



Abschlussprüfung Sommer 2025 Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit  
**Automatisierung der Menükennzeichnung  
und Menüverwaltung**

**Prüfungsbewerber:**

Sergey Grinko  
Lerchenweg 37  
57250 Netphen

**Ausbbildungsbetrieb:**

CONZE Informatik GmbH  
Friedrichstraße 18  
57072 Siegen

## Inhaltsverzeichnis

Abbildungsverzeichnis .....	III
Tabellenverzeichnis .....	III
Listings .....	III
Glossar .....	IV
Abkürzungsverzeichnis .....	VI
1 Einleitung .....	1
1.1 Projektumfeld .....	1
1.2 Projektbeschreibung .....	1
1.3 Projektziel .....	1
1.4 Projektbegründung .....	1
1.5 Projektschnittstellen .....	1
2 Projektplanung .....	2
2.1 Ressourcenplanung .....	2
2.2 Entwicklungsprozess .....	2
3 Analysephase .....	2
3.1 Ist-Analyse .....	2
3.2 Wirtschaftlichkeitsanalyse .....	2
3.2.1 „Make or Buy“-Entscheidung .....	2
3.2.2 Projektkosten .....	2
3.2.3 Amortisationsdauer .....	3
3.2.4 Nicht-monetäre Aspekte .....	3
3.3 Nutzwertanalyse .....	4
3.4 Anwendungsfälle .....	4
3.5 Lastenheft .....	4
4 Entwurfsphase .....	5
4.1 Zielplattform .....	5
4.2 Architekturdesign .....	5
4.3 Lokale Anwendung .....	6
4.4 Datenmodell .....	7
4.5 Web-Anwendung .....	7
4.6 Maßnahmen zur Qualitätssicherung .....	8
4.7 Pflichtenheft .....	8
5 Implementierungsphase .....	9
5.1 Prototyping mit Node-Red .....	9
5.2 Implementierung der lokalen Anwendung .....	9
5.3 Implementierung des Datenmodells .....	10
5.4 Implementierung der Web-Anwendung .....	10
6 Qualitätssicherung und Abnahme .....	11
6.1 Einführung und Schulung .....	11

7 Dokumentation.....	11
A Anhang .....	i
A.1 Verpackungsbeschriftung - handschriftlich vs. automatisiert.....	i
A.2 Verwendete Ressourcen.....	i
A.3 Aktivitätsdiagramm des Essensempfangs mit der manuellen Menükennzeichnung .....	ii
A.4 Amortisation – Break-Even-Point.....	ii
A.5 Use-Case-Diagramm .....	iii
A.6 Lastenheft (Auszug) .....	iii
A.7 Deployment-Diagramm .....	iv
A.8 Paket-Diagramm .....	v
A.9 Klassendiagramm – Klassen und Schnittstellen .....	vi
A.10 Aktivitätsdiagramm – Etikettendruck .....	vii
A.11 Sequenzdiagramm – Barcode-Scanning und Benachrichtigung .....	vii
A.12 Oberflächenentwürfe .....	viii
A.13 Pflichtenheft (Auszug) .....	ix
A.14 Node-RED-Flow zur Barcode-Verarbeitung mit Debug-Ausgabe.....	x
A.15 Klassendiagramm-Drucker.....	xi
A.16 Codebeispiel 1: Drucksteuerung .....	xii
A.17 Codebeispiel 2: Testfall zur Befehlserzeugung mit Strategie-Muster und Konsolenausgabe ..	xiii
A.18 Codebeispiel 3: Singleton-Logger .....	xiv
A.19 Codebeispiel 4: Löschfunktion .....	xv
A.20 Codebeispiel 5: Gericht hinzufügen mit Go und Gin.....	xv
A.21 Codebeispiel 6: HTML-Template für Fehler- und Erfolgsmeldungen .....	xvi
A.22 Ablauf der Menülöschung in der Webanwendung .....	xvii
A.23 Prozessablauf des Akzeptanztests (Bildcollage) .....	xviii
A.24 Qualitätssicherungskonzept – Übersicht der Testmethoden .....	xix

## Abbildungsverzeichnis

Abbildung 1: Entity-Relationship-Modell	7
Abbildung 2: Verpackungsbeschriftung - handschriftlich vs. automatisiert	i
Abbildung 3: Aktivitätsdiagramm des Essensempfangs mit der manuellen Menükennzeichnung	ii
Abbildung 4: Amortisation – Break-Even-Point	ii
Abbildung 5: Use-Case-Diagramm	iii
Abbildung 6: Deployment-Diagramm	iv
Abbildung 7: Paket-Diagramm	v
Abbildung 8: Klassendiagramm – Klassen und Schnittstellen	vi
Abbildung 9: Aktivitätsdiagramm – Etikettendruck	vii
Abbildung 10: Sequenzdiagramm – Barcode-Scanning und Benachrichtigung	vii
Abbildung 11: Mock-Ups Collage	viii
Abbildung 12: Node-RED-Flow zur Barcode-Verarbeitung mit Debug-Ausgabe	x
Abbildung 13: Klassendiagramm. Drucker	xi
Abbildung 14: Testergebnis der Befehlserzeugung mit Strategie-Muster	xiv
Abbildung 15: Ablauf der Menülöschung in der Webanwendung	xvii
Abbildung 16: Prozessablauf des Akzeptanztests	xviii

## Tabellenverzeichnis

Tabelle 1: Kostenaufstellung	3
Tabelle 3: Nutzwertanalyse	4
Tabelle 4: Entscheidungsmatrix	6
Tabelle 5: Testkonzept	xix

## Listings

Codebeispiel 1: Drucksteuerung	xii
Codebeispiel 2: Testfall zur Befehlserzeugung mit Strategie-Muster und Konsolenausgabe	xiii
Codebeispiel 3: Singleton-Logger	xiv
Codebeispiel 4: Löscherfunktion	xv
Codebeispiel 5: Gericht hinzufügen mit Go und Gin	xv
Codebeispiel 6: HTML-Template für Fehler- und Erfolgsmeldungen	xvi

## Glossar

**AAA-Regel.** AAA steht für: Arrange (Im Arrange wird die Startsituation des Tests definiert: es werden z.B. die Variablen erzeugt), Act (Im Act wird die eigentliche Aktion durchgeführt), Assert (im Assert wird überprüft, ob das erwartete Ergebnis durch das Act erzielt wurde) und beschreibt den Aufbau eines Unit-Tests. (<https://binaris-informatik.de/>)

**Akzeptanztest** oder Abnahmetest, engl. User Acceptance Tests (UAT) ist in der Softwaretechnik die Überprüfung, ob eine Software aus Sicht des Benutzers wie beabsichtigt funktioniert und dieser die Software akzeptiert. (<https://de.wikipedia.org/>)

**All-in-One-Lösung** in IT ist eine integrierte Systemlösung, die alle erforderlichen Funktionen in einer einzigen Plattform vereint. (<https://chatgpt.com/>)

**Black-Box-Test.** Beim Black-Box-Test wird ein System ausschließlich von außen beurteilt, ohne dass der Nutzer oder Tester weiß, was innerhalb des Systems geschieht. (<https://www.computer-weekly.com/>)

**Code-Review** (manchmal auch als Peer Review bezeichnet) ist eine Aktivität zur Qualitätssicherung von Software, bei der eine oder mehrere Personen den Quellcode eines Computerprogramms prüfen, entweder nach der Implementierung oder während des Entwicklungsprozesses. ([https://en.wikipedia.org/wiki/Code\\_review](https://en.wikipedia.org/wiki/Code_review))

**Code-Smells** bezeichnen funktionierenden, aber schlecht strukturierten Quellcode. Sie deklarieren Konstrukte in der Programmierung, die ein Refactoring nahelegen. Im übertragenen Sinne handelt es sich um übelriechenden Code. (<https://t2informatik.de/>)

**Common Unix Printing System** ist das am weitesten verbreitete Drucksystem auf Linux-Systemen. Über CUPS lassen sich zahlreiche Drucker nutzen, verwalten und im Netzwerk freigeben. (<https://wiki.ubuntuusers.de/CUPS/>)

**Corporate Design** bezeichnet die visuelle Gestaltung eines Unternehmens. Als visueller Teil der Corporate Identity bezieht es sich auf alle Elemente, die in ihrer Gesamtheit eine Marke sichtbar machen: von Logo Design über Farbwelt, Bildsprache, Typografie und viele weitere Details. (<https://www.benchmark-design.de/>)

**Datenbank**, auch Datenbanksystem genannt, ist ein System zur elektronischen Datenverwaltung. Die wesentliche Aufgabe einer Datenbank ist es, große Datenmengen effizient, widerspruchsfrei und dauerhaft zu speichern und benötigte Teilmengen in unterschiedlichen, bedarfsgerechten Darstellungsformen für Benutzer und Anwendungsprogramme bereitzustellen. (<https://de.wikipedia.org/>)

**Don't repeat yourself** (DRY, englisch für „wiederhole dich nicht“; auch bekannt als once and only once „einmal und nur einmal“) ist ein Prinzip, das besagt, Redundanz zu vermeiden oder zumindest zu reduzieren. (<https://de.wikipedia.org/>)

**Fassade** (englisch facade) ist ein Entwurfsmuster aus dem Bereich der Softwareentwicklung, das zur Kategorie der Strukturmuster (engl. structural design patterns) gehört. Es bietet eine einheitliche und meist vereinfachte Schnittstelle zu einer Menge von Schnittstellen eines Subsystems. (<https://de.wikipedia.org/>)

**First-Class Citizen** ist eine Entität, die alle Operationen unterstützt, die anderen Entitäten allgemein zur Verfügung stehen. Zu diesen Operationen gehören typischerweise die Übergabe als Argument, die Rückgabe von einer Funktion und die Zuweisung an eine Variable . (<https://de.wikipedia.org/>)

**Funktion höherer Ordnung** (englisch higher-order function) ist in der Informatik eine Funktion, die Funktionen als Argumente erhält und/oder Funktionen als Ergebnis liefert. (<https://de.wikipedia.org/>)

**Goroutine** ist eine leichte, unabhängig ausgeführte Arbeitseinheit, die von der Go-Laufzeit verwaltet wird. Im Gegensatz zu herkömmlichen Threads sind Go-Routinen keine OS-Threads; sie werden vom Go-Scheduler verwaltet, was eine effiziente Parallelität mit minimalem Overhead ermöglicht. (<https://durgeshatal1995.medium.com/>)

**Keep it short and simple** manchmal auch KISS-Formel, KISS-Gesetz oder in der Schreibweise K.I.S.S.-Prinzip, fordert, zu einem Problem eine möglichst einfache Lösung anzustreben. (<https://de.wikipedia.org/>)

**Lasttest** ist ein Softwaretest, der eine in der Regel sehr hohe Last auf dem zu testenden System erzeugt und dessen Verhalten untersucht. Dazu kann eine Simulation eingesetzt werden. (<https://de.wikipedia.org/>)

**Lokale Anwendung** ist Software, die direkt auf einem bestimmten Gerät installiert und ausgeführt wird, ohne eine ständige Internetverbindung zu benötigen. (<https://www.studysmarter.de/>)

**Monolithische Softwarearchitektur** ist ein traditionelles Modell für den Entwurf von Softwareanwendungen, bei dem alle Komponenten und Funktionen einer Anwendung in einer einzigen, einheitlichen Codebasis eng miteinander verbunden sind. Das bedeutet, dass die gesamte Anwendung als eine einzige, in sich geschlossene Einheit entwickelt, bereitgestellt und verwaltet wird. (<https://www.computer-weekly.com/>)

**Mutationstests** (oder Mutationsanalysen oder Programmutationen) werden verwendet, um neue Softwaretests zu entwickeln und die Qualität bestehender Softwaretests zu bewerten. Bei Mutationstests wird ein Programm geringfügig geändert. (<https://en.wikipedia.org/>)

**Mutex** oder MUTual EXclusion ist in Go im Grunde eine Möglichkeit, sicherzustellen, dass immer nur eine Goroutine mit einer gemeinsam genutzten Ressource herumspielt. (<https://victoriometrics.com/>)

**Node-Red** ist ein von IBM entwickeltes grafisches Entwicklungswerkzeug. Die Software ermöglicht es, Anwendungsfälle im Bereich des Internets der Dinge mit einem einfachen Baukastenprinzip umzusetzen. Die einzelnen Funktionsbausteine werden durch Ziehen von Verbindungen verbunden. (<https://de.wikipedia.org/>)

**Proof of Concept**, also die Prüfung eines Konzepts, dient als Nachweis für die grundsätzliche Machbarkeit eines Vorhabens. Insbesondere in IT-Projekten ist ein PoC ein klassischer Meilenstein: Er bestätigt, dass Projekte nach der Idee machbar sind und schafft so die Grundlage für die weitere Arbeit des Projektteams. (<https://www.modernizing-applications.de/>)

**Race Condition** (deutsch Wettschlag-Bedingung) oder Race Hazard (deutsch Wettschlag-Risiko), mitunter auch „kritische Wettschlagsituation“, bezeichnet in der Programmierung eine Konstellation, in der das Ergebnis einer Operation vom zeitlichen Verhalten bestimmter Einzeloperationen oder der Umgebung abhängt. Der Begriff stammt von der Vorstellung, dass zwei Signale wettschlagen, um die Ausgabe als erstes zu beeinflussen. (<https://de.wikipedia.org/>)

**REST-API** (REST steht für Representational State Transfer; API steht für Application Programming Interface) ermöglicht die Interaktion zwischen einem Client, wie zum Beispiel einer Web-Anwendung oder einem mobilen Gerät, und einem Server. (<https://www.impulsphase.de/>)

**Single-Board Computer** - Ein Einplatinencomputer, ist ein Computersystem, bei dem sämtliche zum Betrieb nötigen elektronischen Komponenten auf einer einzigen Leiterplatte zusammengefasst sind. (<https://de.wikipedia.org/>)

**Singleton** (selten auch Einzelstück genannt) ist ein in der Softwareentwicklung eingesetztes Entwurfsmuster und gehört zur Kategorie der Erzeugungsmuster (engl. creational patterns). Es stellt sicher, dass von einer Klasse genau ein Objekt existiert. Dieses Singleton ist darüber hinaus üblicherweise global verfügbar. (<https://de.wikipedia.org/>)

**Strategie** (englisch strategy) ist im Bereich der Softwareentwicklung ein Entwurfsmuster und gehört zur Kategorie der Verhaltensmuster (englisch behavioral design patterns). Die Strategie definiert eine Familie austauschbarer Algorithmen. (<https://de.wikipedia.org/>)

**Unit-Tests** (=Komponententests) überprüfen, ob die von den Entwicklern geschriebenen Komponenten so arbeiten, wie diese es beabsichtigen. (<https://www.it-agile.de/>)

**Web-Anwendung**. Mit dem Begriff Web-Anwendung werden Anwendungsprogramme bzw. Software bezeichnet, die von verschiedenen Webbrowsern (Google Chrome, Mozilla Firefox, Apple Safari) und Betriebssystemen (Windows, macOS, Chrome OS, Linux) aufgerufen und ausgeführt werden können. (<https://www.modernizing-applications.de/>)

**You Aren't Gonna Need It** zu Deutsch: „Du wirst es nicht brauchen“. Es bezeichnet ein Prinzip des Extreme Programming (XP), das besagt, dass in einem Programm erst dann Funktionalität implementiert werden sollte, wenn klar ist, dass diese Funktionalität tatsächlich gebraucht wird. (<https://de.wikipedia.org/>)

## Abkürzungsverzeichnis

**CSS** - Cascading Style Sheets

**DRY** - [don't repeat yourself](#)

**GUI** - Graphical User Interface

**HMI** - Human Machine Interface

**HTML** - Hypertext Markup Language

**HTTP** - Hypertext Transfer Protocol

**JSON** - JavaScript Object Notation

**KISS** - [Keep it short and simple](#)

**SBC** - [Single-Board Computer](#)

**SMTP** - Simple Mail Transfer Protocol

**USB** - Universal Serial Bus

**UI-/UX-Design** - User Interface/ User Experience Design

**YAGNI** - [You Aren't Gonna Need It](#)

## 1 Einleitung

### 1.1 Projektumfeld

Das Projekt wird bei der **CONZE Informatik GmbH** realisiert, einem Unternehmen für User Interface Engineering. Seit 2009 entwickelt die Firma grafische Benutzeroberflächen ([GUIs](#), [HMIs](#)) für Medizintechnik, Healthcare sowie Industrie- und Automotive-Anwendungen. Neben [UI-UX-Design](#) überträgt das Team seine Expertise auf softwaregesteuerte Systeme.

