Abschlussarbeit 2025

IHK Software Developer Kurs 3380

ERP-System für Einsteiger

Vorgelegt von

Philip Kottmann
geboren am 03.01.1984 in Göppingen

Im Juli 2025

Betreuer: Erwin Reichel

Inhaltsverzeichnis

Abkürzungsverzeichnis

3D CAD	3-dimensionales Computer Aided Design
CPU	Central Processing Unit (Hauptprozessor)
ERP	Enterprise Ressource Planning
GPU	Graphics Processing Unit (Grafikkarte)
GUI	Graphical User Interface (Benutzeroberfläche)
IDE	Integrated Development Environment (Entwicklungsumgebung)
IHK	Industrie- und Handelskammer
KI	Künstliche Intelligenz
MVC	Model View Controller (Entwurfsmuster)
OOP	Objektorientierte Programmierung
OS	Operation System (Betriebssystem)
OSS	Open Source Software
PC	Personal Computer
RAM	Random Access Memory (Arbeitsspeicher)
SAP R/3	ERP-Lösung der SAP SE
SQL	Structured Query Language
SSD	Solid State Disk (Festplatte)

Abbildungsverzeichnis

Tabellenverzeichnis

1 Zusammenfassung

In dieser Projektarbeit, im Rahmen des IHK[1]-Zertifikatslehrgangs zum Software Developer, entsteht ein ERP[2]-System für Einsteiger in der Programmiersprache Python.
Das System ist als Einzelplatz-Anwendung konzipiert, über welches die Fachbereiche Vertrieb, Disposition und Bauteilmanagement ihre jeweiligen Geschäftsvorgänge auf einer zentralen Datenbasis abwickeln können.
Die GUI[3] ist mit dem Python-Modul tkinter entworfen und die Datenverwaltung findet in einer SQLite-Datenbank statt. Als Entwurfsmuster kommt das MVC[4]-Pattern zum Einsatz. Die Planung der Software-Entwicklung wird anhand des Wasserfall-Modells durchgeführt und die Entwicklungsschritte parallel dazu in dieser Dokumentation festgehalten.
Erstellt wird diese Dokumentation mit LaTeX, als Vorlage dient [2]. Bei der Erstellung der Funktionalitäten und des Quelltextes kommen sowohl die unterrichtsbegleitenden Materialien (Skripte, Bücher), die Aufzeichnungen der Vorlesungen, selbst angeschaffte Literatur sowie diverse Webseiten und Online-Tutorials zum Einsatz. Diese sind am Ende der Dokumentation im Literaturverzeichnis aufgelistet. Auf den Einsatz einer KI[5] zur Quellcode-Generierung wird bewusst verzichtet.
Der vorgegebene Zeitrahmen wurde ausgereizt und ein Großteil der formulierten Use Cases umgesetzt. Das Ergebnis ist am Ende gut gelungen, der Workflow für die Anlage eines Kunden bis zur Auftragsbestätigung ist komplett implementiert und getestet. Die erstellten Funktionalitäten haben einen logischen Aufbau und sind durch ein entsprechendes Fehlerhandling abgesichert. Das Ziel, dass Nutzer ein Verständnis für die Abläufe in einem ERP-System bekommen und der Quellcode als Basis für zukünftige Erweiterungen dienen kann, wurde erreicht. Das persönliche Fazit sowie der Ausblick mit Ideen für eine Erweiterung des Funktionsumfangs runden diese Projektarbeit ab.

[1] Industrie- und Handelskammer
[2] Enterprise Ressource Planning
[3] Graphical User Interface (Benutzeroberfläche)
[4] Model View Controller (Entwurfsmuster)
[5] Künstliche Intelligenz

2 Einleitung

Im Rahmen meiner beruflichen Tätigkeit stelle ich fest, dass die Zukunft meiner Branche nicht mehr in der reinen Produktion von Maschinen, sondern zu einem großen Teil in der Entwicklung und Integration von Software liegen wird. Klassisches Maschinenbau-Können im mechanischen Bereich reicht zukünftig nicht mehr aus, um die Bedürfnisse der Kunden in einer zunehmend digitaler werdenden Welt zu erfüllen. Durch die Einbindung von technischen Geräten, Maschinen und Gadgets aller Art in ein Software- und Server-Ökosystem ist die Bereitstellung, Wartung und Beherrschung eines solchen Software-Systems notwendig und für die Zukunftsfähigkeit von erfolgreichen Unternehmen von großer Bedeutung.
Durch private Software-Projekte für den Sportverein (webbasierte Auswerte-Software für Laufveranstaltungen mit Frontend, Backend und Datenbankanbindung) konnte ich erste Erfahrungen in der Software-Entwicklung sammeln. Dadurch kam mein Wunsch zustande, diese Erfahrungen durch den IHK-Zertifikatslehrgang zum Software Developer zu vertiefen und mir von erfahrenen Software Entwicklern aus der Praxis die Sachverhalte näherbringen zu lassen. Nicht nur die reine Programmierung stand bei mir im Vordergrund, auch wollte ich die begleitenden Prozesse wie Software-Projektmanagement, Anforderungsdefinition, Entwurf, Implementierung und Testing (inkl. Bugfixing) kennenlernen. Diese erworbenen Fähigkeiten setze ich in der vorliegenden Abschlussarbeit im Rahmen der Entwicklung eines *ERP-Systems für Einsteiger* um. Als Vorlage für den Inhalt dient mir das SAP R/3[1]-System, welches bei mir in der beruflichen Praxis tagtäglich zum Einsatz kommt und aus welchem ich teilweise Funktionalitäten nachahmen werde.
Bei der Namensgebung der Klassen, Methoden und Attribute habe ich mich entschieden, auf englische Begriffe zurückzugreifen, da dies meiner bisherigen Erfahrung nach die übliche Vorgehensweise bei der Erstellung von Quellcode darstellt.
Aus Gründen der besseren Lesbarkeit verzichte ich in dieser Abschlussarbeit bei Personenbezeichnungen und personenbezogenen Hauptwörtern auf den gleichzeitigen Einsatz verschiedener geschlechtsbezogener Formulierungen. Es ist für mich selbstverständlich, dass die hier gewählte Form gleichermaßen für alle Geschlechter (m/w/d) gilt.

[1] ERP-Lösung der SAP SE

3 Projektübersicht

3.1 Motivation

Die Erfahrungen in der beruflichen Praxis zeigen, sobald Materialien verwaltet werden sollen und ein entsprechendes System fehlt, der Griff sofort zur Tabellenkalkulation geht, welche dann als Warenwirtschaft und Lagerverwaltung dient. Dies ist bei Projekten mit bis zu zwei Mitarbeitern problemlos möglich. Bei steigender Anzahl an Mitarbeitern kommt dieses System an seine Grenzen. Bei einer Zusammenarbeit über Abteilungs-, Länder- und Kontinentgrenzen hinweg muss auf ein zentrales System mit zuverlässigen Informationen zugegriffen werden können. Fehlerhafte Zahlen und Informationen sorgen letztendlich für Unstimmigkeiten im internen Ablauf und mit dem Kunden, der sich auf Liefertermine seiner bestellten Waren verlässt.
Da bei einer solchen Tabellenkalkulation keine Zugriffsbeschränkungen bestehen, keine Versionierung stattfindet und die Gefahr besteht, dass mit der Zeit mehrere Kopien mit unterschiedlichem Inhalt im Umlauf sind, vergrößert sich die Gefahr für Probleme verschiedenster Art in hohem Maße. Diese Warenwirtschaft in einem zentralen System abzubilden ist meine Motivation für diese Projektarbeit. Dadurch greifen die Mitarbeiter auf einen zentralen Datensatz zu, der eine zuverlässige und tagesaktuelle Datenbasis für die weitere Bearbeitung der Kundenaufträge bietet.

3.2 Projektumfeld

Das Projekt entsteht in einem privaten Rahmen und ist dafür gedacht, in die Systematik der Warenwirtschaft und der dahinterliegenden Workflows einzusteigen. Zusätzlich kann dieses Projekt anderen dazu dienen, ebenfalls ein Verständnis für die Abläufe in der Warenwirtschaft, beginnend von der Anlage von Kunden und Sachnummern bis hin zur Lieferbestätigung zu schaffen. Ich nutze für die Umsetzung die Programmiersprache Python, welche uns im Lehrgang als eine von zwei Programmiersprachen nähergebracht wurde. Die Benutzeroberfläche gestalte ich mithilfe des Python-Moduls tkinter. Durch diese intensive Beschäftigung mit der Programmiersprache Python verspreche ich mir eine Festigung meiner Fähigkeiten im Umgang mit Python sowie der Erweiterung dieser mächtigen Programmiersprache mithilfe von externen Modulen. Zur Datenablage kommt die relationale Datenbank SQLite zum Einsatz. Dadurch werden zum Einen meine Fähigkeiten im Um-

gang mit SQL[1]-Abfragen gestärkt, zum Anderen bietet es perspektivisch die Möglichkeit auch eine serverbasierte Datenbank-Lösung ohne umfangreichen Anpassungsbedarf anzubinden. Zu guter Letzt wird durch die Erstellung des Quellcodes auch meine Routine im Umgang mit dem gewählten Programmierparadigma gefestigt.
Da ich als Einzelperson agiere und dieses Projekt keinen Bezug zu meinem Arbeitgeber hat, verzichte ich an dieser Stelle auf ein klassisches Lastenheft sowie eine Wirtschaftlichkeitsberechnung. Das Lastenheft ersetze ich durch User Stories, die mir in dieser Form im beruflichen Alltag begegnen.

3.3 Projektbeschreibung

Das ERP-System beinhaltet die Bereiche der Auftragsabwicklung und der Lagerverwaltung. Weitere Bereiche werden nicht abgedeckt, jedoch mit berücksichtigt und als Ausblick für zukünftige mögliche Erweiterungen in einem weiteren Kapitel festgehalten. Die Auftragsabwicklung geht von einem internen Auftraggeber in Form des Vertriebs aus. Der Vertriebsmitarbeiter erstellt einen Auftrag basierend auf den verfügbaren Sachnummern im ERP-System. Er bestimmt eine jeweilige Stückzahl der benötigten Komponente und erhält am Ende eine Auftragsbestätigung. Der Vertrieb erhält eine Möglichkeit zur Umsatzauswertung (über einen spezifischen Zeitraum) bezogen auf Kunden oder Materialnummern. Somit bietet das System Möglichkeiten, qualifizierte Entscheidungen für die zukünftige Ausrichtung des Portfolios zu treffen. Die Disposition kann das System nutzen, um Bestände einzusehen, Fehlbestände zu Kundenaufträgen aufzulisten und neue Waren ins Lager zuzubuchen. Der Bauteilmanager nutzt das System um neue Sachnummern anzulegen oder zu pflegen. Er pflegt außerdem Wiederbeschaffungszeiten und Stammdaten für einzelne Sachnummern, die für die weitere Auftragsabwicklung von Bedeutung sind.

3.4 Projektziel

Ziel des Projekts ist es, eine anwenderfreundliche GUI zu erstellen, die den beteiligten Fachbereichen Vertrieb, Disposition und Bauteilmanagement die Möglichkeit der spezifischen Bedienung bietet. Zusätzlich soll die Software robust gegenüber Fehlbedienung ausgeführt werden, sie soll die Funktionalitäten korrekt abbilden und für zukünftige Anpassungen eine gute Wartbarkeit aufweisen. Zur Bedienung stehen verschiedene Schaltflächen sowie Eingabe- und Auswahlfelder mit unterschiedlichen Funktionalitäten zur Verfügung. Die Datenverwaltung erfolgt in einer SQLite-Datenbank, die beim Programmstart lokal er-

[1]Structured Query Language

zeugt, bzw. bei bereits bestehender Datenbank, weitergenutzt wird. Das Programm wird für die vereinfachte Nutzung durch den Endanwender als ausführbare Datei zur Verfügung gestellt.

4 Abgrenzung

Der vorgegebene Zeitrahmen von maximal 80 Stunden wie auch die Ausarbeitung in Eigenregie werden keinen vollständigen Nachbau der SAP R/3-Funktionalitäten zulassen. Nachfolgend erstelle ich eine Übersicht über die Nicht-Funktionalitäten des Software-Programms, die dieses gegenüber den möglichen Erwartungshaltungen abgrenzt.
Es wird keine Möglichkeit zur Stücklistenanlage und -verwaltung sowie keine Integration von Freigabeworkflows für Entwicklungsdaten (Zeichnungen und 3D CAD[1]) geben.
Es ist keine Nutzerverwaltung vorgesehen - jeder Nutzer des Programms hat die gleichen Zugriffsrechte auf alle Bereiche des Warenwirtschaftssystems. Eine Anbindung von Lieferanten und der Datenaustausch mit ihnen ist nicht vorgesehen. Ebenfalls ist keine Rechnungsstellung und keine entsprechende Portalanbindung für Zahlungsdienstleistungen vorgesehen.
Das Programm ist nur für den Betrieb auf einer Workstation mit Tastatur und Monitor vorgesehen. Eine mobile Anwendung ist nicht Teil dieses Projekts.

[1] 3-dimensionales Computer Aided Design

5 Analyse und Methodik

5.1 Vorgehensmodell

Als Vorgehensmodell für die Umsetzung des Projekts dient das Wasserfallmodell (mit Rückkopplung), siehe Abbildung 5.1.

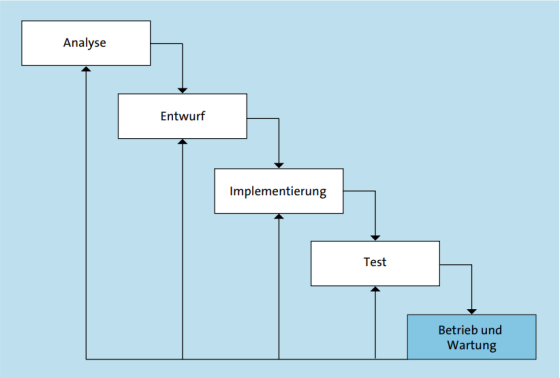

Abbildung 5.1: Wasserfallmodell mit Rückkopplung [1, S. 179]

Laut [1, Kapitel 4.9] gliedert sich das Wasserfallmodell in folgende Schritte, die nacheinander - vergleichbar mit einem Wasserfall - abgearbeitet werden. Durch die Rückkopplung ist es möglich, nach Abschluss der Testphase in eine vorherige Phase zurückzuspringen, Korrekturen vorzunehmen und von dort den Wasserfall erneut zu durchlaufen.
In der Analysephase erfolgt die Definition der Anforderungen an die Software. Hier sind Use Cases aus Sicht der Nutzer der Software berücksichtigt.
In der Entwurfsphase wird die Software-Architektur definiert. Hier werden Funktionalitäten (per Flussdiagramm) sowie das Datenbankdesign festgelegt. Zusätzlich wird ein Entwurf der GUI anhand der Kundenbedürfnisse erstellt.
In der folgenden Implementierungsphase wird der Quellcode erstellt, der für die Funktionalität der Software notwendig ist. Benötigte externe Module werden eingebunden und auf geeignete Funktionalität geprüft. Das Schreiben von Unit-Tests fällt ebenfalls in diese Phase der Softwareentwicklung.
Die Testphase bildet den Abschluss des Entwicklungszyklus, in dem die Software definierten Testkriterien standhalten muss. Hier werden manuell per Schreibtischtest Szenarien geprüft und die zuvor erstellten Unit-Tests automatisiert durchgeführt. Sollten sich in der Testphase Auffälligkeiten ergeben, werden diese in einer weiteren Entwicklungsschleife behoben und erneut getestet.

In der darauffolgenden Phase von Betrieb und Wartung wird die Software dem Kunden in geeigneter Form bereitgestellt und anhand Kundenfeedbacks (Verbesserungspotenziale oder bislang unentdeckte Fehler) überarbeitet und revisioniert. Ein Deployment auf ein Kundensystem ist nicht vorgesehen. Das lauffähige Programm kann per USB-Stick, per E-Mail oder per Cloud-Dienst bereitgestellt werden.

5.2 Zeitplanung

Vom Hauptdozenten des Lehrgangs wurde ein maximaler Zeitrahmen von 80 Stunden zur Bearbeitung der Aufgabe vorgegeben.
Die folgende Tabelle 5.1 zeigt die zeitliche Planung des Softwareprojekts anhand der verschiedenen Phasen. Begleitend zu den einzelnen Phasen des Wasserfallmodells findet die kontinuierliche Dokumentation des Projekts statt.

Arbeitspaket	**Zeitaufwand [h]**
Analyse	3
Entwurf	7
Implementierung	30
Testing	10
Dokumentation	30
Gesamt	**80**

Tabelle 5.1: Zeitplanung

Für die Analyse und den Entwurf (Ausgangssituation, Anforderungsklärung, Abgrenzung, GUI, Datenbankentwurf) wurden in Summe zehn Stunden eingeplant. Die Implementierung wurde im Vorfeld mit 30 Stunden, das kontinuierliche Testen mit zehn Stunden und die begleitende Dokumentation mit 30 Stunden Arbeitsaufwand abgeschätzt.

5.3 Analyse Ist-Zustand

Da es sich bei dieser Projektarbeit um ein rein privat initiiertes Projekt handelt, besteht keine betriebliche Infrastruktur, die als Basis für Vergleiche herangezogen werden kann. Lediglich meine berufliche Praxis als Entwicklungsingenieur und die damit einhergehenden positiven wie negativen Erfahrungen tragen zur Ausarbeitung dieses Projekts bei. Vorbild stellt hierzu das SAP R/3-System dar, das als quasi Standard für betriebliche ERP-Systeme im Maschinenbau angesehen werden kann.

Die Ausarbeitung beginnt demnach ohne Vorlage und wird von Grund auf neu gestaltet. Bei der Benutzeroberfläche wird darauf geachtet, dass für die Anzeige der Daten eine Tabellenform beibehalten wird, um Nutzern von Tabellenkalkulationen ein vertrautes Umfeld zu bieten. Im Rahmen des IHK-Zertifikatlehrgangs wurden uns Java und Python als Programmiersprachen näher gebracht. Für die Umsetzung dieses Projekts habe ich mich für Python als Programmiersprache entschieden.

5.4 Konzeptbeschreibung

Die Grundüberlegung ist, dass in dem hier ausgearbeiteten Konzept ein Basis-Funktionsumfang für die Fachbereiche Vertrieb, Disposition und Bauteilmanagement geschaffen wird. Der Vertrieb ist für den Kontakt zum Kunden notwendig und generiert die erhaltenen Aufträge im System und bestätigt am Ende den Versand mithilfe eines Lieferscheins. Der Vertrieb kann ebenfalls neue Kunden in der Datenbank anlegen sowie die Umsatzkennzahlen nach Sachnummer oder Kunden klassifiziert auswerten. Die Disposition bekommt die Möglichkeit, sich um die Auftragsabwicklung und Warenbestände zu kümmern. Sie möchte Bestände einsehen und entsprechend bei Fehlbeständen oder Mindermengen informiert werden um neue Waren zu beschaffen und einbuchen zu können. Das Bauteilmanagement ist dafür da, neue Sachnummern und Stücklisten anzulegen und die Stammdaten der Sachnummern zu pflegen. In Tabelle 5.2 sind die jeweiligen Use Cases übersichtlich aufgelistet und gewichtet (Muss - Soll - Kann).

