbildung 4 dargestellt.
Die MATLAB Callbacks befinden sich in der App hinter den Buttons zum Auslösen der Funktionen und in den Dropdowns zur Aufnahme der geänderten Parameter.

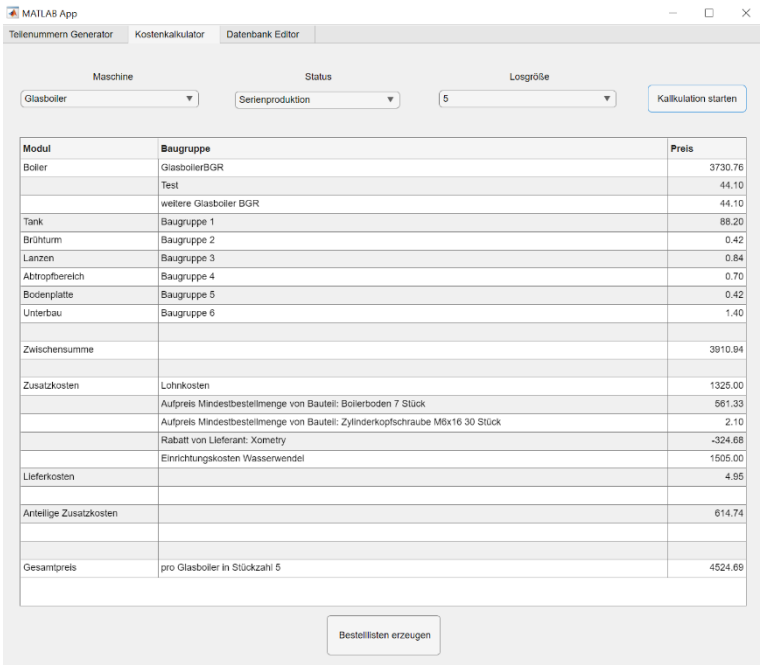

Abbildung 4: GUI-Kostenkalkulator

4.2 Parameterwahl

Für die Kalkulation können drei verschiedene Parameter eingestellt werden: die Maschine, der Status und die Losgröße. Die Auswahlmöglichkeiten sind hierfür in dem Dropdown Element selbst hinterlegt und nicht im Programmcode. Alle drei haben eine null Bedingung, die eintritt, falls die "Bitte (…) wählen" Option selektiert wird. Diese stellt keine valide Auswahlmöglichkeit dar und das Programm kann somit nicht in die Kalkulationsphase wechseln. Diese Variable ist in den Menüs beim Starten der App immer voreingestellt.

Die Maschinenwahl umfasst die Optionen Bitte Maschine wählen, Glasboiler und Labormaschine. Aufgrund dieser Auswahl wird die entsprechende Stückliste aus der Datenbank aufgerufen. Durch das Ändern der Variable wird der Wert mit der Funktion "strcmpi" verglichen und für die übereinstimmenden Bedingung der entsprechende SELECT String generiert.

```
value = app.MaschineDropDown.Value;
if strcmpi(value,'Glasboiler')
   app.maschinenwahl='SELECT * FROM ...
                      geschmacksachekaffee.stueckliste_glasboiler';

elseif strcmpi(value,'Labormaschine')
   app.maschinenwahl='SELECT * FROM ...
                      geschmacksachekaffee.stueckliste_labormaschine';
else
      app.maschinenwahl = 0;
end
```

4.2.1

Für die Wahl des Status stehen die Variablen Prototyp und Serienfertigung zur Verfügung. Über die Funktion "strcmpi" wird ebenfalls die Auswahl mit dem entsprechenden Wert abgeglichen und die Variable Statuswahl mit der dafür definierten numerischen Zahl belegt.

```
value = app.StatusDropDown.Value;

if strcmpi(value,'Prototyp')
      app.statuswahl = 1;
elseif strcmpi(value,'Serienproduktion')
      app.statuswahl = 2;
else
      app.statuswahl = 0;
end
```

4.2.2

Als dritter Parameter wird die Losgröße benötigt. Für diese stehen die Werte 1, 5, 10 und 20 zur Verfügung. Die Funktion "strcmpi" wird hierbei lediglich für das Herausfiltern, der wie bereits im vorherigen Absatz beschriebenen, nicht validen Auswahlmöglichkeit, verwendet. Abweichend zu den anderen Dropdown-Menüs muss hier die Eingabe mit der Funktion "str2num" in eine Zahl umgewandelt werden.

```
value = app.LosgreDropDown.Value;

if strcmpi(value,'Bitte Losgröße wählen')
      app.lot = 0;
 else
      app.lot = str2num(value);
end
```

4.2.3

Damit die Kalkulation erst durch Drücken des Buttons ausgelöst werden kann, wenn alle Parameter erfolgreich eingegeben wurden, ist der Kostenkalkulator in eine Bedingung gehüllt. Diese fragt ab, ob einer der Parameter gleich null ist. Diese werden zu null beim Starten der App und falls eine invalide Option gewählt wird. Sollte dies der Fall sein, färbt sich der Button rot und es geschieht nichts.

```
if  app.maschinenwahl == 0 | app.statuswahl == 0 | app.lot == 0

      app.KalkulationstartenButton.BackgroundColor = 'r';
```

4.2.4

```
else
      %Kostenkalkulator Programmcode (s. 4.3.1 bis 4.6.10)
end)
```

4.3 Kalkulationsliste

Als erster Schritt für jede Kalkulation wird die entsprechende Stückliste mit den enthaltenen Teilen in der jeweiligen Quantität benötigt. Diese wird aus der Datenbank in MATLAB importiert und weist, wie bereits im Kapitel zur Datenbankbeschrieben, folgendes Schema auf:

Tabelle 9: DB Tabelle Stückliste mit Beispieldaten

id_list	id_teil	id_baugruppe	anzahl
1	11101	111	10

Die erste Spalte stellt hierbei lediglich eine Datenbank ID dar und ist vergleichbar zu einer Positionsnummer in einer Zeichnungsstückliste. Diese gibt für den Kostenkalkulator keine wertvolle Information her und wird somit nicht benötigt. Das Importieren der entsprechenden Stückliste mit den benötigten Spalten erfolgt in MATLAB über den folgenden Code:

```
stueckliste = select(app.conn, app.maschinenwahl);
```
4.3.1

Aus der importierten Stückliste wird die Kalkulationstabelle erstellt. Zwischen den beiden Listen gibt es einen gravierenden Unterschied. In der Stückliste ist der Teilebedarf für eine einzelne Maschine hinterlegt, während die Kalkulationsliste den Teilebedarf für die zu fertigende Losgröße enthält. Außerdem werden in der Kalkulationsliste die Stückzahlen gleicher Bauteile addiert, um den richtigen Staffelpreis zu ermitteln. Diese Addition wird durch die Funktion "groupsummary" umgesetzt und die errechnete Stückzahl anschließend mit der gewählten Losgröße multipliziert. Im Zwischenschritt werden die nicht benötigten Informationen aus der Tabelle entfernt.

```
kalkliste = groupsummary(stueckliste, "id_teil", 'sum');
kalkliste(:,[2 3 4]) = [];
kalkliste.sum_anzahl = kalkliste.sum_anzahl * app.lot;
```
4.3.2

Nachdem die Kalkulationsliste erstellt ist, muss das jeweils zugehörige Bauteil über die Teile-ID aus der Datenbank selektiert werden. Es wird für die richtigen Kalkulationsdaten die Teilenummer des entsprechenden Bauteils benötigt. Dabei gilt es zu beachten, dass es mehrere Teilenummern für eine Teile-ID geben kann. Grund hierfür sind unterschiedliche Versionsstände, Lieferanten und Status. Für jedes Bauteil wird also zunächst die Abfrage an die Datenbank gestartet, welche Datensätze es zur Teile-ID gibt. Danach wird die Abfrage auf die benötigten Status gekürzt. Für die Option Serienteil werden nur Teile mit gleichem Status beibehalten. Für den Status Prototyp werden nur Teile mit dem Status freigegeben beibehalten. Falls es keine Teile mit diesem Status geben sollte, wird auf den Status Serienteil zurückgegriffen.

```
sqlvalue = kalkliste{c,1};

selectquery = ['SELECT * FROM geschmacksachekaffee.teil WHERE ...
                id_teil = ', num2str(sqlvalue)];

sql_abfrage = select(app.conn, selectquery);

if app.statuswahl == 1

      proto_check = strcmp(sql_abfrage.status, 'F - Freigegeben');

   if proto_check == 1

      sql_abfrage = sql_abfrage(sql_abfrage.status == "F - Freigegeben",:);
   else
```
4.3.3

```
      sql_abfrage = sql_abfrage(sql_abfrage.status == "S - Serienteil",:);

   end

elseif app.statuswahl == 2

      sql_abfrage = sql_abfrage(sql_abfrage.status == "S - Serienteil",:);

end
```

Aus einer Liste vieler Bauteile wurde die Auswahl somit auf die zutreffenden Bauteile eingeschränkt. Um nun die passende Teilenummer zu finden, muss noch der Versionsstand verglichen werden. Hierbei darf nur auf aktuelle Versionen und somit folglich die höchste Version zugegriffen werden. Sollte es mehrere Bauteile mit dem gleichen und höchsten Versionsstand geben, handelt es sich um Teile mit mehreren Lieferanten. Diese werden in eine eigene Liste eingefügt, um sie vergleichen zu können und später das günstigste davon zu wählen. In der Kalkulationsliste werden diese entfernt.

```
topversion = max(sql_abfrage.version);
sql_abfrage = sql_abfrage(sql_abfrage.version == topversion,:);
%Bauteile mit mehreren Lieferanten in eigene Liste:
if height(sql_abfrage) > 1
      start_row = height(multi_lief);

      for z = 1:height(sql_abfrage)
            y_idx = start_row + z;
            multi_lief{y_idx,1} = kalkliste{c,1};
            multi_lief{y_idx,2} = kalkliste{c,2};
            multi_lief{y_idx,3} = sql_abfrage.teile_nr{z};
            multi_lief{y_idx,4} = sql_abfrage.beschreibung_teil{z};
      end

      row_indx(1, (length(row_indx)+1)) = c;

else
%Teilenummer und Beschreibung für Bauteil in Kalkulationsliste aufnehmen:
      kalkliste{c,3} = sql_abfrage.teile_nr;
      kalkliste{c,4} = sql_abfrage.beschreibung_teil;

end
```

4.3.4

Die vorherigen Programmierabschnitte bezüglich Bauteilversion und -status müssen für jedes Element in der Kalkulationsliste separat durchgeführt werden und sind daher in eine Schleife eingebettet. Nach der Schleife sind alle Teilenummern und Beschreibungen für Bauteile mit nur einem Lieferanten eingefügt und Bauteile mit mehreren Lieferanten in einer eigenen Liste mit denselben Informationen. Die Bauteile mit mehreren Lieferanten werden nach der Schleife aus der Kalkulationsliste mit folgender Programmzeile entfernt.

```
kalkliste(row_indx,:) = [];
```

4.3.5

Das Grundgerüst für die Preiskalkulation ist somit erstellt und es können die weiterführenden Informationen über die Teilenummer aus der Datenbank abgefragt werden. Von jedem Bauteil wird zunächst die Mindestbestellmenge benötigt, um diese mit der benötigten Stückzahl abgleichen zu können. Sollte diese höher sein als die benötigte Stückzahl, muss die zu bestellende Stückzahl erhöht werden. Danach kann mit der zu bestellenden Stückzahl der jeweilige Staffelpreis (1, 10, 100) ausgewählt werden. Eine Besonderheit stellen hierbei Normteile dar, da diese nur in Packungseinheiten bestellt werden können. Für diese reicht deshalb kein simpler Abgleich, ob die Mindestbestellmenge erreicht ist, sondern es muss immer auf die nächsthöhere Packungseinheit aufgerundet werden.