Es handelt sich um ein internes Projekt. Auftraggeber ist die Geschäftsführung.

### 1.2 Projektbeschreibung

Die Kantine der CONZE Informatik GmbH erhält regelmäßig tiefgekühlte Gerichte von der **Hofmann Menü-Manufaktur**. Diese werden per LKW geliefert und müssen schnell in den Tiefkühlschrank eingearbeitet werden.

Die Verpackungen werden übereinandergestapelt gelagert, wobei der Barcode mit Menünummer und Gerichtsnamen nur auf der Oberseite sichtbar ist. Mitarbeitende beschriften jede Schachtel seitlich **handschriftlich** mit der Menünummer (siehe [Anhang A.1 Verpackungsbeschriftung - handschriftlich vs. automatisiert, Seite i](#), Foto [links](#)), um eine Identifikation ohne Herausnehmen zu ermöglichen. Diese Methode ist zeitaufwändig, fehleranfällig und unangenehm, da die kalten Verpackungen währenddessen in der Hand gehalten werden müssen.

Da Bestellungen **monatlich** erfolgen, müssen regelmäßig größere Mengen etikettiert werden. Verschiedene Mitarbeitende aus den Unternehmensabteilungen übernehmen den Essensempfang. Die Kantine wird täglich von 2 bis 15 Personen genutzt.

### 1.3 Projektziel

Der manuelle Prozess soll automatisiert, vereinfacht und beschleunigt werden. Ein Barcode-Scanner (Honeywell Eclipse MS5145 LS USB) liest den Barcode auf der Oberseite der Menüschahteln aus und formatiert ihn. Ein Etikettendrucker (ZEBRA ZD 410) erstellt daraufhin ein Etikett mit der Menünummer. Ist die Nummer in der [Datenbank](#), wird auch der **Gerichtsname** gedruckt. Fehlt der Eintrag, erhält der Administrator eine automatische E-Mail-Benachrichtigung, um den Namen nachzutragen. Das Etikett wird anschließend seitlich aufgeklebt.

Das System soll als [lokale Anwendung](#) mit direkter Hardware-Anbindung entwickelt werden. Die Datenbankverwaltung soll über eine separate [Web-Anwendung](#) erfolgen. Vor der Umsetzung soll ein **Prototyp** in [Node-Red](#) erstellt werden, um die Interaktion von Scanner, Drucker und Datenverarbeitung zu testen.

Ziel ist eine „[All-in-One](#)“-Lösung, die alle erforderlichen Funktionen integriert. Das System soll auf einem Single-Board-Computer ([SBC](#)) umgesetzt werden, der Prozessor, Speicher und Anschlüsse vereint. Zudem soll ein Webserver auf dem SBC die Benutzeroberfläche bereitstellen.

### 1.4 Projektbegründung

Die Automatisierung der Menükennzeichnung soll **Zeit** sparen, den **Personalaufwand** reduzieren und **Fehler** minimieren. Die **Lesbarkeit** der Menünummern soll verbessert, die **Nachverfolgbarkeit** durch ein digitales Protokoll gewährleistet und die **Verwaltung** über die Benutzeroberfläche erleichtert werden. Dadurch soll der Arbeitsaufwand in der Kantine sinken, der Ablauf effizienter und fehlerfreier werden und die **Arbeitsbedingungen** verbessert werden.

### 1.5 Projektschnittstellen

Die Anwendung soll Barcodes mit dem Barcode-Scanner Honeywell Eclipse MS5145 LS USB erfassen und Etiketten mit dem Etikettendrucker ZEBRA ZD 410 drucken. Ein Webserver auf einem [SBC](#) soll die Verwaltungsoberfläche bereitstellen, die gescannten Daten sollen in einer **lokalen JSON-Datenbank** gespeichert und der Administrator über ein **E-Mail-System** informiert werden.

Die CONZE Informatik GmbH genehmigt und finanziert das Projekt. Die **Geschäftsführung** bewertet den Projekterfolg.

**Hauptnutzer** sind die Mitarbeitenden beim Essensempfang sowie eine HR-Mitarbeiterin zur Verwaltung der Daten.

In der Testphase erfolgen ein [Black-Box-Test](#) durch eine **HR-Mitarbeiterin**, ein [Code-Review](#) durch die

**IT-Abteilung** und ein [Akzeptanztest](#) durch **Endnutzer**. Ein **Support-Mitarbeiter** steht für Hardware-Fragen bereit.

## 2 Projektplanung

### 2.1 Ressourcenplanung

Eine Übersicht der verwendeten Ressourcen befindet sich in [Anhang A.2: Verwendete Ressourcen \(Seite i\)](#), einschließlich **Hardware**, **Software** und **personelle** Ressourcen. Bei der Softwareauswahl wurde auf lizenzzfreie Lösungen geachtet, um die Projektkosten niedrig zu halten.

Die Auswahl des [Single-Board-Computers](#) basiert auf einer Nutzwertanalyse (siehe [Kapitel 3.3](#)).

### 2.2 Entwicklungsprozess

Für das Projekt wird das **Wasserfallmodell** gewählt, da die Anforderungen definiert und die Projektlaufzeit festgelegt sind. Das Modell folgt einer sequenziellen Vorgehensweise, bei der jede Phase auf der vorherigen aufbaut. Es wird jedoch die Möglichkeit vorgesehen, bei neuen Erkenntnissen oder Problemen in eine frühere Phase zurückzukehren, um Anpassungen vorzunehmen und Risiken frühzeitig zu identifizieren.

## 3 Analysephase

### 3.1 Ist-Analyse

Wie bereits im [Abschnitt 1.2 Projektbeschreibung](#) erwähnt, erfolgt die **Menükennzeichnung** in der Kantine der CONZE Informatik GmbH **manuell**. Nach der Lieferung der tiefgekühlten Gerichte werden diese aus den Kisten entnommen und mit einer fünfstelligen Menünummer per Hand mit einem Marker beschriftet. Handschriftliche Nummern können durch Kondenswasser oder Reibung oft **unleserlich** werden (siehe [Anhang A.1 Verpackungsbeschriftung - handschriftlich vs. automatisiert](#) (links), [Seite i](#)). Für den **Essensempfang** sind sechs Personen erforderlich, was etwa 45 Minuten dauert. Das Aktivitätsdiagramm ([Anhang A.3, Seite ii](#)) veranschaulicht diesen Ablauf.

Zur Beseitigung dieser Probleme ist eine Automatisierung der Menükennzeichnung notwendig. Der Einsatz eines Barcode-Scanners und eines Etikettendruckers soll die Menünummer direkt aus einem Barcode auslesen und anschließend als Etikett ausdrucken, wodurch handschriftliche Fehler eliminiert und der Prozess beschleunigt werden. Zusätzlich wurde in einem Gespräch mit dem **Auftraggeber** festgelegt, dass der gesamte Arbeitsprozess digital **protokolliert** werden muss, um eine lückenlose **Nachverfolgbarkeit** zu gewährleisten. Der **Code** muss verständlich und ausführlich **kommentiert** sein, und die Menüliste muss einfach **erweiterbar** bleiben, da regelmäßig neue Gerichte hinzukommen. Es soll **keine zusätzliche** Software für die Menüverwaltung auf den Arbeitsgeräten installiert werden, und die **GUI** zur Verwaltung der Menünummern muss direkt auf dem [SBC](#) laufen. Außerdem muss die Benutzeroberfläche das [Corporate Design](#) integrieren, einschließlich **Firmenlogo** und **-symbole**, und die Software muss in einer im Unternehmen **etablierten Programmiersprache** entwickelt werden.

Die Anforderungen wurden während der Gespräche als Notizen in **User-Story-Form** festgehalten.

### 3.2 Wirtschaftlichkeitsanalyse

Die Einführung des automatisierten Systems optimiert den Prozess und senkt langfristig die Kosten. Die Investitionskosten führen zu **direkten Einsparungen** im laufenden Betrieb.

#### 3.2.1 „Make or Buy“-Entscheidung

Die Marktrecherche ergab, dass es **keine passende Lösung** gibt. Auch der aktuelle Lieferant, die Hofmann Menü-Manufaktur, bietet keine automatische Menükennzeichnung an. Daher ist eine Eigenentwicklung erforderlich.

#### 3.2.2 Projektkosten

Die Kosten setzen sich aus Personal- und Ressourcenkosten zusammen. Da interne Stundensätze nicht offengelegt werden dürfen, basieren die Berechnungen auf Werten der Personalabteilung:

- Stundensatz der Geschäftsführung – **50€/h**;
- Entwickler/in (inkl. technischer Support) – **25€/h**;

- HR-Mitarbeiter/in – **24€/h**;
- Umschüler/in – **4€/h**.

Für Büroarbeitsplätze, Hardware, Software und Gemeinkosten (z. B. Strom) wird pauschal **18€/h** angesetzt. Die jährlichen laufenden Kosten für das System nach der Umsetzung betragen **67€**.

**Hardwarekosten:** Die Anschaffungskosten fließen vollständig in die Projektkosten ein. Der Raspberry Pi wurde für **142,90€** gekauft. Für bereits vorhandene Geräte, ursprünglich für ein anderes Projekt beschafft, wurden die aktuellen Preise angesetzt:

- Barcode-Scanner (Honeywell Eclipse MS5145, USB) **63,79€**,
- Etikettendrucker (Zebra ZD410) **149,90€**.

Der [Akzeptanztest](#) (Mitarbeitererschulung, Durchführung, Nachbesprechung) wurde mit 3 Entwicklern/innen ( $25\text{€}/\text{h} + 18\text{€}/\text{h} = 43\text{€}/\text{h}$  pro Person) und 1 HR-Mitarbeiterin ( $24\text{€}/\text{h} + 18\text{€}/\text{h} = 42\text{€}/\text{h}$ ) durchgeführt.

Berechnung für den Akzeptanztest:

$(3 \times 43) + (1 \times 42) = 171\text{€}$ . Die Kosten dafür sind in der Tabelle als ‚oben berechnet‘ eingetragen:

Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklungskosten	80h	$4,00\text{€} + 18,00\text{€} = 22,00\text{€}$	1.760,00€
Technischer Support	0,5h	$25,00\text{€} + 18,00\text{€} = 43\text{€}$	21,5€
Code-Review	2h	$25,00\text{€} + 18,00\text{€} = 43\text{€}$	86€
Black-Box Test	0,5h	$24,00\text{€} + 18,00\text{€} = 42\text{€}$	21€
Akzeptanztest	1h	oben berechnet	171€
Abnahme	1h	$50,00\text{€} + 18,00\text{€} = 68\text{€}$	68€
SBC (Raspberry Pi)	-	Einmalig	142,90€
Scanner	-	Einmalig	63,79€
Etikettendrucker	-	Einmalig	149,90€
<b>Projektkosten gesamt</b>			<b>2.484,09€</b>

Tabelle 1: Kostenaufstellung

### 3.2.3 Amortisationsdauer

Da die Arbeit in der Kantine von wechselnden Personen übernommen wird, basiert die Berechnung auf dem durchschnittlichen Stundensatz der Entwickler/-innen. Eine HR-Mitarbeiterin ist stets involviert. Die Lieferung erfolgt monatlich.

**Vor der Einführung** des Systems benötigten 6 Personen ca. 45 Minuten:

- 5 Entwickler/-innen ( $43\text{€}/\text{h}$  pro Person),
- 1 HR-Mitarbeiterin ( $42\text{€}/\text{h}$ ).

Monatliche Kosten vor Umsetzung:  $(5 \times 43 \times \frac{3}{4}) + (1 \times 42 \times \frac{3}{4}) = 192,75\text{ €}$

**Nach Einführung** des Systems erledigen 4 Personen die Arbeit in 20 Minuten:

- 3 Entwickler/-innen ( $43\text{ €}/\text{h}$ ),
- 1 HR-Mitarbeiterin ( $42\text{ €}/\text{h}$ ).

Monatliche Kosten nach Umsetzung:  $(3 \times 43 \times \frac{1}{3}) + (1 \times 42 \times \frac{1}{3}) = 57\text{ €}$

Laufende Kosten pro Monat:  $67 \div 12 \approx 5,59\text{ €}$

Gesamtkosten (monatliche und laufende Kosten) nach Umsetzung:  $57 + 5,59 = 62,59\text{ €}$

**Ersparnis** pro Monat:  $192,75 - 62,59 = 130,16\text{ €}$

**Amortisationsdauer:** Die Projektkosten betragen  $2.484,09\text{ €}$ . Berechnung:  $2.484,09 \div 130,16 \approx 19,08$  Monate. Das System amortisiert sich vollständig nach **1 Jahr und 8 Monaten**.

Eine grafische Darstellung des Break-even-Points ist im [Anhang A.4](#) auf [Seite ii](#) zu sehen.

### 3.2.4 Nicht-monetäre Aspekte

Das Projekt verbessert langfristig **Effizienz** und **Arbeitsbedingungen**.

Mitarbeitende müssen Tiefkühlverpackungen nicht mehr lange halten, was die Arbeit angenehmer

macht. Die automatische Etikettierung sorgt für eine einheitliche, gut lesbare Kennzeichnung, erleichtert das Auffinden der Menüs und reduziert Fehler. Zudem entlastet die Automatisierung die Entwickler, da sie sich stärker auf ihre **Hauptaufgaben** konzentrieren können.

### 3.3 Nutzwertanalyse

Nach Rücksprache mit dem technischen Support wurden folgende SBC-Modelle ausgewählt:

1. **Raspberry Pi 5 (8GB)** Starter-Kit,
2. **Raspberry Pi 4 Model B (4GB)**,
3. **Orange Pi 5 (8GB)** Starter-Kit,
4. **ASUS Tinker Board 2**.

Die Bewertung erfolgte anhand folgender **Kriterien**:

- **Leistung (30%)** – Hohe Rechenleistung für eine schnelle und zuverlässige Menükennzeichnung.
- **Preis (25%)** – Kosteneffizienz ist wichtig, aber nicht vorrangig gegenüber Stabilität und Leistung.
- **Software-Support (20%)** – Umfassende Treiber- und Community-Unterstützung.
- **Anschlüsse & Erweiterbarkeit (15%)** – Notwendig für Barcode-Scanner und Etikettendrucker.
- **Kühlung & Zubehör (10%)** – Kontinuierlicher Betrieb erfordert gute Kühlung; offizielles Zubehör erleichtert Inbetriebnahme.

Die Kriterien wurden prozentual gewichtet und auf einer Skala von 1 bis 5 bewertet (5 die höchste Punktzahl).

Kriterium	Gewichtung	Raspberry Pi 5		Raspberry Pi 4		Orange Pi 5		ASUS Tinker Board 2	
		Punkte	Wert	Punkte	Wert	Punkte	Wert	Punkte	Wert
<b>Leistung</b>	30	5	<b>150</b>	3	90	5	<b>150</b>	3	90
<b>Preis</b>	25	3	75	5	<b>125</b>	2	50	4	100
<b>Software-Support</b>	20	5	<b>100</b>	5	<b>100</b>	3	60	3	60
<b>Anschlüsse &amp; Erweiterbarkeit</b>	15	4	60	4	60	5	<b>75</b>	4	60
<b>Kühlung &amp; Zubehör</b>	10	5	<b>50</b>	3	30	4	40	2	20
<b>Gesamtwert</b>	100		<b>435</b>		405		375		330

Tabelle 2: Nutzwertanalyse

**Der Raspberry Pi 5 (8GB)** erhielt die höchste Punktzahl, da er eine sehr hohe Leistung bei gutem Preis-Leistungs-Verhältnis, exzellenten Software-Support, ausreichend Anschlüsse und eine effiziente Kühlung bietet.

Nach Abstimmung mit dem Auftraggeber wurde die Bestellung über **Amazon** durchgeführt, da das Unternehmen dort bereits gute Erfahrungen mit schneller Lieferung und zuverlässigem Kundenservice gemacht hat.