Vertrieb	**Disposition**	**Bauteilmanagement**
...möchte neue Kunden anlegen **(Muss)**	...möchte Bestände einsehen **(Muss)**	...möchte Sachnummern anlegen **(Muss)**
...möchte einen Kundenauftrag bezogen auf eine Materialnummer platzieren **(Muss)**	...möchte neue Waren ins Lager zubuchen **(Muss)**	...möchte Stammdaten pflegen **(Muss)**
...möchte den Versand der Kundenbestellungen bestätigen **(Muss)**	...möchte zu den Kundenaufträgen Fehlbestände und Mindermengen auflisten *(Soll)*	...möchte Sachnummern löschen (Kann)
...möchte Jahresumsätze je Kunde auflisten *(Soll)*	...möchte eine Übersicht zur Mengenplanung erhalten *(Soll)*	

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

Vertrieb	**Disposition**	**Bauteilmanagement**
...möchte Jahresumsätze je Sachnummer auflisten *(Soll)*		
...möchte eine Auftragsbestätigung (ggf. mit Lieferdatum) erhalten (Kann)		

Tabelle 5.2: Use Cases Fachbereiche

5.5 Sachmittel

Bei der Auswahl an Sachmitteln wurde auf die privat verfügbare Hardware zurückgegriffen sowie frei verfügbare Software (nach Möglichkeit OSS[1]) eingesetzt.
Als Umgebung steht das eigene Büro („Homeoffice“) mit entsprechend ergonomischer Ausstattung in Form von Bürostuhl und höhenverstellbarem Schreibtisch zur Verfügung.

5.5.1 Hardware

1. PC[2]: HP EliteDesk 800 G5 Desktop Mini
2. CPU[3]: Intel Core i7-9700T (Coffee Lake, 8-Core)
3. RAM[4]: 32 GB
4. GPU[5]: Intel CoffeeLake-S GT2 [UHD Graphics 630]
5. SSD[6]: Kingston 1 TB (SNV3S1000G)
6. Peripherie: Bildschirme, Tastatur und Maus

[1]Open Source Software
[2]Personal Computer
[3]Central Processing Unit (Hauptprozessor)
[4]Random Access Memory (Arbeitsspeicher)
[5]Graphics Processing Unit (Grafikkarte)
[6]Solid State Disk (Festplatte)

5.5.2 Software

1. OS[7]: Linux Mint 22.1 Xia (Kernel: 6.8.0-62-generic, Desktop Cinnamon 6.4.8)
2. IDE[8]: Visual Studio Code Version 1.102.1
3. Python Interpreter: Python 3.12.3
4. LaTeX zur Erstellung der Dokumentation:
   a) Editor: TeXstudio 4.7.2
   b) TeX-Compiler: LuaLaTeX (LuaHBTeX, Version 1.17.0)
5. JabRef 3.8.2: Literatur-/Quellenmanagement

[7]Operation System (Betriebssystem)
[8]Integrated Development Environment (Entwicklungsumgebung)

6 Entwurf

In diesem Kapitel wird der Entwurf des Programms erläutert. Es werden die Benutzeroberfläche, die Anwendungslogik, der Datenbankentwurf, beispielhafte Prozessschritte und die Vorgehensweise erläutert.

6.1 Zielplattform

Es wird eine Anwendung entworfen, die als Einzelplatzanwendung an einer Workstation mit Monitor, Tastatur und Maus bedient wird. Das Programm wird als lauffähige Anwendung für Linux entwickelt.

6.2 Benutzeroberfläche und Anwendungslogik

Die Benutzeroberfläche wird in einem Fenster dargestellt, welches in drei Spalten aufgeteilt ist (Schema siehe Abbildung 6.1). Es bestehen keine Corporate Design-Vorgaben, welche die Farben oder das Erscheinungsbild bestimmen. Die Farbgebung und das Erscheinungsbild entspricht dem Standard des verwendeten Python-Moduls in Kombination mit dem zugrunde liegenden Betriebssystem. Die linke Spalte ist in drei Zeilen unterteilt mit je einem der drei Fachbereiche Vertrieb, Disposition und Bauteilmanagement. Jedem Fachbereich stehen per Schaltflächen verschiedene Funktionalitäten zur Verfügung (siehe Use Cases in Tabelle 5.2). Hierbei gibt es zwei Szenarien, die sich dadurch unterscheiden, ob es sich um eine reine Anzeigefunktionalität oder um einen interaktiven Dialog mit Nutzereingaben handelt. Diese Unterscheidung und die dahinterliegende Logik wird später näher erläutert.
Die mittlere Spalte dient dazu, die Objekte in Tabellenform aufzulisten, die ggf. durch die Auswahl der zugehörigen Schaltfläche in der linken Spalte angefordert wurden. Damit sind bspw. die Kundendaten, die Umsätze oder die Bestände an Sachnummern gemeint. Die rechte Spalte dient dazu, Detailinformationen zu den ausgewählten Objekten aufzulisten. Entweder kann dies durch die Auswahl einer geeigneten Schaltfläche in der linken Spalte geschehen oder durch die Auswahl eines aufgelisteten Objekts aus der Tabelle in der mittleren Spalte.

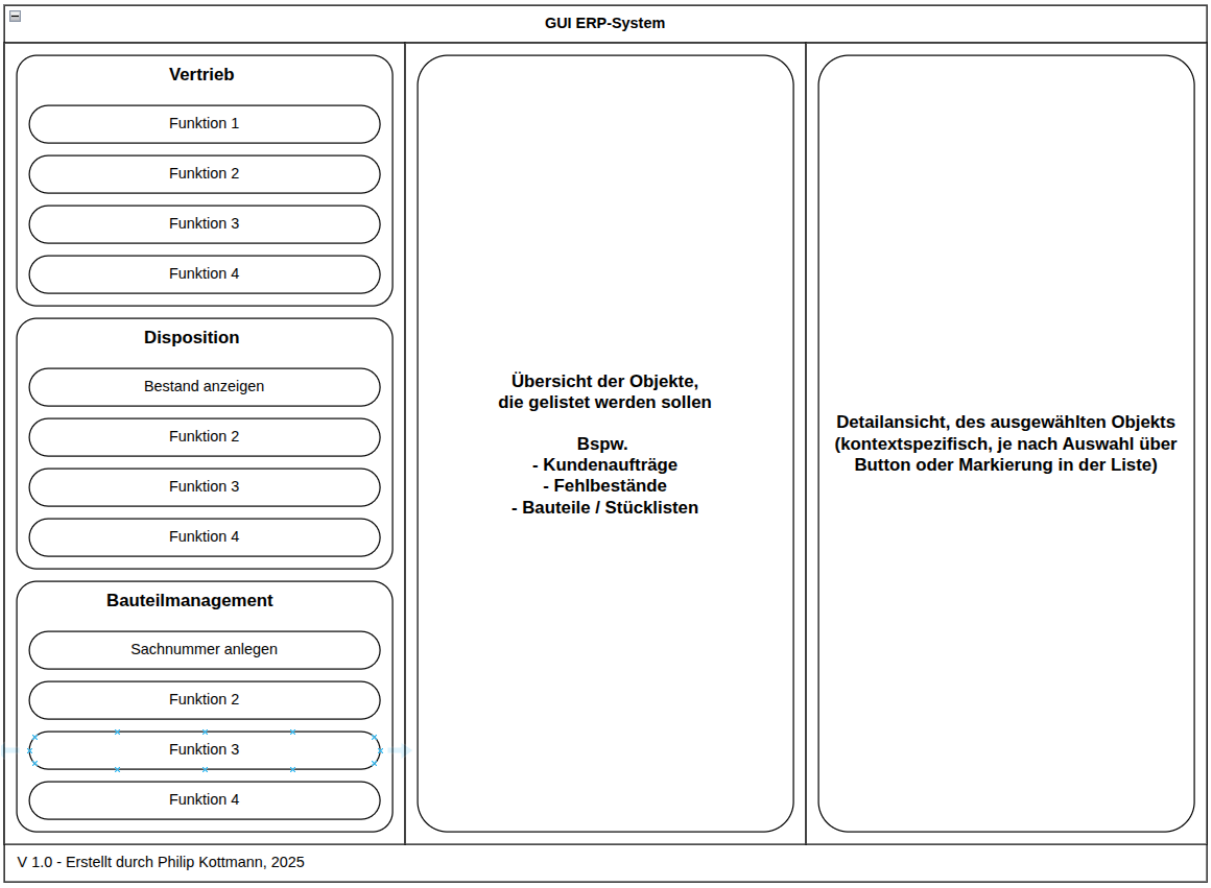

Abbildung 6.1: GUI-Entwurf (mit draw.io erstellt)

Die Funktionalität, die sich hinter den jeweiligen Schaltflächen befindet, kann zu zwei verschiedenen Szenarien der Nutzer-Interaktion führen. Szenario 1 ist eine reine Anzeigelogik wie bspw. die Anzeige der Bestände. Hierbei wird nach Klick auf die Schaltfläche in der mittleren Spalte die entsprechende Auswahl in Tabellenform aufgelistet. Szenario 2 ist eine Funktionalität bei der der Nutzer eine Eingabe tätigen möchte, wie bspw. die Neuanlage einer Sachnummer. Dies bewirkt, dass sich ein Pop-Up Fenster im Vordergrund öffnet, in welchem der Nutzer seine gewünschten Eingaben vornehmen und anschließend bestätigen kann. Auf den folgenden beiden Abbildungen (Abbildung 6.2 und Abbildung 6.3) werden die beiden Szenarien schematisch dargestellt.

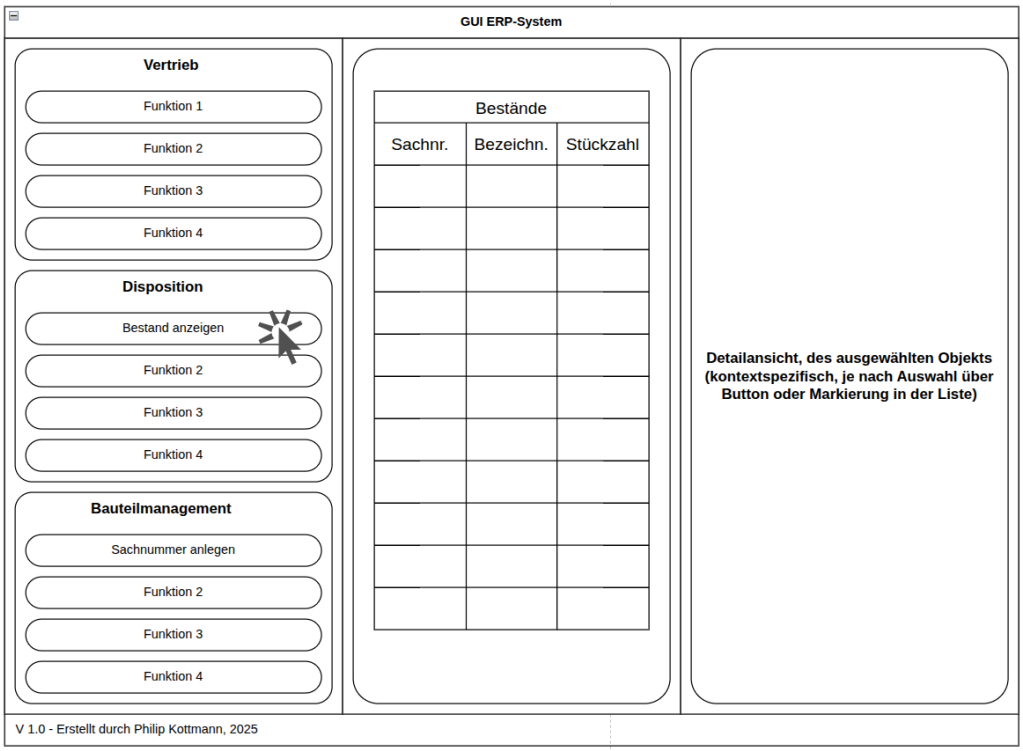

Abbildung 6.2: Szenario 1 - Auswahl Bestände (mit draw.io erstellt)

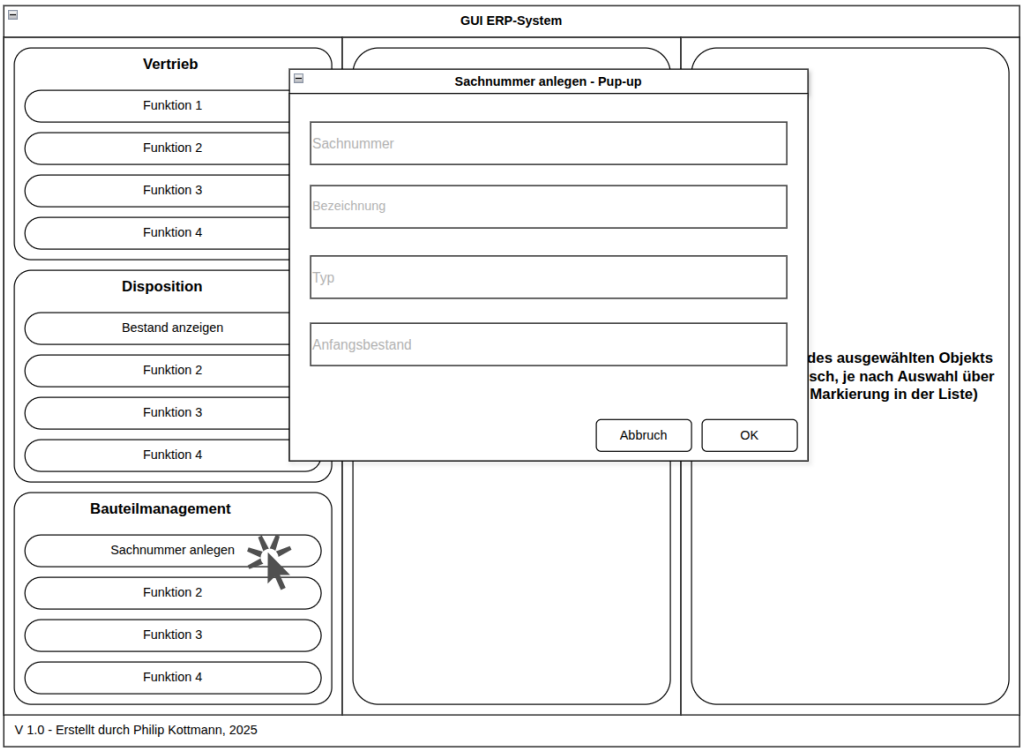

Abbildung 6.3: Szenario 2 - Auswahl Nutzereingabe (mit draw.io erstellt)

Die tabellarische Ansicht in der mittleren Spalte ist so ausgeführt, dass mit einem Doppelklick auf eine Zeile die jeweiligen Stammdaten dazu in der rechten Spalte angezeigt werden. Zum Beispiel werden zu einer Sachnummer die Zusatzdaten in Listenform übersichtlich dargestellt. Diese werden per SQL-Befehl aus der Datenbank gelesen und in Listenform aufbereitet.
Der Entwurf der Datenbank mit den zugehörigen Entitäten und Beziehungstypen wird in einem späteren Abschnitt erläutert.

6.3 Entwurfsmuster

Der Entwurf folgt dem MVC-Pattern. Dabei wird eine Schichtentrennung in der Anwendung realisiert, welches die Datenhaltung (Model), die Steuerung (Controller) und die Ansicht (View) voneinander trennt ([1, Kapitel 6.3.2]). Das Model kümmert sich um die Speicherung und Validierung der Daten und ist außerdem für die Datenbankanbindung zuständig. Die View bildet die Anzeigelogik ab und reicht Benutzereingaben weiter. Der Controller verbindet Model und View indem er den Datenaustausch zwischen den beiden Elementen organisiert. Auf Abbildung 6.4 ist das Schema dieses Entwurfsmusters dargestellt.

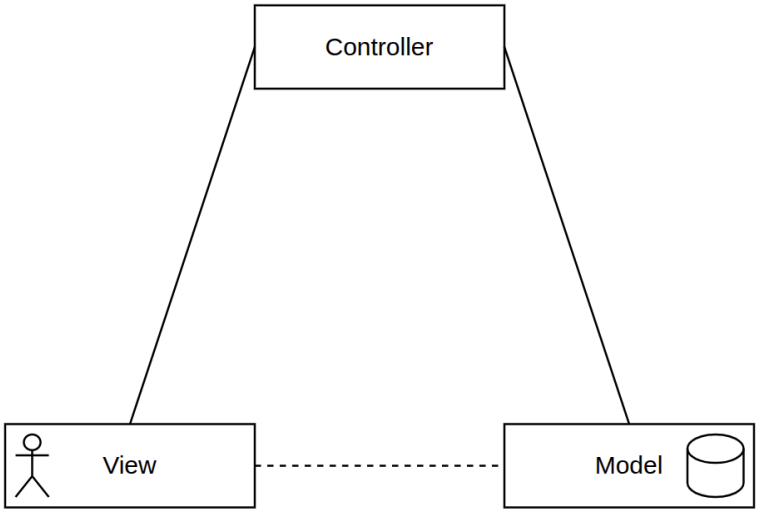

Abbildung 6.4: MVC-Pattern (mit draw.io erstellt)

6.4 Programmierparadigma

Das Programmierparadigma folgt der OOP[1]. Im Gegensatz zur prozeduralen Programmierung, bei der der Quellcode von oben nach unten durchlaufen wird, basiert bei der OOP jedes Objekt (ein abzubildendes Element aus der realen Welt) auf einer Klasse (Bauplan dieses Objekts) mit eigenen Attributen (Variablen) und Methoden (Funktionen). Dies hat den Vorteil, dass die Daten gekapselt sind, sie sind abstrahiert und sie können wiederverwendet und vererbt werden. Sobald ein Objekt aus einer Klasse erzeugt wurde (eine Instanz davon angelegt wurde) hat es für die Dauer des Programmablaufs die aus der Klasse vorgegebenen Eigenschaften und Fähigkeiten und kann diesen entsprechend eingesetzt werden (vgl. [1, Kap. 2.3]).

6.5 Datenbankentwurf

Die benötigten Daten wie Kundendaten, Aufträge, Sachnummern etc. werden in einer SQ-Lite Datenbank abgelegt. Falls es der erste Start des Programms auf diesem Rechner ist, wird diese Datenbank neu angelegt. Sollte bereits mit dem Programm auf dem Rechner gearbeitet worden sein, wird auf die bereits bestehende Datenbank zugegriffen. Die Datenbank wird den Namen *erp_system.db* tragen und ist mit den Tabellen aus dem Entity-Relationship-Diagramm aus Abbildung 6.5 aufgebaut.