Um die Teilenummer als Variable für die SQL-Abfrage zu verwenden, muss diese mit einem Singlequote beginnen und enden, da es sich um ein Stringelement handelt. Dies geschieht durch die Funktion "strjoin" und der wiederholten Eingabe des Singlequotes. Dabei gilt es zusätzlich zu beachten, dass MATLAB in den Charakterstring Leerzeichen als Abtrennung zu den Singlequotes einfügt. Diese können mit zur Hilfenahme von "isspace" entfernt werden. Die SQL-Abfrage und das Bestimmen der Mindestbestellmenge zeigt der folgende Abschnitt.

```
sqlvalue = strjoin(['''',kalkliste{c,3} ,'''']);
sqlvalue = sqlvalue(~isspace(sqlvalue));

selectquery = ['SELECT * FROM geschmacksachekaffee.teil_lieferant WHERE ...
                teile_nr = ', sqlvalue];

sql_abfrage = select(app.conn, selectquery);

kalkliste{c,5} = sql_abfrage.mindestbestellmenge;
```

4.3.6

Daraufhin kann die Mindestbestellmenge mit der benötigen Stückzahl abgeglichen werden. Die Unterscheidung zwischen Normteilen erfolgt hierbei durch eine Bedingung mit der Funktion "contains". Alle Normteile besitzen ein unverkennbares N in der Teilenummer, auf welches ein simpler Abgleich gemacht wird. Damit die Packungsgrößen nur als ganzzahliges Ergebnis ausgegeben werden, wird noch die Funktion "ceil" benötigt. Diese rundet immer auf die nächsthöhere Integerzahl auf (z. B. 4.1 → 5).

```
if contains(kalkliste{c,3}, 'N') == 1

   kalkliste{c,6} = ceil(double(kalkliste{c,2}) / ...
                     double(kalkliste{c,5})) * kalkliste{c,5};

elseif kalkliste{c,5} > kalkliste{c,2}

   kalkliste{c,6} = kalkliste{c,5};

else

   kalkliste{c,6} = kalkliste{c,2};

end
```

4.3.7

Sobald die zu bestellende Stückzahl ermittelt ist, kann über diese der zugehörige Staffelpreis aus der Datenbank abgefragt werden.

```
if kalkliste{c,6} >= 100
      kalkliste{c, 7} = sql_abfrage.stueckpreis_100;
elseif kalkliste{c,6} >= 10
      kalkliste{c, 7} = sql_abfrage.stueckpreis_10;
else
      kalkliste{c, 7} = sql_abfrage.stueckpreis_1;
end
```

4.3.8

Da diese Preisbestimmung für jedes Element in der Kalkulationsliste nötig ist, wird der entsprechende Programmcode ebenfalls in eine Schleife gesetzt. In dieser werden zusätzlich noch die Lieferant-ID und Bestellnummer in die Kalkulationsliste mit aufgenommen.

```
kalkliste{c, 8} = sql_abfrage.id_lieferant;
kalkliste{c, 9} = sql_abfrage.bestellnummer;
```

4.3.9

Die Vorgehensweise für die Liste mit Mehrfachlieferanten ist analog zu den in den vorherigen Absätzen beschriebenen Verfahren. Deshalb wird auf eine ausführliche Erklärung verzichtet, da sich im Grunde nur die Variablen von "kalkliste" auf "multi_lief" ändern. Allerdings gibt es dabei zwei Besonderheiten. Eine ist, dass die Zusatzkosten gleich in dieser Schleife abgefragt werden, da diese für die Bestimmung des

günstigeren Lieferanten benötigt werden. Die andere ist, dass sich die Schleife in einer Bedingung befindet, damit diese nur aufgeführt wird, wenn es für mindestens ein Bauteil zwei oder mehr Lieferanten gibt.

Die Abfrage bzgl. der Zusatzkosten wird über untenstehenden Programmcode durchgeführt. Dabei wird die hinterste Zelle mit dem Wert befüllt. Die dazwischen liegenden Zellen werden ebenfalls gleich mit einer null versehen, um einerseits das richtige Format zu haben und anderseits, da manche den Wert null erhalten. Genauere Informationen dazu folgen im nächsten Kapitel.

```
if height(multi_lief) > 1

%Schleife wie bei Kalkulationsliste

   selectquery2 = ['SELECT * FROM geschmacksachekaffee.zusatzkosten ...
                    WHERE teile_nr = ', sqlvalue];

   sql_abfrage_zusatzkosten = select(app.conn, selectquery2);

   if height(sql_abfrage_zusatzkosten) > 0

      if sql_abfrage_zusatzkosten.einmalkosten(1) == 0
         multi_lief{c,15} =  sql_abfrage_zusatzkosten.betrag(1);
         (...)
         multi_lief{c,11} = 0;
      else
         multi_lief{c,15} = 0;
         (...)
         multi_lief{c,11} = 0;
      end

   else
      multi_lief{c,15} = 0;
      (...)
      multi_lief{c,11} = 0;
   end

   %Preis- und Datenzuordnung wie bei Kalkulationsliste (s. 4.3.6 bis 4.3.9)

end
```

4.3.10

4.4 Mehrfach Lieferanten

Dieses Thema stellt die größte Herausforderung an den Kostenkalkulator dar. Die Bauteile müssen hierfür auf den realen Stückpreis heruntergebrochen werden, um diese vergleichen zu können. Auf den realen Stückpreis haben die Faktoren Rabatt, Lieferkosten, Mindestbestellsumme und Zusatzkosten (z. B. Rüstkosten) einen Einfluss. Dabei gilt es zu beachten, dass sich die Summen sowohl positiv als auch negativ auf den Preis auswirken können. Sollte z. B. die Mindestbestellsumme erst durch die zusätzliche Bestellung dieser Teile erfüllt sein, gilt dies als Einsparung anzusehen, da diese Summe dennoch bezahlt werden müsste. Deshalb müssen vor dem Vergleich zunächst die Bestellsummen aller Lieferanten auf Basis der anderen Bauteile bestimmt werden, bevor der Realpreis ermittelt werden kann. Die jeweiligen Entscheidungsbaum-Diagramme sollen die Fälle für Rabatt, Lieferkosten und Mindestbestellsumme vereinfacht darstellen. Für die Zusatzkosten gibt es keine ausschreitende Entscheidung, da diese vorhanden oder nicht vorhanden sein können.

Für die Übersichtlichkeit werden in den Diagrammen die Bezeichnungen Bestellsumme anderer Bauteile (BS-ABT), was die Summe aus den zu bestellenden Bauteilen bei diesem Lieferanten beschreibt und die Bestellsumme dieses Bauteils (BS-DBT), worunter nur die reine Bestellsumme für das Teil in der zu bestellenden Stückzahl zu verstehen ist. Folgende Auflistung beinhaltet die in den Diagrammen verwendeten Abkürzungen:

- VK: Verrechnungskosten
- LK: Lieferkosten
- BS-MIN: Mindestbestellsumme
- BS-ABT: Bestellsumme anderer Bauteile (bei diesem Lieferanten)
- BS-DBT: Bestellsumme dieser Bauteile
- BS-LKF: Bestellsumme, ab welcher keine Lieferkosten anfallen

In Abbildung 5 ist das Entscheidungsbaum-Diagramm für die Findung der Verrechnungskosten, in Bezug auf das Kriterium Mindestbestellsumme bildlich dargestellt. Für die Fälle, dass es keine Mindestbestellsumme für den entsprechenden Lieferanten gibt oder die Mindestbestellsumme bereits durch andere Teile erreicht ist, fallen keine Verrechnungskosten an.

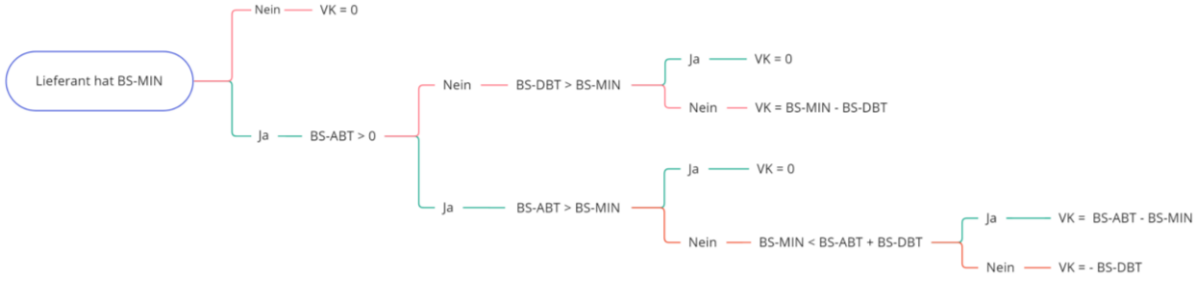

Abbildung 5: Entscheidungsbaum-Diagramm Mindestbestellsumme

Die anderen Fälle müssen genauer betrachtet werden, da diese sowohl positive als auch negative Auswirkungen auf den realen Bauteilpreis haben können. Für den Fall, dass der Lieferant eine Mindestbestellsumme aufweist und sonst keine Bauteile dort bestellt werden würden, erhöht sich der Realbauteilpreis um den Restbetrag der Mindestbestellsumme geteilt durch die entsprechende Stückzahl. Dann gibt es noch die Fälle, dass die Mindestbestellsumme durch dieses Bauteil in entsprechender Stückzahl erfüllt werden würde, oder immer noch nicht erfüllt ist. Das folgende Rechenbeispiel soll den Unterschied noch einmal deutlicher in Zahlen darstellen.