### 3.4 Anwendungsfälle

Während der Analysephase wurde ein Use-Case-Diagramm erstellt, das die beteiligten Akteure und ihre Anforderungen definiert. Es ist im Anhang A.5: Use-Case-Diagramm auf Seite iii zu finden. Die **Akteure** sind in Mitarbeiter und Administrator unterteilt. Eine **include-Beziehung** zeigt, dass der Druckvorgang stets nach einem Barcode-Scan erfolgt. Eine **extend-Beziehung** stellt dar, dass der Gerichtsname mitgedruckt wird, sofern er in der Datenbank vorhanden ist. Das Diagramm veranschaulicht die grundlegenden Systemfunktionen.

### 3.5 Lastenheft

Am Ende der Analysephase wurde gemeinsam mit dem Auftraggeber ein Lastenheft erstellt, das die Anforderungen an das System definiert. Ein Auszug ist im Anhang A.6 Lastenheft (Auszug) zu finden.

## 4 Entwurfsphase

In der Entwicklung wird besonderer Wert auf Prinzipien wie [DRY](#), [KISS](#) und [YAGNI](#) gelegt.

### 4.1 Zielplattform

Die Zielplattform für die Automatisierung der Menükennzeichnung und -verwaltung kombiniert Hardware- und Software-Komponenten, die speziell für die Anforderungen des Projekts ausgewählt wurden. Die Auswahl erfolgte nach folgenden Kriterien:

**Programmiersprache:** Go bietet hohe Performance, geringe Speicherauslastung, klare Syntax und statische Typisierung, was die Codequalität verbessert. Zudem ist Go bereits im Unternehmen etabliert (Auftraggeberanforderung), was Wartung und Weiterentwicklung erleichtert.

**Web-Anwendung:** [HTML](#) und [CSS \(Bootstrap\)](#) wurden für die Web-Gestaltung gewählt. Diese standardisierten Technologien bieten hohe Browser-Kompatibilität und vordefinierte Styles, was den Entwicklungsaufwand reduziert und die Anpassung an das [Corporate Design](#) erleichtert.

**Prototyping:** Für das Prototyping kam [Node-Red](#) zum Einsatz, das eine schnelle visuelle Entwicklung ermöglicht und einfache Hardware-Integration unterstützt.

**Programmierparadigmen:** Das Projekt basiert auf prozeduraler Programmierung mit Elementen aus objektorientierter und funktionaler Programmierung.

**Datenbank:** Die Menüdaten werden in einer JSON-Datei gespeichert, da keine komplexen relationalen Abfragen erforderlich sind und die Datenstruktur einer einfachen Key-Value-Organisation entspricht. [JSON](#) bietet eine einfache und effiziente Speicherung sowie leichte Erweiterbarkeit.

**Hardware:** Die Hardware wurde nach Leistung, Energieeffizienz und einfacher Integration ausgewählt. Bereits vorhandene Komponenten wie der Barcode-Scanner und der Etikettendrucker wurden ins Projekt integriert. Die zentralen Komponenten sind der Raspberry Pi 5 (kompakt, kostengünstig, energieeffizient), der Honeywell Eclipse MS5145 Barcode-Scanner (zuverlässig, schnell, Plug-and-Play) und der Zebra ZD410 Etikettendrucker (platzsparend, USB-fähig).

Diese Plattform gewährleistet eine robuste, wartungsarme und erweiterbare Lösung für die Menükennzeichnung.

### 4.2 Architekturdesign

Die Systemarchitektur, einschließlich der Hardware- und Softwarekomponenten sowie deren Interaktionen, wird im Deployment-Diagramm ([Anhang A8](#), [Seite iv](#)) dargestellt.

**Deployment-Diagramm:** Die Anwendung läuft auf einem Raspberry Pi 5 und umfasst folgende Systemkomponenten:

1. **Raspberry Pi 5:** Zentrale Einheit, die die Applikation für die lokale und [Web-Anwendung](#) enthält. Die JSON-Datenbank speichert die Menüinformationen und wird von beiden Anwendungen genutzt. Über [USB](#) sind der Barcode-Scanner und der Etikettendrucker angeschlossen.
2. **Lokale Anwendung:** Sie ist verantwortlich für das Scannen, Verarbeiten und Drucken der Menüinformationen und kommuniziert über [USB](#) mit den Geräten. Außerdem greift sie auf die JSON-Datenbank zu und unterstützt optionale E-Mail-Benachrichtigungen über [SMTP](#).
3. **Web-Anwendung:** Wird als Webserver auf dem Raspberry Pi ausgeführt und stellt eine Web-Oberfläche für die Menüverwaltung bereit, die über [HTTP](#) mit den Clients kommuniziert.
4. **Client:** Externe Benutzergeräte, die über einen Browser auf die Web-Anwendung zugreifen und E-Mail-Benachrichtigungen empfangen.

Das **Paket-Diagramm** im [Anhang A.8](#) auf [Seite v](#) veranschaulicht die Projektstruktur. Es umfasst nur relevante Elemente.

Die [lokale Anwendung](#) folgt einer modularen [monolithischen Architektur](#), was einfache Entwicklung, Wartung und Debugging ermöglicht und eine gute Modularität für spätere Anpassungen bietet.

Die [Web-Anwendung](#) ist plattformunabhängig und bietet durch zentrale Wartung und Flexibilität bei Anpassungen Vorteile. Sie folgt dem MVC-Architekturmuster, wobei das Model (**M**) die Menüdaten verwaltet, die View (**V**) die Web-Oberfläche darstellt und der Controller (**C**) die Benutzerinteraktionen steuert.

**Bewertung und Auswahl des Web-Frameworks:** Zur Auswahl eines geeigneten Web-Frameworks wurde eine Recherche durchgeführt, auf deren Basis eine [Entscheidungsmatrix](#) erstellt wurde.

Kriterium	Gin	Fiber	Echo	net/http
<b>Leichtgewichtig &amp; performant</b>	9	9	8	10
<b>Einfache Routenverwaltung</b>	10	8	10	5
<b>Gute Dokumentation &amp; Community</b>	10	7	8	10
<b>Gesamtbewertung (Summe)</b>	<b>29</b>	24	26	25

Tabelle 3: Entscheidungsmatrix

Die Entscheidung fiel auf **Gin**, da es leichtgewichtig und performant ist, eine einfache Routenverwaltung bietet und gute Dokumentation sowie eine starke Community hat.

Die verwendeten Frameworks und Bibliotheken umfassen **tarm/serial** und **bufio** für die serielle Kommunikation mit dem Barcode-Scanner, **os/exec** für die Ausführung von Druckbefehlen, sowie **log** für die Protokollierung. Zur Verarbeitung der [Datenbank](#) wird **encoding/json** genutzt, und für die Erstellung von HTML-Templates kommt **html/template** zum Einsatz. E-Mail-Benachrichtigungen werden über **net/smtp** versendet, während **testing** für [Unit-Tests](#) verwendet wird. Für [Mutationstests](#) wird die Bibliothek **gramontina/oze** eingesetzt.

### 4.3 Lokale Anwendung

Die [lokale Anwendung](#) wird modular aufgebaut, um eine klare Trennung der Verantwortlichkeiten sicherzustellen. Die Hauptmodule sind Barcode-Scanner, Formatter und Drucker, wobei jede Komponente eine eigene **Steuerungsfunktion** hat: StartBarcodeScannen(), StartFormatter(), StartDrucker().

#### Entwurfsmuster

Diese Funktionen kapseln die Logik der jeweiligen Komponenten und starten den Hauptprozess. Die Architektur folgt dem Entwurfsmuster [Fassade](#), da sie eine vereinfachte Schnittstelle für den Gesamtprozess bietet. Das Entwurfsmuster [Strategie](#) wird für den Druckvorgang angewendet, um verschiedene Druckstrategien flexibel zu nutzen. Zur zentralen Protokollierung kommt das Entwurfsmuster [Singleton](#) zum Einsatz, um eine einzige Logger-Instanz zu gewährleisten.

#### Programmieransätze

Interfaces und Strukturen sorgen für eine strukturierte Entwicklung, während funktionale Konzepte wie [Funktionen höherer Ordnung](#), Funktionen als [First-Class Citizens](#) und Rekursionen genutzt werden. Die [lokale Anwendung](#) umfasst den gesamten Prozess vom Barcode-Scan über die Menünummer-Identifikation bis hin zum Etikettendruck. Fehlt eine Zuordnung in der [Datenbank](#), wird eine E-Mail-Benachrichtigung ausgelöst. Die Prozesse werden durch Klassendiagramm, Aktivitätsdiagramm und Sequenzdiagramm modelliert.

#### Klassendiagramm

Das Klassendiagramm (siehe [Anhang A.9, Seite vi](#)) stellt die zentralen Komponenten der lokalen Anwendung und deren Interaktionen dar:

Der **Barcode-Scanner** erfasst Barcodes, während der **Formatter** die Menünummer extrahiert. Der **Drucker** erstellt die Druckbefehle, und das **Benachrichtigungssystem** (EmailVersanddienst, EmailNachrichtService) versendet E-Mails, wenn Daten unvollständig sind. Der **TimerBenachrichtigungsManager** verwaltet verzögerte Benachrichtigungen, und der **Logger** protokolliert alle relevanten Systemereignisse.

#### Aktivitätsdiagramm

Das Aktivitätsdiagramm (siehe [Anhang A.10, Seite vii](#)) beschreibt den Ablauf der Menükennzeichnung:

1. Der Barcode wird gescannt.
2. Die ersten fünf Ziffern werden extrahiert.
3. Das System prüft die [Datenbank](#):
  - a. Bei einem Treffer wird das Etikett mit Gerichtsnamen gedruckt.
  - b. Andernfalls erfolgt der Druck ohne Namen.

### Sequenzdiagramm

Das Sequenzdiagramm (siehe [Anhang A.11, Seite vii](#)) zeigt den Ablauf des Barcode-Scannens und der E-Mail-Benachrichtigung innerhalb der Funktion **StarteBarcodeScannen()**. Die Funktion **StarteBarcodeScannen(barcodeChannel)** wird in der Hauptmethode aufgerufen, wobei eine Scanner-Instanz erstellt und mit **initialisiereScanner(geraet, baudRate)** konfiguriert wird. Der SMTP-Dienst wird mit **ErstelleObjektSmtpEmailDienst(serverDaten, userDaten)** instanziert. Der Timer-Handler wird ebenfalls mit **ErstelleTimerBenachrichtigungsHandler(dauer, smtpEmailDienst)** erstellt. In der Barcode-Scan-Schleife wird fortlaufend **scanneBarcode()** aufgerufen. Nach jedem Scan wird **SetzeTimerZurueckUndSendEmail(timer)** aufgerufen. Läuft der Timer ab und existieren unvollständige Gerichte (**len(UnvollstaendigeGerichteMap) != 0**), wird **SendEmail(emailNachricht)** ausgeführt.

Die strukturierte Umsetzung gewährleistet klare Zuständigkeiten und eine wartbare Architektur.

### 4.4 Datenmodell

Das [Entity-Relationship-Modell](#) zeigt, dass das Datenmodell **einen Entitätstyp mit zwei Attributten** umfasst. Daher wurde die Speicherung in einer JSON-Datei gewählt, da sie für die geringe Datenmenge effizient ist und den Verwaltungsaufwand im Vergleich zu relationalen Datenbanken reduziert. Die Key-Value-Struktur von [JSON](#) eignet sich besonders gut für die Speicherung von Menünummern und Gerichtsnamen. Auf dem Raspberry Pi ist [JSON](#) vorteilhaft, da es wenig Rechenleistung benötigt und direkt mit Go verarbeitet werden kann, während eine relationale Datenbank zusätzliche Ressourcen beanspruchen würde.

Die **CRUD-Operationen** (Create, Read, Update, Delete) werden selbst implementiert. Zur Vermeidung von Dateninkonsistenzen und [Race Conditions](#) wird [Mutex](#) verwendet, um sicherzustellen, dass immer nur eine [Goroutine](#) gleichzeitig auf die JSON-Datei zugreift.

#### Datenstruktur:

Die JSON-Datei speichert Menüdaten im Schlüssel-Wert-Format:

```
{  
    "12345": "Spaghetti Bolognese",  
    "98765": "Vegetarische Lasagne"  
}
```

- **Schlüssel** (Menünummer): Eindeutige Identifikationsnummer des Gerichts.
- **Wert** (Gerichtsname): Name des Gerichts.

Nicht zugeordnete Nummern werden in einer separaten Map gespeichert, um unvollständige Einträge später zu ergänzen.

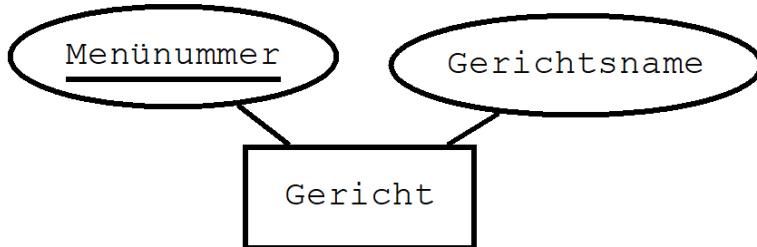


Abbildung 1: Entity-Relationship-Modell

### 4.5 Web-Anwendung

Für die Menüverwaltung wird eine browserbasierte [Web-Anwendung](#) entwickelt, die eine einfache Bedienung ohne lokale Installation ermöglicht – eine zentrale Vorgabe des Auftraggebers. Die Benutzeroberfläche wird intuitiv gestaltet und an den Arbeitsablauf der Mitarbeitenden angepasst.

**Struktur und Funktionen:** Die Anwendung umfasst:

1. **Loginseite:** Anmeldung für Administratoren.
2. **Startseite:** Begrüßungsseite mit Unternehmenslogo.
3. **CRUD-Seiten:** Die Kernfunktionen der Anwendung:
  - **Anzeigen:** Übersichtliche Darstellung der Gerichte in Tabellenform.
  - **Hinzufügen:** Erfassung neuer Gerichte mit Validierung der fünfstelligen Gerichtsnummer.

- **Aktualisieren:** Bearbeitung bereits gespeicherter Gerichte mit Fehlermeldungen bei fehlerhaften Eingaben.
- **Löschen:** Entfernung von Gerichten aus der [Datenbank](#) mit Fehlermeldung bei ungültigen Eingaben.

Die **Steuerung** erfolgt über den Controller, der als Vermittler zwischen der Benutzeroberfläche und der Geschäftslogik fungiert, HTTP-Anfragen verarbeitet, HTML-Seiten rendert und Datenbankzugriffe steuert. Der Webserver wird auf einem Raspberry Pi laufen und alle Benutzerinteraktionen werden zur Fehlerverfolgung protokolliert.

Die **Benutzerfreundlichkeit** wird durch kleine Hinweise unterhalb der Such- und Eingabefelder sowie durch Beispieleingaben in den Eingabefeldern gefördert.

Die **Usability-Richtlinien** umfassen eine intuitive Bedienung mit klaren Buttons und Icons, hohen Kontrastwerten sowie großen Schaltflächen für eine gute Lesbarkeit. Das [Corporate Design](#) des Unternehmens wird durch das Logo und firmenspezifische Symbole eingehalten.

Die Navigation erfolgt über eine horizontale Menüleiste mit einem Logout-Button oben rechts. Da die Anwendung ausschließlich am Desktop genutzt wird, ist keine Responsivität erforderlich.

Das **Zusammenspiel** der Seiten wurde händisch skizziert. Die **einzelnen Seiten** wurden mit [Figma](#) entworfen. Die Oberflächenentwürfe sind im [Anhang A.12](#) auf [Seite viii](#) dokumentiert.

Die Anwendung wird auf **Google Chrome**, **Mozilla Firefox** und **Microsoft Edge** getestet, um die Kompatibilität sicherzustellen.

## 4.6 Maßnahmen zur Qualitätssicherung

Die Qualitätssicherung stellt sicher, dass die entwickelte Lösung funktionsfähig, zuverlässig und benutzerfreundlich ist. Dafür werden während der Entwicklung und in der Testphase verschiedene Maßnahmen ergriffen:

### Sicherstellung der Code-Qualität:

Der **Database-First-Ansatz (Test Last)** sieht vor, frühzeitig [Unit-Tests](#) für Kernfunktionen und [Mutations-Tests](#) zur Bewertung der Qualität der Unit-Tests durchzuführen. Die Unit-Tests folgen der [AAA-Regel](#) und nutzen Test-Tabellen zur systematischen Abdeckung verschiedener Testfälle. Ein erfahrener Entwickler führt [Code-Review](#) durch, um Performance, Wartbarkeit und mögliche [Code-Smells](#) zu überprüfen.