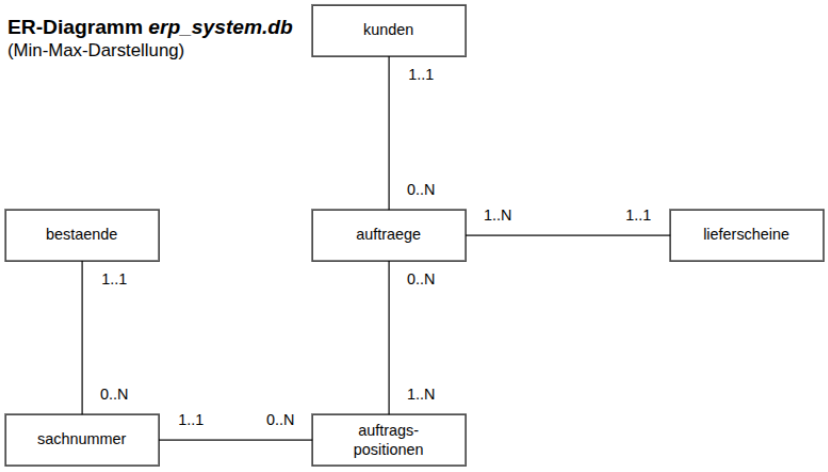

Abbildung 6.5: Entity-Relationship-Diagramm (mit draw.io erstellt)

Die Beziehungen der einzelnen Entitätsmengen zueinander werden in der Min-Max-Darstellung definiert. Bspw. lässt sich in dieser Darstellung erkennen, dass ein *Auftrag* genau einem

[1]Objektorientierte Programmierung

Kunden zugeordnet sein muss, jeder *Kunde* aber keinen, einen oder mehrere *Aufträge* platzieren kann. In der Relationenschreibweise in Abbildung 6.6 wird definiert, welche Attribute die einzelnen Tabellen besitzen und in welcher Beziehung diese zueinander stehen. Dadurch wird ersichtlich, wie die Tabellen untereinander verknüpft sind. Die fett gedruckten Attribute entsprechen den **Primärschlüsseln** (einzigartig pro Tabelle), die kursiv und unterstrichenen Attribute sind die *Fremdschlüssel* (Verknüpfung zu anderen Tabellen).

Relationenschreibweise:

sachnummer:	**sNrID**, materialnummer, bezeichnung, kennungAufbauzustand, stueckliste, warenwert, wiederbeschaffungszeit,
bestaende:	**bestID**, *sNrID*, anzahl
kunden:	**kundID**, firmenname, strasse, hausnummer, plz, ortsname, telefon, email
lieferscheine:	**lieferID**, versanddatum
auftraege:	**aufID,** auftragseingang, auftragsabschluss, *kdNrID*, *lieferID*
auftragspositionen:	**auftrPosID**, *aufID*, *sNrID*

Abbildung 6.6: Relationenschreibweise (mit draw.io erstellt)

Die Primärschlüssel werden als Datentyp Integer mit der Funktion „AUTOINCREMENT“ definiert, was bedeutet, dass sich das Datenbankmanagement-System bei der Neueintragung eines Datensatzes automatisch darum kümmert, dass dieser Wert um den Wert 1 erhöht wird. Die Text-Felder werden als „TEXT“ definiert um den eingegebenen Wert abzulegen. Die Fremdschlüssel werden als „INTEGER“ angelegt, da es sich hierbei um die jeweiligen Fremdschlüssel aus den verbundenen Tabellen handelt.

6.6 Prozessschritte

Im Folgenden sind beispielhaft zwei Flussdiagramme zu sehen (Abbildungen 6.7 und 6.8), die die Workflows „Kundenauftrag anlegen“ und „Sachnummer anlegen“ darstellen.

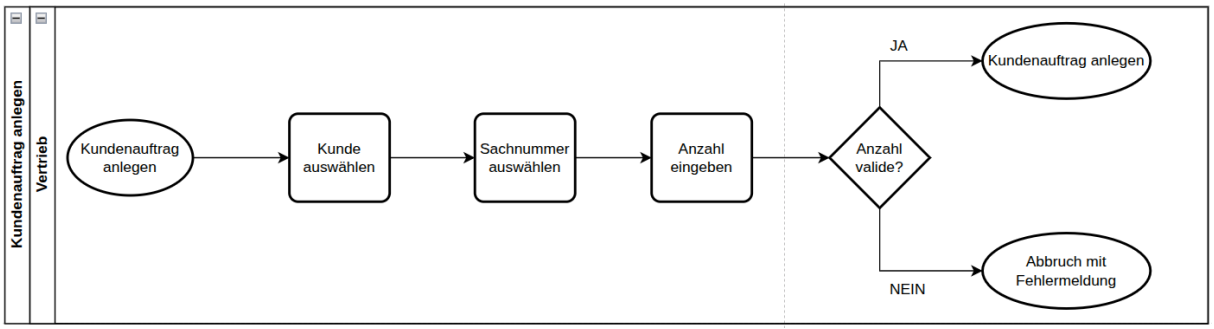

Abbildung 6.7: Flussdiagramm „Kundenauftrag anlegen“ (mit draw.io erstellt)

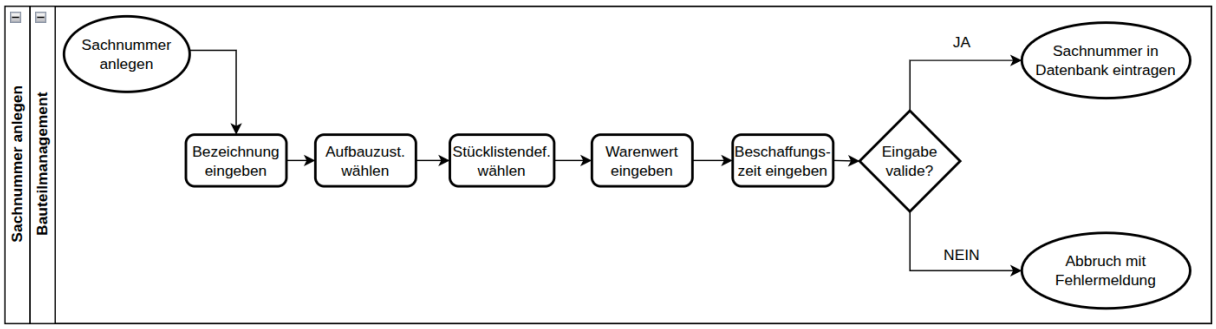

Abbildung 6.8: Flussdiagramm „Sachnummer anlegen“ (mit draw.io erstellt)

6.7 Vorgehensweise

Die Funktionalitäten bei einem Klick auf die Schaltflächen der Fachbereiche werden fest im Quellcode verankert, sodass an dieser Stelle keine Prüfung von unzulässigen Nutzereingaben erfolgen muss. Die Prüfung dieser Funktionalitäten wird in der Testing-Phase durchgeführt und bei Fehlverhalten entsprechend korrigiert.
Individuelle Nutzereingaben über die Tastatur müssen vor Eintragung in die Datenbank einer Validierung unterzogen werden. Tastatureingaben erfolgen beispielsweise bei der Neuanlage eines Kunden, eines Auftrags oder einer Sachnummer. Hier muss bei den Nutzereingaben geprüft werden, ob es sich um zulässige Eingaben handelt. Zum Einen wird geprüft, ob es sich um einen zulässigen Datentyp handelt (also bspw. kein String wenn ein Integer erwartet wird). Zum Anderen muss geprüft werden, dass keine „böswilligen“ Eingaben in Form einer SQL-Injection geschehen und dadurch die Datenbank kompromittiert wird.
Bei der Anlage eines Auftrags wird die Auftragsnummer anhand der ID in der Datenbanktabelle angelegt. Der Nutzer muss den entsprechenden Kunden aus einem Drop-Down-Menü auswählen und die jeweilige Auftragsposition aus einer Liste auswählen. Mit dem Klick auf eine Bestätigungsschaltfläche wird der Kundenauftrag abschließend angelegt. Bei der Anlage eines neuen Kunden müssen die Nutzereingaben auf Plausibilität geprüft werden. Bei der Eingabe der Kontaktdaten (Firmenname, Anschrift) muss geprüft werden, ob es sich um einen String handelt und nicht eine reine Zahlenfolge eingegeben wird. Bei der Eingabe der E-Mail-Adresse muss auf korrekte Formatierung (zum Beispiel: test@beispiel.de) geachtet werden und die Telefonnummer muss auf ein zulässiges Zahlenformat (zum Beispiel: 07123-987654321) geprüft werden.

7 Implementierung und Testing

Im folgenden Abschnitt gehe ich auf die konkrete Umsetzung in der Implementierungsphase ein. Die Code-Ausschnitte sind Auszüge aus dem realen Quellcode. Zur besseren Lesbarkeit sind sie eingekürzt und nicht mit allen Parametern versehen. Der vollständige und funktionale Quellcode ist im Anhang dieser Projektarbeit zu finden.

7.1 MVC-Pattern - Entwurfsmuster

Das Entwurfsmuster folgt dem MVC-Pattern. Ich habe mich dafür entschieden, das Model, die View und den Controller jeweils über eine separate Klasse abzubilden und diese in getrennten Python-Dateien abzulegen. Durch die Aufteilung in diese drei Klassen ist eine Trennung der Funktionalitäten möglich und jede Klasse übernimmt die für sie vorgesehene Aufgabe. Auf weitere Aufteilungen in zusätzliche Unterklassen habe ich der Einfachheit halber verzichtet, auch wenn dies bedeutet, dass diese drei Klassen am Ende einen großen Quelltextumfang haben werden. Wie in Kapitel 6.3 beschrieben ist das Model für die Datenhaltung und Geschäftslogik zuständig, die View für die Anzeigelogik in der GUI und der Controller kümmert sich um den Datenaustausch zwischen diesen beiden Modulen. In den folgenden Abschnitten erläutere ich den grundsätzlichen Aufbau, sowie die konkrete Ausführung im Quellcode.
Die **Model**-Klasse erbt aus keiner anderen Klasse und erzeugt auch selbst keine Instanz einer anderen Klasse. In ihrer *___init()___*-Methode, die beim Erzeugen einer Model-Instanz aufgerufen wird, baut sie die Verbindung zur Datenbank auf. Durch den Kontextmanager mit dem Schlüsselwort *with* wird sicher gestellt, dass am Ende des Prozesses die Datenbankverbindung automatisch geschlossen wird und man nicht manuell die *close()*-Methode auf dem *self.connection*-Objekt anwenden muss. Sollte keine Datenbank mit dem angegebenen Namen existieren (in diesem Fall „erp_system.db“), wird die Datenbank in diesem Zuge angelegt. Mit der Methode *execute(PRAGMA foreign_keys = ON)* wird die SQLite-Datenbank so konfiguriert, dass sie auch Fremdschlüssel-Beziehungen zulässt. Mit *self.connection.commit()* wird dieser zuvor erzeugte Befehl an die Datenbank gesendet und ausgeführt. Das Objekt *self.cursor* dient dazu, ein *cursor()*-Objekt auf die Datenbank zu erzeugen, mit welchem auf deren Tabellen sowie Attribute zugegriffen werden kann.
Hier ein Ausschnitt der Definition der Model-Klasse:

```
class Model:
  def __init__(self):
```

```
with sqlite3.connect("erp_system.db") as self.connection:
    self.connection.execute("""PRAGMA foreign_keys = ON""")
    self.connection.commit()
    self.cursor = self.connection.cursor()
    ...
```

Die **View**-Klasse erbt alle Attribute und Methoden aus der tkinter-Klasse *Tk()*. Angegeben wird dies mit dem Befehl *class View(tk.Tk)*. *tk* steht in diesem Fall für den Alias des *tkinter-Moduls*, welcher beim Import des Moduls definiert werden kann um eine verkürzte und damit übersichtlichere Schreibweise des Modulaufrufs sicherzustellen. Über die *super().__init()__*-Methode kann die View-Klasse auf alle Methoden der vererbenden Klasse (in diesem Fall *tk.Tk()*) zugreifen. In der *__init()__*-Methode werden zusätzlich die Klassenattribute und -konstanten definiert - hier beispielhaft an *self.PAD = 7* dargestellt (Konstante für den Außenabstand der Widgets, die in ihrer Gesamtheit die GUI bilden).
In der View-Klasse befindet sich auch die *main()*-Methode, die dazu dient, das tkinter-Fenster mithilfe der *mainloop()*-Methode in Dauerschleife darzustellen, bis es der Benutzer beendet.
Hier ein Ausschnitt der Definition der View-Klasse:

```
class View(tk.Tk):
  def __init__(self, controller):
    super().__init__()
    self.PAD = 7     # Aussenabstand allgemein
    ...

  def main(self):
    self.mainloop()
```

Die **Controller**-Klasse ist das Verbindungsglied zwischen View und Model. In der Controller-Pythondatei befindet sich das *if __name_ == "__main__"*-Konstrukt, das sicherstellt, dass nur beim Ausführen der Controller-Datei das Programm gestartet wird. Die Systemvariable *__name__* wird abgefragt und sobald dort der Wert „*__main__*“ hinterlegt ist, wird der darin enthaltene Quellcode ausgeführt. Dieser erzeugt durch *erp_system = Controller()* und *erp_system.main()* eine Instanz der Controller-Klasse, speichert diese im *erp_system*-Attribut und führt daraufhin die *main()*-Methode des Controllers aus. Diese sorgt dafür, dass die *main()*-Methode der View-Klasse ausgeführt und damit die GUI gestartet wird. Bei der Erzeugung der Instanz der Controller-Klasse wird über die *__init()__*-Methode eine Instanz des Models (*self.model = Model()*) sowie eine Instanz der View-Klasse (*self.view = View(self)*) erzeugt. Der View-Klasse wird das Schlüsselwort *self*

übergeben, was bewirkt, dass die View-Klasse zukünftig auf den Controller zugreifen und Befehle an ihn weiterreichen kann.
Hier ein Ausschnitt der Definition der Controller-Klasse:

```
class Controller:
  def __init__(self):
    self.model = Model()
    self.view = View(self)
    self.next_partnumber = 0
    ...

  def main(self):
    self.view.main()

if __name__ == "__main__":
  erp_system = Controller()
  erp_system.main()
```

7.1.1 Model

7.1.1.1 Datenbankanbindung

In diesem Abschnitt erläutere ich anhand eines Beispiels, wie die Datenbankerzeugung und die Interaktion mit dieser abläuft. Beim Erzeugen der Instanz der Model-Klasse wird automatisch die Datenbankstruktur aus Kapitel 6.5 angelegt, sofern sie nicht bereits besteht. Dies geschieht über folgende Methode *_create_table(self, sql)*.

```
def _create_table(self, sql):
  self.cursor.execute(sql)
  self.connection.commit()
```

Diese Methode bekommt als Parameter das SQL-Statement übergeben und führt es entsprechend aus. Nachfolgend beispielhaft ein Methodenaufruf inkl. SQL-Statement zum Anlegen einer Tabelle in der Datenbank. Dabei werden die Attribute (Spalten) der Tabelle mit den jeweils zulässigen Datentypen angelegt. Die beiden Schlüsselwörter „PRIMARY KEY“ und „AUTOINCREMENT“ bedeuten, dass dieses Attribut der Primärschlüssel der Tabelle ist und automatisch beim Anlegen eines neuen Datensatzes um den Wert 1 erhöht wird. Mit dieser Logik werden auch die weiteren benötigten Tabellen angelegt. Der komplette Quellcode ist im Anhang dieser Projektarbeit zu finden.

```
self._create_table("""CREATE TABLE IF NOT EXISTS sachnummern
        (sNrID INTEGER PRIMARY KEY AUTOINCREMENT,
        materialnummer INTEGER,
        bezeichnung TEXT,
        kennungAufbauzustand TEXT,
        stueckliste INTEGER,
        warenwert INTEGER,
        wiederbeschaffungszeit INTEGER)""")
```

7.1.1.2 Geschäftslogik

Bei der Geschäftslogik innerhalb der Model-Klasse gehe ich beispielhaft auf eine Methode ein, die die höchste angelegte Sachnummer aus der Tabelle *sachnummern* ermittelt und als Integer-Wert über ein return-Statement zurückgibt. Folgender Quellcode liegt dieser Methode zugrunde:

```
def read_highest_ID_from_sachnummern(self):
  self.cursor.execute("""SELECT sNrID
                FROM sachnummern ORDER BY sNrID DESC LIMIT 1""")
  last_ids = self.cursor.fetchone()
  if last_ids is None:
    return 0
  else:
    for last_id in last_ids:
      return int(last_id)
```

Hierbei werden über das SQL-Statement die Sachnummern (*sNrID*) aus der Tabelle *sachnummern* gelesen und absteigend sortiert. Mit dem Befehl *LIMIT 1* wird erreicht, dass nur der letzte Eintrag aus der Datenbank zurückgegeben wird. Dieser entspricht durch die Sortierung dem höchsten Index. Mit der cursor-Methode *fetchone()* wird der Eintrag gelesen und als Tupel im Attribut *last_ids* abgelegt. Mit der darauffolgenden Kontrollstruktur in Form einer if-else-Verzweigung wird geprüft, ob die Datenbank noch keine Einträge enthält (*if last_ids is None*). Ist das der Fall bricht die Methode ab und kehrt zur Stelle des Methodenaufrufs zurück. Sollte das Attribut *last_ids* nicht leer sein, wird dieses Tupel mit einer for-Schleife durchlaufen (es wird über die Elemente des Tupel iteriert) und der gelesene Eintrag per Type-Casting als Integer-Wert zur weiteren Verarbeitung zurückgegeben.

7.1.2 View

7.1.2.1 GUI

Die GUI besteht aus einem 3-Spalten-Layout, wie in der nachfolgenden Abbildung 7.1 dargestellt. Das Fenster wird auf eine feste Größe von 1200 x 800 Pixel eingestellt und lässt sich nicht in der Größe ändern.
Die linke Spalte dient der Nutzerinteraktion über Schaltflächen. Die mit * markierten Schaltflächen sind lediglich Platzhalter, die Schaltflächen ohne Markierung sind mit Funktionen belegt.
Die mittlere Spalte zeigt in Listenform die Einträge aus der Datenbank an und in der rechten Spalte werden kontextabhängig Detailinformationen zu den ausgewählten Einträgen der mittleren Spalte angezeigt. In den folgenden Abschnitten gehe ich beispielhaft auf die Implementierung einzelner Elemente (in tkinter „Widgets“ genannt) ein. Zur moderneren Darstellung der einzelnen Widgets werden aus dem tkinter-Modul die ttk-Widgets importiert, die ein zeitgemäßeres Erscheinungsbild als die klassischen tk-Widgets aufweisen.