Für beide Fälle ist die Mindestbestellsumme des Lieferanten 500€ und die Bestellsumme anderer Bauteile 400€. Falls nun die Bestellsumme für das zu bestimmende Bauteil 80€ beträgt, ist die Mindestbestellsumme immer noch nicht erreicht, aber anstelle einer Überbezahlung von 100€, beträgt diese nunmehr 20€. Das bedeutet der Realpreis sinkt um anteilig 80€. Sollte die Bestellsumme für das zu bestimmende Bauteil 120€ betragen, ist die Bestellsumme erreicht. Eine Überbezahlung ist nun nicht mehr nötig und der geldwertende

Vorteil ist die Differenz der Mindestbestellsumme und der Bestellsumme anderer Teile. Dadurch sinkt der Realpreis anteilig um 100€.

In MATLAB werden diese drei Fälle durch eine bedingte Anweisung realisiert. Dabei müssen die drei Enden des Entscheidungsbaums abgegriffen werden, in denen die Verrechnungskosten nicht null betragen. Falls keiner dieser Fälle zutrifft, können die Verrechnungskosten einfach zu null gesetzt werden. Die entsprechenden Zeilen im Programm sehen dafür wie folgt aus.

```
%BS-MIN < BS-ABT + BS-DBT
if sql_abfrage.mindestbestellsumme(1) > multi_lief.bestellsumme_anderer_teile(c)...
   && sql_abfrage.mindestbestellsumme < (multi_lief.bestellsumme(c) + ...
   multi_lief.bestellsumme_anderer_teile(c))

      multi_lief.verrechnung_mindestbestellsumme(c) = ...
     multi_lief.bestellsumme_anderer_teile(c) - sql_abfrage.mindestbestellsumme(1);

%BS-MIN > BS-ABT + BS-DBT
elseif sql_abfrage.mindestbestellsumme(1) > multi_lief.bestellsumme_anderer_teile(c) ...
       && sql_abfrage.mindestbestellsumme > (multi_lief.bestellsumme(c) + ...
       multi_lief.bestellsumme_anderer_teile(c))

       multi_lief.verrechnung_mindestbestellsumme(c) = -multi_lief.bestellsumme(c);

%BS-DBT < BS-MIN
elseif multi_lief.bestellsumme_anderer_teile(c) == 0 && ...
       sql_abfrage.mindestbestellsumme(1) > multi_lief.bestellsumme(c)

       multi_lief.verrechnung_mindestbestellsumme(c) = ...
       sql_abfrage.mindestbestellsumme(1) - multi_lief.bestellsumme(c);

%für alle anderen Fälle
else
    multi_lief.verrechnung_mindestbestellsumme(c) = 0;
end
```

4.4.1

Für die Lieferkosten ist das entsprechende Diagramm in Abbildung 6 unten dargestellt. Hierbei gibt es zwei Fälle, die Einfluss auf den Realpreis haben. Einer davon tritt ein, wenn beim Lieferanten bereits bestellt wird und dieser ab einer gewissen Summe lieferkostenfrei zustellt. Wird die Bedingung für die lieferkostenfreie Zustellung durch das zu vergleichende Bauteil überschritten, entsteht hierbei ein geldwertender Vorteil. Die Lieferkosten würden in diesem Fall gespart werden. Im Gegenzug dazu besteht die Möglichkeit, dass bei dem Lieferanten für das zu vergleichende Bauteil sonst nicht bestellt werden würde. Fallen beim entsprechenden Lieferanten dadurch Lieferkosten an, sind diese als Verrechnungskosten anzusehen.

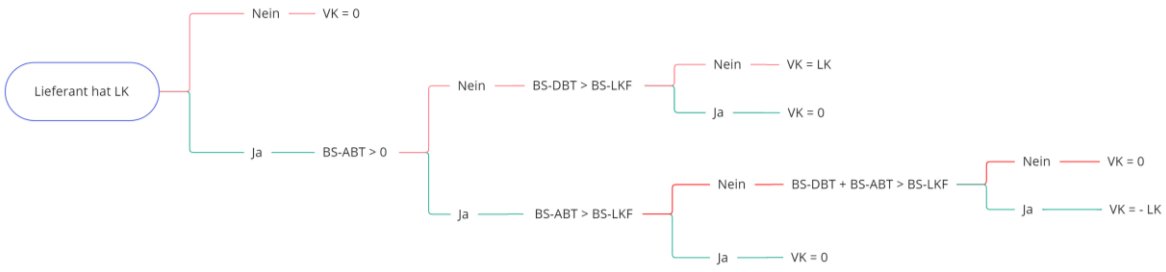

Abbildung 6: Entscheidungsbaum-Diagramm Lieferkosten

Die beiden Fälle werden ebenfalls durch eine bedingte Anweisung abgegriffen.

```
%Lieferkostenfrei durch dieses Bauteil erreicht
if sql_abfrage.lieferkosten > 0 && sql_abfrage.lieferkosten_frei_ab <= ...
   (multi_lief.bestellsumme(c) + multi_lief.bestellsumme_anderer_teile(c)) && ...
   multi_lief.bestellsumme_anderer_teile(c) >= sql_abfrage.lieferkosten_frei_ab ...
   && sql_abfrage. lieferkosten_frei_ab ~= 0


   multi_lief.verrechnung_lieferkosten(c) = -(sql_abfrage.lieferkosten(1));

%Lieferkosten fallen durch dieses Bauteil an
elseif sql_abfrage.lieferkosten > 0 && multi_lief.bestellsumme_anderer_teile(c) ...
       == 0 && sql_abfrage.lieferkosten_frei_ab >= multi_lief.bestellsumme(c)

    multi_lief.verrechnung_lieferkosten(c) = sql_abfrage.lieferkosten(1);

else
    multi_lief.verrechnung_lieferkosten(c) = 0;
end
```

4.4.2

Als letztes wird für die Verrechnungskosten der Rabatt betrachtet. Hierfür ist das entsprechende Entscheidungsbaum-Diagramm in Abbildung 7 dargestellt. Für den Rabatt gilt generell, dass es keine Verrechnungssumme gibt, falls der Lieferant keinen Rabatt anbietet oder die Kriterien nicht erfüllt sind. Ist die Summe erreicht, ab welcher es entsprechend einen Rabatt gibt, gilt zu unterscheiden, ob diese bereits durch die Summe anderer Bauteile erfüllt ist oder erst hierdurch erfüllt wird. Falls diese bereits erreicht ist, setzen sich die Verrechnungskosten aus dem Produkt in Höhe des Rabattes und der Bestellsumme der zu betrachtenden Bauteile zusammen. Sollte die Rabattsumme erst durch die zu betrachtenden Bauteile erfüllt werden, sind die Verrechnungskosten im Betrag höher und setzten sich aus dem Produkt der gesamten Bestellsumme beim Lieferanten und dem Rabatt zusammen. Rabatte fließen in die Berechnung in beiden Fällen negativ ein, da der Realpreis sinkt.

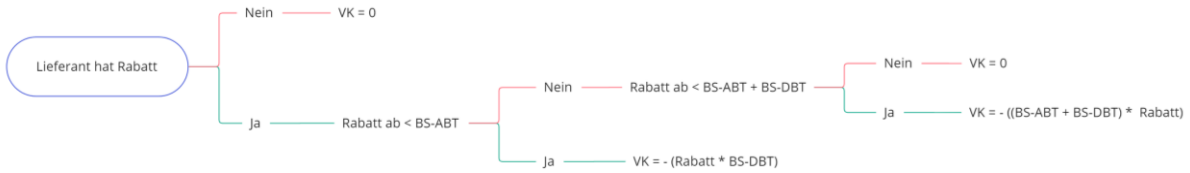

Abbildung 7: Entscheidungsbaum-Diagramm Rabatt

Die Umsetzung in die Programmiersprache erfolgt ebenfalls durch eine bedingte Anweisung. Dabei werden die beiden Fälle für einen vorliegenden Rabatt abgegriffen und unterschieden. Sollte keiner der beiden Fälle erfüllt sein, gibt es auch keinen Rabatt und die Verrechnungskosten aus dem Rabatt sind somit null.

```
%Rabattbedingung bereits erfüllt
if sql_abfrage.rabatt_in_prozent(1) > 0 && sql_abfrage.rabatt_ab(1) <= ...
   multi_lief.bestellsumme_anderer_teile(c)

   multi_lief.verrechnung_rabatt(c) = 0 -(sql_abfrage.rabatt_in_prozent(1) * ...
   multi_lief.bestellsumme(c))/100;

%Rabattbedingung wird dadurch erst erfüllt
elseif sql_abfrage.rabatt_in_prozent(1) > 0 && sql_abfrage.rabatt_ab(1) <= ...
       (multi_lief{c,10} + multi_lief{c,11})

   multi_lief.verrechnung_rabatt(c) = 0 -(sql_abfrage.rabatt_in_prozent(1) * ...
   (multi_lief.bestellsumme(c) + multi_lief.bestellsumme_anderer_teile(c)))/100;

else
   multi_lief.verrechnung_rabatt(c) = 0;
end
```

4.4.3

Nachdem alle Einflussfaktoren für den realen Stückpreis ermittelt sind, kann dieser einfach aus der Summe der Zusatzkosten und des Bauteilpreise dividiert durch die zu bestellende Anzahl bestimmt werden. Dadurch erhält man den tatsächlichen Stückpreis für ein einzelnes Bauteil. Im Programmcode sind dafür folgende Zeilen zuständig.

```
multi_lief{c,16} = (multi_lief.bestellsumme(c) + ...
multi_lief.verrechnung_mindestbestellsumme(c) + ...
multi_lief.verrechnung_rabatt(c) + multi_lief.verrechnung_lieferkosten(c) +
multi_lief.zusatkosten(c) ) / multi_lief.zu_bestellende_anzahl(c);
```