### Systemvalidierung:

Ein [Proof of Concept](#) (PoC) validiert die geplante Hardware (Barcode-Scanner, [SBC](#), Etikettendrucker) mit einem frühen Prototyp. Ein [Lasttest](#) für den Drucker und die [Datenbank](#) simuliert den gleichzeitigen Zugriff auf die Datenbank und prüft das Druckerverhalten unter Last.

### Web-Anwendungsprüfung:

Die [Web-Anwendung](#) wird durch [Black-Box-Test](#) in Google Chrome, Firefox und Edge auf Benutzerführung und typische Anwendungsfälle getestet. Zum Abschluss erfolgt ein [Akzeptanztest](#) unter realen Bedingungen, bei dem die Anwendung im echten Essensempfangsprozess getestet wird, um die zuverlässige Zusammenarbeit aller Komponenten sicherzustellen.

Die wichtigsten Maßnahmen wurden tabellarisch als Testkonzept erfasst. Während der Umsetzung des Projekts und Qualitätssicherung wurden diese erfolgreich durchgeführt (siehe [Anhang A.25 Testkonzept](#) auf [Seite xix](#)).

Diese Maßnahmen gewährleisten, dass die Anwendung den Anforderungen entspricht und effizient eingesetzt werden kann.

## 4.7 Pflichtenheft

Am Ende der Entwurfsphase wurde auf Basis der erstellten Entwürfe ein Pflichtenheft ausgearbeitet. Es dokumentiert die detaillierte Umsetzung der im Abschnitt [3.5 \(Lastenheft\)](#) definierten Anforderungen. So kann am Projektende überprüft werden, ob alle Anforderungen erfüllt und wie vorgesehen umgesetzt wurden. Ein Auszug ist im [Anhang A.13: Pflichtenheft \(Auszug\)](#) auf [Seite ix](#) enthalten.

## 5 Implementierungsphase

In der Implementierungsphase wurden alle Artefakte der Entwurfsphase genutzt. Die Software wird direkt auf dem **Raspberry Pi 5** entwickelt und ausgeführt, sodass kein separates Deployment erforderlich ist. Der Zugriff erfolgte über eine **Remotedesktopverbindung** mit hinterlegten Anmeldedaten als Sicherheitsmaßnahme von einem Windows-Rechner aus. Die dem Raspberry Pi zugewiesene statische IP-Adresse stellt die dauerhafte Erreichbarkeit für Wartungszwecke sicher.

Zu Beginn wurde das **Go-Projekt** angelegt.

### Nebenläufige Verarbeitung

Zur effizienten und reaktionsschnellen Verarbeitung kamen Goroutinen und Channels zum Einsatz.

- Goroutinen ermöglichen parallele Aufgaben wie Barcode-Scannen, Datenformatierung und Etikettendruck, wodurch eine ressourcenschonende Nebenläufigkeit ohne separate Betriebssystem-Threads erreicht wird.
- Channels gewährleisten die synchronisierte Kommunikation zwischen den parallel laufenden Komponenten, ohne komplexe Synchronisation.

Nach Abschluss der Entwicklung wurde das Programm als **Systemdienst** eingerichtet, sodass es beim Start des Raspberry Pi automatisch ausgeführt wird. Dies sorgt für Zuverlässigkeit und Automatisierung, da das Programm nach einem Neustart oder Absturz eigenständig neu startet. Somit reicht es aus, den Raspberry Pi einzuschalten, um den Etikettendruckprozess zu starten und die Web-Anwendung bereitzustellen.

### 5.1 Prototyping mit Node-Red

Das Prototyping könnte auch in der Entwurfsphase erfolgen, wurde jedoch hier platziert, da es den ersten Schritt der Implementierung darstellt und mit dem Prototyp die erste funktionierende Version entwickelt wurde. Zunächst wurde die Hardware mit Unterstützung eines Support-Mitarbeiters eingerichtet. Dabei wurden CUPS ([Common Unix Printing System](#)), der Druckertreiber und die Druckereinstellungen konfiguriert. Ein **symbolischer Link** wurde erstellt, um sicherzustellen, dass der Barcode-Scanner immer unter dem gleichen Namen erreichbar bleibt.

Beim [Proof of Concept \(PoC\)](#) trat eine Herausforderung auf: Der Prototyp reagierte nicht auf Scan-Vorgänge. Es stellte sich heraus, dass der Scanner in den seriellen Modus umgestellt werden musste. Nach Rücksprache mit dem Support und Recherche konnte der Modus durch Scannen eines speziellen Barcodes erfolgreich geändert werden.

#### Node-Red Flow: Serieller Barcode-Scanner

Der im [Anhang A.14](#) auf [Seite x](#) dargestellte **Node-Red-Flow** verarbeitet Barcodes und generiert Druckbefehle für den Zebra-Etikettendrucker:

1. **Test-Eingabe:**
  - Inject-Knoten (Test\_Barcodes): Simuliert die Barcode-Eingabe.
2. **Barcode-Erfassung und Verarbeitung:**
  - serial-in-Knoten (Barcode Scanner): Liest den Barcode über die serielle Schnittstelle.
  - function-Knoten (Barcode Zusammenstellen): Setzt die Zeichen zu einem vollständigen Barcode zusammen und leitet ihn weiter.
  - function-Knoten (Barcode Schneiden): Extrahiert die ersten fünf Zeichen als Menünummer.
3. **Debugging:**
  - debug-Knoten (Barcode zusammengestellt): Zeigt den vollständigen Barcode.
  - debug-Knoten (Barcode abgeschnitten): Zeigt die extrahierte Menünummer.
4. **Barcode-Druck:**
  - function-Knoten (Barcode zu ZPL-Befehl): Erstellt den Druckbefehl im ZPL-Format.
  - exec-Knoten (Drucken): Sendet den Befehl an den Zebra-Drucker.

Nach diesen Schritten wurde die Hardware validiert und die einwandfreie Funktion sichergestellt.

### 5.2 Implementierung der lokalen Anwendung

Die lokale Anwendung wurde gemäß den in [4.3 Lokale Anwendung](#) beschriebenen Konzepten und Mustern implementiert. Die Hauptkomponenten (Barcode-Scanner, Formatter, Drucker) wurden als eigenständige Module umgesetzt. Ihre Steuerung erfolgt über dedizierte Funktionen, die dem [Fassade](#)-

Muster folgen. Obwohl die zugrunde liegenden Strukturen und Interfaces mehrere Methoden enthalten und somit verschiedene Aufgaben und Verantwortlichkeiten abdecken, wurde bei der Entwicklung jeder einzelnen Methode das **Single Responsibility Principle** beachtet. Jede Methode wurde möglichst auf eine spezifische Aufgabe beschränkt. Zusätzlich zu den geplanten Methoden wurden weitere Funktionen und ergänzende Logik (z. B. das Strategie-Muster) integriert (siehe Anhang A.15: Klassendiagramm-Drucker auf Seite xi).

#### **Strategie-Muster für den Drucker:**

Der Druckprozess wurde flexibel gestaltet, indem zwei Druckstrategien implementiert wurden (siehe Anhang A.16 Codebeispiel: Drucksteuerung auf S. xii, Zeilen 6 und 10). Hinweis: Kommentare in allen Codebeispielen wurden reduziert, und die Protokollierungslogik entfernt, um die Lesbarkeit zu erhöhen.

**Verwendete funktionale Konzepte:** (siehe Anhang A.16 Codebeispiel: Drucksteuerung, S. xii): Funktionen als First-Class Citizens (Zeile 38), Higher-Order Functions (Zeile 39), Rekursionen (Zeile 55).

Die korrekte Anwendung des Strategie-Musters wurde durch **Unit-Test** überprüft, in dem die Methode `erzeugeZPLBefehl()` getestet wurde. Dies stellt sicher, dass die passenden Druckstrategien ausgewählt und die ZPL-Befehle korrekt erzeugt werden (siehe Anhang A.17 Codebeispiel: Testfall zur Befehlserzeugung mit Strategie-Muster und Konsolenausgabe, Seite xiii).

**Protokollierung & Singleton-Pattern:** Alle Ereignisse werden mit Zeitstempel und den Log-Leveln INFO, WARNUNG und FEHLER protokolliert. Das Singleton-Pattern sorgt für eine zentrale Logger-Instanz, um doppelte Initialisierungen zu vermeiden und Ressourcen zu sparen (Anhang A18: Singleton-Logger auf Seite xiv). Info-Nachrichten sind in Zeile 36; andere Log-Levels folgen demselben Prinzip).

#### **Sicherheitsmaßnahmen und Konfigurationsmanagement:**

- E-Mail-Adressen und sensible Daten sind nicht hartkodiert, sondern über Umgebungsvariablen verwaltet.
- Diese Variablen werden aus einer **.env-Datei** geladen, die nicht in das Versionskontrollsyste hochgeladen wird.

### 5.3 Implementierung des Datenmodells

Die Menüdaten werden in einer **JSON-Datei** gespeichert, die als leichtgewichtige Datenbanklösung dient. Die Daten aus der JSON-Datei werden als **map[string]string** geladen. Der string-basierte Schlüssel verhindert das Entfernen führender Nullen. Ein Mutex schützt vor Race Conditions und sichert Lese- und Schreiboperationen.

#### **CRUD-Operationen:**

- **Erstellen/Aktualisieren:** `GerichtInJSONHinzufuegen()`
- **Lesen:** `LadeGerichteAusDatei()`
- **Löschen:** `GerichtAusJSONLoeschen()` (siehe Anhang A.19 Codebeispiel: Löschfunktion)

Zusätzliche Funktionen wie `speichereGerichteInJsonDatei()` und `BeleereMapVorhandeneGerichte()` gewährleisten eine effiziente Menüverwaltung. Änderungen werden nach jeder CRUD-Operation gespeichert.

### 5.4 Implementierung der Web-Anwendung

Die Web-Anwendung wurde gemäß der in der Entwurfsphase definierten Struktur umgesetzt. Sie basiert auf dem Gin-Webframework in Go, das auf dem Raspberry Pi als Webserver dient.

Das **Backend** verarbeitet HTTP-Anfragen und stellt eine REST-API bereit:

- **Authentifizierung:**
  - GET /login, POST /login: Login-Seite und Anmeldung
- **Gerichtsverwaltung (/admin) – Beispiele für Anzeige, Hinzufügen und Aktualisierung**
  - GET /admin/table: Zeigt gespeicherte Gerichte.
  - GET /admin/add: Formular zum Hinzufügen.
  - POST /admin/add: Erstellt ein neues Gericht. (siehe Anhang A.20: Codebeispiel: Gericht hinzufügen mit Go und Gin, Seite xv)
  - GET /admin/update: Suchseite für die Aktualisierung eines Gerichts
  - GET /admin/updateinjson: Seite zur Aktualisierung eines Gerichts
  - POST /admin/update: Sucht ein Gericht zur Aktualisierung
  - POST /admin/updateinjson: Speichert die Änderungen am Gericht

Die Middleware **adminMiddleware()** sichert den Admin-Bereich ab.

Die **Benutzeroberfläche** basiert auf den in der Entwurfsphase erstellten Mock-Ups. **HTML-Templates** (Siehe [Anhang A21: Codebeispiel: HTML-Template für Fehler- und Erfolgsmeldungen, Seite xvi](#)) in Kombination mit **Go**, **Gin** und **Bootstrap** sorgen für eine konsistente Darstellung. Die Seiten wurden, wie in Abschnitt [4.5 Web-Anwendung](#) geplant, umgesetzt:

- **Login-Seite (/login):** Anmeldung mit Validierung.
- **Startseite (/admin):** Begrüßung, Firmenlogo, Navigation.
- **Gerichtsverwaltung (Create, Read, Update, Delete):**
  - Anzeigen (/admin/table)
  - Hinzufügen (/admin/add)
  - Aktualisieren/Löschen (/admin/update, /admin/delete) (Screenshots zur Löschenseite sind im [Anhang A.22 Ablauf der Menülöschung in der Webanwendung, Seite xvii](#))

Die **horizontale Menüleiste** ermöglicht schnellen Zugriff auf Funktionen, der **Logout-Button** ist oben rechts.

**Corporate Design:** Die Anwendung enthält **Firmenlogos und Symbole** (Monsterchen-Symbole oben rechts, Logo als Buchstabe oben links, Logo als Wort unten).

Nach der Erstellung der Web-Anwendung wurde die Implementierungsphase abgeschlossen.

## 6 Qualitätssicherung und Abnahme

Zur Sicherstellung der Funktionalität und Benutzerfreundlichkeit wurden verschiedene Tests durchgeführt:

- [Unit-Tests](#) und [Mutationstests: AAA-Regel](#), Test-Tabellen
- [Lasttest](#): Simuliert gleichzeitigen Datenbankzugriff und prüft das Druckerverhalten unter Last
- [Code-Review](#): Prüfung auf Verständlichkeit, Lesbarkeit und [Code-Smells](#). Korrekturen umgesetzt oder als **TODOs** dokumentiert
- [Black-Box-Test](#): Test durch HR-Mitarbeiterin ohne technisches Vorwissen
- [Akzeptanztest](#): Live-Test beim Essensempfang (siehe [Anhang 23: Prozessablauf des Akzeptanztests \(Bildcollage\), Seite xviii](#))

Siehe [Anhang A.24 Testkonzept auf Seite xix](#).

**Das Feedback zur Abnahme** wurde aus [Code-Review](#), [Akzeptanztest](#) und [Black-Box-Test](#) eingeholt. Nach erfolgreicher Prüfung erfolgte die **offizielle Abnahme durch den Auftraggeber**. Dabei wurde die Anwendung in Betrieb gezeigt und Verpackungen mit neuen Etiketten präsentiert.

### 6.1 Einführung und Schulung

Die Entwicklung und Konfiguration erfolgten direkt auf dem Zielsystem.

#### Benutzerschulung

- Vor dem [Akzeptanztest](#) wurden Mitarbeiter geschult, um die Anwendung direkt zu testen.
- Nach dem [Black-Box-Test](#) wurde die Nutzung der [GUI](#) mit dem Benutzerhandbuch erklärt.

## 7 Dokumentation

Es wurde ein **Benutzerhandbuch** erstellt. Das Handbuch enthält detaillierte Anleitungen mit Screenshots und Erklärungen, die die Bedienung der Anwendung sowie mögliche Interaktionen erläutern. Außerdem beschreibt es den Etikettierungsprozess sowie den Umgang mit der Web-Anwendung und ergänzt diesen durch grafische Darstellungen, darunter Fotos und Screenshots der Benutzeroberflächen. Die **Entwicklerdokumentation** wurde direkt im Quellcode in Form von Kommentaren verfasst.

A Anhang

## A.1 Verpackungsbeschriftung - handschriftlich vs. automatisiert



Abbildung 2: Verpackungsbeschriftung - handschriftlich vs. automatisiert

## A.2 Verwendete Ressourcen

## **Hardware:**

- Barcode-Scanner Honeywell Eclipse MS5145 LS USB – Erfassung der Barcodes.
  - Etikettendrucker ZEBRA ZD 410 – Drucken der generierten Etiketten.
  - Raspberry Pi 5 – Entwicklung und Ausführung des Programms sowie Bereitstellung des Webservers.