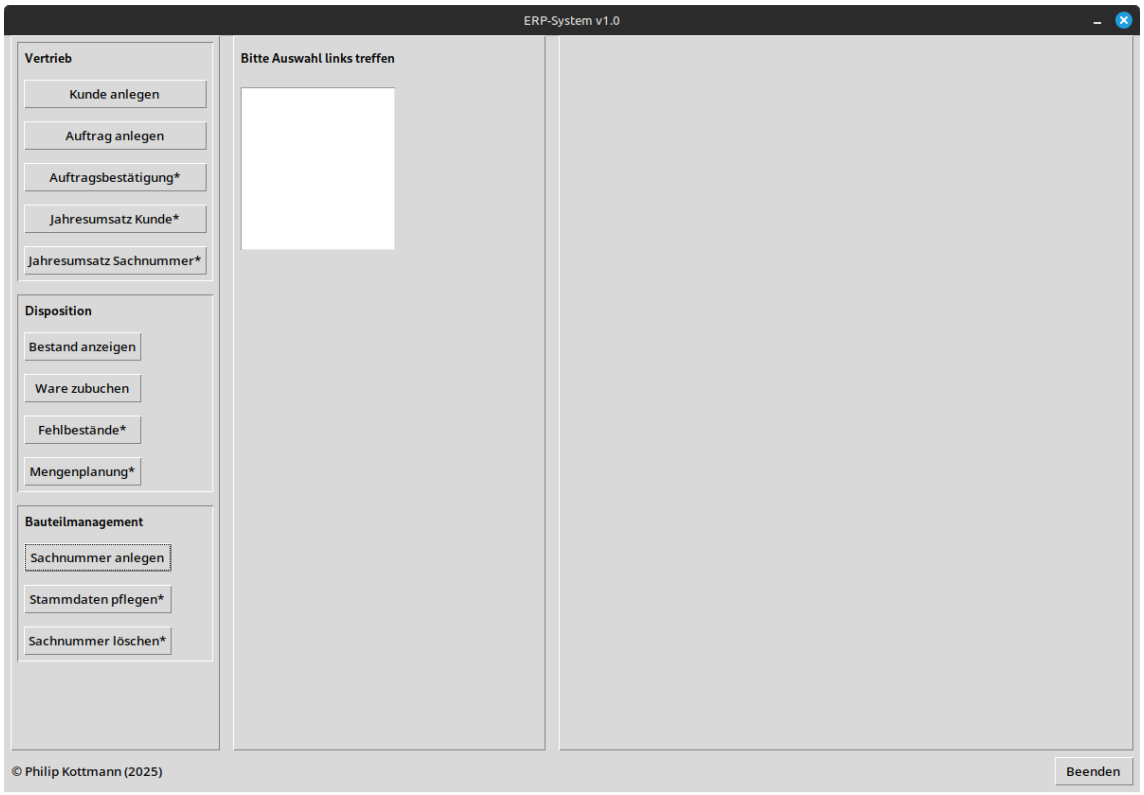

Abbildung 7.1: GUI - Übersicht (gesamtes Programmfenster)

7.1.2.2 Spaltenlayout

Das Spaltenlayout wird über ttk-Frame-Objekte erzeugt, die mit dem grid-Layoutmanager innerhalb des äußeren Fensters platziert werden. Der Quellcode für das Erzeugen des linken Frames sieht (etwas verkürzt) folgendermaßen aus:

```
def _create_left_frame(self):
    self.left_frame = ttk.Frame(self)
    self.left_frame.grid(row=0, column=0)
    self._create_sections()
```

Der ungekürzte Quellcode befindet sich im Anhang dieser Projektarbeit.
In dem hier dargestellten Ablauf wird das Frame-Objekt erzeugt und im Attribut *self.left_frame* abgelegt. Über das self-Attribut wird definiert, dass der Frame im übergeordneten Fenster („Parent-Element“) platziert wird. Über die *grid()*-Methode wird bestimmt, an welchem Platz in der Rasteransicht (grid = Raster) dieser Frame platziert werden soll. Die Methode *self._create_sections()* ruft eine selbst geschriebene Methode auf, die die Elemente innerhalb dieses Frames erzeugt und platziert (siehe Quellcode im Anhang).

7.1.2.3 Schaltflächen

In diesem Abschnitt erläutere ich, wie die Schaltflächen („Buttons“) erzeugt und entsprechend in den passenden Frames platziert werden. Für die Erzeugung und Platzierung ist eine geschachtelte for-Schleife nötig. In der ersten Stufe werden die drei Frames für Vertrieb, Disposition und Bauteilmanagement erzeugt. In der zweiten Stufe werden die Schaltflächen für die einzelnen Funktionalitäten generiert und am Ende entsprechend platziert. Nachfolgend ein gekürzter Auszug des Quellcodes. Die ungekürzte Version befindet sich ebenfalls im Anhang. DEPARTMENTS ist eine Liste, die die einzelnen Fachbereiche beinhaltet: *self.DEPARTMENTS = ["Vertrieb", "Disposition", "Bauteilmanagement"]*

```
def _create_sections(self):
  for i in range(3):
    section = ttk.Frame(self.left_frame, relief="ridge")
    section.grid(row=i, column=0, sticky="EW")
    section_label = ttk.Label(section, text=self.DEPARTMENTS[i])
    section_label.grid(row=0, column=0, sticky="nesw")

    if i == 0:
        department_counter = 5 # Anzahl Btn fuer Vertrieb
    elif i == 1:
        department_counter = 4 # Anzahl Btn fuer Disposition
    elif i == 2:
```

```
        department_counter = 3 # Anzahl Btn fuer Bauteilman.

    for j in range(department_counter):
        if i == 0 and j == 0:
            button = ttk.Button(section, text="""Kunde anlegen""")
        elif i == 0 and j == 1:
            button = ttk.Button(section, text="""Auftrag anlegen""")
        elif i == 0 and j == 2:
            button = ttk.Button(section, text="""Auftragsbest""")
        elif i == 0 and j == 3:
            button = ttk.Button(section, text="""Jahresumsatz Kd""")
        elif i == 0 and j == 4:
            button = ttk.Button(section, text="""Jahresumsatz SNr""")
        elif i == 1 and j == 0:
            button = ttk.Button(section, text="""Ware zubuchen""")
        elif i == 1 and j == 1:
            button = ttk.Button(section, text="""Mindermengen""")
        elif i == 1 and j == 2:
            button = ttk.Button(section, text="""Mengenplanung""")
        elif i == 1 and j == 3:
            button = ttk.Button(section, text="""Bestand anzeigen""")
        elif i == 2 and j == 0:
            button = ttk.Button(section, text="""Sachnummer anlegen""")
        elif i == 2 and j == 1:
            button = ttk.Button(section, text="""Stueckliste anlegen""")
        elif i == 2 and j == 2:
            button = ttk.Button(section, text="""Stammdaten pflegen""")
        else:
            button = ttk.Button(section, text=f"""Button {i+1}-{j+1}""")
        button.grid(row=j+1, column=0, sticky="""EW""")
```

Mit der ersten for-Schleife (*for i in range(3)*) werden die drei Frames für die Fachbereiche erzeugt. Im Objekt *ttk-Label()* wird per Index i auf das jeweilige Element in der Konstanten *self.DEPARTMENTS[i]* zugegriffen und somit der entsprechende String als Überschrift angezeigt. Innerhalb einer Iteration (vollständiger Durchlauf der Schleife) der for-Schleife wird per *if-elif-else*-Verzweigung geprüft, welcher Fachbereich bearbeitet wird und über das Attribut *department_counter* festgelegt, wieviele Schaltflächen in der folgenden Schleife automatisch erzeugt werden. Der Zähler der zweiten for-Schleife wird mit j bezeichnet. Durch die *if-elif-else*-Abfragen innerhalb dieser Schleife kann durch die entsprechende Kombination von Abfragen nach i und j der Button dem jeweiligen Fachbereichs-Frame zugeordnet werden. Nach diesem Schritt sieht die GUI aus wie auf Abbildung 7.2 dargestellt.

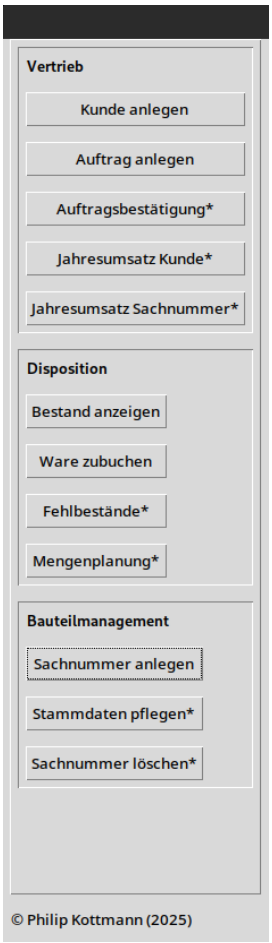

Abbildung 7.2: GUI - linke Spalte

7.1.2.4 Listenansicht

Die mittlere Spalte dient dazu, ausgelesene Daten aus der Datenbank in Listenform darzustellen. Dazu wird das ttk-Widget *Listbox* verwendet. Dieses Widget hat außerdem den Vorteil, dass Zeilen angeklickt und entsprechend auf diesen (Doppel-)Klick reagiert werden kann. Die Methode zur Erzeugung der Listview sieht in verkürzter Form folgendermaßen aus. Der ungekürzte Quelltext hierfür ist auch im Anhang zu finden.

```
def create_listview(self, content, heading, category):
  self.content = content
  category = category

  if self.content == 0:
        self.listbox_label = ttk.Label(self.center_frame)
  else:
        self.listbox_label = ttk.Label(self.center_frame)
  self.listbox_label.grid(row=0, column=0, sticky="""EW""")
```

```
self.listbox = tk.Listbox(self.center_frame)
self.listbox.config(font=("Courier", 10))
self.listbox.grid(row=1, column=0, sticky=""EW"")

if self.content == 0:
      self.listbox.insert(0, """ """)
else:
  heading_listbox = f"""S-Nr./Bezeichnung      /{category}"""
  self.listbox.insert(0, heading_listbox)

for item in self.content:
      row = f"{item[1]}|{item[2]}|{item[3]}"
      self.listbox.insert(tk.END, row)

self.listbox.bind('<Double-Button-1>', self.get_cursor_title)
```

Dieser Methode werden die Parameter *content* (Inhalt aus der Datenbank), *heading* (Überschrift des Frames) und *category* (z.B. Stückzahl, z.B. Anzahl etc.) für die Überschrift der Spalten in der Listview übergeben. Die *if-else*-Kontrollstrukturen dienen dazu, verschiedene Zustände abzufangen und entsprechend in der Beschriftung und Darstellung zu reagieren. Mit dem Attribut *heading_listbox* wird die erste Zeile in der Listbox erzeugt und dient als Überschrift der einzelnen Spalten. Ein Doppelklick auf diese Zeile wird allerdings ignoriert. Erst die darauffolgende erste Zeile wird per Doppelklick ausgewertet. Mit der *insert()*-Methode wird der Listbox ein Element zugeordnet. Werden von der Datenbank mehrere Elemente zurückgegeben, wird über eine for-Schleife sichergestellt, dass die Elemente nacheinander durchlaufen und in der Listbox zeilenweise aufgeführt werden. Die *bind()*-Methode reagiert auf einen Doppelklick (*<Double-Button-1>*) und ruft die Methode *self.get_cursor_title* auf.
Diese self.get_cursor_title()-Methode reagiert auf den Doppelklick, wertet den entsprechenden Index (entspricht der sNrID) aus und gibt diesen zurück. Die Methode ist folgendermaßen implementiert:

```
def get_cursor_title(self, event):
  selection = self.listbox.curselection()

  if selection:
    index = selection[0]-1
    if index < 0:
      return
    else:
      partnumber = self.content[index]
```

```
if isinstance(partnumber, str):
    self.show_message_box("""Auswahl""", """Bitte Auswahl unten treffen""")
    return
else:
    self.show_context_partnumbers(partnumber)
```

Diese Methode bekommt das Doppelklick-Event übergeben und kann mit *curselection()* auf den Index der Auswahl zugreifen. Da die erste Zeile in der *Listview* durch die Überschrift belegt ist, entspricht der erste zurückgegebene Index der 1. Da dies in den Werten aus der Datenbank nicht dem ersten Index (diese Zählweise beginnt bei 0) entspricht, muss die Auswahl (*selection[0]*) durch Subtraktion von 1 verringert werden.
Mit der Methode *isinstance()* wird geprüft, ob es sich beim Index um einen String handelt (bei Doppelklick auf die erste Zeile, die der Überschrift entspricht). Wenn dies der Fall ist, wird der Doppelklick ignoriert und der Nutzer kann erneut eine Zeile auswählen. Sobald als Index ein Integer (Attribut *partnumber*) ausgewählt ist, wird die Methode *show_context_partnumbers()* aufgerufen, die die Details zu der Sachnummer aus der Datenbank liest und an die View zur Darstellung zurückgibt. Zu diesem Zeitpunkt sieht die GUI aus wie folgt (Abbildung 7.3):

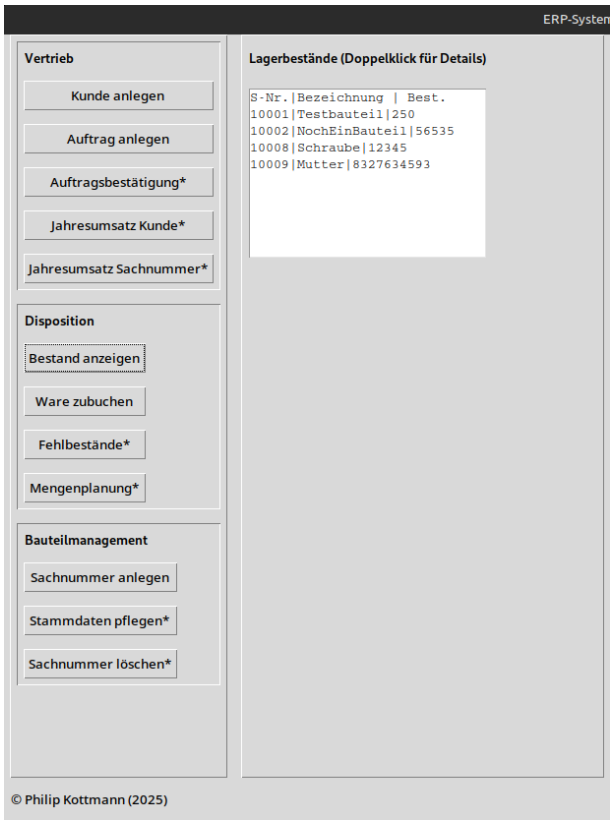

Abbildung 7.3: GUI - mit Listenansicht

7.1.2.5 Detailansicht

Nach der Übergabe der Details der Sachnummer an die View wird nun auch die rechte Spalte mit einbezogen und mit Informationen gefüllt. Diese dient zur Darstellung der Details der Auswahl, die in der *Listview* durch den Nutzer getroffen wurde (bspw. Details zur Sachnummer oder zum Kunden). Diese Ansicht wird im rechten Frame aus einer Aneinanderreihung von ttk.Labels mit dem entsprechenden Inhalt generiert. Die Methode wird hier in stark verkürzter Form dargestellt, da sie keine neuen Aspekte beinhaltet. Der komplette Quellcode ist auch hierzu im Anhang zu finden. Die Methode ist folgendermaßen definiert:

```
def show_context_partnumbers(self, partnumber):
    self.result = self.controller.read_context_partnumber(partnumber[0])
    ...
```

Dieser Methode wird die Sachnummer übergeben. Sie ruft über das Controller-Objekt die Methode zum Auslesen der Detaildaten aus der Datenbank auf (*self.controller.read_context_partnumber(partnumber[0])*) und legt den Rückgabewert im Attribut *self.result* zur weiteren Verarbeitung ab. Die GUI sieht zu diesem Zeitpunkt folgendermaßen aus:

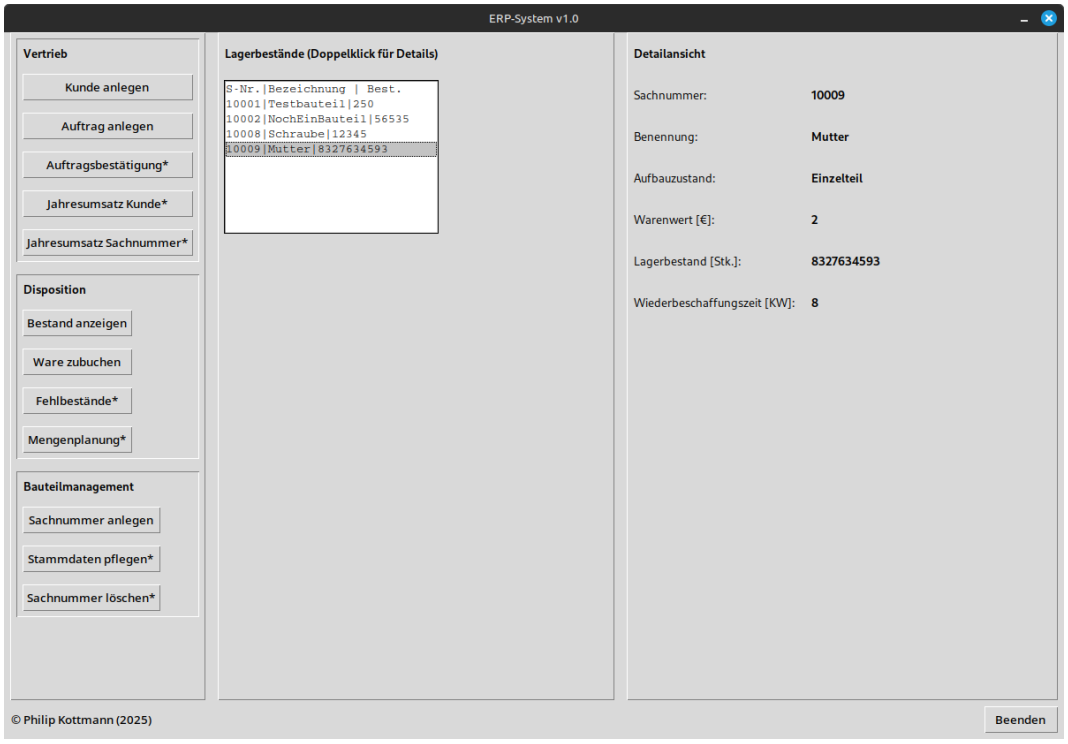

Abbildung 7.4: GUI - mit Detailansicht

7.1.2.6 Datenvalidierung

Bevor Nutzereingaben von der Tastatur in die Datenbank geschrieben werden, müssen sie auf Plausibilität geprüft werden. Zum Einen muss eine SQL-Injection vermieden werden, damit niemand durch Einschleusen von schadhaftem Code die Datenbank kompromittieren kann. Zum Anderen müssen Nutzereingaben auf Logikfehler geprüft und abgefangen werden. Bspw. muss eine Postleitzahl eine fünfstellige Ganzzahl sein. Sie darf nicht kürzer oder länger sein und sie darf auch keine Buchstaben enthalten. Auch muss die Eingabe der E-Mail-Adresse auf korrekte Formatierung geprüft werden. Eine E-Mail-Adresse besteht immer aus *Nutzername@Domain.Top-Level-Domain*, also bspw. test@example.com. Beispielhaft habe ich bei der Anlage eines neuen Kunden die Postleitzahl auf Korrektheit (Datentyp Integer, fünfstellige Ganzzahl) und die E-Mail-Adresse über „regular expressions“ auf das passende Format geprüft. Eingebettet ist diese Prüfung in einen try-except-Block, der auf entsprechende Fehler reagiert. Sollte eine Fehleingabe vorgenommen worden sein, erhält der Nutzer einen passenden Korrekturhinweis in Form eines Pop-Up Fensters. Der Quellcode hierzu ist gestaltet wie in der folgenden Übersicht. Die komplette Quellcode-Datei ist im Anhang zu finden.

```
# Abfangen von Fehleingaben
try:
  if int(self.entry_company_zip.get()) and (len(
                                    str(self.entry_company_zip.get())) == 5):
    if re.match(r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$',
                                    self.entry_company_email.get().strip()):
      if self.controller.add_customer(customer_data):
        self.pop_up_win.destroy()
        tk.messagebox.showinfo("Erfolg", """Erfolgreich in DB eingetragen!""")
      else:
        tk.messagebox.showerror("Fehler", """Keine Kundendaten vorhanden!""")
        self.pop_up_win.destroy()
    else:
      tk.messagebox.showwarning("Fehleingabe", """Falsches E-Mail-Format\n

  else:
    tk.messagebox.showwarning("Fehleingabe", """PLZ: 5-stellige Ganzzahl""")
except ValueError:
  tk.messagebox.showwarning("Fehleingabe", """PLZ: 5-stellige Ganzzahl""") \\
```