4.4.4

Die Vergleichsbasis für die Entscheidungsfällung ist somit vollständig. Allerdings stellen die oberen Teilabschnitte noch keinen funktionsfähigen Programmcode dar. Die vorhandenen Variablen lassen bereits auf die Verwendung einer Schleife schließen. Diese und die dafür notwendigen Schritte sollen im Folgenden noch genauer betrachtet werden. Für den Abgleich, ob Teile bereits bei dem jeweiligen Lieferanten bestellt werden, wird eine Hilfstabelle erstellt, welche aus der bereits benötigten Lieferantenbestellsummentabelle generiert wird. Diese enthält nur die Lieferanten-IDs, bei denen andere Teile bestellt werden. Mit Hilfe dieser werden die bereits vorliegenden Bestellsummen in die Multilieferantentabelle integriert. Dafür wird die Funktion "find" in Kombination mit einer Bedingung benutzt. In Worten ausgedrückt wird verglichen, ob die jeweilige ID vorkommt und bereits bei dem Lieferanten bestellt wird. Falls ja, wird die Summe übertragen und anders wird eine null eingetragen. Für den Abgleich, ob der Lieferant eine Mindestbestellsumme, Rabatt oder Lieferkosten hat, wird wieder eine Datenbankabfrage benötigt. Der vollständige Programmausschnitt für die Entscheidungsbasis gestaltet sich also wie folgt.

```
multi_lief.bestellsumme = multi_lief{:,7}.*multi_lief{:,6};

search_table = table2array(lieferanten_sum(:,1));

for c = 1:height(multi_lief)
    row_find = find(search_table == multi_lief.lieferant(c));

   if row_find > 0
      multi_lief.bestellsumme_anderer_teile(c) =
lieferanten_sum{row_find,3};
      row_find = 0;
   else
      multi_lief.bestellsumme_anderer_teile(c) = 0;
   end

   sqlvalue = multi_lief.lieferant(c);

   selectquery = ['SELECT * FROM geschmacksachekaffee.lieferant WHERE ...
                  id_lieferant = ', num2str(sqlvalue)];

   sql_abfrage = select(app.conn, selectquery);

   %Verrechnungskosten (siehe Programmcode 4.4.1)
   %Mindestbestellsumme (siehe Programmcode 4.4.2)
   %Lieferkosten (siehe Programmcode 4.4.3)
   %Rabatt (siehe Programmcode 4.4.4)

end
```

4.4.5

Nun gilt es noch, das günstigste Bauteil von den zur Auswahl stehenden zu identifizieren und die Kalkulationsliste zurückzuspielen. Dazu werden die MATLAB Funktionen "sortrows" und "unique" verwendet. Mit ersterer wird die Tabelle nach der Bauteil-ID und dem Realpreis sortiert. Dadurch wird erreicht, dass für jede Bauteil-ID in der ersten Zeile, in der diese vorkommt, der günstigere der Realpreise steht. Mit der zweiten Funktion kann nun der Reihenindex identifiziert werden, in dem die Bauteil-ID das erste Mal auftritt. Die entsprechenden Zeilen können dann aus der Liste mit den benötigten Spalten extrahiert werden. Dadurch haben diese Bauteile wieder die gleiche Tabellenstruktur wie die

Kalkulationsliste und können einfach in diese eingefügt werden. In MATLAB sieht die Umsetzung nun wie folgt aus.

```
multi_lief = sortrows(multi_lief, [1 16]);
[~,uniquePartIndex] = unique(multi_lief(:,1),'first');
multi_lief_guenstigste = multi_lief(uniquePartIndex,1:10);

%Tabellen zusammenführen
kalkliste = [kalkliste; multi_lief_guenstigste];
```

4.4.6

Wichtig ist an dieser Stelle noch zu erwähnen, dass die Preiszuordnung für die "multi_lief" Tabelle nur stattfindet, wenn diese Einträge enthält. Sollte also kein Bauteil mindestens zwei Lieferanten haben, ist die Tabelle ein leeres Element und es passiert durch die zuvor beschriebenen Programmabschnitte für Zusatzkosten nichts.

4.5 Zusatzkosten

Die Zusatzkosten bezeichnen alle zusätzlichen Kosten, welche nicht direkt auf ein Einzelteil zurückzuführen sind. Die erste Position, die bei jeder Maschine anfällt, sind die Lohnkosten. Deshalb werden diese beim Starten der Kalkulation in eine eigene Tabelle importiert. Diese Tabelle wird im Laufe des Programms um alle anfallenden Zusatzkosten ergänzt. Die Lohnkosten werden für die Weiterverarbeitung mit der Losgröße multipliziert.

```
zusatzkosten = select(app.conn,'SELECT * FROM ...
                               geschmacksachekaffee.zusatzkosten');

zusatzkosten = zusatzkosten(1, [2 3]);
zusatzkosten.betrag(1) = zusatzkosten.betrag(1) * app.lot;
```

4.5.1

Weitere Zusatzkosten können durch Mindestbestellmengen entstehen. Dabei müssen von einem Bauteil eine bestimmte Summe beim zugehörigen Lieferanten bestellt werden. Dabei gilt es zu unterscheiden, ob es sich um Normteile oder Konstruktionsbauteile handelt. Für Normteile gibt es nur bestimmte Packeinheiten. Das bedeutet, bei einer benötigten Stückmenge von 51 und einer Mindestbestellmenge von 50 Stück müssen trotzdem 49 Einheiten zu viel bestellt werden. Die Problematik zur Mindestbestellmenge ist bereits bei der Preisfindung in Kapitel 6 beschrieben. Für die Zusatzkosten wird in einer Schleife die Bedingung geprüft, ob die zu bestellende Anzahl höher als die benötige ist. Falls ja, werden die zu viel bestellten Stück mit dem Stückpreis multipliziert und mit einer einschlägigen Bezeichnung, welche aus der Anzahl und Bauteilbeschreibung besteht, in die Zusatzkostentabelle übernommen. Der Abgleich kann erst stattfinden, wenn die Entscheidung bzgl. der Teile mit mehreren Lieferanten getroffen wurde und das jeweils günstigste in die Kalkulationsliste zurückgeführt wurde.

```
for c = 1:height(kalkliste)

 if kalkliste {c,6} > kalkliste{c,2}
   rowc = height(zusatzkosten)+1;
   zusatzkosten{rowc, 1}= (kalkliste {c,6}-kalkliste{c,2})*kalkliste{c,7};

   zusatzkosten{rowc, 2}= cellstr(strjoin(['Aufpreis Mindestbestellmenge...
                          von Bauteil:', kalkliste{c,4}, string((...
                          kalkliste {c,6} - kalkliste{c,2})), 'Stück']));
 end
end
```

4.5.2

Weitere Zusatzkosten sind Kostenvorteile durch Rabatte, Aufpreise durch Mindestbestellsummen und Lieferkosten. Für den Abgleich und die Berechnungen werden die Bestellsummen pro Lieferant benötigt. Diese können, nachdem die Bauteile mit mehreren möglichen Lieferanten in die Kalkulationsliste eingefügt wurden, erzeugt werden. Die Lieferantensumme wird unter Verwendung der Funktion "groupsummary" in folgenden zwei Zeilen umgesetzt. Dabei werden aus der Kalkulationstabelle nur die Spalte mit der Lieferanten-ID und der Bestellsumme (Summe aus Stückzahl und Stückpreis) verwendet.

```
lieferanten_sum = kalkliste(:,[8 10]);
lieferanten_sum = groupsummary(lieferanten_sum, "lieferant", 'sum');
```
4.5.3

Die Prüfung auf Rabatte, Mindestbestellsummen und Lieferkosten wird in einer Schleife umgesetzt. Diese Schleife läuft so viele Durchgänge, wie es unterschiedliche Lieferanten in der dafür generierten Summentabelle gibt. In der Schleife sind zwei Bedingungen integriert. Eine gleicht ab, ob der Lieferant eine Mindestbestellsumme hat, die nicht erreicht wird. Falls dies nicht der Fall ist, werden die Kriterien für einen Rabatt geprüft. Die zweite Bedingung bestimmt die Lieferkosten. Die benötigten Daten werden wieder durch eine SQL-Abfrage mit der Lieferanten-ID als Suchkriterien in MATLAB importiert.

```
for c = 1:height(lieferanten_sum)
    sqlvalue = lieferanten_sum{c,1};

    selectquery = ['SELECT * FROM geschmacksachekaffee.lieferant WHERE ...
                   id_lieferant = ', num2str(sqlvalue)];

    sql_abfrage = select(app.conn, selectquery);

    rowc = height(zusatzkosten)+1;

    %Mindestbestellsumme
    if sql_abfrage.mindestbestellsumme(1) > lieferanten_sum{c, 3}

      zusatzkosten{rowc, 1} = sql_abfrage.mindestbestellsumme(1) - ...
                              lieferanten_sum.sum_bestellsumme(c);

      zusatzkosten{rowc, 2} = cellstr(strjoin(['Aufpreis Mindestbestellsumme...

                      von Lieferant:' , sql_abfrage.lieferant_name(1)]));
    %Rabatt
    elseif sql_abfrage.rabatt_in_prozent(1) > 0 && sql_abfrage.rabatt_ab(1) ...
                                            <= lieferanten_sum{c, 3}
      zusatzkosten{rowc, 1} = -(sql_abfrage.rabatt_in_prozent(1) * ...
                                lieferanten_sum{c, 3})/100;
      zusatzkosten{rowc, 2} = cellstr(strjoin(['Rabatt von ...
                                Lieferant:', sql_abfrage.lieferant_name(1)]));

     End

     %Lieferkostenbestimmung
     if sql_abfrage.lieferkosten > 0 && sql_abfrage.lieferkosten_frei_ab >= ...
               lieferanten_sum{c, 3} && sql_abfrage.lieferkosten_frei_ab ~= 0

        lieferkosten = lieferkosten + sql_abfrage.lieferkosten(1);

     end
end
```
4.5.4

Die entstehenden Zusatzkosten für Rabatt und Mindestbestellsummen werden direkt in die Zusatzkostentabelle mit einer zusammengesetzten Bezeichnung übertragen. Die Lieferkosten werden in der Schleife nur addiert, da diese als eine Gesamtsumme ausgegeben werden.

Die letzte Art der Zusatzkosten entsteht durch bauteilspezifische Sonderkosten, wie Rüst- und Einrichtungskosten. Dabei gilt zu unterscheiden, ob diese einmalig bei der ersten Bestellung oder bei jeder Bestellung fällig werden. Die Abfrage dieser Zusatzkosten findet über die jeweilige Teilenummer statt. Da die Abfrage für jedes einzelne Teil in der Kalkulationsliste stattfinden muss, befindet sich der Programmcode dafür in der selben Schleife wie die Mindestbestellmengenbestimmung. Dadurch wird ein extra Durchlauf durch eine eigene Schleife eingespart. Da es sich bei der Abfragevariable wieder um die Teilenummer handelt, muss diese wie beschrieben in Singlequotes ohne Leerzeichen gesetzt werden.

```
for c = 1:height(kalkliste)
```
4.5.5

```
  sqlvalue = strjoin(['''',kalkliste{c,3} ,'''']);
  sqlvalue = sqlvalue(~isspace(sqlvalue));

  selectquery = ['SELECT * FROM geschmacksachekaffee.zusatzkosten WHERE ...
                  teile_nr = ', sqlvalue];

  sql_abfrage = select(app.conn, selectquery);

  if height(sql_abfrage) > 0 && sql_abfrage.erledigt_am(1) == 0

     bauteil_zusatzdaten = sql_abfrage(:,[3 2]);
     zusatzkosten = [zusatzkosten; bauteil_zusatzdaten];

  end

   %Mindestbestellmengen vgl. Programmcode 4.4.2

end
```

Falls über die SQL-Abfrage Zusatzdaten für das entsprechende Bauteil festgestellt werden, wird abgeglichen, ob diese erledigt wurden. Das Erledigungsdatum ist nur für Einmalzahlungen hinterlegt, die bereits getätigt wurden. Sollte dieses Datum null sein, werden aus der Zusatzkostentabelle in der Datenbank die Beschreibung und die Summe in die Zusatzkostentabelle in MATLAB übertragen.