## **Software:**

- Betriebssystem:
    - Raspberry Pi 5: Debian GNU/Linux 12.
    - Arbeitsrechner: Windows 11 Pro.
  - Bootstrap – CSS-Framework zur Gestaltung der Benutzeroberfläche.
  - CUPS ([Common Unix Printing System](#)) – Drucksystem zur Verwaltung von Druckaufträgen.
  - Dia – Tool zur Erstellung technischer Diagramme und Modelle.
  - Figma – Design-Software zur Erstellung von UI-Mockups für die Web-Anwendung.
  - Gin-Framework – Web-Framework zur Bereitstellung des Webservers für die Anwendung.
  - GitHub – Versionskontrollsystem zur Verwaltung des Quellcodes und Nachverfolgbarkeit von Änderungen.
  - Google Chrome, Mozilla Firefox, Microsoft Edge – Webbrowser zum Testen und Anzeigen der Web-Anwendung.
  - gtramontina/ooze – Tool zur Durchführung von Mutationstests.
  - JSON-basierte Datenbank – Speicherung und Verwaltung der Menü- und Benutzerinformationen.
  - Microsoft Designer - Tool zur Bearbeitung und Gestaltung von Bildern
  - Node-Red – Visuelle Entwicklungsumgebung zur Prototypenerstellung und frühen Validierung von Datenflüssen.
  - Redmine – Projektmanagementsoftware zur Planung und Verwaltung von Aufgaben.
  - Remotedesktopverbindung (Windows) – Fernzugriff auf den Raspberry Pi.
  - testing – Go-Testpaket zur Durchführung von Unit-Tests.
  - Visual Studio Code – Entwicklungsumgebung zur Programmierung und Fehleranalyse.

## Personal:

- Auftraggeber – Festlegung der Anforderungen und Abnahme des Programms.
  - Entwickler/-in – Durchführung des Code-Reviews.
  - HR-Mitarbeiterin – Durchführung eines Black-Box-Tests zur Systemprüfung.

- Support-Mitarbeiter – Technische Unterstützung bei der Einbindung der Hardware.
- Umzuschulender Fachinformatiker für Anwendungsentwicklung – Umsetzung des Projekts.

**Sonstiges:**

- Büroarbeitsplatz

### A.3 Aktivitätsdiagramm des Essensempfangs mit der manuellen Menükennzeichnung

Essensempfang

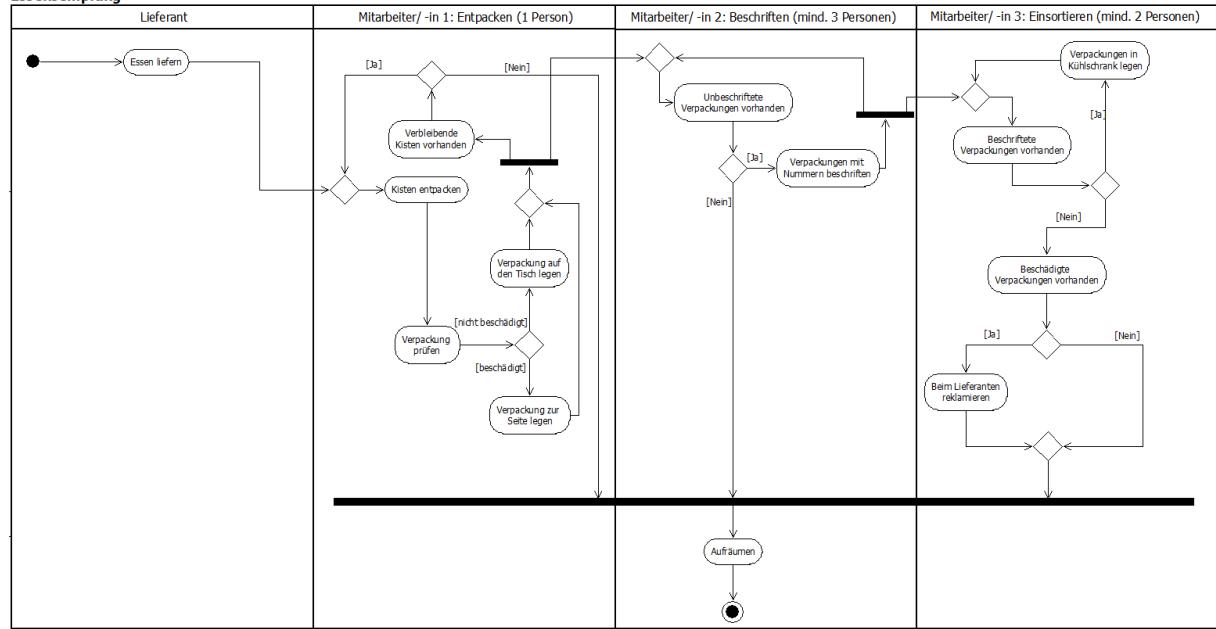


Abbildung 3: Aktivitätsdiagramm des Essensempfangs mit der manuellen Menükennzeichnung

### A.4 Amortisation – Break-Even-Point

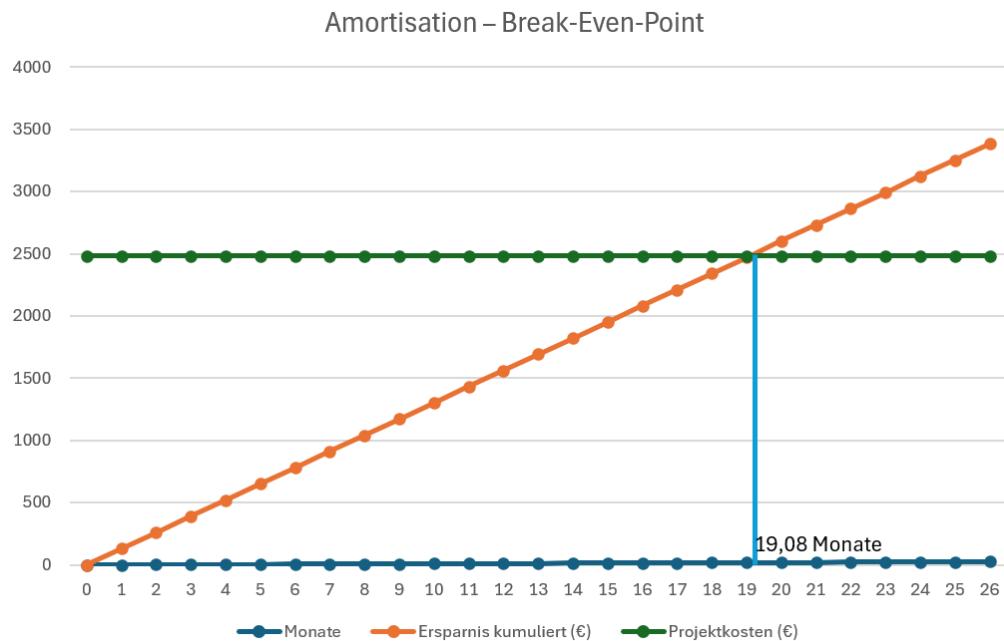


Abbildung 4: Amortisation – Break-Even-Point

### A.5 Use-Case-Diagramm

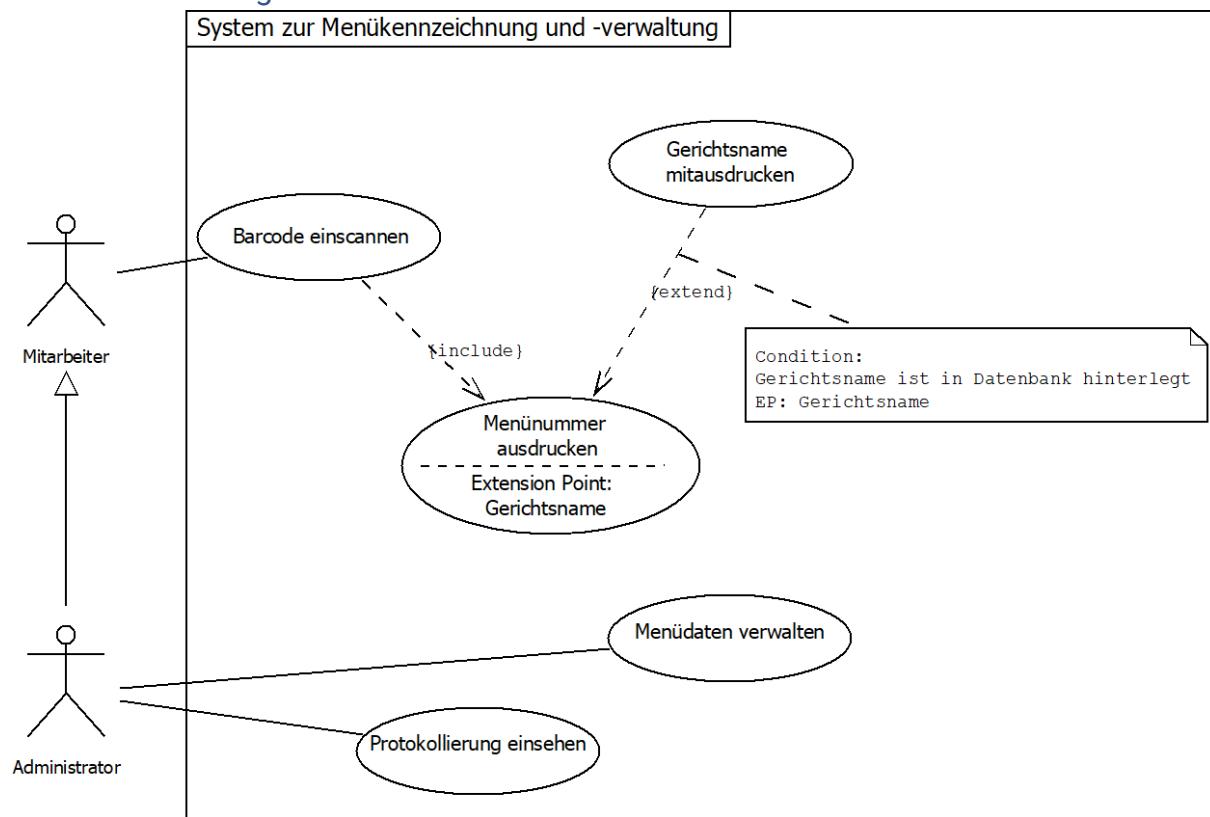


Abbildung 5: Use-Case-Diagramm

### A.6 Lastenheft (Auszug)

- Das System muss den Barcode auf der Oberseite der Menüschahteln auslesen und die ersten fünf Stellen als Menünummer extrahieren.
- Das System muss die Barcodes formatieren, an einen Etikettendrucker übergeben und ein Etikett mit Menünummer sowie – falls vorhanden – dem Namen des Gerichts drucken.
- Das System muss alle Menünummern und Gerichtsnamen in einer lokalen Datenbank speichern.
- Falls eine gescannte Menünummer nicht in der Datenbank existiert oder in der Datenbank ohne Namen hinterlegt ist, muss das System innerhalb von 15 Minuten nach dem letzten Scan eine E-Mail-Benachrichtigung an den Administrator senden.
- Der Druckvorgang muss automatisch nach jedem erfolgreichen Scan ausgelöst werden.
- Die Web-Anwendung muss eine tabellarische Übersicht der gespeicherten Menüdaten bereitstellen.
- Der Administrator muss Menüs suchen, hinzufügen, aktualisieren und löschen können.
- Die Web-Anwendung muss durch Authentifizierung geschützt und nur im Intranet erreichbar sein.
- Die grafische Benutzeroberfläche muss das Corporate Design (Logo und Symbole des Unternehmens) berücksichtigen.
- Alle relevanten Ereignisse, einschließlich Fehlermeldungen und Benutzerinteraktionen, müssen mit Zeitstempel in einer Protokolldatei gespeichert werden.
- Das System muss Ereignisse mit den entsprechenden Log-Leveln INFO, WARNUNG und FEHLER protokollieren.
- Das System muss eine Fehlerprotokollierung und -behandlung enthalten.
- Der Code muss verständlich dokumentiert und ausführlich kommentiert sein.
- Der manuelle Beschriftungsprozess muss vollständig entfallen.

- Die Benutzeroberfläche muss intuitiv bedienbar sein und eine kurze Einarbeitungszeit erfordern.
- Die Software muss in einer im Unternehmen etablierten Programmiersprache entwickelt werden.
- Das System muss autark auf einem Single-Board-Computer (SBC) laufen („All-in-One“-Lösung).
- Das System muss einen Webserver auf dem SBC bereitstellen.
- Die Web-Anwendung muss mit Google Chrome, Mozilla Firefox und Microsoft Edge kompatibel sein.
- Das System muss mit dem Barcode-Scanner Honeywell Eclipse MS5145 LS USB und dem Etikettendrucker ZEBRA ZD 410 kompatibel sein.
- Das System muss modular aufgebaut sein, um spätere Anpassungen zu erleichtern.
- Zur Qualitätssicherung müssen Code-Review, Black-Box-Tests und Akzeptanztests durchgeführt werden.
- Die Ergebnisse von Code-Review, Black-Box-Tests und Akzeptanztests müssen dokumentiert und für die Abnahme als Feedback berücksichtigt werden.