7.1.2.7 Struktogramm Fehlerhandling Kundendaten

Das Struktogramm zu diesen Kontrollstrukturen gestaltet sich wie auf der folgenden Abbildung 7.5 dargestellt:

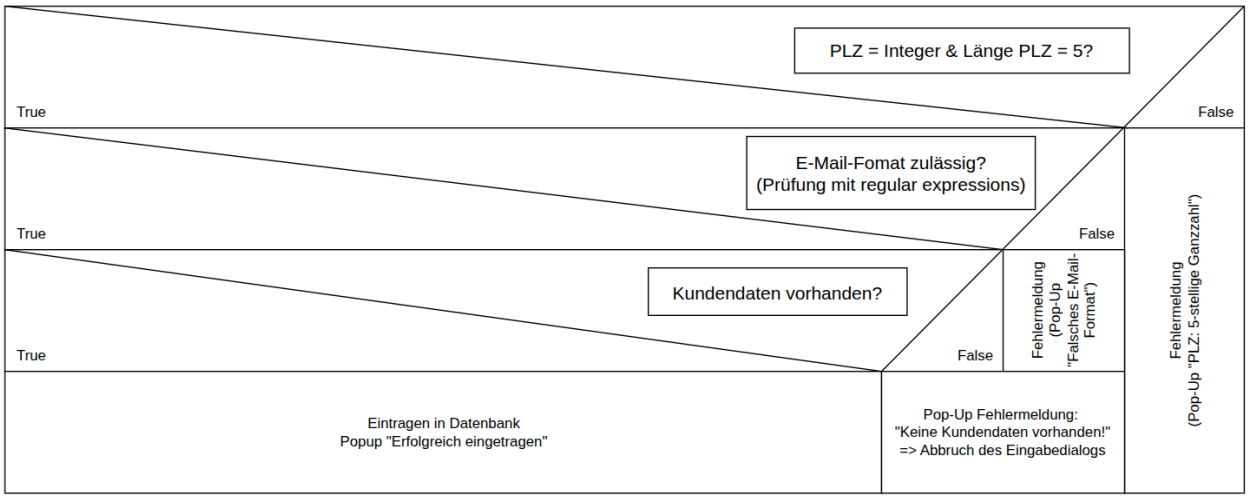

Abbildung 7.5: Struktogramm Fehlerhandling Eingabe Kundendaten

7.1.2.8 Fehlermeldungen

Die realen Fehlermeldungen und Hinweistexte in der GUI sind auf den Abbildungen 7.6 (Fehler Postleitzahl) und Abbildung 7.7 (Fehler E-Mail-Adresse) dargestellt.

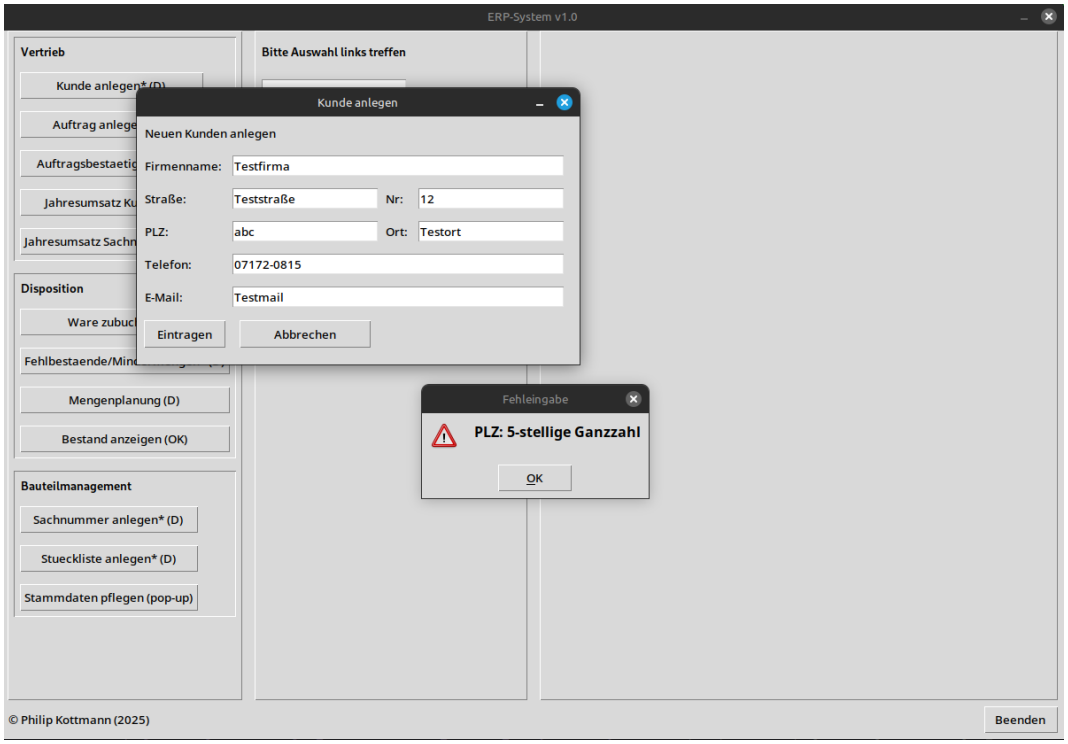

Abbildung 7.6: Fehler Postleitzahl

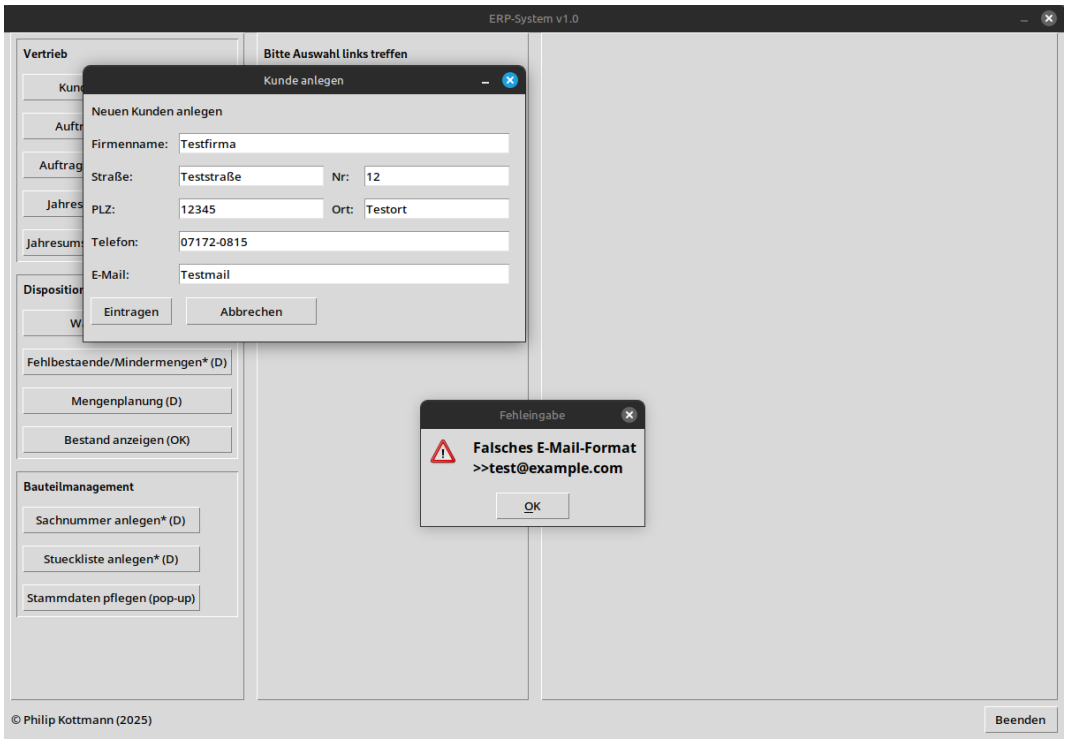

Abbildung 7.7: Fehler E-Mail-Adresse

7.1.3 Controller

Der Controller agiert als Verbindungsglied zwischen der View und dem Model. Er nimmt Anfragen der View entgegen und aktualisiert sie nach Änderungen am und durch das Model. Beispielhaft beschreibe ich den Aufruf der Methode zum Auslesen der Detaildaten einer Sachnummer (*read_context_partnumber(self, partnumber)*). Diese Methode wird aus der View über den Doppelklick auf eine Zeile in der Listview angesprochen und ruft die Funktion zum Auslesen der Daten der Sachnummer aus dem Model auf. Der Quellcode sieht folgendermaßen aus:

```
def read_context_partnumber(self, partnumber):
        return self.model.read_context_partnumber(partnumber)
```

Als Rückgabewert gibt die Funktion eine Liste mit den abgefragten Daten aus der Datenbank zurück. Diese werden in der View verarbeitet und an den entsprechenden Stellen für den Nutzer dargestellt.

7.2 Testing

In diesem Kapitel beschreibe ich Testfälle zum Prüfen der verschiedenen Funktionalitäten. Dabei habe ich auf zwei unterschiedliche Testverfahren zurückgegriffen. Das Erste ist ein manueller Schreibtischtest, den man als Software-Entwickler „per Hand“ vornimmt um seine Implementierung mithilfe von Grenzfällen abzuprüfen. Bei einer Division ist solch ein Grenzfall bspw. das Teilen durch 0. Bei der Verarbeitung von Nutzereingaben wird z.B. geprüft, ob der eingegebene Wert dem gewünschten Datentyp entspricht oder ob bspw. anstatt des Geburtsdatums ein Text eingegeben wurde.
Das Zweite ist ein Unit-Test, bei dem automatisiert definierte Testszenarien geprüft und bewertet werden. Bei Anpassungen am Quellcode kann auf diese Weise sehr schnell geprüft werden, ob die gewünschte Funktionalität weiterhin gegeben ist oder sich ein Fehler eingeschlichen hat.

7.2.1 Schreibtischtest

In dieser beispielhaften Darstellung werde ich die Funktion zum Anlegen eines Kunden einem Schreibtischtest unterziehen. Prüfen möchte ich im Speziellen das Fehlerhandling bei Eingabe der Postleitzahl. Die Testfälle, das erwartete Ergebnis und das reale Ergebnis sind in der nachfolgenden Tabelle 7.1 aufgelistet.

Testfall	**erwartetes Ergebnis**	**reales Ergebnis**
Eingabe 4-stelliger Ganzzahl: 4711	Fehler: „PLZ: 5-stellige Ganzzahl“	✓
Eingabe 6-stelliger Ganzzahl: 987654	Fehler: „PLZ: 5-stellige Ganzzahl“	✓
Eingabe float-Wert: 4.0	Fehler: „PLZ: 5-stellige Ganzzahl“	✓
Eingabe Zeichenkette „Hallo!“	Fehler: „PLZ: 5-stellige Ganzzahl“	✓
Eingabe 5-stelliger Ganzzahl: 73547	„Kunde erfolgreich angelegt“	✓

Tabelle 7.1: Testfalltabelle Schreibtischtest

Somit sind diese manuellen Testfälle alle positiv abgeprüft und haben den Schreibtischtest bestanden.

7.2.2 Unit-Test

Ein Unit-Test ist ein Aspekt des automatisierten Testens der im Rahmen der testgetriebenen Entwicklung (engl. test-driven development = TDD) eingesetzt wird. Er prüft einen kleinen Abschnitt des Programms anhand automatisierter Testdurchläufe auf Funktionalität und eventuelle Fehler. Als Modul für den Unit-Test habe ich *pytest*, als zu prüfende

Methode habe ich eine Division gewählt, die in der Controller-Klasse integriert ist. Der Quellcode der Methode sieht folgendermaßen aus:

```
def division_method(zaehler, nenner):
        return zaehler / nenner
```

Die zugehörige *pytest*-Datei enthält den folgenden Quellcode:

```
import pytest
from controller import Controller

def test_division_method_standard():
        assert Controller.division_method(10, 2) == 5
        assert Controller.division_method(-12, 4) == -3
        assert Controller.division_method(100, 100) == 1

def test_division_method_float():
        assert Controller.division_method(1, 4) == 0.25
        assert Controller.division_method(10, 2.0) == 5.0
        assert pytest.approx(Controller.division_method(10, 3), 0.01) == 3.33

def test_division_method_division_by_zero():
        with pytest.raises(ZeroDivisionError):
                Controller.division_method(5, 0)
```

Die Methode *test_division_method_standard()* prüft die Standard-Fälle der Division ab. Hier werden Ganzzahlen (auch mit negativem Vorzeichen) dividiert und das erwartete Ergebnis vermerkt. In der Methode *test_division_method_float()* werden Gleitkommawerte geprüft und gegen das erwartete Ergebnis verglichen. In der dritten Zeile wird eine Division vorgenommen, die als Ergebnis eine unendliche Zahl ergibt. Deshalb wird mit der pytest-Methode *approx()* das Ergebnis (..._method(10,3)) gekürzt (Toleranz von 0.01) und mit dem erwarteten Wert (3.33) verglichen. Die letzte Testmethode prüft auf die Division durch 0, die mathematisch nicht möglich ist. Hier wirft Python einen *ZeroDivisionError*, der abgeprüft werden soll.
Das Ergebnis nach Ausführen der Testdatei zeigt einen fehlerlosen Durchlauf des Unit-Tests, vgl. Abbildung 7.8

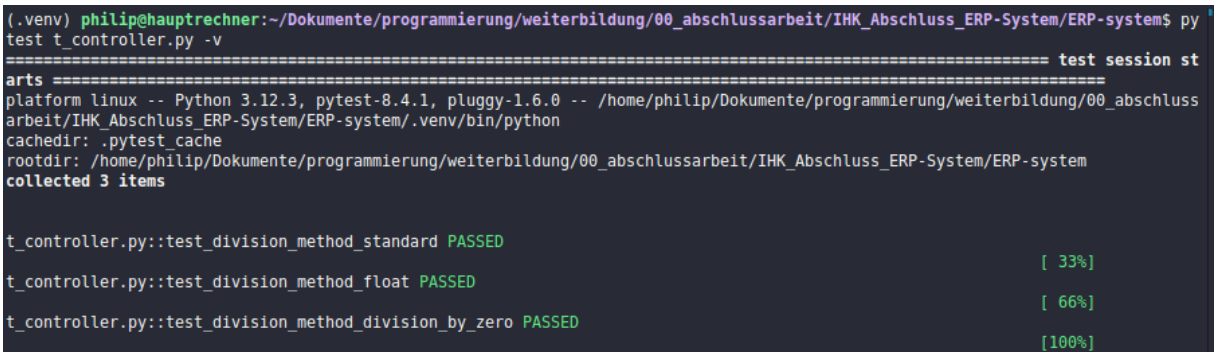

Abbildung 7.8: pytest Ergebnisfenster

Somit sind die vorher definierten Kriterien erfüllt und durch den Test bestätigt worden. Auf diese Weise kann bei Änderungen am Algorithmus der Methode auf einfache Art und Weise die Funktionsfähigkeit bestätigt werden.

8 Ergebnisse (Soll-Ist-Vergleich)

8.1 Realer Zeitaufwand

Der geplante Zeitaufwand wurde in Tabelle 5.1 aufgeführt. Nachfolgend stelle ich den realen Zeitaufwand dar, der für die Bearbeitung der Projektarbeit angefallen ist. Die Differenz zeigt den zeitlichen Unterschied zur Planung.

Arbeitspaket	**Zeitaufwand** [h]	**Differenz** [h]
Analyse	2,5	-0,5
Entwurf	7	0
Implementierung	35	+5
Testing	7	-3
Dokumentation	28	-2
Gesamt	**80**	**79,5**

Tabelle 8.1: Realer Zeitaufwand

Den zeitlichen Rahmen konnte ich insgesamt gut einhalten und hatte nur eine größere Abweichung bei der Implementierungsphase. Grund dafür ist der Einsatz des tkinter-Moduls und die damit einhergehende Einarbeitungsphase in die verschiedenen Bereiche dieses Moduls. Neben der Entwicklung der GUI und der dahinterliegenden Funktionen habe ich parallel immer wieder die Grenzfälle direkt abgeprüft, sodass ich die Testingphase effizienter gestalten konnte und sie sich dadurch etwas verkürzen ließ. Auch die Dokumentation konnte in dem vorgegebenen Rahmen von ca. 30 h erstellt werden.

8.2 Umsetzungsgrad

8.2.1 Use Cases

Einen großen Teil der formulierten Use Cases aus Tabelle 5.2 habe ich wie geplant umgesetzt. Die Use Cases habe ich entsprechend ihrer Priorisierung abgearbeitet und zuerst die Muss-Kriterien implementiert. Da sich in dem zeitlichen Rahmen nicht alles umsetzen ließ, sind die Use Cases mit den Kann- und Soll-Prioritäten nicht umgesetzt. Auch ist die Funktion der Versandbestätigung der Kundenaufträge noch nicht implementiert. Hier war ange-

dacht, dass zur jeweiligen Auftragsnummer ein Lieferschein in der entsprechenden Tabelle angelegt und dadurch der Versand bestätigt wird.

8.2.2 Potenziale

Bezogen auf den derzeitigen Funktionsumfang und die aktuelle Implementierung sehe ich noch Potenziale in der Gestaltung der GUI sowie dem Verhalten bei Anlage eines Auftrags.
Die GUI muss so überarbeitet werden, dass sich die Aufteilung der Spaltenbreite nicht kontextbezogen ändert, sondern ein einheitliches Rastermaß beibehalten wird. Ein zweiter Lösungsansatz wäre auch, die Benutzeroberfläche nicht statisch zu belassen, sondern bei Vergrößerung oder Verkleinerung des Fensters durch den Nutzer entsprechend dynamisch zu reagieren ("Responsive Design").
Bei Anlage eines Auftrags wird derzeit keine Änderung am Lagerbestand vorgenommen. Dies muss angepasst werden, da mit Anlage eines Auftrags auch eine Entnahme von Bauteilen aus dem Lager einhergeht. Dazu muss eine weitere Methode implementiert werden, die den aktuellen Lagerbestand um die entsprechende Stückzahl reduziert. Auch muss im Zuge der Anlage eines Auftrags abgeprüft werden, ob die entsprechende Anzahl an Bauteilen noch im Lager vorrätig ist. Sollte das nicht der Fall sein, muss eine entsprechende Fehlermeldung darauf hinweisen, damit der Disponent die Möglichkeit bekommt, einen solchen Fehlbestand aufzulisten und fehlende Ware zuzubuchen.