4.6 Gesamtpreis und Ausgabe

Für die Vorbereitung der Ausgabe müssen zunächst die ermittelten Preise aus der Kalkulationsliste zurück in die Stückliste gespielt werden. Strukturbedingt enthält die Kalkulationsliste jedes Bauteil nur einmal, während die Stückliste die Bauteile in der verwendeten Position und Anzahl enthält. Für die Zusammenführung der Tabelle wird ein Join verwendet. Dafür wird eine rechte und eine linke Tabelle benötigt. Die linke Tabelle ist die Stückliste, in der die Preise eingefügt werden sollen. Die rechte Tabelle wird aus der Kalkliste nur mit den Preisen und den Teile-IDs erstellt. Der Rest wird nicht benötigt. Als Schlüssel für die Zusammenführung dient die Teile-ID. Der Stückpreis muss dann erneut mit der zugehörigen Stückzahl multipliziert werden, um die Positionssumme zu erhalten.

```
righttable = kalkliste(:, {'id_teil' 'stueckpreis'});
stueckliste = join(stueckliste, righttable, 'Keys', 'id_teil');

%Summenpreis bestimmen
stueckliste.stueckpreis = stueckliste.stueckpreis.*double(stueckliste.anzahl);
```

4.6.1

Die Stückliste enthält nun die strukturierten Preise für jedes Teil in der verwendeten Stückzahl in der jeweiligen Baugruppe. Um die Preise pro Baugruppe auszugeben, müssen diese noch zusammengefasst werden. Hierfür wird wieder die "groupsummery" Funktion angewandt. Die entstehende Tabelle enthält dann nur noch die Baugruppen-IDs mit der zugehörigen Summe.

```
ausgabe = groupsummary(stueckliste, "id_baugruppe", 'sum');
```

4.6.2

Die Zwischensumme wird aus der Summe der Ausgabetabelle berechnet. Für die Zusatzkosten wird die Summe aus der Zusatzkostentabelle gebildet und durch die gewählte Losgröße geteilt, um diese anteilig zu bestimmten. Beide Variablen werden als Array erstellt, um das Ausgabeformat der GUI bzgl. der drei Spalten zu treffen. Die mittlere Spalte ist hierbei leer.

```
%Zwischensumme bestimmen
zwischensumme{1,1} = 'Zwischensumme';
zwischensumme{1,2} = ' ';
zwischensumme{1,3} = sum(ausgabe.sum_stueckpreis);

%Zusatzkosten anteilig bestimmen
```

4.6.3

```
summezusatzkosten{1,1} = 'Anteilige Zusatzkosten';
summezusatzkosten{1,2} = ' ';
summezusatzkosten{1,3} = (sum(zusatzkosten.betrag)+lieferkosten)/app.lot;
```

Der Gesamtpreis ermittelt sich ebenfalls über die Summe der Ausgabetabelle und die anteiligen Zusatzkosten. Hier wird zusätzlich noch die Beschreibung eingefügt, um welche Maschine es sich handelt sowie die zugehörige Losgröße. Die Daten werden hierfür aus den Auswahlmenüs bezogen.

```
gesamtpreis{1,1} = 'Gesamtpreis';

gesamtpreis{1,2} = append('pro ',app.MaschineDropDown.Value, ' ...
                          in Stückzahl ', num2str(app.lot));

gesamtpreis{1,3}= sum(ausgabe.sum_stueckpreis) + ...
                      sum(zusatzkosten.betrag)/app.lot;
```

4.6.4

Zu den Preisen in der Ausgabe fehlen noch die Bezeichnungen des jeweiligen Moduls und der Baugruppe, damit diese zugeordnet werden können. Hierfür wird über den jeweiligen Identifier die Beschreibung aus der Datenbank abgefragt. Dazu werden zunächst zwei Spalten hinzugefügt, in denen die Bezeichnung dann über eine Schleife eingefügt wird. Die Schleife muss für jedes Element der Ausgabetabelle durchgeführt werden.

```
b = width(ausgabe)+1;
m = width(ausgabe)+2;
for c = 1:height(ausgabe)

    sqlvalue = ausgabe{c,1};

    %Baugruppen Beschreibung
    selectquery = ['SELECT * FROM geschmacksachekaffee.baugruppe WHERE ...
                   id_baugruppe = ', num2str(sqlvalue)];

    sql_abfrage = select(app.conn, selectquery);
    ausgabe(c, b) = sql_abfrage.beschreibung_baugruppe;

    %Modul Beschreibung
    sqlvalue = sql_abfrage.id_modul(1);

    selectquery = ['SELECT * FROM geschmacksachekaffee.modul WHERE ...
                   id_modul = ', num2str(sqlvalue)];

    sql_abfrage = select(app.conn, selectquery);
    ausgabe{c, m} = sql_abfrage.beschreibung_modul;

end
```

4.6.5

Nachdem die Bezeichnungen eingefügt sind, können alle nicht mehr benötigten Spalten aus der Ausgabetabelle entfernt werden. Dazu werden die zwei hinzugefügten Spalten zunächst umbenannt und die Tabelle in ein Array umgewandelt, damit diese an die UI-Table übertragen werden kann.

```
ausgabe.Properties.VariableNames([b m]) = {'baugruppe' 'modul'};
ausgabe = table2cell(ausgabe(:, {'modul' 'baugruppe' 'sum_stueckpreis'}));
```

4.6.6

Die Zusatzkosten müssen vor dem Zusammenfügen ebenfalls in ein Array umgewandelt und die Beschreibung eingefügt werden. Zudem muss noch ein Array für die Lieferkosten im gleichen Stil erzeugt werden.

```
%Zusatzkosten benennen und umwandeln
zusatzkosten{:,'modul'} =  'Zusatzkosten';
```

4.6.7

```
zusatzkosten = table2cell(zusatzkosten(:,{'modul' ...
                          'beschreibung_zusatzkosten' 'betrag'}));

%Lieferkosten Array erstellen
lieferkosten_ag{1,1} = 'Lieferkosten';
lieferkosten_ag{1,2} = ' ';
lieferkosten_ag{1,3} = lieferkosten;
```

In den folgenden Programmzeilen wird die Ausgabe final zusammengeführt. Dafür wird noch eine Leerzeile in der Tabelle benötigt, welche durch ein Array mit leeren Zeichenfolgen umgesetzt wird.

```
%Leerzeile erzeugen
blank{1,1} = ' ';
blank{1,2} = ' ';
blank{1,3} = ' ';

%Ausgabe kombinieren
ausgabe = [ausgabe; blank; zwischensumme; blank; zusatzkosten; ...
     lieferkosten_ag; blank; summezusatzkosten; blank; blank;
gesamtpreis];
```

4.6.8

Die Ausgabetabelle besitzt nun die vollständigen Informationen. Allerdings sind Modulbeschreibungen noch doppelt vorhanden, da mehrere Baugruppen zu einem Modul gehören. Für die Übersichtlichkeit soll nur die erste Nennung des Moduls stehen bleiben. Dies wird durch folgende Programmcodezeilen mit der Verwendung der "unique" Funktion realisiert.

```
uniqStr = unique(ausgabe(:,1));

for c = 1:length(uniqStr)
    idx = strmatch(uniqStr{c}, ausgabe(:,1), 'exact');
    for z = 2:length(idx)
        ausgabe{idx(z),1} = ' ';
    end
end
```

4.6.9

Dabei wird zunächst jede Bezeichnung nur genau einmal in ein Array geladen. Für dieses Array werden dann in einer Schleife die zugehörigen Indexe ermittelt, in welchen dieses vorkommt. Mittels einer zweiten Schleife werden dann all jene, welche ab dem zweiten Mal vorkommen, gelöscht bzw. durch einen leeren Wert ersetzt.
Nun kann die generierte Ausgabetabelle final an die dazu bestimmte UI-Tabelle übergeben werden. Für die UI-Tabelle muss noch die Preisspalte in den Datentyp "blank" umgewandelt werden, damit die Preise mit zwei Nachkommastellen ausgegeben werden. Außerdem werden die Spaltenbreiten für eine bessere Darstellung fest hinterlegt.

```
app.PreisAusgabe.ColumnFormat(3) = {'bank'};
app.PreisAusgabe.ColumnWidth = {175, 'auto', 100};
app.PreisAusgabe.Data = ausgabe;
```

4.6.10

4.7 Bestelllistengenerierung

Die Bestelllisten werden aus der Kalkulationsliste nach erfolgreicher Kalkulierung übergeben. Der gesamte im Folgenden beschriebene Programmcode, ausgenommen der Übergabe der Kalkulationsliste, befindet sich hinter dem Callback des Buttons Bestelllisten. Hierzu werden die benötigten Daten über untenstehende Programmcodezeilen übertragen. Für die Bestellung bei einem Lieferanten werden im Wesentlichen die Anzahl und die Daten zur Bauteilidentifikation sprich Teilenummer, Beschreibung und Bestellnummer benötigt. Bei der Generierung einer Bestellliste werden zusätzlich die Kontaktdaten des jeweiligen Lieferanten benötigt.

```
app.bestelllisten = kalkliste(:,{'lieferant' 'zu_bestellende_anzahl'
'teile_nr' 'beschreibung_teil' 'bestellnummer' });
```

4.7.1

Damit der Bestelllisten-Button erst betätigt werden kann, sobald die Kalkulation durchgeführt wurde, wird der auszuführende Code in eine Bedingung eingebettet. Diese frägt die Höhe der Bestelllisten ab und wird wie folgt realisiert.

```
if height(app.bestelllisten) < 1
      app.BestelllistenerzeugenButton.BackgroundColor = 'r';
else
      %Bestelllisten Generierung (siehe Programmcode 4.7.3 bis 4.7.13)
end
```

4.7.2

4.7.1 Aufteilen

Bei der Variable "app.bestelllisten" handelt es sich um eine Tabelle, welche alle zu bestellenden Bauteile enthält. Um diese aufzuteilen, muss die entsprechende Gruppierung festgelegt werden. Die Listen sollen nach Lieferanten aufgetrennt werden. Dies wird durch die Funktionen "findgroups" und "splitapply" in MATLAB umgesetzt.

```
gruppierung = findgroups(app.bestelllisten.lieferant);
T_split = splitapply( @(varargin) varargin, app.bestelllisten,
gruppierung);
subTables = cell(size(T_split, 1),2);
```