## A.7 Deployment-Diagramm

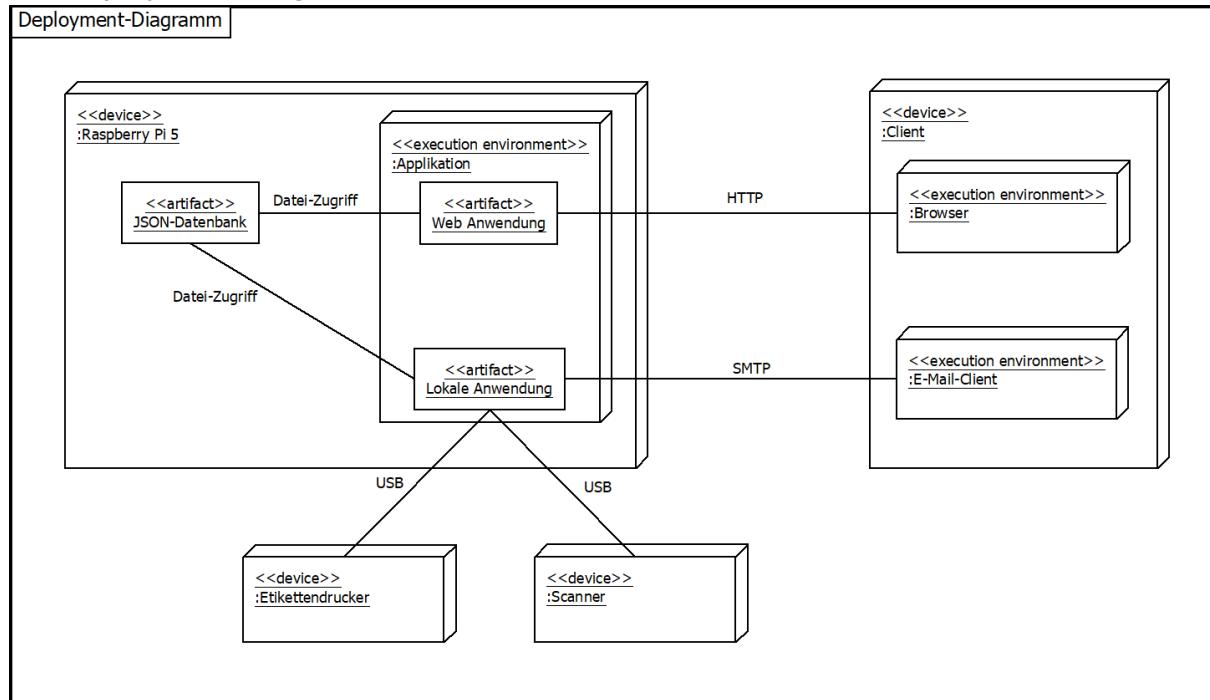


Abbildung 6: Deployment-Diagramm

### A.8 Paket-Diagramm

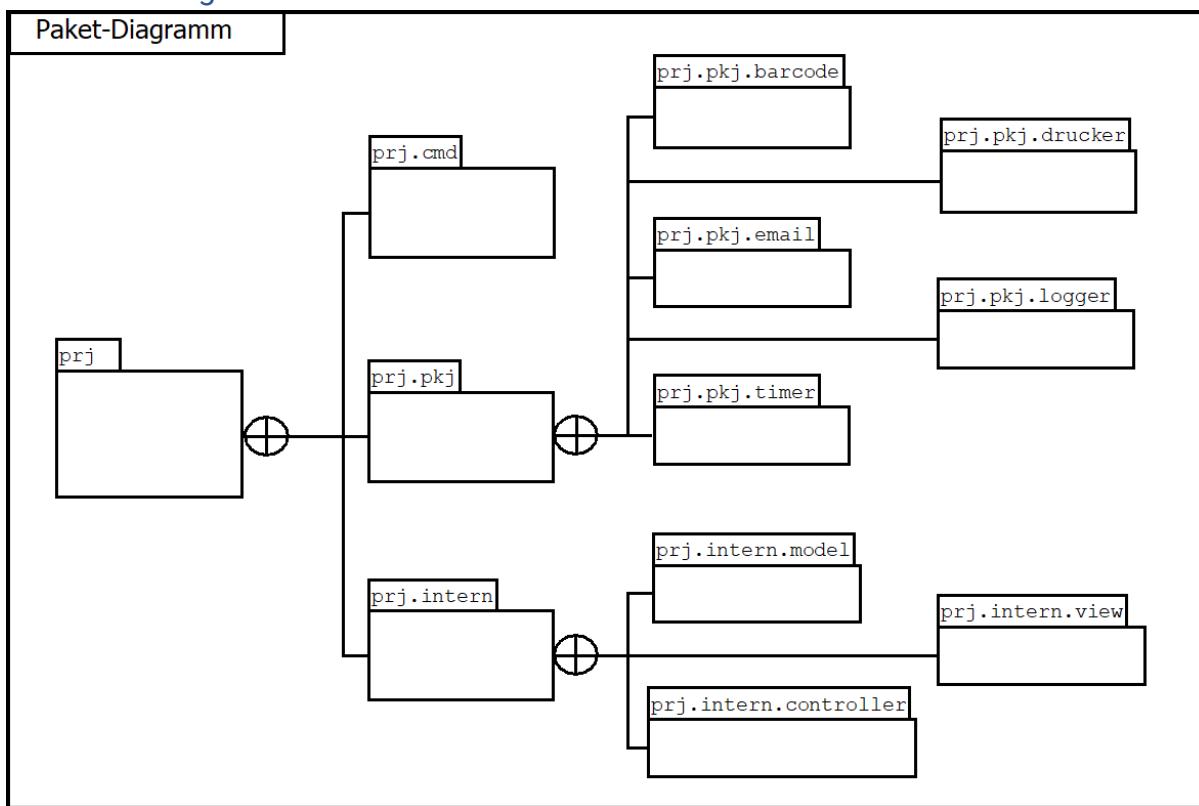


Abbildung 7: Paket-Diagramm

## A.9 Klassendiagramm – Klassen und Schnittstellen

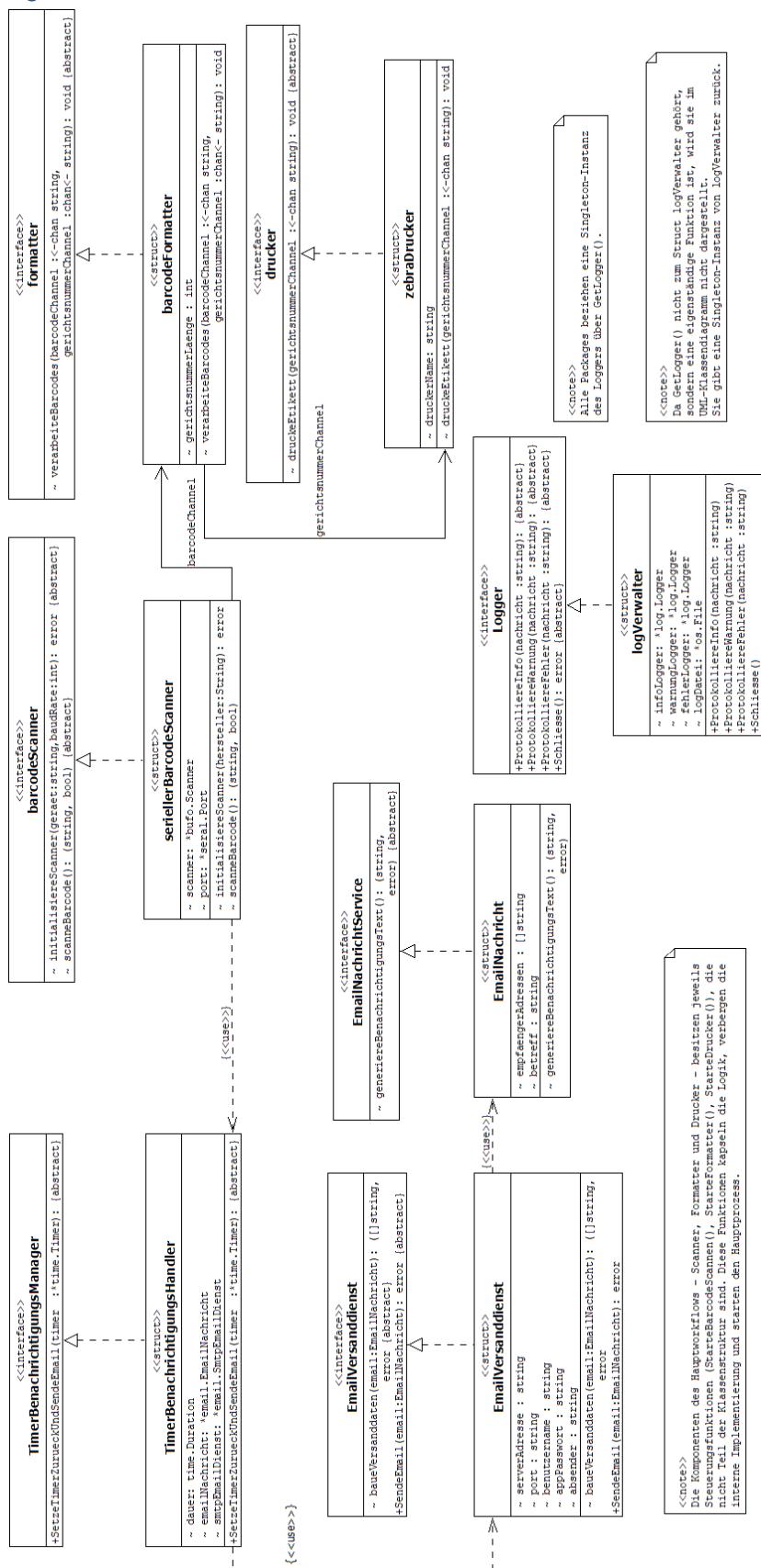


Abbildung 8: Klassendiagramm – Klassen und Schnittstellen

## A.10 Aktivitätsdiagramm – Etikettendruck

### Etikettendruck

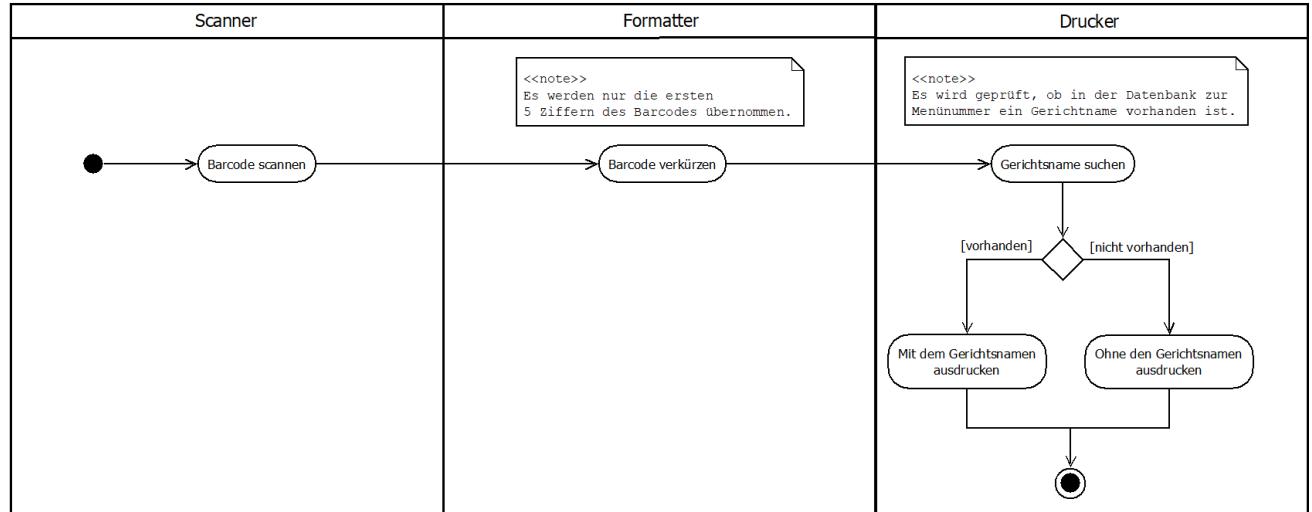


Abbildung 9: Aktivitätsdiagramm – Etikettendruck

## A.11 Sequenzdiagramm – Barcode-Scanning und Benachrichtigung

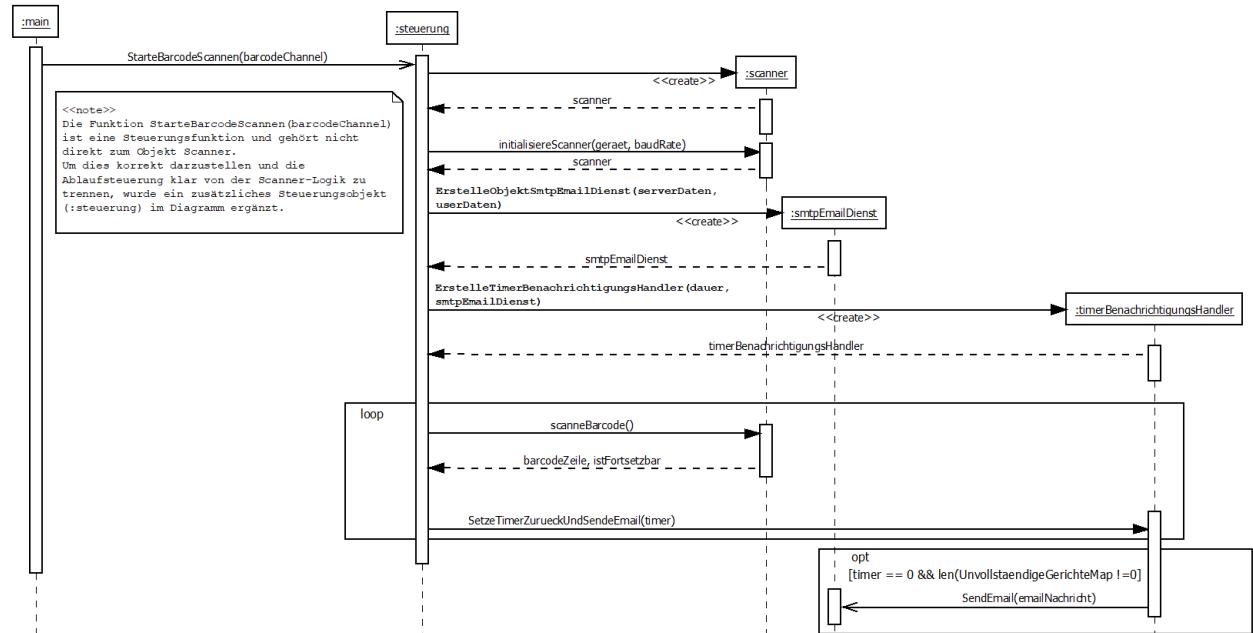


Abbildung 10: Sequenzdiagramm – Barcode-Scanning und Benachrichtigung

## A.12 Oberflächenentwürfe

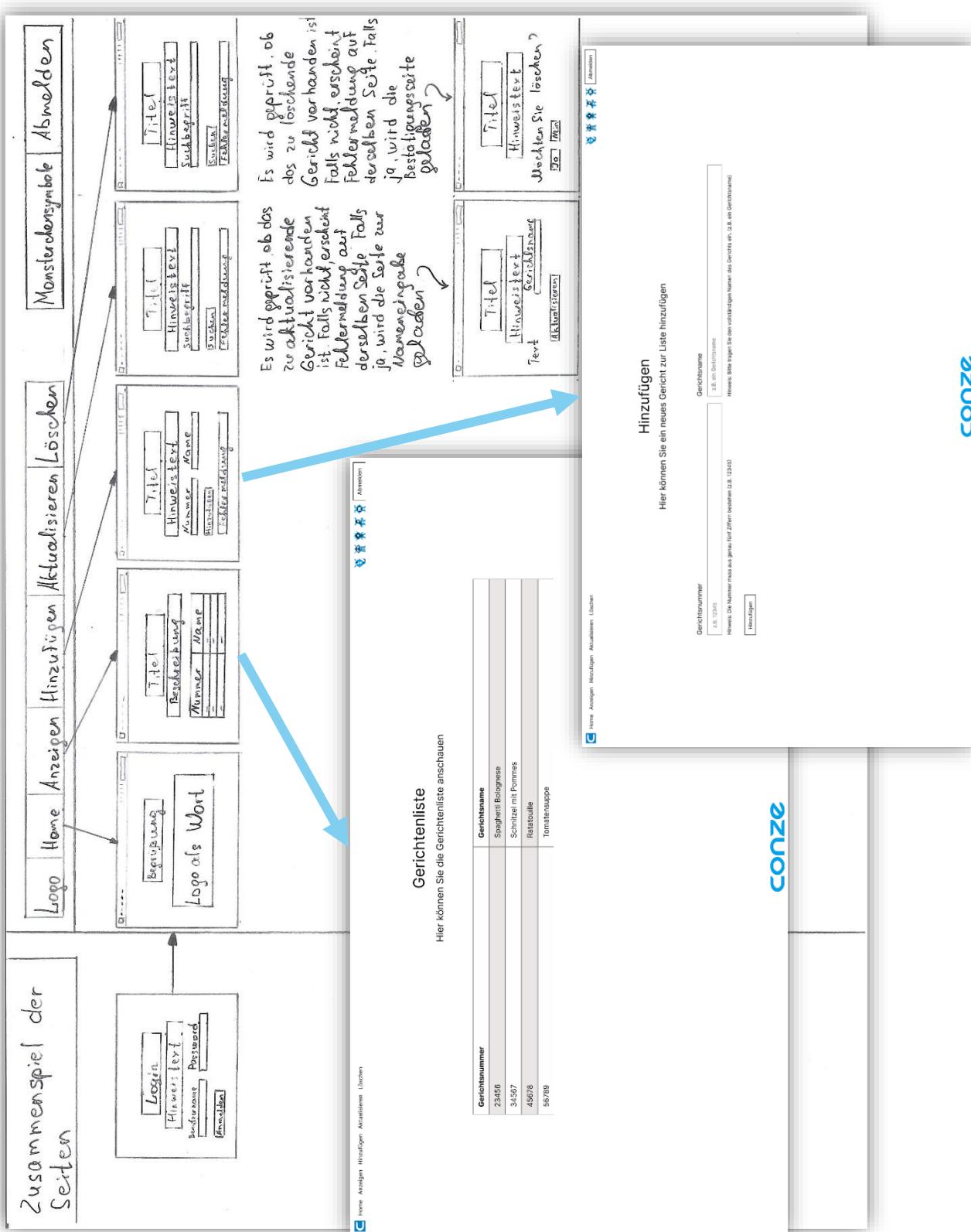


Abbildung 11: Mock-Ups Collage

### A.13 Pflichtenheft (Auszug)

- Die Anwendung wird mit Go implementiert.
- Barcode-Scanning, Datenverarbeitung und Etikettendruck werden als lokale Applikation mit einer modularen monolithischen Architektur entwickelt.
- Die Verwaltung der Menüdaten (Erstellung, Bearbeitung, Löschung) erfolgt über die Web-Anwendung.
- Die Webkomponente wird mit HTML, CSS und Bootstrap entwickelt und folgt dem MVC-Muster.
- Die Web-Anwendung wird nur im Intranet erreichbar sein.
- Die Anwendung wird autark auf dem Single Board Computer (SBC) laufen und für den regulären Betrieb keine Internetverbindung erfordern.
- Als Single Board Computer (SBC) wird ein Raspberry Pi 5 betrieben.
- Der Webserver wird auf Gin (Go-Framework) basieren und wird auf dem Single-Board-Computer (SBC) bereitgestellt.
- Das System wird den Barcode-Scanner Honeywell Eclipse MS5145 LS und den Etikettendrucker ZEBRA ZD410 für die Datenerfassung und den Druck unterstützen.
- Die E-Mail wird 15 Minuten nach dem letzten Scan gesendet, falls Menünummern ohne Gerichtsnamen erfasst wurden oder wenn Menünummern nicht in der Datenbank sind.
- Der Druckvorgang wird automatisch nach jedem erfolgreichen Scan gestartet.
- Alle relevanten Systemereignisse (Scans, Fehler, Änderungen) werden mit Zeitstempel und Log-Level (INFO, WARNUNG, FEHLER) in einer Log-Datei gespeichert.
- Die Nebenläufigkeit wird durch Goroutinen und Channels sichergestellt.
- Der Code wird verständlich kommentiert und dokumentiert.
- Fehlerprotokollierung und -handling werden integriert.
- Die Menüdaten werden in einer JSON-Datei gespeichert.
- Für die Arbeit mit den gespeicherten Daten werden die benötigten CRUD-Operationen (Create, Read, Update, Delete) eigenständig implementiert.
- Race Conditions bei gleichzeitigen Zugriffen werden durch Mutex verhindert.
- Der Zugriff auf den Raspberry Pi sowie auf Web-Anwendung wird durch Authentifizierung geschützt.
- Das Design wird gemäß dem Corporate Design der CONZE Informatik GmbH umgesetzt.
- Die Anwendung wird mit Google Chrome, Mozilla Firefox und Microsoft Edge kompatibel sein.
- Die Benutzeroberfläche wird so gestaltet, dass sie eine intuitive Bedienung mit minimaler Einarbeitungszeit ermöglicht.
- Das System wird modular aufgebaut, um spätere Anpassungen zu erleichtern.
- Menüartikel können jederzeit hinzugefügt, aktualisiert oder gelöscht werden.
- Unit-Tests werden mit der Standardbibliothek „testing“ entwickelt.
- Mutationstests werden mit gtramontina/ooze durchgeführt.
- Das Prototyping zur Hardware-Validierung wird mit Node-Red durchgeführt.
- Die Benutzerführung und die korrekte Funktionsweise typischer Anwendungsfälle werden mit einem Black-Box-Test der Web-Anwendung überprüft.
- Ein Code-Review wird zur Sicherstellung der Code-Qualität durchgeführt.
- Ein Akzeptanztest in der realen Umgebung wird die Erfüllung der Anforderungen validieren.