9 Fazit und Ausblick

9.1 Fazit

Zusammenfassend bin ich sehr zufrieden mit dem Verlauf der Projektarbeit. Ich konnte die verschiedenen Themen aus den Unterrichtseinheiten anwenden (Python, OOP, Datenbankentwurf, Projektmanagement, Testing) und mich intensiv mit Python als Programmiersprache befassen. Die Lernkurve was das GUI-Modul tkinter betrifft war sehr steil - sobald ich das Grundkonzept verstanden hatte, war ich in der Lage ohne große Unterstützung aus Online-Tutorials oder Fachliteratur die weiteren Widgets aufzubauen und im Hauptfenster zu positionieren. Der Datenbankentwurf und die Integration von SQLite fielen mir recht leicht. Das Anwenden des MVC-Patterns war anfangs eine Hürde, mit Fortschreiten der Projektarbeit ist es mir immer leichter gefallen, diese Trennung der Schichten beizubehalten.
Handlungsfelder sehe ich noch in der effizienten Erstellung des Quellcodes. Überraschend schwer fiel es mir, die Methoden- und Attribute-Namen nach den Clean Code-Regeln zu erstellen. Sprechende Namen, die ihre eigene Funktion kurz und prägnant beschreiben, sind schwer zu definieren. Dies konsequent durch den kompletten Quellcode aufrechtzuerhalten ist mir nicht an allen Stellen gelungen. Auch war ich bestrebt, den Quellcode so zu gestalten, dass er möglichst abstrahiert und wiederverwendbar gestaltet ist (DRY-Prinzip). Dies konnte ich nicht bei allen Methoden konsequent anwenden. Durch die anfänglichen Verständnisschwierigkeiten beim MVC-Pattern ist es mir bei den ersten Teilen der Implementierung nicht gelungen, die Schichten trennscharf voneinander zu separieren. Hier besteht noch Bedarf für Nacharbeit und einer Optimierung des Quellcodes. Bei der Ausarbeitung der Projektarbeit habe ich mich konsequent an den Ablauf im Wasserfallmodell gehalten und zuerst analysiert, geplant und entworfen bevor ich die ersten Zeilen Quellcode geschrieben habe. Mir wurde mit zunehmender Projektdauer klar, dass diese Schrittfolge unbedingt einzuhalten ist. Man hat dadurch einen klaren Plan, kennt das Ziel sowie die benötigten Funktionsumfänge und neigt nicht dazu, Zusatzfunktionen zu implementieren, die weder gefordert noch sinnvoll sind.
Dass ich nicht alle definierten Use Cases im Programm umsetzen konnte war mir gegen Ende der Entwurfsphase bereits klar. Das Ziel war von Anfang an ambitioniert formuliert, jedoch bin ich mit dem erreichten Ergebnis sehr zufrieden. Die bisher implementierten Funktionalitäten sind aufeinander abgestimmt, sie haben einen logischen Aufbau und ein entsprechendes Fehlerhandling für die Nutzereingaben. Das Ziel, dass die Nutzer des Pro-

gramms ein grundsätzliches Verständnis für die Abläufe in einem ERP-System bekommen, konnte ich erreichen. Interessierte Software-Entwickler haben hiermit eine stimmige Basis für eine Erweiterung der Funktionalitäten und eine Steigerung der Robustheit des Quellcodes. Die Erweiterungen der Funktionalitäten, um die definierten Use Cases zu erreichen, ist ein Ziel für die Zeit nach dem IHK-Zertifikatslehrgang. Auch der Einsatz einer KI als Unterstützung für die Quellcode-Erstellung ist in der Zukunft ein probates Mittel um effizienter zu werden.

9.2 Ausblick

Mit Blick auf die Zukunft muss die Software noch um einige Funktionen erweitert werden, welche sie für einen produktiven Einsatz in einem Unternehmen benötigt. Dazu zählen z.B. Funktionen um Ware manuell auszubuchen oder Funktionen wie Kundendaten und Aufträge bearbeiten oder löschen zu können. Des Weiteren ist es wichtig, einen Freigabeworkflow für eigene Zeichnungen oder externe Spezifikationen zu integrieren. Dadurch wird sichergestellt, dass alle beteiligten Parteien über Änderungen und den aktuellen Stand zu Sachnummern informiert sind. Hierbei bietet es sich an, einen zentralen Änderungsdienst zu integrieren. Dieser kann anhand von aktuellen Lagerbeständen und zukünftigen Bedarfen eine Aus- und Einlaufsteuerung von geänderten Bauteilen definieren und damit Lager- und Schrottkosten minimieren. Des Weiteren bietet es sich an, dass ein Portal implementiert wird, an welchem die Zulieferer angebunden sind und mit Informationen zu Änderungen, Abrufzahlen und Bedarfsplanungen versorgt werden. Als letzte Idee für eine Erweiterung schwebt mir noch vor, Logistikdienstleister ebenfalls mit anzubinden, sodass entweder direkt eine Lieferung beauftragt oder zumindest ein Versandetikett erstellt werden kann.

10 Anhang

10.1 Kundendokumentation

Der Nutzer kann das Programm grundsätzlich auf zwei verschieden Arten zur Anlage eines Auftrags nutzen. Auf Abbildung 10.1 sind die benötigten Schritte dargestellt.
Die einfachste Möglichkeit (linke Spalte) ist die, dass die gewünschte Sachnummer bereits besteht und lediglich der Kunde sowie der Auftrag im System angelegt werden müssen.
Die umfangreichere Möglichkeit (rechte Spalte) sieht vor, dass - zusätzlich zur Anlage des Kunden - noch eine neue Sachnummer angelegt sowie Lagerbestand von dieser aufgebaut werden muss bevor letztendlich der Auftrag generiert werden kann.

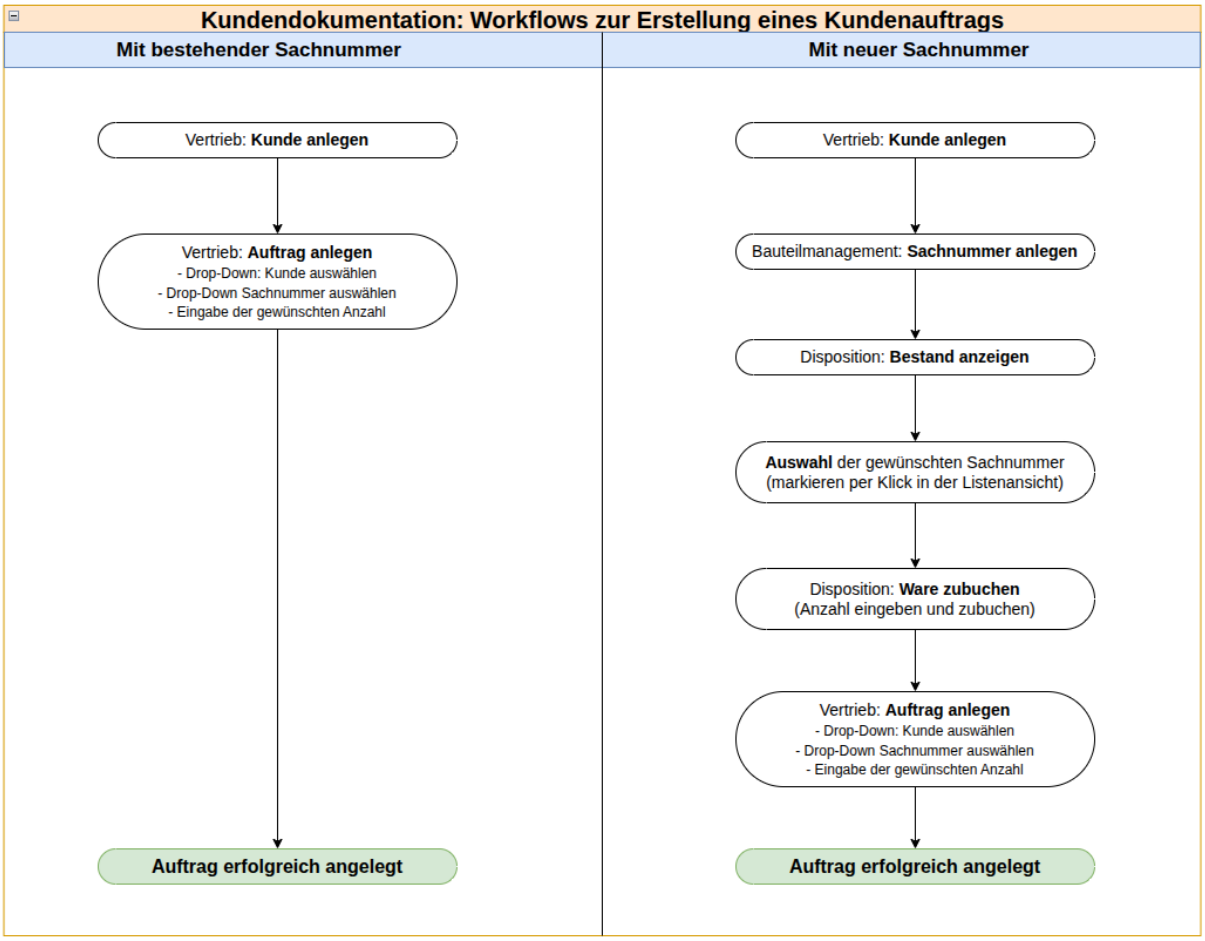

Abbildung 10.1: Kundendokumentation: Workflows zur Anlage eines Auftrags (mit draw.io erstellt)

10.2 Quellcode

10.2.1 model.py

```
# Autor: Philip Kottmann
# Datum: 7.7.2025
# Beschreibung: Model-Klasse

import sqlite3 # Datenbank-Modul

class Model:
    def __init__(self):
        with sqlite3.connect("erp_system.db") as self.connection:
            self.connection.execute("PRAGMA foreign_keys = ON")
            self.connection.commit()
            self.cursor = self.connection.cursor()

        # Tabelle "Sachnummern"
        self._create_table("""CREATE TABLE IF NOT EXISTS sachnummern
                           (sNrID INTEGER PRIMARY KEY AUTOINCREMENT,
                           materialnummer INTEGER,
                           bezeichnung TEXT,
                           kennungAufbauzustand TEXT,
                           stueckliste INTEGER,
                           warenwert INTEGER,
                           wiederbeschaffungszeit INTEGER)""")

        # Tabelle "Bestände"
        self._create_table("""CREATE TABLE IF NOT EXISTS bestaende
                           (bestID INTEGER PRIMARY KEY AUTOINCREMENT,
                           sNrID INTEGER NOT NULL,
                           anzahl INTEGER NOT NULL,
                            FOREIGN KEY (sNrID)
                                REFERENCES sachnummern (sNrID))""")

        # Tabelle "Kunden"
        self._create_table("""CREATE TABLE IF NOT EXISTS kunden
                           (kundID INTEGER PRIMARY KEY AUTOINCREMENT,
                           firmenname TEXT,
                           strasse TEXT,
                           hausnummer INTEGER,
                           plz INTEGER,
                           ortsname TEXT,
                           telefon TEXT,
```

```
41                          email TEXT)""")
42
43          # Tabelle "Lieferscheine"
44          self._create_table("""CREATE TABLE IF NOT EXISTS lieferscheine
45                          (lieferID INTEGER PRIMARY KEY AUTOINCREMENT,
46                          versanddatum TEXT)""")
47
48          # Tabelle "Auftraege"
49          self._create_table("""CREATE TABLE IF NOT EXISTS auftraege
50                          (aufID INTEGER PRIMARY KEY AUTOINCREMENT,
51                          auftragseingang TEXT,
52                          auftragsabschluss TEXT,
53                          kundID INTEGER,
54                          lieferID INTEGER,
55                           FOREIGN KEY (kundID)
56                               REFERENCES kunden (kundID),
57                           FOREIGN KEY (lieferID)
58                               REFERENCES lieferscheine (lieferID))""")
59
60          # Tabelle "Auftragspositionen"
61          self._create_table("""CREATE TABLE IF NOT EXISTS auftragspositionen
62                          (auftrPosID INTEGER PRIMARY KEY AUTOINCREMENT,
63                          aufID INTEGER,
64                          sNrID INTEGER,
65                          anzahl INTEGER,
66                           FOREIGN KEY (aufID)
67                               REFERENCES auftraege (aufID),
68                           FOREIGN KEY (sNrID)
69                               REFERENCES sachnummern (sNrID))""")
70
71
72      def _create_table(self, sql):
73          self.cursor.execute(sql)
74          self.connection.commit()
75
76      def read_database(self):
77          self.cursor.execute("SELECT * FROM sachnummern ORDER BY materialnummer
                ASC")
78          results = self.cursor.fetchall()
79
80      def read_all_from_database(self, table):
81          self.cursor.execute(f"SELECT * FROM {table}")
82          self.results = self.cursor.fetchall()
83          return self.results
```

```
84
85      def read_highest_ID_from_sachnummern(self):
86          self.cursor.execute("SELECT sNrID FROM sachnummern ORDER BY sNrID DESC
                LIMIT 1")
87          last_ids = self.cursor.fetchone()
88          if last_ids is None:
89              return 0
90          else:
91              for last_id in last_ids:
92                  return int(last_id)
93
94      def read_context_partnumber(self, partnumber):
95          self.partnumber = partnumber
96          self.cursor.execute(f"SELECT * FROM (sachnummern, bestaende) WHERE
               sachnummern.sNrID = {self.partnumber} AND bestaende.sNrID = {self.
               partnumber}")
97          self.result = self.cursor.fetchall()
98          return self.result
99
100     def read_shortages(self):
101         pass
102
103     def read_stock(self):
104         # self.cursor.execute(f"SELECT sachnummern.sNrID, sachnummern.
               materialnummer, sachnummern.bezeichnung, bestaende.anzahl FROM (
               sachnummern, bestaende) WHERE sachnummern.sNrID = bestaende.sNrID
               ")
105         self.cursor.execute(f"SELECT sachnummern.sNrID, sachnummern.
               materialnummer, sachnummern.bezeichnung, bestaende.anzahl FROM (
               sachnummern, bestaende) WHERE sachnummern.sNrID = bestaende.sNrID"
               )
106         self.result = self.cursor.fetchall()
107         return self.result
108
109     def calculate_partnumber(self):
110         basic_number = 10000    # Start der Materialnummernvergabe bei 10000
111         id_to_add = self.read_highest_ID_from_sachnummern()
112         return (basic_number + id_to_add + 1)
113
114     def add_customer(self, customer_data):
115         self.customer = customer_data
116         sql = "INSERT INTO kunden (firmenname, strasse, hausnummer, plz,
               ortsname, telefon, email) VALUES (?,?,?,?,?,?,?)"
117         self.cursor.execute(sql, self.customer)
```

```
118         self.connection.commit()
119
120     def add_partnumber(self, partnumber_data):
121         self.partnumber_data = partnumber_data
122         sql = "INSERT INTO sachnummern (materialnummer, bezeichnung,
                kennungAufbauzustand, stueckliste, warenwert,
                wiederbeschaffungszeit) VALUES (?,?,?,?,?,?)"
123         self.cursor.execute(sql, self.partnumber_data)
124         self.connection.commit()
125         self.last_id = self.cursor.lastrowid
126         sql = "INSERT INTO bestaende (sNrID, anzahl) VALUES (?,?)"
127         self.cursor.execute(sql, [self.last_id, "0"])
128         self.connection.commit()
129
130     def get_current_quantity(self, partnumber):
131         sql = f"SELECT anzahl FROM bestaende WHERE bestaende.sNrID = {
                partnumber}"
132         self.cursor.execute(sql)
133         result = self.cursor.fetchone()
134         return result
135
136     def add_quantity(self, partnumber, quantity):
137         read_current_quantity = self.get_current_quantity(partnumber)
138         for current_quantity in read_current_quantity:
139             current_quantity = current_quantity
140         quantity_to_add = quantity
141         new_quantity = current_quantity + quantity_to_add
142         sql = f"UPDATE bestaende SET anzahl = {new_quantity} WHERE sNrID = {
                partnumber}"
143         self.cursor.execute(sql)
144         self.connection.commit()
145
146     def add_order(self, kundID, sNrID, quantity):
147         sql_orders = f"INSERT INTO auftraege (auftragseingang,
                auftragsabschluss, kundID, lieferID) VALUES (date('now'), NULL, {
                kundID}, NULL)"
148         lastID = self.cursor.execute(sql_orders).lastrowid
149
150         sql_orders = f"INSERT INTO auftragspositionen (aufID, sNrID, anzahl)
                VALUES ({lastID}, {sNrID}, {quantity})"
151         self.cursor.execute(sql_orders)
152
153         self.connection.commit()
154         return lastID
```

```
155
156     def select_customers_for_order(self):
157         sql = "SELECT kundID, firmenname FROM kunden ORDER BY firmenname"
158         self.cursor.execute(sql)
159         result = self.cursor.fetchall()
160         return result
161
162     def select_partnumbers_for_order(self):
163         sql = "SELECT sNrID, materialnummer, bezeichnung FROM sachnummern"
164         self.cursor.execute(sql)
165         result = self.cursor.fetchall()
166         return result
```