4.7.3

Da in jeder Bestellliste zu den Bauteildaten noch die Lieferantendaten hinzugefügt werden sollen, findet die Trennung in einem Array statt. Das Array umfasst so viele Zeilen, wie es unterschiedliche Lieferanten gibt und zwei Spalten. Die erste Spalte enthält die Tabellen mit den Bauteildaten und die zweite soll später die Lieferantendaten enthalten. Diese Aufteilung wird durch folgenden Abschnitt im Programmcode umgesetzt:

```
for i = 1:size(T_split, 1)
      subTables{i} = table(T_split{i, :}, 'VariableNames', ...
      app.bestelllisten.Properties.VariableNames);
end
```

4.7.4

4.7.2 Lieferantendaten einfügen

Für jede der erstellten Bestelllisten müssen noch die entsprechenden Kopfzeilen eingefügt werden, welche alle nötigen Informationen bzgl. des Lieferanten enthalten. Dazu wird eine Schleife erstellt, welche so oft durchläuft, wie es aufgeteilte Tabellen bzw. unterschiedliche Lieferanten gibt. Für die SQL-Abfrage wird die Lieferanten-ID verwendet, welche jeweils in der ersten Zelle der zugehörigen Tabelle im Array steht. Das Schema des Headers ist in Tabelle 9 dargestellt. Die leeren Zeilen sollen dabei in der Bestellliste als Abstandszeile dienen, um die Daten übersichtlicher darzustellen. Der Programmcode mit der zugehörigen Schleife und die SQL-Variable sind in den folgenden Zeilen realisiert. Dabei wird über die Lieferanten-ID die Datenzeile vom zugehörigen Lieferanten aus der Tabelle Lieferant in MATLAB importiert.

Tabelle 10: Kopfzeilen für Lieferantendaten

Lieferantname	
Anschrift	
Anschrift	
Anschrift	
Internetadresse:	Text
Ansprechpartner:	Text
Emailadresse:	Text
Telefonnummer:	Text

```
for c = 1:height(subTables)

    sqlvalue = subTables{c,1}{1,1};

    selectquery = ['SELECT * FROM geschmacksachekaffee.lieferant ...
                   WHERE id_lieferant = ', num2str(sqlvalue)];

    sql_abfrage = select(app.conn, selectquery);

    %Daten Extrahierung (siehe Programmcode 4.7.6 bis 4.7.9)

end
```

4.7.5

Der Lieferantname und die Anschrift werden jeweils in die zugehörige Tabelle aus der SQL-Abfrage eingefügt. Danach wird die Leerzeile durch einen leeren String eingefügt.

```
subTables{c,2}(1,1) = sql_abfrage.adresszeile_1;
subTables{c,2}(1,2) = sql_abfrage.adresszeile_2;
subTables{c,2}(1,3) = sql_abfrage.adresszeile_3;
subTables{c,2}(1,4) = sql_abfrage.adresszeile_4;
subTables{c,2}(1,5) = {''};
```

4.7.6

Für die anderen Daten, also Internetadresse, Ansprechpartner, E-Mailadresse und Telefonnummer soll die entsprechende Bezeichnung vor den Wert geschrieben werden, damit dieser zugeordnet werden kann. Allerdings hat nicht jeder Lieferant einen Ansprechpartner. Es ist außerdem nicht auszuschließen, dass die anderen Daten ebenfalls nicht vorhanden sind oder für die Bestellung keine Rolle spielen und deshalb nicht gepflegt werden. Deshalb soll die jeweilige Beschreibung nur eingefügt werden, wenn auch ein Wert dafür vorhanden ist. Dies wird durch eine jeweilige bedingte Anweisung realisiert, die prüft, ob die Eintragung länger als zwei Zeichen ist. Zwei Zeichen aus dem Grund, dass womöglich ein Leerschritt eingetragen sein kann und ein Zeichen mehr um auf Nummer sicher zu gehen. Es gibt keine validen Daten für die Variablen, die eine Länge darunter besitzen würden. Zwischen der Internetadresse und dem Ansprechpartner wird erneut eine Leerzeile eingefügt. Der Programmcode ist in den folgenden Zeilen beschrieben.

```
%Internetadresse einfügen
if strlength(sql_abfrage.internetadresse(1)) > 2
   subTables{c,2}(1,6) = {'Internetadresse:'};
   subTables{c,2}(2,6) = sql_abfrage.internetadresse;
end

%Leerzeile einfügen
subTables{c,2}(1,7) = {''};

%Ansprechpartner einfügen
if strlength(sql_abfrage.ansprechpartner(1)) > 2
   subTables{c,2}(1,8) = {'Ansprechpartner:'};
   subTables{c,2}(2,8) = sql_abfrage.ansprechpartner;
end

%Emailadresse einfügen
if strlength(sql_abfrage.email(1)) > 2
   subTables{c,2}(1,9) = {'Emailadresse:'};
   subTables{c,2}(2,9) = sql_abfrage.email;
end
%Telefonnummer einfügen
if strlength(sql_abfrage.telefon(1)) > 2
   subTables{c,2}(1,10) = {'Telefonnummer:'};
   subTables{c,2}(2,10) = sql_abfrage.telefon;
end
```

4.7.7

Für die finale und korrekte Ausgabe der Bestelllisten müssen die Daten zunächst noch etwas bereinigt werden. In den Tabellen mit den Bauteilen steht bis zu diesem Zeitpunkt noch die zugehörige Lieferanten-

ID. Diese wurde im vorherigen Abschnitt noch benötigt, ist allerdings für den Lieferanten und die Bestellung bedeutungslos. Deshalb wird diese bereinigt und im selben Schritt werden die Spaltenbezeichnungen auch in normaler Schreibweise angepasst.

```
subTables{c, 1}(:,1) = [];
subTables{c, 1}.Properties.VariableNames = ["Anzahl" "Teilenummer" ...
                                            "Beschreibung" "Bestellnummer"];
```
4.7.8

Außerdem befindet sich die Teilenummer noch in der Datenbankschreibweise, sprich die Zuordnung für die Maschine befindet sich an erster Stelle der Zeichenfolge. Diese soll noch in der Anwendungsform ausgegeben werden. Dazu wird der String in ein Array nach dem Bindestrich aufgeteilt und die erste Position entfernt. Danach wird dieser wieder mit einem Bindestrich zusammengesetzt. Ergebnis ist die Teilenummer ohne die führende Ziffer aus der Datenbank. Das Ganze muss in einer Schleife passieren, da es für jeden Wert der jeweiligen Tabelle geschehen muss, die durch die äußere Schleife (s. oben) bearbeitet wird. In MATLAB wird dies in den folgenden Programmzeilen durchgeführt. Dabei muss für den Zähler der Schleife ein anderer Wert als für die äußere Schleife verwendet werden, sonst entsteht ein kritischer Fehler.

```
for z = 1:height(subTables{c,1}(:,2))
    teile_nr = split(subTables{c,1}{z,2}, '-');
    teile_nr(1,:) = [];
    subTables{c,1}{z,2} = cellstr(strjoin(teile_nr, '-'));
end
```
4.7.9

4.7.3 Speichern

Das Speichern der Bestelllisten erfolgt nach der Erstellung automatisch. Der Dateiname setzt sich hierfür aus dem Wort Bestelllisten, dem variablen Zusatz der zugehörigen Maschine und dem tagesaktuellen Datum fest zusammen. Die Bezeichnung der Maschine wird hierfür aus dem Dropdown genommen.

```
filename = append(strjoin({'Bestelllisten',app.MaschineDropDown.Value, ...
num2str(yyyymmdd(datetime("today")))}, '_'), '.xlsx');
```
4.7.10

Zum Dateinamen ist ebenfalls der Speicherort wichtig. Hierfür ist das Downloadverzeichnis des Benutzers vorgesehen und setzt sich folgendermaßen zusammen.

```
filname_plus_pathname = [fullfile(getenv('USERPROFILE'), 'Downloads') ...
                         '\' filename];
```
4.7.11

Als letzter Schritt werden noch die jeweils zusammengehörenden Bauteil- und Lieferantendaten in ein Excelarbeitsblatt zusammengesetzt. Jeder Lieferant bekommt ein eigenes Arbeitsblatt, aber es entsteht nur eine Datei. In MATLAB wird hierfür die "write" Funktion benutzt, welche in eine Schleife gesetzt wird. Die Schleife läuft so viele Durchgänge, wie es unterschiedliche Lieferanten gibt.

```
for c = 1:height(subTables)
      writecell(subTables{c,2}', filname_plus_pathname,'Sheet',c)
      writetable(subTables{c,1}, filname_plus_pathname,'Sheet',c, ...
                 'Range','A13')
end
```
4.7.12

Damit der Benutzer ein interaktives Feedback erhält, ob das Erstellen der Bestelllisten funktioniert hat, wird die generierte Excel-Datei automatisch geöffnet. Dies wird durch die Funktion "system" und den zugehörigen Speicherpfad umgesetzt.

```
system(filname_plus_pathname)
```
4.7.13

5 Datenbank Editor

Der Datenbank Editor ermöglicht es Benutzern, die Daten innerhalb der MySQl-Datenbank direkt einzusehen, intuitiv zu bearbeiten und gewissen Datensätze neu hinzuzufügen. Dies ist mit dem Tool Datenbank Editor über die MATLAB-GUI möglich, ohne dass eine lokale MySQL Installation vorliegen muss. Dazu greift dieser direkt auf die vorhandenen Tabellen in der Datenbank zu und importiert diese nach der gewünschten Auswahl in die Übersicht der GUI.

5.1 GUI-Datenbank-Editor

Die MATLAB GUI für den Datenbank Editor besitzt im oberen Teil ein Dropdown-Menü, mit dem die zu bearbeitende Tabelle ausgewählt werden kann. Zusätzlich kann mit dem Eingabefeld und der zugehörigen Schaltfläche die ausgewählte Tabelle nach einem eingegebenen Wert durchsucht werden. Die jeweilige Auswahl, ob gefiltert oder nicht, erscheint mittig in der großen Ausgabetabelle, welche den Namen Serverdatentabelle erhält. Im unteren Bereich der App befindet sich eine zusätzliche Tabelle, mit der ein ausgewählter Datensatz manipuliert werden kann. Diese Tabelle wird im Folgenden als Änderungstabelle bezeichnet. Nebenstehend findet man den Speichern-Button, um die Änderung in die Datenbank aufzunehmen. Zusätzlich gibt es eine weitere Schalfläche, mit der sich für vorgemerkte Tabellen Zeilen hinzufügen lassen, wenn Daten nicht bearbeitet, sondern neu eingetragen werden sollen. Diese Schaltfläche ist für nicht dafür vorgesehene Auswahloptionen des Dropdown-Menüs unsichtbar geschaltet. Die GUI ist in Abbildung 8 dargestellt.