A.14 Node-RED-Flow zur Barcode-Verarbeitung mit Debug-Ausgabe

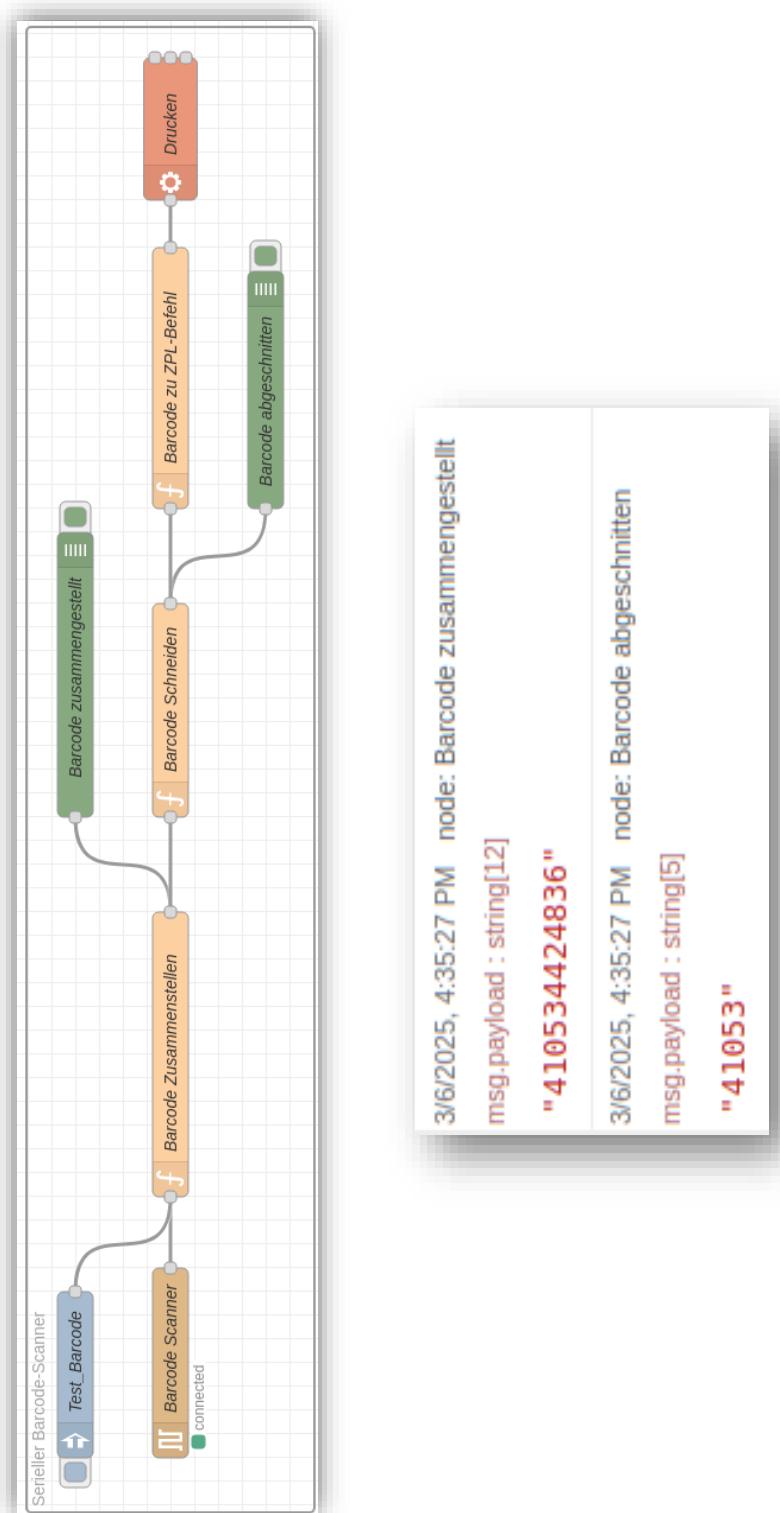


Abbildung 12: Node-RED-Flow zur Barcode-Verarbeitung mit Debug-Ausgabe

### A.15 Klassendiagramm-Drucker

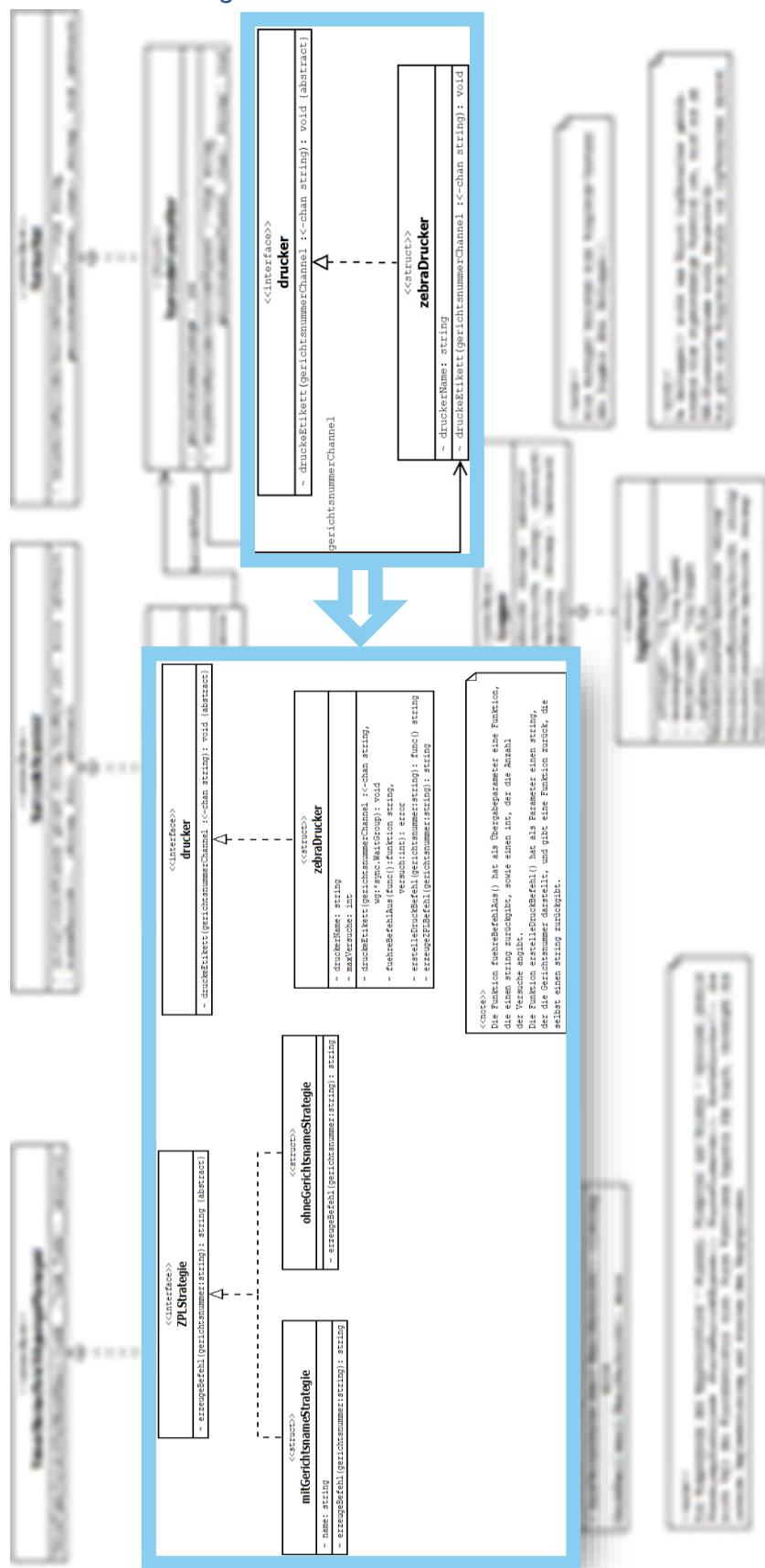


Abbildung 13: Klassendiagramm. Drucker

### A.16 Codebeispiel 1: Drucksteuerung

```

1. // ZPLStrategie definiert das Interface für das Erstellen von ZPL-Befehlen.
2. type ZPLStrategie interface {
3.     erzeugeBefehl(gerichtsnummer string) string
4. }
5. // mitGerichtsnameStrategie erstellt einen ZPL-Befehl für Gerichte mit Namen.
6. type mitGerichtsnameStrategie struct {
7.     name string
8. }
9. // ohneGerichtsnameStrategie erstellt einen ZPL-Befehl für Gerichte ohne Namen.
10. type ohneGerichtsnameStrategie struct{}
```

**1****Strategie**

```

11. // befehlAusfuehrer definiert ein Interface zum Ausführen von Befehlen
12. type befehlAusfuehrer interface {
13.     ausfuehren(befehl string) error // Führt einen Shell-Befehl aus
14. }
15. // bashAusfuehrer ist eine konkrete Implementierung für Bash-Befehle
16. type bashAusfuehrer struct{}
17. // ausfuehren führt den übergebenen Bash-Befehl aus
18. func (b bashAusfuehrer) ausfuehren(befehl string) error {
19.     cmd := exec.Command("bash", "-c", befehl) // Erstellt Bash-Kommando
20.     return cmd.Run() // Führt es aus
21. }
```

```

22. // StarteDrucker initialisiert den Druckprozess
23. func StarteDrucker(d drucker, gerichtsnummerChannel <-chan string, wg *sync.WaitGroup) {
24.     d.druckeEtikett(gerichtsnummerChannel, wg)
25. }
```

**4**  
**Fassade**

```

26. // erzeugeBefehl generiert einen ZPL-Befehl mit Gerichtsnamen für die Druckausgabe.
27. func (strategie *mitGerichtsnameStrategie) erzeugeBefehl(gerichtsnummer string) string {
28.     return fmt.Sprintf('echo -e
"XA^CI28^FO12,20^A0N,50,50^FB400,1,0,C,0^FD%s^FS^FO12,70^A0N,40,40^FB350,4,0,C,0^FD\"%s\"^FS^XZ" | lp -d
Zebra_Technologies_ZTC_ZD410-203dpi_ZPL -o raw', gerichtsnummer, strategie.name)
29. }
30. // erzeugeBefehl generiert einen ZPL-Befehl ohne Gerichtsnamen für die Druckausgabe.
31. func (strategie *ohneGerichtsnameStrategie) erzeugeBefehl(gerichtsnummer string) string {
32.     return fmt.Sprintf('echo -e "XA^FO74,66^A0N,100,100^FD%a^FS^XZ" |
lp -d Zebra_Technologies_ZTC_ZD410-203dpi_ZPL -o raw', gerichtsnummer)
33. }
```

**2****Strategie**

```

34. // druckeEtikett liest Gerichtsnummern aus dem Kanal und startet den Druckprozess
35. func (z *zebraDrucker) druckeEtikett(gerichtsnummerChannel <-chan string, wg *sync.WaitGroup) {
36.     defer wg.Done()
37.     for gerichtsnummer := range gerichtsnummerChannel {
38.         befehl := z.erstelleDruckBefehl(gerichtsnummer) // Funktion wird einer Variable zugewiesen
39.         if err := z.fuehreBefehlAus(befehl, 1); err != nil {
40.             os.Exit(1)
41.         }
42.     }
43. }
```

**5****Funktionale Programmierung**

```

44. // fuehreBefehlAus führt einen Druckbefehl rekursiv aus
45. func (z *zebraDrucker) fuehreBefehlAus(funktion func() string, versuch int) error {
46.     if versuch < 0 {
47.         return fmt.Errorf("Kritischer Fehler: Der Parameter 'versuch' darf nicht negativ sein (übergeben: %d)", versuch)
48.     }
49.     if versuch > z.maxVersuche {
50.         return fmt.Errorf("Kritischer Fehler: Der Druckbefehl konnte nach %d Versuchen nicht
ausgeführt werden.", z.maxVersuche)
51.     }
52.     err := z.befehlAusfuehrer.ausfuehren(funktion()) // Führt Befehl über Interface aus
53.     if err != nil {
54.         time.Sleep(1 * time.Second) // Wartezeit vor erneutem Versuch
55.         return z.fuehreBefehlAus(funktion, versuch+1) // Rekursiver Aufruf
56.     }
57.     return nil
58. }
```

**6****Rekursion**

```

59. // erstelleDruckBefehl erstellt den Befehl als Funktion, die den ZPL-Befehl zurückgibt
60. func (z *zebraDrucker) erstelleDruckBefehl(gerichtsnummer string) func() string {
61.     return func() string {
62.         return z.erzeugeZPLBefehl(gerichtsnummer)
63.     }
64. }
65. // erzeugeZPLBefehl formatiert die gerichtsnummer als ZPL-Befehl für den Drucker.
66. func (z *zebraDrucker) erzeugeZPLBefehl(gerichtsnummer string) string {
```

```

67.     model.LadeGerichteAusDatei() // Lade die aktuellste Version der Gerichte
68.     var strategie ZPLStrategie
69.     if name, vorhanden := model.VorhandeneGerichteMap[gerichtsnummer]; vorhanden {
70.         strategie = &mitGerichtsnameStrategie{name: name}
71.     } else {
72.         strategie = &ohneGerichtsnameStrategie{}
73.     }
74.     return strategie.erzeugeBefehl(gerichtsnummer)
75. }
```