Listing 10.1: model.py

10.2.2 view.py

```python
# Autor: Philip Kottmann
# Datum: 7.7.2025
# Beschreibung: View-Klasse

import tkinter as tk # tkinter-GUI-Modul
import re # regular expressions
from tkinter import ttk # ttk-Widgets
# Funktionalität Popup-Fenster
from tkinter.messagebox import showinfo, showwarning, showerror


class View(tk.Tk):
    def __init__(self, controller):
        super().__init__()

        # Aussenabstand allgemein
        self.PAD = 7

        # Konstante Fachbereiche
        self.DEPARTMENTS = ["Vertrieb", "Disposition", "Bauteilmanagement"]

        # Definition der Laengen- und Breitenmasse des Hauptfensters:
        self.WINDOW_SIZE_TOTAL_WIDTH = 1200
        self.WINDOW_SIZE_TOTAL_HEIGHT = 800
        self.FOOTER_HEIGHT = 30
        self.main_view_height = self.WINDOW_SIZE_TOTAL_HEIGHT - self.
            FOOTER_HEIGHT

        # Spaltenbreite
        self.LEFT_COLUMN_WIDTH = 275
        self.remaining_width_without_left_column = self.
            WINDOW_SIZE_TOTAL_WIDTH - self.LEFT_COLUMN_WIDTH
        self.center_column_width = self.remaining_width_without_left_column/2
        self.right_column_width = self.remaining_width_without_left_column/2

        # Breite der Frames
        self.left_frame_width = self.LEFT_COLUMN_WIDTH - self.PAD
        self.center_frame_width = self.center_column_width - self.PAD
        self.right_frame_width = self.right_column_width - self.PAD

        # Klassenattribute
        self.controller = controller
```

```
41          self.next_partnumber = 0
42          self.customer_data = []
43
44          # Hauptfenster wird per Unter-Methode "zusammengebaut"
45          self._create_main_window()
46
47      # main()-Methode, die das Fenster per mainloop() in Dauerschleife
            offenhält
48      def main(self):
49          self.mainloop()
50
51      def _create_main_window(self):
52          self.title("ERP-System v1.0")
53          self.geometry(str(self.WINDOW_SIZE_TOTAL_WIDTH)+"x"+str(self.
                WINDOW_SIZE_TOTAL_HEIGHT))
54          self.maxsize(self.WINDOW_SIZE_TOTAL_WIDTH, self.
                WINDOW_SIZE_TOTAL_HEIGHT)
55          self.resizable(0,0)
56
57          self.grid_rowconfigure(0, weight=1)
58          self.grid_rowconfigure(1, weight=0)
59          self.grid_columnconfigure(0, weight=0)
60          self.grid_columnconfigure((1,2), weight=1)
61          self.grid_propagate(False)
62
63          self._create_left_frame()
64          self._create_center_frame()
65          self._create_right_frame()
66          self._create_footer_fame()
67          self.create_listview(0, "", "")
68
69      def _create_left_frame(self):
70          self.left_frame = ttk.Frame(self, relief="groove", width=self.
                left_frame_width)
71          self.left_frame.grid(row=0, column=0, sticky="nesw", padx=self.PAD)
72          self._create_sections()
73
74      def _create_center_frame(self):
75          self.center_frame = ttk.Frame(self, relief="groove", width=self.
                center_frame_width)
76          self.center_frame.grid(row=0, column=1, sticky="nesw", padx=self.PAD)
77
78      def _create_right_frame(self):
```

```
79          self.right_frame = ttk.Frame(self, relief="groove", width=self.
                right_frame_width)
80          self.right_frame.grid(row=0, column=2, sticky="nesw", padx=self.PAD)
81
82      def _create_footer_fame(self):
83          self.footer_frame = ttk.Frame(self, relief="flat", padding=2)
84          self.footer_frame.grid(row=1, column=0, sticky="nsew")
85          self._create_copyright()
86          self._end_application()
87
88      # Erzeuge Abschnitte mit ueberschriften und Buttons:
89      def _create_sections(self):
90          for i in range(3):
91              section = ttk.Frame(self.left_frame, relief="ridge")
92              section.grid(row=i, column=0, sticky="EW", pady=self.PAD, padx=
                    self.PAD)
93              section_label = ttk.Label(section, text=self.DEPARTMENTS[i], font
                    =("Segoe UI", 10, "bold"))
94              section_label.grid(row=0, column=0, sticky="nesw", padx=self.PAD,
                    pady=self.PAD)
95
96              if i == 0:
97                  department_counter = 5 # Anzahl Buttons fuer Vertrieb
98              elif i == 1:
99                  department_counter = 4 # Anzahl Buttons fuer Disposition
100             elif i == 2:
101                 department_counter = 3 # Anzahl an buttons fuer
                        Bauteilmanagement
102
103             for j in range(department_counter):
104                 if i == 0 and j == 0:
105                     button = ttk.Button(section, text="Kunde anlegen", command
                            =self.add_customer_popup)
106                 elif i == 0 and j == 1:
107                     button = ttk.Button(section, text="Auftrag anlegen",
                            command=self.add_order_popup)
108                 elif i == 0 and j == 2:
109                     button = ttk.Button(section, text="Auftragsbestätigung*",
                            command=self.controller.confirm_delivery)
110                 elif i == 0 and j == 3:
111                     button = ttk.Button(section, text="Jahresumsatz Kunde*",
                            command=self.controller.sales_yearly_customer)
112                 elif i == 0 and j == 4:
```

```
113                     button = ttk.Button(section, text="Jahresumsatz Sachnummer
                            *", command=self.controller.sales_yearly_partnumber)
114                 elif i == 1 and j == 0:
115                     button = ttk.Button(section, text="Bestand anzeigen",
                            command=self.controller.show_stock)
116                 elif i == 1 and j == 1:
117                     button = ttk.Button(section, text="Ware zubuchen", command
                            =self.add_quantity)
118                 elif i == 1 and j == 2:
119                     button = ttk.Button(section, text="Fehlbestände*", command
                            =self.controller.show_shortage)
120                 elif i == 1 and j == 3:
121                     button = ttk.Button(section, text="Mengenplanung*",
                            command=self.controller.show_quantities_needed)
122                 elif i == 2 and j == 0:
123                     button = ttk.Button(section, text="Sachnummer anlegen",
                            command=self.add_partnumber_popup)
124                 elif i == 2 and j == 1:
125                     button = ttk.Button(section, text="Stammdaten pflegen*",
                            command=self.controller.modify_core_data)
126                 elif i == 2 and j == 2:
127                     button = ttk.Button(section, text="Sachnummer löschen*",
                            command=self.controller.modify_bom)
128                 else:
129                     button = ttk.Button(section, text=f"Button {i+1}-{j+1}")
130                 button.grid(row=j+1, column=0, sticky="EW", padx=self.PAD,
                        pady=self.PAD)
131
132     def create_listview(self, content, heading, category):
133         self.content = content
134         category = category
135
136         if self.content == 0:
137             self.listbox_label = ttk.Label(self.center_frame, text="Bitte
                    Auswahl links treffen", font=("Segoe UI", 10, "bold"))
138         else:
139             self.listbox_label = ttk.Label(self.center_frame, text=(heading +
                    " (Doppelklick für Details)"), font=("Segoe UI", 10, "bold"))
140         self.listbox_label.grid(row=0, column=0, sticky="EW", padx=self.PAD,
                pady=(self.PAD * 2))
141
142         self.listbox = tk.Listbox(self.center_frame, relief="ridge")
143         self.listbox.config(font=("Courier", 10))
```

```
144         self.listbox.grid(row=1, column=0, sticky="EW", padx=self.PAD, pady=
                self.PAD)
145 
146         if self.content == 0:
147             self.listbox.insert(0, "")
148         else:
149             heading_listbox = f"S-Nr.|Bezeichnung | {category}" # "Überschrift
                    " der Listview-Spalten
150             self.listbox.insert(0, heading_listbox)
151 
152             for item in self.content:
153                 row = f"{item[1]}|{item[2]}|{item[3]}"
154                 #self.listbox.insert(tk.END, item[1:])
155                 self.listbox.insert(tk.END, row)
156 
157         self.listbox.bind('<Double-Button-1>', self.get_cursor_title_double)
158         # self.listbox.bind("<<ListboxSelect>>", self.get_cursor_title_single)
159 
160     # Auslesen des Zeilenindex bei Doppelklick
161     def get_cursor_title_double(self, event):
162         selection = self.listbox.curselection()
163         if selection:
164             index = selection[0]-1
165             if index < 0:
166                 return
167             else:
168                 partnumber = self.content[index]
169                 if isinstance(partnumber, str):
170                     self.show_message_box("Auswahl", "Bitte Auswahl unten
                            treffen")
171                     return
172 
173                 else:
174                     self.show_context_partnumbers(partnumber)
175 
176     # Auslesen des Zeilenindex bei einfachem Klick ("Markieren")
177     def get_cursor_title_single(self, event):
178         selection = self.listbox.curselection()
179         if selection:
180             index = selection[0]-1
181             if index < 0:
182                 return
183             else:
184                 partnumber = self.content[index]
```

```
185                 if isinstance(partnumber, str):
186                     self.show_message_box("Auswahl", "Bitte Auswahl unten
                            treffen")
187                     return
188 
189                 else:
190                     self.show_context_partnumbers(partnumber)
191 
192     # Anzeige der Stammdaten der Sachnummer im rechten Frame
193     def show_context_partnumbers(self, partnumber):
194         self.result = self.controller.read_context_partnumber(partnumber[0])
195 
196         self.lbl_heading = ttk.Label(self.right_frame, text="Detailansicht",
                font=("Segoe UI", 10, "bold"))
197         self.lbl_heading.grid(row=0, column=0, columnspan=2, sticky="EW", padx
                =self.PAD, pady=(self.PAD * 2))
198 
199         self.lbl_partnumber_heading = ttk.Label(self.right_frame, text="
                Sachnummer: ", font=("Segoe UI", 10))
200         self.lbl_partnumber_heading.grid(row=1, column=0, sticky="EW", padx=
                self.PAD, pady=(self.PAD * 2))
201         self.lbl_partnumber = ttk.Label(self.right_frame, text=self.result
                [0][1], font=("Segoe UI", 10, "bold"))
202         self.lbl_partnumber.grid(row=1, column=1, sticky="EW", padx=self.PAD,
                pady=(self.PAD * 2))
203 
204         self.lbl_description_heading = ttk.Label(self.right_frame, text="
                Benennung: ", font=("Segoe UI", 10))
205         self.lbl_description_heading.grid(row=2, column=0, sticky="EW", padx=
                self.PAD, pady=(self.PAD * 2))
206         self.lbl_description = ttk.Label(self.right_frame, text=self.result
                [0][2], font=("Segoe UI", 10, "bold"))
207         self.lbl_description.grid(row=2, column=1, sticky="EW", padx=self.PAD,
                 pady=(self.PAD * 2))
208 
209         self.lbl_build_condition_heading = ttk.Label(self.right_frame, text="
                Aufbauzustand: ", font=("Segoe UI", 10))
210         self.lbl_build_condition_heading.grid(row=3, column=0, sticky="EW",
                padx=self.PAD, pady=(self.PAD * 2))
211         self.lbl_build_condition = ttk.Label(self.right_frame, text=self.
                result[0][3], font=("Segoe UI", 10, "bold"))
212         self.lbl_build_condition.grid(row=3, column=1, sticky="EW", padx=self.
                PAD, pady=(self.PAD * 2))
213 
```

```
214         self.lbl_value_heading = ttk.Label(self.right_frame, text="Warenwert
                €[]: ", font=("Segoe UI", 10))
215         self.lbl_value_heading.grid(row=4, column=0, sticky="EW", padx=self.
                PAD, pady=(self.PAD * 2))
216         self.lbl_value = ttk.Label(self.right_frame, text=self.result[0][5],
                font=("Segoe UI", 10, "bold"))
217         self.lbl_value.grid(row=4, column=1, sticky="EW", padx=self.PAD, pady
                =(self.PAD * 2))
218
219         self.lbl_stock_heading = ttk.Label(self.right_frame, text="
                Lagerbestand [Stk.]: ", font=("Segoe UI", 10))
220         self.lbl_stock_heading.grid(row=5, column=0, sticky="EW", padx=self.
                PAD, pady=(self.PAD * 2))
221         self.lbl_stock = ttk.Label(self.right_frame, text=self.result[0][9],
                font=("Segoe UI", 10, "bold"))
222         self.lbl_stock.grid(row=5, column=1, sticky="EW", padx=self.PAD, pady
                =(self.PAD * 2))
223
224         self.lbl_replacement_time_heading = ttk.Label(self.right_frame, text="
                Wiederbeschaffungszeit [KW]: ", font=("Segoe UI", 10))
225         self.lbl_replacement_time_heading.grid(row=6, column=0, sticky="EW",
                padx=self.PAD, pady=(self.PAD * 2))
226         self.lbl_replacement_time = ttk.Label(self.right_frame, text=self.
                result[0][6], font=("Segoe UI", 10, "bold"))
227         self.lbl_replacement_time.grid(row=6, column=1, sticky="EW", padx=self
                .PAD, pady=(self.PAD * 2))
228
229     # Popup um Stammdaten zu ändern
230     def modify_partnumber_popup(self):
231         pop_up_win = tk.Toplevel()
232         pop_up_win.title("Stammdaten")
233         pop_up_win.geometry("250x150")
234         pop_up_win.resizable(0,0)
235         label = tk.Label(pop_up_win, text="Stammdaten modifizieren")
236         label.grid(row=0, column=0, columnspan=2, sticky="W", padx=self.PAD,
                pady=self.PAD)
237
238         self.next_partnumber = self.controller.get_next_partnumber()
239         lbl_partnumber = tk.Label(pop_up_win, text="Sachnummer: ", )
240         lbl_partnumber.grid(row=1, column=0, sticky="W", padx=self.PAD, pady=
                self.PAD)
241         partnumber = tk.Label(pop_up_win, text=str(self.next_partnumber))
242         partnumber.grid(row=1, column=1, sticky="W", padx=self.PAD, pady=self.
                PAD)
```

```
243
244         button_close = tk.Button(pop_up_win, text="Abbrechen", command=
                pop_up_win.destroy)
245         button_close.grid(row=2, column=0, sticky="EW", padx=self.PAD, pady=
                self.PAD)
246
247         button_change = tk.Button(pop_up_win, text="Ändern", command=
                pop_up_win.destroy)
248         button_change.grid(row=2, column=1, sticky="EW", padx=self.PAD, pady=
                self.PAD)
249
250     # Popup um Kundendaten einzutragen
251     def add_customer_popup(self):
252         self.company_name = tk.StringVar
253         self.company_street = tk.StringVar
254         self.company_number = tk.IntVar
255         self.company_zip = tk.IntVar
256         self.company_city = tk.StringVar
257         self.company_phone = tk.StringVar
258         self.company_email = tk.StringVar
259
260         self.pop_up_win = tk.Toplevel()
261         self.pop_up_win.title("Kunde anlegen")
262         self.pop_up_win.geometry("500x280")
263         self.pop_up_win.resizable(0,0)
264         self.label = tk.Label(self.pop_up_win, text="Neuen Kunden anlegen")
265         self.label.grid(row=0, column=0, columnspan=2, sticky="W", padx=self.
                PAD, pady=self.PAD)
266
267         # Firmenname
268         self.lbl_company_name = tk.Label(self.pop_up_win, text="Firmenname: ")
269         self.lbl_company_name.grid(row=1, column=0, sticky="W", padx=self.PAD,
                 pady=self.PAD)
270         self.entry_company_name = ttk.Entry(self.pop_up_win, justify="left",
                textvariable=self.company_name)
271         self.entry_company_name.grid(row=1, column=1, columnspan=3, sticky="EW
                ")
272
273         # Strasse
274         self.lbl_company_street = tk.Label(self.pop_up_win, text="Straße: ")
275         self.lbl_company_street.grid(row=2, column=0, sticky="W", padx=self.
                PAD, pady=self.PAD)
276         self.entry_company_street = ttk.Entry(self.pop_up_win, justify="left",
                 textvariable=self.company_street)
```

```
277         self.entry_company_street.grid(row=2, column=1, sticky="W")
278 
279         # Hausnummer
280         self.lbl_company_number = tk.Label(self.pop_up_win, text="Nr: ")
281         self.lbl_company_number.grid(row=2, column=2, sticky="W", padx=self.
                PAD, pady=self.PAD)
282         self.entry_company_number = ttk.Entry(self.pop_up_win, justify="left",
                 width=5, textvariable=self.company_number)
283         self.entry_company_number.grid(row=2, column=3, sticky="EW")
284 
285         # PLZ
286         self.lbl_company_zip = tk.Label(self.pop_up_win, text="PLZ: ")
287         self.lbl_company_zip.grid(row=3, column=0, sticky="W", padx=self.PAD,
                pady=self.PAD)
288         self.entry_company_zip = ttk.Entry(self.pop_up_win, justify="left",
                textvariable=self.company_zip)
289         self.entry_company_zip.grid(row=3, column=1, sticky="EW")
290 
291         # Ort
292         self.lbl_company_city = tk.Label(self.pop_up_win, text="Ort: ")
293         self.lbl_company_city.grid(row=3, column=2, sticky="W", padx=self.PAD,
                 pady=self.PAD)
294         self.entry_company_city = ttk.Entry(self.pop_up_win, justify="left",
                textvariable=self.company_city)
295         self.entry_company_city.grid(row=3, column=3, sticky="EW")
296 
297         # Telefon
298         self.lbl_company_phone = tk.Label(self.pop_up_win, text="Telefon: ")
299         self.lbl_company_phone.grid(row=4, column=0, sticky="W", padx=self.PAD
                , pady=self.PAD)
300         self.entry_company_phone = ttk.Entry(self.pop_up_win, justify="left",
                textvariable=self.company_phone)
301         self.entry_company_phone.grid(row=4, column=1, columnspan=3, sticky="
                EW")
302 
303         # E-Mail
304         self.lbl_company_email = tk.Label(self.pop_up_win, text="E-Mail: ")
305         self.lbl_company_email.grid(row=5, column=0, sticky="W", padx=self.PAD
                , pady=self.PAD)
306         self.entry_company_email = ttk.Entry(self.pop_up_win, justify="left",
                textvariable=self.company_email)
307         self.entry_company_email.grid(row=5, column=1, columnspan=3, sticky="
                EW")
308 
```

```
309         self.btn_insert = tk.Button(self.pop_up_win, text="Eintragen", command
                =self.insert_customer)
310         self.btn_insert.grid(row=6, column=0, sticky="EW", padx=self.PAD, pady
                =self.PAD)
311 
312         self.button_change = tk.Button(self.pop_up_win, text="Abbrechen",
                command=self.pop_up_win.destroy)
313         self.button_change.grid(row=6, column=1, sticky="EW", padx=self.PAD,
                pady=self.PAD)
314 
315     # Kundendaten in Datenbank schreiben
316     def insert_customer(self):
317         customer_data = [self.entry_company_name.get().strip(),
318                          self.entry_company_street.get().strip(),
319                          self.entry_company_number.get(),
320                          self.entry_company_zip.get(),
321                          self.entry_company_city.get().strip(),
322                          self.entry_company_phone.get(),
323                          self.entry_company_email.get().strip()]
324 
325         # Abfangen von Fehleingaben
326         try:
327             if int(self.entry_company_zip.get()) and (len(str(self.
                    entry_company_zip.get())) == 5):
328                 if re.match(r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$
                        ', self.entry_company_email.get().strip()):
329                     if self.controller.add_customer(customer_data):
330                         self.pop_up_win.destroy()
331                         tk.messagebox.showinfo("Erfolg", "Kunde erfolgreich
                                angelegt!")
332                     else:
333                         tk.messagebox.showerror("Fehler", "Keine Kundendaten
                                vorhanden!")
334                         self.pop_up_win.destroy()
335                 else:
336                     tk.messagebox.showwarning("Fehleingabe", "Falsches E-Mail-
                            Format\n>>test@example.com")
337             else:
338                 tk.messagebox.showwarning("Fehleingabe", "PLZ: 5-stellige
                        Ganzzahl")
339         except ValueError:
340             tk.messagebox.showwarning("Fehleingabe", "PLZ: 5-stellige Ganzzahl
                ")
341 
```

```
342     # Popup um Sachnummer anzulegen
343     def add_partnumber_popup(self):
344         self.partnumber = self.controller.get_next_partnumber()
345         self.description = tk.StringVar
346         self.build_up_status = ["Einzelteil", "Fertigteil"]
347         self.bom = ["Ja", "Nein"]
348         self.piece_price = tk.IntVar
349         self.sourcing_time = tk.IntVar
350 
351         self.pop_up_win = tk.Toplevel()
352         self.pop_up_win.title("Sachnummer anlegen")
353         self.pop_up_win.geometry("380x310")
354         self.pop_up_win.resizable(0,0)
355         self.label = tk.Label(self.pop_up_win, text="Neue Sachnummer anlegen")
356         self.label.grid(row=0, column=0, columnspan=2, sticky="W", padx=self.
                PAD, pady=self.PAD)
357 
358         # Sachnummer - automatisch generiert
359         self.lbl_partnumber = tk.Label(self.pop_up_win, text="Sachnummer: ")
360         self.lbl_partnumber.grid(row=1, column=0, sticky="W", padx=self.PAD,
                pady=self.PAD)
361         self.entry_partnumber_auto = ttk.Label(self.pop_up_win, text=f"{self.
                partnumber}\t(autom. generiert)")
362         self.entry_partnumber_auto.grid(row=1, column=1, columnspan=2, sticky=
                "EW")
363 
364         # Bezeichnung
365         self.lbl_description = tk.Label(self.pop_up_win, text="Bezeichnung: ")
366         self.lbl_description.grid(row=2, column=0, sticky="W", padx=self.PAD,
                pady=self.PAD)
367         self.entry_description = ttk.Entry(self.pop_up_win, justify="left",
                textvariable=self.description)
368         self.entry_description.grid(row=2, column=1, columnspan=2, sticky="EW"
                )
369 
370         # Aufbauzustand
371         self.lbl_build_up_status = tk.Label(self.pop_up_win, text="Aufbau: ")
372         self.lbl_build_up_status.grid(row=3, column=0, sticky="W", padx=self.
                PAD, pady=self.PAD)
373         self.entry_menu_build_up_status = ttk.Combobox(self.pop_up_win, values
                =self.build_up_status)
374         self.entry_menu_build_up_status.set("Bitte Auswahl treffen")
375         self.entry_menu_build_up_status.grid(row=3, column=1, columnspan=2,
                sticky="EW")
```

```
376
377         # Stückliste
378         self.lbl_bom = tk.Label(self.pop_up_win, text="Stückliste: ")
379         self.lbl_bom.grid(row=4, column=0, sticky="W", padx=self.PAD, pady=
                self.PAD)
380         self.entry_menu_bom_status = ttk.Combobox(self.pop_up_win, values=self
                .bom)
381         self.entry_menu_bom_status.set("Bitte Auswahl treffen")
382         self.entry_menu_bom_status.grid(row=4, column=1, columnspan=2, sticky=
                "EW")
383
384         # Warenwert
385         self.lbl_piece_price = tk.Label(self.pop_up_win, text="Warenwert €[]:
                ")
386         self.lbl_piece_price.grid(row=5, column=0, sticky="W", padx=self.PAD,
                pady=self.PAD)
387         self.entry_piece_price = ttk.Entry(self.pop_up_win, justify="left",
                textvariable=self.piece_price)
388         self.entry_piece_price.grid(row=5, column=1, columnspan=3, sticky="EW"
                )
389
390         # Wiederbeschaffungszeit
391         self.lbl_sourcing_time = tk.Label(self.pop_up_win, text="
                Beschaffungszeit [KW]: ")
392         self.lbl_sourcing_time.grid(row=6, column=0, sticky="W", padx=self.PAD
                , pady=self.PAD)
393         self.entry_sourcing_time = ttk.Entry(self.pop_up_win, justify="left",
                textvariable=self.sourcing_time)
394         self.entry_sourcing_time.grid(row=6, column=1, columnspan=3, sticky="
                EW")
395
396         self.btn_insert = tk.Button(self.pop_up_win, text="Eintragen", command
                =self.insert_partnumber)
397         self.btn_insert.grid(row=7, column=0, sticky="EW", padx=self.PAD, pady
                =self.PAD)
398
399         self.button_change = tk.Button(self.pop_up_win, text="Abbrechen",
                command=self.pop_up_win.destroy)
400         self.button_change.grid(row=7, column=1, sticky="EW", padx=self.PAD,
                pady=self.PAD)
401
402     # Sachnummer in Datenbank eintragen
403     def insert_partnumber(self):
404         partnumber_data = [
```

```
405             self.partnumber,
406             self.entry_description.get().strip(),
407             self.entry_menu_build_up_status.get(),
408             self.entry_menu_bom_status.get(),
409             self.entry_piece_price.get().strip(),
410             self.entry_sourcing_time.get().strip()]
411 
412         # Abfangen von Fehleingaben
413         try:
414             if int(self.entry_piece_price.get().strip()):
415                 try:
416                     if int(self.entry_sourcing_time.get().strip()):
417                         self.controller.add_partnumber(partnumber_data)
418                         self.pop_up_win.destroy()
419                         tk.messagebox.showinfo("Erfolg", "Sachnummer
                                erfolgreich eingetragen!")
420                 except ValueError:
421                     tk.messagebox.showwarning("Fehleingabe", "Beschaffungszeit
                            muss eine Ganzzahl sein")
422         except ValueError:
423             tk.messagebox.showwarning("Fehleingabe", "Warenwert muss eine
                Ganzzahl sein")
424 
425     # Ware zum bestand zubuchen
426     def add_quantity(self):
427         selection = self.listbox.curselection()
428         if selection:
429             index = selection[0]-1
430             if index < 0:
431                 tk.messagebox.showwarning("Fehler", "Bitte Sachnummer aus
                    Liste wählen")
432                 return
433             else:
434                 self.partnumber = self.content[index]
435         else:
436             tk.messagebox.showwarning("Fehler", "Bitte zuerst Auswahl in
                Bestand treffen")
437             return
438 
439         self.current_quantity = self.controller.get_quantity(self.partnumber
            [0])
440         self.new_quantity = tk.IntVar
441 
442         self.pop_up_win = tk.Toplevel()
```

```
443         self.pop_up_win.title("Ware zubuchen")
444         self.pop_up_win.geometry("350x200")
445         self.pop_up_win.resizable(0,0)
446         self.label = tk.Label(self.pop_up_win, text="Lieferung einbuchen")
447         self.label.grid(row=0, column=0, columnspan=2, sticky="W", padx=self.
                PAD, pady=self.PAD)
448 
449         # Sachnummer anzeigen - automatisch generiert
450         self.lbl_partnumber = tk.Label(self.pop_up_win, text="Sachnummer: ")
451         self.lbl_partnumber.grid(row=1, column=0, sticky="W", padx=self.PAD,
                pady=self.PAD)
452         self.entry_partnumber_auto = ttk.Label(self.pop_up_win, text=f"{self.
                partnumber[1]}")
453         self.entry_partnumber_auto.grid(row=1, column=1, columnspan=2, sticky=
                "W")
454 
455         self.lbl_current_quantity = tk.Label(self.pop_up_win, text="aktueller
                Bestand: ")
456         self.lbl_current_quantity.grid(row=2, column=0, sticky="W", padx=self.
                PAD, pady=self.PAD)
457 
458         self.entry_current_quantity = tk.Label(self.pop_up_win, text=self.
                current_quantity)
459         self.entry_current_quantity.grid(row=2, column=1, sticky="W", padx=
                self.PAD, pady=self.PAD)
460 
461         self.lbl_new_quantity = tk.Label(self.pop_up_win, text="Zuzubuchende
                Anzahl: ")
462         self.lbl_new_quantity.grid(row=3, column=0, sticky="W", padx=self.PAD,
                 pady=self.PAD)
463         self.entry_new_quantity = ttk.Entry(self.pop_up_win, justify="left",
                textvariable=self.new_quantity)
464         self.entry_new_quantity.grid(row=3, column=1, columnspan=3, sticky="EW
                ")
465 
466         self.btn_insert = tk.Button(self.pop_up_win, text="Einbuchen", command
                =self.add_new_quantity)
467         self.btn_insert.grid(row=4, column=0, sticky="EW", padx=self.PAD, pady
                =self.PAD)
468 
469         self.button_change = tk.Button(self.pop_up_win, text="Abbrechen",
                command=self.pop_up_win.destroy)
470         self.button_change.grid(row=4, column=1, sticky="EW", padx=self.PAD,
                pady=self.PAD)
```

```
471
472     def add_new_quantity(self):
473         self.new_quantity = int(self.entry_new_quantity.get().strip())
474         try:
475             if int(self.new_quantity):
476                 self.controller.add_quantity(self.partnumber[0], self.
                        new_quantity)
477                 self.pop_up_win.destroy()
478                 tk.messagebox.showinfo("Erfolg", f"{self.new_quantity} Stk.
                        zugebucht!")
479         except ValueError:
480             tk.messagebox.showwarning("Fehler", "Anzahl muss eine Ganzzahl
                    sein!")
481
482     # Popup um Auftrag anzulegen
483     def add_order_popup(self):
484         quantity = tk.IntVar
485         self.customers = self.controller.select_customer_for_orders()
486         customers_list = [customer for _, customer in self.customers]
487
488         self.partnumbers = self.controller.select_partnumber_for_orders()
489         partnumbers_list = [(partnumber, description) for _, partnumber,
                description in self.partnumbers]
490
491         self.pop_up_win = tk.Toplevel()
492         self.pop_up_win.title("Auftrag anlegen")
493         self.pop_up_win.geometry("330x200")
494         self.pop_up_win.resizable(0,0)
495         self.label = tk.Label(self.pop_up_win, text="Neuen Kundenauftrag
                anlegen")
496         self.label.grid(row=0, column=0, columnspan=2, sticky="W", padx=self.
                PAD, pady=self.PAD)
497
498         # Auswahl Kunde
499         self.lbl_customer = tk.Label(self.pop_up_win, text="Kunde: ")
500         self.lbl_customer.grid(row=1, column=0, sticky="W", padx=self.PAD,
                pady=self.PAD)
501         self.entry_menu_customer = ttk.Combobox(self.pop_up_win, values=
                customers_list)
502         self.entry_menu_customer.set("Bitte Auswahl treffen")
503         self.entry_menu_customer.grid(row=1, column=1, columnspan=2, sticky="
                EW")
504
505         # Auswahl Sachnummer
```

```
506         self.lbl_partnumber_orders = tk.Label(self.pop_up_win, text="
                Sachnummer: ")
507         self.lbl_partnumber_orders.grid(row=2, column=0, sticky="W", padx=self
                .PAD, pady=self.PAD)
508         self.entry_menu_partnumber_orders = ttk.Combobox(self.pop_up_win,
                values=partnumbers_list)
509         self.entry_menu_partnumber_orders.set("Bitte Auswahl treffen")
510         self.entry_menu_partnumber_orders.grid(row=2, column=1, columnspan=2,
                sticky="EW")
511
512         # Eingabe Anzahl der Position
513         self.lbl_order_pos_quantity = tk.Label(self.pop_up_win,text="Anzahl [
                Stk.]: ")
514         self.lbl_order_pos_quantity.grid(row=3, column=0, sticky="W", padx=
                self.PAD, pady=self.PAD)
515         self.entry_order_pos_quantity = ttk.Entry(self.pop_up_win, justify="
                left", textvariable=quantity)
516         self.entry_order_pos_quantity.grid(row=3, column=1, columnspan=2,
                sticky="EW")
517
518         self.btn_insert = tk.Button(self.pop_up_win, text="Eintragen", command
                =self.add_order)
519         self.btn_insert.grid(row=4, column=0, sticky="EW", padx=self.PAD, pady
                =self.PAD)
520
521         self.button_abort = tk.Button(self.pop_up_win, text="Abbrechen",
                command=self.pop_up_win.destroy)
522         self.button_abort.grid(row=4, column=1, sticky="EW", padx=self.PAD,
                pady=self.PAD)
523
524     # Auftrag in Datenbank eintragen
525     def add_order(self):
526         index_customer = self.entry_menu_customer.current()
527         kundID = self.customers[index_customer][0]
528
529         index_partnumber = self.entry_menu_partnumber_orders.current()
530         sNrID = self.partnumbers[index_partnumber][0]
531
532         quantity = self.entry_order_pos_quantity.get()
533
534         try:
535             if int(quantity):
536                 order_number = self.controller.add_order(kundID, sNrID,
                        quantity)
```

```
537                 self.pop_up_win.destroy()
538                 tk.messagebox.showinfo("Erfolg", f"Auftrag Nr. {order_number}
                        erfolgreich angelegt!")
539             except ValueError:
540                 tk.messagebox.showwarning("Fehler", "Anzahl muss eine Ganzzahl
                    sein!")
541
542     def clear_right_frame_for_refresh(self):
543         for widget in self.right_frame.winfo_children():
544             widget.destroy()
545
546     def _create_entry(self):
547         entry = ttk.Entry(self.left_frame, justify="left", textvariable=self.
                value_var)
548         entry.grid(row=0, column=0, sticky="nsew")
549
550     # Methode um Message-Box aufzurufen
551     def show_message_box(self, title, message):
552         tk.messagebox.showinfo(title=title, message=message)
553
554     # Inhalt Fußzeile
555     def _create_copyright(self):
556         label = ttk.Label(self, text="© Philip Kottmann (2025)")
557         label.grid(row=1, column=0, sticky="w", padx=self.PAD, pady=self.PAD)
558
559     # Beenden-Button
560     def _end_application(self):
561         button = ttk.Button(self, text="Beenden", command=self.destroy)
562         button.grid(row=1, column=2, sticky="e", padx=self.PAD, pady=self.PAD)
```