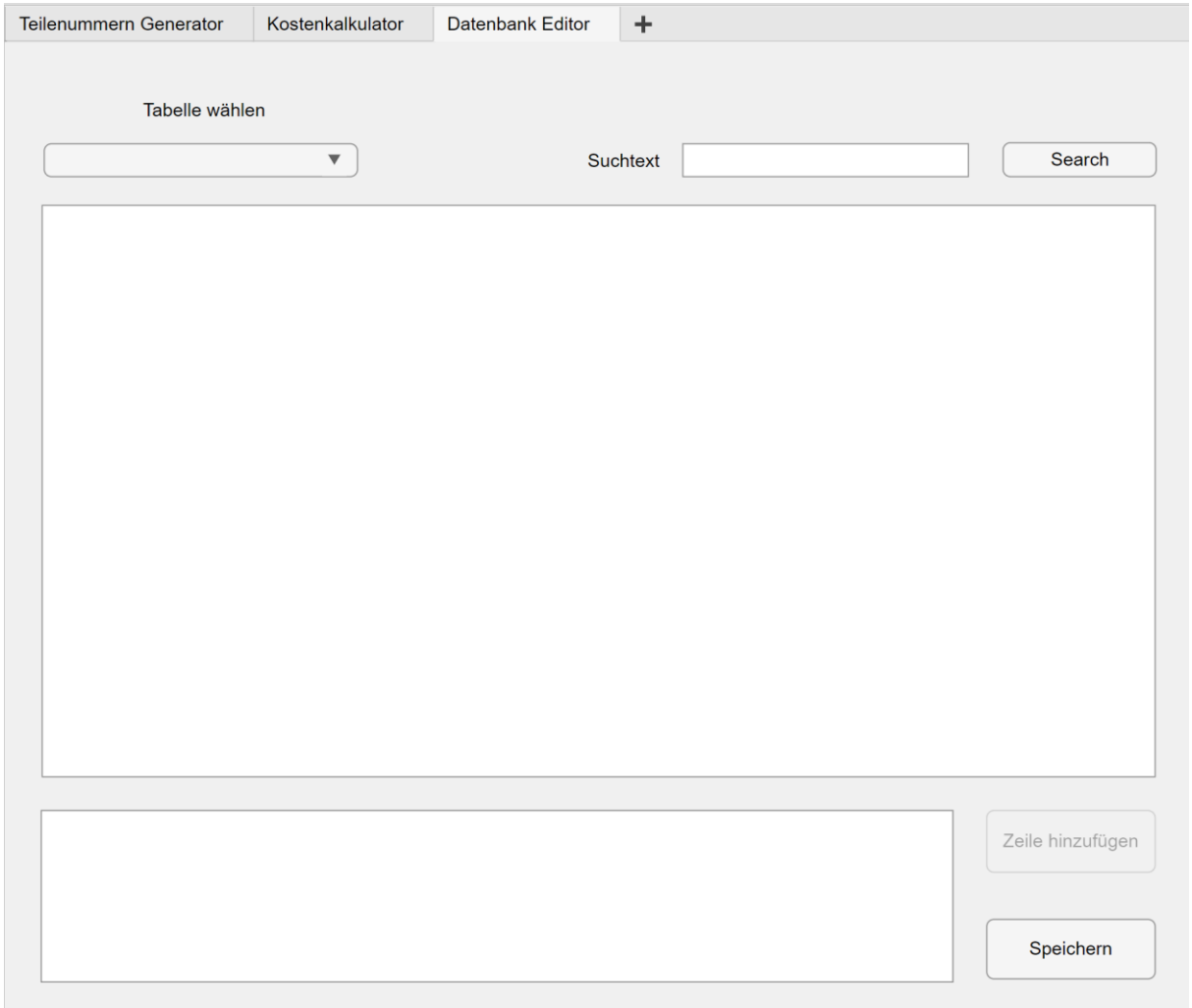

Abbildung 8: GUI-Datenbank-Editor

5.2 Importieren der Datenbanktabellen

Durch einen Callback, der auf den Wechsel des aktuellen Tabs reagiert, wird im Programmcode abgeglichen, ob das Tab des Datenbankeditors ausgewählt wurde. Sobald dieses angeklickt wurde, werden die vorhanden Tabellen in der Datenbank abgefragt und die Auswahl an das Dropdown-Menü übergeben. Für den Vergleich wird die Funktion "strcmpi" verwendet. Um die vorhandenen Tabellen in einer Datenbank über MATLAB abzufragen wird die Funktion "sqlfind" benötigt. Durch diese erhält man alle Tabellen mit den zugehörigen Bezeichnungen, Größen und Dateitypen zusammengefasst in einer Tabelle. Für die Auswahlmöglichkeit muss die Standardauswahl mit aufgenommen werden, bevor diese übergeben werden können. Grund hierfür ist, dass Dropdown-Menüs nur auf die Änderung der ausgewählten Variable reagieren und somit eine Vorauswahl getroffen werden muss.

```
if strcmpi(selectedTab.Title,'Datenbank Editor')

   data = sqlfind(app.conn,"",'Catalog',"geschmacksachekaffee");

   dropDownTabellenwahl = data(:,3);

   dropDownTabellenwahl = [{'Bitte Tabelle wählen'}, ...
                           table2cell(dropDownTabellenwahl)'];

   app.TabellewhlenDropDown.Items = dropDownTabellenwahl;

end
```

5.2.1

5.3 Dropdown Auswahl

Als nächster Schritt wird die zu bearbeitende Tabelle im Dropdown-Menü ausgewählt. Hierfür gilt es zu beachten, dass die Eingabeaufforderung keine valide Auswahl darstellt und abgegriffen werden muss. Dafür wird eine bedingte Anweisung erstellt. Ebenfalls werden die Daten der Änderungstabelle gelöscht bzw. zu einem leeren Array gesetzt. Dadurch wird die Tabelle geleert, falls die Auswahl durch den Benutzer geändert wird. Durch den jeweils ausgewählten Wert wird anschließend die zugehörige Tabelle aus der Datenbank abgefragt. Danach wird die Serverdatentabelle vorbereitet. Dazu werden die Spaltennamen aus der Datenbanktabelle an die UI-Tabelle übergeben. Die Spaltennamen werden auch direkt an die Änderungstabelle übergeben.

```
value = app.TabellewhlenDropDown.Value;
app.Aenderungstabelle.Data = {};

if strcmpi(app.TabellewhlenDropDown.Value,'Bitte Tabelle wählen') == 0

  readtable = append('geschmacksachekaffee.', value);

  sql_abfrage = sqlread(app.conn, readtable);

  %Optionen für Ausgabetabelle

  app.ServerDaten.ColumnName = sql_abfrage.Properties.VariableNames;

  app.Aenderungstabelle.ColumnName = sql_abfrage.Properties.VariableNames;

  app.ServerDaten.Data =  sql_abfrage;

  app.ServerDaten.SelectionType = 'row';

  app.ServerDaten.RowName = 'numbered';

end
```

5.2.1

Für manche Tabellen ist es vorgesehen, durch den Datenbank Editor Zeilen hinzuzufügen. Durch eine bedingte Anweisung wird der Button hierfür auf sichtbar oder unsichtbar geschalten. Dies wird durch die Funktion "strcmpi" erreicht, in dem der Name der ausgewählten Tabelle mit den dafür vorgesehenen Tabellen verglichen wird. Zusätzlich wird in selben Schritt die maximale ID aus den Serverdaten zwischengespeichert, um mit dieser später die Zeilen hinzufügen zu können.

```
if strcmpi(value,'stueckliste_glasboiler') || strcmpi(value, ...
     'zusatzkosten') || strcmpi(value,'lieferant') || ...
     strcmpi(value,'modul') || strcmpi(value,'stueckliste_labormaschine')

     app.ZeilehinzufgenButton.Visible = 'on';
     app.max_id = max(app.ServerDaten.Data{:,1});

else
    app.ZeilehinzufgenButton.Visible = 'off';
end
```

5.2.2

5.4 Tabellen

Nachdem die Auswahl für die zu bearbeitende Tabelle über das entsprechende Dropdown-Menü getroffen und die Daten in die Serverdatentabelle importiert wurden, wird durch das Auswählen einer beliebigen Zelle die ganze Zeile bzw. der Datensatz in die Änderungstabelle übergeben. Dort können die Daten entsprechend angepasst werden. Die folgenden Programmzeilen zeigen diesen Übertrag. Zusätzlich wird die Variable Funktion für das Speichern zu eins gesetzt.

```
selection = app.ServerDaten.Selection;
app.Aenderungstabelle.Data = app.ServerDaten.Data(selection,:);

app.funktion = 1;
```

5.3.1

Für das Bearbeiten der Daten gilt es zu beachten, dass nicht jeder Eintrag in einer Datenbank für eine Änderung vorgesehen ist. Primär- und Fremdschlüssel dürfen nicht ohne weiteres verwendet werden, da der Zusammenhang sonst verloren geht. Deshalb werden im Folgenden über eine bedingte Anweisung mit Hilfenahme der Funktion "strcmpi", die jeweiligen Spalten zur Bearbeitung frei gegeben oder gesperrt.

```
if strcmpi(app.TabellewhlenDropDown.Value,'stueckliste_glasboiler') || ...
       strcmpi(app.TabellewhlenDropDown.Value,'zusatzkosten') || ...
       strcmpi(app.TabellewhlenDropDown.Value,'lieferant') || ...
       strcmpi(app.TabellewhlenDropDown.Value,'stueckliste_labormaschine')

   app.Aenderungstabelle.ColumnEditable = [false true true true true ...
        true true true true true true true true true true];

elseif strcmpi(app.TabellewhlenDropDown.Value,'baugruppe') || ...
         strcmpi(app.TabellewhlenDropDown.Value,'komponente') || ...
         strcmpi(app.TabellewhlenDropDown.Value,'modul')

    app.Aenderungstabelle.ColumnEditable = [false false true];

elseif strcmpi(app.TabellewhlenDropDown.Value,'teil_lieferant')

    app.Aenderungstabelle.ColumnEditable = [false false true true true ...
                            true true true true true true true true true];
else

    app.Aenderungstabelle.ColumnEditable = false;

    app.SpeichernButton_2.Visible = 'off';

end
```

5.3.2

5.5 Hinzufügen von Datensätzen

Durch Drücken der Schaltfläche werden vorgefertigte leere Zeilen für die Tabellen Zusatzkosten, Lieferant, Modul und die Stücklisten hinzugefügt. Damit dies auch möglich ist, nachdem bereits eine Auswahl in der Serverdatentabelle getroffen wurde und Daten an die Änderungstabelle übertragen wurden, wird zunächst die Änderungstabelle über eine bedingte Anweisung geleert.

```
if app.ServerDaten.Selection ~= 0

   app.Aenderungstabelle.Data = {};

   app.ServerDaten.Selection = {};

end
```