3

Strategie

## A.17 Codebeispiel 2: Testfall zur Befehlserzeugung mit Strategie-Muster und Konsoleausgabe

```

1.   func TestErzeugeBefehl(t *testing.T) {
2.       // Arrange
3.       var strategie ZPLStrategie // Variable für die Strategie, die die ZPL-Befehle erstellt
4.       testfaelle := []struct {
5.           testName    string // Name des Testfalls
6.           gerichtsnummer string // Gerichtsnummer, die in den Befehl eingefügt wird
7.           gerichtsname  string // Name des Gerichts (falls vorhanden)
8.           erzeugterBefehl string // Erwarteter ZPL-Befehl, der aus der Strategie erzeugt werden soll
9.       }()
10.      // Testfall: Strategie mit Gerichtsnamen
11.      {"Strategie-Pattern mit Gerichtsnamen", "46902", "Apfelringe", fmt.Sprintf(`echo -e
"^-XA^CI28^FO12,20^A0N,50,50^FB400,1,0,C,0^FD46902^FS^FO12,70^A0N,40,40^FB350,4,0,C,0^
FD\`Apfelringe\`^FS^XZ" | lp -d Zebra_Technologies_ZTC_ZD410-203dpi_ZPL -o raw`)},
12.      // Testfall: Strategie ohne Gerichtsnamen
13.      {"Strategie-Pattern ohne Gerichtsnamen", "12345", "", fmt.Sprintf(`echo -e
"^-XA^FO74,66^A0N,100,100^FD12345^FS^XZ" | lp -d Zebra_Technologies_ZTC_ZD410-
203dpi_ZPL -o raw`)},
14.      }
15.      // Iteriere über alle Testfälle
16.      for _, testfall := range testfaelle {
17.          t.Run(testfall.testName, func(t *testing.T) {
18.              // Act
19.              // Wenn ein Gerichtsname vorhanden ist, wird die Strategie für Gerichte mit Namen
20.              // verwendet.
21.              if testfall.gerichtsname != "" {
22.                  strategie = &mitGerichtsnameStrategie{
23.                      name: testfall.gerichtsname,
24.                  }
25.              } else {
26.                  // Wenn kein Gerichtsname vorhanden ist, wird die Strategie für Gerichte ohne Namen
27.                  // verwendet.
28.                  strategie = &ohneGerichtsnameStrategie{ }
29.              }
30.              // Rufe die erzeugeBefehl-Methode auf, um den ZPL-Befehl zu generieren
31.              ergebnis := strategie.erzeugeBefehl(testfall.gerichtsnummer)
32.              // Assert
33.              if testfall.erzeugterBefehl != ergebnis {
34.                  // Wenn die Befehle nicht übereinstimmen, gib eine Fehlermeldung aus
35.                  t.Errorf("[%s]: strategie.erzeugeBefehl() gab %s zurück; erwartet: %s", testfall.testName,
36.                  ergebnis, testfall.erzeugterBefehl)
37.              }
38.          })
39.      }
40.  }
```

```

==== RUN TestErzeugeBefehl
==== RUN TestErzeugeBefehl/Strategie-Pattern_mit_Gerichtsnamen
--- PASS: TestErzeugeBefehl/Strategie-Pattern_mit_Gerichtsnamen (0.00s)
==== RUN TestErzeugeBefehl/Strategie-Pattern_ohne_Gerichtsnamen
--- PASS: TestErzeugeBefehl/Strategie-Pattern_ohne_Gerichtsnamen (0.00s)
--- PASS: TestErzeugeBefehl (0.00s)
PASS
ok      github.com

```

Abbildung 14: Testergebnis der Befehlserzeugung mit Strategie-Muster

### A.18 Codebeispiel 3: Singleton-Logger

```

1.   // init wird automatisch aufgerufen, bevor main startet
2.   func init() {
3.     initialisiereProtokollierer(dateiPfad) // Hier wird die Logdatei initialisiert
4.   }
5.   // initialisiereProtokollierer erstellt und konfiguriert eine Singleton-Instanz des Log-Verwalters
6.   func initialisiereProtokollierer(logDateiname string) {
7.     once.Do(func() { // once.Do() verhindert, dass die Initialisierung mehrfach durchgeführt wird
8.       logDatei, err := os.OpenFile(logDateiname, os.O_APPEND|os.O_CREATE|os.O_WRONLY,
9.         0644)
10.      if err != nil {
11.        fmt.Printf("Fehler beim Öffnen der Logdatei: %v\n", err)
12.        return // Abbruch der Initialisierung
13.      }
14.      // Schreiben der Startzeile in die Logdatei
15.      _, err = logDatei.WriteString("\nStart der Protokollierung. Zeitstempel: " +
16.        time.Now().Format("2006-01-02 15:04:05") + "\n")
17.      if err != nil {
18.        fmt.Printf("Fehler beim Schreiben in die Logdatei: %v\n", err)
19.        return
20.      }
21.      // Initialisieren des logVerwalters
22.      globLogVerwalter = &logVerwalter{
23.        infoLogger: log.New(logDatei, "INFO: ", log.Ldate|log.Ltime|log.Lshortfile),
24.        warnungLogger: log.New(logDatei, "WARNUNG: ", log.Ldate|log.Ltime|log.Lshortfile),
25.        fehlerLogger: log.New(logDatei, "FEHLER: ", log.Ldate|log.Ltime|log.Lshortfile),
26.        logDatei: logDatei,
27.      }
28.    }
29.    // GetLogger gibt die Singleton-Instanz des Log-Verwalters zurück.
30.    func GetLogger() *logVerwalter {
31.      if globLogVerwalter == nil {
32.        initialisiereProtokollierer(dateiPfad)
33.      }
34.      return globLogVerwalter
35.    }
36.    // ProtokolliereInfo schreibt eine Info-Nachricht in die Logdatei
37.    func (l *logVerwalter) ProtokolliereInfo(nachricht string) {
38.      mu.Lock()
39.      defer mu.Unlock()
40.      l.infoLogger.Println(nachricht)
41.    }

```

### A.19 Codebeispiel 4: Löschfunktion

```

1.  // GerichtAusJSONLoeschen entfernt ein Gericht aus der JSON-Datenbank
2.  func GerichtAusJSONLoeschen(gerichtsnummer string) bool {
3.      if len(VorhandeneGerichteMap) == 0 {
4.          LadeGerichteAusDatei()
5.      }
6.
7.      mu.Lock()           // Sperrt den Mutex, um Race Conditions zu vermeiden
8.      delete(VorhandeneGerichteMap, gerichtsnummer) // Entfernt das Gericht aus der Map
9.      mu.Unlock()         // Entsperrt den Mutex
10.     // Speichert die aktualisierte Map in der JSON-Datei
11.     if speichereGerichteInJsonDatei() {
12.         BeleereMapVorhandeneGerichte() // Leert die Map nach erfolgreichem Speichern
13.         return true               // Löschkvorgang erfolgreich
14.     } else {
15.         return false             // Fehler beim Speichern
16.     }
17. }
```

### A.20 Codebeispiel 5: Gericht hinzufügen mit Go und Gin

```

1.  // neuesGerichtHinzufuegenAPI fügt ein neues Gericht basierend auf Benutzerinput hinzu.
2.  func neuesGerichtHinzufuegenAPI(c *gin.Context) {
3.      // Liest Benutzerinput für die Gerichtsnummer und den Namen
4.      gerichtsnummer := c.PostForm("gerichtsnummerInput")
5.      gerichtsname := c.PostForm("gerichtsnameInput")
6.
7.      if len(gerichtsnummer) != 5 { // Validierung: Gerichtsnummer muss genau 5 Zeichen lang sein
8.          c.HTML(http.StatusBadRequest, "hinzufuegen.html", gin.H{
9.              "Titel":      "Hinzufügenseite",
10.             "seitenTitel": "Gericht hinzufügen",
11.             "seitenBeschreibung": "Fügen Sie hier ein neues Gericht zur Liste hinzu.",
12.             "error":       "Die Nummer muss genau fünf Ziffern enthalten.",
13.         })
14.         return
15.     }
16.
17.     if gerichtsname == "" { // Validierung: Gerichtsname darf nicht leer sein
18.         c.HTML(http.StatusBadRequest, "hinzufuegen.html", gin.H{
19.             "Titel":      "Hinzufügenseite",
20.             "seitenTitel": "Gericht hinzufügen",
21.             "seitenBeschreibung": "Fügen Sie hier ein neues Gericht zur Liste hinzu.",
22.             "error":       "Der Gerichtsname darf nicht leer sein.",
23.         })
24.         return
25.     }
26.
27.     if model.PruefenGerichtInJSON(gerichtsnummer) { // Überprüfung, ob die Gerichtsnummer
28.         bereits existiert
29.         c.HTML(http.StatusBadRequest, "hinzufuegen.html", gin.H{
30.             "Titel":      "Hinzufügenseite",
31.             "seitenTitel": "Gericht hinzufügen",
32.             "seitenBeschreibung": "Fügen Sie hier ein neues Gericht zur Liste hinzu.",
33.             "error":       "Die Gerichtsnummer existiert bereits.",
34.         })
35.         return
36.     }
37.
```

```

35.     }
36.
37.     if model.GerichtInJSONHinzufuegen(gerichtsnummer, gerichtsname) { // Gericht zur Map
38.         c.HTML(http.StatusOK, "hinzufuegen.html", gin.H{
39.             "Titel":      "Hinzufügenseite",
40.             "seitenTitel": "Gericht hinzufügen",
41.             "seitenBeschreibung": "Fügen Sie hier ein neues Gericht zur Liste hinzu.",
42.             "message":    "Das Gericht erfolgreich zur JSON-Datei hinzugefügt.",
43.         })
44.         return
45.     } else {
46.         c.HTML(http.StatusBadRequest, "hinzufuegen.html", gin.H{
47.             "Titel":      "Hinzufügenseite",
48.             "seitenTitel": "Gericht hinzufügen",
49.             "seitenBeschreibung": "Fügen Sie hier ein neues Gericht zur Liste hinzu.",
50.             "error":       "Fehler beim Hinzufügen des Gerichts.",
51.         })
52.         return
53.     }
54. }
55.
56. // rendereHinzufuegenSeite rendert die HTML-Seite zum Hinzufügen eines neuen Gerichts.
57. func rendereHinzufuegenSeite(c *gin.Context) {
58.     c.HTML(http.StatusOK, "hinzufuegen.html", gin.H{
59.         "Titel":      "Hinzufügenseite",
60.         "seitenTitel": "Gericht hinzufügen",
61.         "seitenBeschreibung": "Fügen Sie hier ein neues Gericht zur Liste hinzu.",
62.     })
63. }

```

## A.21 Codebeispiel 6: HTML-Template für Fehler- und Erfolgsmeldungen

```

1. {{define "meldungen"}} <!-- Definiert ein Template namens "meldungen" -->
2. <div class="mt-4"> <!-- Fügt oben einen festen Abstand hinzu -->
3.
4. {{if .message}} <!-- Prüft, ob eine Erfolgsnachricht (.message) vorhanden ist -->
5.     <div class="alert alert-success alert-dismissible fade show" role="alert">
6.         <strong>Erfolg:</strong> {{.message}} <!-- Zeigt den Text der Erfolgsnachricht an -->
7.         <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close">
8.             </button> <!-- Schließen-Button für die Nachricht -->
9.         </div>
10.    {{end}}
11.
12. {{if .error}} <!-- Prüft, ob eine Fehlermeldung (.error) vorhanden ist -->
13.     <div class="alert alert-danger alert-dismissible fade show" role="alert">
14.         <strong>Fehler:</strong> {{.error}} <!-- Zeigt den Text der Fehlermeldung an -->
15.         <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close">
16.             </button> <!-- Schließen-Button für die Fehlermeldung -->
17.
18.     </div>
19. {{end}} <!-- Beendet das Template "meldungen" -->

```

## A.22 Ablauf der Menülöschung in der Webanwendung

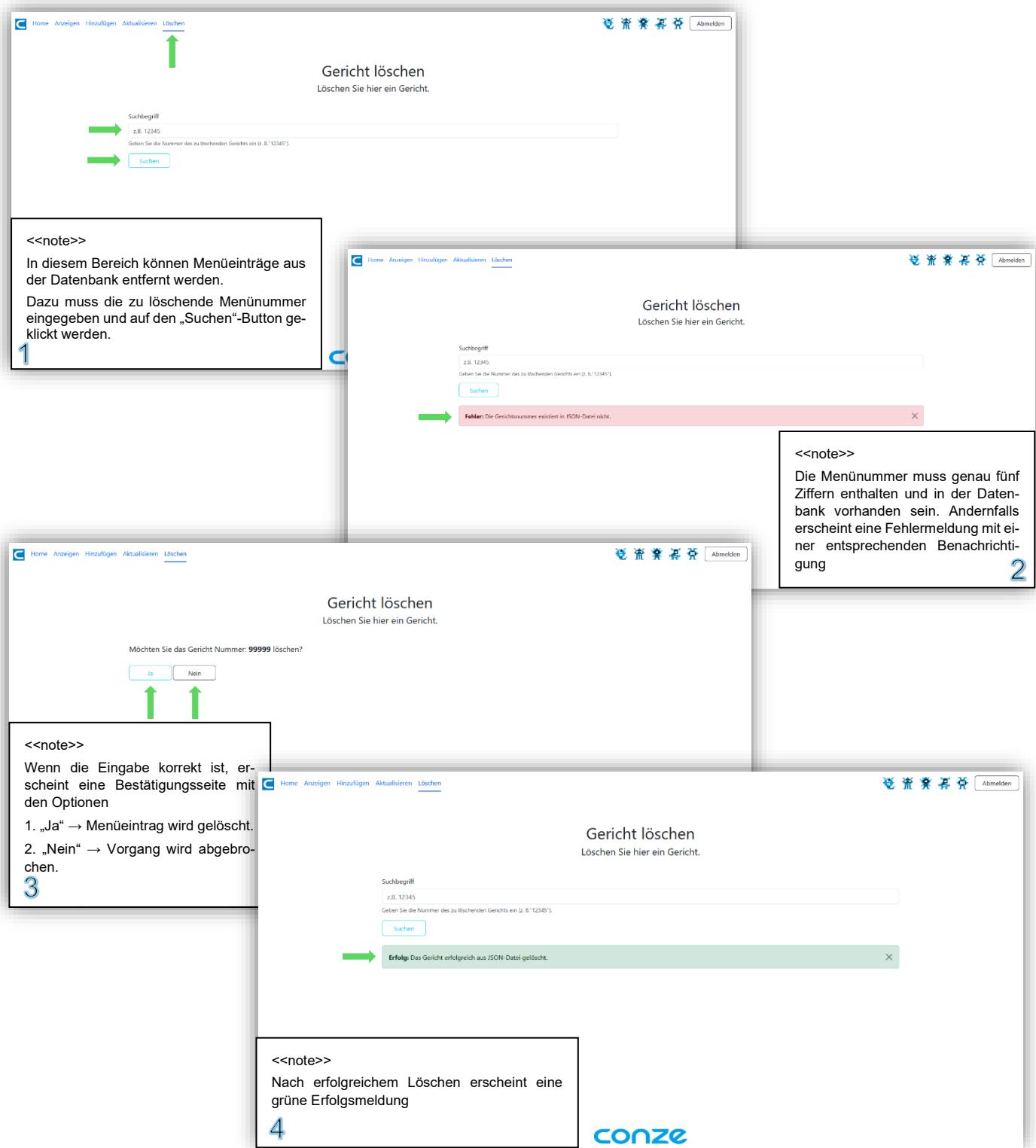


Abbildung 15: Ablauf der Menülöschung in der Webanwendung

A.23 Prozessablauf des Akzeptanztests (Bildcollage)



Abbildung 16: Prozessablauf des Akzeptanztests

## A.24 Qualitätssicherungskonzept – Übersicht der Testmethoden

Maßnahme	Beschreibung	Durchgeführt	Feedback
<b>Proof of Concept</b>	Validierung der geplanten Hardware durch einen frühen Prototyp, um die grundsätzliche Machbarkeit sicherzustellen		Nicht erforderlich
<b>Unit-Tests mit Mutationstests</b>	Durchführung von Unit-Tests für Kernfunktionen sowie Mutationstests zur Bewertung der Qualität der Unit-Tests		Nicht erforderlich
<b>Lasttest</b>	Simulation gleichzeitiger Datenbankzugriffe und Prüfung des Druckerverhaltens unter Last		Nicht erforderlich
<b>Code-Review</b>	Überprüfung der Performance, Wartbarkeit und Identifikation möglicher Code-Smells		
<b>Black-Box-Test</b>	Test der Web-Anwendung aus Anwendersicht ohne Kenntnis des Quellcodes zur Überprüfung der Funktionsweise		
<b>Akzeptanztest</b>	Test unter realen Bedingungen im Essensempfangsprozess, um die zuverlässige Zusammenarbeit aller Komponenten sicherzustellen		

Tabelle 4: Qualitätssicherungskonzept