Listing 10.2: view.py

10.2.3 controller.py

```python
# Autor: Philip Kottmann
# Datum: 7.7.2025
# Beschreibung: Controller-Klasse

from model import Model # Import Model-Klasse
from view import View # Import View-Klasse

class Controller:
    def __init__(self):
        self.model = Model()
        self.view = View(self)
        self.next_partnumber = 0
        self.current_partnumber = 0
        self.current_quantity = 0

    def main(self):
        self.view.main()

    def add_customer(self, customer_data):
        self.model.add_customer(customer_data)
        return True

    def confirm_delivery(self):
        self.view.clear_right_frame_for_refresh()
        self.view.show_message_box("Lieferbestätigung", "Dummy: Lieferung bestätigt!")

    def sales_yearly_customer(self):
        self.view.clear_right_frame_for_refresh()
        self.view.show_message_box("Kundenumsatz", "Dummy: Jahresumsatz nach Kunde")

    def sales_yearly_partnumber(self):
        self.view.clear_right_frame_for_refresh()
        self.view.show_message_box("Sachnummernumsatz", "Dummy: Jahresumsatz nach Sachnummer")

    def select_customer_for_orders(self):
        return self.model.select_customers_for_order()

    def select_partnumber_for_orders(self):
        return self.model.select_partnumbers_for_order()
```

```
40
41      def show_shortage(self):
42          self.view.clear_right_frame_for_refresh()
43          #self.view.create_listview(0, "Fehlbestände/Mindermengen", "Best.")
44          self.view.show_message_box("Fehlbestände", "Dummy: Fehlbestände")
45
46      def show_quantities_needed(self):
47          self.view.show_message_box("Mengenplanung", "Dummy: Mengenplanung")
48
49      def show_stock(self):
50          self.result = self.model.read_stock()
51          self.view.create_listview(self.result, "Lagerbestände", "Best.")
52
53      def add_partnumber(self, partnumber_data):
54          self.model.add_partnumber(partnumber_data)
55          return True
56
57      def modify_bom(self):
58          self.view.clear_right_frame_for_refresh()
59          self.view.show_message_box("Sachnmummer löschen", "Dummy: Sachnummer
                gelöscht")
60
61      def modify_core_data(self):
62          self.view.clear_right_frame_for_refresh()
63          self.view.show_message_box("Stammdaten pflegen", "Dummy: Stammdaten
                pflegen")
64
65      def read_content_listview(self, table):
66          return self.model.read_all_from_database(table)
67
68      def read_context_partnumber(self, partnumber):
69          return self.model.read_context_partnumber(partnumber)
70
71      def add_quantity(self, partnumber, quantity):
72          self.model.add_quantity(partnumber, quantity)
73
74      def add_order(self, kundID, sNrID, quantity):
75          return self.model.add_order(kundID, sNrID, quantity)
76
77      # SETTER:
78      def set_next_partnumber(self):
79          self.next_partnumber = self.model.calculate_partnumber()
80
81      # GETTER:
```

```
82      def get_next_partnumber(self):
83          self.set_next_partnumber()
84          return self.next_partnumber
85
86      def get_quantity(self, partnumber):
87          self.partnumber = partnumber
88          return self.model.get_current_quantity(self.partnumber)
89
90      def division_method(zaehler, nenner):
91          return zaehler / nenner
92
93
94  # Aufruf der main()-Methode, sobald das Programm aus controller.py aufgerufen
        wird
95  if __name__ == "__main__":
96      erp_system = Controller()
97      erp_system.main()
```

Listing 10.3: controller.py

10.2.4 t_controller.py - Testdatei

```python
import pytest
from controller import Controller

def test_division_method_standard():
    assert Controller.division_method(10, 2) == 5
    assert Controller.division_method(-12, 4) == -3
    assert Controller.division_method(100, 100) == 1

def test_division_method_float():
    assert Controller.division_method(1, 4) == 0.25
    assert Controller.division_method(10, 2.0) == 5.0
    assert pytest.approx(Controller.division_method(10, 3), 0.01) == 3.33

def test_division_method_division_by_zero():
    with pytest.raises(ZeroDivisionError):
        Controller.division_method(5, 0)
```

Listing 10.4: t_controller.py

Literaturverzeichnis

[1] Krypczyk, Veikko ; Bochkor, Elena: *Handbuch für Softwareentwickler*. 2., überarbeitete und aktualisierte Auflage. Rheinwerk Verlag, 2022. – ISBN 978–3–8362–7977–2

[2] LaTeX-Tutorial: *LaTeX-Vorlage Abschlussarbeit*. Youtube https://www.dropbox.com/scl/fi/6rgt2bvw5ca1zziw4j65f/Vorlage-TeX.zip, Dezember 2013

[3] Kofler, Michael: *Python Der Grundkurs*. 3., aktualisierte Auflage. Rheinwerk Verlag, 2024. – ISBN 978–3–367–10118–4

[4] Dalwigk, Florian A.: *python <FÜR EINSTEIGER>*. Eulogia Verlag, 2022. – ISBN 978–3–96967–224–2

[5] Ernesti, Johannes ; Kaiser, Peter: *Python 3*. 7., aktualisierte Auflage 2023, 2., korrigierter Nachdruck 2024. Rheinwerk Verlag, 2024. – ISBN 978–3–8362–9129–3

[6] Elter, Stephan: *Schrödinger programmiert Python*. 2., aktualisierte und erweiterte Auflage. Rheinwerk Verlag, 2025. – ISBN 978–3–367–10519–9

[7] *www.pythontutorial.net*. `https://www.pythontutorial.net`. Version: Juli 2025

[8] *www.python-kurs.eu*. `https://www.python-kurs.eu`. Version: Juli 2025

[9] (Youtube) @lifeincode94: *Python MVC-Tutorial*. `https://www.youtube.com/@lifeincode94`. Version: Juli 2025

[10] https://docs.python.org/3/: *Offizielle Python 3 Dokumentation*. `https://docs.python.org/3/`. Version: Juli 2025

[11] *https://www.geeksforgeeks.org/category/python/*. `https://www.geeksforgeeks.org/category/python/`. Version: Juli 2025

[12] (Youtube) @anthonywritescode: *pytest Tutorial*. `https://www.youtube.com/@anthonywritescode`. Version: Januar 2023

[13] *www.sqlitetutorial.net*. `https://www.sqlitetutorial.net`. Version: Juli 2025

[14] `https://hellocoding.de/blog/coding-language/python/pyinstaller-app-exe-erstellen`