5.4.1

Durch die folgenden Programmzeilen wird eine neue Zeile in die Änderungstabelle eingefügt. Dabei werden die bereits vorhandenen Zeilen inkl. Daten zwischengespeichert und ggf. um eine Zeile ergänzt. Dadurch können erst Daten bearbeitet werden und zu einem späteren Zeitpunkt weitere Zeilen hinzugefügt werden, ohne diese zu löschen. Beim Hinzufügen von Zeilen werden mit den Funktionen "cellfun" und "isempty" alle leeren Zellen mit einer null als Platzhalter versehen. Dadurch müssen Zahlenspalten nicht in ein Zahlenformat umgewandelt werden. Zusätzlich wird die Funktion für das Speichern auf zwei gesetzt.

```
current_data = app.Aenderungstabelle.Data;
colc = width(app.ServerDaten.Data);
current_data = [current_data; cell(1, colc)];

tf = cellfun('isempty',current_data);
current_data(tf) = {0};

app.Aenderungstabelle.Data = current_data;
app.funktion = 2;
```

5.4.2

Im nächsten Programmabschnitt wird noch für die jeweilige Tabelle automatisch die richtige ID erzeugt. Dazu wird die höchste ID der abgerufenen Serverdaten um eins erhöht. Für die Tabelle Modul ist dies nicht möglich, da Module für Normteile von 999 fallend durchnummeriert werden. Deshalb wird hierfür in einer bedingten Anweisung für die Modultabelle eine Eingabeaufforderung in der Änderungsdatentabelle ausgegeben.

```
if strcmpi(app.TabellewhlenDropDown.Value,'modul')

   app.Aenderungstabelle.Data{height(current_data), 3} = 'IDs bitte ...
                                                  selbst  logsich vergeben';

   app.Aenderungstabelle.ColumnEditable = [true true true];

else

   app.Aenderungstabelle.Data{height(current_data), 1} = app.max_id + ...
                                                  height(current_data);

app.Aenderungstabelle.ColumnEditable = [false true true true true ...
                        true true true true true true true true true];

end
```

5.4.3

5.6 Suchfunktion

Durch die Suchfunktion soll es dem Benutzer vereinfacht werden, die gewünschten Daten zu finden. Durch die Eingabe einer Zeichenfolge oder auch Zahlenfolge wird die gesamte ausgewählte Tabelle nach diesem Wert durchsucht und die Zeilen mit mindestens einen Treffer in der Serverdatentabelle ausgegeben. Da diese Funktion, alle Spaltenköpfe nach einem Wert zu durchsuchen, in SQL so nicht vorgesehen ist, muss etwas nachgeholfen und ein längerer Selektionsstring erstellt werden. Jede zu durchsuchende Spalte muss einzeln mit der Suchvariable genannt und mit einem logischen Oder (OR) zu den anderen Spalten abgetrennt werden. Zunächst wird dafür die Suchvariable aufbereitet, indem davor und danach ein Prozentzeichen als Platzhalter eingefügt wird. Das bedeutet, dass vor und nach der Suchvariable eine beliebige Anzahl an Zeichen stehen kann. Für Erzeugen des Selektionsstrings wird die Funktion "strjoin" verwendet. Mit dieser werden die Spaltennamen der gewählten Tabelle mit den benötigten SQL-Bezeichnungen und der Suchvariable verkettet. Damit dieser Programmcode nur für gültige Auswahlmöglichkeiten des Dropdown-Menüs ablaufen kann, wird er in eine bedingte Anweisung eingehüllt. Hierfür wird die nicht gültige Vorauswahl über die Funktion "strcmpi" herausgefiltert.

```
if strcmpi(app.TabellewhlenDropDown.Value,'Bitte Tabelle wählen') == 0

   colnames = app.ServerDaten.ColumnName';
   search_var = ['''','%', app.SuchtextEditField.Value, '%', ''''];

   deli =  [' LIKE ', search_var, ' OR '];

   selectquery = ['SELECT * FROM geschmacksachekaffee.', ...
                  app.TabellewhlenDropDown.Value, ' WHERE ', ...
                  strjoin(colnames, deli), ' Like ', search_var];

   sql_abfrage = select(app.conn, selectquery);
   app.ServerDaten.Data =  sql_abfrage;

end
```

5.5.1

5.7 Speicher Funktion

Da über den Datenbank Editor sowohl Daten geändert als auch neu angelegt werden können, wird für die Speicher Funktion ein Switch Case mit zwei Fällen angewandt. Der erste Fall ist für das Ersetzten von Daten zuständig. Hierzu werden die Spaltennamen aus der Serverdatentabelle entnommen und die Speicherdaten aus der Änderungstabelle. Die Daten werden mittels der Funktion "update" in die Datenbank übertragen. Als Parameter zu den Daten und Spaltennamen benötigt diese noch den sogenannten Whereclause und die zugehörige Tabelle in der Datenbank, wodurch vorgegeben wird, wo die Daten ersetzt werden sollen. Für den Whereclause werden die Datenbank-IDs der jeweiligen Tabelle verwendet. In den Tabellen, in welchen diese die Teilenummer ist, muss diese mit single quotes eingeklammert werden. Für die Unterscheidung wird eine bedingte Anweisung verwendet. Die Datenbanktabelle, in der der Datensatz ersetzt werden soll, wird aus dem Dropdown-Menü genommen und mit dem zugehörigen Präfix versehen. Nach der "update" Funktion wird der Speichern-Button zum Signalisieren einer erfolgreichen Datenübertragung grün eingefärbt.

```
switch app.funktion
   case 1
        colnames = app.ServerDaten.ColumnName';
        saving_data = table2cell(app.Aenderungstabelle.Data);

        if strcmpi(colnames{1,1},'teile_nr')
           whereclause = strjoin(string(['WHERE teile_nr = ',''''', ...
                                         saving_data{1,1}, '''']));
        else
           whereclause = strjoin(string(['WHERE ', colnames{1,1}, ...
                                 ' = ', num2str(saving_data{1,1})]));
       end
```

5.6.1

```
        db_table = append('geschmacksachekaffee.', ...
        app.TabellewhlenDropDown.Value);
        update(app.conn, db_table, colnames, saving_data, whereclause)

        app.SpeichernButton_2.BackgroundColor = 'g';


  case 2

        %neu Anlegen von Daten (siehe Programmcode 5.6.2)

end
```

Für den zweiten Fall wird die "sqlwrite" Funktion zum Anlegen der Daten verwendet. Damit keine leeren Zellen gespeichert werden können, wird mit Hilfe der Funktionen "cellfun" und "isempty" in einer bedingten Anweisung die Speichern Schaltfläche ggf. rot eingefärbt und der Programmcode für das Speichern nicht ausgeführt. Die zu speichernden Daten werden aus der Änderungstabelle und die Spaltenköpfe aus der Zusatzdatentabelle entnommen. Aus Kombination dieser Elemente werden die Speicherdaten in eine Tabelle umgewandelt. Die Datenbanktabelle, in welcher die Daten angelegt werden sollen, werden aus dem Dropdown-Menü entnommen und mit dem entsprechenden Datenbank Präfix versehen. Anschließend werden die Daten gespeichert und die Schaltfläche grün eingefärbt.

```
if sum(cellfun('isempty', app.Aenderungstabelle.Data)) > 0

   app.SpeichernButton_2.BackgroundColor = 'r';

else
   colnames = app.ServerDaten.ColumnName';

   saving_data = cell2table(app.Aenderungstabelle.Data , ...
                            "VariableNames", colnames);

   db_table = append('geschmacksachekaffee.', ...
                     app.TabellewhlenDropDown.Value);

   sqlwrite(app.conn, db_table, saving_data)

   app.SpeichernButton_2.BackgroundColor = 'g';

end
```

5.6.2

6 Zusammenfassung und Ausblick

Für das Lieferantenmanagement und den Kostenkalkulator wurde ein geeignetes skalierbares Datenbankmodell in der Software MySQL erstellt. Aufbauend auf diese Datenbank wurde eine umfangreiche App über MATLAB erzeugt, durch welche vom Benutzer Datensätze ohne die Verwendung von MySQL angelegt und verwaltet werden können. Dabei werden Teilenummern automatisch erstellt und zugewiesen. Zusätzlich ist die MATLAB-GUI imstande automatisch die Preise der jeweiligen Maschine nach der entsprechenden Eingabe zu berechnen und Bestelllisten zur generieren. Unabhängig von dieser Nutzung der Datenbank können über diese auch künftig schnelle Datenabfragen getätigt werden. Dabei wäre es zum Beispiel konkret möglich auf einen Blick mit einer entsprechenden Abfrage zu sehen, welche Bauteile von einem bestimmten Lieferanten bezogen werden. Dies stellt nur einen von vielen möglichen Anwendungsfällen dar.

Durch Probleme mit der Hochschul-IT konnte der MySQL Server nicht online gehostet werden. Deshalb konnte die Datenbank und die zugehörige MATLAB-GUI nur lokal programmiert und getestet werden. Darum wird eine leere Datenbankhülle übergeben, die in MySQL importiert werden kann, sobald der Server zur Verfügung steht. Der Connectionstring muss dann nach Kapitel 2.6.2 entsprechend angepasst werden.

Die Datenbank und MATLAB-GUI wurden auf Basis der Daten der Glasboilermaschine entwickelt und aufgebaut. Jedoch wurde beides so gestaltet, dass es ohne großen Aufwand um beliebig viele weitere Maschinen erweitert werden kann. Hierbei wird noch einmal die Skalierbarkeit relationaler Datenbanken und des entwickelten Datenbankmodells klar.

Literaturverzeichnis

[1] Thomas Theis, 2021, "Einstieg in PHP 8 und MySQL" (14. Auflage). Bonn: Rheinwerk Verlag

[2] Adams Ralf, 2022, " SQL : der Grundkurs für Ausbildung und Praxis : mit Beispielen in MySQL/ MariaDB, PostgreSQL und T-SQL" (4., aktualisierte Auflage). München: Carls Hanser Verlag.

[2] MathWorks® documentation, kein Datum, Online. Available: https://de.mathworks.com/help/database/. [Zugriff am 05.02.2023]

[4] Jan Schmager, 08.04.2020, "MySQL-Datentypen", Online. Available: https://www.schmager.de/mysql.php. [Zugriff am 05.02.2023]

[5] MySQL™ documentation, kein Datum, Online. Available: https://dev.mysql.com/doc/. [Zugriff am 05.02.2023]

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang

1 Datenbankmodell